ANL/ALCF/ESP-13/17

ALCF-2 Early Science Program Technical Reports

Compendium of Individual Technical Reports for the 16 ESP Projects

Argonne Leadership Computing Facility

ANL/ALCF/ESP-13/17

ALCF-2 Early Science Program Technical Reports

Compendium of Individual Technical Reports for the 16 ESP Projects

Edited by
Timothy J. Williams
Argonne Leadership Computing Facility, Argonne National Laboratory

July 18, 2013

ALCF-2 Early Science Program Technical Reports

Editor: Timothy J. Williams, Argonne Leadership Computing Facility

July 2013

Contents

Title Page Photos: *Top:* Mira—one of three rows of 16 IBM Blue Gene/Q racks. *Bottom:* Attendees at the ALCF Early Science Program Kick-Off Workshop, 18-19 October 2010, in front of Argonne National Laboratory's TCS Conference Center.

1 The Early Science Program

The Early Science Program (ESP) is managed for the Department of Energy by the Argonne Leadership Computing Facility (ALCF). ESP was funded to prepare key scientific applications for the architecture and scale of *Mira*, a 48K-node IBM Blue Gene/Q system that went into production at ALCF in April 2013, and to solidify libraries and infrastructure that will pave the way for other production applications. To distinguish it from past and future Early Science Programs, we denote this one as the ALCF-2 ESP (ALCF-2 is the project name associated with procurement of *Mira*).

The 16 Early Science projects are the result of a call for proposals in 2010, and were chosen based on computational and scientific reviews. The projects, in addition to promising delivery of exciting new science, are all based on state-of-the-art, petascale, parallel applications. Starting in October 2010, the project teams, in collaboration with ALCF staff and IBM, have undertaken intensive efforts to adapt their software to take advantage of *Mira's* Blue Gene/Q architecture, which, in a number of ways, is a precursor to future HPC architectures. Together, the 16 projects span a diverse range of scientific fields, numerical methods, programming models, and computational approaches. The latter include particle-mesh methods, adaptive meshes, spectral methods, Monte Carlo, molecular dynamics, and ab initio computational chemistry methods. These applications also represent a large portion of the ALCFs current and projected production computational workload.

The official Early Science period lasted only about three months (between machine acceptance and commencement of production), during which the ESP had dedicated use of *Mira*. Projects have about 2 billion core-hours to use on *Mira*, as much as possible during that 3 month period, so It was essential that the projects were "ready to run" when the clock started ticking. The long lead time of the Program, and dedicated postdoctoral appointees for most projects, working with ALCF staff, helped make that possible.

1.1 These Technical Reports

The purpose of the Technical Reports gathered here is to document and publicly circulate what the ESP projects have done and learned in preparing their application codes for *Mira*. This includes, for example, approaches for better use of in-node thread parallelism, accessing the BG/Q's QPX functionality (4-way SIMD vectorization), use of BG/Q specific libraries such as the low-level PAMI communication system, and using ESSL and Mass libraries. We hope that these examples will inform and help our production users on *Mira*, and those working on optimization applications *en route* to applying for computer time grants on ALCF systems via the INCITE and ALCC programs.

2 Acknowledgements

For all projects in these reports: This research used resources of the Argonne Leadership Computing Facility at Argonne National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under contract DE-AC02-06CH11357.

3 Climate-Weather Modeling Studies Using a Prototype Global Cloud-System Resolving Model

PI: V. Balaji *(Geophysical Fluid Dynamics Laboratory)*

Project Summary

We expect our understanding of the role of clouds in climate to undergo a qualitative change as the resolutions of global models begin to encompass clouds. At these resolutions, non-hydrostatic dynamics become significant and deep convective processes are resolved. We are poised at the threshold of being able to run global scale simulations that include direct, non-parameterized, simulations of deep convective clouds. The goal of this project is to use Mira to explore the frontier of weather prediction and climate modeling with the newly developed Geophysical Fluid Dynamics Laboratory (GFDL) global cloud-resolving model. A single, unified atmospheric modeling system with a cubed-sphere dynamical core and bulk cloud microphysics running at hydrostatic ($\sim$10 km) and non-hydrostatic ($\leq$5 km) resolutions will be run with the goal of capturing the climatology of clouds and severe storms in a warming world. The ability to reproduce historical tropical storm statistics will be used as a test of this ground-breaking model. The purpose of the experiments proposed is to validate the global cloud-resolving climate model via hurricane hindcasts. For this purpose, we will perform hurricane verification studies for the 2008 Atlantic Season. Storms in 2008 lasted a total of 100 days; performing 5-day forecasts on each of the days would give a total of 500 forecast days for the season.

Report originally released as ANL/ALCF-ESP-13/1

Climate-Weather Modeling Studies Using a Prototype Global Cloud-System Resolving Model

Research Team Members:

V. Balaji – Computational Scientist, Princeton University
Shian-Jiann Lin – Physical Scientist, GFDL
Christopher Kerr – Computational Scientist, GFDL

Executive Summary:

Clouds remain the largest source of uncertainty in our understanding of global climate change. Different aspects of the planetary cloud field can provide positive and negative feedback to the Earth's energy balance, and clouds of course are directly implicated in changes to the planetary distribution of precipitation. A fundamental problem in our current understanding of the role of clouds in the dynamics of climate are that current resolutions do not resolve the fundamental length scales associated with clouds. We expect our understanding of the role of clouds in climate to undergo a qualitative change as the resolutions of global models begin to encompass clouds. At these resolutions (which roughly scale with the tropopause height of 10km) non-hydrostatic dynamics become significant and deep convective processes are resolved. We are poised at the threshold of being able to run global scale simulations that include direct, non-parameterized, simulations of deep convective clouds. The goal of this research is to use the Argonne Leadership Computing Facility to explore the frontier of weather prediction and climate modeling with the newly developed Geophysical Fluid Dynamics (GFDL) global cloud-resolving model. A single unified atmospheric modeling system with a cubed-sphere dynamical core and bulk cloud micro-physics running at 3.5km resolutions was run with the goal of capturing the climatology of clouds and severe storms in a warming world. The ability to reproduce historical tropical storm statistics will be used as a test of this ground-breaking model.

Project Summary:

We have completed several 3-months of experiments with the 3.5km resolution global hydrostatic atmospheric model. The figures below show the preliminary diagnostics from the experiment.

A significant amount of development has been undertaken to improve the computational and I/O performance of HiRAM. Work has also continued on improving the post-processing diagnostic packages for the high-resolution experiments.

Computational Infrastructure:

The GFDL weather and climate models are built on the Flexible Modeling System (FMS). Details of FMS can be found at: http://www.gfdl.noaa.gov/fms. FMS is written primarily in FORTRAN 90 with approximately 0.75M lines of executable code. The model also utilizes a high-level hybrid MPI/OpenMP model in all of the component models (atmosphere, land …). The parallel I/O layer allows for both single and multi-threaded I/O, as well as quilted I/O from a subset of MPI ranks. Output data uses the netCDF4 library, including its parallel I/O, chucking and deflation options.

Performance Studies:

Considerable progress has been made to improvements to the computational FMS infrastructure. These improvements have included:

- Enabling the performance scaling of the FMS infrastructure. The results of this work are shown for HiRAM and Held-Suarez in Figures 1 and 2
- Implementing a memory footprint that scales with increasing core counts
- Incorporating a high-level hybrid programming model in all component models
- Providing an I/O scheme that scales with increasing core counts

Understanding the performance characteristics of HiRAM is an important component of the study. Through development of diagnostics tools, in collaboration with IBM Watson Research, the results of the studies with the diagnostic tools show that HiRAM at 3.5km resolution has the following computational foot-print:

- HPM characteristics: 40% FPX-60% FXU. 10B/cycle (not bandwidth limited)
- Model timer counters: Atmosphere=75%, Physics=15%, land and coupler=10%. (70% compute and 30% communication)

We have also produced line and function-level timing data which we are using to direct optimizations. The have identified additional regions for OpenMP threading and elimination of data copies at statement-level and across subroutine boundaries.

Improving the post-processing performance of the FMS infrastructure is a critical component of the project. The model currently generates 250GB of history data per model day. These history files are currently downloaded and post-processed at GFDL. There are approximately 0.10M lines of post-processing scripts and eventually these will need to be moved to a system closer to BG/Q.

The development activity are being expanded based on the success of this work and this will enable the entire FMS software including the component ocean and ice models to execute efficiently on BG/Q. We have also begun studies to:

- Extend the scalability of HiRAM beyond the 250,000 hardware threads current used in production
- Improve single core performance of the code. The areas for study include the implementation of: prefetch, transactional memory, and vectorization in the code
- Exploit additional forms of higher-level parallelism in the codes including improvements to our OpenMP implementation
- Examine the trade-offs between process and thread based parallelism.
- Possible implementation of partitioned global address space (PGAS) in the codes
- Implement a parallel I/O scheme for the non-distributed arrays.

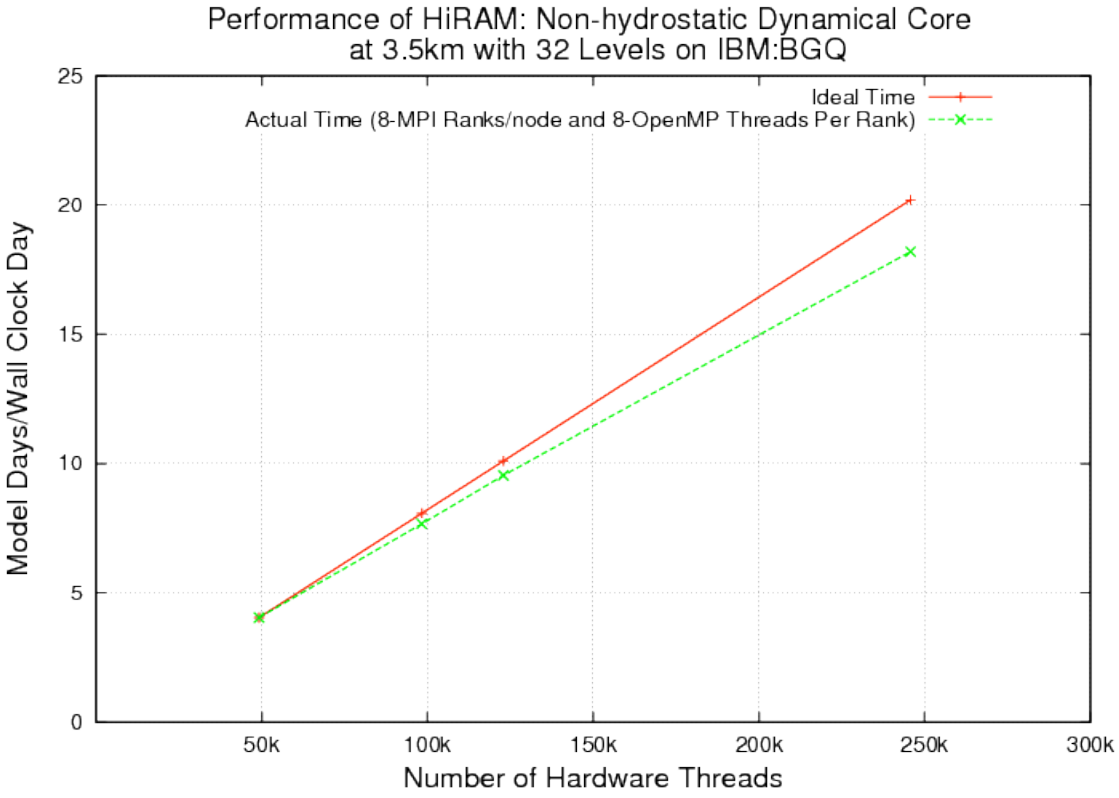


Figure 1: Shows the scaling of the global non-hydrostatic cloud-resolving model: HiRAM at 3.5km resolution. The model is configured to execute on 8-MPI Ranks/node and 8-OpenMP threads/rank. In production, HiRAM executes on either 15,360 or 30,720 MPI ranks with 8-MPI Ranks/node and 8-OpenMP threads/rank.

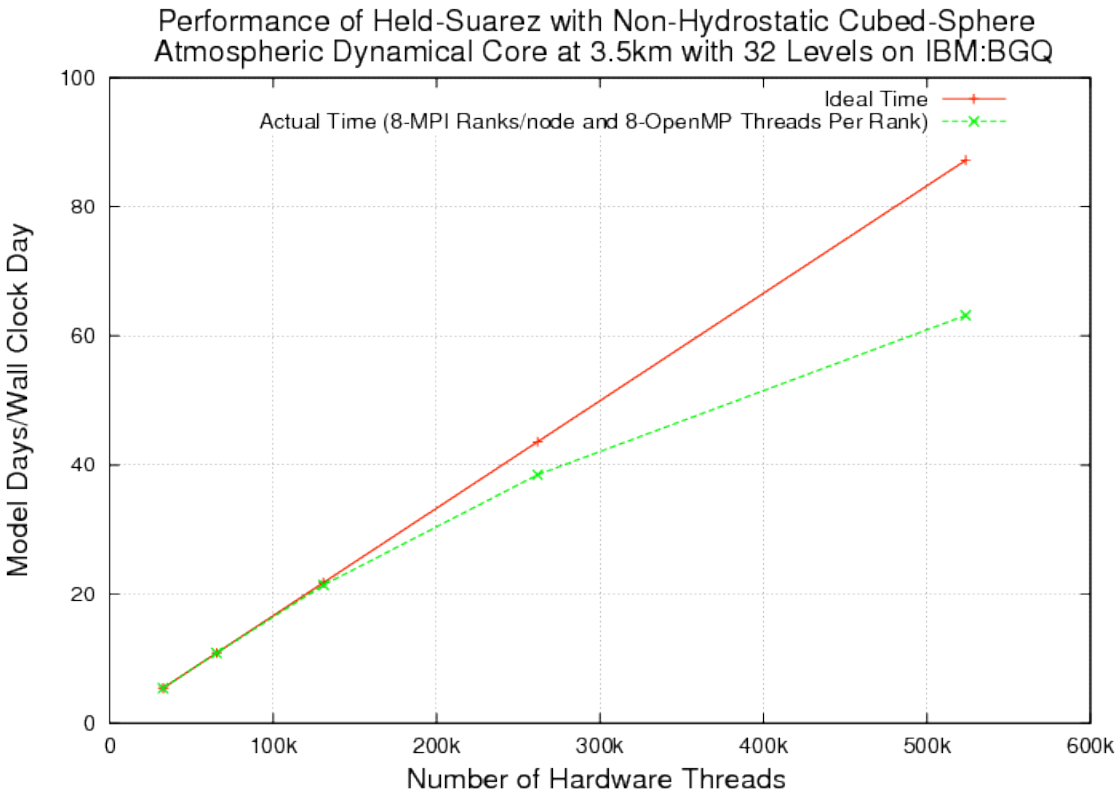


Figure 2: Shows the scaling of the Held-Suarez Test-Case with the Cubed-Sphere Dynamical Core at 3.5km resolution. The model is configured to execute on 8-MPI Ranks/node and 8-OpenMP threads/rank.

4 Materials Design and Discovery: Catalysis and Energy Storage

PI: Larry Curtiss *(Argonne National Laboratory)*

Project Summary

New materials may help solve global energy challenges

Our energy future hinges on the design and discovery of new materialslike materials to replace the oils currently used to make plastics, and materials to power electric vehicles.

Scientists at Argonnes Center for Nanoscale Materials and the Material Science Division are pairing the power of the Blue Gene/Q with newly available electronic structure codes to conduct massively parallel quantum mechanical calculations for use in the design of breakthrough materials that may have energy-related applications.

Materials reduce greenhouse gases, power electric vehicles

A team of researchers, led by Larry Curtiss with the Material Science Division and the Center for Nanoscale Materials at Argonne, is focusing research efforts on catalytic materials and on materials used for electric energy storage. Catalytic materials are used for bond-specific activation for efficient chemical transformations. This research could yield new strategies for more energy-efficient, environmentally friendly chemical synthesis to help reduce greenhouse gases, or in new methods for replacing petrochemicals with inexpensive, abundant small alkanes.

Creating new materials for electrical energy storage, (specifically, for the interface between electrolyte and electrode) could lead to safer, longer-range batteries for electric vehicles.

Finding better solutions, faster with the Blue Gene/Q

Using the extreme compute power of the Blue Gene/Q, researchers will employ high-accuracy quantum mechanical calculations using density functional theory (DFT) and quantum monte carlo (QMC). In preparation for the new architecture of the Blue Gene/Q, key computational kernels will be re-written to employ OpenMP nested parallelism.

Report originally released as ANL/ALCF-ESP-13/2

Mira Early Science Program Final Technical Report Materials Design and Discovery: Catalysis and Energy Storage

Anouar Benali and Nichols A. Romero
Leadership Computing Facility, Argonne National Laboratory

April 3, 2013

Introduction

The investigation and design of new classes of materials for energy and catalysis requires a multi-facetted approach to simulation. Multiple methods are needed to study materials on the length scale 0.1 nm - 10 nm. For simulations where the atomic (and electronic) degrees of freedom are relevant, the methods of choice in the surface science, condensed matter physics, and material science communities are classical molecular dynamics (CMD), Density Functional Theory (DFT), and quantum Monte Carlo (QMC).

The original scope for this Early Science Program (ESP) project was to perform fast-accurate DFT calculations on materials for energy and catalysis using the GPAW[1, 2, 3] code on Blue Gene/Q. The types of calculations included significantly reduced time-to-solution on systems sizes accessible on Blue Gene/P ($\sim$10, 0000 valence electrons), but also systems which were were at least a factor of two larger ($\sim$20, 000 valence electrons). GPAW is a real-space DFT code using the projector augemented wave (PAW) method. DFT calculations on Blue Gene/P were executed on over $>$100, 000 cores using GPAW; thus it was consider a success on Mira's predecessor system, Intrepid.

One of the co-PIs (NAR), determined that the work necessary to allow the GPAW code for these aformentioned types of calculations could not be accomplish within the time frame of the ESP. This was not simply due to human time required to implement OpenMP parallelism, but also from intrinsic algorithmic limitations in supporting libraries, most notably ScaLAPACK. Additionally, the return-on-investment on $\mathcal{O}(N^3)$ DFT code has become some what tenable at best. NAR argues that what is really needed in preparation for exascale computing and to enable high-fidelity materials research is robust reduced scaling, $\mathcal{O}(N)$ or $\mathcal{O}(N)\log(N)$, DFT approaches.

Thus, in aggreement with the other co-PIs, the decision was made to purse QMC as a complimentary method on Mira since it would be vaulable for the scientific community and could easily leverage the massive parallelism that would be provided by the Blue Gene/Q. We note that the two other atomic-scale methods mentioned here, CMD and DFT, are being pursued by other ESP projects on Mira. The remainder of this report will focus on our progress on QMC.

Beyond Density Function Theory

DFT provides *qualitative* accuracy for many well-behaved systems but lacks *quantitative* accuracy for most materials. One example where DFT consistently performs poorly is van der Waals dominated systems; additionally, chemical accuracy, generally considered to be 1 kcal/mol (=4 kJ/mol or 1 meV) cannot be achieved. This accuracy can only be achieved by an accurate description of the electronic correlations of the system and therefore making it difficult to use mean field methods, such as DFT or Hartree Fock (HF).

Accurate many-body methods, such as Coupled Cluster (CC), provides accurate estimates of the energies by solving the many body Schroedinger equation, but becomes rapidly computationally intractable as the number of electrons increases, scaling as poorly as N^7. QMC, within the variational Monte Carlo (VMC) and diffusion Monte Carlo (DMC) methods are "stochastic approaches for evaluating quantum mechanical expectation values with many-body Hamiltonians and wave functions. [..] The main attraction of these methods is that the computational cost scales as some reasonable power (normally from N^2 to N^4) of the number of particles N. This scaling makes it possible to deal with hundreds or even thousands of particles, allowing applications to condensed matter."[4]

We therefore solve the Schröedinger equation with the manybody Hamiltonian. QMC formalism is the usual Monte Carlo (MC) formalism in the sense that it solves multi-dimensional integrals by sampling randomly the space and allowing the system to evolve in the imaginary time using MC steps. Only moves that lower the energy are accepted. The obtained total energy comes at a cost of a variance and an error bar due to its stochastic nature.

$$\sigma^2 = \left\langle E_T^2 \right\rangle - \left\langle E_T \right\rangle^2, \qquad \delta = \frac{\sigma}{\sqrt{M}} \tag{1}$$

It becomes evident that to reduce the error bar, one should run the simulation longer or simply increase the number of samples, which suits particularly well large supercomputers systems.

A good sampling requires starting close to the right answer. In order to do so, we use a trial wave function that is created from a Slater determinant of single-particle orbitals (obtained from a previous DFT calculation) in combination with Slater-Jastrow parameter that explicty incorporate electron correlation effects.

Splines Single-particle orbitals (SPO) are a set of functions in (R^3), one function per orbital that describes its quantum state. During simulation, a walker (R^{3N}) samples spatial regions for the many-body state at that point, requiring an evaluation of all orbital functions. The Slater determinant of single-particle orbitals depends on the choice of basis set (molecular orbitals, plane waves etc...). For the class of materials we are interested in, the most practical choice is the use of plane waves. The B-pline approximation in QMC reports significant reduction of time of calculation while maintaining plane wave level of precision. The mesh size in the X, Y, and Z dimensions determines the accuracy of the representation. When a point in space is evaluated, a minicube of coefficients (64 coefficients) surrounding that point is required to interpolate its value.

Twists Generating a QMC trial wavefunction can be accomplished by generating a DFT wavefunction that contains all of the k-points necessary to express the many-body wavefunction with the different boundary conditions in the QMC simulation cell. Then the resulting QMC calculations will be performed with all of these different boundary conditions. Practically speaking, one needs to specify the boundary conditions for the QMC calculations using a k-point grid of $n \times n \times n$ twists. This also has the effect to change the type of the wave function from real when no twists are used, to complex when they are present.

Workflow

A common workflow for QMC consists on generating a trial Slater-Jastrow wave function, running a VMC optimization and finally running a DMC to reach the desired accuracy. VMC treats the square of the wave function as the probability distribution on which to do Monte Carlo. This means the form of the wave function limits the minimum energy you can reach in VMC. DMC, uses a branching, birth/death process based on the imaginary-time version of the Schröedinger equation to guide the random walkers in their random walk. Its minimum energy is limited only by the nodal surface, derived from the trial wave function, which prevents random walkers from moving between different-signed areas of the configuration space.

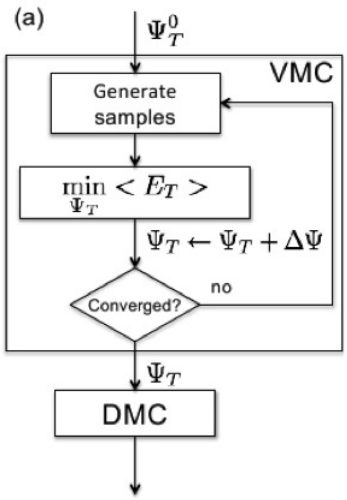


Figure 1: QMC workflow chart.

QMCPACK on MIRA

We use the QMCPACK[5, 6] simulation package for this project. The code was developed starting 2002 by the Ceperley Group at UIUC. Since then, the community of developers led by Dr. Jeongnim Kim (ORNL) has grown and spread amongst different institutions and National Laboratories.

QMCPACK is a heavily templated C++ code using all aspects of object-oriented coding, STL libraries and MPI and OpenMP for communication. This should have made it ready for porting on BGQ, however, as we experienced with our early access, many C++ standard were not compatible between different compilers and the use of OpenMP led to some serious problems (race conditions and thread unsafe regions) that had to be addressed before going further.

Once we were certain that the code was producing the right answers we proceeded to optimizing it.

Profiling We used HPM and GPROF as main profiling tools. Most of the systems of importance to us use twists and therefore we focused on the complex-valued wave functions which exercises the complex-valed code paths in QMCPACK.
Running GPROF showed that 71% of the application time was spent in

the spline evaluation of the wavefunction. This time was spread between two functions, `Eval-Z` evaluating the spline and `Eval-Z-VGH` evaluating the spline, the gradient and the hessian. (These functions exist also for the real-valued type in two versions, a double precision and a single precision version). Computationally, each function consists on 4 nested loops ($4\times4\times4\times N$) where $4\times4\times4$ corresponds to the number of coefficients in the minicube and N is the number of orbitals in the system, as described in the splines section.
In order to optimize the code we modified these two types of algorithms (for complex type and double/single precision type) using two general algorithms, then adding a layer of QPX and finally prefetching when it was possible.

Algo M. Algorithm M. consists of fusing the $4\times4\times4$ loops and unrolling the inner loop with a stride of 8. For Mira, we used QPX instructions to manually load and store data after using the fused multiply-add functions. As a last step we added prefetching on the spline-only evaluation function to improve memory management.

Algo B. Algorithm B. consists on reversing the order of the loops to $N\times4$ and unrolling the other loops. The mathematical expression of the problem is modified decreasing substantially the number of floating-point operations. For Mira we managed to use a similar algorithm to replace most of the instructions by QPX functions but were not able to benefit from prefetching.

As said previously, according to the type of system one can study (solid, nanocluster, molecules etc...) the use of twists will make the wavefunction either complex or real. This will exert 2 different parts of the code, a complex-valued type and a real-valued type (with a double precision and a single precision version). We applied the same methodology to optimize the spline evaluation functions and show the results in table-1. Results show that Algo. M is more efficient for the simple evaluation of the spline, while Algo B. is more efficient when spline, gradient and hessian are evaluated. The increased latency in Algo B. is hidden by the very important decrease of the total number of instructions (see table-2. However, Algo M doesn't have a better management of memory access and does not reduce as effectively the number of instructions, but for a smaller function, it is far more efficient than Algo. B.

When profiling the same problem with the new optimized algorithms we

Speed up	Eval-Z	Eval-D	Eval-S
Algo. B	0.38	0.81	0.39
Algo. M	2.48	0.91	1.02
Algo. (X) with QPX	3.94 (M)	1.08 (M)	1.26 (M)
QPX + Prefetch.	4.5	1.23	1.81
Speed up	Eval-Z-VGH	Eval-D-VGH	Eval-S-VGH
Algo. B	1.59	0.93	1.62
Algo. M	2.15	1.01	0.95
Algo. (X) with QPX	7.62 (B)	1.58 (B)	1.31 (B)

Table 1: QMCPACK speed up for three different types of wave functions exerting the complex-valued part (`Eval-Z`, `Eval-Z-VGH`),the real part (double precision) (`Eval-D` and `Eval-D-VGH`) and its single precision version (`Eval-S` and `Eval-S-VGH`).

see that the percentage of peak, the memory management and the number of instructions per cycle completed per core dropped significantly (see table-2). However, the number of instructions has decreased significantly which hides the latency in the memory management and most of all, the time spent on the spline evaluation and the time to solution was reduced by a factor 2.67 (see Fig-2). We selected the most efficient algorithms and applied them to QMCPACK. Results are shown in Fig-2.

Conclusion

Quantum Monte Carlo algorithms (due to their stochastic nature and the independence between samples) can benefits greatly from massively parallel supercomputers. The large number of cores on Mira can be leverage by QMC to study very large systems at chemical accuracy in extremely short time by using a very large number of samples, corresponding to a very large number of cores. The use of QPX and prefetching improved substantially the time to solution making the code even more efficient with a 2.67 speedup. Working with QMCPACK on BGQ Mira allows us to study a larger spectrum

Profiling	Original Version	Mira Optimized
Time spent on Spline evaluation	70.97 (%)	22.33 (%)
Percentage of Peak	6.55 %	5.33%
All XU Instructions (in Billion)	27,644	8,581
All AXU Instructions (in Billion)	22,786	4,896
FP Operations (in Billion)	43,043	13,017
Instructions/cycle completed/core	0.6138	0.4417
L1 d-cache hits	94.03 (%)	88.60 (%)
L1P buffer hits	5.36(%)	5.92 (%)
L2 cache hits	0.35 (%)	4.50 (%)
DDR hits	0.26 (%)	0.98 (%)

Table 2: Performance Comparison between Original version of QMCPACK and Mira modified version of QMCPACK

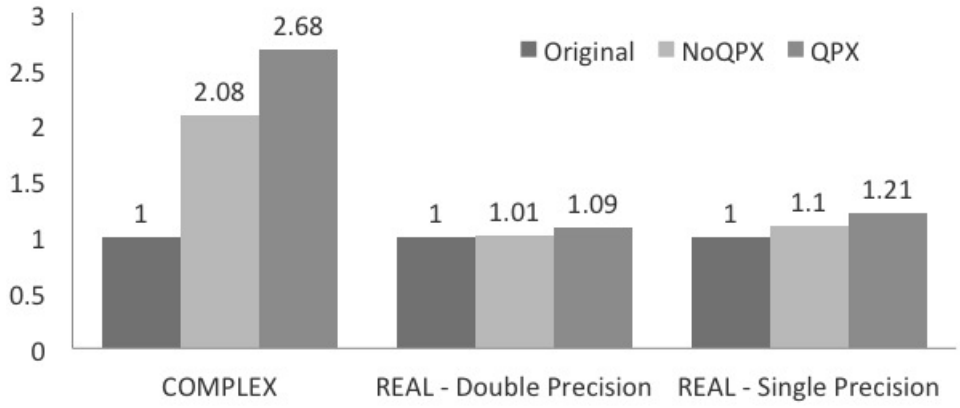


Figure 2: QMCPACK Speed up using compared to the original version using our cross platform algorithm (NoQPX) and QPX for all three types of wavefunctions.

of materials at the chemical accuracy which is a great achievement. Many applications, from material design to biochemistry are being investigated and should soon be submitted to high impact journals.

The work on QMCPACK, specially on Mira is far from being over. As one can notice on this report, most of the efforts were focused on porting

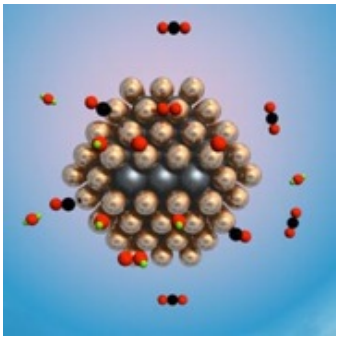


Figure 3: Pt solids and Nanoclusters for Catalysis

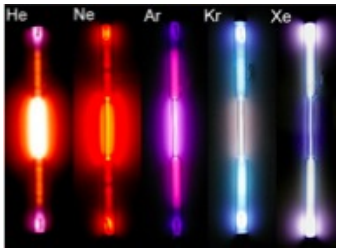


Figure 4: Ar, Kr and Xe Solid (Simulation of Van der Waals dominated solids

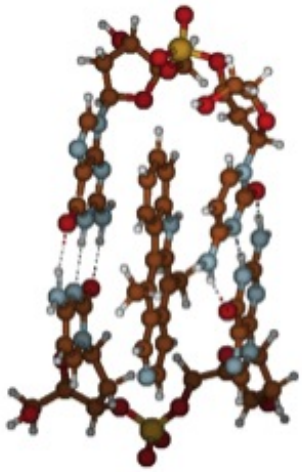


Figure 5: Molecule of Ellipticine with DNA fragments

the code to BGQ, optimizing the main kernel using QPX and "prefetching". However very little has been done in implementing nested open OpenMp (hybrid paralellization) which in theory, could reduce the time to solution by a factor 4. Hopefully this next step will be undertaken in the near future.

Bibliography

[1] J. J. Mortensen, L. B. Hansen, and K. W. Jacobsen, "Real-space grid implementation of the projector augmented wave method," *Phys. Rev. B*, vol. 71, p. 035109, Jan 2005.

[2] J. Enkovaara, C. Rostgaard, J. J. Mortensen, J. Chen, M. Dułak, L. Ferrighi, J. Gavnholt, C. Glinsvad, V. Haikola, H. A. Hansen, H. H. Kristoffersen, M. Kuisma, A. H. Larsen, L. Lehtovaara, M. Ljungberg, O. Lopez-Acevedo, P. G. Moses, J. Ojanen, T. Olsen, V. Petzold, N. A. Romero, J. Stausholm-Möller, M. Strange, G. A. Tritsaris, M. Vanin, M. Walter, B. Hammer, H. Häkkinen, G. K. H. Madsen, R. M. Nieminen, J. K. Nørskov, M. Puska, T. T. Rantala, J. Schiøtz, K. S. Thygesen, and K. W. Jacobsen, "Electronic structure calculations with GPAW: a real-space implementation of the projector augmented-wave method," *J. Phys.: Condens. Matter*, vol. 22, p. 253202, 2010.

[3] J. Enkovaara, N. A. Romero, S. Shende, and J. J. Mortensen, "GPAW - massively parallel electronic structure calculations with Python-based software," *Procedia Computer Science (2011)*, vol. 4, pp. 17–25, 2011.

[4] Needs, M. D. Towler, N. D. Drummond, and P. L. Ríos, "Continuum variational and diffusion quantum monte carlo calculations," *Journal of Physics: Condensed Matter*, vol. 22, no. 2, p. 023201, 2010.

[5] J. Kim, K. Esler, J. McMinis, and D. M. Ceperley, "QMCPACK simulation suite." unpublished.

[6] J. Kim, K. Esler, J. McMinis, and D. M. Ceperley, "Quantum monte carlo algorithms: making most of large-scale multi/many-core clusters," Conference Series, (Chattanooga, TN), Scientific Discovery through Advanced Computing (SciDac), J. of Physics, Jun 2010.

5 Direct Numerical Simulation of Autoignition in a Jet in a Cross-Flow

PI: Christos Frouzakis *(ETH Zürich)*

Project Summary

Lean combustion turbines provide excellent opportunities for environmentally friendly propulsion and electricity generation, but are severely limited by the danger of autoignition of the fuel-air mixture before its proper location. Further development of next-generation devices hinges upon better understanding of autoignition in flows that are characterized by considerable fluctuations of velocity, composition, and temperature. The aim of this project is to study the fundamental aspects of autoignition in a fuel-air mixing pattern directly applicable to mixing ducts in gas turbines. The Nek5000-based combustion code will be used to perform very large-scale direct numerical simulations of autoignition of a diluted hydrogen jet in a cross-flow of hot turbulent air in a laboratory-scale configuration. Detailed description of chemistry and molecular transport will be used to investigate the flow and scalar fields under cold and reactive conditions. It will also be used to construct databases that will be explored for years by engineers and scientists working in engine development for the construction and validation of advanced combustion models for engineering-type computational fluid dynamics codes.

Report originally released as ANL/ALCF-ESP-13/3

Direct Numerical Simulation of Autoignition in a Jet in a Cross-Flow Configuration

Ammar Abdilghanie[1,1], Christos E. Frouzakis[2,2] and Paul F. Fischer[3,3]

[1]Leadership Computing Facility, Argonne National Laboratory, Argonne, IL 60439
[2]Aerothermochemistry and Combustion Systems Laboratory, Swiss Federal Institute of Technology Zurich (ETHZ), Zurich, Switzerland
[3]Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439

April 4, 2013

[1]aabdilghanie@alcf.anl.gov
[2]frouzakis@lav.mavt.ethz.ch
[3]fischer@mcs.anl.gov

Abstract

Autoignition in turbulent flows is a challenging fundamental problem due to the intricate coupling of different physical and chemical processes extending over multiple flow and chemistry scales. At the same time, the improved understanding and ability to predict autoignition in flows characterized by considerable fluctuations of velocity, composition, and temperature is essential for the development of novel low-emission concepts for power generation. The aim of this project is to study the fundamental aspects of autoignition in a fuel-air mixing device directly applicable to mixing ducts in gas turbines. The NEK5000-based code for low Mach number reactive flows is used to perform very large scale direct numerical simulations of autoignition of a diluted hydrogen jet ejected in a cross-flowing stream of hot turbulent air in a laboratory-scale configuration. We report on our experience running NEK5000 on the new BGQ system at ALCF **mira** during the early science period (ESP). First of all, the most efficient problem size per MPI-rank is obtained through core-level efficiency metric measured from the target simulation. Furthermore, the most efficient number of ranks is found through strong scaling experiments. Finally, low-level insight into the observed parallel efficiency is enabled through IBM's HPC Toolkit libraries.

Contents

Chapter 1

Introduction

Understanding the conditions under which autoignition of reacting mixtures occurs is of primary importance for the design and operation of modern lean premixed (LP) and lean premixed prevaporized (LPP) combustion devices such as low-NO_x stationary gas turbines (Dobbeling *et al.*, 2007) and propulsion devices such as subsonic ramjets and supersonic scramjets (Micka and Driscoll, 2012). In particular, the enhanced turbulent mixing between the fuel and oxidizer streams of the jet in cross-flow (JICF) configuration makes it an essential component in the design of premixing sections of modern high efficiency, low NO_x combustors.

The main objective of this study is to investigate the sensitivity of the JICF dynamics to the cross-flow temperature. In particular we ask the following questions: When and where do localized flame kernels form? Could the observed ignition time be correlated to ignition-delay time in the corresponding homogeneous system? Do these kernels develop into stable flame later on? How does the mixture get prepared for ignition and what is the role of the vorticity and pressure in mixing the jet (fuel) with the cross-flow (oxidizer) fluid? What is the local combustion mode associated with the observed ignition scenarios?

1.1 Overview of the Numerical Method

A weak formulation of the compressible reactive Navier-Stokes for ideal gas mixtures at the low Mach number limit were solved, using a parallel spectral-element code based on NEK5000 (Fischer *et al.*, 2008). The spectral element method is a high-order weighted residual technique that couples the rapid convergence of global spectral methods with the geometric flexibility of finite element methods (Deville *et al.*, 2002).

A high-order operator splitting technique is used to split the thermochemistry (species and energy equations) from the hydrodynamic subsystem (continuity and momentum equations) (Tomboulides *et al.*, 1997). The latter is integrated in time using a 3rd-order semi-implicit method, with explicit treatment of the nonlinear terms and implicit treatment of the viscous and pressure terms, while the thermo-

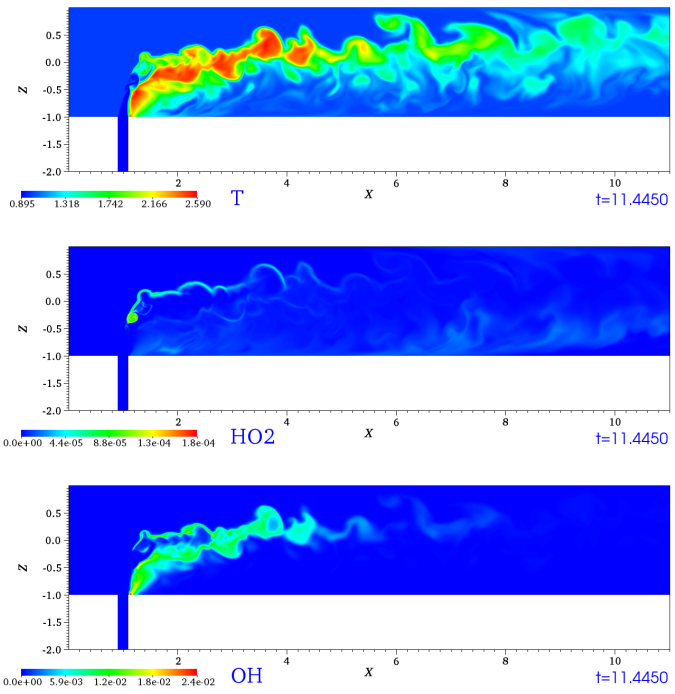


Figure 1.1: Snapshots of temperature and mass fractions of HO_2, OH for the T_{cf} = 950 K case on a cross section through the symmetry plane.

chemistry subsystem is solved fully implicitly by CVODE, a scalable BDF-based stiff ODE solver (Cohen and Hindmarsh, 1996). Detailed chemistry, thermodynamic properties and mixture-averaged transport properties are evaluated using Chemkin (Kee *et al.*, 1986).

Tensor product factorization is the core computational kernel in NEK5000. It is used to evaluate spectral element operators, interpolation, integration and differentiation. It is cast in terms of a sequence of dense and non-square matrix products (mxm). Optimizing the mxm kernel was performed on the BG/P using loop tiling and architecture-specific assembly code (Kerkemeier, 2010). The code developments resulted in improved core throughput and cache utilization (through data reuse) compared to highly optimized BLAS libraries. Finally, Kerkemeier (2010) developed a chemistry calculation kernel for the computation of the production rates, transport and thermodynamic property evaluation. All expensive computations were vectorized using the highly optimized math libraries MASS/MASSV which allowed best floating point, load and store performance.

1.1.1 Summary of Numerical Simulations

Exploratory numerical simulations on experimental computational grids as well as production runs were performed during the early science period on Mira. The computational grid is composed of $N_E = 1,591,752$ spectral elements within which the solution is interpolated using three-dimensional tensorial product of $p = 6$-th order Lagrange polynomials.

Post processing of the solution was performed using NEK5000 on the two datasets including computations of key species generation rates, mixture fraction, scalar dissipation rate, Takeno flame index and vorticity components. Analysis of the two large data sets was also mainly facilitated through visualizations with VisIt on the ALCF's GPU system (Eureka). Two crossflow stream temperatures (T_{cf} =930 and 950 K) were simulated at a Reynolds number, based on the friction velocity and the channel half-width, of $Re_\tau = 180$. Typical instantaneous snapshots of the fully ignited case are shown in fig. 1.1.

The simulations performed up till now employed close to 345 million grid points corresponding to 4.8 billion degrees of freedom (14 unknowns per grid point). We are currently developing the computational mesh for a planned high Reynolds number simulation ($Re_\tau = 590$). It is estimated that the total number of elements will be close to 3 millions and the polynomial order will be at least 8.

Chapter 2

Parallel Scaling and parallel efficiency metrics

2.1 Measurement of Parallel Efficiency

Parallel code scaling focuses on one of two forms: strong scaling or weak scaling. The goal of strong scaling is to reduce execution time for a fixed total problem size by adding processors (and hence reducing problem size per worker/MPI rank). On the other hand, ideal weak scaling behavior is to keep the execution time constant by adding processors in proportion to an increasingly larger problem size (and hence keeping problem size per worker fixed).

Parallel efficiency, η, for a problem run on N_1 processes/MPI ranks is defined relative to a reference run on N_2 ranks as

$$\eta_p = \frac{N_1\, t(N_1)}{N_2\, t(N_2)},$$

where $t(N)$ is the execution time on N ranks.

In order to ensure core utilization we measure the MPI-threading efficiency which we define as

$$\eta_t = \frac{N_1\, t(N_1)}{N_2\, t(N_2)},$$

where N_1 is the number of cores for the one MPI rank/thread per core configuration and $N_2 = 2N_1$ in the two threads/core and $N_2 = 4N_1$ in the four threads/core configurations. Note that a net speedup is obtained for threading efficiency of more than 50% in the two threads/core configuration and more than 25% in the four threads/core.

2.1.1 Core-level MPI Threading Efficiency

Instantiation of two and four MPI-ranks per core is enabled on Mira and hence full core compute power utilization is assured through the use of more than one rank per

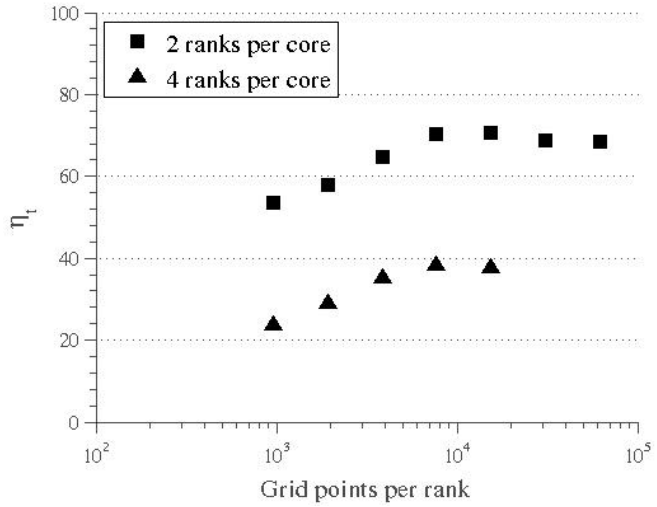


Figure 2.1: MPI threading efficiency for autoignition simulation using NEK5000. Note that the abscissa represents grid size per MPI rank.

core. However, this may lead to performance degradation for very large problem size because of the resulting on-chip resource contention. We have conducted a core-level parallel efficiency measurements on Mira for the target simulations using different number of cores and either two or four ranks per code.

Figure 2.1 compares the MPI threading efficiency for two and four ranks per core. It is clear that using two ranks per core is more efficient for same problem size (i.e. grid points) per rank and that the optimum grid points per rank is 7,000-10,000. This size ensures maximum utilization of the core's compute-power while minimizing on-chip resource contention among MPI threads.

2.1.2 Strong Scaling Experiment

A strong scaling is performed under the two MPI-ranks per core configuration for an autoignition simulation with a total of $\approx$ 345 million grid points.. Figure 2.2 shows that an ideal efficiency of a 100% is mainated up to approximately 130,000 MPI-ranks (65,000 cores) and that up to 60% efficiency is sustainable using nearly 500,0000 ranks.

2.2 Profiling and Hardware Performance Monitoring

Performance monitoring is enabled through a mechanism for obtaining information about the use of MPI routines (profiling) or wall clock time and hardware counters

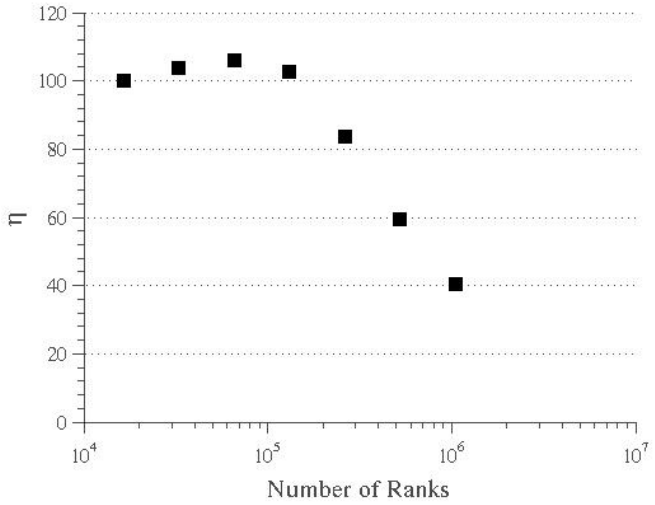


Figure 2.2: Parallel efficiency for autoignition simulation: strong scaling with ≈ 345 Million gridpoints and two MPI ranks per core.

for a code as a whole or a segment that is usually bracketed with calls to initiate and stop monitoring libraries.

2.2.1 IBM HPC Toolkit

IBMs' HPC Toolkit provides a mechanism for tracking the use of MPI routines during a programs execution. This is done through the use of a library which intercepts calls to MPI routines, records information about the call, and then continues with the MPI call. The primary types of information that are gathered are

- MPI Profile Data: A summary of MPI usage information typically consisting of the number of times each MPI routine was called, how much time was spent in each MPI routine, and the average size of a message for that routine.
- MPI Trace Data: A detailed time-history of every invocation of an MPI routine that the program made.
- Point-to-Point Communication Pattern: This information is derived from the collected trace information and indicates the number of bytes sent between ranks by point-to-point MPI routines.

IBM also provides a HPM Toolkit for hardware perfromance monitoring that has

- A utility *hpmcount*, which starts an application and provides at the end of execution the wall clock time, hardware performance counters inforamation, derived hardware metrics, and resource utilization statistics
- An instrumentation library *libhpm*, which provides instrumented programs with a summary output containing the above information for each instrumented region in a program (resource utilization statistics is provided only once, for the whole section of the program that is instrumented). This library supports serial and parallel (MPI, threaded, and mixed mode) applications, written in Fortran, C, and C++.

