FORSCHUNGSZENTRUM JÜLICH GmbH
Jülich Supercomputing Centre
D-52425 Jülich, Tel. (02461) 61-6402

Technical Report

Jülich Blue Gene/P
Extreme Scaling Workshop 2009

Bernd Mohr, Wolfgang Frings (Eds.)

FZJ-JSC-IB-2010-02

February 2010
(last change: 23-Feb-2010)

Jülich Blue Gene/P Extreme Scaling Workshop 2009

Bernd Mohr, Jülich Supercomputing Centre
Wolfgang Frings, Jülich Supercomputing Centre

Executive Summary

From 26 to 28 October, JSC organized the 2009 edition of its Blue Gene Scaling Workshop. This time, the main focus were application codes able to scale-up during the workshop to the full Blue Gene/P system JUGENE which consists of 72 racks with a total of 294,912 cores – the highest number of cores worldwide available in a single system.

Interested application teams had to submit short proposals which were evaluated with respect to the required extreme scaling, application-related constraints which had to be fulfilled by the JUGENE software infrastructure and the scientific impact that the codes could produce. Surprisingly, not only a handful of proposals were submitted, as the organizers had expected, but ten high-quality applications could be selected, among them two 2009 Gordon Bell Prize finalists. The selected teams were[1]:

- Scaling BAGEL QCD
  P. Boyle
  Univ. of Edinburgh, UK

- Gyrokinetic Turbulence Simulation with GENE
  T. Dannert, T. Görler, F. Jenko, F. Merz
  RZG and IPP, Garching, Germany

- Scalable Neutron Transport Simulations on Petascale Architectures
  D. Kaushik, M. Smith, A. Wollaber, B. Smith, A. Siegel, W.S. Yang
  Argonne National Laboratory, USA

- Osiris strong scaling on full Jugene
  M. Marti, S.F. Martins, R. Fonseca, L.O. Silva
  Instituto de Plasmas e Fusão Nuclear, Instituto Superior Técnico, Portugal

- Full scale simulation of coronary arteries in presence of red blood cells
  S. Melchionna
  Swiss Federal Institute of Technology, Switzerland

[1]Participants to the workshop marked in **bold**

- PEBS: Parallel Evolutionary Biology Suite
  A. Peters, D. Rand, M. Nowak, J. Sircar
  Harvard University, USA

- Adaptive computational fluid dynamics at extreme scale
  O. Sahni, M. Zhou, **M. Rasquin**, M.S. Shephard, K.E. Jansen
  RPI, Troy, USA

- Extending Scalability of MP^2C to more than 250k Compute Cores
  G. Sutmann, W. Frings
  Jülich Supercomputing Centre, Germany

- Budapest-Marseille-Wuppertal HMC Collaboration
  K. Szabo, S. Krieg
  Univ. Wuppertal and MIT, Germany and USA

- Scaling of the tmLQCD software suite on Jugene
  A. Deuzeman, S. Reker, G. Herdoiza, K. Jansen, **P. Gerhold, C. Urbach**
  Univ. of Groningen, NIC/DESY, Humboldt-University Berlin, Bonn Univ.
  The Netherlands and Germany

During the workshop, the teams were supported by four JSC parallel application experts, the JUGENE system administrators and one IBM MPI expert; however, the participants shared a lot of expertise and knowledge, too. Every participating team succeeded to run one or more successful full 72 rack jobs during the course of the workshop, and executing many smaller jobs to investigate the scaling properties of their applications along the way. A total of 398 jobs were launched using 13.3 million core hours of the total 16.6 million core hours reserved for the workshop. This is an 80% utilization which is astonishing especially considering that the system was only partially available on the third day due to necessary hardware repairs. IBM experts repaired half of the machine at a time while the other half was still used for the workshop.

Achieved Results

Argonne National Laboratory

The application team from Argonne National Laboratory represented by Dinesh Kaushik experimented with UNIC, a code for Neutron Transport Simulations. The objective of this work is to perform simulations of fast reactor cores with detailed resolution of geometric heterogeneity and validate them against experiments. Accurate simulation tools like UNIC will lower the uncertainties and biases in rector design calculations, facilitating the development of nuclear reactors that are safe, affordable, and environment friendly. Scaling this code means to overcome several challenges: The seven dimensional neutron transport equation easily exhausts the memory available on today's largest machines. The performance is memory bandwidth-limited due to sparse matrix vector (BLAS

#cores	#angles	total time [s]	weak scaling
32,768	32	579	100%
73,728	72	572	101%
131,072	128	581	100%
163,840	160	691	84%
294,912	288	763	76%

Table 1: UNIC scalability results on Blue Gene/P. Mesh with 7 million vertices and 1.7 million quadratic hexahedral elements; 9 energy groups; 18 billion degrees of freedom.

Level 2) operations. So the algorithms used put an emphasis on saving total and per-core memory. The primary kernel is conjugent gradient solver with an SSOR (Symmetric Sucessive OverRelaxation) precondintioner; ICC (InComplete Cholesky) factorization is not affordable as preconditioner. For the workshop, highly-detailed simulations of the Zero Power Reactor 6 (ZPR6) Experiment 6A on up to the full machine (294,912 cores) were performed. Scaling to these dimensions allowed the team to resolve the full reactor geometry for the first time and compare the results directly with the experimental data. Some scaling results are shown in Table 1.

Budapest-Marseille-Wuppertal

The Budapest-Marseille-Wuppertal collaboration HMC team simulated Lattice Quantum Chromodynamics with dynamical fermions also up to the full machine. The good scaling is achieved by special optimizations:

- The communication library uses the low level SPI software interface. This thin layer contains wrapper functions for compute node kernel calls and allows direct hardware access for user space applications. It contains all required communication routines, such as global sums or broadcasts, persistent or generic communication routines as well as global and node barriers to synchronize all cores or the cores of a single node. With the SPI it is possible to make best use of the hardware capabilities avoiding latencies from high level libraries such as DCMF (Deep Computing Messaging Framework) or MPI. Especially important for efficient communication routines is the ability to directly control the Blue Gene/P's DMA. By explicitly injecting message descriptors into a dedicated injection FIFO and by resetting the FIFO head it is possible to implement persistent communications, which cuts the time required to start the communication process down to a minimum. This functionality is not accessible through DCMF or MPI. Additionally, this way all communication inside the kernels perfectly overlaps with the computations. The performance of the serial Clover operator for instance is virtually unchanged when the communication is switched on.

- In order to be able to optimize large fractions of the code, a set of macros was used that implements - via the built-in functions - the basic mathematical operations that are required, such as SU (3) matrix matrix or matrix vector multiplications or spinor manipulation routines. Using these macros, all remaining compute intense parts of the simulation code could be efficiently optimized. Combined with extensive manual reordering of loops to deal with the limited instruction flow manipulation capabilities of the Blue Gene/P's PPC450 core, almost always a speed up of at least a factor of 3 over the plain C code could be achieved.
- As with any software, when the number parallel threads is to be increased well beyond the 100K threshold, one has to take special care to remove potential scaling hazards, which are likely to be found in any scalar part of the code. The code was throughly checked for such problems, and, if a routine was found that could cause problems, it was modified accordingly. I/O was also optimized, which is especially important when running on the full 294,912 cores.

During the workshop, the team performed a strong scaling analysis with a fixed problem size from 2 to 72 racks, as for QCD weak scaling is trivial. The scaling is almost perfect to the full scale; the kernel (dslash + clover) achieves about 33% efficiency on 72 racks.

Other results

The following chapters give an account on more detailed execution and scaling results achieved by some of the application codes during the workshop provided by the participants themselves.

