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

Technical Report

Jülich Blue Gene/P
Extreme Scaling Workshop 2011

Bernd Mohr, Wolfgang Frings (Eds.)

FZJ-JSC-IB-2011-02

April 2011

(last change: 08-Apr-2011)

Jülich Blue Gene/P Extreme Scaling Workshop 2011

Bernd Mohr and Wolfgang Frings
Jülich Supercomputing Centre

Executive Summary

From February 14 to 16, JSC organized the 2011 edition of its Blue Gene Extreme Scaling Workshop. Similar to the previous workshops in 2009[1] and 2010[2], 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 – still 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. The process was very competitive: Only 8 out of the submitted 15 high-quality proposals were selected[1]:

- Improving Octopus Towards the new Generation of HPC Systems
  J. Alberdi, **X. Andrade**, A. Arruabarrena, J. Muguerza, A. Rubio
  University of the Basque Country, Spain and Harvard University, USA
- Extreme Scaling of Brain Simulations
  S. Benjaminsson, **A. Lansner**
  Royal Institute of Technology, Sweden
- Global Gyrokinetic PIC Simulations of Plasma Microturbulence at Extreme Scale with the GTC-P Code
  S. Ethier, W.M. Tang
  Princeton Plasma Physics Laboratory, USA
- Billions-Body Molecular Dynamics at Extreme Scaling
  A. Fratalocchi, **N. Allsopp**
  King Abdullah University of Science and Technology (KAUST), Saudi Arabia
- Lattice-Boltzmann Methods in Fluid Dynamics: Turbulence and Complex Colloidal Fluids
  D. Groen, **O. Henrich**, **F. Janoschek**, P. Coveney, J. Harting
  University College London, UK and
  Eindhoven University of Technology, The Netherlands and
  University of Stuttgart, Germany

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

- Scalability Test of $\mu\varphi$ and the Parallel Algebraic Multigrid solver of DUNE-ISTL
  O. Ippisch, **M. Blatt**
  Heidelberg University, Germany
- Scaling of Eigenvalue Solver Dominated Simulations
  R. Johanni, **A. Marek**, H. Lederer, V. Blum
  Rechenzentrum der Max-Planck Gesellschaft, Garching and
  Fritz-Haber Institut, Berlin, Germany
- Parallel Simulations of Quantum and Frustration Effects in Molecular-based Nanomagnets
  M. Antkowiak, **L. Kucharski**, P. Kozłowski, R. Matysiak, G. Kamieniarz
  Adam Mickiewicz University, Poznań and
  University of Zielona Góra, Poland

During the workshop, the teams were supported by JSC parallel application experts, the JUGENE system administrators and two IBM MPI and compiler experts; however, the participants shared a lot of expertise and knowledge, too. The workshop was extremely successful: the 8 teams succeeded in submitting one or more successful full 72-rack jobs for 11(!) different applications during the course of the workshop as some teams experimented with more than one code. A total of 308 jobs were launched using 122 rack days of the total 157 rack days reserved for the workshop.

Achieved Results

The following chapters give an account on more detailed execution and scaling results achieved by 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.

References

[1] Bernd Mohr, Wolfgang Frings, Jülich Blue Gene/P Extreme Scaling Workshop 2009, Technical Report FZJ-JSC-IB-2010-02, Forschungszentrum Jülich, February 2010.
`http://www2.fz-juelich.de/jsc/docs/autoren2010/mohr1`

[2] Bernd Mohr, Wolfgang Frings, Jülich Blue Gene/P Extreme Scaling Workshop 2010, Technical Report FZJ-JSC-IB-2010-03, Forschungszentrum Jülich, May 2010.
`http://www2.fz-juelich.de/jsc/docs/autoren2010/mohr2`

Improving Octopus Towards the new Generation of HPC Systems

Joseba Alberdi Rodriguez, University of the Basque Country
Xavier Andrade, Harvard University
Agustin Arruabarrena, University of the Basque Country
Javier Muguerza, University of the Basque Country
Angel Rubio, University of the Basque Country

Description of the Code

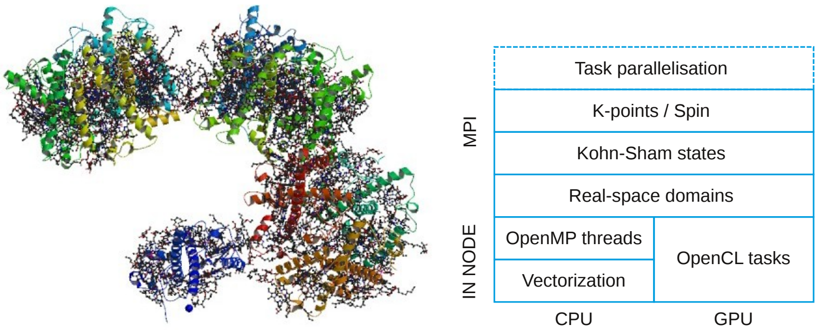
Due to its importance as the main source of energy for biological systems, the photosynthetic process carried out by plants and some bacteria has been extensively studied. It is a complex process that involves biological, chemical and physical phenomena. From a molecular point of view, the basic units for light absorption are chlorophyll molecules. These molecules, combined with other chromophores and proteins, form the different complexes that take part in photosynthetic processes. The large size of these molecules, of the order of thousands or tens of thousand of atoms, makes them very challenging to model computationally. However there are some details of the process that can only be elucidated by performing first principle simulations of the excited states of these systems [1].

The tool we used for such simulation is the Octopus code [2], a scientific software package based on the numerical implementation of the time-dependent density functional theory (TDDFT) [3]. This theory allows us to study from first principles the excited states properties of large electronic systems. As a test system we are focusing on the active part of a light harvesting complex of Spinach, Figure 1(a). The full molecule contains 5879 atoms, but we can also simulate portions of it that contain less atoms. Octopus applies a multilevel parallelisation, using MPI for message-passing among nodes, and OpenMP for intra-node (core) shared memory parallel computation, see Figure 1(b). Also some specific parts are written in vectorial code to take advantage of the Blue Gene dual FPU, x86 SSE/AVX instructions, or GPUs using OpenCL.

Octopus has been used in HPC centres all over Europe and it is routinely used for calculations in parallel systems. From experience we have seen that the scaling of the code is promising enough to use the whole Jugene machine.

Results

During the workshop, we performed runs simulating the absorption of light by large molecular systems using the Octopus code. We were able to run without problems in the whole machine, which we consider already as an achievement for



(a) Light harvesting complex of Spinach (b) Parallelisation strategy

Figure 1:

a complicated software package of almost 200 thousand code lines and multilevel parallelisation based on task and data distribution, using also HPC libraries. We ran in SMP mode using a combined MPI and OpenMP parallelisation.