2.2.2 Nek5000 Instrumentation

In order to focus the analysis on the time stepper part of NEK5000, exclude restart-file reading and recurring IO, the time stepping loop was bracketed by calls to the respective libraries as follows:

```
call summary_start()
call hpm_start("nek_advance")
DO ISTEP=1,NSTEPS
  TIME STEPPING LOOP
ENDDO
call hpm_stop("nek_advance")
call summary_stop()
```

It should be noted that the code needs to be compiled with the debugging flag ”-g” enabled. Finally, the following link to IBM libraries needs to be appended to the list of libraries (USER_FLAGS) used by NEK5000:

```
USR_LFLAGS="${USR_LFLAGS} -L/soft/perftools/hpctw -lmpihpm
-L/bgsys/drivers/ppcfloor/bgpm/lib -lbgpm"
```

2.2.3 Sample MPI Profile

A small section of a sample profile dumped by rank 0 is shown below.

```
Data for MPI rank 0 of 65536
Times and statistics from summary_start() to summary_stop().
-----------------------------------------------------------------
MPI Routine  #calls avg. bytes  time(sec)
-----------------------------------------------------------------
MPI_Isend    508287 915.7 2.083
MPI_Irecv    508287 916.1 0.416
MPI_Waitall 162759 0.0 8.845
MPI_Bcast   1 4.0 0.000
MPI_Barrier 81 0.0 0.002
```

```
MPI_Allreduce  19296 2716.7 17.573
...etc.
-----------------------------------------------------------------
total communication time = 28.919 seconds.
total elapsed time = 195.156 seconds.
heap memory used = 103.090 MBytes.
```

Although the MPI-time is dominated by synchronization (`MPI_Allreduce` during global inner product evaluation for the linear system solve step for the velocity, pressure and thermo-chemistry), it is still representing only about 14% of the total execution time. The communication as well as total execution/elapsed time for all MPI-ranks is plotted in figure 2.3. It can be seen that the communication time as well the total elapsed time (and hence the computation time) is approximately uniform across all MPI-ranks, an indication of load balancing between the MPI ranks.

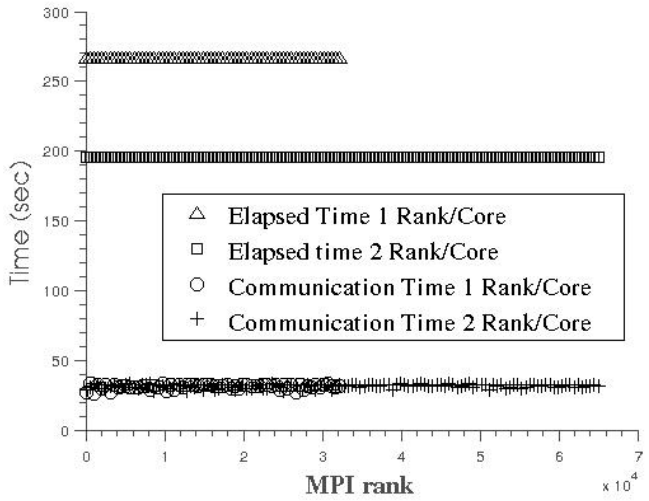


Figure 2.3: Communication and elapsed time for MPI ranks collected with IBM's HPC Toolkit

2.2.4 Hardware Performance

Hardware performance metrics are written to `hpm_summary.rank` files. The file lists information about the overall performance of the whole code as well as information about the bracketed segment of the code (the time stepper in this case).

Table 2.1 shows that with the use of two MPI-ranks per core the floating point operations per second per node has increased from 4.398 to 6.057 GFLOPS ensuring higher utilization of the core compute-power. Finally, it can be seen that there

is a trend of increasing access to both the L1P, L2 Cache and main memory with increasing number of ranks per core. This is most likely the reason beyond performance degradation for the four ranks per core configuration. Measurements from a four ranks per core were not possible due to the overhead of the measurement libraries themselves, even with light-weight versions of the same libraries provided by IBM.

Table 2.1: Summary of Hardware Performance Metrics Measured by hpm Library

	1 Rank/Core	2 Ranks/Core
FPU% , FXU %	20.6-79.4	17.7-82.3
Inst/cycle/core	0.3586	0.5794
GFLOPS/node	4.398	6.057
% of max issue rate/core	35.9	47.7
L1 d-cache hit	96.4	95.5
L1P buffer hit	1.46	2.09
L2 Cache hit	1.85	2.0
DDR hit	0.32	0.4
DDR total traffic (Bytes/cycle)	2.264	3.588

Chapter 3

Discussion of ESP Experience

The highly-scalable low Mach number reactive flow solver based on NEK5000 is used to simulate turbulent autoigniting flows in a laboratory scale jet-in-cross-flow configuration. The code relies heavily on two highly optimized mxm and thermo-chemistry PTTP kernels that were previously optimized on the BGP architecture (Kerkemeier, 2010).

Scaling results on Mira showed that ideal parallel efficiency of 100% can be sustained up to 130000 MPI ranks and up to 60% efficiency on nearly 500,0000 ranks and hence no further tuning is necessary for the current problem size.

We are currently evaluation different design strategies for the computational grid that will be used for the high Reynolds number simulations. Further optimization of grid-generation routines for the Algebraic Multigrid solver is necessary for tackling the high Reynolds number grids and is the subject of ongoing investigation.

Bibliography

Cohen, S. and Hindmarsh, A. (1996) CVODE, a stiff/nonstiff ODE solver in C. *Comp. Phys.* **10**(2), 138–143.

Deville, M., Fischer, P. and Mund, E. (2002) *High-order methods for incompressible fluid flow*. Cambridge University Press.

Dobbeling, K., Hellat, J. and Koch, H. (2007) 25 years of BBC/ABB/Alstom lean premix combustion technologies. *Trans.-ASME J. Eng. Gas Turb. Power* **129**(1), 2.

Fischer, P., J.W., L. and Kerkemeier, S. (2008) nek5000 Web page. http://nek5000.mcs.anl.gov.

Kee, R., Dixon-Lewis, G., Warnatz, J., Coltrin, M. and Miller, J. (1986) Technical report sand86-8246. *Sandia National Laboratories* .

Kerkemeier, S. (2010) Direct numerical simulation of combustion on petascale platforms. Ph.D. thesis, Swiss Federal Institute of Techonolgy Zurich (ETHZ), Nr. 19162.

Micka, D. and Driscoll, J. (2012) Stratified jet flames in a heated air cross-flow with autoignition. *Combust. Flame* **159**(3), 1205–1214.

Tomboulides, A., Lee, J. and Orszag, S. (1997) Numerical simulation of low mach number reactive flows. *J. Sci. Comp.* **12**(2), 139–167.

6 High Accuracy Predictions of the Bulk Properties of Water

PI: Mark Gordon *(Iowa State University)*

Project Summary

Among the ab initio methods, second-order perturbation theory (MP2) predicts highly accurate structures and relative energies for water clusters. Researchers will carry out molecular dynamics simulations of water at the MP2 level. However, full MP2 calculations of even modest-sized water clusters are far too time-consuming for dynamical simulations, even on the next-generation Blue Gene. Therefore, a key element of the current approach will be the use of MP2 in conjunction with the Fragment Molecular Orbital (FMO) method. Compared with todays calculations, researchers will determine the bulk properties at higher levels of accuracy using larger basis sets, larger embedded clusters, and longer dynamics simulations, as permitted by the greater computational capacity available with the Blue Gene. They will target the following bulk properties of water: structure, density, refractive index, diffusion constant, free energy, heat capacity, dielectric constant, vaporization enthalpy, isothermal compressibility, and thermal expansion coefficients. The final eight properties are more difficult to obtain than the first two. While Blue Gene/P gives good estimates, the greater capacity of the next-generation Blue Gene will be critical to establishing convergence of these properties with respect to theory, basis set, cluster size, and simulation length. There have been conflicting reports in the literature about the relevance of chain orring networks in water. The high-accuracy simulations on the next-generation Blue Gene will help settle this argument about the structure of liquid water.

Report originally released as ANL/ALCF-ESP-13/4

ESP Technical Report: February, 2013

High Accuracy Predictions of the Bulk Properties of Water

Staff at ALCF

Maricris Mayes (postdoctoral student)
Graham Fletcher (catalyst group member)
Yuri Alexeev (catalyst group member)

Executive Summary

Water is the most important liquid to humanity and a deep understanding of its behavior is of critical importance to national priority scientific issues such as global warming. Despite this, accurate theory and modeling of water is lacking, and the question of the detailed mechanism behind its bulk properties remains one of the outstanding unsolved problems in science. While many simulation studies of liquid water have been undertaken, resource and capability limitations have forced the vast majority to be performed using empirical model potentials with questionable reliability.

This project is the first to simulate water with high accuracy and high precision, employing recently developed advanced techniques that allow first principles *ab initio* quantum mechanical methods to harness hundreds of thousands of processors and scale up to thousands of water molecules.

Overview of numerical methods

A partial integro-differential equation based on Quantum Mechanics - called the Schroedinger Equation - is solved for the electronic structure of a molecular system in a basis of atom-centered functions. Dense linear algebra is involved. Integration over all space is performed analytically, and this consumes the majority of the execution time. Inter-atomic forces are derived from the electronic structure and used to drive dynamical simulations that yield bulk property predictions.

The project code is the General Atomic and Molecular Electronic Structure System (GAMESS), maintained by the Gordon Group at Iowa State University. We employ the Fragment Molecular Orbital (FMO) method in GAMESS together

with the Restricted Hartree-Fock (RHF) and Moeller-Plesset second order perturbation theory (MP2) levels of theory.

What is enabled by Mira over Intrepid

Following on from a 2010 INCITE award on *Intrepid*, this Early Science project on *Mira* focusses on the bulk properties of water at higher levels of accuracy using larger basis sets, larger embedded clusters, and longer dynamics simulations. While benchmark tests were performed on *Intrepid*, the greater computational capability available with *Mira* is critical to establishing *convergence* of the bulk properties with respect to theory, basis set, cluster size, and simulation length.

Code Modifications, new algorithms: RATTLE

This section describes new functionality - an implementation of the RATTLE method - that has been coded especially for the current Early Science Project by Yuri Alexeev of ALCF. This development stands to significantly reduce the cost of the project, to one half or one quarter of its original cost. The motivation for and impact of RATTLE are as follows.

Energy conservation is critical to any dynamical simulation carried out in a series of discrete time steps. With current technology, energy conservation imposes a maximum time step of around 0.3 femtoseconds. In addition, systems with large numbers of hydrogen atoms (such as water) can waste many steps where the trajectory is trapped within local minima corresponding to the vibrational modes of all the hydrogens.

One way to guide the trajectory through the most important regions of phase space, avoid local minima and, in so doing, permit larger steps, is to apply constrained dynamics. Today, almost all classical molecular dynamics simulations employ constraints, particularly where hydrogen atoms are involved. The most popular methods include SHAKE, RATTLE, and SETTLE. RATTLE is the most accurate because the constraints are applied to both distances and velocities simultaneously. RATTLE is a Lagrange-multiplier-based method developed especially for Velocity Verlet, a method for integrating Newton's equations implemented in GAMESS.

Figure 1 shows an example of 2 water molecules 8 Angstroms apart at 0.5fs intervals (time is plotted along the x-axis, and total energy along the y-axis), already the curve with RATTLE (green) is smoother than without (red).

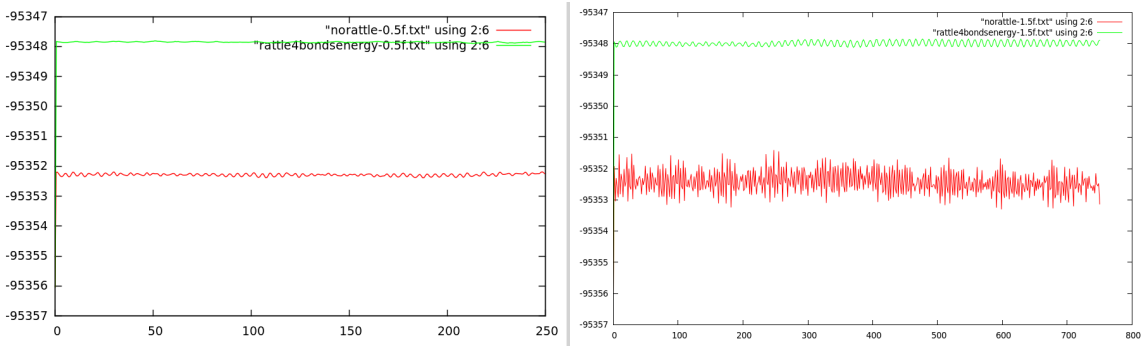


Figure 2, at 1.5fs intervals, shows that without RATTLE (red) the variation in energy can be large and erratic, while with RATTLE (green) the energy is much more stable. In this example, RATTLE allows 2-4x longer time steps to be taken, and we expect similar improvements to be obtained with larger systems. With longer time steps, proportionally fewer steps need to be taken, reducing the overall cost of the simulations.

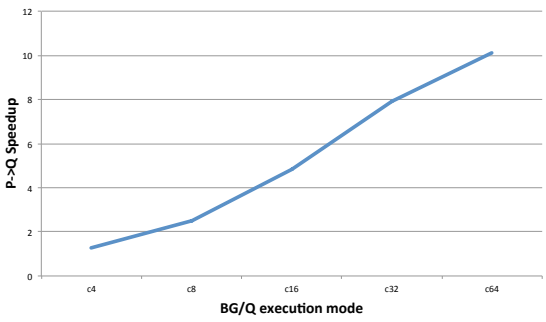
As it smooths out the trajectory, RATTLE also improves overall efficiency by removing any abrupt shifts in molecule positions that may hinder solution of the Schroedinger Equation for each water. Thus, RATTLE allows fewer, larger, more efficient and reliable time steps to be taken.

Performance on Mira

In this section, the performance on Mira is examined in three ways: first, a comparison with BG/P, second, for increasing relevant problem sizes, and lastly with BG/Q execution mode.

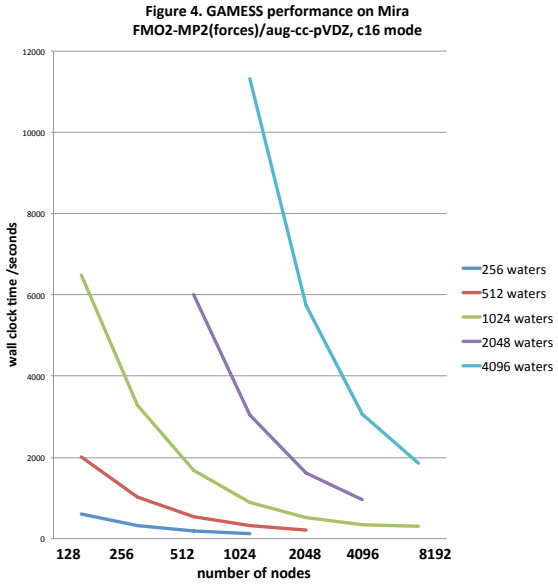
Figure 3 shows the speedup from BG/P to BG/Q of a calculation of the energy and forces at the MP2 level for the benzoquinone molecule with a 6-31G* atomic basis set. The relative improvement on a BG/Q node can be seen to rise monotonically with the execution mode compared to the 'VN' mode on BG/P, reaching a maximum speedup of x10.

Figure 3. GAMESS speedup from BG/P(VN mode) to BG/Q benzoquinone/MP2(forces)/6-31G*



The reasons for such a large improvement are several. BG/Q benefits not only from 4x more processor cores per node, and a 2x higher clock rate, but also from the ability in its microkernel to multi-rank the cores (not available on BG/P) in c32 mode, giving two MPI ranks per

core. This is of great benefit to GAMESS, allowing its 'data-server' processes to reside on the same core as the compute processes and, overall, utilize the cores more efficiently. An additional higher execution mode, c64, with four MPI ranks per core, gives the maximum speedup but the resultant amount of memory available to processes is too small (being somewhat less than 250 megabytes each) to be useful in the rest of the current project.



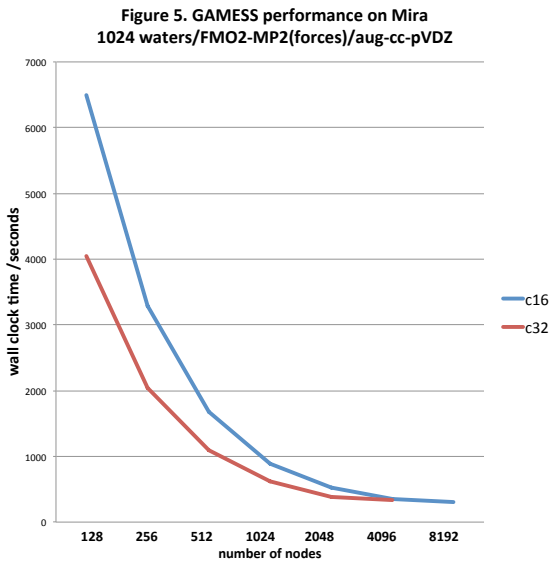


Figure 4 shows the scalability of FMO-MP2 calculations on 128 to 4096 waters with a 6-31G* atomic basis set, in c16 mode. As with c64 mode, some of the larger calculations cannot be accessed from c32 mode due to memory limitations, and c16 allows the greatest range of problem sizes that is useful in the current project to be examined for scalability. It can be seen how scalability improves with problem size, being poorest for the smallest problem (128 waters) and best for the largest problem (4096 waters). In quantum chemistry, where the computational cost is approximately cubic with the problem size while the communication overhead is quadratic, scalability usually improves with problem size.

Figure 5 compares the scalability of the 1024 water calculation from Figure 4 for two execution modes, c16 and c32. It can be seen that while c32 mode is always faster, especially at the lowest node count, the two curves converge as the number of nodes increases. While c32 mode is expected to be more efficient for reasons described above, the convergence with c16 mode toward higher node counts is most likely the result of increased I/O contention as the overall number of ranks increases. We are pursuing the latter hypothesis in our investigation of this phenomenon.

7 Cosmic Structure Probes of the Dark Universe

PI: Salman Habib *(Argonne National Laboratory)*

Project Summary

Dark matter and dark energy are the dominant components of the Universe. Their ultimate nature, however, remains mysterious, especially so of the dark energy. Ambitious ground and space-based missions investigating different aspects of the Dark Universe constitute a national and international investment measured in billions of dollars. The discovery potential of almost all of these missions relies crucially on theoretical modeling of the large-scale structure of the Universe. As observational error estimates for various cosmological statistics edge towards the one percent level, it is imperative that simulation capability be developed to a point that the entire enterprise is no longer theory-limited.

This project is a simulation framework powerful enough to discover signatures of new physics from next- generation cosmological observations. Relevant questions include: (1) Beyond the cosmological constant, what are the detectable signatures of a dynamical equation of state for dark energy? (2) How does modification of general relativity alter the nonlinear regime of structure formation? As for dark matter and related questions: (1) What are the effects of plausible dark matter candidates on the mass distribution? (2) What are the constraints on the neutrino sector from cosmological observations? In addition, the results of the simulations will be very useful for a range of astrophysical investigations, primarily in the areas of galaxy formation and the formation and evolution of galaxy groups and clusters. This is possible because the next generation, 10-petaflops IBM Blue Gene system will provide, at last, the computational power to resolve galaxy-scale mass concentrations in a simulated volume as large as state-of-the-art sky surveys. Researchers will generate numerically a mock galaxy catalog that will allow the determination of the effects of new physics on major cosmological observables.

Report originally released as ANL/ALCF-ESP-13/5

Porting and Tuning HACC on Mira

Hal Finkel

April 4, 2013

Abstract

The HACC (Hybrid/Hardware Accelerated Cosmology Code) framework has achieved over 69% of the floating-point peak running on all of Mira, and is now uniquely positioned to make significant contributions to the science of large-scale structure formation. At the beginning of the Early Science Project (ESP), HACC was attaining under 1% of the peak FLOPS, and lacked the memory-management and file I/O capabilities necessary for running at scale. This report describes the improvements that transformed HACC from an average scientific code to one well-known for its unmatched performance.

1 Introduction

Modern cosmology, and the study of large-scale structure formation in particular, is now a precision science, with both simulation and observation contributing to our overall understanding of a vast quantity of observational data. As the quantity and quality of observational data increases, corresponding increases in our simulation ability are required. In order to tackle questions related to the detailed behavior of dark matter and dark energy, the two dominate, yet mysterious, components of the energy-density of the universe [2, 1], we must be able to simulate the effect of small changes in theory parameters on observable statistical quantities.

After the emission of what is now the Cosmic Microwave Background (CMB) during recombination, the non-linear and long-range nature of the gravitational force becomes increasingly important to the overall evolution of the matter in the universe. At scales larger than that of galaxy clusters, no other forces make a significant contribution[1]. As a result, we can accurately predict how the matter will move and cluster at large scales using a "gravity-only" simulation. Because of the huge dynamic range required almost everywhere in the simulation volume, almost all structure-formation codes use a particle method whereby the matter in the universe is decomposed into many point masses which are evolved using a modified form of Newton's equations. The HACC (Hybrid/Hardware

[1] Assuming that the effect of dark energy on the background evolution of the universe is taken into account.

Accelerated Cosmology Code) framework provides an infrastructure for running just these kinds of simulations and performing some of the analysis necessary to compare the simulated universes to observational data.

Prior to the beginning of the Early Science Project (ESP), large simulations had been run with HACC only on the Roadrunner machine at LANL. The largest of these simulations used a few hundreds of billions of particles, and made use of the PowerXCell 8i GPU-like processors in order to quickly compute the short-range particle-particle force contributions [5]. HACC's ability to run on an massively-parallel CPU-only system like Mira was very limited, and initial tested showed that it achieved under 1% of the peak FLOPS (floating-point operations per second). In addition, its required one-file-per-rank I/O facility would have been impractical for use at scale, and its poor memory management would have precluded long runs. Over the course of the last two years, all of these things have greatly improved. In fact, HACC was a 2012 finalist for the IEEE/ACM Gordon Bell prize in scientific high-performance computing [4], reporting 13.94 PFlops at 69.2% of peak and 90% parallel efficiency on 1,572,864 cores running on the BG/Q machine Sequoia at LLNL. To meet today's science requirements, we'll need to run trillions of particles in several large simulations: HACC is now ready for that challenge!

2 Long-Range Force

The total gravitational force on every particle from every other particle is split into two parts: the long-range part and the short-range part. The long-range force is computed on a grid using a FFT (Fast Fourier Transform)-based technique. Unfortunately, memory limits the size of the grid to a resolution near the initial inter-particle spacing. As the universe evolves, and matter clusters together under gravity, the relevant length scale for particle-particle interactions in clustered regions falls well below the grid resolution. Adding this additional short-range component is the responsibility of the architecture-specific short-range force solver. Nevertheless, the long-range force solver, and specifically the routines used to compute the distributed FFTs on which the solver relies, determine the overall weak scaling of the code. As shown in Figure 1, the overall weak scaling of the code is nearly perfect, across architectures, and specifically on nearly all of Mira.

In order to achieve this scaling on tens of thousands of ranks, the algorithm using to compute the FFTs was changed. Prior to the ESP, HACC used a slab-decomposed FFT, meaning that the grid is decomposed in only one dimension. Unfortunately, this limits the total number of ranks that can participate in the FFT computation (because there can be only as many ranks as there are grid points in the smallest dimension). Due mostly to work by Nicholas Frontiere, this limitation was removed by implementing a distributed slab-decomposed FFT. The slab decomposed FFT enables as many ranks to participate as there are points on the smallest two-dimensional face. This difference is illustrated in Figure 2. Once completed, this slab-decomposed FFT allowed HACC to easily

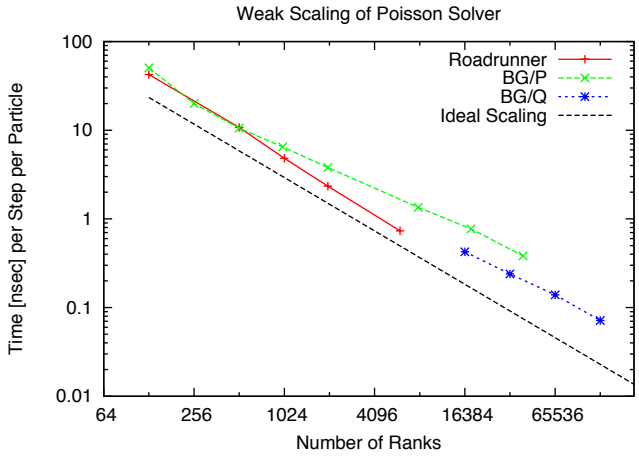


Figure 1: Scaling of the long-range force solver.

scale to the largest-available machines.

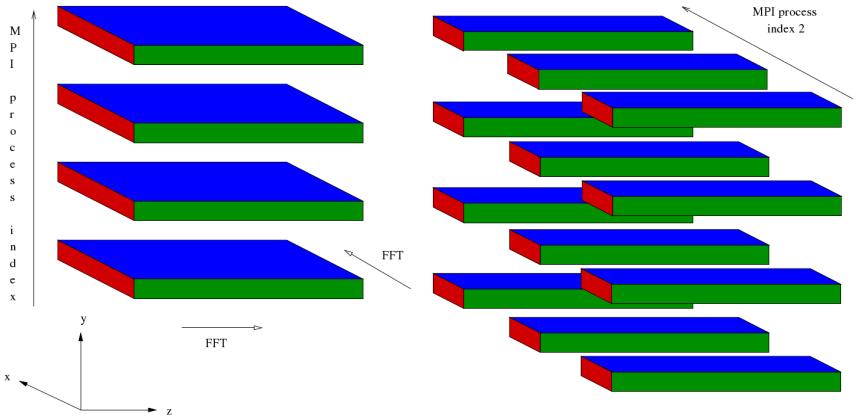


Figure 2: A diagram showing slab (left) and pencil (right) data decompositions [derived from diagrams in [6]].

It is worth mentioning that a significant reason that HACC's inter-rank communication needs are dominated by the FFT, and not by "ghost-region" particle exchange, is because the exchange of particles between ranks happens relatively infrequently. Not only is the short-range force computation subcycled (computed multiple times per long-range force computation), but the ghost-region exchanges do not even need to happen prior to every long-range computation. Compared to the size of the universe, the velocities of matter in the universe are relatively small, and the size of densely-clustered regions is limited. As a

result, we can make the ghost regions large enough to capture whatever large structures might reside on neighboring ranks close to the inter-rank boundary, and we can safely assume that they won't move very far in between ghost-region exchanges. To mitigate edge effects, we also don't compute the short-range force on particles near the outer boundary of the ghost region.

3 Short-Range Force and the RCB Tree

The short-range force is determined by taking the complete gravitational force between any two particles, and subtracting that part of the force that will be captured by the long-range force solver. This subtraction is determined by performing numerical experiments, and then the resulting functional form is created to fit the results of these experiments. Prior to the ESP, HACC used a lookup table to compute the force from its fitting form. Now, HACC uses a 5^{th}-order polynomial fit to the effective short-range part of the force. Specifically, this is:

$$f_{SR}(s) = (s + \epsilon)^{-3/2} - f_{grid}(s) \quad (1)$$

where $s = \mathbf{r} \cdot \mathbf{r}$, $f_{grid}(s) = \text{poly}[5](s)$, and ϵ is a short-distance cutoff.

While on Roadrunner (and on machines with GPUs) HACC can use a statically-partitioned N^2 algorithm to compute the local particle forces, such a naive algorithm would be too slow for Mira's CPUs. Instead, we must use a short-range force computation scheme which effectively approximates the force from farther-away particles on any given particle and only directly computes the contributions from the closest particles. The standard way of accomplishing this feat in an efficient way is to use a "tree" code. This means that some spatial-partitioning tree is used along with a monopole approximation for forces from particles in far-away tree nodes. Before the ESP, HACC has an octree-based tree code, but that implementation was inefficient, both computationally and in terms of memory overhead. Addressing these problems required a new design, and I implemented what has been called in the physics literature a Recursive Coordinate Bisection (RCB) tree[2]. Following the suggestion in [3], a RCB tree recursively splits each spatial node on its longest side such that the dividing plane intersects the center of mass of the particles in the node. As also suggested in [3], when each node is partitioned, the particle data for that node is also partitioned, and the particles are shuffled into the two subpartitions. Finally, this splitting process stops when the nodes reach a certain maximum number of particles. These implementation details, illustrated in Figure 3, have important performance benefits: The center-of-mass splitting makes the computation in each leaf node balanced by insuring that each leaf node has near the maximum number of allowed particles. The partitioning insures that each node's particle data exhibits a high degree of cache locality. Finally, keeping multiple particles in each leaf node gives us a tuning parameter allowing us to

[2]A computer scientist would, however, call it a KD-tree.

trade a fast but asymptotically expensive operation (the particle-particle interactions, which scale as N^2) for a slow but asymptotically favorable operation (the tree walk, which has $N \log N$ scaling). An additional benefit to keeping multiple particles in each leaf node is an increase in the accuracy of the computation. As can be seen from Figure 4, even making aggressive use of the monopole approximation still yields highly-accurate force computations in clustered regions when many particles are contained in each leaf node. On Mira, we normally run with a few hundred particles in each leaf node, and the resulting error from the monopole approximation is quite small.

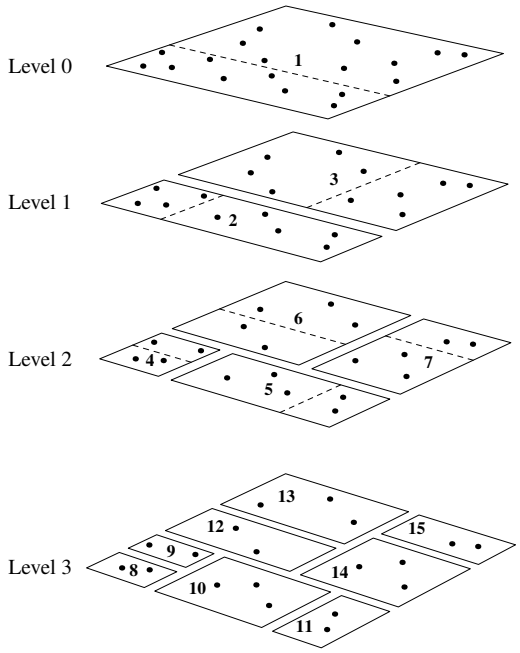


Figure 3: A diagram (from [3]) showing how the particles are recursively partitioned into RCB-tree nodes.

Another critical aspect of the attaining good performance on Mira is how multiple threads per rank are used to compute the short-range force on each particle. At this point, two different schemes have been used in production. The first scheme, which was used in the runs used for the Gordon Bell submission, confines the threading to the force computation within each RCB-tree leaf node. Within each leaf node, the particles are divided among the available threads, and because all particles in a leaf node share the same interaction list, each thread simply computes the force on its particles from those particles in the interaction list. The downsides of this scheme are that the per-leaf-node concurrency is limited by the number of particles per leaf node (which limits scaling), and that the process of walking the tree to compose the interaction list is not threaded. As a result, this scheme could scale to only 8 threads per rank. To overcome this

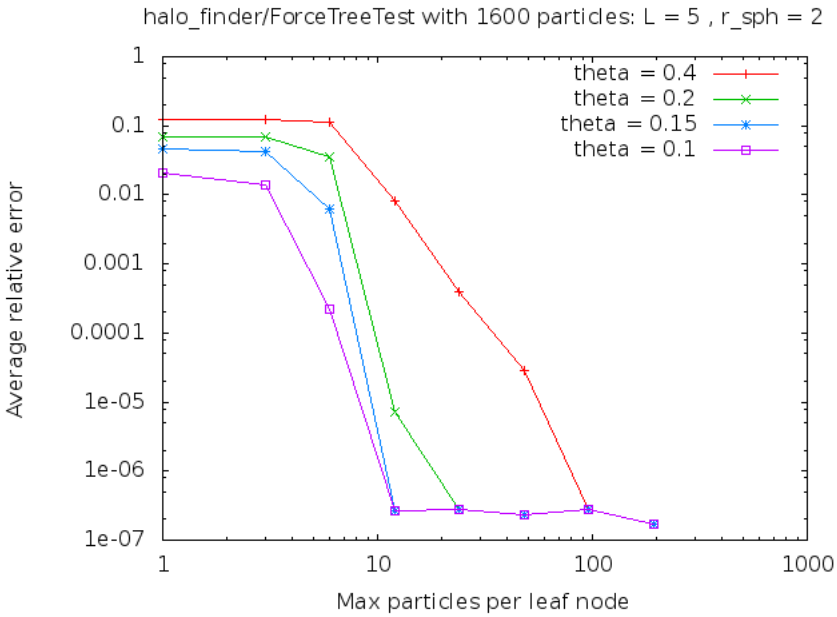


Figure 4: A plot of a force-computation test involving a sphere randomly filled with particles showing the relative error from the monopole approximation. The theta parameter controls the aggressiveness of the monopole approximation.

limitation, I implemented a second scheme. In this second scheme, a queue is composed of all leaf nodes, and these leaf nodes are divided among all available threads for processing (both the tree walk to form the interaction list and the force computation itself). As there are many leaf nodes per rank, as shown in Figure 5, this scheme can scale to 32 thread per rank on Mira. OpenMP was used to implement both schemes.

4 The Short-Range Force Kernel

The BG/Q-specific short-range force kernel, implemented by Vitali Morozov, uses QPX intrinsic functions, and a loop which is manually partially unrolled, in order to quickly evaluate the force on some particle by other particles in the interaction list. While the kernel appears to be a straightforward vectorization of the loop over particles in the interaction list (see Figure 6 for an exerpt), the form of the force computation (for example, the $-3/2$ exponent and the order of the polynomial in Equation 1) were chosen to map well onto the available instructions and the number of available registers.

Several other aspects of the force kernel implementation are worth noting: First, as shown in Figure 7, all data loaded by the kernel in explicitly prefetched

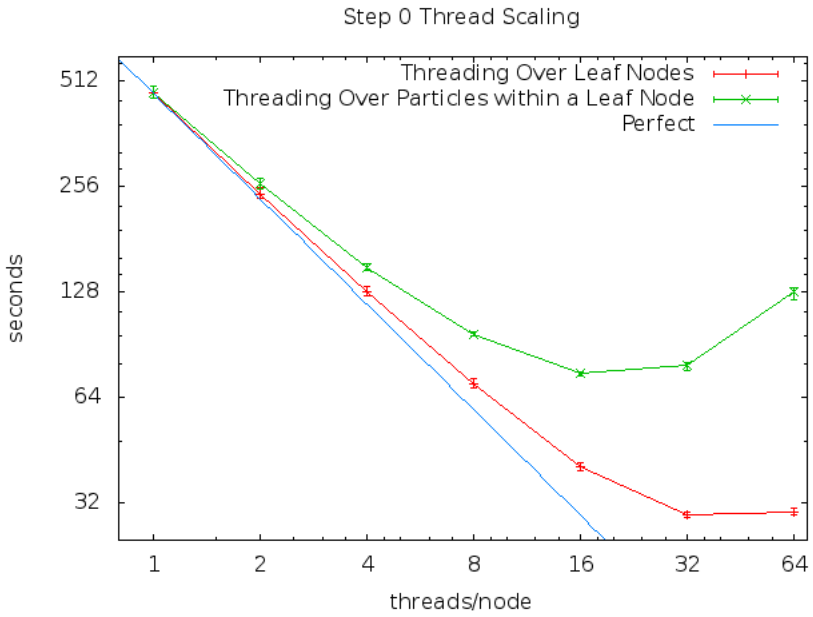


Figure 5: A plot of the time taken by the short-range force computation vs. the number of threads. The old threading scheme could not scale past 8 threads per rank, while the new scheme can scale to 32 threads per rank (running one rank per node).

into the L1 cache. On the BG/Q, even when the L1 prefetcher is prefetching a stream of interest, the data is not put into the L1, but rather stays in a separate L1P buffer, until accessed. Because the L1P access latency is much higher than that associated with accessing data from the L1 cache, explicitly prefetching the data into the L1 cache is critical. Second, we don't need fully IEEE-compliant values for the square root in Equation 1, and it can be computed using the provided reciprocal square root estimate and one Newton-Raphson iteration for refinement. Because the provided estimate is accurate to within one part in 2^{14}, only one iteration is required to provide the single-precision result. Finally, as branches are expensive, we use the provided "select" intrinsic in order to avoid branching as part of the distance filtering in the force computation. These intrinsics are shown in Figure 8. The force kernel itself runs as well over 80% of the available peak FLOPS.

5 Memory management

Pushing the simulation limits of large-scale structure formation means running simulations with as many particles as possible, and this implies that we're con-

```
1   for ( i = 0, j = 0; i < count1−7; i = i + 8, j = j + 32 )
2   {
3       ...
4       b0 = vec_sub( b0, a1 );
5       c0 = vec_sub( c0, a1 );
6
7       b0 = vec_mul( b0, b0 );
8       c0 = vec_mul( c0, c0 );
9
10      b1 = vec_ld( j, yy1 );
11      c1 = vec_ld( j+16, yy1 );
12      ...
13      b1 = vec_madd( b2, a15, a14 );
14      c1 = vec_madd( c2, a15, a14 );
15
16      b1 = vec_madd( b2, b1, a13 );
17      c1 = vec_madd( c2, c1, a13 );
18
19      b1 = vec_madd( b2, b1, a12 );
20      c1 = vec_madd( c2, c1, a12 );
21      ...
```

Figure 6: An excerpt from the short-range force kernel showing some QPX intrinsics and the basic form of the partially-unrolled loop. The polynomial force evaluation makes use of many fused multiply-add instructions.

```
1   for ( i = 0, j = 0; i < count1−7; i = i + 8, j = j + 32 )
2   {
3       __dcbt( (void *)&xx1 [i+offset] );
4       __dcbt( (void *)&yy1 [i+offset] );
5       __dcbt( (void *)&zz1 [i+offset] );
6       __dcbt( (void *)&mass1[i+offset] );
7       ...
```

Figure 7: An excerpt from the short-range force kernel showing that all interaction-list data is prefetched into the L1 cache.

stantly running as close as possible to the memory limit of the machine. As a result, memory fragmentation [3] becomes a large problem. To make matters worse, HACC is required to allocate and free different data structures during different parts of each time step because there is not enough available memory to hold all such structures at the same time. Furthermore, many of these data structures, such as the RCB tree used for the short-range force calculation, have

[3] Memory fragmentation refers to the condition where small allocations dispersed throughout the memory space leave no large contiguous chunks free even though the total amount of free memory may be large.

```
1  for ( i = 0, j = 0; i < count1−7; i = i + 8, j = j + 32 )
2  {
3      ...
4      b1 = vec_rsqrte( b0 );
5      c1 = vec_rsqrte( c0 );
6      ...
7      b0 = vec_sel( b1, a6, b2 );
8      c0 = vec_sel( c1, a6, c2 );
9      ...
```

Figure 8: An excerpt from the short-range force kernel showing the reciprocal square root estimate and "select" intrinsics.

sizes that change dynamically with each new time step. This, combined with other allocations from the MPI implementation, message printing, file I/O, etc. with lifetimes that might outlast a time-step phase, is a recipe for fatal memory fragmentation problems. Indeed, our first at-scale runs on Mira were limited in duration precisely because of this problem. To mitigate this problem I implemented a specialized pool allocator called Bigchunk. As the name implies, this allocator grabs a large chunk of memory, and then distributes it to various other subsystems. During the first time step, Bigchunk acts only as a wrapper of the system's memory allocator, except that it keeps track of the total amount of memory used during each phase of the time step. Before the second time step begins, Bigchunk allocates an amount of memory equal to the maximum used during any phase of the previous time step plus some safety factor. Subsequent allocations are satisfied using memory from the big chunk, and all such memory is internally marked as free after the completion of each time-step phase. This en-mass deallocation design implies that the bigchunk allocator has minimal overhead, and the time it takes to allocate memory from Bigchunk is very small compared to the speed of the system allocator. Because the Bigchunk memory is not released back to the system, the memory fragmentation problem has effectively been solved.

6 I/O

Prior to the ESP, the initial conditions for a HACC simulation were produced by a stand-alone (parallel) program. This program would produce one input file per rank, which was the only input mode supported by HACC at the time. While the process of producing the initial conditions is fairly fast, the requirement to write hundreds of thousands, or millions, of initial-conditions files, and then read them in again would have been prohibitive. As a result, I integrated the initializer into the HACC simulation code, and enhanced it to use the pencil-decomposed FFT discussed above.

Once HACC could generate its own initial conditions, the next challenge

was making sure it could write out particle snapshots for analysis in a reasonable amount of time (meaning under approximately 30 minutes). After initial experiments using single-file MPI-I/O proved unable to meet our performance requirements, in collaboration with Venkat Vishwanath, I designed and implemented a new file I/O framework, called Generic I/O, capable of meeting our needs. Several important features are:

- For small files, using collective MPI-I/O is necessary, but for large writes using non-collective MPI-I/O or POSIX I/O is necessary. Using the Generic I/O framework, we can choose between any of these three options. On the largest scales, using POSIX I/O yields the best performance on Mira.
- The total output file size is pre-calculated before the writes begin, and the file extent is pre-allocated. This reduces locking contention on the parent directory's metadata.
- To avoid locking contention, we write one file per BG/Q I/O node. There is one I/O node per 128 compute nodes on Mira.
- Each rank writes into a disjoint space (without any kind of data reorganization).
- The file format is self-describing, and in addition to ASCII converters, I created a plugin for the SQLite database engine to make querying data in the files straightforward and flexible.
- 64-bit CRC (Cyclic Redundancy Check) "checksum" codes are stored covering all data and metadata, where each variable from each rank has its own CRC code. This allows the data to be validated before it is used for a simulations checkpoint restart or for analysis.

To the last point regarding CRC codes, although on-disk data corruption is rare, and undetected corruption of data while traversing the network is perhaps rarer still, it can still be a significant problem at scale. As a point of reference, a relatively-small simulation recently run at NERSC created 100 checkpoint files, each one TB is size. Among all of those files, in one file, one variable from one rank was corrupted. This corruption was detected when the checkpoint was read in, and the restart was aborted. Had we not protected the data with CRC codes, we might have not noticed, until many CPU hours had been wasted, that the simulation was partially invalid, and it might have been impossible to later determine why. With the CRC-code protection, the user was immediately alerted to the problem, and was able to restart from an earlier checkpoint. In short, I recommend that all groups make sure that there data is protected by CRC codes.

7 Conclusion

Putting together all of the pieces discussed here has resulted in an astounding improvement of HACC's performance and scalability. HACC is now well positioned to make significant contributions to the science of large-scale structure formation. In addition, with additional processing, HACC's large simulated universes can be used as test beds for the analysis software used on real observational data. None of this would have been possible without the dedicated effort of (in no particular order) Salman Habib, Vitali Morozov, Adrian Pope, Katrin Heitmann, Venkat Vishwanath, Nicholas Frontiere, and many others. Working with all of these people has been a great pleasure, and is a good example of how multi-disciplinary collaborations can yield truly-significant results.

References

[1] Weinberg, D.H., M.J. Mortonson, D.J. Eisenstein, C. Hirata, A.G. Riess and E. Rozo, arXiv:1201.2434 [astro-ph.CO].

[2] Frieman, J.A., M.S. Turner, and D. Huterer, Ann. Rev. Astron. & Astrophys. **46**, 385 (2008).

[3] Gafton, E. and S. Rosswog, Mon. Not. R. Astron. Soc., to appear (2011), arXiv:1108.0028v1

[4] Habib, S., et al., arXiv:1211.4864 [cs.DC].

[5] Habib, S., et al., J. Phys.: Conf. Ser. 180 012019 (2009).

[6] Scott, D. M. "Scaling Turbulence Applications to Thousands of Cores: a dCSE Project" (2010)

8 Accurate Numerical Simulations Of Chemical Phenomena Involved in Energy Production and Storage with MADNESS and MPQC

PI: Robert Harrison *(Stony Brook University, Brookhaven National Laboratory)*

Project Summary

Researchers propose to focus on the problems of catalysis and heavy element chemistry for fuel reprocessing—both of which are of immediate interest to the Department of Energy (DOE), are representative of a very broad class of problems in chemistry, and demand the enormous computational resources anticipated from the next generation of leadership computing facilities. Also common to both is the need for accurate electronic structure calculations of heavy elements in complex environments.

Catalysis: A catalyst greatly improves the efficiency of a desired chemical reaction, and catalytic processes are directly involved in the synthesis of 20% of all industrial products. Within the DOE mission, catalysts feature prominently in cleaner and more efficient energy production, exemplified by the fuel cell and storage technologies. To date, catalysts have been designed largely using trial and error, *e.g.*, synthesizing and testing a potential new catalyst to determine if the reaction is more efficient. This process is both expensive and time-consuming and rarely leads to novel catalysts. Computational modeling and simulation can improve this process, supporting experiment by improved analysis and interpretation of data, and ultimately, in partnership with experiment, enabling the design of catalysts from first principles. Researchers will focus, in collaboration with experimentalists at ORNL, on chemical processes on imperfect metal-oxide surfaces.

Heavy element chemistry for fuel reprocessing: In collaboration with experimentalists and theorists, researchers will focus on two aspects of heavy element chemistry for fuel reprocessing: molecular interfacial partitioning and ligand design. Critical to both are rapid, yet quantitative, models for the interaction of heavy elements with novel organic ligands, and the interaction of both with a multispecies solvent. Speed is essential for combinatorial design due to the evaluation of a huge number of candidates, and also to enable ab initio dynamics for the inclusion of finite temperature and entropy.

Report originally released as ANL/ALCF-ESP-13/6

Fast Linear Algebra Libraries in MADNESS: a Numerical Framework for Quantum Chemistry on Petascale Platforms

Álvaro Vázquez-Mayagoitia and Jeff R. Hammond

May 13, 2013

Abstract

In order to solve the electronic structure of large molecular systems on petascale computers using MADNESS, a numerical tool kit, are required fast and accurate implementations for linear algebra. MADNESS uses multiresolution analysis and low separation rank which translates high dimensional functions in tensor products using Legendre polynomial. The multiple tensor products make to the singular value decomposition and matrix multiplication the most intense operations in MADNESS. This work discusses the interfacing of Eigen3 as a C++ substitute of LAPACK and introduces Elemental for the diagonalization of large matrices. Furthermore, the present paper shows the performance these libraries on Blue Gene/ Q.

1 Introduction

Quantum chemistry has influenced many fields in science by revealing the structure of materials and molecules at atomic level, its most notable achievement is to predict accurately chemical and physical properties. The study of materials at atomic level is a complicated task, even very sophisticated experiments are challenged to reproduce observations at such level. In practice, to obtain values from quantum chemistry methods with meaningful precision requires a huge computational effort. Quantum mechanical methods as the so-called Density Functional Theory (DFT), that approaches the electronic

exchange and correlation, in average reproduces atomization energies within an error of 0.2 eV. Although the availability of large modern computers, DFT currently might compute molecular systems with only few thousands of electrons. [1]

On the other hand, along the last decades the theoretical chemistry has changed drastically in part due to the new developments and approximations to apply the quantum chemistry, and in part to the increment of calculation power. Thus, the unprecedented availability of petascale computers for fundamental research has required new software development according with the last architecture capacities, overall it has bee required software that exploits the technology of multi-cores and multi-processors using large blocks of distributed memory, and new codes that can be reusable. Thus, the synergy of modern quantum chemistry and supercomputing demands the production of new generations of codes able to utilize efficiently large computer systems, and desirably, be able to evolve at the same pace as the novel technologies emerge.

The software Multiresolution Adaptive Numerical Environment for Scientific Simulation (MADNESS) [2] is a general numerical framework for massive parallel computations. MADNESS was designed to reduce the programming effort offering a set of high level tools to solve many dimension integral-differential equations and maximize the science productivity by letting to the programmer to be focused in her or his application instead writing complex low level instructions. MADNESS has been successfully used for several applications in nuclear physics, chemistry, atomic physics, among other areas. MADNESS uses a multiresolution analysis (MRA) that relies on the low separation rank (LSR) representation for functions and operators which lead a generalization of one spatial dimension to higher dimensions and yields algorithms that are too costly for practical applications. The current implementation of MADNESS might operate with a large variety of kernels and boundary conditions. For quantum chemistry MADNESS has an implementation to solve the electronic structure problem with the methods DFT, [3, 4, 5] Hartree-Fock [6, 7] and MP2.[8] This software discretizes the orbital functions within an orthogonal basis sets constructed with Legendre polynomia which in conjunction with the LSR of the local potentials leads to solve linearly the electronic structure at a given arbitrary precision. [9]

This report will summarize part of the effort made to port MADNESS and make it efficient to the IBM Blue Gene/Q technology. The text will discuss briefly the most computational costly parts of the MADNESS calcu-

lations related with linear algebra operations and the interface to external algebra libraries in order to speedup the code. This text will avoid any deep discussion of the formulation of the approaches used and/or the hybrid parallel model found MADNESS, nevertheless we will address to the readers for further details in the correspondent references.