Acknowledgements

We would like to thank IBM Germany for their support of the workshop.

Gyrokinetic Turbulence Simulation with GENE

T. Dannert[1], T. Görler[2], F. Jenko[2], F. Merz[2]

[1] Rechenzentrum Garching der Max-Planck Gesellschaft,
Boltzmannstr. 2, 85748 Garching

[2] Max-Planck Institut für Plasmaphysik,
Boltzmannstr. 2, 85748 Garching

1 Description of the Code

The Gyrokinetic Electromagnetic Numerical Experiment (GENE) [1, 2, 3, 4] is an Eulerian Code which solves the nonlinear gyrokinetic Vlasov-Maxwell system of equations on a five dimensional fixed grid. These equations describe micro-turbulence in a hot magnetized plasma, which is relevant for magnetically confined fusion experiments. The GENE code has been developed over many years at the Max-Planck Institute for Plasmaphysics and is still being extended to include more physics and to optimize the numerical schemes and algorithms. It is of high relevance for European fusion research, is freely available and has a wide user base worldwide.

A field-aligned coordinate system is employed to exploit the strong anisotropy of turbulent fluctuations along and perpendicular to magnetic field lines [5, 6]. Two different numerical approaches have to be distinguished. In the *local* approach both perpendicular directions x and y (radial and binormal coordinates) are treated in a pseudo-spectral way, whereas in *global* mode, the spectral approach is used only for the y direction. The first mode is used for flux-tube simulations, where all background quantities (like density, temperature) together with their gradients are taken as constant in the simulation volume (especially in the radial x direction). In the global case this restriction is relaxed and radially varying profiles of density and temperature together with radially dependent geometric coefficients are used.

The *global* code is parallelized in all five dimensions. For the *local* mode, no domain decomposition is applied in the radial direction so that only a 4D grid is parallelized. In both modes the code parallelizes trivially over the number of species (usually a factor of 2-4).

There are three different physically important types of parameter sets.

1. Multiscale problems [7]. Here the coupling of electron and ions scales in gyrokinetic turbulence are investigated. This can be done in the local approach, leading to intensive use of Fourier transforms. The parallelization

in this case is done over velocity space, z direction and in a big part over k_y direction. Fourier transformations need to be applied in both perpendicular directions to allow for an efficient computation of the nonlinearity. They are accompanied by a transpose operation which requires a significant all_to_all communication along the k_y direction. Typical grids have $1024 \times 512 \times 24 \times 48 \times 16$ points for two species and a memory inprint of roughly 2.5TB.

2. Stellarator problems [8]. The complex geometry of stellarators causes strong variations of corresponding quantities along the magnetic field lines. Hence, more parallel and velocity space grid points are required in order to resolve the turbulence features. Typical grid is $128 \times 64 \times 512 \times 64 \times 16$, again for two species. This case leads to a memory need of roughly 2TB. Also the parallelization needs are different. While still running in the local approach, the exchange of ghost cells in z direction has a larger impact than in the first case. The Fourier transforms are less important.

3. Global simulation. A third application of the code is the simulation of plasma turbulence over a larger radial domain with radially varying background quantities with physical boundary conditions. For this purpose, Fourier transforms cannot be applied in the radial direction and hence finite element techniques in direct space are employed to perform, e.g., the gyro-averaging operation. As the number of grid points in this direction can grow to a few thousand, a parallelization is established which can be used for some processes. A projected grid for future investigations is $8192 \times 64 \times 32 \times 128 \times 64$ for up to four species. This leads to a memory need of roughly 100TB.

2 Accomplished Objectives

Multiscale parameters We were able to run the multiscale problem on 4, 8, 16, 32 and 72 racks in SMT mode. We used a strong scaling approach, increasing the number of cores while keeping constant the problem size at the parameters described in sec. 1. This leads to a small problem size per core in the full machine run. The memory usage was there only 10.5MB per core. The time per timestep for these runs is shown in Fig. 1. As one can see, the time is decreasing up to 32 racks, while the parallel efficiency is unfortunately also decreasing to 48%. The full machine run with 72 racks did not show a further decrease of the time per timestep. For finding the cause of this unfortunate scaling, we analyzed the GENE code with the help of the Scalasca tool on 4096 cores. There we found that 37% of the runtime was spent in communication, which is expected to increase for larger core counts. The reason is the heavy use of MPI_Alltoall calls in the computation of the nonlinearity. To also scale to 72 racks, it is expected that we have to increase the problem size.

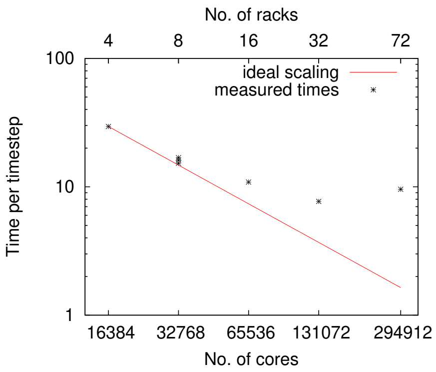

Figure 1: Strong scaling on Jugene for multiscale parameters.

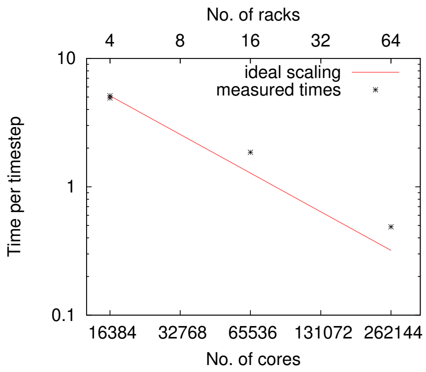

Figure 2: Strong scaling on Jugene for stellarator parameters.

Stellarator parameters For the stellarator problem, runs with 4, 32 and 64 racks were possible in SMT mode. We also kept the problem size constant while increasing the number of cores. The time per timestep results can be found in Fig. 2, where one finds a good scalability of the code in this parameter case. The parallel efficiency from 4 to 64 racks was 65%. Due to the strong scaling the problem uses just 5.5 MB memory per core in the 64 rack case. But although the problem size per core is this small, we can achieve this still remarkably high scaling.

Global simulation For the global simulation, we found that the memory usage of the code does not scale, which is due to a matrix in the first dimension. To scale the global version of the code to larger processor counts, we have to overcome this memory issue.

3 Conclusion

The workshop confirmed and extended the scaling experiences which were achieved on the BlueGene/P system of IDRIS, France in 2008 [9].

We were able to run the code on the Jülich supercomputer and could achieve good scaling results in a strong scaling approach. The problems in the local approach (multiscale and stellarator parameters) were physically meaningful but in one case too small for the full machine. The global approach of the code, which has been developed recently, is not yet able to scale to a larger core count. Current work focuses on these scaling obstacles, whose removal is in principle clear. The global case will then require much more memory than the local ones. Since also for the global code we assume close to ideal weak scaling, good scaling up to the full machine of 72 racks (294912 cores) can be expected for those physically relevant cases.

References

[1] F. Jenko, W. Dorland, M. Kotschenreuther, and B. N. Rogers. *Phys. of Plasmas*, 7:1904, 2000.

[2] T. Dannert and F. Jenko. *Phys. Plasmas*, 12:072309, 2005.

[3] F. Merz and F. Jenko. *Phys. Rev. Lett.*, 100(035005), 2008.

[4] F Jenko and The GENE development team. The GENE code, jan 2009. `http://www.ipp.mpg.de/~fsj/gene`.