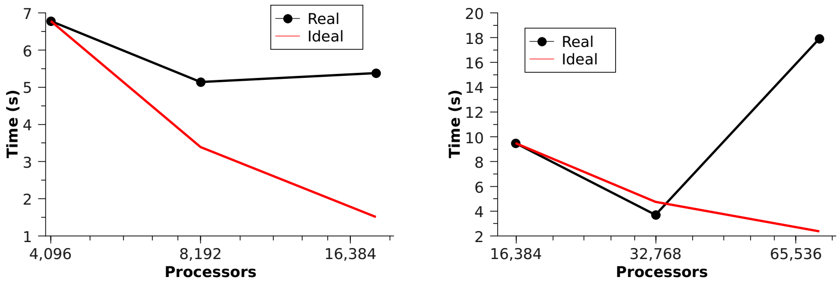
As mentioned before, this was the first time that we have used such a big atomic system with Octopus. This was possible because in the preparation we were able to overcome the memory limitation problem by using the ScaLAPACK library for some dense linear algebra operations.

For this event, there was also a successful implementation of execution pipelining, that we call task-parallelisation. Our main bottleneck, with a huge number of processors, is the solution of the Poisson equation which originally had to be done by all the processors. Now, we are able to solve the Poisson equation on a subset of processors, allowing us to perform other operations by the rest of the processors at the same time.

The task-parallelisation technique gives us a performance improvement of around 22%, for the system of 2676 atoms using 8192 processors. In a first execution, without the task-parallelisation, the time execution per iteration was of in 6.6 seconds. In a second execution, with task-parallelisation, the execution needed a time of 5.13 seconds per iteration, thus, a 78% of the time of the first execution. A similar behaviour can be seen in the Figure 2(b) for the bigger system of 5879 atoms. The execution of the system of 5879 atoms in 16K processors was performed without this improvement and we could see that the next execution, with task-parallelisation, in 32K processors was twice as good. Supposing a linear scaling of the general code, the improvement of the task-parallelism was again around 22%.

With 32K processors and the 5879 atoms system we managed to achieve a time of 3.7 seconds per iteration (Figure 2(b)), which means we could obtain an absorption spectrum in less than 12 hours of computational time. This level of performance makes it possible to obtain scientifically relevant results for these large systems.

The throughput for this run is of at least 11.0 Teraflops which represents a parallel efficiency of at least 25%. With 72 racks, however, scalability was quite poor, the time per iteration increased up to 15 seconds. This slowing down can be explained by the parameter adjustment required by the task-parallelisation, that we have implemented recently and still needs to be improved.

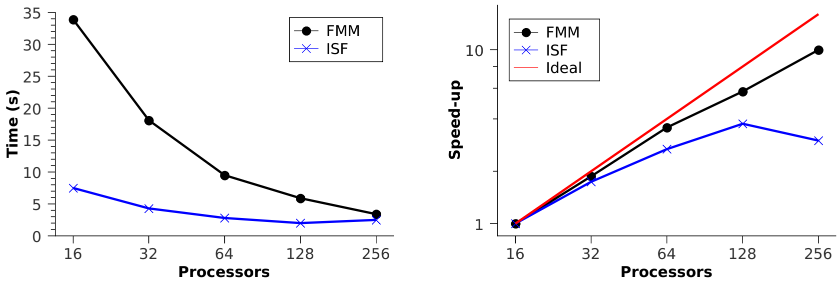


(a) 2676 atoms.

(b) 5879 atoms.

Figure 2: Execution time per iteration for a different number of processors with different system sizes. The ideal line is obtained by assuming a linear scaling starting from the calculation with less processors.

There are other alternatives we are exploring to improve the parallelisation of the Poisson solution by using more efficient parallel solvers based on parallel libraries. The most promising approach is based on the FMM library [4], developed at the JSC, that has a very good parallel scaling. Our previous execution suggested us a parallelisation improvement with a huge number of processors. Although the execution time in a few nodes is greater than with the standard Octopus solver (ISF) [5], Figure 3(a), the trend is that the FMM will continue having good a speed-up, while the ISF will suffer from certain slowing down. As a result, using the FMM will overcome the bottleneck of the Poisson equation and Octopus will run more efficiently. Unfortunately, due to execution problems, that have now been solved, we could not test the solver during the workshop. Another alternative as an efficient parallel Poisson solver is the PFFT library [6], which is currently under development.



(a) Solution time.

(b) Speed-up.

Figure 3: Comparison of parallel performance for the FMM and ISF Poisson solvers for a system with 180 atoms.

Conclusion

The tests done in Jugene were our first opportunity to simulate large systems such as full biological molecules. This type of simulations have a direct implication in our understanding of photo-induced processes in biological systems. Indeed, for the first time in this workshop, we have been able to run molecules with 2676 and 5879 atoms (whole chlorophyll molecule), that we could not run before. In fact, the simulations from first principles of the excited states of a molecule of thousands of atoms pushes the limit on what can be done by the scientific community.

Acknowledgements

We would like to thank the organisers of the workshop for giving us the opportunity to visit the Jülich Supercomputing Centre, as well as interact and discuss with its scientists and run our code in one of the fastest supercomputers in the world. We think that the JSC scaling workshop is an excellent initiative that allows scientists to become closer to the supercomputing community. We also would like to thank IBM for the opportunity to interact with their HPC experts.

We acknowledge funding by the Spanish MEC (FIS2007-65702-C02-01 and FIS2010-21282-C02-01), ACI-promociona project (ACI2009-1036), the Basque government grant "Grupos Consolidados UPV/EHU del Gobierno Vasco" (IT-319-07), and the European Community through e-I3 ETSF project (Contract No.211956). JA acknowledges support from a fellowship from the University of the Basque Country UPV/EHU. XA acknowledges the US National Science Foundation through the SOLAR (DMR-0934480) award.

References

[1] T. K. Ahn, T. J. Avenson, M. Ballottari, Y.-C. Cheng, K. K. Niyogi, R. Bassi, and G. R. Fleming, "Architecture of a charge-transfer state regulating light harvesting in a plant antenna protein," *SCIENCE*, vol. 320, no. 5877, pp. 794–797, 2008.

[2] A. Castro, H. Appel, M. Oliveira, C. A. Rozzi, X. Andrade, F. Lorenzen, M. A. Marques, E. Gross, and A. Rubio, "Octopus: a tool for the application of time-dependent density functional theory," *Physica Status Solidi B Basic Research*, vol. 243, pp. 2465–2488, Apr. 2006. `http://www.tddft.org/programs/octopus/`

[3] E. Runge and E. K. U. Gross, "Density-functional theory for time-dependent systems," vol. 52, pp. 997–1000, 1984.