2 Methods

2.1 Numerical Methods Used

The MRA allows to represent a d-dimensional space in d-dimensional boxes, each box with a basis set formed as a of tensor product of Legendre polynomials. The LSR representation of a $3D$ function is written as follows

$$\phi(x,y,z) \approx \sum_{k_1,k_2,k_3}^{r} s_{k_1,k_2,k_3}\varphi_{k_1}(x)\varphi_{k_2}(y)\varphi_{k3}(z)$$

where $\varphi_i(i)$ is a set of orthogonal and polynomial functions and the coefficients s_{k_1,k_2,k_3} are scalar and are adjusted in an adaptive separation rank r to archive a threshold for the accuracy of ϵ that is given by the difference

$$||\phi(x,y,z) - \sum_{k_1,k_2,k_3}^{r} s_{k_1,k_2,k_3}\varphi_{k_1}(x)\varphi_{k_2}(y)\varphi_{k3}(z)||_2 \leq \epsilon,$$

some elements of the rank r may need a refinement to reach faster the required accuracy. This technique is similar in speed as Fast Fourier Transformation used in spectral algorithms on uniform grids. In the practice, the LSR representation in wavelets give us to formulate our functions as a tensor product

$$F = (((S^T D)^T_{k_1} D)^T_{k_2} D)_{k_3},$$

the indexes k_1, k_2, k_3 run over the number of subspaces in the rank used to represent the original function. The matrices D and S are matrices with the filter coefficients and scalar coefficients of the wavelets. We can anticipate that small matrix-matrix operations are the most intense operations in our calculations.

In quantum chemistry, the key equation to obtain the wavefunction of a time-independent system, composed by electrons and ions, is the Schrödinger

equation $\hat{H}\Psi = E\Psi$, where the electronic Hamiltonian $\hat{H}$ is the sum of the kinetic and potential operators $\hat{H} = \hat{T} + \hat{V}$. The numerical low-rank representation of the wavefunctions allows to solve the Schrödinger equation in an integral equation form as: [10, 11]

$$\Psi = -2 \cdot \hat{G}_\mu(V\Psi),$$

where $\mu^2 = (-2 * E)$, and E is the total energy of system. The wavefunction Ψ can be seen as the auxiliary Kohn-Sham wavefunction and can be applied for DFT. The integral operator $\hat{G}_\mu$ can be written as

$$(\hat{G}_\mu * f)(x) = \int \frac{e^{-\mu|x-x'|}}{4\pi|x-x'|} f(x')dx'.$$

Finally the problem is reduced to find a Ψ that minimizes the energy E, and this is solved iteratively since in this approach the energy is variational.

The application of the integral operator implies the convolution of the real-space functions in the LSR representation. The deconvolution operation is also computationally very demanding and requires the decomposition of the coefficients s_{k_1,k_2,k_3} of the tensor products in the low rank separation. decomposes s_{k_1,k_2,k_3} such as The decomposition of the coefficients s_{k_1,k_2,k_3} is the second most expensive operation and is performed using standard Single Value Decomposition algorithms (SVD). Thus the most intensive operations are applied to small matrices, this leads to implement very specific algorithms for those cases.

Additionally, the wavefunctions in the methods DFT and Hartree-Fock are chosen to be real wavefunctions. DFT and HF establishes a formalism of separable particle functions that permits to write the wavefunction of the system as a single Slater determinant $\Psi = \frac{1}{\sqrt{N!}}|\phi_1(r_1)...\phi_N(r_N)|$. In MADNESS, like in any other quantum chemistry code, to obtain the energies of each electronic state needs to solve a problem of eigenvalues, this problem is written as

$$(\widetilde{H} - \epsilon\widetilde{I})\widetilde{S} = 0.$$

The elements of matrix $\widetilde{H}$ are the integrals of the independent particle Hamiltonian $H_{i,j} = \int d\vec{r}\phi_i(\vec{r})^*\hat{h}\phi_j(\vec{r})$ and the elements of $\widetilde{S}$ are the values of the orbital overlap $S_{i,j} = \int d\vec{r}\phi_i(\vec{r})^*\phi_j(\vec{r})$. The dimension of this problem is the size of the basis set used, that is proportional to the number of electrons in systems. For large calculations this is a potential computational bottle

neck to solve the whole wavefunction. The orbital functions $\phi_i(\vec{r})$ initially are built as a linear expansion of Gaussian functions, and for multiresolution representation we chose a finite basis represented by wavelets.

In summary, MADNESS requires for high performance calculations 1) a fast small matrix-matrix multiplication algorithm, 2) a reliable and fast Single Value Decomposition and 3) a parallel eigen-solver for real Hermitian matrices. MADNESS uses by default the external linear algebra subroutines included in BLAS and LAPACK. In order to improve performance of the code, we substituted BLAS/LAPACK with libraries that are more efficient for the matrix dimensions used in MADNESS, which are generally smaller than those for which BLAS/LAPACK are optimized. In the follow sections we will discuss the results when MADNESS uses linear algebra libraries written in C++.

3 Transition from BG/P to BG/Q

The Blue Gene /Q (BG/Q)architecture is a totally new technology for scientific applications and its closest technology reference available is the previous generation Blue Gene /P (BG/P). When we compare the performance between the two generations BG/Q and BG/P in most of cases we experience a speedup of 3x-4x. In the Figure 1 is plotted the comparison between BG/Q and BG/P with same number of nodes and calculation a cluster of 5 water molecules. In a glance, for small calculations BG/Q is 4 times faster than a BG/P with few nodes.

For large molecular systems, where the number of operations are more intensive, BG/Q has a much better performance than BG/P; and this performance grows directly proportional with the size of the problem, see Figure 2.

4 Lineal Algebra Libraries

4.1 Eigen3

The singular value decomposition for small square matrices are one of the most intensive serial operations in MADNESS. We added the option to substitute LAPACK by the templated C++ library Eigen3 [12]. This library

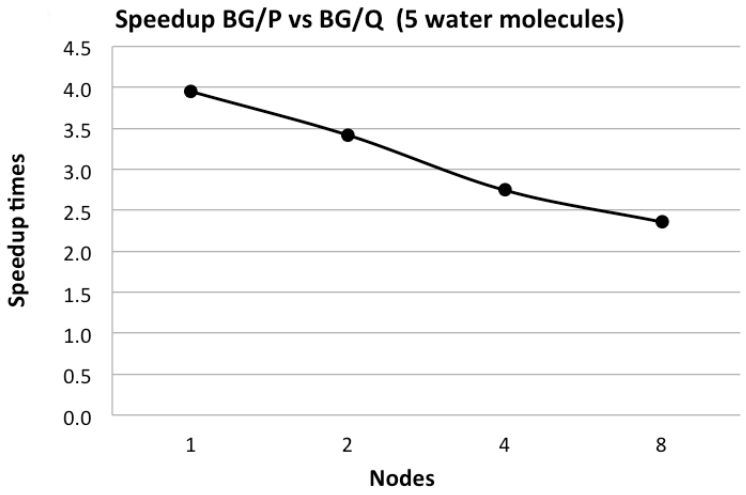


Figure 1: Comparison BG/P vs BG/Q with same number of nodes and same size of molecular system.

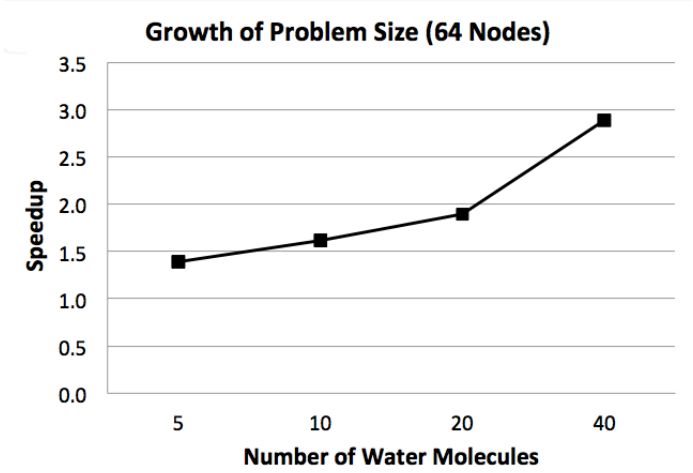


Figure 2: Performance comparison between BG/P and BG/Q when we growth the size of the molecular system computing with 64 nodes.

Program 4.1 Example of single value decomposition application using Eigen3.

```
typedef Matrix<double,Dynamic,Dynamic> MatrixX;
MatrixX U, V, M;
Matrix<double, Dynamic, 1> Sigms;
// init M

// get U and V from M=U.Sigms.V using JacobiSVD decomposition
JacobiSVD <MatrixX> svd(M, ComputeThinU | ComputeThinV);

U = svd.matrixV();
V = svd.matrixU();
Sigms = svd.singularValues();
```

substitutes most of the operations found in LAPACK, providing several algorithms to obtain the eigenvectors and eigenvalues of matrices. Each algorithm could be as fast as the precision required and the size of the matrices involved. Usually more precision means slower calculations. Eigen3 implements a special class to manipulate matrices and some operations for small matrices are hard coded. Initially, the decomposition in the LRS require small matrices operations Eigen was consider an excellent candidate to replace LAPACK in MADNESS.

An example of a small piece of code using Eigen3 to call a SVD calculation is shown in the Program 4.1. Notice that Eigen3 has its own class to represent matrices, `Matrix`. The initialization of matrices from Eigen3 with the matrices in MADNESS is made using pointers. MANDESS and Eigen3 matrix classes have in common objects to insert the directly the elements of the matrices.

In the Figure 3 we show the timings in the decomposition of the real matrix M as $M = U\Sigma V^T$ (where U and V^T are the right and left unitary matrices respectively and Σ is a vector of eigenvalues) with a size (20,20) and smaller using the libraries LAPACK and Eigen3. In this Figure we might notice that Eigen3 is very competitive when computes with small matrices; with sizes less than (16) is faster than LAPACK. Because the SVD procedure is called millions of times, even small increases in performance have a large impact on the overall runtime of the code.

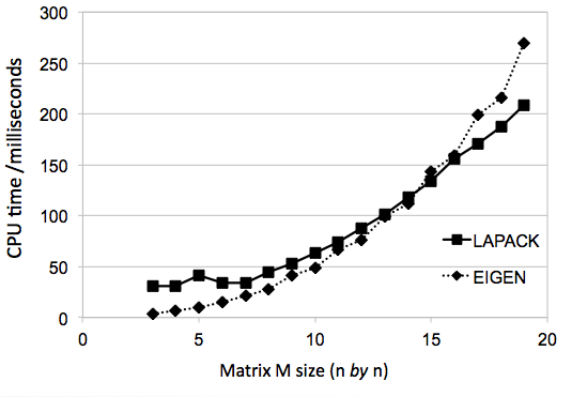


Figure 3: CPU time solving the SVD using Eigen3 and LAPACK in square matrices dimension 20 and smaller.

In spite of the positive results in the performance of Eigen3 for small matrices, its competitiveness is lost with larger matrices, as is shown in the Figure 4 where for medium size and large matrices LAPACK is approximately twice the speed of Eigen3.

4.2 Elemental

In MADNESS in order to obtain orbital energies from the Kohn-Sham or Hartree-Fock methods is necessary to solve a matrix problem of eigenvalues. The dimension of the matrix in our case is the number of occupied orbitals. We interfaced MADNESS matrices to Elemental to facilitate the operations of large matrices and vectors. This is particularly relevant to diagonalize the Hamiltonian matrix and also useful to project the initial Gaussian basis set functions into the polynomial basis set via LRS. For small molecular systems with hundreds of electrons the matrix operations made with LAPACK had negligible times, and single-node libraries where sufficient. Nevertheless, when MADNESS calculates large molecules, with thousands of orbitals, the use of a parallel eigen solver is mandatory, thus, we chose Elemental [13] because it is a modern, object-oriented C++ library that fits with the rest of MADNESS' design and because its performance has been shown to be excellent on Blue Gene systems, i.e. it is faster than ScaLAPACK (this result is not unique to Blue Gene, however).

Elemental maps the MPI processes used on 2D grid and distributes the

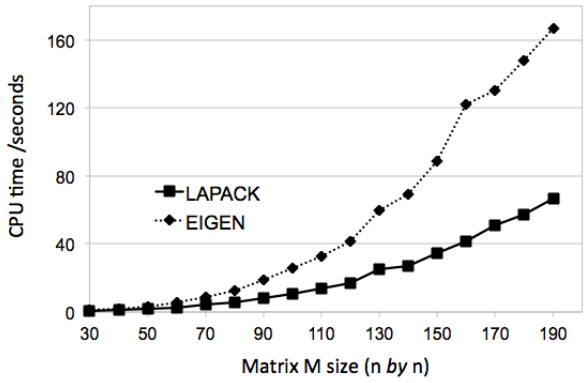


Figure 4: CPU time solving the SVD using Eigen3 and LAPACK in square matrices with dimension between 30 to 200.

matrices data in blocks along that grid. The mathematical operations of the matrices are performed in groups of elements in the grid, the time saved doing this operations are reflected in operation of reduction and collection. Elemental is based in the previous designs found in FLAME and PLAPACK, projects held in the University of Texas, Austin. The snippet in the Program 4.2 exemplify how we call the eigen solver in Elemental using C++. The value `blocksize` should be chosen taking in account the size of the matrix to manipulate and the number of processors. For large matrices in BG/Q (size >3600) we found convenient to set `blocksize=128`.

The Figure 5 shows the performance of Elemental obtaining the eigenvectors and eigenvalues of real matrices with dimension of 3200 to 9000. The speedup values in this plot refer to the timings obtained with 8 nodes of BG/Q. For very large problems, when one passes form 8 to 16 processors the speedup is bigger than the expected, 1.2x . A careful analysis of the plot proves that Elemental is hard to saturate even with 128 processors. The difference between LAPACK and Elemental is extremely large in a multiprocessor scheme, since the former is serial. serial code and the later a parallel one. To exemplify the difference between the too libraries to solve the eigen problem of a matrix with a size 3600 to LAPACK takes 753 seconds, while Elemenetal takes 68.2 seconds when using 16 nodes.

Program 4.2 Example of a solution of an eigenvalues problem kind AxBx using Elemental.

```
const int blocksize = 128; //set block size for the data distribution.
SetBlocksize(blocksize);
Grid GG(MPI_COMM_WORLD); //set grid processors within the MPI rank

// 'int n' is the size of the matrix
DistMatrix<T> B( n, n, GG ), A( n, n, GG );
DistMatrix<double> X( n, n, GG ); //eigenvectors
DistMatrix<double,VR,STAR> w( n, n, GG); //eigenvalues

// init matrices A and B

HermitianGenDefiniteEigType eigType = AXBX; //problem to solve Ax=wBx
UpperOrLower uplo = CharToUpperOrLower('U');
HermitianGenDefiniteEig( eigType, uplo, A, B, w, X ); //get w and X
```

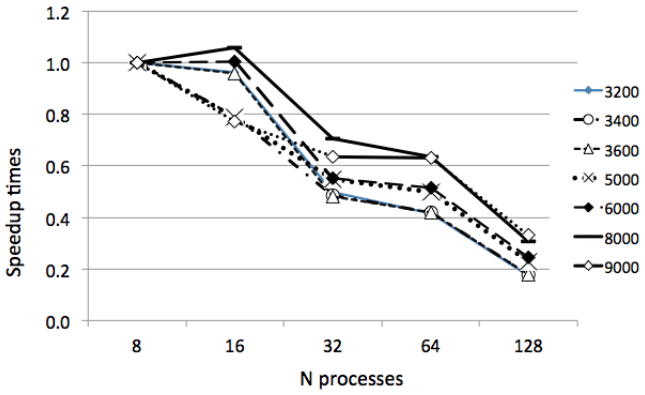


Figure 5: Parallel speedup of Elemental computing the eigenvalues and eigenvectors of matrices with different sizes in BG/Q.

5 Conclusions

MADNESS as a mathematical framework for scientific computation on petascale plataforms was optimized to speedup calculations replacing the most intense subroutines from standard linear algebra packages BLAS/LAPACK to Eigen3 and Elemental, which are more efficient and capable to exploit the particular characteristics of the BG/Q architecture. Eigen3 in BG/Q has better performance than LAPACK for small matrices, up to a size of 16. Nevertheless Eigen3 has a bad performance with bigger matrices. Interfacing of Elemental to MADNESS provides the capability to operate faster with large and distributed matrices, in particular MADNESS was benefited of the parallel eigensolver implemented in Elemental. In the near future we plan to use more features of Elemental to improve MADNESS.

References

[1] W. Koch and M. C. Holthausen, *A chemist's guide to density functional theory*, vol. 2. Wiley-Vch Weinheim, 2001.

[2] Robert J. Harrison, et al., "Multiresolution ADaptive NumErical Scientific Simulation (MADNESS)," 2010. http://code.google.com/p/m-a-d-n-e-s-s/.

[3] R. J. Harrison, G. I. Fann, Z. Gan, T. Yanai, S. Sugiki, A. Beste, and G. Beylkin, "Multiresolution computational chemistry," *Journal of Physics: Conference Series*, vol. 16, no. 1, p. 243, 2005.

[4] R. J. Harrison, G. I. Fann, T. Yanai, and G. Beylkin, "Multiresolution quantum chemistry in multiwavelet bases," in *Computational Science—ICCS 2003*, pp. 103–110, Springer, 2003.

[5] R. J. Harrison, G. I. Fann, T. Yanai, Z. Gan, and G. Beylkin, "Multiresolution quantum chemistry: Basic theory and initial applications," *The Journal of Chemical Physics*, vol. 121, no. 23, pp. 11587–11598, 2004.

[6] T. Yanai, G. I. Fann, Z. Gan, R. J. Harrison, and G. Beylkin, "Multiresolution quantum chemistry in multiwavelet bases: Hartree–Fock exchange," *The Journal of Chemical Physics*, vol. 121, no. 14, pp. 6680–6688, 2004.

[7] F. A. Bischoff and E. F. Valeev, "Low-order tensor approximations for electronic wave functions: Hartree–fock method with guaranteed precision," *The Journal of Chemical Physics*, vol. 134, p. 104104, 2011.

[8] F. A. Bischoff, R. J. Harrison, and E. F. Valeev, "Computing many-body wave functions with guaranteed precision: The first-order møller-plesset wave function for the ground state of helium atom," *The Journal of Chemical Physics*, vol. 137, p. 104103, 2012.

[9] G. I. Fann, R. J. Harrison, G. Beylkin, J. Jia, R. Hartman-Baker, W. A. Shelton, and S. Sugiki, "MADNESS applied to density functional theory in chemistry and nuclear physics," *Journal of Physics: Conference Series*, vol. 78, no. 1, p. 012018, 2007.

[10] B. Alpert, G. Beylkin, D. Gines, and L. Vozovoi, "Adaptive solution of partial differential equations in multiwavelet bases," *Journal of Computational Physics*, vol. 182, no. 1, pp. 149 – 190, 2002.

[11] M. Kalos, "Monte Carlo calculations of the ground state of three-and four-body nuclei," *Physical Review*, vol. 128, no. 4, p. 1791, 1962.

[12] "Eigen3." http://eigen.tuxfamily.org.

[13] "Elemental." https://code.google.com/p/elemental/.

9 Petascale, Adaptive CFD

PI: Kenneth Jansen *(University of Colorado)*

Project Summary

The aerodynamic simulations proposed will involve modeling of active flow control based on synthetic jet actuation that has been shown experimentally to produce large-scale flow changes (*e.g.*, re-attachment of separated flow or virtual aerodynamic shaping of lifting surfaces) from micro-scale input (*e.g.*, a 0.1 W piezoelectric disk resonating in a cavity alternately pushes/pulls out/in the fluid through a small slit to create small-scale vortical structures that interact with, and thereby dramatically alter, the cross flow). This is a process that has yet to be understood fundamentally. Synthetic jet actuators offer the prospect of not only augmenting lift but also other forces and moments in a dynamic and controlled fashion for a range of operating conditions. They have been demonstrated to restore and maintain flow attachment and reduce vibrations in wind turbine blades during dynamic pitch, thereby reducing unsteady loads on gearboxes that are currently the prime failure point. In virtual-shape flight control surfaces for aerial vehicles (including commercial airplanes), conventional control surfaces (*e.g.*, flaps, rudder, etc.) can be augmented or even replaced with active flow control, thus improving their lift-to-drag ratio and/or control power. This projects numerical simulations will give a detailed view of the flow interactions at a Reynolds number and simulation volume approaching engineering application scales for the first time. In these fluid flow problems, anisotropic solution features (like strong boundary and shear layers) can only be located and resolved through adaptivity of the computational mesh.

Report originally released as ANL/ALCF-ESP-13/7

ALCF ESP Technical Report

Petascale Adaptive CFD

PI: Kenneth E. Jansen, University of Colorado Boulder
ESP post-doc: Michel Rasquin, Argonne Leadership Computing Facility

March 2013

1 Description of science

The aerodynamic simulations of this project involve modeling of active flow control based on synthetic jet actuation that has been shown experimentally to produce large-scale flow changes (e.g., re-attachment of separated flow or virtual aerodynamic shaping of lifting surfaces) from micro-scale input [1, 2, 3]. This micro-scale input consists for instance in 0.1 W piezoelectric disk which resonate in a cavity alternately and pushes/pulls out/in the fluid through a small slit to create small-scale vortical structures that interact with, and thereby dramatically alter, the cross flow. This process has yet to be understood fundamentally. Synthetic jet actuators offer the prospect of not only augmenting lift but also other forces and moments in a dynamic and controlled fashion for a range of operating conditions. They have been demonstrated to restore and maintain flow attachment and reduce vibrations in wind turbine blades during dynamic pitch, thereby reducing unsteady loads on gearboxes that are currently the prime failure point. In virtual-shape flight control surfaces for aerial vehicles (including commercial airplanes), conventional control surfaces (e.g., flaps, rudder, etc.) can be augmented or even replaced with active flow control, thus improving their lift-to-drag ratio and/or control power. The goal of the numerical simulations proposed in this project is to provide a complementary and detailed view of the flow interactions at a much higher Reynolds number, approaching engineering application scales for the first time.

2 Overview of numerical methods

2.1 Implicit finite element flow solver PHASTA

Flow computations are performed using a CFD flow solver called PHASTA ("Parallel Hierarchic Adaptive Stabilized Transient Analysis"). This code is based on fully-implicit, stabilized, semi-discrete finite element method for the transient, incompressible Navier-Stokes partial differential equation (PDE) governing fluid flows. In particular, we employ the streamline upwind/Petrov-Galerkin (SUPG) stabilization method to discretize the governing equations [4, 6]. The stabilized finite element formulation currently utilized has been shown to be robust, accurate and stable on a variety of flow problems. In our CFD flow solver, the Navier-Stokes equations (conservation of mass, momentum and energy) plus any auxiliary equations (as needed for turbulence models or level sets in two-phase flow) are discretized in space and time. The discretization in space based on a stabilized finite element method leads to a weak form of the governing equations, where the solution (and weight function) are first interpolated using hierarchic, piecewise polynomials, and followed by the computation of integrals appearing in the weak form using Gauss quadrature. Implicit integration in time is then performed using a generalized-α method which is second-order accurate and provides precise control of the temporal damping to reproduce Gears Method, Midpoint Rule, or any blend in between [5]. On a given time step, the resulting non-linear algebraic equations are linearized to yield a system of linear equations $\mathbf{Ax} = \mathbf{b}$ which are solved using Krylov iterative procedures such as GMRES ("Generalized Minimal RESidual" method).

2.1.1 Parallelization

Finite element methods are very well suited for use on parallel computers as substantial computational effort is divided into two main tasks: 1) the calculation of element level integrals leading to the linear system assembly and 2) the solution of the resulting system of algebraic equations $\mathbf{Ax} = \mathbf{b}$. Both of these work types can be equally divided among the processors by partitioning the aggregate mesh into equal load parts [12, 13]. So far, PHASTA is a pure MPI based code and each process executes of copy of the analysis code to handle the computation work and interactions corresponding to its mesh part. Element-based mesh partitioning is currently used for the domain decomposition approach and leads to a natural parallelization for element-integration/equation-formation stage making it highly scalable. In element-based partitioning, each element is uniquely assigned to a single part but in turn leads to shared

dofs or unknowns in the system of equations, which results from the duplicated vertices on the inter-part boundaries. This element-based partitioning along with the control relationships between multiple images of shared vertices is illustrated in Figure 1.

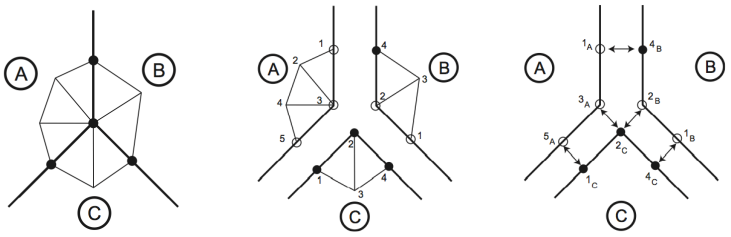
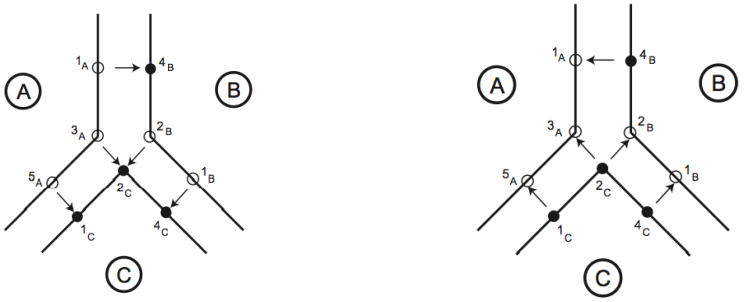


Figure 1: Element-based partitioning and the control relationships between multiple images of duplicated and shared vertices. Solid dot on the middle and right figures denotes an owner image whereas hollow ones indicate non-owners.

Collectively, all the processes have the same information as in the unpartitioned or serial case but no one process holds or has knowledge of the entire tangent matrix, **A**, nor the residual vector **b**. Thus, to be able to progress the computations in parallel and march in time, interactions between shared dofs are completed via communications.

After numerical integration on local part (i.e. task 1 mentioned in the previous paragraph for the system assembly), values in rows of **b** for shared dofs are individually incomplete on each part (referred to as on-part value) because their contributions are distributed among their images (in finite element methods this is due to the compact support of basis or shape functions used). To obtain a complete value for a vector entry associated with a shared dof, peer-to-peer communications related to shared dofs are required and are implemented in two stages, as illustrated in Figure 2. First, data is accumulated at owner image vertices to obtain complete values. Then, complete values are copied from owners to update their non-owner images. Although one could elect to communicate the on-part entries of the tangent matrix **A** to make them complete, our approach does not, limiting communications to vector entries only (such as in **b**). This matrix update will be achieved implicitly during the resolution of the linear system, as explained in the next paragraph.



(a) Non-owners send while owners receive and accumulate.

(b) Owners send whereas non-owners receive and update.

Figure 2: Communication steps involved to obtain complete values for shared dofs.

The second task of our implicit solve involves finding the solution update vector **x**. Krylov iterative solution techniques such as GMRES are currently used for that purpose. These techniques employ repeated products of **A** with a series of vectors (say, **p**) to construct an orthonor-

mal basis of vectors used to approximate $\mathbf{x}$. In this series, the outcome of any matrix-vector product is another vector $\mathbf{q}$ (= $\mathbf{Ap}$) which is used to derive the subsequent vector in the series. The first vector in this series is derived from the residual vector $\mathbf{b}$ which contains complete values at this point of the solve. Even though $\mathbf{A}$ contains only on-part values for shared dofs and is incomplete, it is still possible to perform the basic kernel of (sparse) matrix-vector product, i.e., $\mathbf{q} = \mathbf{Ap}$, provided vector $\mathbf{p}$ has been first updated in the same way as illustrated in Figure 2 for vector $\mathbf{b}$ and contains complete values. Due to the distributive property of $\mathbf{Ap}$ product, the resulting vector $\mathbf{q}$ will in turn contain on-part (incomplete) values. Therefore, after a local $\mathbf{Ap}$ product, our two-pass communication strategy illustrated in Figure 2 must be applied again to obtain complete values in $\mathbf{q}$ (provided $\mathbf{p}$ contained already complete values). Before proceeding to the next product in the series, it is important to note that computation of norms is required to perform orthonormalization. In this step, the norm of vector $\mathbf{q}$, and its dot-product with the previous vectors in the series, are computed. To compute a norm or dot-product, first a local dot-product is computed (requiring no communication) but then, to obtain a complete dot-product, a sum across all cores is needed. A collective communication (of allreduce type) is used to carry out the global summation. To summarize task 2, successive $\mathbf{Ap}$ products are carried out along with peer-to-peer communications to obtain complete values and with global communications to perform the orthonormalization. This series of steps leads to an orthonormal basis of vectors which is used to find an approximate update vector $\mathbf{x}$ and marks the end of a non-linear iteration step.

2.2 Adaptive mesh control and mesh partitioning

In addition to the CFD flow solver PHASTA, adaptive meshing [8, 9, 11, 10] and mesh partitioning [7] techniques are other essential ingredients required to generate and partition significantly large (in the order of 5 billion or more elements) 3D unstructured finite element meshes for the target applications. Indeed, the application of reliable numerical simulations requires them to be executed in an automated manner with explicit control of the approximations made. Since there are no reliable a priori methods to control the approximation errors, adaptive methods must be applied where the mesh resolution is determined in a local fashion based on the spatial distribution of the solution and errors associated with its numerical approximation. Furthermore, the reliability and accuracy of simulations is also a strong function of the mesh quality and configuration. In many physical problems of interest, especially in the field of fluid mechanics, solution features are most effectively resolved using mesh elements which are oriented and configured in a certain manner. For example, in the case of viscous flows, use of boundary layer meshes is central to the ability to effectively perform the flow simulations due to their favorable attributes, i.e., high-aspect ratio, orthogonal, layered and graded elements near the viscous walls. As shown in Section 5, while the quality of the mesh is essential for the reliability and the quality of the solution, the quality of the mesh partition plays a key role in the scalability of the flow solver.

3 Problem size

The usual productive runs on Intrepid concerned cases that included up to O(0.5 billion) finite elements. On Mira, the new available resource allow us to consider typical problems that include O(5 billion) finite elements, that is a factor 10 increase w.r.t. Intrepid. This factor will allow us to increase the Reynolds number of our simulations and approach engineering application scales for the first time.

4 Codes and packages involved in this project

The execution of the flow solver PHASTA and the adaptive meshing and mesh partitioning procedures rely on the following third-party libraries and softwares: MPI, Zoltan, ParMETIS, IPComMan, PIMA, FMDB, phParAdapt, ParMa, Simmetix, ACUSIM and Parasolid/Acis.

5 Performance on Mira

The performance of PHASTA on MIRA is illustrated in Figure 3 and Table 1 with a strong scaling study. These simulations associated with a flow control application are performed on a 3.3 billion finite element mesh. The number of MIRA nodes in this scaling study ranges from 2,048 to 32,768 (corresponding to respectively 16,384 and 524,288 cores) and the number of MPI processes per core varies from 1 to 4. The simulation with 2,048 nodes (32,768 cores) and 1 MPI process per core is considered as the reference hereafter.

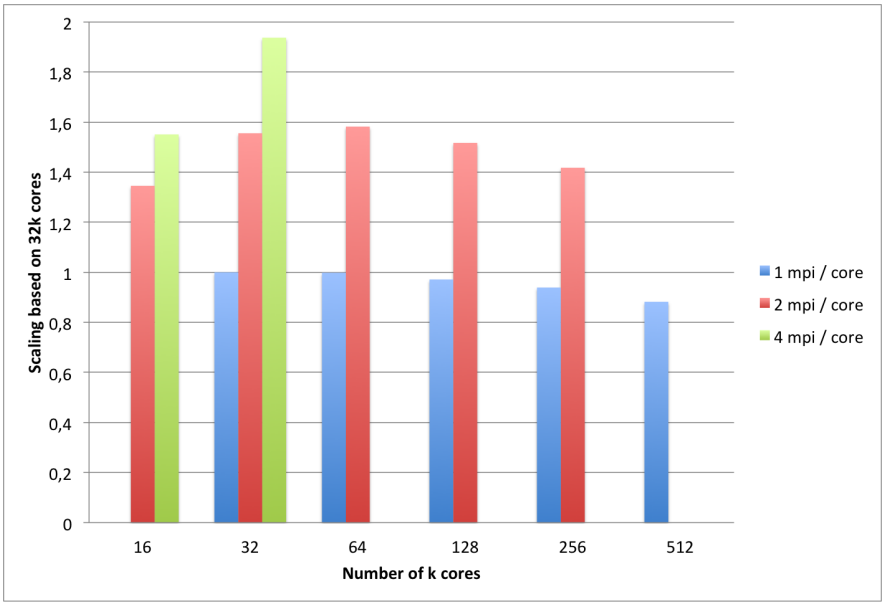


Figure 3: Strong scaling performance for PHASTA on MIRA for a 3.3 billion finite element mesh.

First, a consistent scaling factor of 88.22% is observed on 32,768 nodes (524,288 cores) with one MPI process per core w.r.t. to our reference. Increasing the number of MPI processes per core to 2 or 4 leads to an additional significant improvement of the performance. Indeed, 16,384 nodes (262,144 cores) with one MPI process per core leads to a scaling factor of 92,92%, whereas 2 MPI processes per core leads to a scaling factor of 141,78%. On 2,048 nodes (32,768 cores), 4 MPI processes per core leads to a scaling factor of 193.73%, 2 MPI processes to 155.56% and 1 MPI process per core to 100% (reference case). Finally, MIRA appears to be about 11 times faster than Intrepid on a node basis when 1 MPI process per core is considered on both MIRA and Intrepid. When 2 MPI processes per core are considered on MIRA, the acceleration factor per node rises up to about 18 (with still the standard 1 MPI process per core on Intrepid in this comparison).

As mentioned in the previous section, the scaling of our flow solver strongly depends on the quality of the mesh partitioning. Indeed, one important point to consider during the partitioning of the mesh is that the computational load (in any part) during the system formation stage (i.e., during formation of the tangent matrix $\mathbf{A}$ and residual vector $\mathbf{b}$) depends on the number of elements present in the part, whereas the system solution stage depends on the degrees-of-freedom (dofs) or unknowns in the system of equations on that part, which is proportional to the number of vertices. Typically, element balance (with sufficient load per part) results in a reasonable dof balance as well. As long as the dof balance is preserved, the element-based partitioning also maintains the scalability in the iterative linear solve step, as illustrated in

K Nodes	K Cores	MPI process per core = part/core	K mesh parts = MPI process	Wall Clock (s)	Scaling factor (%) Ref: 2K nodes BG/Q with 1 mpi/core	Mira/Intrepid node factor Ref: 40K nodes BG/P with 1 mpi/core
2 (ref)	32	1	32	206.09	100.00	12.04
4	64	1	64	103.21	99.84	12.02
8	128	1	128	53.04	97.14	11.69
16	256	1	256	27.43	93.92	11.30
32	512	1	512	14.6	88.22	10.62
1	16	2	32	306.31	134.56	16.20
2	32	2	64	132.48	155.56	18.72
4	64	2	128	65.13	158.21	19.04
8	128	2	256	33.96	151.72	18.26
16	256	2	512	18.17	141.78	17.06
1	16	4	64	265.77	155.09	18.67
2	32	4	128	106.38	193.73	23.32

Table 1: Strong scaling results for PHASTA on MIRA with a 3.3 billion finite element mesh.

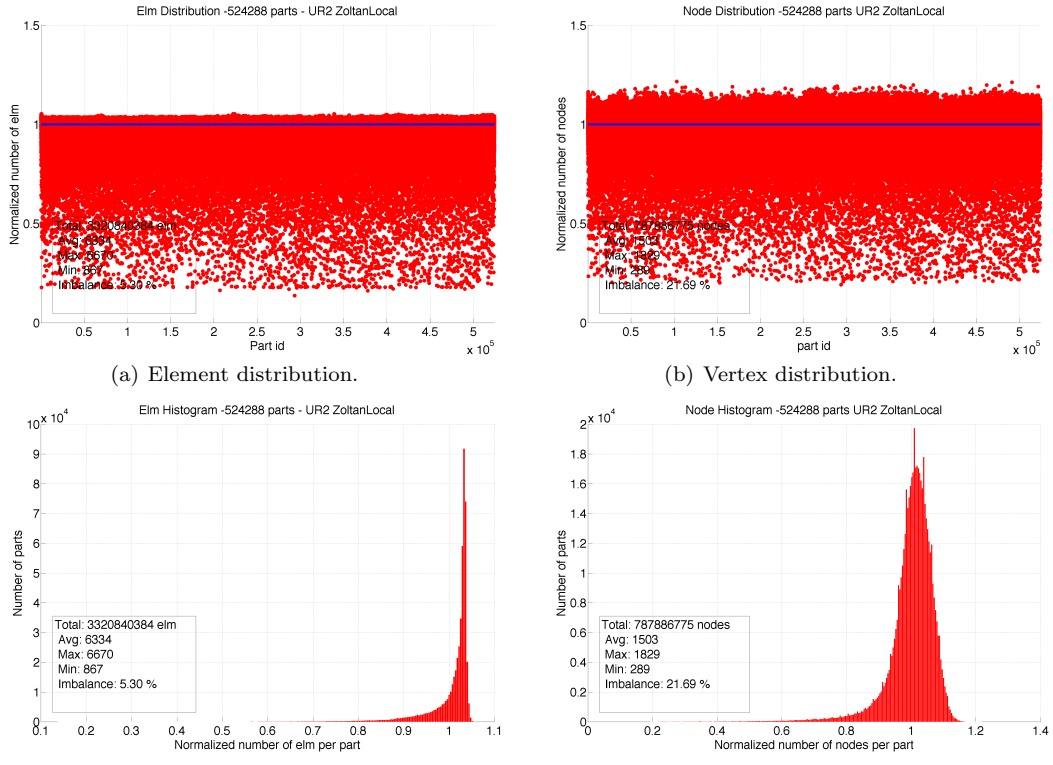
Figure 3 with the simulations performed with 1 MPI process per core.

For a mesh with fixed element topology (e.g. tetrahedra), balanced parts within a partition imply that each part contains as close to the average number of mesh entities (both elements and vertices) as possible. However, in situations where the number of mesh elements per part is relatively small (in the order of few thousand), significant imbalance in dofs can result while the element imbalance remains under control. Indeed, the balance of dofs is not explicitly requested by pure element-base partitioners. This dofs imbalance increase is illustrated in Table 2 and highlighted in particular with the mesh partitioned in 524,288 and 1,572,864 parts. Furthermore, the percentage of shared dofs on part boundaries increases in situations where a fixed-size problem is spread over more and more parts as is the case during this strong scaling study and thus, eventually becomes detrimental to scaling. Finally, it is also common for pure element-based partitioners to generate empty parts in such extreme conditions that flow solvers typically cannot handle.

To illustrate further this dof imbalance as the the mesh is partitioned in more and more parts with a pure element-base partitioner, the mesh entity distribution per part and associated histogram are presented in Figures 4 and 5 for respectively 524,288 and 1,572,864 parts. Figures 4(a) and 4(c) show that most of the parts are well balanced in terms of element. The element histogram also confirms that the node distribution is relatively smooth, as most parts present a number of elements above the target average by only 5.30% at most. However, the conclusion is different for the vertex distribution and histogram presented in Figures 4(b) and 4(d). Indeed, the node distribution is characterized by a number of heavily loaded parts (based on mesh vertices) referred to as spikes, with a number of vertices significantly above above the target average. These spikes are significant contributors to the degradation of the scaling of the iterative linear solver. Moreover, the right tail of the node histogram shows that only a few fraction of the parts are too heavily loaded w.r.t. the target average. This tendency is emphasized in Figure 5, where the element-base partitioning of the mesh in 1,572,864 parts also generates a few empty parts that solvers typically cannot handle. Solutions to remedy to these problems are presented in Section 6 and are currently being implemented.

Parts	32K		64K		128K	
	Elements	Vertices	Elements	Vertices	Elements	Vertices
Total	3.3 billion	628,518,185	cst	658,364,797	cst	690,969,857
Target avg	101,344	19,181	50,672	10,046	25,336	5,272
Max in part	107,087	20,910	53,362	11,053	26,681	5,940
% imbalance	**5.67**	**9.01**	**5.31**	**10.02**	**5.31**	**12.67**
Min in part	56,203	10,790	28,427	5,824	11,993	2,610
Parts	256K		512K		1536K	
	Elements	Vertices	Elements	Vertices	Elements	Vertices
Total	cst	733,192,244	cst	787,886,775	cst	905,248,408
Target avg	12,668	2,797	6,334	1,503	2,111	576
Max in a part	13,340	3,267	6,670	1,829	2,231	875
% imbalance	**5.30**	**16.80**	**5.30**	**21.69**	**5.68**	**51.91**
Min in a part	5,251	1,170	867	289	0	0

Table 2: Element and vertex imbalance for a 3.3 billion finite element mesh



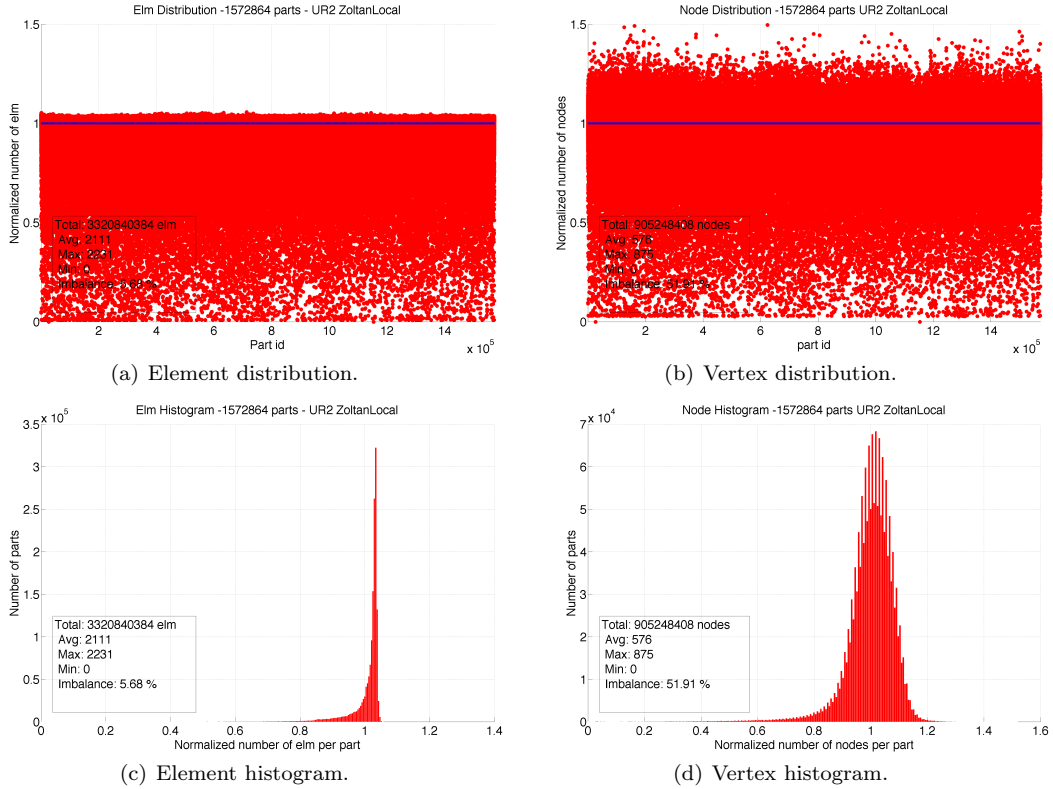
(a) Element distribution.

(b) Vertex distribution.

(c) Element histogram.

(d) Vertex histogram.

Figure 4: Mesh entity distribution per part (top) and histogram (bottom) for a 3.3 billion element mesh partitioned in 524,288 parts with Zoltan and ParMetis. The blue line in the distribution plots represents the target average number of entities per part.



(a) Element distribution.

(b) Vertex distribution.

(c) Element histogram.

(d) Vertex histogram.

Figure 5: Mesh entity distribution per part (top) and histogram (bottom) for a 3.3 billion element mesh partitioned in 1535K parts with Zoltan and ParMetis. The blue line in the distribution plots represents the target average number of entities per part.

6 New algorithms for Mira

6.1 Unstructured mesh partitioning at large scale

Two algorithms aiming at improving the quality of the mesh partitioning are being implemented in a library called Parma, in coordination with the Scientific Computation Research Center at the Rensselaer Polytechnic Institute (RPI). The first algorithm whose first version was presented in [13] consists in migrating locally a small number of elements from heavily loaded parts to relatively lightly loaded neighboring parts in order to reduce the vertex imbalance without perturbing much of the element balance. The result of this algorithm is illustrated in Table 3 and Figures 6 and 7 with an extreme example that includes 180 million elements partitioned in 131,072 parts.

Parts	128K (no Parma)		128K (with Parma)	
	Elements	Vertices	Elements	Vertices
Total	180 million	54,043,396	cst	52,723,643
Target avg	1,378	412	1,378	402
Max in a part	1,455	564	1,604	462
% imbalance	**5.59**	**36.89**	**16.40**	**14.93**
Min in a part	0	0	1	4

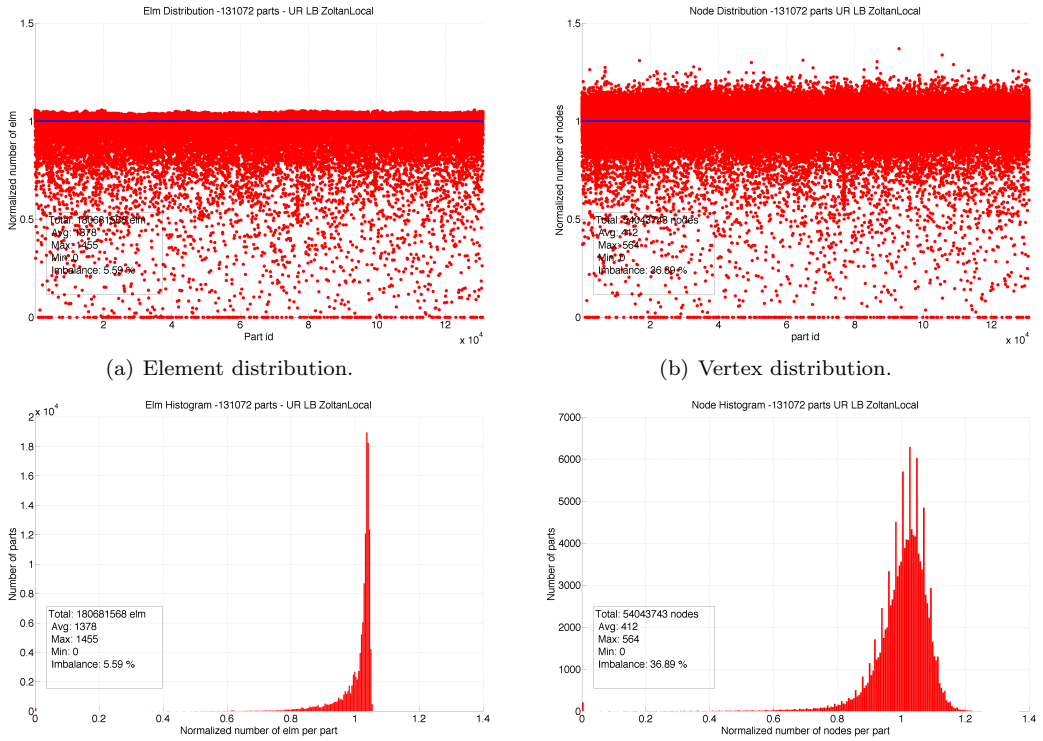
Table 3: Element and vertex imbalance for a 180 million element mesh partitioned in 131,072 parts with Zoltan and ParMetis, and improved with Parma.

From this table and associated figures, one can observe that the vertex imbalance is significantly improved from 36.89% to 14.93%. In this extreme example, the imbalance for both vertices and elements reaches a similar level after this first operation. Migrating a few elements locally from heavily loaded parts to lightly loaded neighboring parts not only reduces the vertex imbalance but also usually decreases the total number of vertices on part boundaries which in turn decreases the total inter part communication. As a first simple solution to fix empty parts, one single element is also migrated from a neighboring part to an empty part.

However, this first algorithm fails at removing all the spikes in the vertex distribution, especially when heavy loaded parts are concentrated in the same region of the mesh. A second algorithm referred to as heavy part splitting approach is currently investigated for that purpose. This algorithm is applied after the smooth element migration described in the previous paragraph and includes three steps. The first step consists in estimating the number of heavy loaded parts in the mesh (both in terms of vertex and elements). The second step consists in generating as many empty parts as needed by grouping lightly loaded parts together. The empty parts potentially generated by the pure element-base partitioning are also recycled for that purpose. Finally, the third step consists in splitting the heavy loaded parts in two or more parts and migrating these new split parts to the empty parts. These heavy loaded parts are split based on their geometry in such a way that the total number of new shared vertices is minimized.