[5] J. W. Connor, R. J. Hastie, and J. B. Taylor. *Phys. Rev. Lett.*, 40:396, 1978.

[6] P. Xanthopoulos and F. Jenko. *Physics of Plasmas*, 13(092301), 2006.

[7] T. Görler and F. Jenko. *Phys. Rev. Lett.*, 100(185002), 2008.

[8] P. Xanthopoulos, F. Merz, T. Görler, and F. Jenko. *Phys. Rev. Lett.*, 99(035002), 2007.

[9] H. Lederer, R. Tisma, R. Hatzky, A. Bottino, and F. Jenko. Application enabling in DEISA: Petascaling of plasma turbulence codes. In C. Bischof et al., editor, *Parallel Computing: Architectures, Algorithms and Applications*, volume 15 of *Advances in Parallel Computing*, page 713. IOS press, 2008.

Osiris strong scaling on full Jugene

Michael Marti Samuel F. Martins Ricardo Fonseca
Luis O. Silva
Instituto de Plasmas e Fusão Nuclear, Instituto Superior Técnico, Portugal

1 Introduction

For the Extreme Scaling Workshop 2009 our team - representing the Group of Lasers and Plasmas of Instituto de Plasmas e Fuso Nuclear, Instituto Superior Tcnico, Portugal - aimed to run the particle-in-cell code OSIRIS on full Jugene (294,912 cpu cores). This report presents the proposed physics case, the steps necessary to get our code running on the B/G architecture, the outcome of the scaling tests on full Jugene, and the finally conclusions and future work.

2 Proposed problem to run at the WS

Several astrophysical scenarios lead to extreme physical regimes, typically observed on Earth in the form of radiation and cosmic rays. These regimes encompass a set of phenomena such as magnetic field generation, shock formation, and non-thermal particle acceleration (for a review, see [1]) associated with relativistic astrophysical shocks. In particular, the acceleration of electrons, positrons, or ions in shocks in active galactic nuclei, gamma-ray bursts, pulsar wind nebulae, and supernova remnants leads to energetic particles, the seed to cosmic rays (CRs), that can then scatter in magnetic and electric fields and emit synchrotron radiation. Despite their critical relevance to interpret the radiation from astronomical observations, the underlying processes inherent to the CR acceleration are not yet fully understood. Simulations play a critical role in the assessment of the physical mechanisms relevant for astrophysical shocks, e.g. in the study of the Weibel instability [2, 3] magnetic field generation [4, 5, 6], and particle acceleration in the precursor region [7]. The self-consistent ab initio kinetic modeling of a relativistic astrophysical shock is computationally very demanding, as large temporal and spatial scales push the numerical techniques to conditions still unexplored requiring extreme computational resources, optimized algorithms, methods for improved energy conservation, and advanced visualization diagnostics.

A full self-consistent simulation of a three-dimensional electron-proton shock, however, is extremely demanding with existing infrastructures (108 GB of memory would be needed to store all the particle information). Therefore, state-of-the-art simulations are done in two-dimensions. These two-dimensional studies already enabled a first step in the study of astrophysical shocks, confirming ab initio CRs Fermi acceleration [8, 9], but a full understanding of the microphysics can only be obtained with three-dimensional simulations.

We propose to use Jugene to perform these massive three-dimensional simulations with a mass ratio up to 32 (100TB of memory required), and thus obtain a scientific breakthrough in the understanding of the complex three-dimensional nonlinear phenomena involved in shocks, with impact in many astrophysical scenarios.

3 OSIRIS

To address this challenge we propose to use OSIRIS [10] a state-of-the-art, fully explicit, fully parallelized, fully relativistic, electromagnetic, and fully object-oriented Particle-in-Cell (PIC) code, written in Fortran 90, using MPI and HDF 5. The code contains algorithms for 1D, 2D, and 3D simulations in Cartesian coordinates and for 2D simulations in azimuthally symmetric cylindrical coordinates. PIC codes model plasmas as particles interacting self-consistently via the electromagnetic fields they themselves produce. These models work at the most fundamental, microscopic level. As a result, they are the most compute intensive model in plasma physics. OSIRIS has been in production since 1999 and it has been deployed in many HPC systems, with many diverse architectures/configurations/interconnects/IO systems, in the US and in Europe, from small clusters to very large systems, including Jugene. It was designed from scratch for production in massively parallel systems. Scalability to 100 k CPUs poses additional challenges to parallel programs and OSIRIS has evolved accordingly. The code has been fitted with a dynamic load balancing module, ensuring the best possible distribution of computational load across nodes, in terms of both particle and grid calculations, and excellent scalability to very large numbers of cpus. Running on these systems poses also significant challenges in terms of data output. OSIRIS now uses parallel I/O, through the HDF5 library, for saving particle and grid data, and achieves a throughput of up to 800 MB/s, allowing for efficient output of simulation data. Even with parallel I/O, deployment to Jugene and other BlueGene machines has required special attention in terms of postprocessing and diagnostics. Saving detailed time evolution of the full simulation data in Jugene would result in prohibitive storage needs, so data reduction algorithms for diagnostics have been implemented into OSIRIS. For grids, it is possible to output a spatial average/ envelope of the grid data, greatly reducing the global grid size, and for particles OSIRIS can generate gridded distribution functions (i.e. phasespaces) of the global particle data. Both algorithms have been designed to operate in situations where the final dataset does not fit in a single parallel node, and allow for arbitrary size diagnostics in memory starved environments.

4 Work done to get OSIRIS ready for the WS

Since the initial conception, OSIRIS was always laid out to run on high performance computers. Several modifications and extensions to the code where necessary throughout time, to make efficient use of ever faster and larger computer systems. This includes MPI message packaging, parallel i/o, and dynamic load balancing. For the B/G architecture in particular, two major changes to the code where necessary. First, the numerical precision for the different parts of the PIC algorithm is now parametrized, and can be chosen at compile-time. In essence this means that we can individually chose the floating point precision used for field quantities (electric and magnetic fields and currents fields) as well as for particle quantities (position, momentum, and charge). For the particles it is furthermore possible to select wether the position should be stored relative to the simulation box or relative to the respective grid cell (although single precision positions always require indexing to the grid cell). This allows us - in accordance with the particular physical problem to be solved - to make more efficient use of the available memory. Second, in order to ensure the correct mapping between the decomposition of the simulation domain and the MPI nodes in the torus network, we rewrote the setup of the node topology in OSIRIS using the B/G specific MPI function MPIX_Cart_comm_create [11]. In this approach the simulation topology is always mapped to the particular B/G of the current job. Consequently it is no longer necessary to request a specific B/G partition upon job submission, which reduces job queuetime and increases overall B/G occupation.

5 Workshop

5.1 Work done at the WS

We have performed strong scaling benchmarks of OSIRIS up to the full machine, for two distinct physical regimes with different communication and load balance characteristics: (i) simulation of a three-dimensional region with charged particles initialized with a thermal distribution and zero bulk velocity and (ii) simulation of a three-dimensional region with two charged particle species initialized with a thermal distribution and with opposite relativistic drifts . The first regime is a best-case scenario with simple particle dynamics, and therefore a more homogeneous load balance across processors along the entire simulation. The second regime is a realistic scenario that leads to the generation of a plasma instability characterized by the formation of strong density fluctuations and current filaments, which contribute to an uneven computational load and to an increased communication stressing.