[4] I. Kabadshow, "The Fast Multipole Method-Alternative Gradient Algorithm and Parallelization," *Report Nr.: Berichte des Forschungszentrums Jülich JUEL-4215*, p. 83, 2006.

[5] L. Genovese, T. Deutsch, A. Neelov, S. Goedecker, and G. Beylkin, "Efficient solution of poisson's equation with free boundary conditions," *The Journal of Chemical Physics*, vol. 125, no. 7, p. 074105, 2006.

[6] M. Pippig, "An Efficient and Flexible Parallel FFT Implementation Based on FFTW," 2011.

Extreme Scaling of Brain Simulations

Simon Benjaminsson and Anders Lansner

Department of Computational Biology
Royal Institute of Technology, Stockholm, Sweden

Description of the Code

Here we are investigating scaling of large-scale neural simulations using an experimental neural simulator (ANSCore) and an experimental code simulating a model of the neocortical network of the brain (BrainCore).

ANSCore is a newly developed large-scale neural network simulator, used as a base for a number of ongoing projects. These include models of the mammalian olfactory system and for use as the information processing engine in artificial olfactory systems based on polymer sensors. Parts of it have also been used for large-scale data analysis [1]. All network communication is performed using MPI collective functions. Abstract neural network operations, such as winner-take-all functions across populations of neural units, are implemented using MPI collective functions and MPI communicators for these specific network parts. This is also used to integrate analysis algorithms of the network dynamics in a scalable parallel fashion, reducing the need to store vast amounts of neural activity for separate post-processing analysis.

BrainCore is an experimental code which simulates an abstract Poisson spiking model of the global neocortical network of the brain. It is designed to obey what is known in terms of the overall structure of the long-range cortico-cortical connections. As in the brain at large, this connectivity is very sparse – the connection matrix is filled to a fraction of about 10^{-6}. All connections are trainable using the BCPNN learning rule [2] and network units are adapting. The brain process we are modeling is cortical associative memory in the form of a recurrent attractor memory. The aim is to design a modular network architecture that can run in real time and scale to arbitrary machine sizes, i.e. that demonstrates perfect weak scaling. Real time is here defined as learning and recall of one pattern, and an inner loop execution time of less than 1 millisecond. The modules, so called "hypercolumns" or "macrocolumns" will fit on one process and the memory and computing demands will be fixed by design. This means that network connectivity becomes increasingly sparse as the network size increases. Also, the number of in- and outgoing spikes is constant for a module. The main performance measure is the loop time which should remain at below one millisecond regardless of the number of cores.

Results

Figure 1 displays scaling behavior of a network simulation using ANSCore comprising 65.5 million neural units and 393.2 billion connections. The model tests the core components of the simulator by associative storage and retrieval of a set of input patterns in a randomly and sparsely connected recurrent network. The timing can be broken up in each individual component, displaying behavior as the simulation is scaled up. As the number of processes increases for the simulated model with a fixed network size, the fraction spent on communication compared to computation starts to become large, which is responsible for the slight deviation from linear scaling. Analysis of the network dynamics is performed by checking the closeness of the network state to memories stored in the network in each time step. Runs over the full 72 rack system were also successfully performed, as well as smaller networks using lower number of processes with a lower baseline than 64K processes.

BrainCore was tested up to 32K processes already in advance of the workshop. The aim of the workshop efforts was to show that the design goals of perfect weak scaling could be achieved on the most parallel machine in the world, JUGENE. The memory performance was tested just by during testing stimulating all hypercolumns except the last one. Then it was checked that the activity was correctly filled in for that hypercolumn. We stored only a hundred random patterns (with 1% activity) so the task was fairly simple. It was solved perfectly in all reported cases. The network was run in virtual node mode. We managed to demonstrate weak scaling performance for several large runs, the largest running on the full machine, i.e. 72 racks with 294912 cores in total. This amounts to more than 29 million spiking units connected by about 295 billion trainable connections. Point-to-point communication was used. The loop time of the inner loop leveled off just a little above 1 millisecond which implies real time performance (Figure 2). We also recorded the number of spikes sent and received for a module during operation. This number was close to 47750 for 200 sec run for all our runs. This means that for the largest run more than 14 billion spikes were communicated, which amounts to about 70 million spikes per second. Due to the network implementation, this corresponds to 100 times more spikes on a unit-to-unit basis. Since one spike message contains only an MPI_INT this is a very small fraction of the maximum network bandwidth.

We further had the intension to compare scaling performance using collective communication with the initially developed point-to-point implementation. Due to time constraints this was not achieved. It also remains to optimize the code using a profiler to get a more precise idea of performance bottlenecks and potential for optimization. We also intend to make the network model somewhat more complete so that we can run some serious associative memory tests with it. Additional runs will continue within the DEISA framework within which we have CPU time on JUGENE.

Conclusion

We have successfully scaled up two experimental codes aimed at large-scale neural simulations to the full system scale of JUGENE. Scaling of core components for building a variety of neural models was featured in ANSCore. There

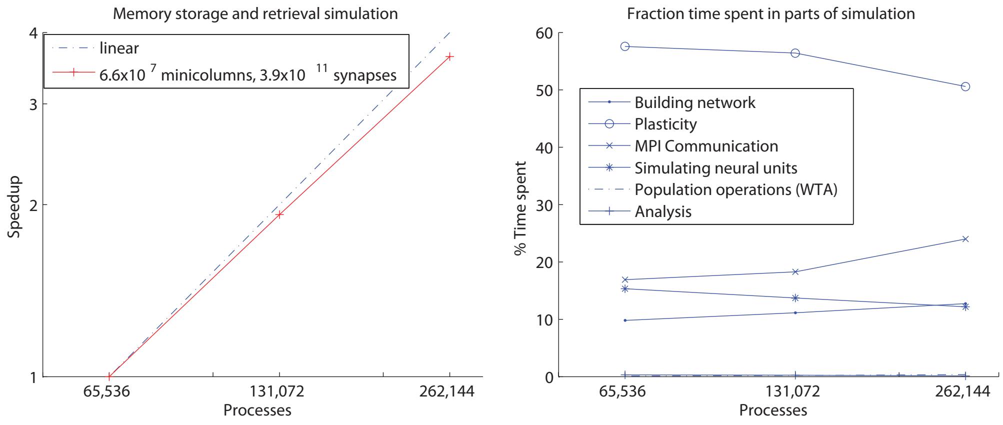


Figure 1: Scaling of a memory task of a network comprising 65.5 million neural units and 393.2 billion connections. Left shows speedup and right displays a division of where the time is spent in the simulation.