6.2 Live and interactive data co-visualization

Fully implicit finite element methods applied to Computational Fluid Dynamics have already been demonstrated to scale very well up to hundred of thousands of processors, provided that the load in terms of both element and vertex per processor is carefully balanced for respectively the equations formation and iterative solve. This scalability of the flow solver is essential for simulation of turbulent flows that requires the Navier-Stokes equations to be solved on highly refined meshes over a considerable number of small time steps. However, it may take orders of magnitude longer time to perform any reasonable assessment of the insight gained due to the



(a) Element distribution.

(b) Vertex distribution.

(c) Element histogram.

(d) Vertex histogram.

Figure 6: Mesh entity distribution and histogram among the mesh parts for a 180M element mesh partitioned in 131,072 parts with Zoltan and ParMetis but without Parma. The blue line in the distribution plots represents the target average number of entities per part.

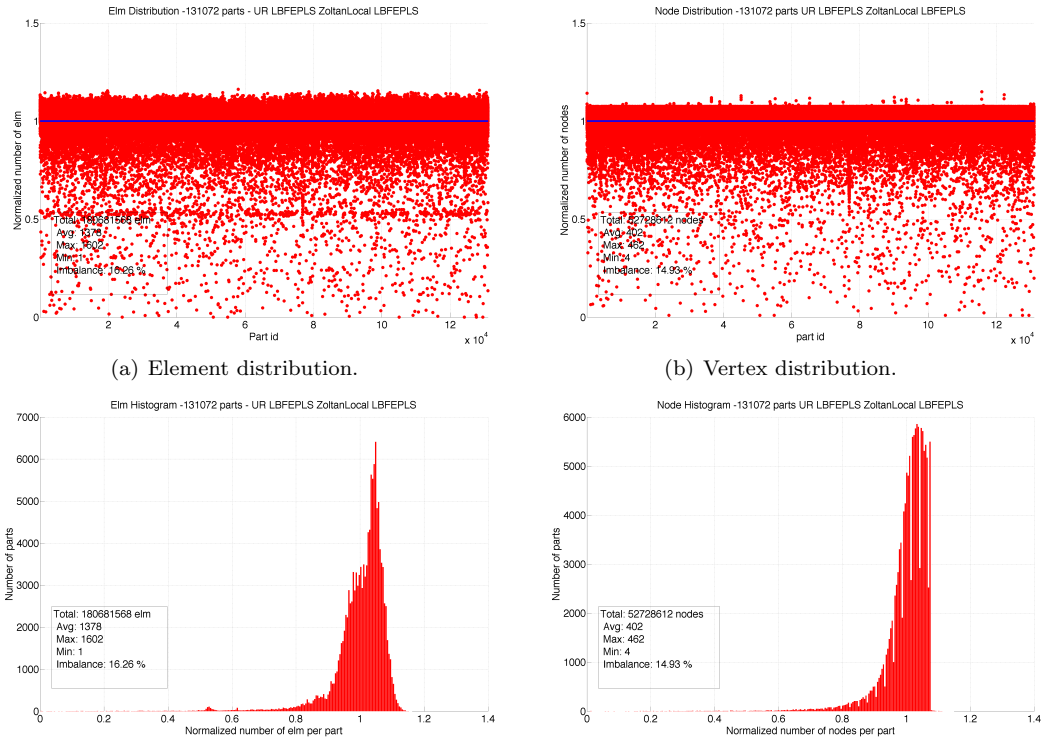
time it takes to write the data, load the data into post processing software, and to analyze and display insightful results. In coordination with the University of Colorado at Boulder, Kitware, INC, the Rensselaer Polytechnic Institute and ALCF, we now consider a more strict definition of "solution" whereby a live data analysis is able to provide continuous and reconfigurable insight into massively parallel simulations, paving the way for interactive simulation and simulation steering. Specifically, we demonstrated our co-visualization concept of either the full data or in situ data extracts on 163,840 cores of the Blue Gene/P Intrepid system tightly linked through a high-speed network to 100 visualization nodes of the Eureka system that share 800 cores and 200 GPUs, as illustrated in Figure 8. In particular, we used this technique to visualize vortical structures that arise from the interaction of a cross flow with an array of synthetic jets on a realistic wing with application to flow control. We plan to port this capability to Mira and Tukey soon.

References

[1] M. Amitay, B.L. Smith, and A. Glezer, *Aerodynamic Flow Control Using Synthetic Jet Technology*, AIAA Paper, 208, 1998

[2] A. Glezer and M. Amitay, *Synthetic Jets*, Annual Review of Fluid Mechanics, 34(1):503-529, 2002.

[3] O. Sahni, J. Wood, K.E. Jansen, and M. Amitay, *Three-dimensional interactions between a finite-span synthetic jet and a crossflow*, Journal of Fluid Mechanics, 671:254-287, 2011.



(a) Element distribution.

(b) Vertex distribution.

(c) Element histogram.

(d) Vertex histogram.

Figure 7: Mesh entity distribution and histogram among the mesh parts for a 180M element mesh partitioned in 131,072 parts with Zoltan and ParMetis, and improved with Parma. The blue line in the distribution plots represents the target average number of entities per part.

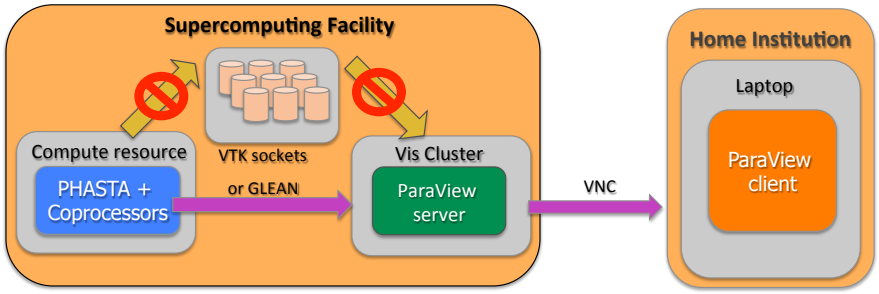


Figure 8: Workflow for live co-visualization of data.

[4] C.H. Whiting and K.E. Jansen *A stabilized finite element method for the incompressible Navier-Stokes equations using a hierarchical basis*, International Journal of Numerical Methods in Fluids, 35:93-116, 2001.

[5] K.E. Jansen, C.H. Whiting and G.M. Hulbert, *A generalized-α method for integrating the filtered Navier-Stokes equations with a stabilized finite element method*, Computer Methods in Applied Mechanics and Engineering, 190:305-319, 1999.

[6] A. K. Karanam, K. E. Jansen, and C. H. Whiting, *Geometry based pre-processor for par-*

allel fluid dynamic simulations using a hierarchical basis, Engineering with Computers, 24(1):1726, 2008.

[7] M. Zhou, O. Sahni, K.D. Devine, M.S. Shephard and K.E. Jansen, *Controlling unstructured mesh partitions for massively parallel simulations*, SIAM Journal of Scientific Computing, 32:3201-3227, 2010

[8] J. Mueller, O. Sahni, X. Li, K.E. Jansen, M.S. Shephard, and C.A. Taylor, *Anisotropic adaptive finite element method for modeling blood flow*, Computer Methods in Biomechanics and Biomedical Engineering, 8(5):295-305, 2005.

[9] O. Sahni, K.E. Jansen, M.S. Shephard, C.A. Taylor, and M.W. Beall, *Adaptive boundary layer meshing for viscous flow simulations*, Engineering with Computers, 24(3):267285, 2008.

[10] M.S. Shephard, K.E. Jansen, O. Sahni, and L.A. Diachin, *Parallel adaptive simulations on unstructured meshes*, Journal of Physics: Conference Series, 78-012053:012053, 2007.

[11] O. Sahni, J. Mueller, K.E. Jansen, M.S. Shephard, and C.A. Taylor, *Efficient anisotropic adaptive discretization of cardiovascular system*, Computer Methods in Applied Mechanics and Engineering, 195(41-43):56345655, 2006.

[12] O. Sahni, M. Zhou, M.S. Shephard, and K.E. Jansen, *Scalable implicit finite element solver for massively parallel processing with demonstration to 160K cores*, Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, 2009, 68:1-68:12.

[13] M. Zhou, O. Sahni, T. Xie, M.S. Shephard and K.E. Jansen, *Unstructured mesh partition improvement for implicit finite element at extreme scale*, The Journal of Supercomputing, 1-11, 2012.

10 Using Multi-scale Dynamic Rupture Models to Improve Ground Motion Estimates

PI: Thomas Jordan *(University of Southern California)*

Project Summary

Researchers will use Southern California Earthquake Center (SCEC) dynamic rupture simulation software to investigate high-frequency seismic energy generation. The relevant phenomena (frictional breakdown, shear heating, effective normal-stress fluctuations, material damage, etc.) controlling rupture are strongly interacting and span many orders of magnitude in spatial scale, requiring high-resolution simulations that couple disparate physical processes (e.g., elastodynamics, thermal weakening, pore-fluid transport, and heat conduction). Compounding the computational challenge, natural faults are not planar but instead have roughness that can be approximated by power laws potentially leading to large, multiscale fluctuations in normal stress. The capacity to perform 3-D rupture simulations that couple these processes will provide guidance for constructing appropriate source models for high-frequency ground motion simulations. SCECs CyberShake system can calculate physics-based (3-D waveform modeling-based) probabilistic seismic hazard analysis (PSHA) curves for California.

On the next-generation Blue Gene, researchers, will calculate a 1Hz PSHA hazard map for California using improved rupture models from our multi-scale dynamic rupture simulations. They will calculate this high-resolution probabilistic seismic hazard map using the technique developed on the SCEC CyberShake project. This calculation will be done after integration of an improved pseudo-dynamic rupture generator into CyberShake system and production of a new and improved UCERF2.0-based Extended Rupture Forecast (ERF). The calculation will provide numerous important seismic hazard results, including a state-wide extended earthquake rupture forecast with rupture variations for all significant events, a synthetic seismogram catalog for thousands of scenario events, and more than 5,000 physics-based seismic hazard curves for California.

Report originally released as ANL/ALCF-ESP-13/8

Using Multi-scale Dynamic Rupture Models to Improve Ground Motion Estimates

PI: Thomas Jordan
ESP Postdoc: Geoffrey Ely

Science Overview

This project uses dynamic rupture simulations to investigate high-frequency seismic energy generation. The relevant phenomena (frictional breakdown, shear heating, effective normal-stress fluctuations, material damage, etc.) controlling rupture are strongly interacting and span many orders of magnitude in spatial scale, requiring high-resolution simulations that couple disparate physical processes (e.g., elastodynamics, thermal weakening, pore-fluid transport, and heat conduction). Compounding the computational challenge, we know that natural faults are not planar, but instead have roughness that can be approximated by power laws potentially leading to large, multiscale fluctuations in normal stress. The capacity to perform 3D rupture simulations that couple these processes will provide guidance for constructing appropriate source models for high-frequency ground motion simulations. The improved rupture models from our multi-scale dynamic rupture simulations will be used to conduct physics-based (3D waveform modeling-based) probabilistic seismic hazard analysis (PSHA) for California. These calculation will provide numerous important seismic hazard results, including a state-wide extended earthquake rupture forecast with rupture variations for all significant events, a synthetic seismogram catalog for thousands of scenario events and more than 5000 physics-based seismic hazard curves for California.

Numerical Method

We simulates spontaneous rupture within a 3D isotropic viscoelastic solid. Wave motions are computed on a logically rectangular hexahedral mesh, using the generalized finite difference method of support operators. Stiffness and viscous hourglass corrections are employed to suppress suppress zero-energy grid oscillation modes. The fault surface is modeled by coupled double nodes, where the strength of the coupling is determined by a linear slip-weakening friction law. External boundaries may be reflective or absorbing, where absorbing boundaries are handled using the method of perfectly matched layers (PML). The hexahedral mesh can accommodate non-planar ruptures and surface topography. Details are described in Ely *et al.* (2008, 2009).

Code

The numerical Method is implemented in the Support Operator Rupture Dynamics code (SORD). SORD is a component of the open-source Computational Seismology Tools (Coseis) which includes other requirements for performing earthquake simulations such as mesh generation, velocity model specification, and visualization. The SORD numerical engine is implemented in Fortran 95, with multi-threaded numerical kernels using OpenMP, domain decomposition using MPI, and parallel I/O using MPI-IO. SORD jobs are configured and launched through the Coseis Python Interface.

Porting to Blue Gene

Prior to ALCF Mira, SORD was first ported to ALCF Intrepid. The Fortran numerical engine required very little modification to run. Substantial effort was required for the supporting utility codes however. Many of these tools required a custom Python install with multiple package dependencies running on the compute nodes; a cumbersome and poorly performing arrangement for Blue Gene. This was improved by reorganized the codes into a two-step process where initial Python-based data processing is performed on the login nodes, with any heavy processing performed by compiled codes on the compute nodes. Additional modifications were needed to the Coseis work-flow tools to adjust to the ACLF Cobalt scheduling environment. Cobalt is a significant departure from prior systems used to run Coseis, such as LoadLeveler, PBS, and SGE. Although the initial porting process required substantial effort, the outcome is highly satisfactory: cleaner, portable, and better performing code.

Performance Tuning

Tests show very good scalability for the MPI domain decomposition scheme on Blue Gene/Q (Fig. 1). However, the initial port achieved only four percent of peak FLOPS performance on Blue Gene/Q. Profiling with Walkup's HPM library revealed memory bandwidth bottlenecks cause by multiple stencil kernel operations sweeping through arrays larger than cache memory. We implemented a cache titling scheme to make better reuse of arrays in cache. We also added OpenMP multi-threading for the kernels. The optimizations achieve a 2.5 times per-core speedup. Taken with the increase core count and bus speed for BG/Q, gives a total of 20 times per-node speedup over the initial BG/P port. QPX-vectorized versions of the computational kernels have been tested, but not integrated into the production code yet.

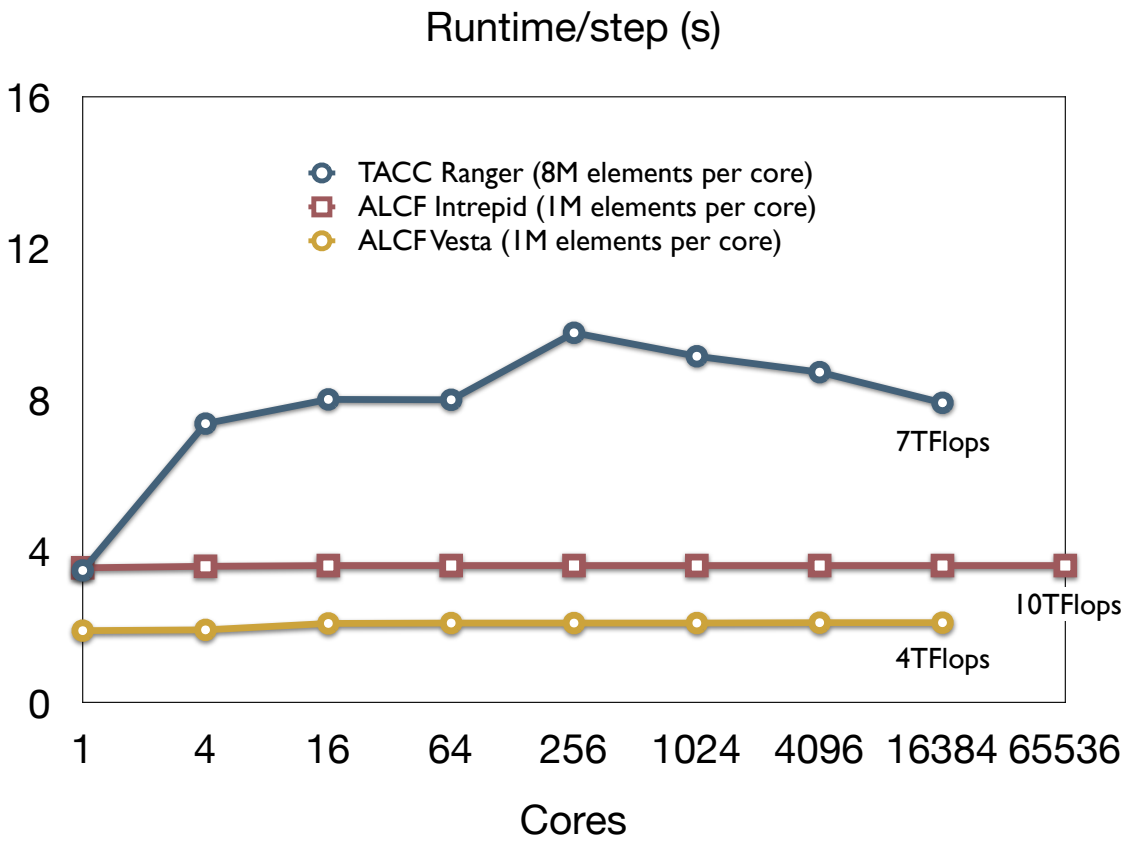
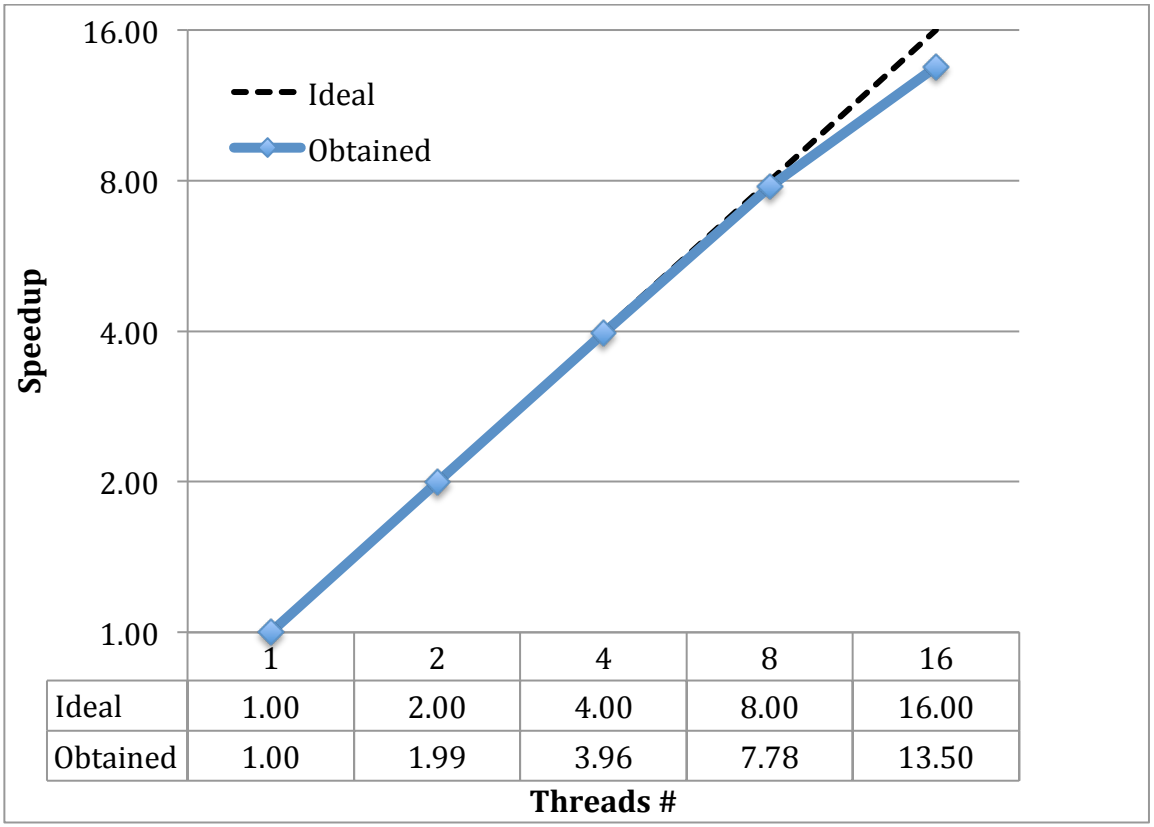


Figure 1: Weak scaling benchmark for SORD in pure MPI mode (no multi-threading). ALCF Intrepid (Blue Gene/P) and Vesta (BG/Q) demonstrate near ideal weak scaling, with BG/Q clock speak increase giving a factor of two speedup relative to BG/P.



	1	2	4	8	16
Ideal	1.00	2.00	4.00	8.00	16.00
Obtained	1.00	1.99	3.96	7.78	13.50

Figure 2: SORD OpenMP strong scaling benchmark for single node Blue Gene/Q.

References

Ely, G. P., S. M. Day, and J.-B. Minster (2008), A support-operator method for visco-elastic wave modeling in 3D heterogeneous media, *Geophys. J. Int.*, *172*(1), 331–344, doi:10.1111/j.1365-246X.2007.03633.x.

Ely, G. P., S. M. Day, and J.-B. Minster (2009), A support-operator method for 3D rupture dynamics, *Geophys. J. Int.*, *177*(3), 1140–1150, doi:10.1111/j.1365-246X.2009.04117.x.

11 High-Speed Combustion and Detonation (HSCD)

PI: Alexei Khokhlov *(University of Chicago)*

Project Summary

This project will gain insight into the physical mechanisms of the burning and detonation of hydrogen-oxygen mixtures. It will produce simulations to be used for the design of safe systems for future use of hydrogen fuel. The goal of the project is to create first-principles, petascale direct numerical simulation tools for understanding and predicting high-speed combustion and detonation (HSCD) phenomena in reactive gases. Researchers want to use first-principles simulations for fundamental understanding of the complex multi-scale physics of the transitory regimes of rapid flame acceleration and deflagration-to-detonation transition (DDT). The next-generation IBM Blue Gene system will enable them to perform first-principles simulations of DDT in a stoichiometric $2H_2 + O_2$ mixture initially at atmospheric pressure in a $100 \times 2.5 \times 2.5$ cm square tube. This is similar to a typical setup of the DDT experiments that measure run distances to detonation in reactive gases. Run distance is a critical parameter used for characterizing sensitivity of a reactive mixture to DDT, and it is used for assessing detonation hazard and designing severe accident mitigation strategies. In the experiments, burning is initiated by igniting a laminar flame in a quiescent gas near the closed end of the tube. As the flame expands, the turbulent boundary layer that forms near the tube walls increases the burning rate, and the flame accelerates rapidly. Secondary shocks and pressure waves generated inside the flame brush add to flame acceleration. Eventually this leads to a localized explosion and the onset of a detonation wave.

Report originally released as ANL/ALCF-ESP-13/9

ALCF ESP Technical Report

High Speed Combustion and Detonation (HSCD)

Alexei Khokhlov, University of Chicago

April 2013

High-level description of science

The goal of the project is to carry out first-principles three-dimensional direct numerical simulations for understanding and predicting high-speed combustion and deflagration-to-detonation transition (DDT) in hydrogen-oxygen gaseous mixtures. DDT and the resulting detonation waves in hydrogen may have especially catastrophic consequences in a variety of industrial and energy producing settings, including the production, transportation and use of hydrogen fuel, and safety on nuclear reactors where hydrogen can be accumulated in cooling pipe systems due to radiolysis of water. We want to use first-principles simulations for fundamental understanding of the physics of the strong, non-linear, multi-scale coupling of the constituent combustion processes leading to DDT, and eventually for predicting the onset of detonation in DDT experiments and engineering devices.

To date, we carried out a series of first-principles reactive compressible Navier-Stokes fluid dynamic simulations of reflected shock bifurcation in $2H_2 + O_2$ and in CO_2 in square tubes and simulations of weak and strong ignition and detonation behind reflected shocks using 8-species reaction kinetics network including H, H_2, O, O_2, OH, H_2O, HO_2, and H_2O_2, multi-species mass diffusion, heat conduction, viscosity, and equation of state relevant to DDT in hydrogen-oxygen mixtures. The theoretical explanation of weak ignition has been an outstanding problem of combustion theory for decades. Three-dimensional first-principle simulations of the process have not been previously performed. They provided new insights into the bifurcation process and the successful simulation of weak ignition is a major accomplishment of the project. The simulations reproduced the two main characteristics of weak ignition - a change in the location of auto-igniting hot spots and the decrease of ignition time compared to one-dimensional predictions. Ignition time delays obtained in our simulations are consistent with experimental data for $2H_2 + O_2$ in both strong and weak ignition regimes.

Code description and numerical methods

First-principles high-speed combustion and detonation (HSCD) problems, DDT in particular, are described by compressible, reactive flow Navier-Stokes (NS) equations of fluid dynamics and must include treatment of shocks. The HSCD code is a distributed memory parallel adaptive mesh refinement (AMR) reactive flow NS code augmented with the equation of state (EOS), microscopic transport, and chemical kinetics suitable for hydrogen combustion. Euler fluxes are calculated using a second-order accurate, Godunov-type, conservative scheme with a Riemann solver. Viscous, mass diffusion, and heat fluxes are calculated using second-order central differencing.

A distinct feature of the code is a dynamic cell-by-cell AMR based on a parallel fully threaded tree (FTT) structure. In ordinary trees pointers are directed from parents to children. In FTT, the pointers are inverted and directed from groups of children to parents and parent's siblings. This arrangement eliminates expensive tree searches which are notoriously difficult to parallelize and it allows all operations, including mesh refinement and de-refinement, to be performed in parallel.

Architecturally, the code consists of three separate layers: (1) the FTT library which provides general services related to all parallel aspects of the code's execution; (2) the reactive Navier-Stokes AMR code, ALLA, which rides on top of FTT and is responsible for numerical integration and AMR; (3) problem-specific algorithms such as material properties routines (equation of state, kinetics, microscopic transport) and problem initialization routines called from ALLA.

The FTT library contains a parallel implementation of a multi-level adaptive computational mesh. To a user, FTT provides a shared view of the mesh with added global services such as I/O, automatic and on-demand load balance, data synchronization across MPI processors, functions for mesh refinement, and iterators for global parallel operations of the mesh. A space-filling curve approach is used in FTT for domain decomposition.

Computations in ALLA are organized as a set of global computational steps, with each step followed by communication work which synchronizes data across the domain decomposition boundaries. Application algorithms are programmed in terms of work-functions which are passed to and executed by the global parallel iterator. The code is parallelized using a hybrid OpenMP/MPI strategy. On each MPI rank the iterator parses the mesh and passes small chunks of cells to a work-function until the entire mesh is processed. The loop over the cells in a chunk is performed inside the work-functions themselves. These loops are parallelized using OpenMP.

The mesh can be refined around shocks, discontinuities and in regions containing large gradients of physical variables such as chemical variables, temperature, vorticity, and so on. AMR is performed every fourth time step after which the cells are rebalanced across the processors using a heuristic to estimate the amount of work required by the cells and to maintain data locality.

The FTT library automatically synchronizes the data in ghost cells after global operations. ALLA algorithms may use cells located at different levels of refinement, and may carry interpolations of physical variables and mesh refinement error indicators between the levels. If parent cells and their children are located on different MPI ranks, the interpolations would require frequent synchronization of ghost cells, and thus additional frequent communication between the MPI ranks. To minimize the amount of communication, the load-balancing heuristic guarantees that after load balancing, children and their parents between the minimum and maximum levels of refinement will be located on the same MPI ranks.

Size/scale of problem possible before Mira, and actually run on Mira

Using an INCITE allocation on BG/P, we were able to simulate and reproduce a process of weak ignition behind a reflected shock in $2H_2 - O_2$ in a 5×5 cm cross-section and 1.6 meter long pipe - something that has not been possible in the past. This work advances us a step closer towards our ultimate goal of understanding and first-principles prediction of DDT. DDT simulations with the same physics in a $5 \times 5 \times 160$ cm tube geometry are being carried out using the current, third-year INCITE allocation and the ESP resources on a new Mira BG/Q supercomputer.

HSCD code has been successfully run on up to 131K cores of BG/P in the OpenMP/MPI mode. Production runs on Intrepid were using up to 64K cores in OpenMP/MPI mode and current production run job sizes on Mira are using 16 racks (1/3 of the full machine).

Modifications in preparation for Mira

The code uses now MPI, OpenMP, VisIt and Silo libraries. In 2012, ALCF staff rewrote the rebalance algorithm to reduce inefficiencies that appear at large scale. The staff also helped port and tune the code on Blue Gene/Q improving the OpenMP implementation and optimized library

use. Algorithmic improvements during the period of the INCITE project running contemporarily to the ESP project include: speedup of the I/O by a factor more than 100 (from costing 100 computational steps to less than one time step), optimization and speedup of the FTT AMR library, and implementation of Silo output for VisIt which solved problems with surface rendering in 3D.

Performance on Mira

There is no thoughtful performance analysis on Mira hitherto other than the comparison of results from time counters and other variables available directly from the code output. However, from these preliminary performance results on the BG/Q Vesta system we know that calculations on BG/Q are $\simeq 2.5$ times faster per core and $\simeq 9$ times faster per node than on BG/P. Figure 1 summarizes strong scaling of the code. The left panel compares code performance on BG/P Intrepid and BG/Q Vesta using a reflected shock tube problem. On both machines, the calculations are dominated by hydrodynamics (including Euler, Navier-Stokes, and chemical reaction terms).

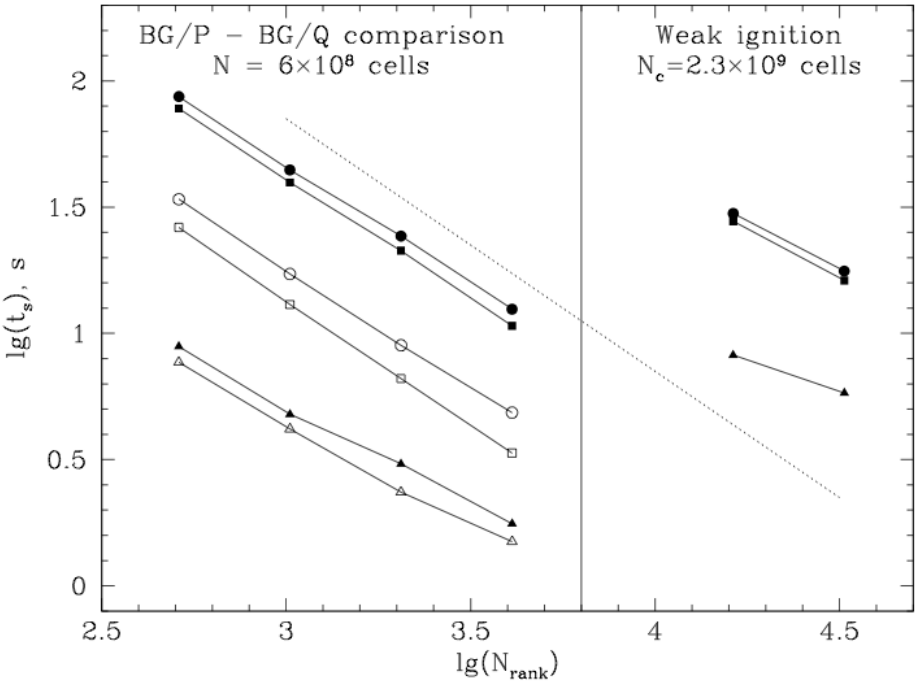


Figure 1: Strong scalability of HSCD code on BG/P Intrepid in MPI/OpenMP mode with 4 threads per rank (black symbols) and on BG/Q Vesta in MPI/OpenMP mode with 16 threads per rank and 4 ranks per node (open symbols). Squares - physics, triangles - AMR and load balance, circles - total. Time t_s is a wall-clock time of one time step. N_{rank} - number of MPI ranks. Left BG/P - BG/Q comparison. Data was averaged over 20 steps when the mesh reached $\simeq 6 \times 10^8$ computational cells in size. Right - data for a weak ignition case for the last hundred time steps of the simulation. Dashed line indicates ideal scaling.

Measurement and performance evaluation

Darshan was used in previous years to acquire data about I/O performance, however, tools like Tuning and Analysis Utilities (TAU), Rice HPCToolkit, IBM HPCT library or even gprof or mpiP

have not been used, at least, in recent versions of the code and on the BG/Q Mira system. Nevertheless, last comparisons of the cost of I/O on Mira at production run sizes (e.g. 16 racks on Mira) show an increase from 1% to 20% of the computational cost of I/O and a set of actions are in progress to get more performance data of the HSCD code. Some of these initiatives are:

- **I/O initiative**: optimize I/O using BG/Q personality and pSet info in order to take advantage of the compute to I/O node ratio (128:1 on Mira/Cetus). Work done in collaboration with Venkatram Vishwanath (MCS/ALCF staff, Performance Team).
- **MPI vs PAMI of one-sided communication section**: the use of PAMI calls instead of MPI for the section of the code that uses one-sided communications is under review. Work done in collaboration with Jeff Hammond (ALCF staff, Performance Team).
- **Visualization improvement**: the visualization subroutines and process will be reviewed by one member of the Visualization Team to tune them. The installation on Tukey of the software needed to visualize HSCD data (VisIt and Silo) have been requested. At the time of this report, we are aware of Silo installation on Tukey and VisIt should be available when Mira goes into production (on April 9th, 2013). Work done in collaboration with Joseph Insley (ALCF staff, Visualization Team).
- **Memory data**: add information about memory consumption on the code output. For commodity systems: mallinfo (C function); for BG/Q: usage of Kernel_GetMemory_Size. Work done in collaboration with Marta García (ALCF staff, Catalyst Team).

Earliest Technical Stories

This ESP project was selected as one of the public relations (PR) stories to go out in ALCF website (and ALCF Newsbytes — March 2013) about what goes on at ALCF. The full article '*High-Speed Combustion and Detonation Project Scaling Up for Mira*' can be found at:

http://www.alcf.anl.gov/articles/high-speed-combustion-and-detonation-project-scaling-mira

12 Petascale Simulations of Turbulent Nuclear Combustion

PI: Donald Lamb *(University of Chicago)*

Project Summary

Type Ia (thermonuclear-powered) supernovae are important in understanding the origin of the elements and are a critical tool in cosmology. These explosions produce a significant fraction of the heavy elements in the universe, and are therefore important in understanding the origin of the elements of which planets and life on Earth are made. Observations using Type Ia supernovae as standard candles revealed that the expansion of the universe is accelerating, which led to the discovery of dark energy. Understanding dark energy ranks among the most compelling problems in all of physical science. Type Ia supernovae are one of the most promising tools for determining the properties of dark energy. This is the purpose of the simulations that the Flash Center will do as an early science user on the next-generation Blue Gene system. To be specific, the Center will use the FLASH code to carry out large-scale, 3-D simulations of two key physical processes in Type Ia supernovae that are not fully understood: (1) buoyancy-driven turbulent nuclear combustion, and (2) the transition from nuclear burning in the flamelet regime to distributed nuclear burning. The simulations will be the largest ever done. The number of grid points in them will exceed by a factor > 20 those in the simulations the Center has done to date on the ALCFs Blue Gene/P computer, Intrepid.

Report originally released as ANL/ALCF-ESP-13/10

Petascale Simulations of Turbulent Nuclear Combustion

Christopher Daley

April 10, 2013

Contents

1 Science objectives

Type Ia (thermonuclear-powered) supernovae are important in understanding the origin of elements and are a critical tool in cosmology. The Flash Center is carrying out large-scale, 3D simulations of two key physical processes in Type Ia supernovae, leading to a better understanding of these explosions.

The two key physical situations for which we are carrying out large-scale 3D simulations are: (1) an initially planar flame in a rectilinear domain with constant gravity and nearly constant density in which turbulence due to the buoyancy of the hot ash drives the burning (which we refer to as RTFlame simulations); and (2) an initially spherical or perturbed spherical flame "bubble" inside of a star in which decaying homogeneous isotropic turbulence drives the burning (which we refer to as DDT simulations). Comparison of the results for these two different physical situations will elucidate the ways in which buoyancy-driven turbulent nuclear combustion differs from turbulent nuclear combustion in a homogeneous, isotropic turbulent background and within a stellar environment.

Application of these results will help improve cosmological distances measured using type Ia supernovae which will ultimately shed light on the nature of Dark Energy. Understanding dark energy ranks among the most compelling problems in all of physical science.

2 The FLASH code

FLASH is a multiphysics, finite-volume Eulerian code containing capabilities suitable for problems in astrophysics, cosmology, high energy density physics and incompressible fluid dynamics. Key capabilities in the early science simulations are Adaptive Mesh Refinement (AMR) which increases resolution in physically important regions of the computational domain, high-order compressible hydrodynamic solvers in directionally split and unsplit formulations which are able to treat non-ideal equations of state (EOS), a multipole self-gravity solver, a flame model and a nuclear energy release model. The code is written in Fortran 90 and C and is parallelized with MPI and more recently OpenMP. It makes use of parallel I/O capabilities provided by either HDF5 or Parallel-netcdf libraries.

The AMR capability is an important feature which can improve time to solution by orders of magnitude compared to a fixed resolution uniform grid and also allows larger problems to be tackled which would not otherwise fit in memory. At the current time, production FLASH simulations use the Paramesh package [1] to provide a block-structured, oct-tree adaptive grid. All blocks/patches in Paramesh contain the same number of cells consisting of internal cells and additional guard cells which store the solution from neighboring blocks. The explicit solvers in FLASH update the solution in the internal cells of "leaf" blocks and then pass control to Paramesh to exchange guard cells and correct fluxes. A leaf block is a block at the finest resolution in each region of the computational domain. Massive parallelism is possible because different blocks are assigned to different MPI tasks.

3 Code changes

We made various code changes so that FLASH early science applications would work well on Mira, the BG/Q platform at the Argonne Leadership Computing Facility (ALCF). The major change of adding hybrid MPI/OpenMP parallelism was part of a longer-term plan and preparation began well in advance of being given access to a BG/Q platform (Section 3.1). Other changes were unplanned and happened after we were given access to BG/Q, but are equally important and necessary to ensure a successful early science program. These include changes for correctness such as altering the source code to avoid issues with the `threadprivate` OpenMP directive on BG/Q (Section 3.2) and creating an unusual FLASH Makefile to obtain expected results (Section 3.3). It also includes changes to improve resource usage such as adding wrapper functions to monitor memory usage (Section 3.4), reducing the amount of communication in a Paramesh initialization subroutine to minimize memory leaks happening in the messaging layer on Mira (Section 3.5) and adding wrapper functions to selectively profile the important parts of FLASH (Section 3.6). The unplanned changes were only possible because of the long early-access period granted by the Early Science Program (ESP).

3.1 Architectural

The FLASH computer code has until recently been a MPI-only code. The MPI-only approach works well on BG/P, where we can run efficiently with 1 MPI rank per core (Virtual Node mode) after making a dedicated effort to reduce the memory footprint of FLASH simulations below 512 MB per MPI rank. In contrast, the approach of 1 MPI rank per core is generally a poor choice on BG/Q which has 4 hardware threads per core. On BG/Q it is highly desirable to place multiple MPI ranks or software threads per core to hide memory latency and pipeline stalls. We chose to multithread the FLASH ESP applications. This is a sensible choice because the new capabilities in the FLASH ESP simulations add to the memory footprint, making it even more challenging to fit in 512 MB per MPI rank and so effectively ruling out even 2 MPI ranks per core on BG/Q.

We have added OpenMP directives to code modules in the FLASH ESP applications including the

hydrodynamics solver, turbulence model, flame model, EOS, and multipole solver. The directives exist at both a coarse-grained and fine-grained granularity in the source code. In the coarse-grained case OpenMP threads update the solution in different Paramesh blocks and in the fine-grained case OpenMP threads update the solution in different cells from the same Paramesh block. The coarse-grained approach is hereafter referred to as thread block list and the fine-grained approach is hereafter referred to as thread within block. In general it is simple to assign independent work to the different threads because of stencil-based or point-wise kernels in the code modules. The one exception is the multipole solver where we needed to create a new moment array for each thread to avoid conflicts during frequent multipole moment updates.

3.2 Working around broken OpenMP features

We encountered problems with the `threadprivate` OpenMP directive on BG/Q which did not occur on BG/P or x86 architectures with various Fortran compilers. We found that our applications would segfault after accessing `threadprivate` data, but we could never reproduce the issue in a small standalone test problem to submit a simple bug report. Our solution was to remove `threadprivate` directives from FLASH by either rewriting code or removing redundant multithreading. The code that required slight rewrites were the Helmholtz EOS and the Multipole solver. In both cases the code frequently read/wrote `threadprivate` module data in the style of old Fortran codes using common block data. The rewrite involved moving this module data to the stack or heap and, where necessary, passing this data through subroutine argument lists. Although it took a little bit of time, these code changes benefit FLASH because it is now possible to use nested OpenMP parallel regions for the first time, e.g. simultaneous use of thread block list and thread within block FLASH multithreaded strategies. Nested parallelism wouldn't have worked in the older version of FLASH because of the rules regarding the persistence of `threadprivate` data [2].

3.3 Creating a custom build for correctness

The biggest issue we had is that we did not get correct FLASH solutions with driver versions before V1R1M2 on Vesta, the test and development BG/Q at ALCF. We found unphysical features in the time history of mass, y-momentum and z-momentum integral quantities. Integral quantities represent the overall state of the simulation and are calculated once per time step by summing over all cells in the computational domain. We found the issues only happened when compiling all FLASH source files with either the OpenMP compiler option or aggressive compiler optimization. The following figures show mass and directional momentum as a function of time in 3 RTFlame test cases run on Vesta BG/Q and Intrepid BG/P. The applications are setup with either the split or unsplit hydrodynamics solver and the parameters which are varied are the flame speed and effective resolution.

Figure 1 shows results from an RTFlame test problem setup with the split hydrodynamics solver and run with a flame speed of 12km/s and an effective resolution of 512^3. The figure includes results from 3 different FLASH builds on Vesta: an MPI-only build, a regular multithreaded build and a selective multithreaded build in which only the files containing OpenMP directives are compiled with the OpenMP compilation flag.

It is clear that there are numerical issues when using a regular multithreaded build of FLASH with either 1 OpenMP thread or 4 OpenMP threads. It is especially concerning that the numerical issues are different in the 1 OpenMP thread and 4 OpenMP threads case because the threads update the solution in independent cells. Figure 2 shows results from an unsplit hydrodynamics build of FLASH but the same test problem. Once again results are bad for a standard multithreaded build of FLASH, but they are also bad for a selective multithreaded build of FLASH with the `-qhot` compilation option. Finally, Figure 3 shows the same issues from a different RTFlame test with flame speed 9km/s and effective resolution 256^3 run to a much later time.

In Figure 1 the standard multithreaded experiments are compiled with `-qsmp=omp:noauto -g -O3 -qnohot -qrealsize=8 -qnosave -c -qthreaded` in this order. We specify the `-qsmp` option first because `-qsmp` (without the `noopt` sub-option) instructs the compiler to optimize as well as parallelize, where, the default optimization is equivalent to `-O2 -qhot` in the absence of other optimization options [3]. In our case the explicit `-O3 -qnohot` at the end of the compilation line should override the implicit optimizations and make the standard multithreaded FLASH build equivalent to the selective multithreaded build. It is therefore puzzling that results depend on the choice of multithreaded FLASH build.

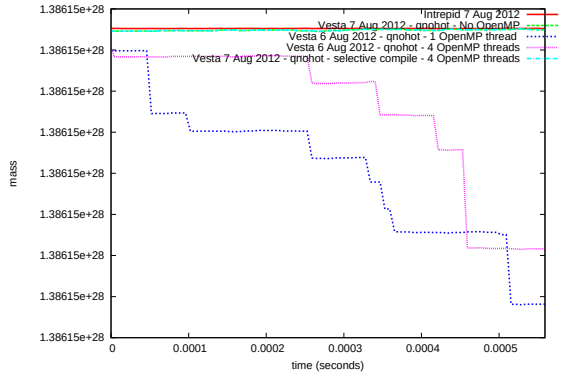
The Vesta results are in good agreement with the Intrepid results when using the selective multithreaded build with `-qnohot` optimization and so we build FLASH in this way for the production early science runs. Recent testing has shown that these issues no longer exist, meaning it is now safe to use `-qsmp=omp:noauto` and `-qhot` on *all* source files. Note that the original issues cannot be reproduced when using the newer V1R1M2 driver but the same May 2012 compiler version.

3.4 Monitoring memory usage

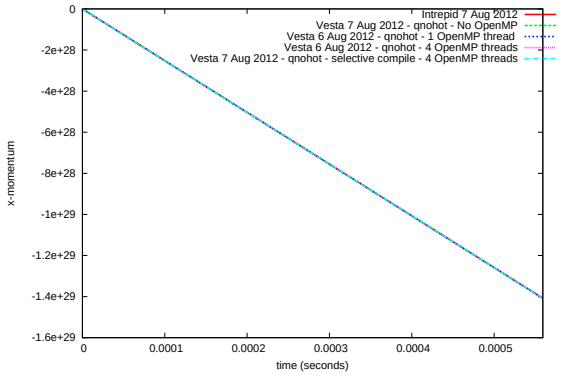
We added wrapper functions around `mallinfo` [4] and `Kernel_GetMemorySize` [5] to monitor FLASH memory usage. This was necessary because the original memory monitoring code in FLASH did not work on BG/Q because of issues with `rusage` [6] on BG/Q. The new memory monitoring code has already allowed us to drill down to a small portion of code in Paramesh which causes a memory leak to happen somewhere in the messaging layer on BG/Q only, see Section 3.5.

3.5 Working around a memory leak

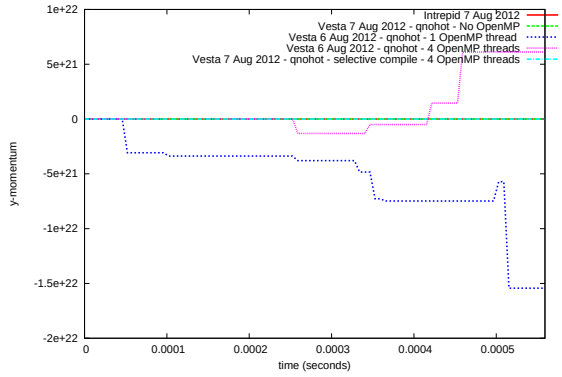
Early runs on Mira revealed an unexpected memory growth during FLASH initialization. We found similar heap memory growth in measurements from both `mallinfo` and `Kernel_GetMemorySize` on Mira BG/Q, but surprisingly no significant memory growth on Intrepid BG/P. The memory growth happens after running a Paramesh subroutine named `find_surrblks` which finds the surrounding block neighbors of every Paramesh block in the domain. It works by passing block metadata around a ring of all MPI ranks in `MPI_Comm_world` using `MPI_Sendrecv_replace` so that each MPI rank can construct a local view of its nearest neighbors in



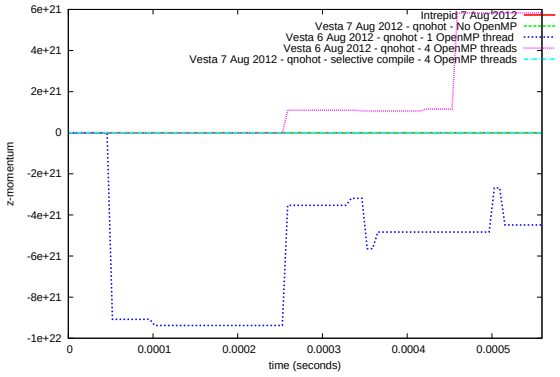
(a) mass



(b) x-momentum

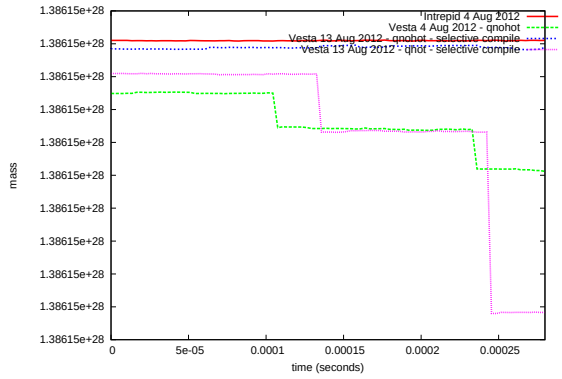


(c) y-momentum

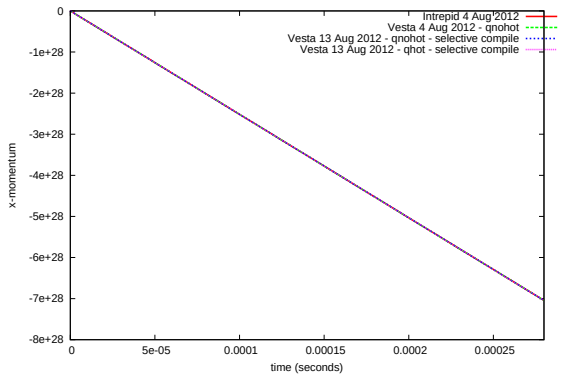


(d) z-momentum

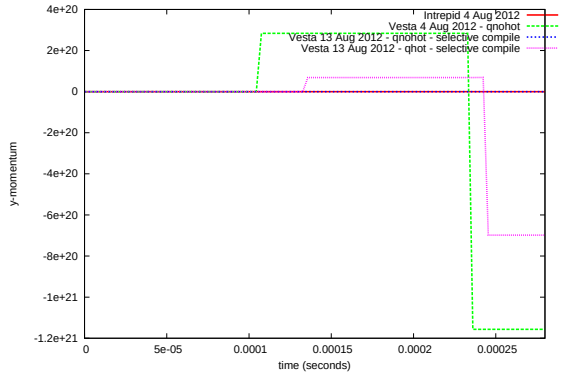
Figure 1: Integrated quantities from a split hydrodynamics RTFlame test problem with flame speed 12km/s and effective resolution 512^3.