5.2 Strong scaling results with OSIRIS

The benchmarks went up to the 294,912 cores, starting with a baseline of 4,096 cores. The scaling efficiencies for the two regimes tested were 81% and 72% for

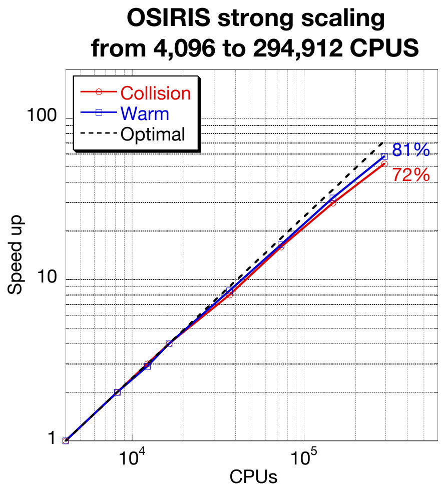

Figure 1: OSIRIS strong scaling from 4'096 to 294'912 CPUS

the best-case and realistic scenarios, respectively (Fig. 1). The small difference between the two regimes emphasizes the relevance of the fast interconnecting network, which results in a small communication overhead.

5.3 Production simulation

Our initial plan for the WS included a possible production simulation of an astrophysical scenario. This would constitute the largest simulation ever performed of a relativistic collisionless shock. The numerical experiment, however, would require a very large allocation of CPU.hours, which were not possible to make available during the WS, since the main priority was to allow users to perform faster tests in the largest number of cores available. We plan to perform these simulations with our 2010 allocation.

5.4 Difficulties encountered

Despite the high efficiency obtained in our strong scaling benchmarks, we have still identified a couple of improvements to the code, mostly related to the large number of CPU's.

For instance, we where not aware that booting the partition does have to be added to the wall-time of the first job running in the new partition. This caused one of our benchmark jobs to exit before it was able to write out the timing measurements, which made that particular run useless. Subsequently we would add an extra 30 minutes to the requested wall-clock time for large jobs. We also discovered some problems with precision of the floating point arithmetics in single precision. These issues were caused by the XL compilers altering semantics of the code. We were not able to reproduce the problem with any other compilers (on different machines), including g95 and Portland. Using the XL compilers on AIX/Power 5 we were able to reproduce part of the problem, and finally solved these issues by rewriting part of the single precision algorithms.

Finally, our code uses local files for check-pointing and to write out the timing results. This causes an overhead of about 30 min per metadata operation. For the sake of benchmarks, we circumvented this problem by disabling check-pointing and only writing timing information on selected nodes. We will address this issue in the future.

6 Conclusions and Future work

We have performed strong scaling benchmarks of the full particle-in-cell code OSIRIS up to the 294,912 Jugene cores. Two test cases were used, for a best-case scenario and a realistic regime of a three-dimensional plasma region. The efficiency at the full machine was 81% for a baseline of 4,096 cores.

Several test-cases as well as further code development are planned in the near future, with the aim to improve overall efficiency of osiris and to overcome some of the difficulties mentioned above.

Foremost OSIRIS's checkpoint functions will be rewritten in order to encapsulate al i/o calls in a single function. This will allow us to easily replace the underlying i/o mechanism. For B/G we will be using sionlib [12], which will reduce the overhead for metadata operations (essentially completed as of december 2009). For the output of timing files it is planned to use mpi function to collect all timing data and only write the output file on node 0 - hence avoiding process local files for timing diagnostics. Furthermore it is planned to run a series of small test jobs in order to test several compiler options, runtime environment switches and to measure and investigate the memory footprint of our code.

References

[1] F. C. Jones, D. C. Ellison, Space Sci. Rev., 58, 259 (1991)

[2] E. S. Weibel, Phys. Rev. Lett., 2, 83 (1959)

[3] M. V. Medvedev, A. Loeb r:x306, ApJ, 526, 697 (1999)

[4] L. O. Silva et al., ApJ, 596, L121 (2003)

[5] J. T. Frederiksen et al., ApJ, 608, L13 (2004)

[6] K. I. Nishikawa et al., ApJ, 622, 927 (2005)

[7] L. O. Silva, AIP Conf. Proc. 856, 109, astro-ph/0610345 (2006)

[8] A. Spitkovsky, ApJL, 682, L5 (2008)

[9] S.F. Martins et al., ApJL, 695, L189-L193 (2009).

[10] R. Fonseca et al., Lecture Notes in Computer Science (2331, p. 342 Springer, Heidelberg, 2002)

[11] IBM System Blue Gene Solution: Blue Gene/P Application Development, http://www.redbooks.ibm.com/abstracts/sg247287.html.

[12] SIONlib: Scalable I/O library for parallel access to task-local files, http://www.fz-juelich.de/jsc/sionlib/.

PEBS: Parallel Evolutionary Biology Suite

Amanda Peters, Harvard University
David Rand, Harvard University
Martin Nowak, Harvard University
Joy Sircar, Harvard University

1 PEBS

The scientic aim of this work is to better understand the evolution of cooperation. This has become possible through the Parallel Evolutionary Biology Suite (PEBS) which is a code package targeting an important algorithm for the simulation of evolutionary game dynamics. Natural selection dictates survival of the ttest and is thought to promote the emergence of selsh strategies over altruistic ones. However, we can empirically see many examples of apparent cooperative behavior in biology, from the interactions of humans to viruses to bacteria. PEBS was developed to study the emergence of cooperative behavior in the context of specic behavioral strategies. In scaling this package to the Juelich Blue Gene system, we hope to address the following two questions:

- Do strategy proles based on longer histories have a stronger predilection for cooperation to evolve?
- Are human social networks designed to promote cooperation?

Current studies look at iterative interactions of behaviors in which moves are based on the action in the round immediately prior. A strong complaint of these models is the lack of historical memory, however, expanding strategies to account for larger historical proles increases the memory and computational requirements exponentially. Current work has not been able to investigate behavioral strategies conditioned on more than one prior move due to the exponential increase of the number of potential strategies as more moves are included. An exhaustive study of these strategy spaces necessitates the use of a system of the size of the Juelich Blue Gene/P. Determining the impact of historical memory on the appearance of cooperation would have implications in elds ranging from biology to economics to traditional game theory.

Social networks have been shown to play an extremely important role in human interactions generally, and particularly in the evolution of cooperation. It is possible for cooperation to be favored by selection when individuals only interact with their neighbors, given the right interaction topology. Previous research has explored the evolution of cooperation on regular and random graphs of different types, but no one has yet studied evolutionary dynamics on empirically derived human social networks. We want to extend previous research by asking whether human social networks are designed in such a way as to promote cooperation. Real network data is presently available, but involves large numbers of agents and thus signicant computational power to investigate using agent based simulations.

2 Workshop Results

PEBS is an evolutionary dynamics package developed at Harvard University specically for large scale computing. The overall program was written in C and parallelized with MPI. The model consists of a population of two or more behavioral strategies. The parallel implementation allows the contest between each individual strategy to be fully played out. After each round the relative tness of each is assessed and certain members of the population will convert to the ttest strategy. The simulation is continued until the population reaches a steady state. This allows for an evolutionary stable strategy to potentially emerge and to determine if the strategy pairs favor altruistic behavior. Current test cases focus on strategies based on one the moves made in the previous round and assume a well-dispersed population. The goal for this workshop was to scale the code to the full system to allow a more exhaustive search of the strategy space.

- Grew the space from strategies looking at one historical step to those looking at five.
- Able to grow the population size by an order of magnitude.
- Successfully scaled to the full 72 racks.
- Achieved 77% efficiency for weak scaling.