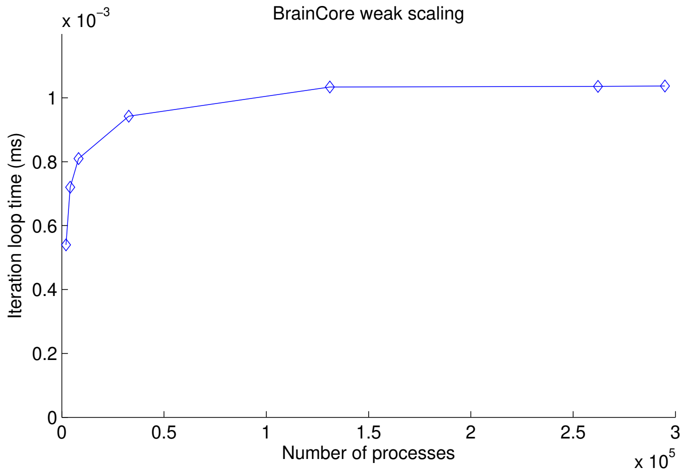


Figure 2: Time spent in BrainCore inner loop. The leveling off around 1 millisecond implies real time performance of simulation compared to a real brain.

communication was handled by MPI collective calls while the BrainCore model featured a large homogenous single network and a straight-forward application in terms of associative memory using point-to-point communication. The results achieved open up for simulation of neural models of sizes comparable to real mammalian nervous systems with a much higher complexity than so far attempted. We will be able to handle spiking communication in models of the neocortical network as these scales up to sizes of real mammalian brains. Furthermore, our study paves the way for use of extremely scalable brain network models for information processing of data obtained with e.g. large-scale sensor arrays. The knowledge gained can also we used to investigate the design of dedicated hardware.

Acknowledgements

The authors would like to thank Bernd Mohr and the Juelich team for the organization of the workshop. Also, the DEISA consortium is acknowledged for the provision of compute time and support.

References

[1] Benjaminsson S., Fransson P. and Lansner A. (2010): A novel model-free data analysis technique based on clustering in a mutual information space: application to resting-state fmri. Front. Syst Neurosci.: 4, 1-8

[2] Sandberg A., Lansner A., Petersson K.-M. and Ekeberg . (2002): Bayesian attractor networks with incremental learning. Network: Computation in Neural Systems: 13(2), 179-194.

Global Gyrokinetic PIC Simulations of Plasma Microturbulence at Extreme Scale with the GTC-P Code

Stéphane Ethier and William M. Tang
Princeton Plasma Physics Laboratory

Description of the Code

The Gyrokinetic Toroidal Code (GTC) is a 3D particle-in-cell (PIC) code developed at the Princeton Plasma Physics Laboratory for the study of turbulent transport in magnetic confinement fusion devices called tokamaks [1]. The non-linear gyrophase-averaged Vlasov-Maxwell system of equations is solved in global toroidal geometry by following the trajectories of charged particles along the characteristics in 5D phase space + time (Lagrangian scheme, ODE equations). The particles interact with each other via a self-consistently generated field evaluated on a grid by solving the Poisson equation (PDE). Figure 1 shows a volume rendering of the electrostatic potential in a typical GTC simulation (curtesy of Chad Jones, UC Davis).

GTC consists of about 10,000 lines of Fortran 90 and some C. The Message Passing Interface (MPI) library is used for communication between the processes. The use of magnetic coordinates allows us to use a second-order Runge-Kutta method to advance in time since the lowest order trajectory of a particle in such coordinates is a straight line. The charge of each particle is deposited on a grid and the gyrokinetic-Poisson equation is solved using a non-spectral solver. Apart from being very fast, the non-spectral nature of this solver allows it to completely avoid the very expensive ALL-TO-ALL communications of the widely used FFT-based solvers. On Blue Gene/P, calls to the vectorized library MASSV have been added to take advantage of the double floating-point unit on the BG/P core. Process placement has also shown to improve the performance of production runs by up to 30% [2].

It is worth noting that two versions of the GTC code were exercised on Jugene. The "classic" version, which has been widely used for both production and benchmarking runs, and the newer GTC-P version, which includes an improved solver and 2D domain decomposition. Both versions implement an MPI-based toroidal domain decomposition and OpenMP-based loop-level shared memory parallelism. However, while the classic GTC further separate the particles within each toroidal domain using MPI, GTC-P implements a full radial domain decomposition that distributes both particles and grid points. This allows GTC-P to simulate much larger tokamak devices than the classic GTC for the same amount of memory [3]. This is especially important for a

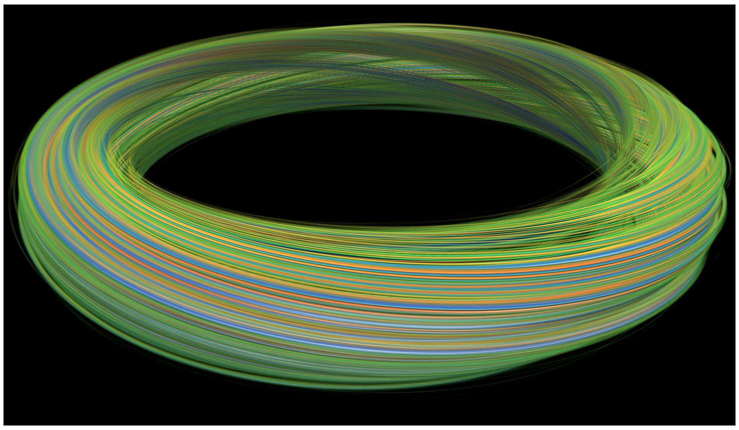


Figure 1: An illustration of the turbulent electrostatic potential self-consistently generated by the charged particles in a GTC simulation.

platform such as Blue Gene where the memory per node is small compared to other systems. GTC-P also includes a fully parallel solver implemented with the PETSc library (www.mcs.anl.gov/petsc). This permits us to easily switch solvers depending on the physics being studied.

Results

What worked:

- 72-rack GTC run in VN mode for a small tokamak (2 million grid points, 95 billion particles)
- 32-rack GTC-P run in VN mode for a tokamak larger than ITER (525 million grid points, 52.5 billion particles)

What did not work

- 72 and 32-rack GTC-P runs in SMP mode for mixed-mode MPI+OpenMP using the 32-rack case or larger.
- 72-rack GTC-P runs in VN mode using the 32-rack case and a proportionally scaled case.