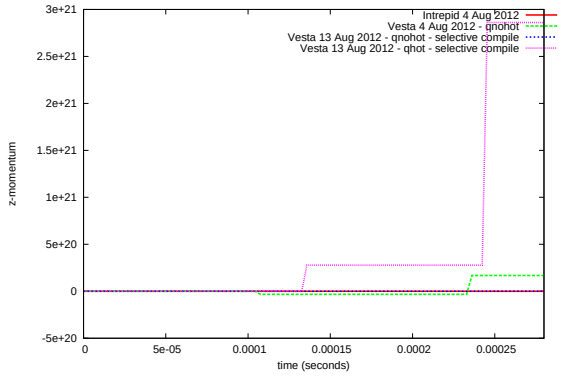
(a) mass



(b) x-momentum

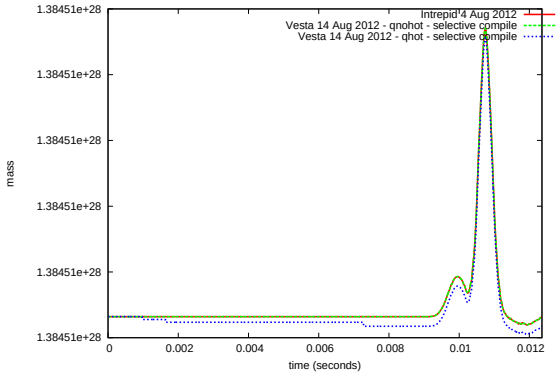


(c) y-momentum

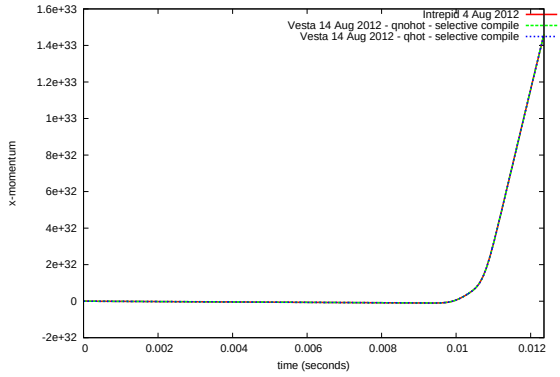


(d) z-momentum

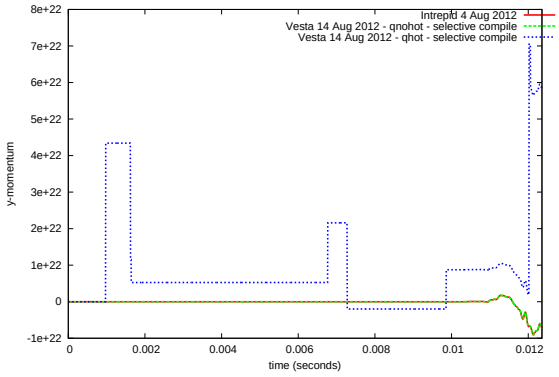
Figure 2: Integrated quantities from an unsplit hydrodynamics RTFlame test problem with flame speed 12km/s and effective resolution 512^3. All Vesta experiments in this figure are run with 4 OpenMP threads.



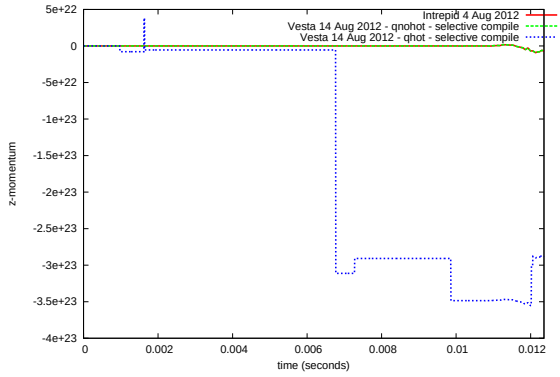
(a) mass



(b) x-momentum



(c) y-momentum

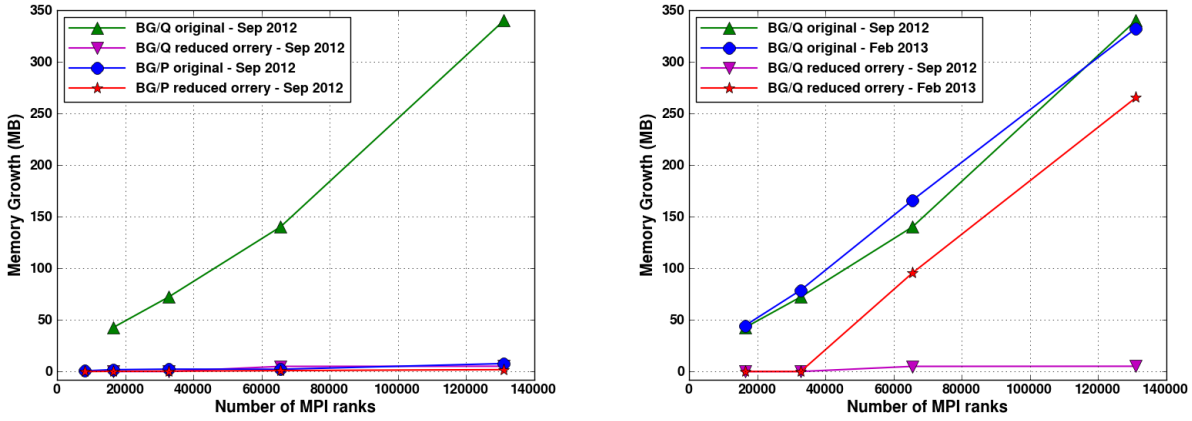


(d) z-momentum

Figure 3: Integrated quantities from an unsplit hydrodynamics RTFlame test problem with flame speed 9km/s and effective resolution 256^3. All Vesta experiments in this figure are run with 4 OpenMP threads.

the global tree. In the past we used this subroutine during initialization and whenever the grid changed during evolution. It is obviously non-scalable. We implemented a scalable method of updating the local view which enabled larger simulations on Intrepid BG/P [7], but the method only works during evolution because it depends on the initial surrounding block neighbors of the top-level blocks. This means there is still a one-time cost at initialization.

We decided to test an optimized version of this subroutine in which the block metadata is circulated in a reduced MPI communicator consisting of only those MPI ranks that own Paramesh blocks. This idea is only applicable at initialization since it depends on there being many more MPI ranks than Paramesh blocks. Figure 4a shows heap memory growth for the MPI rank with the maximum memory usage after executing the `find_surrblks` subroutine, where, heap memory is obtained from `mallinfo (m.hblkhd + m.uordblks)` [8]. We run the FLASH applications in MPI-only mode with 4 MPI ranks per node on BG/P and 16 MPI ranks per node on BG/Q for both the original and optimized version of the code. It is clear that significant memory growth only happens for the original version of the code on BG/Q. Unfortunately, measurements on Mira in February 2013 show that optimized version of the code memory is now also affected by the memory growth issue, although the optimized version does leak approximately 70 MB less than the original version. Figure 4b compares measurements taken in September 2012 with those in February 2013.



(a) (b)

Figure 4: The increase in memory usage after running the original and optimized `find_surrblks` subroutine. (a) shows results from Intrepid BG/P and Mira BG/Q in September 2012, and (b) shows results from Mira BG/Q in September 2012 and February 2013.

We ran a 4096 node FLASH experiment with the `mtrace` memory debugger [9] monitoring all memory allocations that happen during the `find_surrblks` subroutine on MPI rank 64958. This MPI rank is chosen because it had the maximum memory usage after executing the `find_surrblks` subroutine. The memory leaks detected by `mtrace` are shown in Figure 5. These leaks are not in FLASH.

```
Memory not freed:
-----------------
           Address     Size     Caller
0x0000001f80acc020   0xe800  at /bgsys/source/srcV1R1M22012_0907_1744.17581/comm/lib/dev/mpich2/src/util/mem/handlemem.c:204
0x0000001f80ada840   0xe800  at /bgsys/source/srcV1R1M22012_0907_1744.17581/comm/lib/dev/mpich2/src/util/mem/handlemem.c:204
0x0000001f80ae9060   0xe800  at /bgsys/source/srcV1R1M22012_0907_1744.17581/comm/lib/dev/mpich2/src/util/mem/handlemem.c:204
0x0000001f83a2c740     0x30  at /bgsys/drivers/V1R1M22012_0907_1744/ppc64/toolchain/gnu/gcc-4.4.6/libstdc++-v3/libsupc++/new_op.cc:52
0x0000001f83a2c780     0xc8  at /bgsys/source/srcV1R1M22012_0907_1744.17581/comm/sys/buildtools/pami/components/memory/heap/HeapMemoryManager.h:119
0x0000001f83a2c860     0xc8  at /bgsys/source/srcV1R1M22012_0907_1744.17581/comm/sys/buildtools/pami/components/memory/heap/HeapMemoryManager.h:119
0x0000001f83a2ccc0     0x30  at /bgsys/source/srcV1R1M22012_0907_1744.17581/comm/sys/buildtools/pami/components/memory/heap/HeapMemoryManager.h:119
0x0000001f8ab21e80    0xd00  at /bgsys/source/srcV1R1M22012_0907_1744.17581/comm/sys/buildtools/pami/components/memory/heap/HeapMemoryManager.h:119
0x0000001f8ab25d40     0x28  at /bgsys/drivers/V1R1M22012_0907_1744/ppc64/toolchain/gnu/gcc-4.4.6/libstdc++-v3/libsupc++/new_op.cc:52
0x0000001f8ab29c20     0x28  at /bgsys/drivers/V1R1M22012_0907_1744/ppc64/toolchain/gnu/gcc-4.4.6/libstdc++-v3/libsupc++/new_op.cc:52
0x0000001f8ab2d380     0xc0  at /bgsys/source/srcV1R1M22012_0907_1744.17581/comm/sys/buildtools/pami/components/memory/heap/HeapMemoryManager.h:119
0x0000001f8ab2d9e0      0x8  at /bgsys/drivers/V1R1M22012_0907_1744/ppc64/toolchain/gnu/gcc-4.4.6/libstdc++-v3/libsupc++/new_op.cc:52
0x0000001f8ab2da00     0x20  at /bgsys/drivers/V1R1M22012_0907_1744/ppc64/toolchain/gnu/gcc-4.4.6/libstdc++-v3/libsupc++/new_op.cc:52
0x0000001f8ab2da40      0x8  at /bgsys/source/srcV1R1M22012_0907_1744.17581/comm/sys/buildtools/pami/components/memory/heap/HeapMemoryManager.h:119
0x0000001f8ab2da60      0x8  at /bgsys/source/srcV1R1M22012_0907_1744.17581/comm/sys/buildtools/pami/components/memory/heap/HeapMemoryManager.h:119
0x0000001f8ab2da80   0x1374  at /bgsys/source/srcV1R1M22012_0907_1744.17581/comm/sys/buildtools/pami/components/memory/heap/HeapMemoryManager.h:119
0x0000001f8ab2ee00    0x798  at /bgsys/source/srcV1R1M22012_0907_1744.17581/comm/sys/buildtools/pami/components/memory/heap/HeapMemoryManager.h:119
0x0000001f8ab2f640    0x180  at /bgsys/source/srcV1R1M22012_0907_1744.17581/comm/sys/buildtools/pami/components/memory/heap/HeapMemoryManager.h:119
0x0000001f8ab37780   0x2000  at /bgsys/source/srcV1R1M22012_0907_1744.17581/comm/lib/dev/mpich2/src/util/mem/handlemem.c:188
0x0000001f8ab397a0   0xe800  at /bgsys/source/srcV1R1M22012_0907_1744.17581/comm/lib/dev/mpich2/src/util/mem/handlemem.c:204
0x0000001f8ab4a3e0   0xe800  at /bgsys/source/srcV1R1M22012_0907_1744.17581/comm/lib/dev/mpich2/src/util/mem/handlemem.c:204
.................
 253 more leaks of   0xe800  at /bgsys/source/srcV1R1M22012_0907_1744.17581/comm/lib/dev/mpich2/src/util/mem/handlemem.c:204
```

Figure 5: Memory leaks detected by `mtrace` after running the `find_surrblks` subroutine on 4096 nodes of Mira BG/Q in September 2012.

3.6 Adding selective profiling to FLASH evolution

The performance of FLASH on BG/Q is studied using IBM's High Performance Computing Toolkit (HPCT) which provides statement level profiling through `vprof`, hardware counter summary information and MPI performance data. We use the HPCT API to selectively profile FLASH evolution only, i.e. we exclude initialization, which is important for short performance studies. The initialization is always going to be slightly expensive because the initial adaptive mesh needs to be generated from a small number of root blocks which exist on a subset of MPI ranks. We optimized the most expensive part of initialization, see Section 3.5, however, for short performance studies consisting of a small number of time steps the impact of initialization can skew results. Profilers like `vprof` do not provide call-path profiling and so certain subroutines in the call stack of FLASH initialization appear to be more expensive than they actually are. The selective profiling feature allows us to run shorter performance studies and identify expensive parts of FLASH evolution.

4 Optimizations

The early science preparation time has allowed us to increase the level of OpenMP coverage in FLASH and make the multithreading in FLASH truly production ready. In addition to the multithreading we have found opportunities to improve the serial, MPI and parallel I/O performance in FLASH. These improvements were made incrementally during the early science preparation period for both RTFlame and DDT applications. The optimizations for the RTFlame application, which are also usable by the DDT application, are discussed in Section 4.1 and the DDT application specific optimizations are discussed in Section 4.2.

4.1 RTFlame

Since it is interesting to see the performance impact of each incremental change, we take the current multithreaded FLASH code and revert all serial and MPI optimizations. We successively apply the ESP optimizations and then profile each transient version of the code. The baseline measurement is obtained from a FLASH binary compiled with `-O3 -qnohot`. These non-aggressive compilation flags were used by Vitali Morozov when he successfully ported FLASH to an early access BG/Q machine. The effect that each optimization has on time to solution is shown in Figure 6 and the description of the optimizations follow. In the figure blue bars indicate optimizations which are currently being used in the early science production runs and red bars indicate optimizations which are not being used. The optimizations that are in-use reduce time to solution by 32.6% in this test problem which is approximately a 1.5x performance improvement.

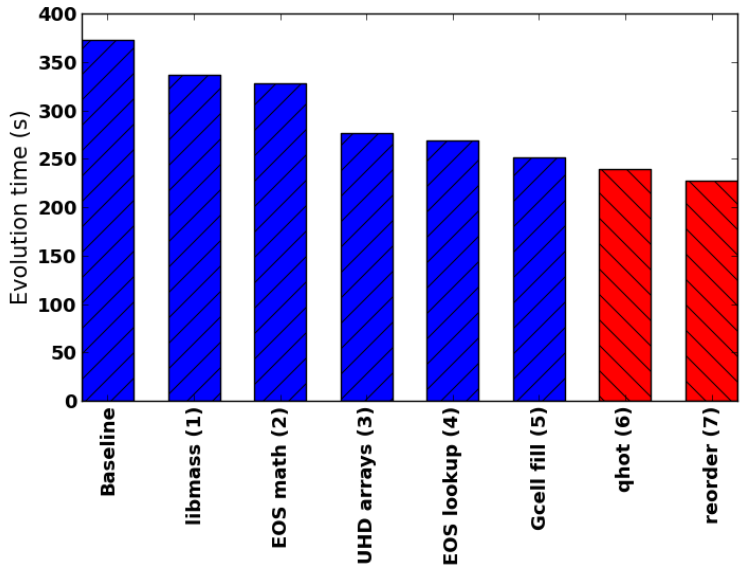


Figure 6: Time to complete 50 steps of an RTFlame test problem on Vesta BG/Q after applying each new optimization. The simulations were run with 16 MPI ranks per node, 4 threads per MPI rank and used the thread within block multithreading strategy.

Linking against the MASS library - libmass (1)
The baseline `vprof` profile shows a large number of counts in symbols `__log` and `__ieee754_log` which are glibc log functions. An extremely simple optimization is to use the accelerated log functions in the IBM Mathematical Acceleration Subsystem (MASS) library instead of those in glibc. The calls are inserted automatically when compiling with aggressive optimization [3], however, aggressive flags, such as `-qhot`, originally led to incorrect FLASH answers as described in Section 3.3. Linking against `libmass.a` explicitly allows us to use the accelerated math functions without needing to compile FLASH with aggressive optimization.

Modifying the Math expressions in EOS - eosmath (2)
The next change is a contribution from Vitali Morozov which consists of faster math computations in the Helmholtz EOS. He replaced the expression `x4=plasg**(0.25e0)` with `x4=sqrt(sqrt(plasg))` and `inv_lami=s1**(1.0e0/3.0e0)` with `inv_lami=sqrt3(s1)`, where `sqrt3` is a fast cube root function which implements Halley's method [10].

Changing the array layout in the unsplit hydrodynamics solver - UHD arrays (3)
The earlier `vprof` profiles show that a large amount of time is spent calling subroutines in the FLASH subroutine `hy_uhd_getRiemannState`. This is understood and happens because the compiler must add code to copy non-contiguous array slices into temporary contiguous arrays for the purpose of the subroutine call. The optimization involves reordering several arrays so that the i, j, k indices are the slowest varying dimensions. A snapshot of the `vprof` profiles for the original and optimized code is shown in Figure 7. On the far left of the figure is the source line number, next to that is the number of clock ticks, where each tick corresponds to 0.01 seconds of CPU time, and on the right is the actual source.

Improving the table lookup locality in EOS - EOS lookup (4)
The `vprof` profiles indicate that interpolation of table lookup data in Helmholtz EOS is expensive. This is partly because the lookup quantities exist in separate arrays which hurts the locality. We show just the bicubic hermite polynomial function for electron positron number densities in the before and after `vprof` fragments in Figure 8. In the original code, `eos_xf` is the electron positron number density and `eos_xfd`, `eos_xft`, `eos_xfdt` are various derivatives of this number density. Performance improvement is possible by placing all electron positron number density quantities next to each other in a derived datatype named `eos_tbl` which has fields `xf`, `xft`, `xfd` and `xfdt`. Note that there are many more fields in this derived datatype which are there to improve the access locality in other Helmholtz EOS functions.

Reducing the number of guard cell fills - Gcell fill (5)
The next optimization involves reducing the number of MPI synchronization points needed to keep guard cells (ghost cells) updated. Frequent guard cell exchanges happen because FLASH applications consist of multiple physics units called in sequence which update solution data according Strang operator splitting [11]. It is essential that the different physics units access current guard cell data to correctly update the solution data and so the safe convention adopted in FLASH is that any unit which accesses guard cells is responsible for making a call to the mesh package to update guard cells. This approach works well and avoids many bugs, but has the side-effect that guard cells can be exchanged too often. In the case of the RTFlame application there are guard cell exchanges in Hydrodynamics, Flame and Lagrangian Tracer Particles units as well as a simulation specific analysis routine. This results in too many guard cell exchanges because there are no writes to mesh data when updating Lagrangian tracer particles or performing simulation specific analysis. Since the guard cells are still valid we can safely remove two guard cell exchange synchronization points.

Compiling with aggressive optimization - qhot (6)
The performance can be improved by using more aggressive compiler optimization, however, this originally resulted in incorrect FLASH answers as discussed in Section 3.3.

Reordering the Paramesh data arrays - reorder (7)
The core grid data structure in Paramesh can be reordered so that the fastest varying dimension is no longer

```
 688        !! Left and right Riemann state reconstructions
 689  1003  call hy_uhd_dataReconstOnestep&
 690             (blockID,blkLimitsGC,i,j,k,dt,del,&
 ...              ...
 709              sig    (1:NDIM,1:HY_VARINUMMAX,i,j,k), &
 710              lambda(1:NDIM,i,j,k,1:HY_WAVENUM),   &
 711              leig   (1:NDIM,i,j,k,1:HY_WAVENUM,1:HY_VARINUM),&
 712              reig   (1:NDIM,i,j,k,1:HY_VARINUM,1:HY_WAVENUM) )
 ...
1039        !! ============ x-direction ============================================
1040        ! YZ cross derivatives for X states
1041   838  call upwindTransverseFlux&
1042           (hy_transOrder,sig(DIR_Z,:,i,j-2:j+2,k),lambda(DIR_Y,i,j,k,:),leig(DIR_Y,i,j,k,:,:),&
1043           reig(DIR_Y,i,j,k,:,:),TransFluxYZ(:))
1044
1045        ! ZY cross derivatives for X states
1046   728  call upwindTransverseFlux&
1047           (hy_transOrder,sig(DIR_Y,:,i,j,k-2:k+2),lambda(DIR_Z,i,j,k,:),leig(DIR_Z,i,j,k,:,:),&
1048           reig(DIR_Z,i,j,k,:,:),TransFluxZY(:))
```

(a) Original code

```
 688        !! Left and right Riemann state reconstructions
 689    72  call hy_uhd_dataReconstOnestep&
 690             (blockID,blkLimitsGC,i,j,k,dt,del,&
 ...              ...
 709              sig    (1,1,i,j,k), &
 710              lambda(1,1,i,j,k),   &
 711              leig   (1,1,1,i,j,k),&
 712              reig   (1,1,1,i,j,k) )
 ...
1039        !! ============ x-direction ============================================
1040        ! YZ cross derivatives for X states
1041   157  call upwindTransverseFlux&
1042           (hy_transOrder,sig(:,DIR_Z,i,j-2:j+2,k),lambda(1,DIR_Y,i,j,k),leig(1,1,DIR_Y,i,j,k),&
1043           reig(1,1,DIR_Y,i,j,k),TransFluxYZ(:))
1044
1045        ! ZY cross derivatives for X states
1046   174  call upwindTransverseFlux&
1047           (hy_transOrder,sig(:,DIR_Y,i,j,k-2:k+2),lambda(1,DIR_Z,i,j,k),leig(1,1,DIR_Z,i,j,k),&
1048           reig(1,1,DIR_Z,i,j,k),TransFluxZY(:))
```

(b) Optimized code

Figure 7: `vprof` profiles showing the impact of changing the array layout in the unsplit hydrodynamics solver.

the mesh variable. This change is too experimental to be used in the FLASH early science campaign.

4.2 DDT

The test DDT simulation originally took over 2 hours to initialize on 8192 MPI ranks on Mira BG/Q. We found that a significant amount of this time was spent reading turbulence field data from a small HDF5 file. This is extremely puzzling because the application code made use of HDF5 parallel I/O and requested collective I/O data transfers. We analyzed core files from a run that exceeded available wall-

```
360          h3x(i,j,w0t,w1t,w0mt,w1mt,w0d,w1d,w0md,w1md) =  &
361                 (eos_xf(i,j)     *w0d   +   eos_xf(i+1,j)     *w0md                    &
362               +  eos_xfd(i,j)    *w1d   +  eos_xfd(i+1,j)     *w1md) *w0t              &
363               + (eos_xf(i,j+1)   *w0d   +   eos_xf(i+1,j+1)   *w0md                    &
364               +  eos_xfd(i,j+1) *w1d   +  eos_xfd(i+1,j+1)   *w1md) *w0mt             &
365               + (eos_xft(i,j)    *w0d   +  eos_xft(i+1,j)     *w0md                    &
366               +  eos_xfdt(i,j)   *w1d   + eos_xfdt(i+1,j)     *w1md) *w1t              &
367               + (eos_xft(i,j+1) *w0d   +  eos_xft(i+1,j+1)   *w0md                    &
368               +  eos_xfdt(i,j+1)*w1d   + eos_xfdt(i+1,j+1)   *w1md) *w1mt

630          !!  electron + positron number densities
631   290    xnefer    = h3x(iat,jat, &
632                          si0t,    si1t,    si0mt,    si1mt, &
633                          si0d,    si1d,    si0md,    si1md)
```

(a) Original code

```
356          h3x(i,j,w0t,w1t,w0mt,w1mt,w0d,w1d,w0md,w1md) =  &
357                 (eos_tbl(i,j) % xf     *w0d   +   eos_tbl(i+1,j) % xf     *w0md          &
358               +  eos_tbl(i,j) % xfd    *w1d   +  eos_tbl(i+1,j) % xfd     *w1md) *w0t    &
359               + (eos_tbl(i,j+1) % xf   *w0d   +   eos_tbl(i+1,j+1) % xf   *w0md          &
360               +  eos_tbl(i,j+1) % xfd  *w1d   +  eos_tbl(i+1,j+1) % xfd   *w1md) *w0mt &
361               + (eos_tbl(i,j) % xft    *w0d   +  eos_tbl(i+1,j) % xft     *w0md          &
362               +  eos_tbl(i,j) % xfdt   *w1d   + eos_tbl(i+1,j) % xfdt     *w1md) *w1t    &
363               + (eos_tbl(i,j+1) % xft  *w0d   +  eos_tbl(i+1,j+1) % xft   *w0md          &
364               +  eos_tbl(i,j+1) % xfdt*w1d   + eos_tbl(i+1,j+1) % xfdt   *w1md) *w1mt

588          !!  electron + positron number densities
589    82    xnefer    = h3x(iat,jat, &
590                          si0t,    si1t,    si0mt,    si1mt, &
591                          si0d,    si1d,    si0md,    si1md)
```

(b) Optimized code

Figure 8: `vprof` profiles showing the impact of storing EOS lookup data in a derived datatype.

time and found that some MPI ranks were in the call-stack of the function `MPI_File_read_at` at job end-time. This is an independent MPI-IO function! We created wrapper functions around the HDF5 API functions `H5Pget_mpio_actual_io_mode` (HDF5 >= 1.8.8) and `H5Pget_mpio_no_collective_cause` (HDF5 >= 1.8.10) to help understand the I/O data transfer. The functions returned `H5D_MPIO_NO_COLLECTIVE` and `H5D_MPIO_DATATYPE_CONVERSION` indicating that a datatype conversion prevented a collective data transfer. This conversion happens because the HDF5 dataset contained little-endian data (`H5T_IEEE_F64LE`) but Blue Genes are big-endian machines. Converting the HDF5 dataset in the file to big-endian (`H5T_IEEE_F64BE`) fixed the performance issue. Figure 9 shows how the read time is affected by the endianness of the HDF5 data and the requested data transfer mode. The results clearly show that a collective I/O transfer with big-endian data is approximately 2 orders of magnitude faster than an independent I/O transfer. It is also clear that the failed collective I/O transfer with little-endian data gives nearly identical performance to an independent I/O transfer. For reference, a similar issue happens when converting from single-precision in memory to double-precision in file [12].

A large portion of time was also spent generating the initial positions for the Lagrangian tracer particles. We traced the slow-down to frequent calls to the `random_number` Fortran function, where we found many initial calls followed by a number of additional calls proportional to the MPI rank in order to give different random numbers on each MPI rank. We fixed the performance issue by removing all the unnecessary initial `random_number` calls which is valid because the only requirement in this application is that the initial density

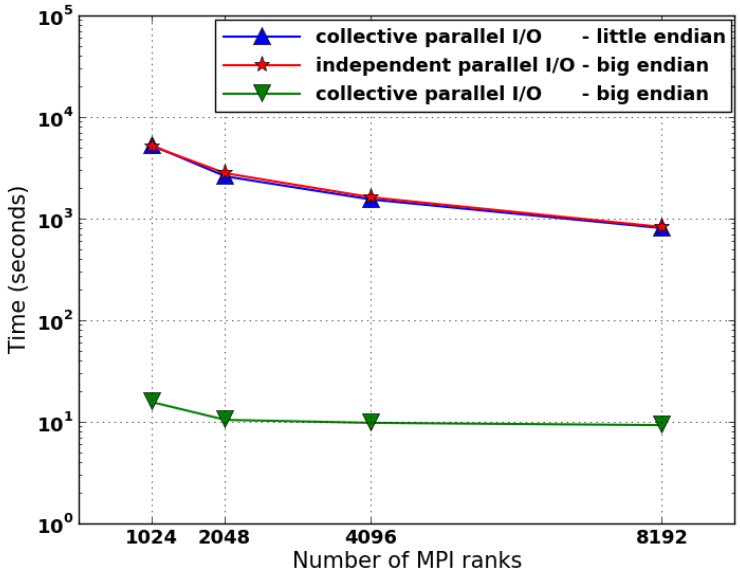


Figure 9: Time to read the turbulence field file on 64, 128, 256 and 512 nodes of Vesta BG/Q. The simulations are run with 16 MPI ranks per node in MPI-only configuration.

of particles is proportional to the gas density. We did not encounter the performance issue on other platforms because those runs used fewer MPI ranks and presumably the `random_number` function was faster. In a brief test we found a single call to `random_number` takes approximately $30ns$ on a x86_64 platform with `gcc-4.4.4` and approximately $4\mu s$ on both BG/P with `xlf-11.1` and BG/Q with `xlf-14.1`. The 2 orders of magnitude performance difference and greater number of MPI ranks explains why this workload was much slower on Blue Gene platforms. Note that this issue was never seen in RTFlame simulations because, here, particles are distributed regularly throughout the computational domain.

5 Performance

A crucial first step is to determine the most efficient way to run our applications of interest on the BG/Q architecture. This is quite a large search space because there are many variables such as the number of MPI ranks and OpenMP threads per node and the chosen FLASH multithreaded strategy. Figure 10 shows how the number of MPI ranks per node and OpenMP threads per MPI rank affect the time to solution in a fixed RTFlame problem on 128 nodes of Vesta BG/Q. The chosen RTFlame test problem makes use of AMR and provides an effective resolution of 256^3 grid points. All data points are from runs using the thread within block multithreaded strategy because this was actually faster (evidence for this will be shown later). The figure shows that it is faster to run FLASH applications on BG/Q in hybrid MPI+OpenMP mode than it is

in MPI-only mode. There are two data points of particular interest in this figure at 32 MPI ranks per node with 2 OpenMP threads per MPI rank and 16 MPI ranks per node with 4 OpenMP threads per MPI rank. The 32 MPI ranks per node with 2 OpenMP threads per MPI rank is the FLASH configuration which gives the fastest time to solution, however, this is not an ideal configuration because it is tricky to fit the FLASH early science applications in 512 MB per MPI rank. The 16 MPI ranks per node with 4 OpenMP threads per MPI rank gives the best compromise between time to solution and memory usage. This configuration is being used for FLASH early science production runs and is used for all performance experiments in the remainder of this document.

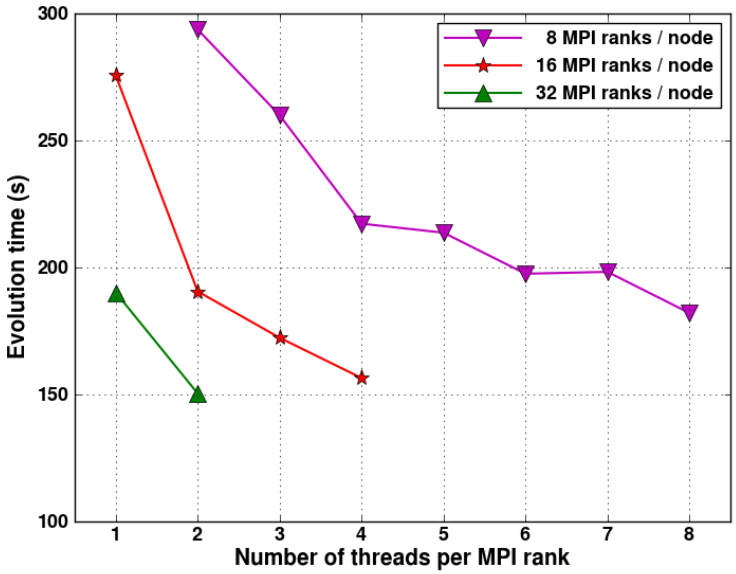


Figure 10: Time to complete 30 steps of a fixed RTFlame test problem on 128 nodes of Vesta BG/Q. The simulations were run with various numbers of MPI ranks and OpenMP threads and used the thread within block multithreading strategy.

The biggest memory consumer in these applications is the new unsplit hydrodynamics solver which requires approximately 2 - 2.5 times the memory of the old split hydrodynamics solver. This is not an inefficient implementation, rather it is the need to save information for all directions. There would be no issue with fitting in 32 MPI ranks per node if using the less accurate split hydrodynamics solver. Moving to 16 MPI ranks per node mode provides 1 GB per MPI rank and makes running FLASH much more comfortable. Many data buffers can be sized larger to accommodate rapid adaptive mesh refinement in regions of the domain and also congregation of many tracer particles on some MPI ranks.

Figure 11 shows how the FLASH multithreading strategy affects the strong scaling of the RTFlame test problem used in Figure 10. The problem initially has 21,462 blocks and 18,786 leaf blocks and is run on 512, 1024, 2048 and 4096 MPI ranks giving approximately 37, 18, 9 and 5 leaf blocks per MPI rank respectively. We do not provide a 8192 MPI rank data point because the load balancing rules in Paramesh

cause 0 leaf blocks, and therefore zero work, to be placed on some of these MPI ranks even though the average is approximately 2 leaf blocks per MPI rank. The figure shows that there is speedup with number of MPI ranks for both threading strategies. The speedup is not ideal and a contributing factor to this is that the grid management calls to exchange guard cells and correct fluxes are serialized. There is better speedup for the finer-grained thread within block strategy. All performance experiments in the remainder of this document use the thread within block strategy.

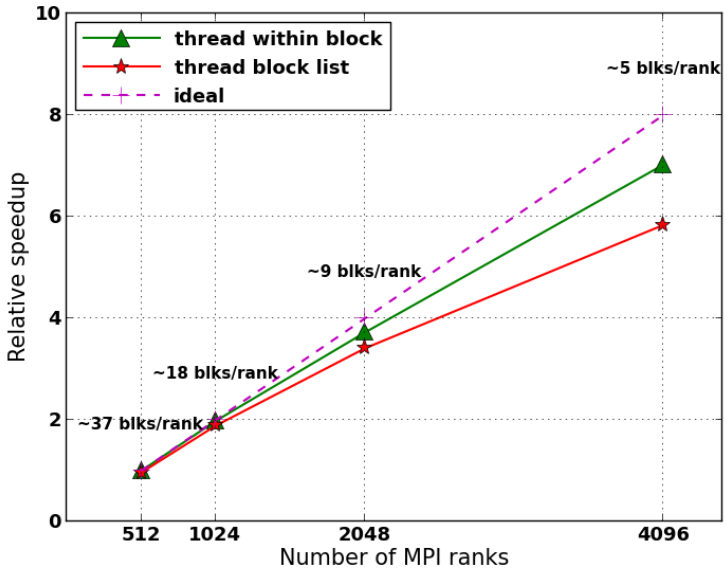


Figure 11: Relative speedup of a fixed RTFlame test problem on Vesta BG/Q. The simulations were run on different numbers of nodes with both FLASH multithreaded strategies but were always run with 16 MPI ranks per node and 4 OpenMP threads per MPI rank.

Figure 12 shows the strong scaling in RTFlame test problems of 256^3, 512^3, 1024^3 and 2048^3 effective resolution. The high resolution runs are of particular interest because previous production campaigns on BG/P studied configurations having 256^3 and 512^3 effective resolution. The largest FLASH run in this figure uses 32,768 nodes and 524,288 MPI ranks with 4 OpenMP threads per MPI rank. The strong scaling is generally good and fits with expectation: performance gets worse when there is less work per MPI rank and also when the resolution increases. The circled data points are from runs with the environment variable settings `PAMI_ALLREDUCE_REUSE_STORAGE=N`, `PAMI_ALLTOALL_PREMALLOC=N`, `PAMI_ALLTOALLV_PREMALLOC=N` and `PAMI_ALLTOALLW_PREMALLOC=N` to reduce the memory overhead. These settings may affect the performance but were necessary to make FLASH run despite an apparent leak of approximately 250 MB from the messaging layer (see Section 3.5). Figure 13 shows weak scaling of the same data. For reference, our early science production runs are using approximately 18 leaf blocks per MPI rank.

The performance advantage of running early science applications on BG/Q compared to BG/P can be summarized by a node-to-node ratio which is given by the expression $(T_p/T_q) \times (N_p/N_q)$, where T is run-

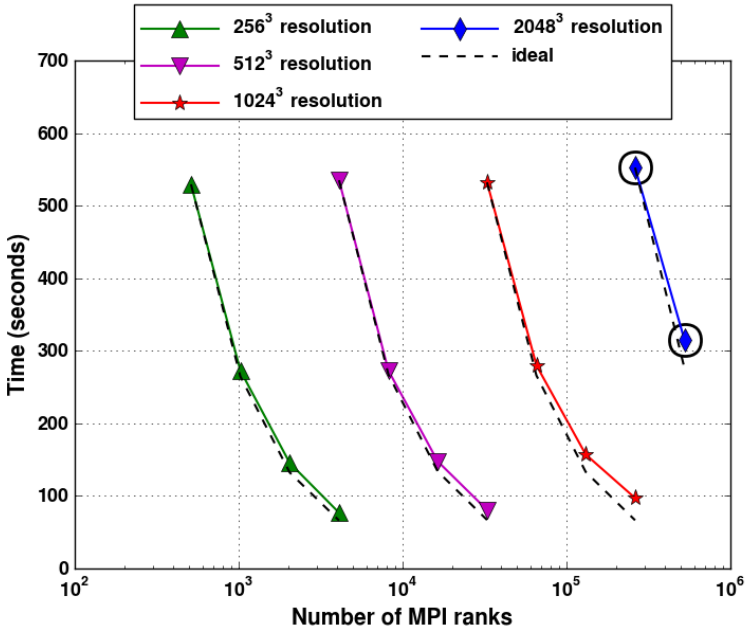


Figure 12: Strong scaling of RTFlame on Mira BG/Q for problems having 256^3, 512^3, 1024^3 and 2048^3 effective resolution. The simulations were run with 16 MPI ranks per node, 4 OpenMP threads per MPI rank and used the thread within block multithreading strategy.

time, N is the number of nodes, and q and p are subscripts indicating BG/Q and BG/P respectively. We run the fully-optimized early science applications in their most efficient configuration on both platforms, which is MPI-only in Virtual Node mode on BG/P and Hybrid MPI+OpenMP on BG/Q, and record FLASH evolution time as the application run-time. We obtain performance advantages of 8.9x (RTFlame) and 7.9x (DDT) which compares well with the ALCF target of 8x to 10x. Note that the FLASH performance advantage is actually slightly higher than shown because it is unrealistic to use the most efficient FLASH configuration on BG/P (i.e. Virtual Node mode) due to the memory footprint of the early science simulations.

Finally, we show HPCT performance counter information from an RTFlame test problem on 128 nodes of Vesta BG/Q in Figure 14. The FLASH application includes all optimizations up to "qhot(6)" (see Section 4.1) and counts were collected during FLASH evolution only. The performance data shows that the integer/load-/store/branch instructions dominate over floating point instructions (FXU=74.05% vs FPU=25.95%). The average weighted GFLOPS per node is 5.5 GFLOPS out of a possible 204.8 GFLOPS which means that this FLASH application is achieving 2.7% of peak floating point performance. This is quite low, however, there are many properties of FLASH applications which contribute to such a low fraction of peak performance: AMR introduces long-range communication and lots of control and integer code to focus floating point computation where it is needed, table lookups in EOS avoid heavy floating point calculations to improve time to solution, and tracer particles add to the total communication without increasing floating point work. The instructions completed per cycle per core is good (0.545) and there are a high percentage of hits in the L1 data cache (92.86% on node 0). Overall this FLASH application is using the BG/Q cores relatively efficiently.

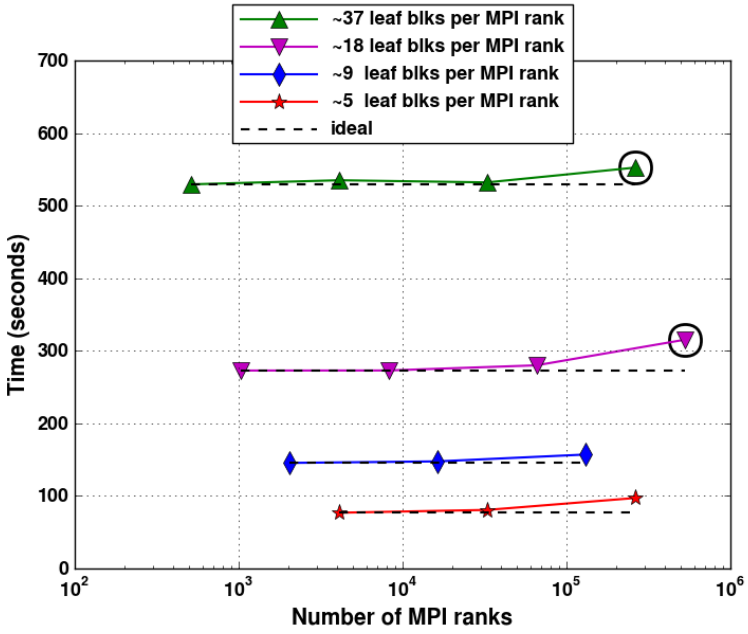


Figure 13: Weak scaling of RTFlame on Mira BG/Q. The number of leaf blocks per MPI rank is kept approximately constant in problems having 256^3, 512^3, 1024^3 and 2048^3 effective resolution. The simulations were run with 16 MPI ranks per node, 4 OpenMP threads per MPI rank and used the thread within block multithreading strategy.

Acknowledgments

Thanks to Vitali Morozov for all his help throughout the project. He ported the FLASH RTFlame simulation to BG/Q, provided support for the HPCT libraries, and helped with debugging and performance optimization of FLASH. Thanks to Dean Townsley for providing FLASH RTFlame and DDT simulations and for answering any questions about these simulations. Thanks also to George Jordan for explaining Flash Center science objectives and Anshu Dubey for advising about the layout of this report. The software used in this work was in part developed by the DOE NNSA-ASC OASCR Flash Center at the University of Chicago. This research used resources of the Argonne Leadership Computing Facility at Argonne National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under contract DE-AC02-06CH11357.

```
FLASH_evolution, call count = 1, avg cycles = 364213522553, max cycles = 364213557651 :
  -- Counter values for processes in this reporting group ----
    min-value  min-rank    max-value  max-rank    avg-value   label
  3.415637e+09      209   4.935353e+09     1822   4.462556e+09  Committed Load Misses
  5.638317e+10      209   6.779761e+10     1601   6.160317e+10  Committed Cacheable Loads
  2.744730e+09      209   4.035232e+09     1822   3.599715e+09  L1p miss
  1.409032e+11      209   1.583207e+11     1712   1.469828e+11  All XU Instruction Completions
  3.923771e+10      209   6.047476e+10     1822   5.151986e+10  All AXU Instruction Completions
  5.949847e+10      209   9.316478e+10     1822   7.853280e+10  FP Operations Group 1

Histogram of floating-point operation counts:
    flop-bin       #ranks
  5.949847e+10          1
  6.190321e+10          0
  6.430794e+10          0
  6.671268e+10         74
  6.911742e+10         29
  7.152215e+10         13
  7.392689e+10          5
  7.633163e+10       1046
  7.873636e+10        240
  8.114110e+10        116
  8.354584e+10        354
  8.595057e+10         85
  8.835531e+10         56
  9.076005e+10         13
  9.316478e+10         16

Derived metrics for code block "FLASH_evolution" averaged over process(es) in the reporting group
Instruction mix:  FPU = 25.95 %,  FXU = 74.05 %
Instructions per cycle completed per core = 0.5450
Per cent of max issue rate per core = 40.36 %
Total weighted GFlops = 706.553228
```

(a) Summary counts over all nodes

```
FLASH_evolution, call count = 1, avg cycles = 364210026210, max cycles = 364213555879 :
  -- Counter values summed over processes on this node -----
0       71127700508   Committed Load Misses
0      996543943304   Committed Cacheable Loads
0       57759693733   L1p miss
0     2370956187483   All XU Instruction Completions
0      827216441627   All AXU Instruction Completions
0     1261426933713   FP Operations Group 1
  -- L2 counters (shared for the node) ------------------
100    569692088884  L2 Hits
100      7121203216  L2 Misses
100      8169110557  L2 lines loaded from main memory
100      6253069690  L2 lines stored to   main memory


Derived metrics for code block "FLASH_evolution" averaged over process(es) on node <0,0,0,0,0>:
Instruction mix:  FPU = 25.87 %,  FXU = 74.13 %
Instructions per cycle completed per core = 0.5488
Per cent of max issue rate per core = 40.69 %
Total weighted GFlops for this node = 5.541
Loads that hit in L1 d-cache =  92.86 %
                 L1P buffer =   1.34 %
                 L2 cache   =   5.08 %
                 DDR        =   0.71 %
DDR traffic for the node: ld = 2.871, st = 2.198, total = 5.069 (Bytes/cycle)
```

(b) Counts on node 0

Figure 14: Performance counter data from an RTFlame test problem on 128 nodes of Vesta BG/Q. The simulation was run with 16 MPI ranks per node, 4 OpenMP threads per MPI rank and used the thread within block multithreaded strategy. Counts were only collected during FLASH evolution.

References

[1] P. MacNeice, K. M. Olson, C. Mobarry, R. de Fainchtein, and C. Packer. PARAMESH: A parallel adaptive mesh refinement community toolkit. *Computer Physics Communications*, 126:330–354, April 2000.

[2] OpenMP Architecture Review Board. OpenMP application program interface version 3.1. `http://www.openmp.org/mp-documents/OpenMP3.1.pdf`, July 2011.

[3] IBM. *IBM XL Fortran for Blue Gene/Q, V14.1. Optimization and Programming Guide*. IBM, 2012.

[4] Statistics for memory allocation with malloc. `http://www.gnu.org/software/libc/manual/html_node/Statistics-of-Malloc.html`, February 2013.

[5] V. Morozov. Blue Gene/Q Tuning Early Experience. `http://www.alcf.anl.gov/sites/www.alcf.anl.gov/files/morozov-bgqtuning-early.pdf`, March 2012. Slides presented at the ESP March Workshop "Code for Q", Argonne National Laboratory.

[6] The GNU C library. Resource usage. `http://www.gnu.org/software/libc/manual/html_node/Resource-Usage.html`, February 2013.

[7] A. Dubey, A.C. Calder, C. Daley, R.T. Fisher, C. Graziani, G.C. Jordan, D.Q. Lamb, L.B. Reid, D.M. Townsley, and K. Weide. Pragmatic optimizations for better scientific utilization of large supercomputers. *International Journal of High Performance Computing Applications*, to appear. Published online 21 November 2012 `http://hpc.sagepub.com/content/early/2012/11/20/1094342012464404`.

[8] Determining memory use. `https://www.alcf.anl.gov/resource-guides/determining-memory-use`, February 2013.

[9] Allocation debugging. How to install the tracing functionality. `http://www.gnu.org/software/libc/manual/html_node/Tracing-malloc.html#Tracing-malloc`, February 2013.

[10] L. Killough. Optimizing Single-Node Performance on BlueGene. `http://press.mcs.anl.gov/gswjanuary12/files/2012/01/Optimizing-Single-Node-Performance-on-BlueGene.pdf`, January 2012. Slides presented at the ALCF Winter Workshop, Argonne National Laboratory.

[11] The Flash Center for Computational Science at the University of Chicago. *FLASH Users Guide. Version 4.0.*, September 2012.

[12] R. Latham, C. Daley, W.K. Liao, K. Gao, R. Ross, A. Dubey, and A. Choudhary. A case study for scientific I/O: improving the FLASH astrophysics code. *Computational Science and Discovery*, 5(1):015001, 2012.

13 Lattice Quantum Chromodynamics

PI: Robert Mackenzie *(Fermilab)*, for the USQCD consortium.

Project Summary

Lattice quantum chromodynamics (LQCD) calculations are required to relate the experimentally observed properties of the strongly interacting particles to QCD, the fundamental theory of quarks and gluons. This research aims to produce the high-precision lattice QCD calculations that are urgently needed in the analysis of crucial experiments in high energy and nuclear physics that have recently been completed or are in progress. The broad aims of the calculations are to determine some of the basic parameters of the standard model of sub-atomic physics; to compute the masses, decay properties, and internal structure of strongly interacting particles; to obtain a quantitative understanding of the behavior of strongly interacting matter under extreme conditions of temperature and density; and to begin the study of strongly interacting theories that may be necessary to describe nature at the shortest distances.