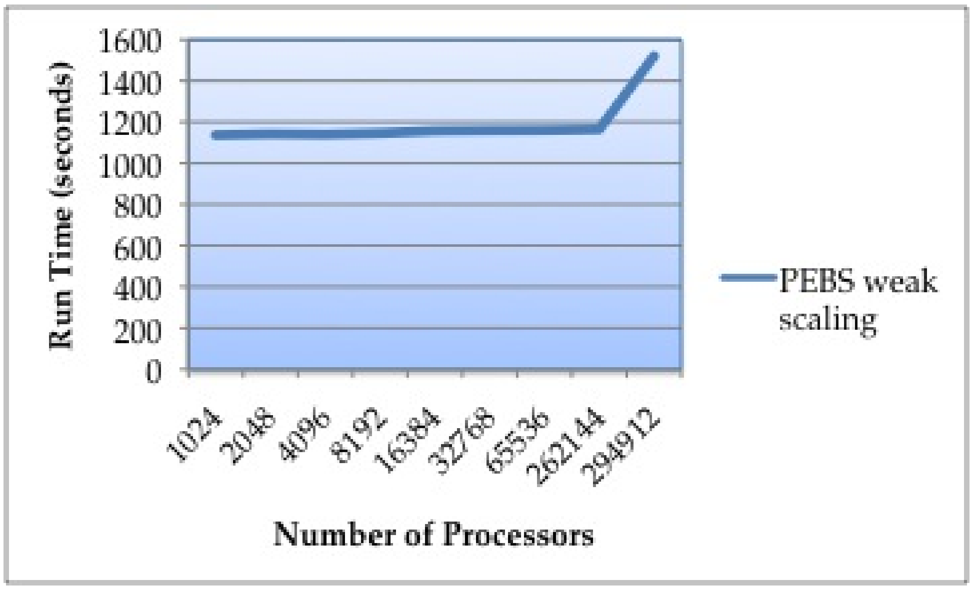

Figure 1. Scaling on Jugene

Attending the workshop had significant impact on our work. At Harvard, we only have a 2 rack BG/L system so this provided the opportunity to scale the application much farther than previously seen. This helped to highlight segments of the code that had a much higher impact on code performance than initially though. One example of this was the method employed for random number generation. The need for use of a steamlined library became apparent even at 4 racks. Furthermore, the work at this conference demonstrated an issue with the current communication patterns. For this application, multiple nodes are given the same base strategy for the simulation. As the population and system size grow, more nodes handle games involving the same strategy and are therefore involved in a more localized analysis at the end of each round. At small system sizes, this had little impact on performance, but as we approached sizes larger than 16 racks, communication increased pretty dramatically. This led to a degradation in performance as shown in the scaling plot above. Portions of the code had a much bigger impact than realized such as the method of random number generation. Participation in this workshop not only allowed us to make more scientifically meaningful runs but to identify this issue with our code. From here we are focusing on optimizing the communication involved

Adaptive computational fluid dynamics at extreme scale

O. Sahni M. Zhou M. Rasquin M.S. Shephard
K.E. Jansen
CII-4011, SCOREC, RPI, 110 8th St., Troy 12180 NY, USA

1 Description of the Code

Overview: The goal of this project is to develop and deploy adaptive computational fluid dynamics at extreme scale. In this project a mature implicit flow solver is being used along with anisotropic adaptive meshing to attack flow problems where solution features such as boundary layers can only be located and resolved through adaptivity. The current solver employs a stabilized finite element method [3] along with a generalized-α (implicit) time integration scheme [1]; which results in two primary work components, i.e., equation-formation and equation-solution. For both of the work components a single consistent partition based on mesh elements is used (requiring no redistribution of data; avoiding significant scaling issues that can arise at high core counts). For details please refer to [2, 4].

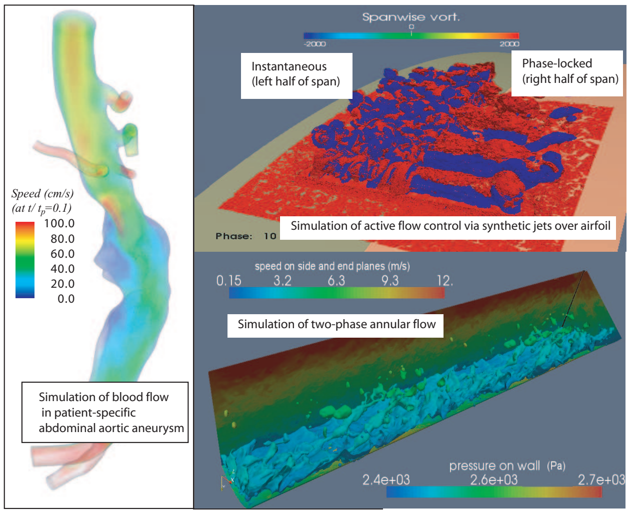

Figure 1: Applications: aerodynamic, cardiovascular and two-phase flows.

Applications of interest: Applications include aerodynamic, cardiovascular and two-phase flows. Figure 1 demonstrates the capabilities of our flow solver on three major flow problems: (a) blood flow simulation in a patient-specific abdominal aortic aneurysm (AAA) where diseased vascular geometry triggers massive separation and transitional flow, (b) simulation of active flow control via synthetic jets for virtual "aero-shaping" of wings and blades in order to improve the performance and to alleviate unsteady aerodynamic loads, and (c) direct numerical simulation of fully-developed two-phase annular flow to predict and prevent catastrophic thermal-hydraulic burnout.

2 Accomplished Objectives

During this workshop we were able to perform strong scaling studies, up to 294,912 cores of IBM BG/P system (i.e., full system scale of JUGENE). Example of AAA model was considered [2]; two meshes with roughly 1B and 5B elements (where B denotes billion) were used. Furthermore, effect of partition improvement algorithm (LIIPBMod) developed in [4] was also studied by considering different partitions.

AAA model with about 1B elements: This test case considers a 1B element mesh of an AAA model. Partitions with different number of parts ranging from 4,096 to 294,912 are created. The element imbalance is within 6% but the worst vertex imbalance is 43.7% in partition with 294,912 parts that is created based on a 4,096 part partition. An imbalance of 43.7% indicates proportionally more work on the corresponding part during the equation solution phase. LIIPBMod algorithm reduces the vertex imbalance dramatically to 17% while increasing the element imbalance to 15%. For another 294,912 part partition which is based on 16,384 part partition, the quality is slightly better, with 5.1% element imbalance and 22% vertex imbalance; where LIIPBMod algorithm reduces the vertex imbalance down to 4.9% while increasing the element imbalance to 12.7%. Table 1 presents scaling factors along with the time usage; where time compression of about 60x is shown (parallel efficiency of 82%).

1.07B element mesh	eqn. form.		eqn. soln.		total	
num. of cores	time	s_factor	time	s_factor	time	s_factor
4,096 (base)	390.17	1	455.51	1	845.68	1
4,096 to 294,912 (*LMod*)	5.82	0.93	9.14	0.69	14.96	0.79
4,096 to 294,912 (no *LMod*)	5.54	0.98	10.38	0.61	15.92	0.74
16,384 to 294,912 (*LMod*)	5.78	0.94	8.60	0.74	14.39	0.82
16,384 to 294,912 (no *LMod*)	5.54	0.98	9.21	0.69	14.75	0.80

Table 1: Strong scaling results on an AAA model with about 1B elements; with and without LIIPBMod algorithm. *LMod* denotes LIIPBMod.

AAA model with about 5B elements: This test case considers a 5B element mesh of the same AAA model. As in the previous example, the elements are balanced within 6%, however the worst vertex imbalance is 18.8% in 294,912 part partition. LIIPBMod algorithm reduces the vertex imbalance to 10.7% with modest impact on element imbalance. Table 2 presents scaling factors along with the time usage; where overall strong-scaling of 95% is shown.