Our goal for the workshop was to run our latest production version of the Gyrokinetic Toroidal Code, GTC-P, which includes a 2D domain decomposition that has been very successful on the Blue Gene system and has allowed us to carry out very large simulations with little memory per MPI process. This latest version, however, makes heavy use of the PETSc library and we ran into problems while trying to push the code to the full 72-rack JUGENE system. GTC-P easily scales to 32 racks on the INTREPID BG/P at the Argonne Leadership Class Facility (ALCF) in the US, where production runs are routinely carried out on 8 racks with this version. The production runs and scaling tests are normally done in virtual node mode (VN) with only MPI although GTC-P, and the classic GTC as well, can be run in mixed-mode MPI+OpenMP, where the OpenMP is implemented at the loop-level. The idea was to compare test runs for both with and without OpenMP to see if OpenMP gives an advantage at large scale. Unfortunately, the only mixed-mode SMP test that worked was the 2-rack case that already ran successfully during preparation for this workshop. I tried to run a 32-rack test case in SMP mode with 4 OpenMP threads, which is essentially a weakly scaled version of the 2-rack case, but the code crashed early on when preparing the matrix for the PETSc solver. I did not have a chance to rerun this case with the debug version of PETSc so I did not get any useful error messages back from PETSc itself. I also tried a 72-rack run with the same test case so that the memory footprint would be even smaller but it also crashed with little useful information.

I then switched to MPI-only tests in VN mode. On the first night of the workshop I had tried a weakly-scaled version of my test case on 72 tacks. This was an extremely large case, perhaps the largest ever attempted for this kind of particle-in-cell simulation: 4.7 billion grid points with 470 billion particles. The code failed with a segmentation violation, most probably due to the problem size being too large. The particles have essentially perfect weak scaling but not the grid. Because of the toroidal geometry of our system, it is practically impossible to balance both the particles and the grid points for all the MPI processes with the 2D domain decomposition. The 72-rack problem size probably tried to allocate too much memory on some of the MPI processes. To make sure that it was not due to a difference in the libraries between JUGENE and INTREPID I ran the same 32-rack case that I normally run at ALCF. That case worked without any issue. I then used that same case and ran it on 72 racks, performing strong scaling to ensure that the case would not run out of memory. Unfortunately, it failed with a segmentation violation inside PETSc but the accompanied message was different from the previous one, indicating that the code did not run out of memory, as expected, but something else produced the error. This will require more investigation by trying to reproduce the same error at smaller scale.

After concluding that PETSc was most likely the cause of the failures, I decided to switch to the classic GTC since it does not make use of the PETSc library or any other external libraries. This version is now widely used for benchmarking purposes although it was the main production version for many years. NERSC, for example, has used it in its last two procurements of large systems (Franklin and Hopper). The MPI parallelism in this version consists of a 1D domain decomposition in the toroidal direction (long way around the torus geometry of our simulation volume), and a particle distribution within each toroidal domain. Each MPI process within a domain holds a fraction of the particles but a full copy of the local grid, essentially 2 toroidal planes (1 real plane and 1 ghost plane). When simulating large tokamaks, such as ITER, the copy of the local grid is too large to fit in the memory of the BG/P nodes. Hence the test case for this version is a relatively small tokamak (2 million grid points) but with a very large number of particles (95 billion). This test case ran without problems on the full 72-rack JUGENE. The use of such a large number of particles per grid point can be justified in the case of a simulation where the particles represent the full distribution function in phase space as opposed to only the perturbed part.

In light of these results, we believe that a full 3D domain decomposition will probably be necessary in GTC as we move to exascale so that both particles and grid get uniformly distributed. Alternatively, we will also explore a highly multi-threaded algorithm that will keep a single plane per node depending on the memory available.

References

[1] Z. Lin, T. Hahm, W. Lee, W. Tang, R. White, *Turbulent transport reduction by zonal flows: Massively parallel simulations*, Science 281 (5384) (1998) pp.1835–1837.

[2] S. Ethier, W. Tang, R. Walkup, L. Oliker, *Large-scale gyrokinetic particle simulation of microturbulence in magnetically confined fusion plasmas*, IBM Journal of Research and Development 52 (1-2) (2008) pp.105 –115.

[3] S. Ethier, M. Adams, J. Carter, and L. Oliker, *Petascale Parallelization of the Gyrokinetic Toroidal Code*, in proceedings of VECPAR 2010 conference on High Performance Computing for Computational Science, Berkeley, CA, June 22–25, 2010 (`http://vecpar.fe.up.pt/2010/papers/2.php`).

Billions-Body Molecular Dynamics at Extreme Scaling

Andrea Fratalocchi and Nicholas Allsopp
King Abdullah University of Science and Technology (KAUST)
Saudi Arabia

Description of the Code

Thanks to our Billions-Body Molecular Dynamics (BBMD) code, we will study for the first time the behavior of large-size structured glasses characterized by tens of Billions of particles. This will permit to answer long-standing questions in the field of complex systems concerning the existence and the dynamics of specific wave vibrations of structurally disordered systems like glasses.

The BBMD code is a highly optimized, parallel C++ MD code for Lennard-Jones particles systems. To perform the time evolution of the system, forces between particles have to be calculated and particle pairs whose distance is below the cutoff range have to be found. For this task, we adopt an O(N) linked-list method, with an inexclusive grid that allows more than one particle to occupy a single cell. Newton's third law is then applied in order to halve the number of neighbors to be checked in the calculation of the forces.

The parallelization was implemented with the domain decomposition strategy, also known as spatial decomposition, where the simulation box is divided into sub-domains and each domain is assigned to each processor. The Message Passing Interface (MPI) standard is then employed to handle all parallel communications among processes. This type of parallelization has the advantage to require only nearest-neighbor communications, with a few limited global collective operations.

Results

The BBMD code was successfully run on all 72 racks of the Julich Supercomputing Centres Blue Gene/P system. The table below shows the strong scaling results for 10 billion particles. The memory footprint is such the code was able to run in VN mode, thus the scaling from 32786 cores (8 racks of BG/P) to 294912 cores (72 racks of BG/P) shows an efficiency of 89%.

The amount of communication to elapsed time can be seen to be less than 2% for 8 racks and only grows to 7% for 72 racks. This shows that utilizing mainly nearest neighbour communication, with a very limited amount of global communication, is very advantageous for extreme scaling.

Racks	**Comms Time**	**Elapsed Time**	**Memory**	**% Ideal Scaling**
8	242	17950	564	
16	208	9296	300	97
32	167	4779	180	94
64	124	2484	143	90
72	136	2251	143	89

Table 1: Measurements using a fixed problem size of 10B (10.0 E+9) particles

Conclusion

The extreme scaling workshop was very helpful to us. It enabled the demonstration of the capability of the BBMD to scale further than had been achieved previously. With a problem size of 10 billion particles the code was able to achieve an excellent scaling of nearly 90% on 72 racks compared to 8 racks of BG/P.