Researchers will use the next-generation Blue Gene to generate gauge configurations that are representative samples of the systems being studied. These configurations will immediately be made available to all members of the U.S. Lattice Quantum Chromodynamics collaboration (USQCD), who will use them to perform a wide range of calculations. Members of USQCD are currently generating gauge configurations with three different formulations of lattice quarks, anisotropic clover, domain wall (DWF), and highly improved staggered quarks (HISQ), each of which has important advantages for different aspects of our work. Researchers expect to be using these formulations at the time of the Early Science Program. The next-generation Blue Gene will enable them to generate configurations that would support calculations of the spectrum, decay properties and internal structure of strongly interacting particles, and tests of the standard model, of unprecedented precision.

Report originally released as ANL/ALCF-ESP-13/11

SPI, mapping, site ordering, and QPX in Lattice QCD code on Mira

Heechang Na and James Osborn
Argonne Leadership Computing Facility, ANL

April 10, 2013

1 Introduction

Lattice QCD is a numerical method to simulate QCD (Quantum Chromodynamics) including non-perturbative effects. Among other methods, lattice QCD is the only successful non-perturbative method that can be systematically improved from first principles. Lattice QCD plays an important role in High Energy Particle Physics (flavor physics, spectroscopy, and beyond the Standard Model physics) and Nuclear Physics (nucleon/nuclear spectrum & structure, quark-gluon plasma, and the QCD equation of state). Moreover, recently there are active developments in applying lattice gauge theory to quantum field theory in general, for example the theoretical description of graphene or cold atom systems.

We define the QCD action on a finite and discrete 'lattice' like grid, so that we can compute the path integral (partition function) on a computer. A current lattice size is $96^3 \times 192$, and the corresponding number of degrees of freedom for the path integral is about 6×10^8. Thus, it requires enormous computing resources. In this report, we describe our overall efforts to improve lattice QCD software on Mira, which include a new communication library using the Message Unit (MU) SPI (System Programming Interface), better node mapping strategy, lattice site reordering, and using the quad floating point (QPX) intrinsics.

2 Message Unit (MU) SPI

We apply a Hybrid Monte Carlo method to compute the QCD path integral. At every step in the Monte Carlo evolution, we must solve a large sparse linear system. The application of the sparse matrix requires communications at the boundary to the nearest neighbors, and all nodes need to communicate at the same time with the same manner. Because the problem size per node is relatively small, the latency is more important than it typically

is for other applications. We found that building a new communication library using the MU SPI would improve the communication time.

The MU SPI [1] is the lowest communication layer; it is lower than MPI or PAMI. It allows one to control the MU hardware directly. The MU SPI provides point-to-point and collective communications with memory fifo, direct put, and remote get methods for each. For our 'qspi' communication library, we utilize the point-to-point communications with the direct put.

A node on Mira has 16+1 groups with 4 subgroups per group. Since the seventeenth group is not used by a normal user, there are 64 useable subgroups per node. Each subgroup has 8 injection FIFOs and 8 reception FIFOs. In other words, there are 32 injection / reception FIFOs per group or 512 injection / reception FIFOs per node.

Mira has a 5D torus network, so that each node can access 10 optical cables with 1.8 GB/sec useable bandwidth. Therefore, 10 FIFOs per rank would be optimal in order to use all cables simultaneously. However, for c64 mode, one rank can have a maxim of 8 FIFOs. Furthermore, if one needs to use MPI with the MU SPI, one should reserve at least 1 FIFO for MPI. In this case, one should consider about the optimal running mode and optimal distribution of the resources.

The qspi communication library API consists of:

- `void qspi_init(void);`

  Allocate FIFOs and base address tables.

- `void qspi_set_send(int dest, void *buf, size_t size, qspi_msg_t send_msg);`

- `void qspi_set_recv(int src, void *buf, size_t size, qspi_msg_t recv_msg);`

  `qspi_set_send` and `qspi_set_recv` prepare the handle variables declared as `qspi_msg_t` type. `dest` and `src` are rank of the destination and source node, `*buf` is the address of the send or receive buffer, and `size` is the size of the messages.

- `void qspi_prepare(qspi_msg_t msgs[], int num);`

  Exchange the handles between senders and receivers using MPI, and sets the descriptors.

- `void qspi_start(qspi_msg_t msg);`

  Inject the descriptors.

- `void qspi_wait(qspi_msg_t msg);`

The process will wait until the receiver counter reaches zero.

- `void qspi_finalize(void);`

As one might notice, `MPI_Init()` should be called after `qspi_init()` for `qspi_prepare`. One important communication pattern in lattice QCD is repeatedly sending and receiving with the same data structure. Therefore, once the communication is prepared, we can use it over and over again. Using MPI for `qspi_prepare` does not affect the overall communication time. We also note that there is no global barrier in qspi, so one would use `MPI_Barrier`.

We measure the communication time with qspi and MPI with several different settings. First, we had a ping-pong test between a set of two nearest neighbors in c1 mode. The results are shown in Fig. 1. As one can see, there is a good speedup for the latency: about 0.6 micro-seconds latency for qspi while MPI gives about 3 micro-seconds latency. As the data size increases the communication time is saturated to the bandwidth limit, so there is almost no speedup for message sizes larger than around 100 KB.

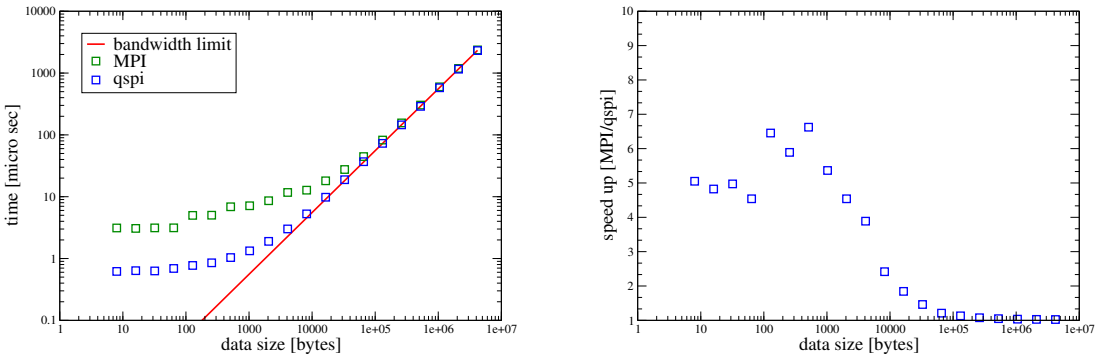


Figure 1: Single ping-pong benchmarks between qspi and MPI with c1 mode. The communication time vs. data size (left figure) and speedup comparing to MPI (right figure) are shown. The red line on the left figure represents the 1.8 GB/sec bandwidth limit.

Next, we tested with a more realistic situation that all ranks are sending and receiving messages to all 5 directions (5D halo exchange test) at the same time. The results are shown in Fig. 2. Latency is about 2.3 micro-seconds for qspi and 19 micro-seconds for MPI. The latency slows even for qspi, since there are much more communications needed, however the speedup comparing to MPI is much larger than for the single ping-pong tests. The speedup is around 9 to 20 times faster, while it was around 6 times faster for the single ping-pong tests.

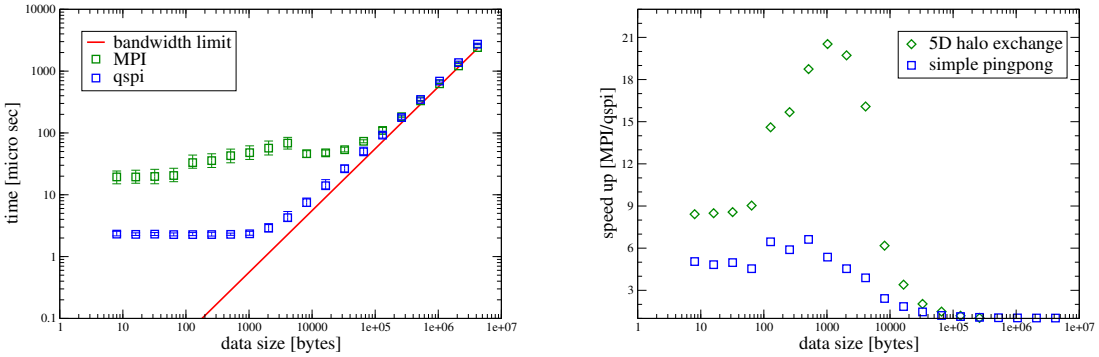


Figure 2: 5D halo exchange benchmarks between qspi and MPI with c1 mode. The communication time vs. data size (left figure) and speedup comparing to MPI (right figure) are shown. The red line on the left figure represents the 1.8 GB/sec bandwidth limit.

We ran the 5D halo exchange tests with c64 mode as well, and the results are shown in Fig. 3. In this case, one interesting question would be what an optimal number of FIFOs is. For the tests in Fig. 3, we assigned 4 FIFOs for each qspi process. We tested with 2 or 6 FIFOs as well, but we did not find any significant difference between the tests. This is because that the competition to get on the optical cable is higher than the competition to get a free FIFO. In c64 mode, there are 64 processes in a node trying to access 10 optical cables.

3 Mapping

Reducing the surface volume at the boundaries and the number of communication hops for each message are desirable, and one can get improvement with an optimal node mapping strategy. An example is shown in Fig. 4. In the figure, each orange dot represents a rank, and each blue box corresponds to a node. In the example, we assumed that we have 4 ranks per node. Lattice sites are allocated to each ranks, so that each rank needs to communicate to the nearest neighbors. As one can see, it has a smaller surface volume and smaller number of external hops with the mapping shown in the right figure of Fig. 4.

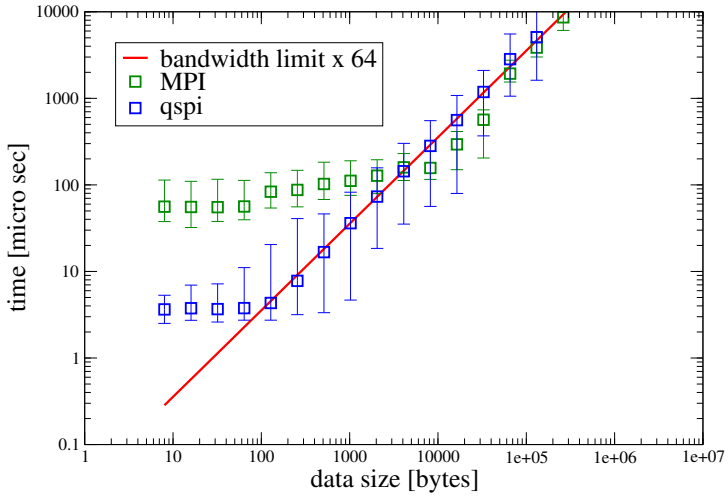


Figure 3: 5D halo exchange benchmarks between qspi and MPI with c64 mode. The communication time vs. data size (left figure) and speedup comparing to MPI (right figure) are shown. The red line represents the 1.8 GB/sec bandwidth limit divided over the 64 ranks per node.

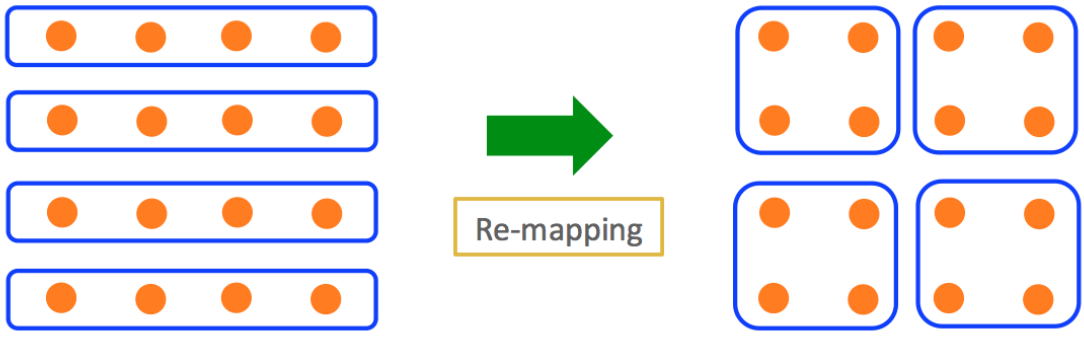


Figure 4: An example of a node mapping strategy in a 2D array with 4 ranks per node.

4 Site ordering

Lattice QCD calculations are typically based on a 4 dimensional lattice, and proper site ordering within a node can reduce the number of memory fragmentations in communications. An example of a 2D lattice site ordering strategy is shown in Fig. 5. The numbers in the figure indicate the order of data in memory. Normal site ordering is shown on the left figure. As one can see, the chunk of data at the boundary (inside of the blue rectangles) to be sent to left or right are not contiguous. However, with a better site ordering as on the right figure in Fig. 5, one can make all messages contiguous except the message to be sent to left.

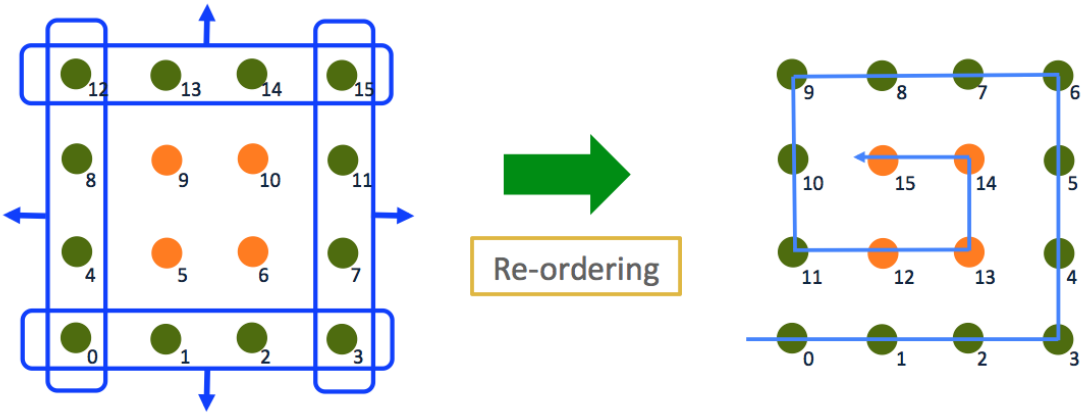


Figure 5: An example of site ordering strategy on a 2D lattice.

5 QPX in QLA

We added a few QPX routines using XLC intrinsics in QLA. QLA is a linear algebra library in the USQCD SciDAC modules [2]. One of most important calculations in lattice QCD is to multiply an array of 3×3 complex matrixes by an array of pointers to length 3 vectors, for example with an array size of N:

```
for i=0 to N-1
  r[i] += M[i] * (*v[i])
```

where `r[i]` and `*v[i]` are the i-th vectors, and `M[i]` is the i-th 3×3 matrix. We combine two matrix-vector multiplications in order to preserve alignment of the arrays when loading them into QPX registers. Since the subelements are complex numbers, when the two subelements are coupled as in Fig. 6, each bunch will fit into a QPX register. This

requires only 9 QPX instructions per array element, which should be 4 times faster ideally than without QPX.

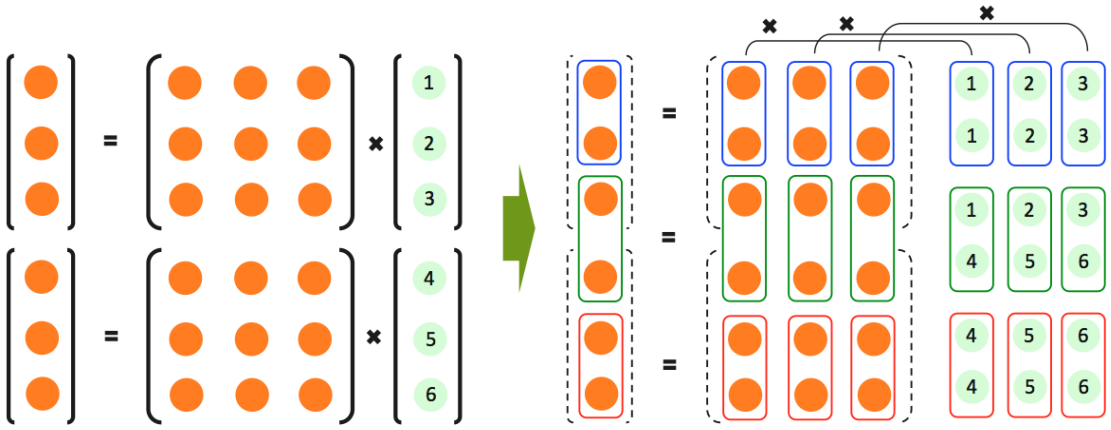


Figure 6: 3×3 matrix multiply by length 3 vector in QPX. All elements are complex numbers. In the right figure, each block with two complex numbers fits into a QPX register.

6 Overall performance and summary

We tested actual lattice QCD code with the improvements. We used the HISQ solver in the USQCD SciDAC modules [2] with a 12^4 lattice size per node on 128 BG/Q nodes. The HISQ solver is developed by the MILC collaboration [3]. The results are shown in Tab. 1. In these tests, we only use the site ordering with the qspi code. The first line of the table shows the result without any improvements. The next three lines represent results with only one improvement applied each. As one can see, all the improvements, QPX, qspi, and mapping, are equally improving the performance. One might notice that even we turned on the two of the improvements, the speedup is not so impressive (the next three lines). However, when we turn on all the three improvements with the site ordering, we obtain 2.3 times speedup as the last line of the table shows.

In this report, we presented several improvements in lattice QCD code on Mira and its impact. We developed a new communication library (qspi) using the MU SPI. With qspi, we got up to around 20 times faster latency than that with normal MPI in our 5D halo exchange tests. The optimal node mapping strategy helps to reduce the surface volume at the boundaries and the number of communication hops. In addition, we reduce the number of memory fragmentations in the communications by applying the optimal site ordering. Moreover, from using QPX in QLA, we gain significant speedup. With all the

Table 1: Overall benchmark results.

QPX	qspi	Mapping	mode	Gflops/node
			c32	11.7
		√	c64	12.9
	√		c32	13.8
√			c32	13.5
√	√		c32	17.6
√		√	c32	16.7
	√	√	c64	18.1
√	√	√	c32	26.7

improvements, we acquire 2.3 times speedup comparing to that without any improvements on BG/Q.

References

[1] Header files in /bgsys/drivers/ppcfloor/spi/include/mu and a html document /bgsys/drivers/ppcfloor/spi/doc/html/index.html in the BG/Q systems in ALCF.

[2] http://usqcd.jlab.org/usqcd-software/

[3] http://physics.indiana.edu/∼sg/milc.html

14 Petascale Direct Numerical Simulations of Turbulent Channel Flow

PI: Robert Moser *(University of Texas)*

Project Summary

Researchers propose to use the petascale computing power of the next-generation Blue Gene system to perform direct numerical simulations (DNS) of high Reynolds number turbulent wall-bounded flow in a channel. This DNS is aimed at developing a nearly complete understanding of the phenomena dominating wall-bounded turbulence, which is central to the energy losses inherent in transportation. The impact of such a development will likely be profound. Approximately 28% of U.S. energy consumption is expended on transportation. This energy expenditure is due to the interaction between solid surfaces (of vehicles or pipes) and the fluid flowing past them, leading to drag and the dissipation of energy by turbulence. Since much of the drag in these flows is due to turbulent skin friction, much of this energy consumption results from wall-bounded turbulent shear layers.

The central emphasis of this research is on reaching a sufficiently high Reynolds number to explore the physics that arise in the overlap region. The overlap region is where the viscous near-wall turbulence interacts with the outer-layer turbulences. This interaction is key to understanding high Reynolds number turbulent wall layers. To investigate this interaction, it is necessary that the Reynolds number be sufficiently high so that there is a substantial disparity in scale between the inner and outer layers. The results can then be extrapolated to arbitrary Reynolds numbers. This simulation will be performed using the supercomputing software that the proposing team has developed and benchmarked on Blue Gene/P and will further optimize for performance on the next-generation Blue Gene.

Report originally released as ANL/ALCF-ESP-13/12

Argonne Early Science Technical Report

MyoungKyu Lee, Nicholas Malaya, Robert D. Moser

Project: Petascale Direct Numerical Simulations of Turbulent Channel Flow
Catalyst: Ramesh Balakrishnan

1 Executive Summary

This project is focused on utilizing the Petascale computing power of Mira to perform direct numerical simulations (DNS) of high Reynolds number turbulent wall-bounded flow in a channel. This DNS is aimed at developing a nearly complete understanding of the phenomena dominating wall-bounded turbulence, which is central to the energy losses inherent in transportation. The impact of such a development will likely be profound. Approximately 28% of US energy consumption is expended on transportation. This energy expenditure is due to the interaction between solid surfaces (of vehicles or pipes) and the fluid flowing past them, leading to drag and the dissipation of energy by turbulence. Since much of the drag in these flows is due to turbulent skin friction, much of this energy consumption is caused by wall-bounded turbulent shear layers.

Engineering developments to reduce drag and energy consumption are greatly impeded by the lack of accurate models of the turbulence phenomena involved. DNS at the Reynolds numbers proposed here and the subsequent analysis of the resulting data can provide the insights needed to develop such models, as well as new concepts for manipulating wall-bounded turbulence.

The central emphasis of this research is on reaching a sufficiently high Reynolds number to explore the physics that arise in the overlap region. The overlap region is where the viscous near-wall turbulence interacts with the outer-layer turbulences. This interaction is key to understanding high Reynolds number turbulent wall layers. To investigate this interaction, it is necessary that the Reynolds number be sufficiently high so that there is a substantial disparity in scale between the inner and outer layers. The results can then be extrapolated to arbitrary Reynolds numbers. Analysis of recent DNS of channel flow at $Re_\tau = 1000$ and 2000 indicated that $Re_\tau \approx 4000$ on a $12288 \times 1024 \times 9216$ grid might yield sufficient scale separation. Due to the substantial power of Mira, as well as software performance increases attained during this Early Science Project, the channel will be run at $Re_\tau \approx 5000$ on a $15360 \times 1536 \times 11520$ mesh. Given the mesh size, this is the largest scientific DNS ever conducted.

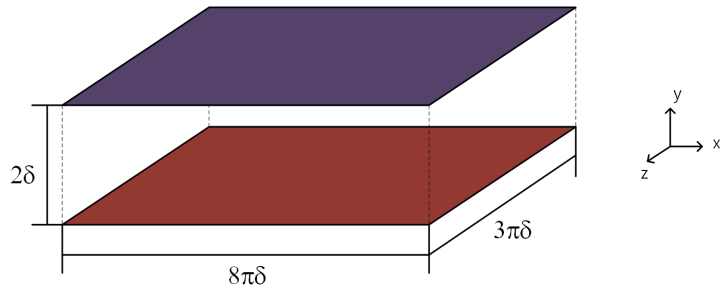


Figure 1: Diagram of the Channel DNS Geometry

2 Numerical Method

$$\frac{\partial u_i}{\partial t} = -\frac{\partial P}{\partial x_i} + H_i + \nu \frac{\partial^2 u_i}{\partial x_j \partial x_j} \tag{1}$$

The incompressible 3D Navier-Stokes equations (1) are solved using the formulation of Kim, Moin and Moser. Their technique involves integrating evolution equations for the wall-normal vorticity ω_y and the Laplacian of the vertical velocity $\nabla^2 v$. Periodic boundary conditions are imposed in the streamwise and spanwise directions, while in the wall normal direction, no slip conditions are imposed at the walls. Fig. 1 details the system geometry. A semi-implicit, third-order Runge–Kutta/Crank–Nicholson scheme is used for the time discretization (Spalart *et al* 1991) The flow is driven by a uniform pressure gradient which is adjusted continuously to maintain a constant mass flux. In space, a Fourier spectral representation is used in the streamwise and spanwise directions. A B-spline collocation representation is used in the wall-normal direction to accommodate the non-uniform grid requirements while providing spectral-like resolution (Kwok 2001). The B-spline breakpoints (y_i) are stretched across the interval [-1,1] per

$$y_i = \sin\left(\frac{\alpha\pi}{2}\frac{(-1+2i)}{n-1}\right) / \sin(\frac{\alpha\pi}{2}). \tag{2}$$

Here, α is a stretching parameter (set to 0.97 to provide an appropriate near-wall resolution), and n is the number of breakpoints. The collocation-based differential operators are formed using the Greville abscissae, also called the Marsden–Schoenberg points, implied by these breakpoints.

The code is written in Fortran2003 and uses hybrid MPI/OpenMP parallelism. Fourier transforms are performed one direction at a time, using FFTW3. In addition, we utilize the new FFTW3.3 MPI API to accomplish the distributed memory transposes necessary at each step. Hand-made libraries are used for linear algebra, and B-splines of arbitrary order are generated with the GNU Scientific Library which uses the recursive relationship of de Boor. This code has

been ported and run on various large-scale machines on various architectures (TACC's Lonestar, Blue Waters, NCSA's Kraken, ALCF's Intrepid and Mira), and scales well on each system.

3 Technical Results

Analysis of recent DNS of channel flow at $Re_\tau = 1000$ and 2000 indicated that $Re_\tau \approx 4000$ on a $12288 \times 1024 \times 9216$ grid might yield sufficient scale separation. Due to the substantial power of Mira, as well as software performance increases attained during this Early Science Project, the channel will be run at $Re_\tau \approx 5000$ on a $15360 \times 1536 \times 11520$ mesh.

3.1 Spin-up

To increase the rate at which the large scale run reaches a statistically stationary state, we performed "spin-up" runs to step our simulation through higher resolution grids, until all essential scales are resolved. These spin-up runs are summarized in Table 1.

Table 1: Parameters of channel DNS spin-up runs. N_x and N_z are numbers of Fourier modes, and N_y is number of collocation points. Δx^+ and Δz^+ resolutions are in terms of Fourier modes; Δy^+_{wall} and Δy^+_{CL} are the breakpoint spacings at the wall and centerline, respectively. T is the time span used to compute statistics. U_b is the bulk velocity, and L_x and h are the domain length in the streamwise direction and the channel half-height, respectively.

Name	N_x	N_z	N_y	Δx^+	Δz^+	Δy^+_{wall}	Δy^+_{CL}	TU_b/L_x
Coarsest	4096	3092	1024	30.68	15.24	0.104	14.91	≈ 0.88
Coarse	8192	4096	1536	15.34	11.50	0.068	9.94	≈ 0.53
Target	15360	11520	1536	8.18	4.09	0.068	9.94	≈ 10

3.2 Scaling

The code performs extremely well at scale. The communication benchmarks shown in Fig. 2 have been performed on as large as 32 racks. For our chosen problem size (8 racks), the parallel efficiency for a strong scaling problem is very nearly ideal, but even larger problem sizes have performed extremely well on Mira's interconnect.

Finally, benchmarks for a timestep of the entire codebase is shown in Fig. 3. The codebase demonstrates excellent strong scaling across a large portion of the entire machine (32 racks). A more detailed discussion of these results will be presented at MiraCon2013.

3.3 I/O

In order to attain portable, high throughput I/O with rich metadata support, we use HDF5 for our restart needs. We have found that I/O is such a common task in turbulence simuations, and therefore have developed a higher level API for common restart requirements. This library, entitled , ExaScale IO (ESIO),

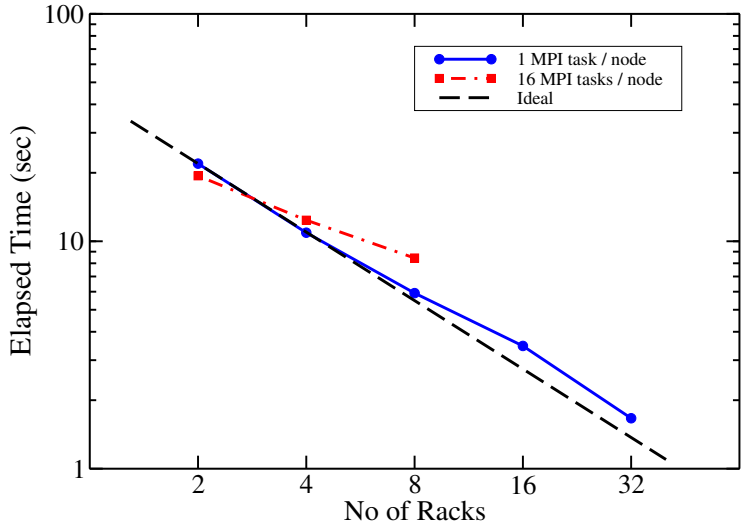


Figure 2: Strong scaling of the communication for the full problem size.

provides simple, high throughput input and output of structured data sets using parallel HDF5. ESIO is designed to support reading and writing turbulence simulation restart files but it may be useful in other contexts. The library is written in C99 and may be used by C89 or C++ applications. A Fortran API built atop the F2003 standard ISO_C_BINDING is also available.

For our DNS runs on Mira, ESIO is writing 1.8 TB in 444 seconds, at full scale (8 racks). This is a resonable write speed of approximately 4 GB/s, especially given the expected contention of at eight racks (1024 nodes$\times$16 tasks$*$ 8 racks $= 131,072$ writers) for the file system.

A restart must be written at least every 1000 steps, or about every three hours of simulation time. There will be at least of 250 restart files saved over the entire simulation. We allocated (as a safety factor) for the time required to read/write restart files to be approximately 5-10% of the total execution time, and in this case it is much less than this.

Nevertheless, we have observed far higher I/O write speeds in testing (peaks of 40 GB/s). As a result, we expect that significantly higher throughput can be attained, and expect to collaborate with ALCF Staff on this at MiraCon2013.

In addition, in late December and early January, during the porting of ESIO to Mira, we found several compiler compatibility issues between the ISO_C_BINDING standards and what was provided by IBM. Working with the ALCF staff identified a probable cause of this internal compiler error, namely that inappropriate fortran compiler scoping was at fault. In particular, an identically named (although different call pattern) function interface 'impl' using the iso_c bindings is producing the ICE. A workaround was demonstrated on a small piece of example code, by which renaming 'impl' to a unique string for

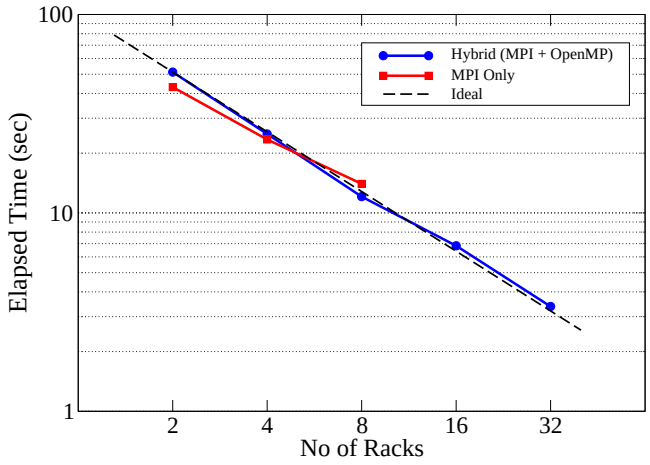


Figure 3: Strong scaling of a full timestep on Mira

every subfunction appeared to avoid the ICE. Thus, production runs were capable of utilizing this workaround, while simultaneously, on January 10th, 2013 PMR 57031,122,000 "XLF ICE on esio.F90" was submitted and confirmed by IBM.

3.4 Future Work

Future work will focus on automating job submission to maximize the time the application spends running. Additional work will focus on developing post-processing for the statistical quantities of interest. These can be developed in parallel with full production runs, and are not anticipated to delay the full project.

Estimating the convergence of statistics from the resulting simulation is critical to building confidence in the correctness of the codebase as well as the outputs of the simulation. However, turbulent velocity fields are correlated in time, so typical statistical sample estimates that rely on independent trials are biased. For this reason, we have developed methods that calculate the "effective sample size" of the statistics. This will be implemented in the codebase and provide an automated estimate of the remaining required runtime. The results of this work, accomplished using ALCF resources, will be the subject of an upcoming Physics of Fluids publication.

This project will transition to an INCITE2013 allocation in April. The process is expected to be continuous, with full scale production runs continuing

unabated.

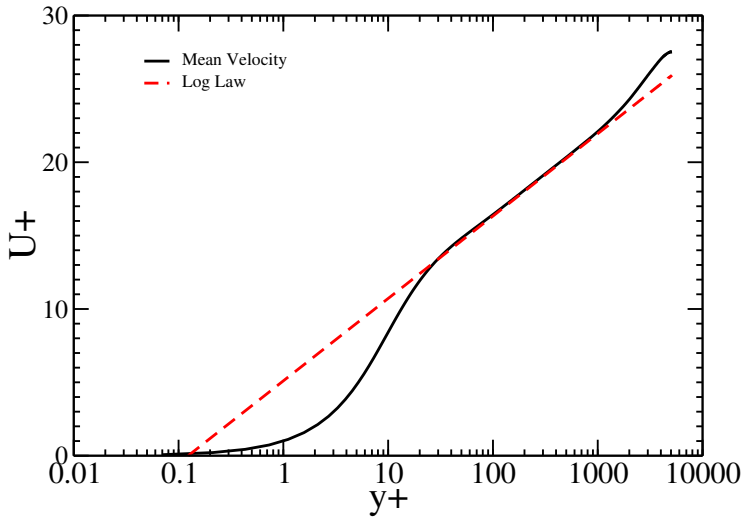


Figure 4: Mean velocity profile for $\mathrm{Re}_\tau \approx 5000$

4 Preliminary Scientific Results

Given the mesh size, this simulation is significantly larger than the 4096^3 homogeneous turbulence simulation (by a factor of ≈ 1.76). As a result, this simulation will be, to our knowledge, the highest resolution DNS ever conducted for scientific purposes.

However, the large scale simulation has not yet come to steady state. Therefore, the the simulation is not yet producing valuable statistical data. As a result, all observations about the field are extremely preliminary.

The mean velocity profile can be seen in Figure 4. This plot demonstrates that the velocity profile does appear to be demonstrating logarithmic scaling with distance from the wall, as anticipated by theory. This is an early verification that the field is behaving as expected.

Given the wall-normal resolution, this simulation will provide an unsurpassed estimate for the Von Karman constant. Fig. 5 demonstrates a profile that appears to be flattening in the logarithmic region. While preliminary, the value appears to be converging upon a value of $1/.41$. In statistically steady state, we have every reason to believe that this is the most precise calculation of the Von Karman constant ever conducted, either experimentally or via simulation.

4.1 Future Work

During the production runs, statistical estimators of various quantities of interest will be provided at `http://turbulence.ices.utexas.edu/`. This will already be of great value to the turbulence community. Furthermore, it must

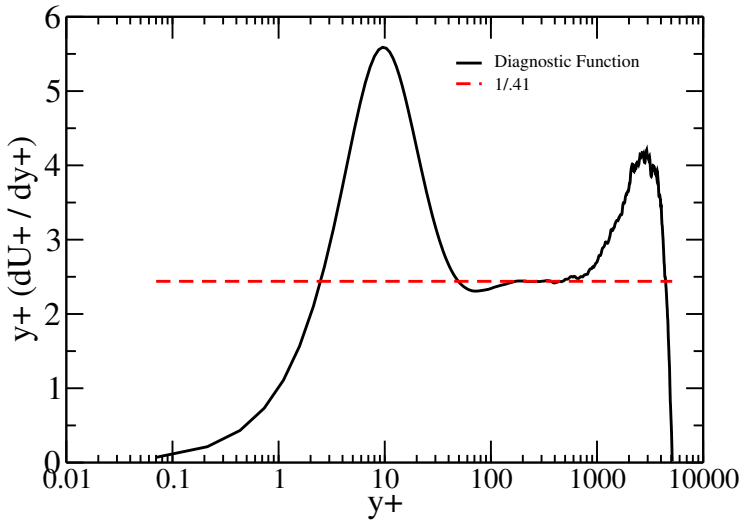


Figure 5: Karman constant estimate

be stressed that even after the completion of production runs, a great body of scientific work remains to be done. Once the simulations have been performed, the fields they generated can be used like an experimental facility to "measure" most any diagnostic quantity of interest. Scientific advances will come from repeated analysis of the simulation data to answer new questions and test hypotheses, by a number of researchers over a period of years. The most complete sequence of cases is for turbulent channels in the friction-Reynolds-number range $Re_\tau = 180 - 2000$, which now constitute the standard reference data set in the field and receive about 30-40 citations per year. As we supplementing these data with $Re_\tau = 5000$, we expect this will establish a reference data set that will remain useful for the turbulence research community for many years.

5 Work with ALCF Staff

During the course of this Early Science Project, we have worked with several members of the ALCF staff. Of particular note, we have dealt with Ramesh Balakrishnan extensively for both technical consulting on the codebase as well programmatic work. We also appreciate the help of Jeff Hammond for his assistance with the ICE Bug detailed in section 3.3, and Kevin Harms for work related to disk space quotas and I/O in general.

15 Ab-initio Reaction Calculations for Carbon-12

PI: Steven Pieper *(Argonne National Laboratory)*

Project Summary

Researchers will calculate several fundamental properties of the ^{12}C nucleus: the imaginary-time response, the one-body density matrix, and transition matrix elements between isospin- 0 and -1 states. These are needed to be able to reliably compute neutrino-^{12}C scattering, which is needed for neutrino detector calibrations; quasi-elastic electron scattering, which is currently being measured at Jefferson Lab (JLab); and the results of older reactions on ^{12}C.

In the past 15 years, researchers have developed Greens function Monte Carlo as a powerful and accurate method for computing properties of light nuclei using realistic two- and three-nucleon potentials. This will be the basis of all the calculations. Understanding the propagation of charges and currents in the nucleus is critical to a real understanding of the physics of nucleonic matter. Electron scattering experiments in the quasi- elastic regime, where the dominant process is knocking a single nucleon out of the nucleus, are under way at Jefferson Lab for a range of nuclei. The separation into longitudinal and transverse response allows one to study the propagation of charges and currents, respectively, in the nucleus. The nontrivial changes as one goes from the nucleon and deuteron to larger nuclei like carbon require one to consider processes well beyond simple, one-nucleon knockout. Researchers will compute the transition density matrices on a two-dimensional grid of the magnitudes of the initial and final positions. A partial wave expansion of the angle between the two vectors will be made.

Report originally released as ANL/ALCF-ESP-13/13

ESP technical report

Alessandro Lovato

*ALCF and Physics division, Argonne National Laboratory**

Steven C. Pieper

Physics division, Argonne National Laboratory

(Dated: April 10, 2013)

* lovato@alcf.anl.gov

I. DESCRIPTION OF SCIENCE

The electroweak response is a fundamental ingredient to describe the neutrino - ^{12}Carbon scattering, recently measured by the MiniBooNE collaboration to calibrate the detector aimed at studying neutrino oscillations. As a first step towards its calculation, we have computed the sum rules for the electromagnetic response of ^{12}C. The cross section of the process

$$e +^{12}\mathrm{C} \to e' + X\,. \tag{1}$$

can be written in Born approximation as [1]

$$\frac{d^2\sigma}{d\Omega_{e'}dE_{e'}} = -\frac{\alpha^2}{q^4}\frac{E_{e'}}{E_e}L_{\mu\nu}W^{\mu\nu}\,, \tag{2}$$

where $\alpha \simeq 1/137$ is the fine structure constant, $d\Omega_{e'}$ is the differential solid angle specified by $\mathbf{k}_{e'}$ and $q = k_e - k_{e'}$ is the four momentum transfer of the process. The leptonic tensor $L_{\mu\nu}$ is fully determined by the measured kinematical variables of the electron, while all information on target structure, which is largely dictated by nuclear interactions, is enclosed in the hadronic tensor

$$W^{\mu\nu} = \sum_X \langle\Psi_0|J^\mu|\Psi_X\rangle\langle\Psi_X|J^\nu|\Psi_0\rangle\delta^{(4)}(p_0 + q - p_X)\,. \tag{3}$$

The sum over the final states includes an integral over $\mathbf{p}_X$, the spatial momentum of the final hadronic state, while p_0 is the initial four-momentum of the nucleus.

In the nonrelativistic approach, the hadronic tensor can be written in terms of the longitudinal and transverse response functions, with respect to the direction of the three-momentum transfer $\mathbf{q}$. For instance, taking $\mathbf{q}$ along the z-axis, the transverse response is defined by [2]

$$R_{xx+yy}(\mathbf{q},\omega) = \sum_X \delta(\omega + E_0 - E_X)\Big[\langle\Psi_0|j^x(\mathbf{q},\omega)|\Psi_X\rangle\langle\Psi_X|j^x(\mathbf{q},\omega)|\Psi_0\rangle + \langle\Psi_0|j^y(\mathbf{q},\omega)|\Psi_X\rangle\langle\Psi_X|j^y(\mathbf{q},\omega)|\Psi_0\rangle\Big] \tag{4}$$

while the longitudinal is given by

$$R_{00}(\mathbf{q},\omega) = \sum_X \delta(\omega + E_0 - E_X)\langle\Psi_0|\rho(\mathbf{q},\omega)|\Psi_X\rangle\langle\Psi_X|\rho(\mathbf{q},\omega)|\Psi_0\rangle \tag{5}$$

The *sum rules* are obtained integrating the response functions over the energy transfer and using the completeness relation of the states $|X\rangle$. For R_{xx+yy} and R_{00} one has

$$\begin{aligned} S_{xx+yy}(\mathbf{q}) &\equiv \int d\omega R_{xx+yy}(\mathbf{q},\omega) = \langle\Psi_0|j^x(\mathbf{q},\omega_{el})j^x(\mathbf{q},\omega_{el}) + j^y(\mathbf{q},\omega_{el})j^y(\mathbf{q},\omega_{el})|\Psi_0\rangle \\ S_{00}(\mathbf{q}) &\equiv \int d\omega R_{00}(\mathbf{q},\omega) = \langle\Psi_0|\rho(\mathbf{q},\omega_{el})\rho(\mathbf{q},\omega_{el})|\Psi_0\rangle\,, \end{aligned} \tag{6}$$

where the energy transfer dependence of the current and density operators is determined at the the quasi-elastic peak: $\omega_{el} = \sqrt{|\mathbf{q}|^2 + m^2} - m$. Hence, the sum rules of the response can be evaluated by computing the expectation values of the electromagnetic currents and density on the ground state of ^{12}C.

II. NUMERICAL METHODS

The calculation of the sum rules requires the knowledge of the nuclear ground state wave-function of ^{12}C. Solving the many-body Schroedinger equation

$$\hat{H}\Psi_0(x_1 \ldots x_A) \ = E_0\Psi_0(x_1 \ldots x_A)\,, \tag{7}$$

where the generalized coordinate $x_i \equiv \{\mathbf{r}_i, s_i, t_i\}$ represents both the position and the spin-isospin variables of the i-th nucleon, is made particularly difficult by the complexity of the interaction. The nuclear potential is indeed spin-isospin dependent and contains strong tensor terms; thus Eq. (7) consists in $2^A\binom{A}{Z}$ complex coupled second order partial differential equations in $3A$ variables. For the actual case of ^{12}C, there are 270,336 coupled equations in 36 variables.

Standard methods for solving partial differential equations are not feasible in this context. Green Function Monte Carlo (GFMC) algorithms use projection techniques to enhance the true ground-state component of a starting trial wave function Ψ_T

$$\Psi_0(x_1 \ldots x_A) = \lim_{\tau\to\infty} e^{-(\hat{H}-E_0)\tau}\Psi_T(x_1 \ldots x_A)\,. \tag{8}$$

In the actual calculation, the imaginary time evaluation is done a sequence of imaginary time steps, each one consisting in a $3A$ dimensional integral, evaluated within the Monte Carlo approach.

In GFMC all the spin-isospin configurations are considered and the wave-function is a vector of $2^A\binom{A}{Z}$ complex numbers. For example the eight spin configurations of the ^{3}H nucleus are represented by [3]

$$|\Psi_{^3H}\rangle = \begin{pmatrix} a_{\uparrow\uparrow\uparrow} \\ a_{\uparrow\uparrow\downarrow} \\ a_{\uparrow\downarrow\uparrow} \\ a_{\uparrow\downarrow\downarrow} \\ a_{\downarrow\uparrow\uparrow} \\ a_{\downarrow\uparrow\downarrow} \\ a_{\downarrow\downarrow\uparrow} \\ a_{\downarrow\downarrow\downarrow} \end{pmatrix} \tag{9}$$

Each coefficient a_α, which is a function of the coordinates $\mathbf{r}_1$, $\mathbf{r}_2$ and $\mathbf{r}_3$, represents the amplitude of a given many-particle spin configuration; for instance

$$a_{\uparrow\uparrow\downarrow} = \langle\uparrow\uparrow\downarrow |\Psi_{^3H}\rangle \,. \tag{10}$$

The application of the spin matrix $\sigma_{12} \equiv \sum_i \sigma_1^i \sigma_2^i$ yields

$$\hat{\sigma}_{12}|\Psi_{^3H}\rangle = \begin{pmatrix} a_{\uparrow\uparrow\uparrow} \\ a_{\uparrow\uparrow\downarrow} \\ 2a_{\downarrow\uparrow\uparrow} - a_{\uparrow\downarrow\uparrow} \\ 2a_{\downarrow\uparrow\downarrow} - a_{\uparrow\downarrow\downarrow} \\ 2a_{\uparrow\downarrow\uparrow} - a_{\downarrow\uparrow\uparrow} \\ 2a_{\uparrow\downarrow\downarrow} - a_{\downarrow\uparrow\downarrow} \\ a_{\downarrow\downarrow\uparrow} \\ a_{\downarrow\downarrow\downarrow} \end{pmatrix} \tag{11}$$

The "new" wave function can be expressed in terms of the coefficients of the old one. Therefore, in order to reduce the computational complexity of the spin and isospin matrix multiplication, a specialized table-drive code is implemented.

III. BEFORE MIRA AND ON MIRA

The GFMC code needed to be deeply revised to better capitalize the resources of a leadership class computer like Intrepid (BQP) and Mira (BGQ).

The branching process of the GFMC algorithm involves replication and killing of the samples, the number of which can undergo large fluctuations. Therefore, to achieve an high

efficiency, the old version of the code did several Monte Carlo samples, say at least 10, per processor. However, a typical ^{12}C calculation involves around 15,000 samples while leadership class computers have many 10,000's of processors, making the old algorithm quite inefficient.

Fortunately, for nuclei as large as ^{10}B and ^{12}C, the calculation of the energy and of the response is complex enough to allow for splitting one sample over many processors. To this extent, the general purpose Automatic Dynamic Load Balancing (ADLB) library [4], was developed on Intrepid and implemented in the code.

Both the direct calculation of the response with the $\Psi_{1^+,T=1}$ state and the evaluation of the sum rules would have not been possible on Intrepid, due to the limited amount of RAM per node (2GB). On the other hand, Mira, with 16 GB of RAM per node enables us to perform such a large calculations.

We were pleased to find that the version of ADLB developed for Intrepid works very well on Mira; no modification was required. The conversion to Mira consisted primarily of timing the OpenMP (OMP) sections of the GFMC code to work well up to 64 threads and developing the new subroutines for the response and for the sum rules.

IV. THE CODE

The scheme of ADLB, illustrated in Fig. 1, shows that the nodes are organized in *servers* and *slaves*; in standard GFMC calculations approximately 3% of the nodes are ADLB servers. A shared work queue, managed by the servers, is accessed by the slaves that either *put* work units, denoted as "work packages" in it or *get* those work package out to work on them. Once a work package has been processed by a slave, a "response package" may be sent to the slave that put the work package in the queue.

ADLB is a general purpose library, which hides communication and memory management from the application, providing a simple programming interface. Besides the initialization and termination functions, the truly essential function calls of the ADLB application programmer interface (API) are the `ADLB_Put`, `ADLB_Reserve` and `ADLB_Get_reserved`. To better illustrate these three function calls, it is worth showing the explicit case of the sum rules subroutines.

The expectation value of Eq. (6) has to be evaluated for momentum transfer directed along x, y and z axis. In each of these cases, $\sim$ 20 values of the discretized momentum transfer magnitude are considered; hence for each configuration $\sim$ 60 independent expectation values

have to be computed. Since the evaluation of the sum rules of the ^{12}C for a single value of **q** takes of about 100 seconds (with 32 OMP threads), we decided to split the calculation in such a way that each ADLB slave calculates the sum rules for a single value of **q**.