5.07*B* element mesh	eqn. form.		eqn. soln.		total	
num. of cores	time	s_factor	time	s_factor	time	s_factor
65,536 (base)	119.64	1	162.59	1	282.23	1
131,072	59.69	1.00	84.09	0.97	143.78	0.98
262,144	30.02	1.00	43.24	0.94	73.26	0.96
294,912	26.71	1.00	39.15	0.92	65.86	0.95
294,912 (no *LMod*)	26.28	1.01	44.23	0.82	70.51	0.89

Table 2: Strong scaling results on an AAA model with about 5B elements; with and without LIIPBMod algorithm. *LMod* denotes LIIPBMod.

3 Summary

Participation in the workshop allowed us to demonstrate strong scalability of our flow solver up to 294,912 cores (i.e., full system scale of JUGENE); with overall strong-scaling up to 95% on 294,912 cores. Note that our flow solver is based on implicit methods using unstructured, anisotropically adapted meshes that allow for an efficient solution strategy for many real-world problems (e.g., simulation-based virtual surgical planning). Scalable solvers employing these methods not only enable solution of extremely-large practical problems but also lead to dramatic compression in time-to-solution.

References

[1] 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. *Comp. Meth. Appl. Mech. Engng.*, 190:305–319, 1999.

[2] 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. In *Proceedings of IEEE/ACM SC '09*, Portland, OR, USA, Nov., 2009. Finalist paper for the Gordon Bell Prize 2009.

[3] C. H. Whiting and K. E. Jansen. A stabilized finite element method for the incompressible Navier-Stokes equations using a hierarchical basis. *Int. J. Numer. Meth. Fluids*, 35:93–116, 2001.

[4] M. Zhou, O. Sahni, K.D. Devine, M.S. Shephard, and K.E. Jansen. Controlling unstructured mesh partitions for massively parallel simulations. *SIAM Scientific Computing*, 2009. under review.

Extending Scalability of MP^{2}C to more than 250k Compute Cores

Godehard Sutmann, JSC Wolfgang Frings, JSC

1 Description of the Code

Mesoscale simulations of hydrodynamic media have attracted great interest during the last years in order to bridge the gap between mircoscopic simulations on the atomistic level on the one side and macroscopic calculations on the continuum level on the other side. Various methods have been proposed which all have in common that they solve the linearized Navier-Stokes equations in different types of discretizations, e.g. Lattice-Boltzmann simulations [1, 2] on a spatial grid or Multi-Particle Collision Dynamics (MPC) — also called Stochastic Rotation Dynamics (SRD) — using discrete particles [3, 4]. In the latter approach, pseudo-particles are considered to carry both hydrodynamic information and thermal noise. Specifying a small set of parameters, i.e. fluid particle density, scattering angle, mean free path of a particle) it is possible to reproduce a variety of hydrodynamic phenomena. In particular, the regime of small Reynolds numbers has been investigated in detail, e.g. Poiseuille flow, shear flow, vortices or hydrodynamic long time tails, to name a few [4, 5]. The method itself is relatively simple and consists basically of two steps: (i) a streaming step, where particles are propagated in space according to their actual velocity and the applied time step; (ii) a collision or momentum transfer step, where, for the case of SRD, the velocity relative to the center-of-mass velocity inside a locally defined collision cell is rotated by a given angle α around a stochastically chosen axis.

One advantage of the MPC method is that a coupling to atomistic simulations can be established in a simple way. The main characteristic of MPC is that particles are sorted into the cubic cells (with lattice constant a) of a randomly positioned collision grid. Different variants of MPC vary in the way how the momentum exchange between particles within a collision cell is performed. In the SRD-version of MPC, relative velocities are rotated randomly by a given angle around a randomly chosen direction, which mimics collisions between solvent molecules. If coupled to atomistic simulations of solute particles, e.g. by molecular dynamics (MD), the MD-particles are sorted together with fluid particles into collision cells and their velocities are included into the stochastic rotation step. Because atomistic time scales are typically smaller than hydrodynamic time scales, MD-particles are basically simulated according to their force-fields. In order to establish a coupling to the hydrodynamic medium, they are included

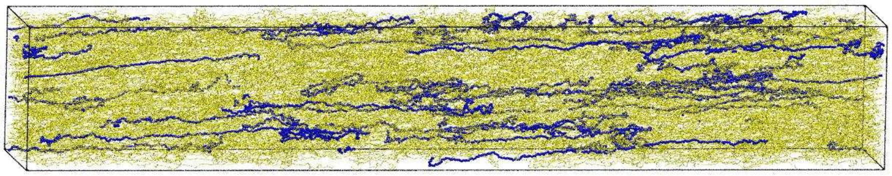

Figure 1: Example for an application of MP^{2}C for the simulation of polymers in shear flow.

every n-th step into a stochastic rotation together with the solvent particles.

Since in typical simulations of colloidal suspensions or semi-diluted polymer systems, fluid particles are $1-5$ orders of magnitude more numerous than MD-particles, system sizes get very large in simulations of 10^4-10^6 MD-particles. Therefore, although mesoscale hydrodynamics techniques are algorithmically simple, the computational demand to study large system sizes on long time scales requires an efficient parallel implementation of the simulation code.

In the present work, the highly scalable progam MP^{2}C (Massively Parallel Multi-Particle Collision) , which couples MPC with MD, is presented [6]. The current version is based on MPI uses a domain decomposition approach, where geometrical domains of the same volume are distributed onto the processors. For convenience, a decomposition of $N_p = 2^n$ is chosen.

The SRD part of the program requires basically only nearest neighbor communication to calculate center of mass velocities uniquely for domain overlapping cells and to distribute a unique set of random numbers to define a random rotation axis for the collision step. In addition, particles have to be imported/exported according to their spatial coordinate after the streaming step. The MD part is implemented via an eigth shell (ES) communication for the force computation and a minimum transfer scheme for the particle export/import. For the SRD part the program scales at present up to 65536 compute cores on the BlueGene/P architecture at JSC (the scaling was measured on the old installation of BG/P at JSC). This particle based multiscale implementation, including various types of boundary conditions, offers the large scale simulation of microscopic systems coupled to a mesoscopic flow model on extended time scales.

2 Accomplished Objectives

During the workshop MP^{2}C was benchmarked in several respects. Of interest was to see the scaling of Input/output operations as well as the speedup and performance of the basic algorithm.

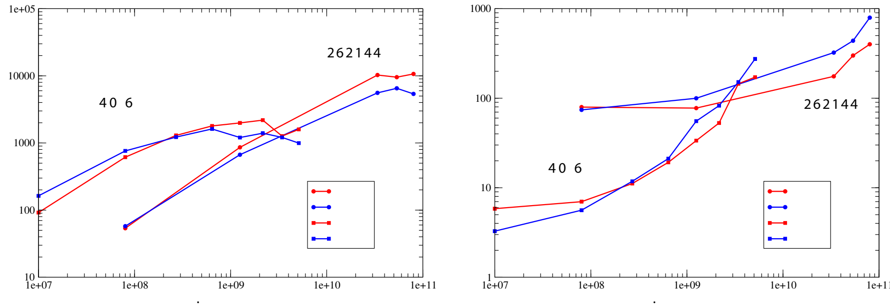

Figure 2: I/O performance of MP^{2}C using SIONlib for $P = 4096$ and $P = 262144$ processors as function of number of particles.