Lattice-Boltzmann Methods in Fluid Dynamics: Turbulence and Complex Colloidal Fluids

Derek Groen, University College London
Oliver Henrich, University College London
Florian Janoschek, Eindhoven University of Technology
Peter Coveney, University College London
Jens Harting, Eindhoven University of Technology
and University of Stuttgart

Description of the Codes

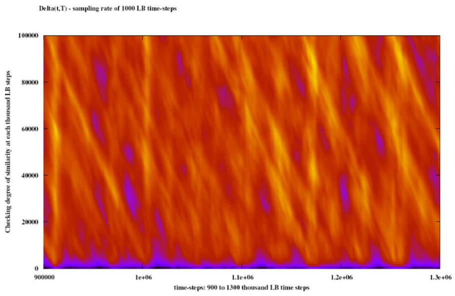
Over the last years, the lattice Boltzmann method proved very successful in modeling fluids in many different applications for science and engineering. Compared to traditional Navier Stokes solvers, the method allows an easy implementation of complex boundary conditions and—due to the high degree of locality of the algorithm—is well suited for the implementation on parallel supercomputers.

LB3D is a general purpose lattice Boltzmann implementation developed over the last 12 years at three research institutes residing in the UK, Germany, and the Netherlands. Among other applications, the code is used to study the rheology of ternary fluid mixtures [1] and blood flow under various conditions [2]. The main benchmark systems during the workshop are emulsions stabilized by colloidal particles (Pickering emulsions) [3], for which a study of droplet size distributions demands simulations of considerable spatial and temporal dimensions.

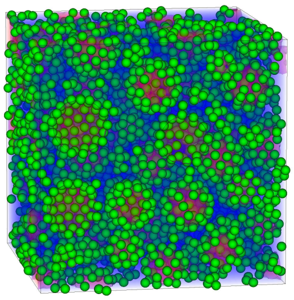
HYPO4D is a four-dimensional extension to the existing lattice Boltzmann implementation, which parallelizes in time as well as in space. The parallelization in time aids us in identifying unstable periodic orbits (UPOs) in turbulent flow. These UPOs provide a countable sequence of orbits and can therefore allow us to characterize the structure and dynamics of the attractor of the system [4]. To accomplish this, we apply our solver with a novel spacetime variational algorithm, which minimizes a candidate orbit to a UPO.

Results

First benchmarks of LB3D with additional timing functions performed during the workshop showed that the coupling of suspended particles to the fluid involved a subroutine that scaled linearly with the number of cores. Due to the small pre-factor this was not recognized earlier. Another finding was that the (serial) output of the Lagrangian particle trajectories would fail for large core counts since it relied on point-to-point communication between every rank and



(a)



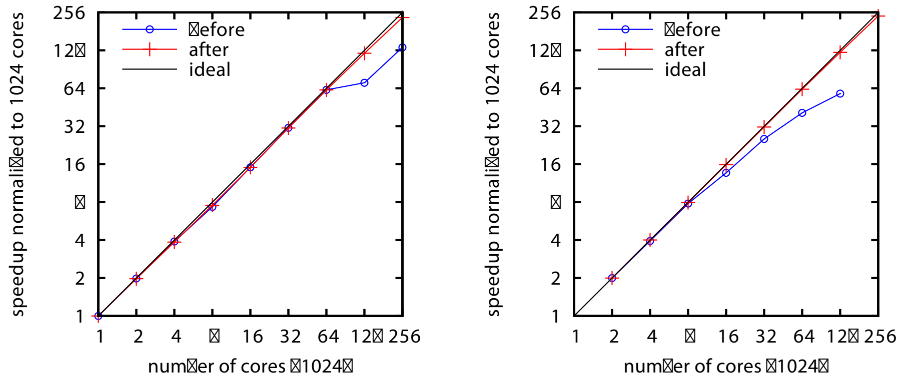
(b)

Figure 1: Physical systems simulated by our codes. A plot with results from our numerical 4D relaxation scheme is given in (a). Here the purple areas are of special interest, as these regions contain UPO candidates. A snapshot of our colloidal benchmark system: a Pickering emulsion (taken from [3]) is shown in (b).

the root process instead of collective MPI calls. Turning off Lagrangian output, we could still successfully run the code on 294912 cores for the first time and produce parallel HDF5 output of velocity and density fields for a volume of $1024^2 \times 1152$ lattice sites containing two fluid components and 454508 colloidal particles at a volume concentration of 20 %.

In discussions, mainly with Pascal Vezolle (IBM), we learned that our code could be optimized by better exploiting the network topology of JUGENE and implementing a hierarchical output scheme for HDF5 data in which only part of the tasks perform actual file IO. However, due to the limited time, it was not possible for us to implement major optimizations at the workshop. Still, in the following weeks, we enabled LB3D for manual matching of the physical geometry of the problem to the hardware topology. This change is the main cause for the improved speedup for a benchmark system of $1024^2 \times 2048$ lattice nodes containing only one fluid species as shown in Figure 2(a). Further improvement seems possible by integration of recent optimizations performed for HYPO4D. Also the linear contribution to the computational cost of the particle-fluid coupling was eliminated. The effect of both modifications together is visible in Figure 2(b) comparing the speedup before and after the workshop for a system of $1024^2 \times 2048$ lattice sites carrying two fluid components and 4112895 colloidal particles.

We also had the opportunity to test and diagnose a number of optimizations which we recently inserted in the HYPO4D code. During the workshop we found that the hand-written collective communication and File I/O operations posed a barrier to perform diagnostics on the code for runs beyond 65536 cores and to scale the code in general beyond 131072 cores. As a result, we spent a large part of the workshop simplifying and rewriting these collectives. This effort has been partially successful as we are now able to run across 131072 cores with diagnostics enabled. Unlike previous version, the use of the collective operations in the diagnostics no longer adversely affects the performance or stability of the code. In addition, we eliminated some of the errors which prevented us from



(a) (b)

Figure 2: Strong scaling of LB3D on JUGENE before and after the workshop. (a) relates to a system with only one fluid component, (b) to one with two fluid species and suspended particles similar as in Figure 1(b). The absolute execution times for small core counts did not change significantly.

running across 262144 cores, and are in the process of diagnosing and resolving several other issues which prevent the code from completing successfully at that core count.