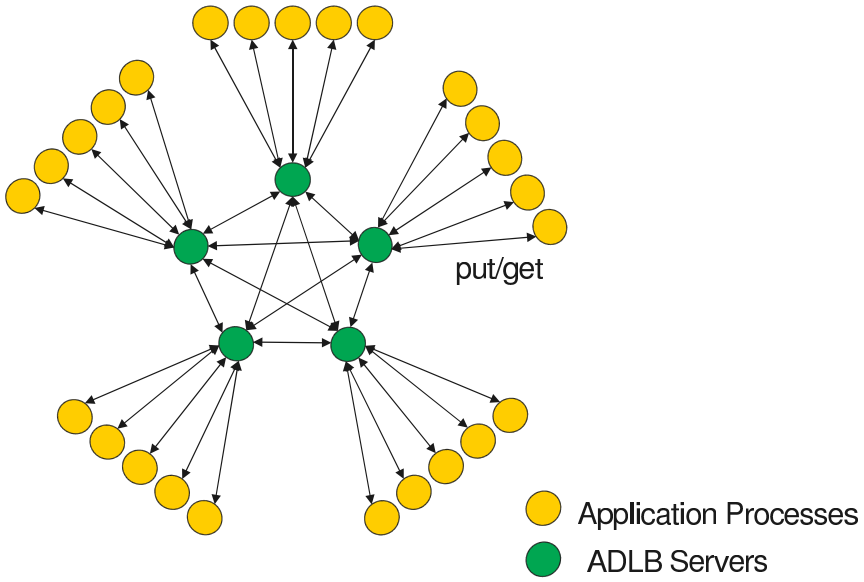


Figure 1. Automatic Dynamic Load Balancing work flow.

- `subroutine o_em_wk`

  Let us concentrate on a particular ADLB energy slave, managing a single configuration. It enters `o_em_wk` and immediately puts into the work pool the part of work package independent on **q**

  ```
  call ADLB_Begin_batch_put (rwp%cfl,respon_wp_len_common,ierr)
  ```

  where `rwp%cfl` indicates the beginning of the work package, `respon_wp_len_common` denotes its size and `ierr` will get a return code.

  Afterwards, the **q** dependent parts of the work packages are placed in the work pool for each of the ~ 60 cases.

  ```
  call ADLB_PUT(rwp%qh,respon_wp_len_var,-1,myid, adlbwp_respon,i_prior,ierr)
  ```

The size of the **q** dependent part of the work package is specified by `rwp%qh,respon_wp_len_var`, while `myid` identifies the energy slave from which the work package originates.

As a matter of fact, the work packages can be processed either by the same ADLB energy slave which put them in the work pool or by another one. However, only the slave that sent the work package can retrieve the corresponding response package by means of the following call

```
call get_adlb_respon_work_ans ( ierr, node )
```

which is iterated until all the response packages have been collected.

- `get_adlb_respon_work_ans`

  To retrieve a unit of work, the energy slave uses this subroutine to call `ADLB_reserve`

```
call ADLB_reserve ( (/ adlbwp_respon_ans, adlbwp_respon, -1 /),  &
  &   i_wrk_type, i_prior, i_handle, i_len, i_answer, ierr )
```

  specifying that it is looking either for a work package or for a response package. If either one is present, ADLB will find it and send back a handle (`i_handle`) , a global identifier (`i_wrk_type`) along with the size of the reserved work unit (`i_len`) and the origin identifier (`i_answer`). The priority of the answer is set much larger than the priority of the work packages, so that they will be preferentially returned.

  If an answer has been found by ADLB, it is retrieved by

```
call ADLB_GET_RESERVED_TIMED ( rap, i_handle, qtime, ierr )
```

  where `rap` denotes the response answer package and the energy slave returns into the subroutine `o_em_wk`.

  If a work package is instead found by `ADLB_reserve`, the energy slave processes it. It has to be remarked that other ADLB slaves than the energy ones can process a work package. For this purpose an `entry` appears in the subroutine

```
entry process_adlb_respon_work (ii_prior,ii_handle,ii_len,ii_answer,ierr)
```

Analogously to what happens for the answer package, `ADLB_GET_RESERVED_TIMED` is called

```
call ADLB_GET_RESERVED_TIMED ( rwp, i_handle, qtime, ierr )
```

with `rwp` appearing as a first argument instead of `rap`.

Using the retrieved work package, the actual calculation of the sum rule is performed for a single value of **q**

```
call o_em_wk_q(rwp%iptb,rwp%if2,rwp%actf,rwp%q,rwp%qh,rwp%weight,  &
      &  rwp%rpart0,rwp%cfl,cfdl,rwp%cfr,cfdr,rap%fxtt_q,rap%fxll_q, &
      &  iqq,iqh,.false.)
```

If the work package answer is addressed to a different ADLB slave from the one that made the computation, `myid .ne. i_answer`, then the answer needs to be put in the work pool

```
call ADLB_PUT ( rap, respon_ans_len, i_answer, myid, adlbwp_respon_ans,&
       &  i_prior+1000, ierr)
```

Otherwise, the answer package is not put in the pool and the energy slave returns in the subroutine `o_em_wk`.

- `master_get_work`

  The main program continuously calls the subroutine `master_get_work` to look for work packages. These can be of any type (except answers). The appropriate subroutine is called to process the work and then `master_get_work` is used on each slave, again.

V. TUNING THE CODE AND PERFORMANCE ON MIRA

The conversion of the GFMC code from Intrepid to Mira did not show particular difficulties. The ADLB performance turned out to be even better on Mira than on Intrepid without modifications. Moreover, OpenMP scales well with the number of threads.

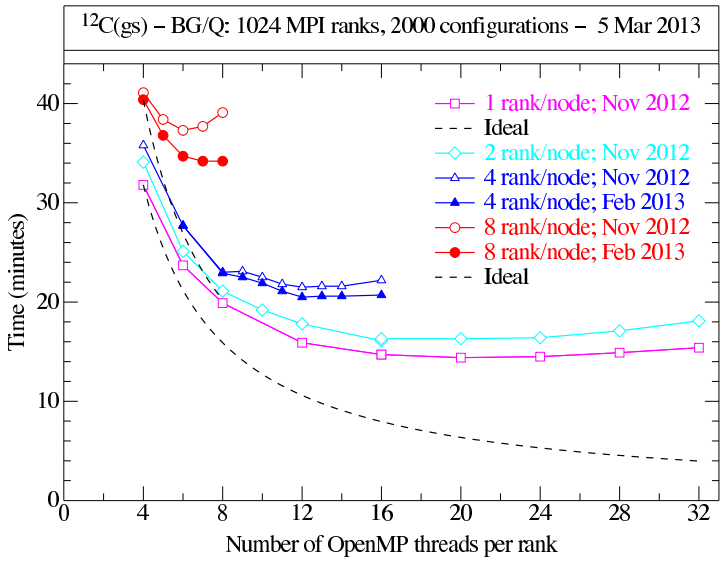


Figure 2. OMP strong scaling performance for ^{12}C GFMC calculation with 2048 configurations, 1024 MPI ranks: total wall time comparison.

A. OpenMP (OMP) strong scaling

First of all let us analyze the OMP scaling of the total wall time required to do a GFMC calculation for 2048 configurations of the ^{12}C ground state using 1024 MPI ranks, displayed in Figure 2. The different colours indicate the different number of ranks per node and hence the total number of nodes. As expected, the single rank per node case exhibits the best OMP scaling, as there are no threads associated with different ranks competing for memory on the same node. Due to the competition among the threads belonging to the same rank, the scaling saturates at about 20 threads, remaining fairly above the ideal case, represented by the dashed curve, where the wall time decreases as 1/(#OMP threads).

Since in every node there are at most 64 threads, keeping fixed the number of MPI ranks and increasing the number of threads results in a larger node usage. Thus, a more meaningful scaling test consists in studying the number of configurations processed by a single node with different combinations of ranks per node and threads per rank, keeping in mind that the product of these two quantities cannot exceed 64. The results of Fig. 3 show that, with the new driver installed in February 2013, the most efficient configuration is 8 ranks per node, 8 threads per rank. In this case a performance of 6.4 GFLOPS per node (about 3.1% of the

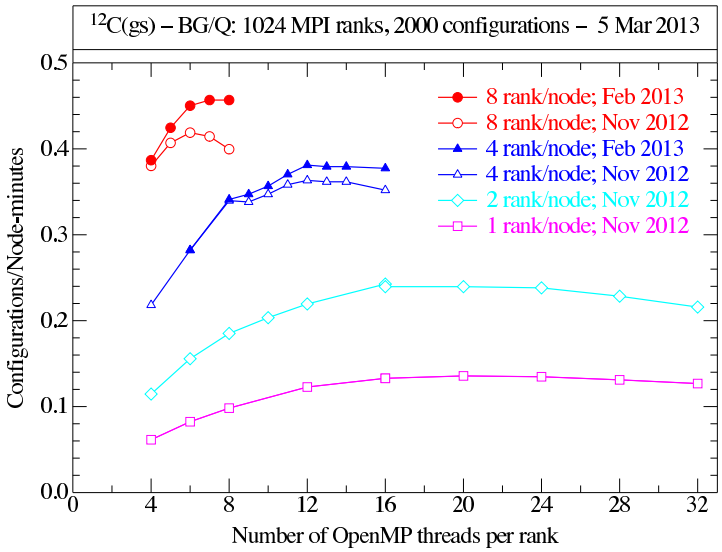


Figure 3. OMP strong scaling performance for ^{12}C GFMC calculation with 2048 configurations, 1024 MPI ranks: number of configurations per node per minute.

peak) is achieved. Is should be noticed that with the former version of the driver using more than 6 threads per rank resulted in worse performances. Finally, the limit of 8 ranks per node is dictated by memory requirements.

An analogous analysis, shown in Fig. 4 has been performed for the sum rules calculation with 32 MPI ranks. Due to the large size of the wave function derivative, not more than 1 rank per node can be used in the calculation; however if the derivatives are disregarded, 4 ranks per node can be used.

Because of large loops over the spin and isospin indices of the wave functions, OMP keeps improving up to 64 threads, although very slowly beyond 32 threads. However, as for the energy, while the minimum total wall time consumption is obtained with 1 rank per node and 64 threads per rank, the highest efficiency of about 12 GFLOPS per node is achieved with 4 ranks per node and 16 threads per rank.

B. ADLB weak scaling

The ADLB library was not significantly exercised in the results shown in the former section, as the number of MPI ranks was limited to 1024. By looking at Fig. 5, in which the total wall

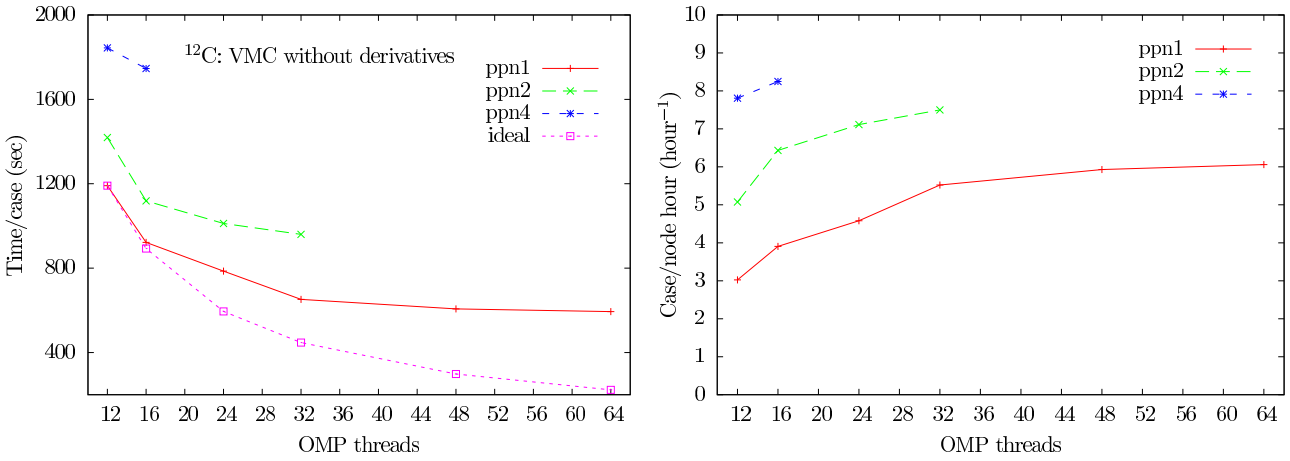


Figure 4. OMP strong scaling performance for sum rule calculation with 2 values of momentum transfer. The total wall time (left panel) and the case/node per hour (right panel) for 32 MPI ranks are shown.

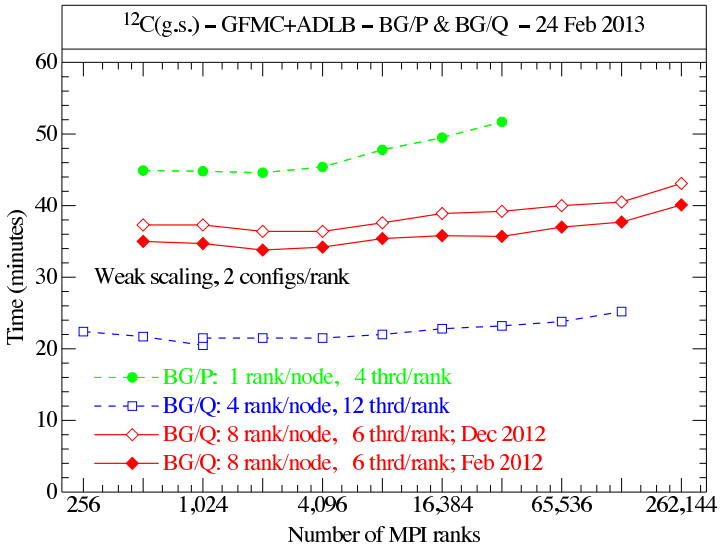


Figure 5. ADLB weak scaling performance for energy calculation with 2 configurations per rank: total walltime

time used for computing 2 configurations per MPI rank is plotted against the number of MPI ranks, it is possible to appreciate the improvements brought about by ADLB. A good scaling, fairly close to the ideal case of a straight and horizontal line, is shown up to 260,000 ranks, 524,688 cores, 1,572,864 threads. As for the OMP scaling, it is worth analyzing the scaling of

the configurations per node per minute, displayed in Fig. 5. Despite the smallest total wall time being consumed by using 4 ranks per node and 12 threads per rank, the most efficient configuration is the one with 8 ranks per node and 6 threads per rank.

Finally, it is interesting to notice that a Mira node is almost ten times faster than an Intrepid one.

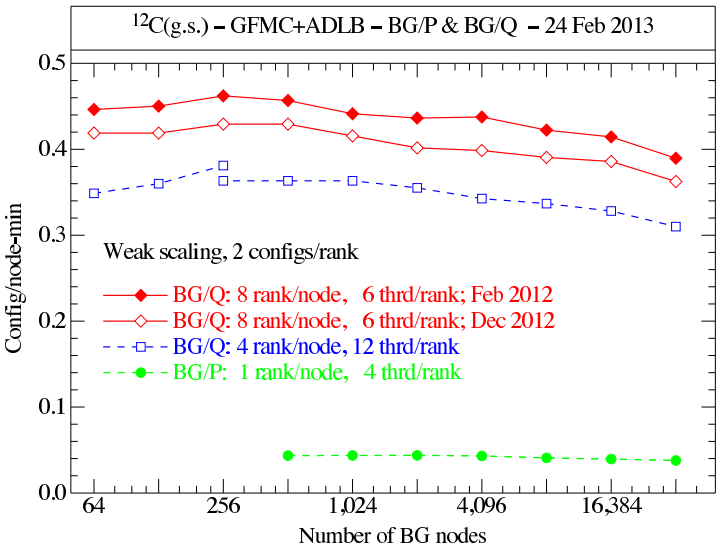


Figure 6. ADLB weak scaling performance for energy calculation with 2 configurations per rank: number of configurations per node per minute

VI. FIRST SUM RULE RESULTS

In Fig. 7 we show the preliminary results for the sum rule of the transverse electromagnetic response of ^{12}C, obtained neglecting the derivatives of the wave functions. It has been obtained by averaging over $\sim$ 1000 configuration for each of the 5 imaginary time values, $\tau = 0.8, 0.18, 0.28, 0.40, 0,48$, after the constrained path has been released.

The two-body currents have a prominent effect, relatively much larger than the difference between GFMC and VMC calculations, that at this level of accuracy, provide compatible results.

Many more configurations are needed for quantities defined in terms of differences between

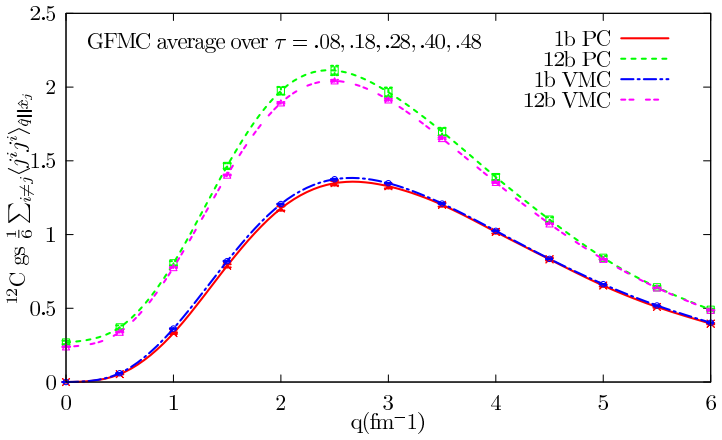


Figure 7. ^{12}C transverse sum rule for the electromagnetic transverse response.

large Monte Carlo estimates, such as the longitudinal Coulomb sum rule

$$S_L(\mathbf{q}) = \frac{1}{6[G_p^e(q^2)]^2}\Big[\langle\Psi_0|\rho(\mathbf{q},\omega_{el})\rho(\mathbf{q},\omega_{el})|\Psi_0\rangle - \langle\Psi_0|\rho(\mathbf{q},\omega_{el})|\Psi_0\rangle^2\Big] \tag{12}$$

where $G_p^e(q^2)$ is the proton electric form factor. We currently are in the production phase, and will soon have the necessary statistical significance, hopefully allowing us to predict the data of a recent Jefferson Lab experiment which is nearing publications.

This is not yet the end of the story: the subroutines for the sum rules of the weak response have been presently developed by the Los Alamos group and already tested in VMC calculations of small nuclei. We plan to implement them in the GFMC code and tune them for Mira in the very next months.

[1] Omar Benhar, Donal Day, and Ingo Sick. Inclusive quasielastic electron-nucleus scattering. *Rev. Mod. Phys.*, 80:189–224, Jan 2008.

[2] G. Shen, L.E. Marcucci, J. Carlson, S. Gandolfi, and R. Schiavilla. Inclusive neutrino scattering off deuteron from threshold to GeV energies. *Phys.Rev.*, C86:035503, 2012.

[3] Steven Pieper. Monte carlo calculations of nuclei. In Jesùs Navarro and Artur Polls, editors, *Microscopic Quantum Many-Body Theories and Their Applications*, volume 510 of *Lecture Notes in Physics*, pages 337–357. Springer Berlin / Heidelberg, 1998. 10.1007/BFb0104530.

[4] Ralph M. Butler Ewing L. Lusk, Steven C. Pieper. More scalability, less pain: A simple pro-

gramming model and its implementation for extreme computing. `http://www.cs.mtsu.edu/~rbutler/adlb/`.

16 NAMD - The Engine for Large-Scale Classical MD Simulations of Biomolecular Systems Based on a Polarizable Force Field

PI: Benoit Roux *(University of Chicago, Argonne National Laboratory)*

Project Summary

Biology, at the atomic and molecular level, is governed by complex interactions involving a large number of key constituents, including water, ions, proteins, nucleic acids, and lipid membranes. The goal of this project is to develop new technologies to simulate virtual models of biomolecular systems with an unprecedented accuracy. Large-scale molecular dynamics (MD) simulations based on atomic models play an increasingly important role in the life sciences. Success with MD simulations of large-scale biomolecular systems hinges on the accuracy of the potential function and the efficiency of the dynamic propagation algorithm for adequate sampling of motions.

This project is focused on the program NAMD, currently one of the most optimal and efficient programs to carry out classical simulations of biomolecular systems. To enhance the sampling efficiency beyond that of brute-force MD simulations, researchers propose to implement several advanced strategies based on multiple copies such as replica-exchange MD (REMD) and/or Hamiltonian tempering (H-REMD). Because the quality and accuracy of the potential function (force field) is critical for meaningful MD simulations, the researchers will implement a new force field that incorporates the effect of induced polarization. They will carry out simulations covering a wide range of canonical and non-canonical DNA and RNA molecules for which a wealth of experimental data is available. In the case of the structures determined via X-ray crystallography, simulations will be performed in solution as well as in the crystal environment allowing for the impact of crystal contacts on the simulated structure and dynamics. The researchers will examine the performance of the new force field for a suite of key problems where induced polarization is anticipated to be critical. Work will include calculation of pKa shifts in selected proteins, redox potentials, cooperative binding of Ca2+ to the EF-hands in calbindin D9k, and interfacial potentials of lipid monolayers and bilayers.

Report originally released as ANL/ALCF-ESP-13/14

NAMD - The Engine for Large-Scale Classical MD Simulations of Biomolecular Systems Based on a Polarizable Force Field

PI: Benoit Roux
ESP Postdoc: Yun (Lyna) Luo
Catalyst: Wei Jiang

ALCF Early Science Program Technical Report – April 2013

Large-scale Molecular dynamics (MD) simulation based on atomic models provide a powerful tool to understand the structure-dynamics-function relationships of important biological systems. Recent advances in computing power, especially in supercomputer, make MD simulation more and more popular in modern scientific community. However, the power of classical MD simulation has been limited mainly by the accuracy of the potential energy function and the efficiency of the dynamic algorithm enabling the adequate configurational sampling. This Early Science Project is aimed to make MD simulation method go beyond current limit using the leadership supercomputer Blue Gene/Q Mira. To address the issue of potential energy accuracy, classical Drude oscillator is developed to take into account the electronic polarizability in molecular systems. In order to develop and test the Drude polarizable force field for biomolecules, extensive MD simulations of typical polypeptides, lipids are needed. This requires efficient dynamic sampling algorithm that allows fast sampling. The sampling issue can be addressed by using advanced strategies based on multiple copies in order to enhance the sampling efficiency of brute-force MD. One such method is called replica-exchange MD (REMD).

In the replica-exchange MD (REMD) approach, several copies of the molecular system are simulated concurrently under slightly different conditions, e.g., different temperatures or Hamiltonians. Attempts are periodically made to exchange parameters between different replicas using a Metropolis Monte Carlo acceptance criterion, thus insuring Boltzmann-weighted statistics. Such multiple copy algorithms (MCAs) have been shown that enhance the sampling and free energy convergence.

NAMD is currently one of the most optimal and efficient programs to carry out classical MD simulations of biomolecular systems. NAMD is built on top of charm++ that obtains adaptive overlap of communication and computation. However the previous implementation of REMD on NAMD is driven by external job script, which can only be used on small clusters. This limits the use of REMD algorithm on the leadership supercomputer platform. The real power of REMD relies on the use large number of replicas (up to thousands of replicas), and high frequency of exchange (>1 ps^{-1}). Such requirement is beyond the reach of small computer clusters. It has become essential to develop extremely scalable MPI level REMD scheme to make full use of leadership supercomputers evolving toward multi-millions of cores.

The standard Charm++ was enhanced to support parallel/parallel simulations with multiple copies within a single program execution. The implementation of multiple

copies is completely at the MPI level. In the MPI machine layer of Charm++, MPI_Comm_split function splits the default MPI_COMM_WORLD into multiple local sub-communicators, each of which runs an independent Charm++ and NAMD instance. A secondary set of MPI communicators, each spanning like ranks within the local (sub)communicators, allows exchanges between the independent NAMD instances. This exchange communication is implemented through new APIs in both Charm++ converse layer and the NAMD Tcl scripting interface to provide a user-friendly interface. The breakthrough of such implementation is shown in three aspects: Scalability, Generality and Flexibility, Easy Post Processing.

I. Scalability on Blue Gene/Q Mira

MPI parallel/parallel multiple copy algorithm in NAMD is much more efficient than the old Tcl server and socket connections driving a separate NAMD process for every replica. Because NAMD program does not need to be restarted after each exchange, and the communication overhead is minimized by swapping parameters (temperature, lambda or biasing potential in ColVars) instead of coordinates (as in CHARMM), there is almost no performance lost even with large number of replicas and high exchange frequencies.

Pthread is implemented in charm++ so NAMD runs well in SMP mode on BG/Q Mira. The machine layer of charm++ interfaces with PAMI. QPX is implemented in the pairwise force/energy calculation of NAMD with xlc compiler intrinsic functions. Multithreaded mpi-smp version Charm++ 6.4.0 is built for running REMD

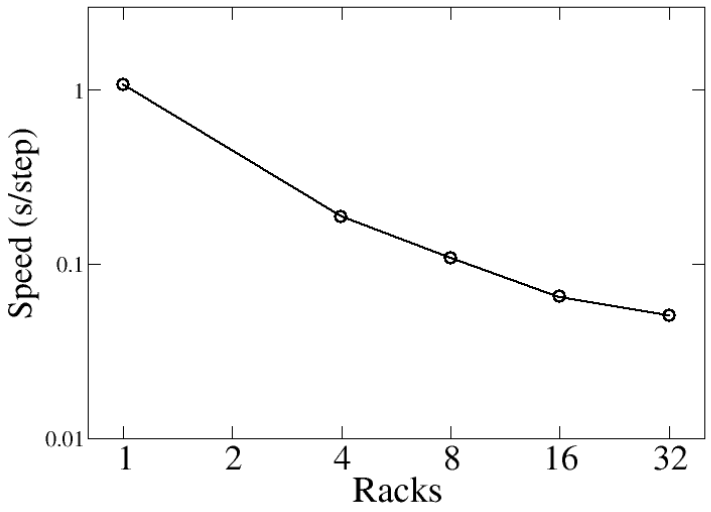


Figure 1 gives benchmarks for a multiple copies run with 1024 replicas. The calculation was carried out on BG/Q Mira using NAMD 2.9 with Charm++ 6.4.0 enhanced to support extremely scalable MCAs simulations. The REMD simulation scales up to 32 racks on BG/Q (i.e., 195 atoms/core) using16 MPI ranks per node and 2 working threads and 1 communication thread per rank.

II. Generality and Flexibility

To apply REMD and its variant, one usually needs to change the source code of some molecular dynamics (MD) simulation packages, which may not be so easy for general users. Here, we tested that a variant of REMD algorithms (T-REMD, FEP/REMD, US/REMD and REXAMD) could be performed by a standard NAMD2.9 package without touching the source code. Through the Tcl script user can define which parameter (temperature, lambda or biasing potential in ColVars) to exchange and which acceptance criterion to apply. Both NVT and NPT ensemble can be applied to REMD. All of the topology is contained in the user-defined replica_neighbors Tcl proc. User can define a list of exchange neighbors for a given replica and define any exchange pattern on any topology. Thus it works for any-dimensional sampling in any-shaped regions.

III. Easy Post Processing

Replica exchanges and energies are recorded in the .history files and the biasing potential applied in US/REMD is recorded in colvars.traj files. sortreplicas, found in the namd2 binary directory, is a program to un-shuffle replica trajectories to place the same temperature or order parameter frames in the same file.

17 Global Simulation of Plasma Microturbulence at the Petascale & Beyond

PI: William Tang *(Princeton Plasma Physics Laboratory, Princeton University)*

Project Summary

As the current global energy economy focuses on alternatives to fossil fuels, there is increasing interest in nuclear fusion, the power source of the sun and other stars, as an attractive possibility for meeting the worlds growing energy needs. Properly understanding turbulent transport losses, which demands the application of computational resources at the extreme scale, is of the utmost importance for the design and operation of future fusion devices, such as the multi-billion dollar international burning plasma experiment known as ITER a top priority investment in the Department of Energys Office of Science. This Early Science project will achieve significantly improved understanding of the influence of plasma size on confinement properties in advanced tokamak systems such as ITER. This will demand a systematic analysis of the underlying nonlinear turbulence characteristics in magnetically confined tokamak plasmas that span the range from current scale experiments, which exhibit an unfavorable "Bohm-like" scaling with plasma size to the ITER scale plasma that is expected to exhibit a more favorable "gyro-Bohm" scaling of confinement. The "scientific discovery" aspect of such studies is that while the simulation results can be validated against present-day tokamaks, there are no existing devices today that are even one-third of the radial dimension of ITER. Accordingly, the role of high physics fidelity predictive simulations takes on an even more important roleespecially since the expected improvement in confinement for ITER-sized devices cannot be experimentally validated until after it is constructed and operational. In dealing with this challenge, researchers will deploy GTC-P and GTS, which are highly scalable particle-in-cell gyrokinetic codes used for simulating microturbulence-driven transport in tokamaks.

Report originally released as ANL/ALCF-ESP-13/15

Optimizing the GTC Code for Blue Gene/Q

ALCF-2 Early Science Program Technical Report

William Tang,[1,2] Stephane Ethier,[1] Bei Wang,[2] Timothy Williams,[3]
Khaled Ibrahim,[4] Kamesh Madduri,[5,4] Samuel Williams,[4] and Leonid Oliker[4]

[1]*Princeton Plasma Physics Laboratory*
[2]*Princeton University*
[3]*Argonne Leadership Computing Facility, ANL*
[4]*Lawrence Berkeley National Laboratory*
[5]*The Pennsylvania State University*

I. SCIENCE

Figure 1 shows, schematically, the *tokamak* device, whose intent is to magnetically confine a high-temperature plasma and produce energy through nuclear fusion. The confined plasma, shown in pink, is toroidal in shape. For several decades now, scientists have been studying and improving these devices, working toward a successful fusion *ignition*—creating a burning plasma—and sustaining it to generate more power than it took to create it. The next big experimental step will be the International Thermonuclear Experimental Reactor, a twenty billion dollar burning plasma device under construction in France, involving the partnership of seven governments.[1] The human in Figure 2 illustrates the scale of ITER.

The magnetic field in the tokamak confines the plasma—ions, electrons, and their heat and momentum. Various kinds of instabilities in the plasma work against that confinement. Turbulent fluctuations can cause transport of particles and energy across the magnetic field lines, toward the outside of the plasma, where it is lost. Loss of plasma and energy, of course, works against confinement and successful fusion. This *microturbulence* is something the fusion community needs to understand and control.

In past and present tokamaks, measurements have shown that the transport of energy and particles caused by microturbulence depends on the size of the tokamak—larger means more transport. This is *Bohm* scaling[2]. However, theoretical arguments predict that beyond a certain size, which ITER will be beyond, the size dependence goes away, leading to relatively smaller turbulent losses and thus better confinement. This is *GyroBohm* scaling[3]. The key target of this ESP project is to simulate microturbulent transport for devices of different sizes up through ITER size, and validate and understand GyroBohm scaling.

II. NUMERICAL METHOD

To study low-frequency microturbulence for magnetically confined plasmas, we start from the Vlasov equation in six-dimensional phase space for each particle species. In the gyrokinetic approach[4], by removing the high frequency motion of the particles that is not important to turbulent transport, we reduce the six-dimensional equation to a five-dimensional Vlasov equation:

$$\frac{df_\alpha}{dt} = \frac{\partial f_\alpha}{\partial t} + \frac{d\mathbf{R}}{dt} \cdot \frac{\partial f_\alpha}{\partial \mathbf{R}} + \frac{dv_\parallel}{dt}\frac{\partial f_\alpha}{\partial v_\parallel} = 0, \tag{1}$$

where $f_\alpha(\mathbf{R}, v_\parallel, \mu)$ is the five-dimensional phase space distribution function for species α in the gyrocenter coordinates $\mathbf{R}$ and $v_\parallel$ is the velocity parallel to the magnetic field.

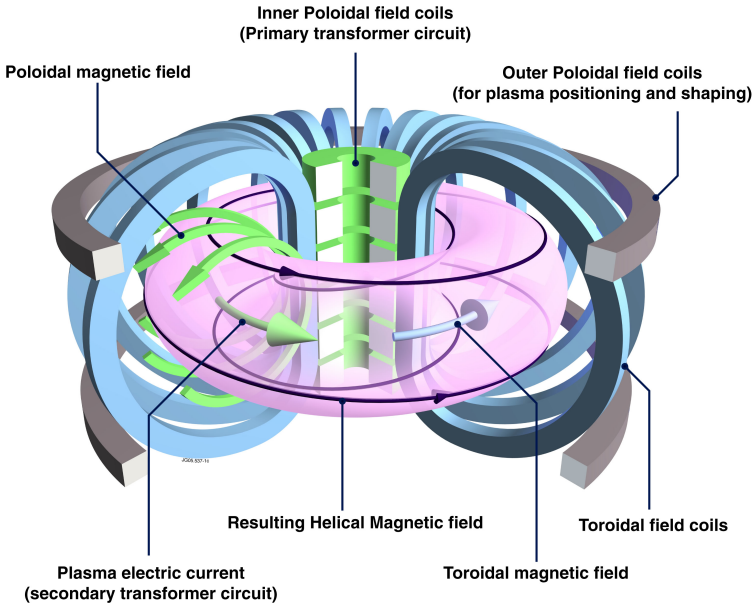


FIG. 1: **Tokamak. (Source: EFDA-JET)**

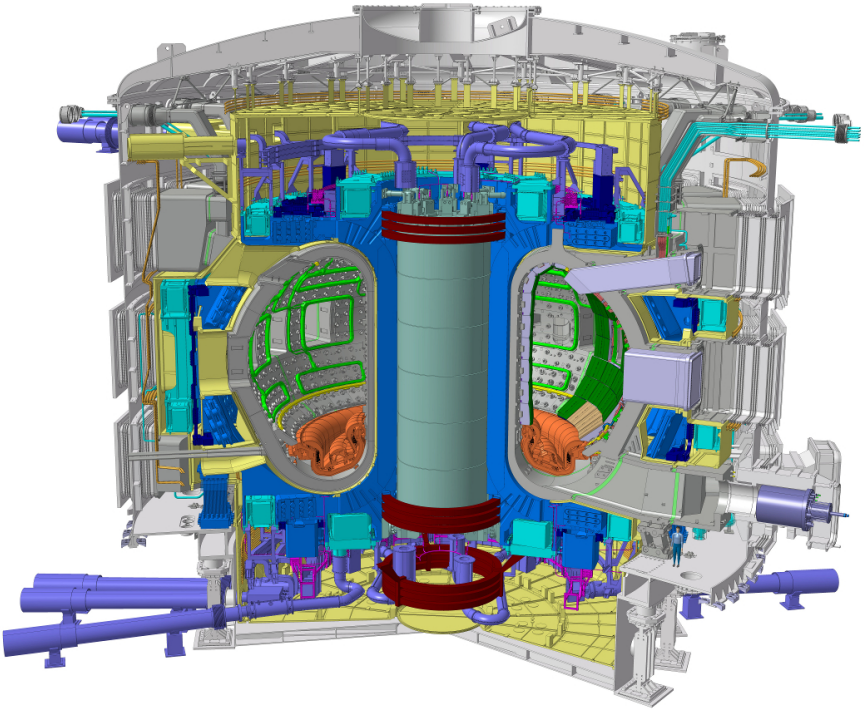


FIG. 2: **ITER. Human figure toward lower right corner indicates scale. (Image credit: © ITER Organization, http://www.iter.org/)**

Since plasmas are charged, we must also solve the relevant electromagnetic field equations. In the electrostatic gyrokinetic regime we're studying, the full Maxwell equations reduces to a single gyrokinetic Poisson equation.

In this project, we solve gyrokinetic Vlasov-Poisson system of equations using the Particle-In-Cell (PIC) approach.[4] Put simply, the ions are represented by a set of particles having the shape of

charged rings perpendicular to the magnetic field, having radius equal to the ion gyroradius. The electric field is represented on a grid, through which the particles move continuously. The field is interpolated from the grid to each of four points on the gyro-orbit, which then is averaged to get the force that moves the particles (the particle *push*). Given the position of the particles, the electric charge of each particle is deposited onto a charge density array defined on the grid (one particle contributes to a small neighborhood of grid points around each of the 4 gyro-orbit points on the ring. Given the charge density on the grid, the nonlinear gyrokinetic Poisson equation is solved using a custom solver. The gradient of this potential field gives the electric field, which is then used to push the particles again, and so on. Figure 3 illustrates the process for a time step. In the work discussed here, the electrons in the plasma are treated as *adiabatic*, meaning they follow a simple Boltzmann response function (*i.e.,* not treated as particles).

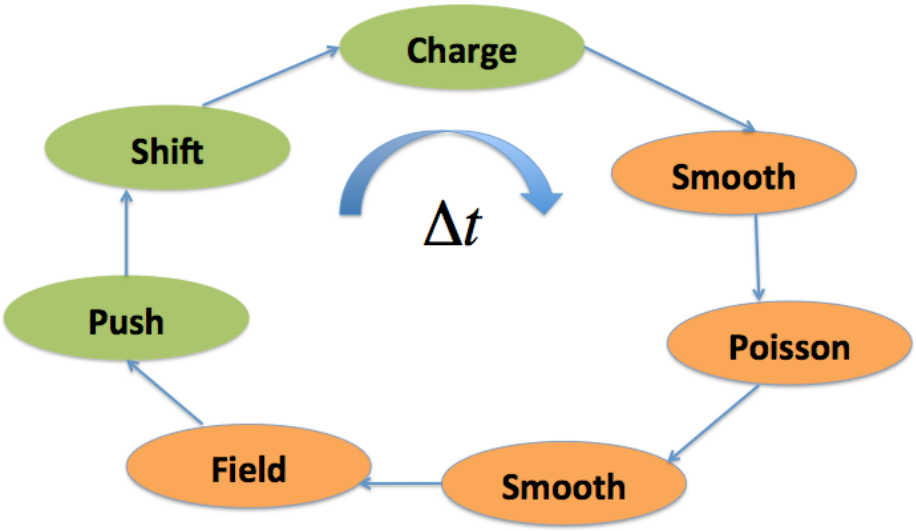


FIG. 3: Timestep elements for self-consistent evolution of plasma particles and electrostatic field. *Push:* interpolate electric field to positions of 4 gyro-orbit points of particles, use these to compute force and move particles. *Shift*: Send and receive particles that move between separate subdomains in the parallel decomposition of particles and fields. *Charge:* for 4 points on gyro-orbit of each particle, accumulate fractional charge density onto a neighborhood of grid points. *Smooth:* Smooth charge density and potential with a filter on the grid. *Poisson:* Solve gyrokinetic Poisson equation to get potential on the grid.

The grid geometry we use approximates the toroidal cross section as circular. Grid lines the long way around the torus follow magnetic field lines, which wind around helically. Figure 4 illustrates the coordinates, geometry, and several particles. The particles are rings with the 4 gyro-orbit points indicated, along with the neighborhood of grid points used to accumulate the charge density from or interpolate the force (electric field) from those 4 points. Not shown is the interpolation along the zeta direction. Psi is the radial coordinate, theta is the poloidal angle, and zeta is the toroidal coordinate.

III. CODES

The implementations we discuss here derive from the Gyrokinetic Toroidal Code (GTC)[5]. The variants we use and compare are two implementations of GTC-P, an electrostatic code with a circular cross section. Most newer tokamaks, and ITER, have D-shaped plasma cross sections, but

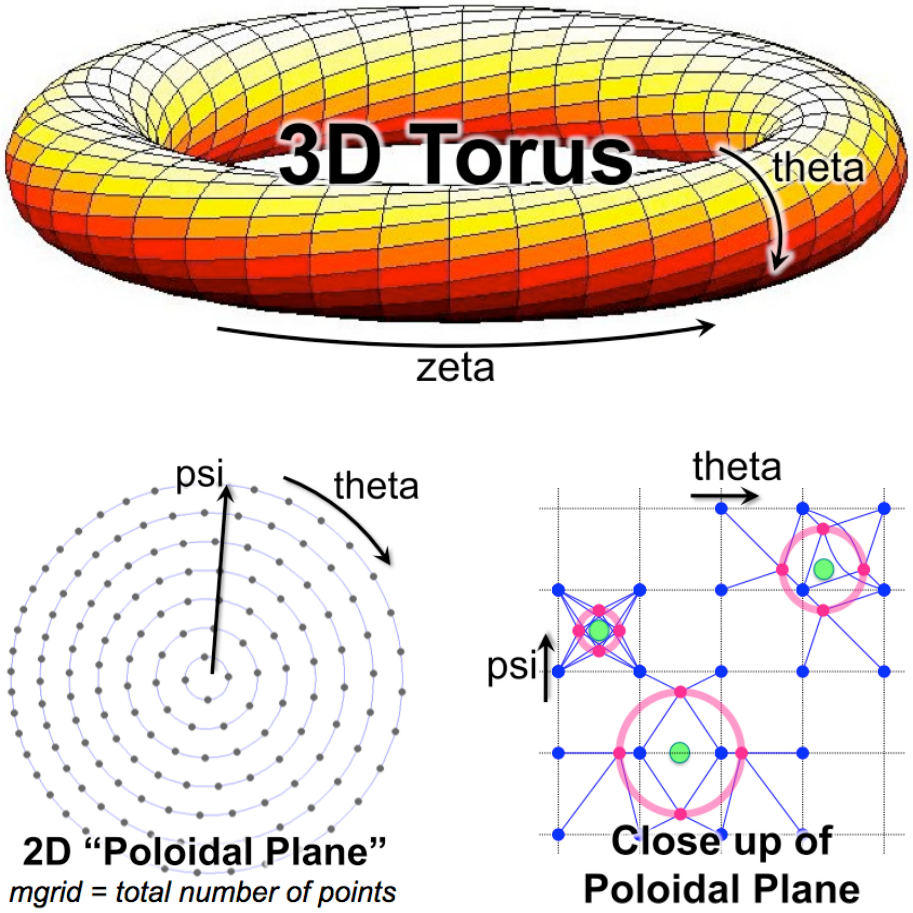


FIG. 4: An illustration of the 3D toroidal grid (top), a cross section (lower left), and the 4-point gyrokinetic averaging scheme employed in the charge deposition and push steps (lower right).

the effects of that geometry are not necessary to the basic physics we're investigating. When the ESP project started, until recently, we used a Fortran implementation, which we'll call GTC-P Fortran. More recently, we have moved to a new, from-scratch implementation in C, called GTC-P C.

IV. PARALLELIZATION

The original GTC code used 3 levels of parallelization:

1. One-dimensional domain decomposition in the toroidal (zeta) direction. Because of the physics of the plasma in the regime of interest, and because of the efficiency of the magnetic-field-line-following grid, we need only 64 grid zones toroidally, even for the largest problems we run. This limits parallelism in this dimension to 64; the MPI ranks are divided into a maximum of 64 subsets, one per toroidal subdomain.

2. Particle decomposition. Within each toroidal subdomain, the particles in that domain are distributed among the MPI ranks. Each rank maintains its own copy of the whole sub grid for the subdomain—all the poloidal and radial grid points, usually on two poloidal planes

bounding one toroidal grid zone. For the charge deposition from particle gyro-orbit points to the grid, it is necessary to use an `MPI_Allreduce` to sum up the contributions from all the MPI ranks' local grid copies.

3. Thread parallelism. For particle and grid operations within local toroidal subdomains, we use OpenMP to parallelize relevant loops over both particles and grid points.

More recently, we added an additional level of MPI parallelism: radial decomposition. Radial domain decomposition begins by partitioning a poloidal plane to non-overlapping domains with equal area. Assuming particle density is uniform, this partitioning divides all particles in one toroidal section equally across multiple processes. Next, the non-overlapping domain is extended to line up with the mesh boundary in the radial direction (shown as valid grid in Figure 5). Finally, the valid grid is extended on each side with ghost cells accounting for charge deposition with 4-point approximation (shown as local grid in Figure 5). In general, 3 to 8 ghost cells are sufficient. The 2D domain decomposition is implemented with MPI using two different communicators: a toroidal communicator and a radial communicator. The particles move between domains with nearest-neighbor communication in a circular fashion. Since the number of particles moving in the radial dimension is much smaller than the particles moving in the toroidal dimension, radial partitioning results in minimal communication. Within radial subdomains, we still allow partitioning of particles among more than one MPI rank.

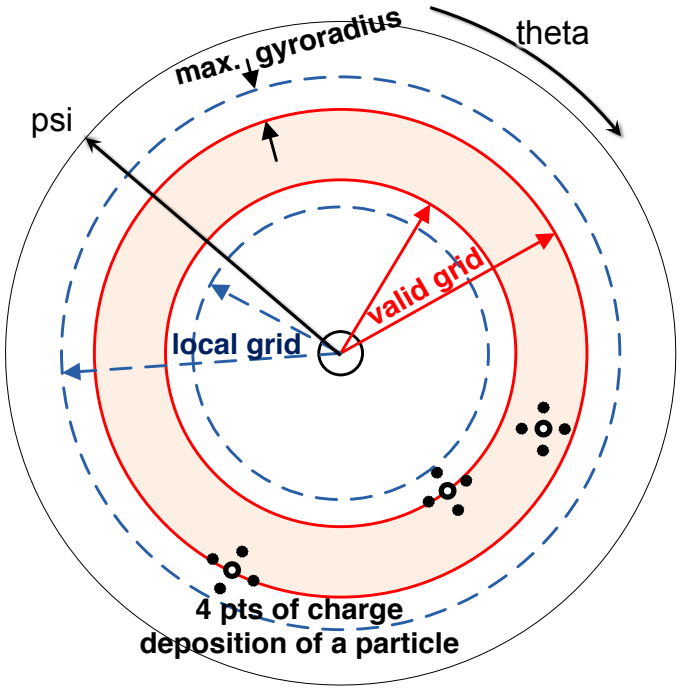


FIG. 5: A geometric radial partitioning with extended ghost zones. The valid region contains guiding centers for the MPI rank owning the radial subdomain. The full local region includes a number of ghost zones based on the maximum gyroradius of the particles (constant, typically 8, and the same for all radial subdomains).

V. KERNELS

Figure 3 shows 6 basic kernels in GTC-P, which can be grouped into 4 groups:

Charge Deposit charge from the 4 gyro-orbit points for each particle onto a neighborhood of grid points (neighborhoods are up to 32 grid points for each gyro-orbit point). This *scatter-add* operation must be managed when multiple ranks and/or threads are updating the same grid zones—either through using locks or using separate local copies of the grid, which then must be summed across ranks and/or threads.

Poisson/Field/Smooth Solve the gyrokinetic Poisson equation, compute the electric field, and smooth the charge density and potential fields. These are all grid-only operations. Because of the physics modeled (Debye shielding term much smaller than ion polarization term), it is sufficient to solve only independent 2D Poisson equations, one for each poloidal-radial plane.

Push Gather the force at each gyro-orbit point for each particle and use it to compute the force and advance the particle's phase space coordinates. The gather operation is the inverse of the scatter-add operation in the **charge** kernel. Here, we're only reading from grid arrays, so there is no locking or synchronization needed.

Shift As particles (that is, their gyro-orbit centers) move through space, they will cross subdomain boundaries into regions owned by other MPI ranks. they must be buffered and sent to the appropriate neighboring rank.

VI. OPTIMIZATIONS

Since the beginning of the Early Science Program, we have worked on two basic types of optimization of GTC-P in preparation for running on *Mira*:

1. Generic optimizations in single-core computations, parallel decomposition and message passing, and OpenMP threading.

2. IBM Blue Gene/Q specific optimizations

Here we will briefly touch on the generic optimizations, then discuss the BG/Q-specific optimizations. The full set of optimizations have been made only in the C version, GTC-P C, though any of them could also have been made to GTC-P Fortran. We discuss the performance impact of the optimizations in section VII.

A. Generic Optimizations

GTC-P Fortran used arrays of structures for particle data (each element stores all data quantities carried by a particle, including extras to handle two time levels in time integration). One optimization in GTC-P C is to change this to structures of arrays, to improve data locality of access when streaming through all the particles. This optimization benefits SIMD architectures in generally, including BG/Q and also GPGPU machines.[6–8]

As discussed in section III, GTC-P Fortran required each MPI rank to store a complete copy of the 2D radial-poloidal grid. The radial decomposition discussed there means that storage can be

reduced as needed for optimal performance on a given architecture by tuning the radial decomposition.

To increase data locality and improve cache hits on grid accesses for the **charge** and **push** kernels, we periodically sort particles into radial bins. This tends to improve cache reuse for data loaded from the grid arrays. The binning algorithm is multithreaded.[6] Locality can further be improved by accounting for the 4 gyro-orbit points associated with each particle (charged ring). We bin the coordinates of the gyro-orbit points radially, using a preliminary pass through the particles to store the 4 points. The second pass through all these binned points only touches field grid points in a band of narrow width radially; compared to a gyro-radius-wide band of points if you are going through all 4 points for a single particle. In this way, data locality and cache reuse is improved. We use this two-level binning for the **charge** kernel; it doesn't improve performance on present-day machines in the **push** kernel.