2.1 Data-Input/Output with SIONlib

As mentioned, MP^{2}C needs to run a large number of particles in order to describe hydrodynamic phenomena on moderately large length scales. In order to write-out or read-in restart configurations for the solvent particles, the program needs a highly efficient I/O interface. For this purpose, the recently developed I/O-library SIONlib [7] is used in MP^{2}C . Although restart information has to be saved during a simulation roughly only every 10000 to 50000 steps, it may induce a severe bottleneck in scalability. On the other hand, trajectory information of solvated particles, which are simulated by means of MD, write out very much more frequently information, which may also limit the scalability dramatically. As was reported in Ref. [8], for the interconnect at JSC from BG/P to the filesystem, which had a bandwidth of $\approx 5\,GB/s$, the performance which was reached with MP^{2}C was close to the bandwidth capacity. Shortly before the workshop, a new filesystem hardware with theoretical bandwidth of $\approx 30\,GB/s$ for read- and $\approx 15\,GB/s$ for write-operations was installed. However, from the results obtained the read- and write-performance is less than the expected one. The reason for this behavior has to be clarified.

As is seen in Fig. 2, the performance is $\approx\ 2\,GB/s$ for small number of processors and $\approx\ 10\,GB/s$ for large number of processors. As was outlined in Ref. [8] SIONlib writes out chunks of $2\,MB$, which is the size of the file system block size. For $P = 4096$ processors this means that only from $N = 1.5 \times 10^8$ particles $2\,MB$ of particle data are written out. For smaller numbers, $2\,MB$ chunks are written out, which only contain a fraction of relevant data. Therefore, the time for writing and reading for $N \leq 1.5 \times 10^8$ is more or less constant. Similar, when $P = 262144$ processors are considered, about $N = 1 \times 10^{10}$ particles have to be simulated in order to equal the file system block size equivalent.

The difference in performance is not straitforwardly explained. One reason is that the theoretical bandwidth capacity of $30\,GB/s$ is more or less the integrated performance over the whole system. Only from 24 racks on the whole

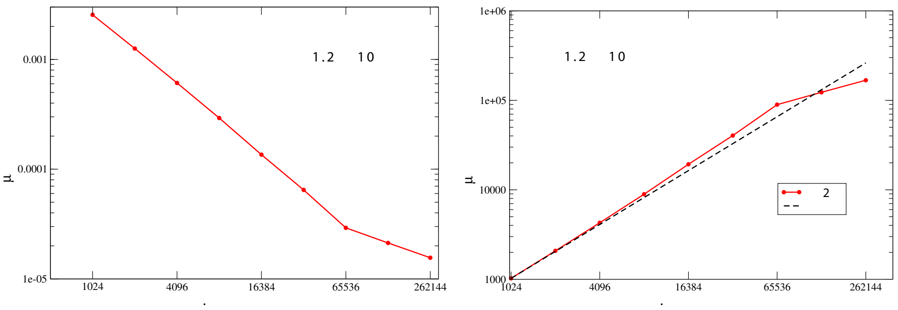

Figure 3: Strong scaling of MP^{2}C for $N = 1.25^9$ particles; left: wall clock time; right: speedup.

I/O bandwidth capacity is available. Therefore, considering less nodes, the performance decreases. But the factor, which is observed in the benchmark seems to be not only to be explained by this fact.

2.2 Strong Scaling Benchmark

Another benchmark which was performed was a strong scaling over a large range of processors ($P \in [1024, 262144]$). As can be seen from Fig. 3 the decrease in time, respective the speedup is more or less ideal until $P = 65536$ processors. From there a kink is seen which degrades the scaling. The reason for this is analysed from a time infrastructure analysis, where the different routines, contributing to the performance are measured individually. This analysis, shown in Fig. 4, clearly shows that actually all routines, performing the core routines of the algorithm, scale ideally. The calculation of relative velocities of particles in the collision cells even shows a superlinear decrease in time, which may be attributed to cache effects. The routines, which cause the saturation of scaling are clearly identified as the communication routines, responsible for the exchange of cell information in the forward- and backward-direction, needed in the collision step, as well as the particle exchange routines, transferring particles to neighbored processors, which have left a local domain. A non-ideal behavior might be expected from these routines, as they scale theoretically not like $O(N^{-1})$ but as $O(N^{-2/3})$ because of the surface scaling of the domains. However, especially for the cell exchange in forward and backward direction, a sudden increase in communication time is observed, which is not explainable without a more detailed analysis of the communication processes. During the workshop there was no time for this analysis and it is left as speculation that the distribution of nodes in the computation grid might not correspond ideally to the mesh structure of physical organisation of processors in the machine. To analyse this suspicion, different mappings of node-ID's onto the physical mesh have to be exploited and it is has to be checked whether an optimal choice of a mapping might be found.

It might be worthwhile to mention here two technical problems, which oc-

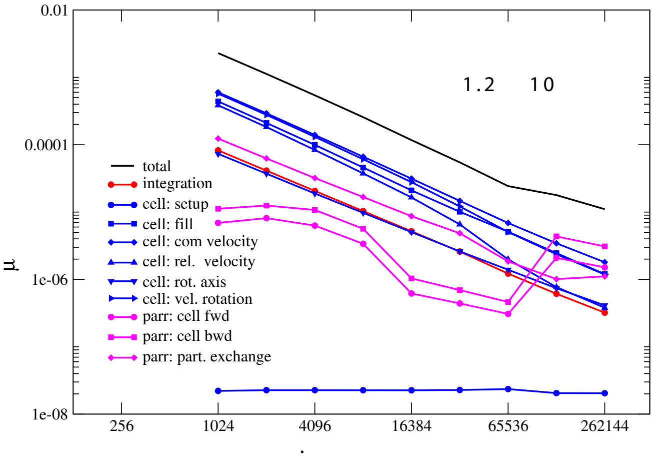

Figure 4: Infrastructure of times spent in different parts in MP^{2}C .

cured during the benchmark:

1. tag numbers of communicating processors, which are 4 byte integer numbers in MPI, were chosen in the original code as `n_nodes * rcv_id + loc_id` for the receiver node and correspondingly as `n_nodes * loc_id + snd_id` for the sender node. For more than $\sqrt{2^{32}-1}$ nodes this results in a negative tag number, which causes the program to terminate with an error. This problem could easily be removed by using `tag_offset + rcv_id + loc_id` for the receiver and `tag_offset + snd_id + loc_id` for the sender process.

2. a similar problem occured in the code with declaring absolute indices as 4 byte integer numbers. Simulating more than $N = 2.147 \times 10^9$ particles results in an integer overflow and consequently generates negative number of particles in the system, which also causes the program to stop. Consequently the program had to be scanned throughout for integers, which had to be stored as 8 byte numbers.

After solving these problems, related to number representations, the program could be translated and run. In summary it may be stated that the porting of MP^{2}C to this large number of processors was not a major problem.

3 Acknowlegment

The authors would greatly like to thank the organisers of this scaling workshop for assistance and discussions. Also the generous provision of compute time from JSC on the Blue Gene/P system is gratefully acknowledged.

References

[1] S. Succi. *The Lattice Boltzmann Equation: For Fluid Dynamics and Beyond.* Oxfor University Press, Oxford, 2001.

[2] B. Dünweg and A.J.C. Ladd. Lattice Boltzmann Simulations of Soft Matter Systems. *Adv. Polym. Sci.*, 221:89–166, 2009.

[3] A. Malevanets and R. Kapral. Mesoscopic model for solvent dynamics. *J. Chem. Phys.*, 110:8605–8613, 1999.