Converting the Lagrangian output of LB3D to collective operations cost unexpectedly large effort: first, sending custom data types of zero lengths in `MPI_gatherv()` led to errors which were hard to identify since the MPI call still pretended to have finished successfully. Secondly, `MPI_gatherv()` proved unacceptably slow. Using `MPI_allgatherv()` instead, we could speedup the communication by a factor of 77 on 131072 cores at the cost of allocating large receive buffers for every task. In first tests, the new implementation successfully wrote trajectories and full checkpoints for 435600 particles on 131072 cores.

Outlook

While all groups using LB3D profit from its increased efficiency the improvement of the Lagrangian output is a crucial prerequisite for massively parallel production runs. These will be employed soon for the study of Pickering emulsions. The know-how obtained at the workshop is likely to foster further optimizations in the future.

The knowledge acquired at the workshop also led to a number of revisions in the HYPO4D code already, but we have planned a large number of additional changes. First priority is to eliminate the remaining obstacles which prevent us from running across 262114 cores. We also intend to implement the hierarchical file I/O solution described earlier in this report, and to optimize the 4D relaxation scheme both for accuracy and performance.

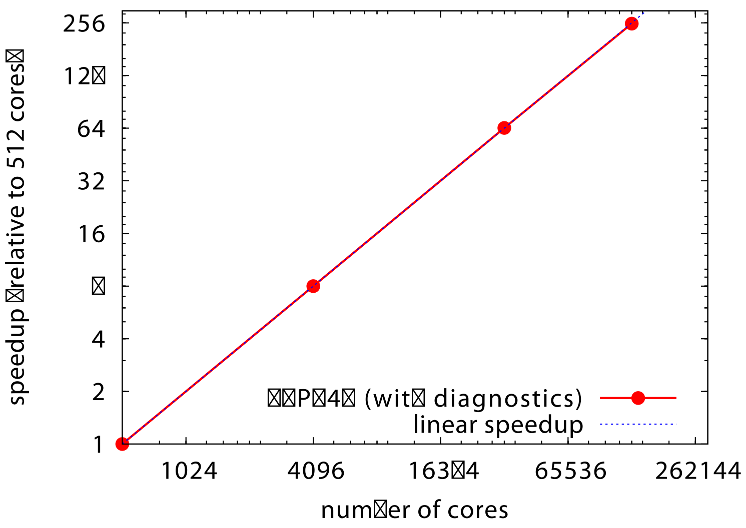


Figure 3: Weak scaling of HYPO4D on JUGENE

Acknowledgments

Besides support from the experts from JSC and IBM present at the workshop we acknowledge fruitful discussions with our colleagues Luis Fazendeiro, Stefan Frijters, Sebastian Schmieschek, and Florian Günther. Additional computing resources on JUGENE were granted by PRACE (project `pra036`, PI F. Toschi). Our visit was partially funded by the European MAPPER project (FP7:RI-261507).

References

[1] R.S. Saksena and P.V. Coveney. Rheological response and dynamics of the amphiphilic diamond phase from kinetic lattice-Boltzmann simulations. *Proc. R. Soc. A*, 465:1977, 2009.

[2] F. Janoschek, F. Toschi, and J. Harting. Simplified particulate model for coarse-grained hemodynamics simulations. *Phys. Rev. E*, 82:056710, 2010.

[3] F. Jansen and J. Harting. From bijels to pickering emulsions: a lattice Boltzmann study. *Phys. Rev. E*, in press, 2011. arXiv:1004.4414.

[4] Ditza Auerbach, Predrag Cvitanović, Jean-Pierre Eckmann, Gemunu Gunaratne, and Itamar Procaccia. Exploring chaotic motion through periodic orbits. *Phys. Rev. Lett.*, 58:2387, 1987.

Scalability Test of $\mu\varphi$ and the Parallel Algebraic Multigrid solver of DUNE-ISTL

Olaf Ippisch and Markus Blatt
Interdisciplinary Center for Scientific Computing
Heidelberg University, Germany

Description of the Code

Solution of Richards' Equation with $\mu\varphi$

Our objective was a scalability test of the application $\mu\varphi$ (MuPhi) to solve Richards' equation

$$\frac{\partial\theta(\psi_m,\vec{x})}{\partial t} \quad - \quad \nabla\cdot\{K(\theta,\vec{x})\cdot[\nabla\psi_m - \rho_w g\vec{e_z}]\} + r_w = 0, \tag{1}$$

where ψ_m is the matrix potential, ρ_w the density of water, g gravitational acceleration, $\vec{e}_z$ the unity vector in the vertical and r_w a source sink term. $\theta(\psi_m,\vec{x})$ is the volumetric water content and $K(\theta,\vec{x})$ is the hydraulic conductivity function. Both are highly nonlinear, spatially heterogeneous material functions of ψ_m.

Richards' equation is a second order partial differential equation (PDE) describing water flow in partially water saturated porous media. For time dependent problems Richards' equation is a non-linear (probably degenerated) parabolic equation. For steady-state saturated porous media it is an elliptic PDE (θ and K are then spatially variable constants).

Richards' equation is solved using a cell-centred finite-volume scheme with full up-winding in space and an implicit Euler scheme in time. Linearisation of the nonlinear equations is done by an inexact Newton method with line search.

Algebraic Multigrid as Parallel Linear Solver

Most of the computation time is consumed by solving the sparse linear systems arising in the Newton method. For this we use the biconjugate gradient stabilized method [7] preconditioned by a massively parallel algebraic multigrid solver based on aggregation [1].

We decompose the unknowns between the processes such that each unknown is owned by exactly one process. Each process stores whole rows of the matrix corresponding to unknowns owned by the process. Additionally each process has to store all unknowns j with $a_{ij} \neq 0$ for an unknown i it owns. All rows corresponding to rows not owned are set to $a_{jj} = 1$ and $a_{ij} = 0$ for $j \neq i$. Note that for the matrix-vector product we are able to compute the unknowns owned by a process with a local matrix-vector product. For a more detailed description see [2].

During the setup phase each process computes the aggregates for the unknowns it owns. After communicating this information it sets up the prolongation and coarse level matrix locally. Once the number of unknowns is below a given threshold we agglomerate the data onto fewer processes for a better ratio between communication and computation time. As a side effect we thus facilitate aggregation across process boundaries. The new data decomposition is computed by ParMETIS [5] using the graph of the communication pattern for the matrix-vector product. Unfortunately the parallel algorithm of ParMETIS cannot handle our graph on the full machine. Therefore we have to use the recursive bisection graph partitioning algorithm sequentially on one process. This procedure is repeated on coarser levels until the whole linear system of the coarsest level is stored on one process and is directly solved using SuperLU [3]. As a smoother we use hybrid Gauss-Seidel [9].