We have also used loop-fusion to improve computational intensity. We fused OpenMP loops where possible to minimize thread creation overhead. We flattened 2D and 3D grid arrays to 1D arrays. Additionally, we pre-allocated memory buffers that are used for temporary storage in every time step.

Grid-based subroutines in GTC-P included nested loops with *psi* is the outer loop and *theta* is the inner. Near the center radially, the number of *theta* grid points changes significantly (as a percentage) with each increasing radial grid level. To mitigate load imbalance in multithreading these loop nests, we flattened the nest into a single loop over all grid points. Finally, GTC-P Fortran used version 2.3.3 of the PETSc library[9–11], which is not multithreaded, to solve the gyrokinetic Poisson equation; we replaced this with a multithreaded, hand-coded solver in GTC-P C. (*N.B.:* Newer versions of PETSc have added thread support.)

B. Blue Gene/Q Specific Optimizations

Many of the generic optimizations discussed previously led to improvement in GTC-P performance on *Mira*. The general goal was scaling up to much higher levels of concurrency, which is what we got with *Mira*, both in total number of cores ($\sim$ 750 million)and in total number of threads (4 hardware threads per core).

One BG/Q optimization that is important when running on large numbers of nodes is the mapping of MPI ranks onto the physical nodes and cores on the machine. IBM defines a naming convention for the communication dimensions in the torus: "ABCDET" means that MPI ranks are mapper to the system in a particular order: Each letter is associated with a dimension in the 5D torus (with "T" being an index across the MPI ranks (processes) in a single node).[12] The last dimension (T in this example) is fastest varying, so, if running with one MPI rank per node:

- MPI rank 0 is assigned to coordinates $\langle 0, 0, 0, 0, 0, 0 \rangle$
- MPI rank 1 is assigned to coordinates $\langle 0, 0, 0, 0, 1, 0 \rangle$
- MPI rank 2 is assigned to coordinates $\langle 0, 0, 0, 0, 2, 0 \rangle$
- ....

The second-to-last dimension (E in this example) is the next-fastest varying, and so on. The highest value for each dimension depends on the number of nodes in the partition you're using within the machine. For anything 512 nodes or larger, you have your own isolated partition on BG/Q, generally itself a torus. The optimized GTC-P C code uses a two-dimensional topology for point to point communication, where the first dimension (toroidal dimension) has fixed dimensionality

64. On a BG/Q system with 5D torus network, we can thus group two or three torus dimensions together to match 64 for an optimized placement layout by setting the environment variable RUNJOB_MAPPING. Table I shows some examples of configurations and groupings when we run the application with one process per node. This explicit process mapping leads up to a 45% communication improvement for particle shift in the toroidal dimension using 8 racks (8192 nodes) of *Mira*.

Configuration	**Torus Shape**	**Grouping**
256 nodes (1/4 rack)	4 2 4 4 2	ABCE×D×T
512 nodes (1/2 rack)	4 4 4 4 2	ABC×DE×T (default)
1024 nodes (1 rack)	4 4 4 8 2	ABC×DE×T (default)
2048 nodes (2 racks)	4 4 4 16 2	ABC×DE×T (default)
4096 nodes (4 racks)	4 4 8 16 2	ACE×BD×T
8192 nodes (8 racks)	4 4 16 16 2	AC×BDE×T

TABLE I: Process Mapping on BG/Q. Not shown in the torus shape are the on-node"T" dimensions, whose maximal value depends on the chosen number of MPI ranks per node.

As BG/Q is a highly multithreaded architecture (up to 64 OpenMP threads per MPI rank), efficient use of OpenMP is essential for attaining high performance. Some of the generic optimizations in section VI A, especially those for grid-based calculations, improved the OpenMP performance and scalability. It is very important to be aware of environment variables controlling thread behavior on BG/Q. The settings BG_SMP_FAST_WAKEUP=YES and OMP_WAIT_POLICY=active make threads use an atomic-based spin barrier instead of a slower sleep-based approach. For some of the grid-based routines, these settings resulted in a 10× speedup when using 64 threads per MPI rank.

VII. PERFORMANCE

Here we present some performance measurements on a set of different problem sizes. These are the four problem sizes used to study the Bohm-to-GyroBohm scaling transition discussed in section I. The parameters are shown in Table II; the number of particles is based on the *micell* parameter, which is the number of particles per grid cell. Grid sizes A and B correspond to the majority of existing tokamaks in the world, C corresponds to the JET tokamak, the largest device currently in operation [10], and D corresponds to to ITER.

Grid Size	**A**	**B**	**C**	**D**
mpsi	90	180	360	720
mthetamax	640	1280	2560	5120
mgrid (grid points per plane)	32449	128893	513785	2051567
chargei grid (MB)	0.5	1.97	7.84	31.30
evector grid (MB)	1.49	5.90	23.52	93.91
Total particles *micell*=100 (GB)	0.29	1.16	4.64	18.56

TABLE II: The GTC numerical settings for different plasma sizes. The grid and particle memory requirement are for one toroidal domain only. A simulation typically consists of 64 toroidal domains. *mpsi* and *mthetamax* are the number of grid points in the radial (*psi*) and poloidal (*theta*) coordinates.

A. Strong Scaling

First, we consider strong scaling—running a fixed problem size on an increasing sequence of compute nodes. We denote the problem as D100. The problem size is D (ITER size), and we use the typical production setting of 100 particles per cell. (This is typical when running a *delta-f* simulation, where the particles represent only the difference of the phase space distribution with respect to a constant Maxwellian background distribution.) In the toroidal direction, we use *ntoroidal*=64 grid cells. This is distributed among 64 sets of MPI ranks, so each toroidal subdomain has two poloidal planes having 2 million grid points each. This global simulation is 130 million grid points and 13 billion particles.

For the first strong scaling test, we increase the number of radial partitions from 32 to 512. We run on *Mira* with 4 MPI ranks per node and 16 OpenMP threads per rank; the simulations scale from 512 to 8192 nodes (8192 to 131,072 cores). Table III shows the results. The GTC-P C code uses the optimizations detailed in section VI, which combine to give a substantial improvement in overall runtime ($\sim 2\times$ on 32768 nodes) and a substantial improvement in parallel efficiency. Note that these runs did not use particle decomposition; this allows strong scaling up to even larger node counts.

MPI Ranks	Radial Partitions	GTC-P Fortran	Parallel Efficiency	GTC-P C	Parallel Efficiency	Speedup
2048	32	9.282	1.0	5.275	1.0	1.76
4096	64	4.685	0.99	2.651	0.99	1.76
8192	128	2.536	0.92	1.373	0.96	1.85
16384	256	1.453	0.80	0.726	0.91	2.00
32768	512	0.873	0.60	0.414	0.80	2.21

TABLE III: Wall-clock time (sec) for one time step with strong scaling in radial domain decomposition for D100 using GTC-P Fortran and GTC-P C. In all experiments, we use 4 processes/node and 16 threads/process. GTC-P C attains a $2\times$ speedup due to our optimizations.

For the second strong scaling test, we consider scaling with respect to the number of OpenMP threads per MPI rank. Table IV shows the results. We hold the number of MPI ranks (processes) constant at 32768. As we increase from 2048 nodes to 32768 nodes by doubling, the amount of available hardware thread concurrency doubles, since we're reducing freeing up more cores. With perfect thread scaling, the run times would halve each step. For the D100 problem, the parallel efficiency at the largest number of nodes has dropped to 59%.

D100: 13 Billion particles, 32768 total processes, with 400556 particles per process											
Nodes	Processes×Threads	Charge	Push	Shift_t	Shift_r	Binning	Poisson	Field	Smooth	Total	Eff.
2048	16×4	0.6691	0.4569	0.3073	0.0326	0.0602	0.0197	0.0050	0.0069	1.558	1.0
4096	8×8	0.3397	0.2313	0.1536	0.0169	0.0340	0.0078	0.0027	0.0047	0.791	0.99
8192	4×16	0.1822	0.1178	0.0779	0.0094	0.0167	0.0044	0.0021	0.0039	0.414	0.94
16384	2×32	0.1110	0.0591	0.0722	0.0063	0.0087	0.0033	0.0022	0.0039	0.267	0.73
32768	1×64	0.0709	0.0298	0.0487	0.0050	0.0039	0.0022	0.0018	0.0035	0.166	0.59

TABLE IV: Strong scaling of threads per process for D100 using GTC-P C on *Mira*. The time is the wall-clock time (sec) for one time step. We use 64-way toroidal and 512-way radial partitioning.

For the third strong scaling test, we look at how the optimizations made in GTC-P C improve fine-grained parallelism with respect to GTC-P Fortran. Table V breaks down the relative perfor-

mance improvement by numerical kernel. The problem and set of runs are the same as in Table IV. The biggest speedups are in the grid-based kernels, because of the loop flattening and other optimizations in section VI. The speedups in **charge** and **push** are mainly because of binning, and avoiding synchronization in **charge**. Both the Fortran and C codes employ the private grid replication strategy on a per thread basis for charge deposition. However, GTC-P Fortran uses a `critical section` to merge charges from all copies private of threads. This serial portion of the code can be easily avoided by carefully reorganizing the summation order.

Nodes	**2048**	**4096**	**8192**	**16384**	**32768**
Thread/Process	**4**	**8**	**16**	**32**	**64**
charge	1.14×	2.06×	1.95×	3.52×	8.80×
push	1.71×	1.71×	1.69×	1.72×	1.70×
shift	1.37×	2.05×	2.23×	2.23×	2.85×
poisson	6.84×	12.05×	13.57×	13.55×	16.45×
field	41.02×	35.74×	28.43×	20.32×	20.11×
smooth	28.87×	23.87×	21.72×	16.46×	16.40×
overall speedup	1.43×	1.78×	2.11×	2.87×	5.56×

TABLE V: Speedup (GTC-P C vs GTC-P Fortran) on D100 problem by kernels with different threads per process. The total processes (32768), the number of particles in each process (400556) and the number of radial partitions (512) are fixed. Shift includes shift_t, shift_r and binning in Table IV.

B. Weak Scaling

Next, we consider weak scaling—running a sequence of problem size on an increasing sequence of compute nodes, keeping the amount of work per node constant. We use the four problem sizes used to study the Bohm-to-GyroBohm scaling transition discussed in section I. We use 100 particles per cell, one MPI rank per node, and 64 threads per rank.

The parallelism in GTC is denoted as three parameters: the number of toroidal partitions (*ntoroidal*), the number of radial partitions (*nradiald*), and the number of particle partitions in one spatial subdomain (*npe_radiald=npartdom/nradiald*, where *npartdom* is the total number of particle partitions in one toroidal partition). Table VI shows the settings for the weak scaling study. Figure 6 shows the results, comparing GTC-P Fortran and GTC-P C. Some efficiency loss is still seen for GTC-P C, but keep in mind that here we are using 2/3 of *Mira*: 524,288 cores.

Problem Size	**Nodes**	***ntoroidal***	***npartdom***	***nradiald***	***npe_radialid***
A	512	64	8	1	8
B	2048	64	32	1	32
C	8192	64	128	1	128
D	32768	64	512	1	512

TABLE VI: Parameters for weak scaling study

Acknowledgments

Dr. Wang was supported by the NSF OCI-1128080/G8 Initiative: G8 Research Councils Initiative on Multilateral Research Funding. Authors from Princeton Plasma Physics Laboratory were by the DOE

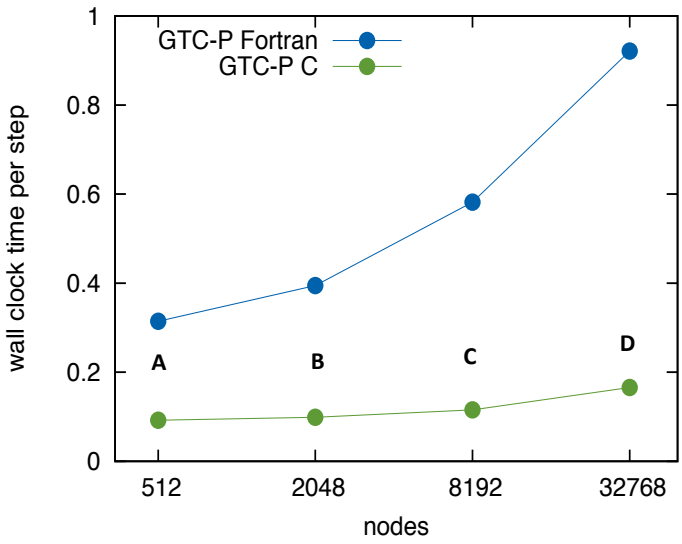


FIG. 6: Weak scaling from A to D size plasmas using GTC-P Fortran and GTC-P C on *Mira*.

contract DE-AC02-09CH11466. Authors from Lawrence Berkeley National Laboratory were supported by the DOE Office of Advanced Scientific Computing Research under contract number DE-AC02-05CH11231. Dr. T. Williams was supported by the DOE contract number DE-AC02-06CH11357. This research used resources of the Argonne Leadership Computing Facility, which is supported by the Office of Science of the U.S. Department of Energy under contract DE-AC02-06CH11357. This research used resources of the National Energy Research Scientific Computing Center, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

---

[1] Y. Shimomura, R. Aymar, V. Chuyanov, M. Huguet, R. Parker, and ITER Joint Central Team. Iter overview. *Nuclear Fusion*, 39(9Y):1295, 1999.

[2] Andrew Guthrie and Raymond Kornelious Wakerling. *The characteristics of electrical discharges in magnetic fields*, volume 5. McGraw-Hill, 1949.

[3] G. Manfredi and M. Ottaviani. Gyro-Bohm Scaling of Ion Thermal Transport from Global Numerical Simulations of Ion-Temperature-Gradient-Driven Turbulence. *Physical Review Letters*, 79(21):4190–4193, November 1997.

[4] W.W Lee. Gyrokinetic particle simulation model. *Journal of Computational Physics*, 72(1):243–269, September 1987.

[5] Z. Lin. Turbulent Transport Reduction by Zonal Flows: Massively Parallel Simulations. *Science*, 281(5384):1835–1837, September 1998.

[6] Kamesh Madduri, Eun-Jin Im, Khaled Z. Ibrahim, Samuel Williams, Stphane Ethier, and Leonid Oliker. Gyrokinetic particle-in-cell optimization on emerging multi- and manycore platforms. *Parallel Computing*, 37(9):501 – 520, 2011.

[7] Kamesh Madduri, Khaled Z. Ibrahim, Samuel Williams, Eun-Jin Im, Stephane Ethier, John Shalf, and Leonid Oliker. Gyrokinetic toroidal simulations on leading multi- and manycore hpc systems. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '11, pages 23:1–23:12, New York, NY, USA, 2011. ACM.

[8] Kamesh Madduri, Samuel Williams, Stéphane Ethier, Leonid Oliker, John Shalf, Erich Strohmaier, and Katherine Yelicky. Memory-efficient optimization of Gyrokinetic particle-to-grid interpolation for multicore processors. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis - SC '09*, SC '09, pages 48:1–48:12, New York, New York, USA, November 2009.

ACM Press.

[9] Satish Balay, Jed Brown, Kris Buschelman, William D. Gropp, Dinesh Kaushik, Matthew G. Knepley, Lois Curfman McInnes, Barry F. Smith, and Hong Zhang. PETSc Web page, 2012. http://www.mcs.anl.gov/petsc.

[10] Satish Balay, Jed Brown, , Kris Buschelman, Victor Eijkhout, William D. Gropp, Dinesh Kaushik, Matthew G. Knepley, Lois Curfman McInnes, Barry F. Smith, and Hong Zhang. PETSc users manual. Technical Report ANL-95/11 - Revision 3.3, Argonne National Laboratory, 2012.

[11] Satish Balay, William D. Gropp, Lois Curfman McInnes, and Barry F. Smith. Efficient management of parallelism in object oriented numerical software libraries. In E. Arge, A. M. Bruaset, and H. P. Langtangen, editors, *Modern Software Tools in Scientific Computing*, pages 163–202. Birkhäuser Press, 1997.

[12] IBM Redbooks — IBM System Blue Gene Solution: Blue Gene/Q Application Development - Update, April 2013.

18 Multiscale Molecular Simulations at the Petascale

PI: Gregory Voth *(University of Chicago)*

Project Summary

Project results will directly impact the understanding of cellular-scale biological processes via coupling of multiscale computer simulation methodology with petascale computational algorithms and hardware. When combined with leading-edge experimental research, the project will provide key scientific advances relevant to human health and the understanding of the biological world.

Researchers will apply multiscale bio-simulation methodology to three scientific problems: (1) Simulation of key steps of the HIV viral replication cycle. The human immunodeficiency virus type 1 (HIV-1) begins assembly with multimerization of the Gag polyprotein near the plasma membrane of infected cells. There has been debate over exactly how many Gags are in the immature virion. The researchers will use their methodology to construct coarse-grained (CG) models directly from structural biology experiments and physics-based interaction modeling. (2) Simulation of the Ribosome. Protein biosynthesis is a core life process in cells, which is mainly achieved by ribosomes. Since the ribosomes are targets for drugs such as antibiotics, a deep understanding of the mechanism of protein synthesis can aid in drug discovery. (3) Simulation of microtubules. Microtubules are one of the key components of the cytoskeleton, involved in trafficking, structural support, and cytokinesis. Microtubules are polymers assembled from alpha/beta tubulin dimers in the form of tubes with a diameter of 25 nm and variable lengths. Because of their size and the long timescale dynamics of these assemblies, these large and important protein assemblies inherently require a petascale multiscale simulation approach.

Report originally released as ANL/ALCF-ESP-13/16

Parallelization of Reactive Force Field Model for Blue Gene/Q

Adrian W. Lange,[1] Gard Nelson,[2] Christopher Knight,[3] and Gregory A. Voth[2,4]

[1]*Leadership Computing Facility, Argonne National Laboratory, Argonne, IL 60439, USA*
[2]*Department of Chemistry, University of Chicago, Chicago, IL 60637, USA*
[3]*Computing, Environment, and Life Sciences, Argonne National Laboratory, Argonne, IL 60439, USA*
[4]*James Franck Institute, Institute for Biophysical Dynamics, and Computation Institute, University of Chicago, Chicago, IL 60637, USA*

(Dated: April 12, 2013)

Our efforts in developing a highly parallel implementation of the reactive force field model, the Multi-state Empirical Valence Bond (MS-EVB) method, are discussed. We have introduced multi-threading, a state decomposition parallelism, and replica exchange capabilities. Example calculations are presented to demonstrate the improved productivity and scalability of the new code on the Blue Gene/Q supercomputer, Mira. We show that we are now able to successfully scale to half of Mira and have the potential to scale even further.

I. INTRODUCTION

Modeling the dynamics of complex biological molecular systems, such as proteins solvated in water, is a crucial facet of understanding how life works at the level of atomic detail and also understanding how we might be able to develop biotechnology for energy production and/or medical purposes. The vast number of atoms present in biological systems, though, prohibits the application of highly accurate quantum mechanical electronic structure methods, because the computational cost of these methods scales exponentially with respect to the system size (*i.e.*, the number of electron basis functions). In order to study the dynamics of such molecular systems with atomic detail, one instead opts to employ a classical molecular mechanics force field whose energy and forces can be computed at orders of magnitude faster than electronic structure. A typical molecular mechanics force field developed specifically for biological molecules approximate chemical bonds as harmonic springs between two atoms. While this approximation works reasonably well for most molecular systems near equilibrium, it is unable to account for the realistic quantum mechanical nature of chemical bonds, which can break and form dynamically.

The Multi-state Empirical Valence Bond (MS-EVB) method[1,2] is one way to incorporate chemical reactivity into an otherwise non-reactive force field, enabling the study of chemical reaction dynamics in complex biological systems without suffering the expense of electronic structure calculations. In short, MS-EVB makes the *ansatz* that a reactive molecular system can be decomposed into a set of diabatic states, each representing one of several possible chemical bonding topologies, akin to the "resonance structures" familiar to most chemists. These diabatic states form a basis set, and the total system is then a linear combination of the states, written as

$$|\Psi\rangle = \sum_I c_I \, |\psi_I\rangle \ , \tag{1.1}$$

where $|\psi_I\rangle$ is a diabatic state with coefficient c_I. MS-EVB then constructs a model Hamiltonian matrix, $\mathbf{H}$. The diagonal elements, H_{II}, are simply the total energy of each state. The off-diagonal elements, H_{IJ}, are a coupling energy between two states that serve as a reactant and a product state (see Ref. 2 for more detail). Finally, the Hamiltonian is diagonalized according to a Schrödinger-like equation,

$$\mathbf{Hc} = E\mathbf{c} \tag{1.2}$$

to yield a set of eigenvalue energies, E, and eigenvectors, $\mathbf{c}$, containing the coefficients c_I. The minimum eigenvalue energy is selected as the final MS-EVB energy, and forces on each atom are computed via the Hellman-Feynman theorem. The MS-EVB approach can thus be viewed as a coarse-graining (or, a multi-scale) approach to quantum chemistry electronic structure, which allows us to access larger molecular systems sizes and longer time scales.

MS-EVB is a straightforward framework, yet it is a challenge to implement effectively in a code capable of taking advantage of modern supercomputers with thousands of core processors. In the remainder of this work, we discuss how we have tackled this challenge with several advances in our MS-EVB code, Rapid Approach to Proton Transport and Other Reactions (RAPTOR).[3] Specifically, we discuss the parallelization of RAPTOR tailored for running on the IBM Blue Gene/Q (BGQ) supercomputer, Mira, housed at the Argonne Leadership Computing Facility.

Throughout this work, we focus primarily on an example application of MS-EVB for proton transport through a transmembrane protein, Cytochrome *c* Oxidase (CcO). Cco serves as a representative system of interest, although the RAPTOR code is generalizable to a variety of other reactions and molecular systems as well. Our model CcO system consists of 159,519 atoms treated with the CHARMM22 force field for protein molecules and the CHARMM36 force field for the lipid molecules. Water molecules are treated with the SPC/Fw force field, in accord with the MS-EVB3 model,[2] which is used to model the proton transfer reactivity. We use a cutoff

of 10 Å (smoothly attenuated to zero with a switching function beginning at 8 Å) for van der Waals and short-range Coulomb pairwise interactions. Molecular dynamics (MD) simulations are carried out at constant volume and temperature with a Nose-Hoover thermostat. MD is propagated with Velocity-Verlet integration with 1 femtosecond time step intervals.

All calculations presented below are performed on the BGQ architecture supercomputers—each compute node containing 16 core processors and each core containing a quad floating point unit (FPU)—at the Argonne Leadership Computing Facility. Codes have been compiled with the IBM XL compilers with level -O3 optimizations. Performance of the codes is measured as productivity in terms of MD time steps per wall second.

II. RAPTOR IMPLEMENTATION

A. Interface to LAMMPS

The RAPTOR code is an optionally installed user package interface to the LAMMPS,[4] a widely used and freely available, open-source parallel molecular dynamics code written in C++. LAMMPS is parallelized mainly through a domain decomposition scheme, wherein a given unit cell, simulated with periodic boundary conditions (PBC), is divided in Cartesian space into many smaller domains, each of which is treated separately on an individual rank with Message Passing Interface (MPI). Intra-domain interactions are thereby computed entirely in parallel, and inter-domain interactions are handled with minimal MPI communications with neighboring domains.

The domain decomposition scheme in LAMMPS is very effective and scalable for the bonded interactions (*e.g.*, bonds, angles, dihedrals) and short-range non-bonded interactions (*e.g.*, van der Waals interactions) of a force field. Because the magnitude of short-range non-bonded interactions approaches zero rapidly, an interactions cutoff radius can be used. However, for a fixed size system, one cannot divide the unit cell into domains of size smaller than the cutoff radius without errors and/or loss of parallel efficiency, placing a limit on how many MPI ranks can be used with domain decomposition scheme alone. In addition, pairwise Coulomb interactions usually cannot be approximated with a cutoff because $1/r$ does not approach zero fast enough to ignore without possibly introducing severe errors.

The Particle-Particle Particle-Mesh[5] (PPPM) approach (available in LAMMPS) is therefore used in our calculations to divide the work into a short-range part, handled in conjunction with the pairwise van der Waals interactions via domain decomposition, and a long-range part, handled with a Fast Fourier Transform (FFT) performed as parallel calls to the FFTW library. The long-range electrostatics computation (*i.e.*, the k-space computation), despite being $O(NlogN)$ scaling, is unfortunately not nearly as efficiently parallel as the domain decomposition (for the relatively small three dimensional grids used in our calculations) because it involves a substantial amount of MPI communication in order to broadcast the electrostatics from the PPPM mesh globally to all processors, an all-to-all style communication. Moreover, each PPPM evaluation requires 4 3D-FFTs, one forward and three reverse. For MS-EVB calculations, this is compounded further by the fact that the MS-EVB model requires electrostatic energies in the off-diagonal coupling matrix elements, adding to the amount of k-space work. Indeed, at large counts of MPI ranks, the k-space computation can dominate the wall time in MS-EVB calculations with RAPTOR.[3,6] For this reason, previous work in our research group[3] had been devoted to developing a partitioning scheme in which the real-space bonded and non-bonded short-range interactions are performed on one partition of MPI ranks and the k-space computation is performed concurrently on another partition. Thereby, one could assign fewer MPI ranks to the k-space work to hide some of its poorly scaling communication, resulting in improved parallel efficiency and a modest overall speedup at a large number of processors.

To demonstrate this, we compare results of productivity for the Cco model using only the MPI domain decomposition in LAMMPS and also the k-space partitioning scheme (MPI/split) in Figure 1. At 64 nodes (1024 MPI ranks), the domain decomposition has reached the limit where further spatial division competes with the size of the cutoffs, hitting the intrinsic domain limit mentioned above.

As one can infer from Figure 1, RAPTOR does not scale very well with only the domain decomposition or the k-space partitioning scheme, dropping below 50% parallel efficiency at just 16 nodes, a far cry from the 49,152 nodes on Mira. The k-space partitioning improves the scalability slightly, but, again it cannot scale further due to the domain decomposition limit. Therefore, we are compelled to develop new parallelism approaches to improve scalability.

B. Multithreading with OpenMP and QPX SIMD

LAMMPS has recently added the ability (through an optional user package) to take advantage of shared memory multithreading parallelism via the OpenMP API. It does so with a so-called "force decomposition" scheme, where, for example, the pairwise additive N^2 force loop is distributed across threads. Similar loop-level parallelism is implemented for bonded interactions and parts of the PPPM k-space calculation, such as interpolating charges to the mesh. Note that the OpenMP force decomposition is in addition to the usual MPI domain decomposition in LAMMPS.

This multithreaded code, however, was implemented for the general purpose parts of LAMMPS, of which RAPTOR only uses for its diagonal matrix elements in

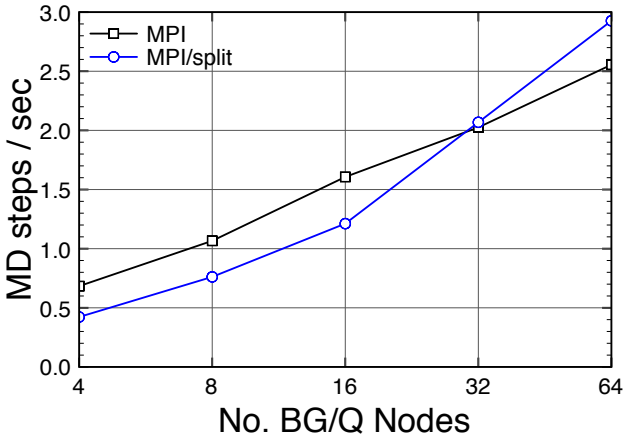


FIG. 1: Strong scaling of RAPTOR code with the CcO model comparing domain decomposition only (MPI) and k-space partitioning (MPI/split) schemes. Runs are carried out in c16 mode (*i.e.* with 16 MPI ranks per node). The Cco system contains 15 MS-EVB states on average.

the model Hamiltonian. So it was our task to merge this existing OpenMP code with RAPTOR and also rewrite a number of portions of RAPTOR that would benefit from such force decomposition multithreading. For instance, this involved multithreading certain pairwise loops for off-diagonal coupling matrix elements as well as rewriting parts of the LAMMPS PPPM interface with RAPTOR. In the process of this re-coding, we also introduced a handful of basic serial optimizations (*e.g.*, loop unrolling or conditional hoisting) that the compilers were unable to recognize in complex loop structures, amounting to a 15% speedup even with running only a single thread. In the PPPM interface of RAPTOR, we were also able to share a few of the FFT calls across thread partitions, providing some additional concurrency. The OpenMP multithreading was further combined with the k-space partitioning scheme, code which has also been introduced to the general release of LAMMPS now.

In addition to OpenMP multithreading, we explored the special BGQ-specific QPX vector intrinsics API to take advantage of the possibility of SIMD parallelism on BGQ. For our purposes, we were introduced QPX into the short-range non-bonded pairwise loops as well as parts of the PPPM code specific to RAPTOR. Since these loops have a number of branching conditional statements, it was not straightforward to take full advantage of QPX. We ultimately found that a "buffer/flush" approach was most successful. In this approach, we use a set of `vector4double` buffers to temporarily store information needed to complete a pairwise interaction (or other quantity). The bulk of the floating point operations in the loop are delayed until we have filled the buffers, and then the buffers are flushed by computing four pair interactions simultaneously with QPX vector intrinsic functions. We note that the usual LAMMPS pairwise loop uses a lookup table for the short-range Coulomb interaction, which otherwise involves a somewhat expensive polynomial expansion to approximate the error function. The lookup table is typically faster than the explicit evaluation of the polynomial, but a lookup table is not amenable to QPX vectorization. Thus, we applied QPX to the polynomial expansion and omitted the lookup table. Compared to no QPX vector intrinsics and without the lookup table, our buffer/flush code exhibits a $\sim$50% speedup for the non-bonded force kernel. Compared to no QPX and with the lookup table, though, we observe a $\sim$30% speedup for the non-bonded force kernel. While this is clearly shy of the theoretical speedup of QPX, we nonetheless have found it a welcome enhancement.

Putting all of the above together, we show in Figure 2 the improvements to scaling that are realized with OpenMP and QPX. The BGQ is capable of hyperthreading the quad FPU with OpenMP threads, but this may not always prove beneficial since it may prevent the use of QPX SIMD or possibly increase cache misses due to less memory per MPI rank. We examine this in Figure 2 with two different modes, c4o16 being 4 MPI ranks per node with 16 threads per rank (hyperthreaded mode), and c1o16 being 1 MPI rank per node with 16 thread per rank. It is quite clear that multithreading provides a significant speedup compared to the MPI domain decomposition or k-space partitioning for the same number of nodes. It also pushes the domain decomposition limit to greater node counts since threads reduce the number of MPI ranks per node. Notice that the c1o16 mode (the non-hyperthreaded mode) exhibits lower productivity than c4o16 mode (the hyperthreaded mode) at low node counts, but because c1o16 mode has better parallel efficiency, c1o16 scales further and eventually is more productive as node count increases. This appears to be the result of a combination of causes. One is the ability to use QPX in c1o16 mode both in the FFTs (automatically vectorized by compiler) and our special non-bonded force kernels. Another appears to be that the MPI communication is appreciably faster in c1o16 mode (as compared to c4o16 mode), likely because there is no intra-node communication in c1o16, making the MPI ranks closer in the communication network.

C. State Decomposition

Multithreading has certainly improved RAPTOR's parallelism, yet there has still been one aspect of RAPTOR that has been treated serially up to this point: the matrix elements in the model Hamiltonian are evaluated one at a time. In our model Cco example, there are on average 15 states in the MS-EVB Hamiltonian, and each state's energy and forces can be computed independently of the others. This glaringly obvious source of parallelism had gone untapped until now when we introduced our new "state decomposition" parallel scheme.

We accomplish the state decomposition in a manner

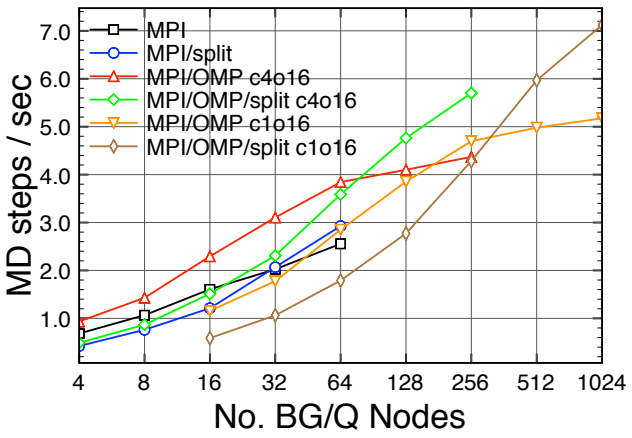


FIG. 2: Strong scaling of RAPTOR code with the CcO model with OpenMP multithreading and QPX (data labeled with OMP). Results from Figure 1 are included for comparison. The mode is listed for each OMP run. c4o16 = 4 MPI ranks per node and 16 threads per MPI rank (hyperthreaded mode). c1o16 = 1 MPI rank per node and 16 threads per MPI rank. MPI/OMP/split combines the OpenMP threading with the k-space partitioning scheme. Each line of data is extended to the domain decomposition limit.

analogous to the k-space partitioning scheme, but instead of dividing by real-space and k-space work for all MS-EVB states, we divide the work by individual MS-EVB state. This allows us to have not just two partitions (as in k-space partitioning) but as many partitions as we have MS-EVB states, distributing the work of both real-space and k-space across more processors. In practice, though, because the number of MS-EVB states is dynamic during the course of an MD simulation, we find that it is most efficient to use slightly fewer state partitions than there are MS-EVB states in order to avoid workless partitions.

The state decomposition scheme in RAPTOR is carried out by running LAMMPS in partitioned mode, which simply uses `MPI_Split` to create a new MPI subcommunicator for each partition. All partitions share the same coordinates and velocities so that they all end up with the same set of MS-EVB states after completing the MS-EVB state search algorithm.[2] Once the states have been determined, each partition decides which state(s) to work on with a simple static load balance and then computes the work. Thus, each partition stores in memory only a subset of the entire set of MS-EVB state energies and forces. With the energies and couplings in hand, the partitions perform an `MPI_Allreduce` of the MS-EVB Hamiltonian [a modest communication of data size $O(N_{\text{states}}^2)$ with N_{states} usually less than 20], which is subsequently diagonalized by each partition to yield the minimum eigenvalue energy (*i.e.*, the MS-EVB energy) and its corresponding eigenvector coefficients. Each partition computes the MS-EVB force on atoms according to the Hellman-Feynman theorem:

$$F_i = -\sum_{I,J}^{\text{states}} c_I c_J \frac{\partial H_{IJ}}{\partial \mathbf{x}_i} \,, \tag{2.1}$$

where F_i is the force on the i-th atom with respect to the $\mathbf{x}_i$ Cartesian coordinate, and c_I is the I-th coefficient from the minimum energy eigenvector. Since each partition has only stored in memory those matrix elements ($\partial H_{IJ}/\partial \mathbf{x}_i$) that it has worked on, another MPI all-reduce communication is necessary, which, in order to take advantage of MPI collective communications, is performed within an MPI communication group containing the same spatial domains of different partitions. The time step can then be propagated as usual on each partition, updating coordinates and velocities for the next iteration.

We present the productivity of the state decomposition scheme in Figure 3. Note that the state decomposition is a layer of parallelism on top of the MPI domain decomposition and OpenMP multithreading. We observe an appreciable speedup as we add more state partitions, with 8 partitions scaling best for the CcO model, which has on average 15 MS-EVB states in the MD run. At low node counts, however, state partitioning is not as productive as the other schemes. This is not especially surprising since adding more state partitions reduces the number of ranks in the domain decomposition, a tradeoff in computational speed. Nonetheless, the state decomposition exhibits better parallel efficiency and eventually is more productive as the node count is increased.

The new state decomposition and multithreading has certainly improved the scalability and productivity of RAPTOR by a great deal. Before, RAPTOR could hardly scale to 32 BGQ nodes, but now we are capable of scaling to multiple BGQ racks (1 rack = 1,024 nodes) with acceptable parallel efficiency.

III. REPLICA EXCHANGE UMBRELLA SAMPLING

In addition to the advances made in parallelizing RAPTOR for single trajectory MD simulations, we also have implemented a novel ensemble-run (replica exchange umbrella sampling) interface to perform enhanced sampling ensemble MD simulations.

A. Background

Umbrella sampling is a commonly used technique to compute the free energy along a chosen path. The path is usually referred to as a "collective variable" (CV) because it often involves a combination of atom coordinates, such as the distance or angles between groups of atoms. The CV is then divided into multiple "windows", and each window is given an artificial restraint (*i.e.*, a bias) to

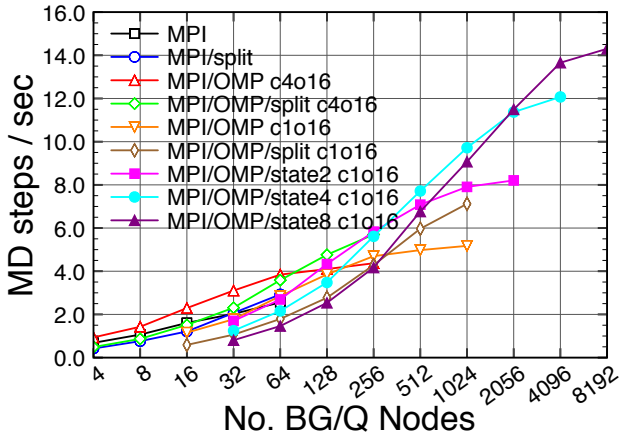


FIG. 3: Strong scaling of RAPTOR code with the CcO model with state decomposition. Results from Figure 1 and Figure 2 are included for comparison. The mode is listed for each OMP run. c4o16 = 4 MPI ranks per node and 16 threads per MPI rank (hyperthreaded mode). c1o16 = 1 MPI rank per node and 16 threads per MPI rank. The number of state partitions is listed, where "state2" is 2 state partitions, for example. Note that the data for MPI/OMP c1o16 is the equivalent of using one state partition. Each line of data is extended to the domain decomposition limit.

be applied to the molecular system of interest in order to keep the system near that window in the CV space. By doing so, one ensures good sampling of each window along the CV, which might otherwise not be the case if parts of the CV are energetically unfavorable. After sampling the windows, usually with MD runs, one uses statistics of the biased energy to compute a relative free energy value at each window, amounting to a free energy surface along the CV.

The umbrella sampling approach described above can be performed by running each window independently in separate calculations. However, it is now well-known that independent runs like this can produce artifacts and/or poorly converged free energy surfaces if the MD runs are not long enough and/or neighboring windows do not have substantial overlap in CV space. One solution to overcome this issue is to perform a loosely-coupled ensemble simulation, where all windows are run simultaneously in one big single calculation. At certain points in time, the ensemble simulation is halted, and one attempts to swap umbrella sampling restraints between neighboring windows. Swaps are accepted or rejected according to the canonical Metropolis Monte Carlo criteria to satisfy detailed balance. This approach is known as replica exchange umbrella sampling (REUS), and it has been shown to reduce artifacts and speed convergence in umbrella sampling.[7] The swapping of windows increases the sampling overlap between neighboring windows, and it helps to prevent the system from becoming "stuck" in metastable energy wells. REUS requires more computational power to run all the windows simultaneously, but it actually reduces the time to solution of a converged free energy surface as compared to completely uncoupled umbrella sampling windows, thereby actually requiring less overall CPU time.

B. LAMMPS Ensembles Implementation

In our work, we are interested in computing free energy surfaces of proton transport, and we want to use REUS enhanced sampling to speed the process. In the CcO model, we have identified a CV for proton uptake along a certain channel of the protein that leads to a catalytic complex buried within the interior of the protein. Details of this proton uptake channel will not concern us here and can be found elsewhere in the literature. For the purpose of the current work, we instead want to focus on the computational aspect.

Prior to this report, we were only able to perform the conventional uncoupled umbrella sampling with RAPTOR. We have now developed an implementation of REUS for RAPTOR, which is part of a development code by the name of LAMMPS Ensembles (LE). LE works by using `MPI_Split` to create a set of MPI subcommunicators in which each subcommunicator creates its own instance of the LAMMPS program class. Each LAMMPS instance runs MD of its own replica of the molecular system within its subcommunicator. At a specified time interval, the ensemble of replicas must synchronize in order to attempt swaps according to the REUS algorithm.

We have modified LE to perform REUS with RAPTOR. LE is yet another layer of parallelism on top of the previously described domain decomposition, state decomposition, and multithreading. A challenge unique to MS-EVB in REUS, though, is that there can exist a substantial load imbalance between replicas. Every replica contains the same molecular system but with different coordinates, and since the number of MS-EVB states depends on the spatial configuration of water molecules in our simulations, every replica has a different amount of work to compute. Fewer states means less work and will complete a fixed number of time steps in less wall time than a replica with more states. The issue is that REUS must synchronize the ensemble runs to attempt swapping, and this must be done after completing a predetermined number of MD steps. Those replicas with few states will, therefore, wait in an `MPI_Barrier` until the replica with the most number of states completes, an obvious waste of time.

To address this the load imbalance issue, we have introduced a dynamic load balance algorithm to REUS. In this scheme, we assign one of the LE replica subcommunicators to act as a listener. After completing the predetermined number of MD steps, each replica sends a message to the listener to inform the listener of being finished. The listener continues to receive messages until all replicas have reported in, at which time the listener broadcasts a signal to all other replicas that swapping

may proceed. In the meantime, while the non-listener replicas are waiting for the signal from the listener, the replicas asynchronously continue to take more MD steps, checking back for the signal after every few MD steps. The listener replica also continues to run similarly during this time. The result is that the ensemble continues to produce MD sampling while concurrently waiting to synchronize for the swapping. Ultimately, the replicas with fewer states produce several more MD steps than those with more states. But, because of the swapping in REUS, the "fast" replicas traverse the windows such that no individual window is over-sampled compared to others, which could be the case in conventional umbrella sampling.

In Figure 4, we present a weak scaling plot of using the REUS code with RAPTOR. The molecular system is still the usual CcO model from before, and we compare running REUS with different numbers of replicas using the usual synchronous static load balance versus using our asynchronous dynamic load balance, described above. Each replica in these data runs the CcO model system on 128 nodes in c1o16 mode using 2 states in the state decomposition scheme. Swaps are attempted after 200 MD steps (i.e., 200 femtoseconds). In the asynchronous dynamic load balance scheme, this is the number of steps a replica takes before sending a message to the listener replica. The full REUS run involves reported here involve 20,000 MD steps, but because replicas continue to run while waiting in the asynchronous scheme, the total number of MD steps in the end may differ. So, as the measure of performance, we present the mean productivity of the replicas in the ensemble,

$$\text{Mean productivity} = \frac{1}{tN_{\text{rep}}} \sum_{i}^{\text{replicas}} N_{i,\text{step}} \,. \qquad (3.1)$$

where $N_{i,\text{steps}}$ is the number of MD steps taken by replica i, t is the wall time, and N_{rep} is the number of replicas in the ensemble. Also, the standard deviation in the total number of MD steps taken is reported for the asynchronous scheme.

Figure 4 shows the REUS algorithm has fairly good weak scaling properties, having not dropped below half of parallel efficiency after increasing the number of replicas by 16 times both in the synchronous and asynchronous algorithms. However, the asynchronous algorithm is more productive by nearly as much as 3 MD steps/second on average at 64 replicas, a result of the improved load balance. Furthermore, the asynchronous algorithm does not make certain replicas over-sample any window too much, as the standard deviation of MD steps taken suggests.

Finally, we present strong scaling data for an REUS run encompassing the full CV of our CcO system in Table I. The CV is divided in total into 96 windows (*i.e.*, 96 replicas) running in c1o16 mode with 2 states in the state decomposition scheme and using the asynchronous dynamic load balance algorithm for REUS. Note that 24576 nodes represents half of Mira. We observe that

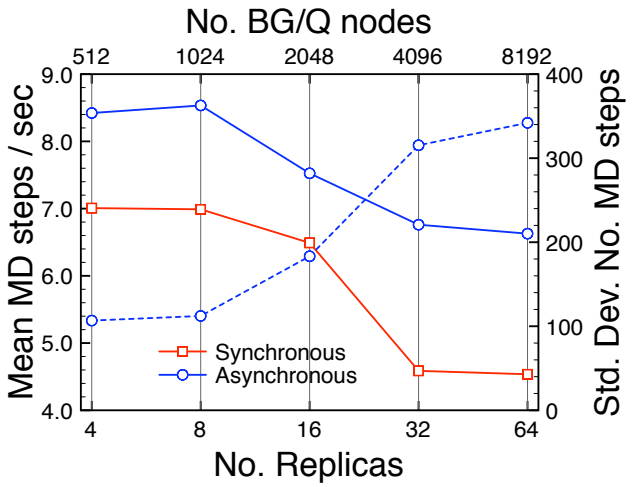


FIG. 4: Weak scaling of REUS with RAPTOR comparing the usual synchronous static load balance and the asynchronous dynamic load balance. Replicas are run on 128 nodes in c1o16 mode using 2 states in the state decomposition scheme.

TABLE I: Strong scaling of REUS with RAPTOR. 96 replicas run in c1o16 mode with 2 states in state decomposition scheme using asynchronous dynamic load balance algorithm.

No. BGQ nodes	Replica mean MD steps/sec[a]	Parallel efficiency[b]	Ensemble MD steps/sec[c]
3072	2.49	1.00	239.0
6144	4.33	0.85	415.7
12288	7.08	0.67	679.7
24576	10.43	0.50	1001.3

[a] Eq. (3.1).
[b] Relative to 3072 nodes.
[c] Eq. (3.2).

our code performs quite well in strong scaling, remaining above 50% parallel efficiency in all cases. In addition to the mean productivity, we present the ensemble productivity,

$$\text{Ensemble productivity} = \frac{1}{t} \sum_{i}^{\text{replicas}} N_{i,\text{step}} \,, \qquad (3.2)$$

which provides a measure of how much useful sampling data is being produced per second with the single calculation. That is, at 24,576 nodes (or 393,216 core processors), our REUS run is producing, on average, just over 1 picosecond total of reactive MD every wall second, a truly astounding achievement possible only on a leadership scale supercomputer like Mira.

IV. CONCLUSION

We have provided an overview of our efforts in transforming the originally poorly scaling RAPTOR code into one which can harness up to half of Mira (24 racks)

with appreciable parallel efficiency. This has been accomplished by introducing multithreading via OpenMP, developing a new state decomposition parallel algorithm, and creating a novel implementation of replica exchange umbrella sampling with the LAMMPS Ensembles code. Our new highly scalable RAPTOR code is currently being used on Mira to compute free energy surfaces of proton transport in complex biological systems at a unprecedented rate. We expect to report the results of these calculations in future work.

V. ACKNOWLEDGEMENTS

This work has been supported through the Early Science Project at the Argonne National Laboratory Leadership Computing Facility. A. W. Lange would like to extend a special thanks to Jeff Hammond at the Argonne Leadership Computing Facility as well as to Luke Westby at the University of Notre Dame for providing the development code for LAMMPS Ensembles, which has served as the foundation of our REUS code.

[1] U. W. Schmitt and G. A. Voth. The computer simulation of proton transport in water. *J. Chem. Phys.*, 111(9361), 1999.

[2] Y. Wu, H. Chen, F. Wang, F. Paesani, and G. A. Voth. An improved multistate empirical valence bond model for aqueous proton solvation and transport. *J. Phys. Chem. B*, 112:467–482, 2008.

[3] Y. Peng, C. Knight, P. Blood, L. Crosby, and G. A. Voth. Extending parallel scalability of lammps and multiscale reactive molecular dynamics. Technical report, 2012.

[4] S. Plimpton. Fast parallel algorithms for short-range molecular dynamics. *J. Comp. Phys.*, 117:1–19, 1995.

[5] R. W. Hockney and J. W. Eastwood. *Computer Simulation Using Particles*. Taylor and Francis Group, New York, 1988.

[6] T. Yamashita, Y. Peng, C. Knight, and G. A. Voth. Computationally efficient multiconfigurational reactive molecular dynamics. *J. Chem. Theory and Comput.*, 8:4863–4875, 2012.

[7] W. Jiang, Y. Luo, L. Maragliano, and B. Roux. Calculation of free energy landscape in multi-dimensions with hamiltonian-exchange umbrella sampling on petascale supercomputer. *J. Chem. Theory and Comput.*, 8:4672–4680, 2012.

Argonne Leadership Computing Facility
Argonne National Laboratory
9700 South Cass Avenue, Bldg. 240
Argonne, IL 60439

www.anl.gov