[4] G. Gompper, T. Ihle, D.M. Kroll, and R.G. Winkler. Multi-Particle Collision Dynamics: A Particle-Based Mesoscale Simulation Approach to the Hydrodynamics of Complex Fluids. *Adv. Polym. Sci.*, 221:1–87, 2009.

[5] R. Kapral. Multiparticle Collision Dynamics: Simulation of Complex Systems on Mesoscales. In S.A. Rice, editor, *Adv. Chem. Phys.*, volume 140, page 89. Wiley, 2008.

[6] G. Sutmann, R. Winkler, and G. Gompper. Simulating hydrodynamics of complex fluids: Multi-particle collision dynamics coupled to molecular dynamics on massively parallel computers, 2009. (submitted to J. Comp. Phys.).

[7] Wolfgang Frings, Felix Wolf, and Ventsislav Petkov. Scalable massively parallel I/O to task-local files. In *SC*. ACM, 2009.

[8] J. Freche, W. Frings, and G. Sutmann. High-Throughput Parallel I/O for Mesoscopic Particle Dynamics Simulations on Massively Parallel Computers. (ParCo 2009, submitted), 2009.

Scaling of the tmLQCD software suite on Jugene

A. Deuzeman, S. Reker, University of Groningen
G. Herdoiza, K. Jansen, NIC, DESY
P. Gerhold, Humboldt-University Berlin
C. Urbach, HISKP, Bonn University

1 Description of the Code

The tmLQCD software suite – the code we tested on the Blue Gene/P installation in Jülich – provides an implementation of tools to simulate the theory of strong interactions – Quantum Chromodynamics (QCD) – on a discrete space-time torus of size $L^3 \times T$, a lattice. tmLQCD is freely available under the GNU General Public License and published in [1]. The latest version may be downloaded from `www.carsten-urbach.eu`.

The kernel operation of these lattice QCD (LQCD) tools is the application of the lattice Dirac operator to a spinor field, which defines a linear operation on the 3+1 dimensional space-time torus connecting only neighbouring lattice sites. It is hence this kernel operation, or more precisely an even/odd preconditioned version of it (see [1] for details), that we investigated in the context of this Blue Gene scaling workshop.

The application of the lattice Dirac operator is parallelised for the Blue Gene/P by dividing the global lattice into local sub-lattices of size $l_t \times l_x \times l_y \times l_z$ on a $N_t \times N_x \times N_y \times N_z$ processor grid. The z-direction is identified with the node-internal shared memory network (i.e. $N_z = 4$), while the remaining $N_t \times N_x \times N_y$ grid must be matched to the physical network shape of a given BG/P partition. We can then take benefit of the fast next-neighbour communication network of the BG/P.

The Dirac operator itself is written in IBM XL compiler intrinsic functions in order to make use of the Double Hummer instruction set of the corresponding Power PC ship. The parallelisation is realised using the Message Passing Interface (MPI) using non-blocking communication routines. We are considering here a double precision implementation of the Dirac operator.

A second important part of tmLQCD tools is file IO, since we regularly need to read and write a large amount of data from all processors simultaneously from and to the same file. This is implemented using the MPI parallel IO functionality, in particular the collective `MPI_File_write_all` and `MPI_File_read_all` with individual file pointers.

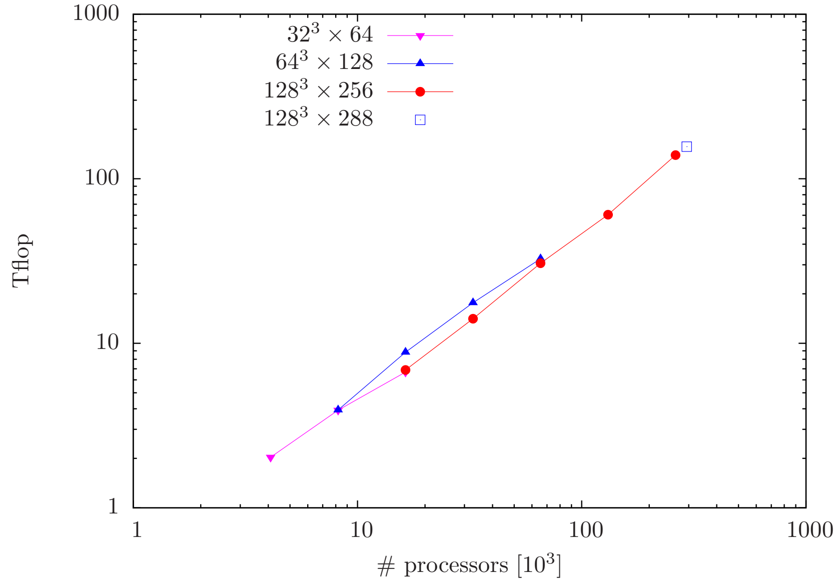

Figure 1: Strong Scaling of the Lattice Dirac operator on Jugene for three different global lattice volumes.

2 Accomplished Objectives

Our first objective was to test the strong scaling of the lattice Dirac operator on the BG/P installation in Jülich on various global volumes. Since we have to always fulfil $L = N_x \cdot l_x$ (and equally for the other directions), we are restricted to certain partitions of the installation, depending on the global lattice volume.

In figure 1 we show the strong scaling of the lattice Dirac operator for three different volumes – a $32^3 \times 64$, a $64^3 \times 128$ and a $128^3 \times 256$ global volume. We plot the total performance in Tflop as a function of the number of processors in a double log plot. It is clearly visible that the scaling is linear for all shown volumes where we could perform tests. Also the different volumes give very similar results. The slight deviations we attribute to the variation of the local lattice volume and the amount of data to be communicated.

The single point in figure 1 represents the performance measurement for a global $128^3 \times 288$ volume. This odd lattice volume was chosen to be able to run on the full 72 rack installation. In this run we observed a sustained performance of roughly 150 Tflop corresponding to 15% of peak performance. This figure can be increased to 250 Tflop if we use single precision acceleration as explained in ref. [1].

Our second objective was to test the parallel IO performance on large partitions of the BG/P. During the workshop we were able to test this on two global volumes: a $96^3 \times 192$ and a $128^3 \times 256$ lattice, corresponding to files of 97.84 Gb and 309.24 Gb, respectively. Our first observation during the tests was that file IO needs some kind of warm up phase: the first collective IO operation is usually few orders of magnitude slower than the following ones.

Unfortunately we did not have time to run the IO test on as many volumes as we wanted to. In addition we could not acquire statistics, the results compiled in table 1 are based on a single run for each case. Hence, it is not clear which conclusions to draw from these results. One definite conclusion from these experiments is that parallel IO is by two orders of magnitude faster for our application, since serially we usually reach write performances of about 0.04 [Gb/s]. It would be very useful for us to further investigate the file IO issue.

# procs.	$L^3 \times T$	read [Gb/s]	write [Gb/s]
32768	$96^3 \times 192$	5.50	6.95
32768	$128^3 \times 256$	4.25	4.36
65536	$96^3 \times 192$	9.59	10.74
65536	$128^3 \times 256$	8.23	5.16

Table 1: MPI parallel IO performance.

The BG scaling workshop was useful to us in two respects: first of all we gained confidence that we can scale our problem to the full BG/P installation in Jülich. This will allow us to tackle large scale problems which will be very relevant for investigating QCD on the lattice. Secondly, we observed a tremendous potential performance gain from the use of parallel IO routines. To what extent we shall be able to leverage this in the context of production code is currently still an open question.

[1] K. Jansen and C. Urbach, Comp. Phys. Comm. **180**(2009)2717.