Test Cases

We formulated two test cases for a groundwater flow problem and for a (time dependent) unsaturated flow problem, respectively. For the test cases the material parameters were either

- homogeneous (thus for the groundwater flow problem the Laplace equation was solved), scenario homog
- heterogeneous with a heterogeneity which was the same on each node, scenario block
- heterogeneous with a structure which was much larger, scenario large

The heterogeneity was created as an equally weighted sum of two log-normal autocorrelated Gaussian random fields with different correlation length (100 voxel horizontally and 20 voxel vertically for the coarse scale and 2 voxel for the fine scale). The random numbers were generated with the Quantim image processing library[8] and had a mean of 0 and a variance of 1.5 for the groundwater test case and 1.0 for the unsaturated test case.

The hydraulic parameters were scaled with the exponential of these values according to the principle of Miller similarity[6]. A van Genuchten/Mualem model was used for the basic curve with the parameters of a medium sand:

Parameter	θ_s	θ_r	K_s	n	α	τ
Value	0.34	0.0	40.0 cm/h	2.0	5.0	0.5

For the groundwater test case Dirichlet boundary conditions were used at the west and east boundary imposing an average pressure gradient of 1 m/m. The potential at the lower edge of the east boundary was set to the size of the domain (thus the whole domain was water saturated). No-flow boundary conditions were used at all other boundaries. Initial condition was a full-saturation at hydraulic equilibrium. The size of one grid element was 0.1 m in all directions. We tested weak scalability in each step doubling the size of the domain using 80^3 grid cells per process resulting in 147 billion unknowns on 287'496 processes.

For the unsaturated test case no-flow boundary conditions were used at all side boundaries, a constant potential of zero was given at the bottom and a constant flux of 2 mm/h was applied at the top. Initial condition was hydraulic

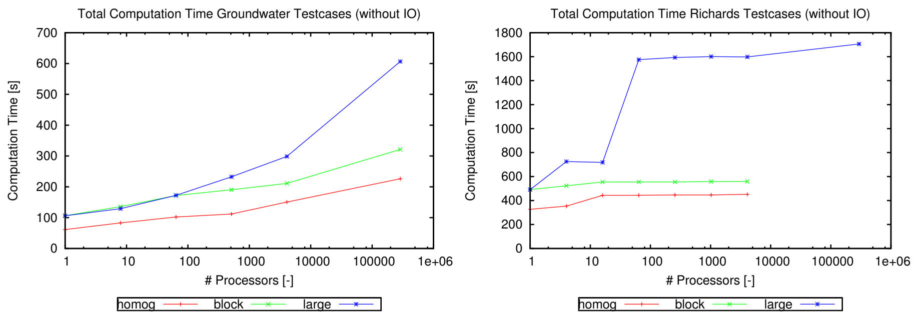


Figure 1: Total computation time without I/O for the different groundwater (left) and unsaturated (right) test scenarios.

equilibrium with a potential of 0 cm at the bottom. The size of a grid element was $0.01 \times 0.01 \times 0.01$ m. One time step of one hour was simulated. In the weak scalability test we used $64 \times 64 \times 128$ grid cells per process. The domain was only increased in the horizontal direction. Up to 294'849 processes were used.

Results

Besides performing simulations for the test cases we were able to speed up the data output (by a factor of two) and to get rid of a long delay at the start of the program resulting from dynamical linking of the system libraries. With static linking the delay vanished completely.

The linear solver with an algebraic multigrid preconditioner has an expected complexity of $O(N \log N)$, where N is the number of unknowns. The computation time (Figure 1) shows for the groundwater test case the expected linear rise in a logarithmic plot of total computation time against the number of unknowns/processes. For the large scenario there is a non-linear increase in the number of iterations needed at the beginning until the full structure (which was periodic with a length of 1024^3) is resolved. Afterwards the computation time rises again linear, but with a higher slope.

For the unsaturated test case the total computation time is even constant for all scenarios as soon as the structure is fully resolved. This is a consequence of the strict diagonal dominance of the matrix due to the additional time derivative term.

In an analysis of the computation time per step of defect calculation, matrix assembly, generation of the coarse grid hierarchy of the algebraic multigrid solver (coarsening) and application of the linear solver, respectively, most components of the code scale perfectly with a parallel efficiency close to one (Figure 2). There is a decrease in the efficiency of the linear solver per step for the unsaturated laplace scenario. However, the iteration time for the laplace case is at the beginning smaller than for all other scenarios. With increasing number of processes/unknowns the iteration time approaches the time needed in the other scenarios.

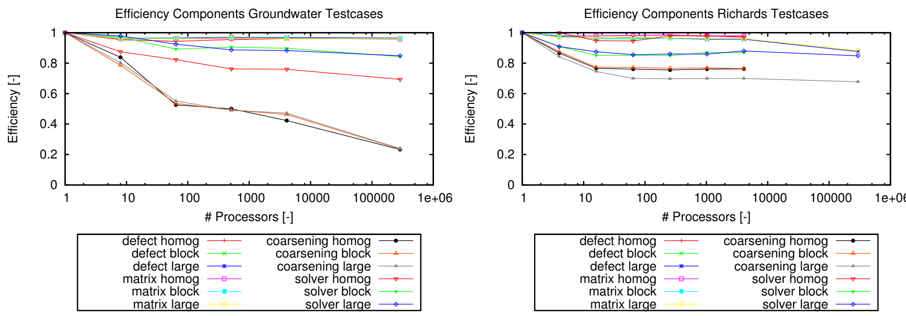


Figure 2: Efficiency per step of the different components of the code for the groundwater (left) and unsaturated (right) test scenarios.

For the groundwater test case the remaining unknowns are redistributed to fewer processes if the number of unknowns becomes too small. Finally on the coarsest level the system was solved exactly with SuperLU on one processes. The efficiency of this redistribution was difficult to achieve as ParMETIS could not be run in parallel (see above) and thus a sequential complexity was introduced. However, after an initial steep decrease it seems to scale perfectly well reaching an efficiency of 25 per cent at 287.496 processes.

For the unsaturated test case the redistribution was not necessary. The coarsening was performed as long as possible and the remaining linear equation system was solved iteratively with a parallel BiCGStab solver. This resulted in a much better efficiency of the coarsening but was only possible as the flow was mainly vertical with little horizontal coupling and the unknowns in the vertical direction were always completely on one process.

The reading of structures with as many points as the unknowns in the computation (up to 150 billion) was performed with parallel HDF5 (only for the block and large scenarios). For the block scenarios the structure was read sequentially by one process and broadcasted to the other processes, for the large scenario each process read its own hyperslab. For the unsaturated large scenario this took 1559 seconds, which is not really satisfying. The efficiency for block scenarios was better (Figure 3). For the groundwater large scenario a pre-partitioned structure data in the SIONlib [4] format was used which reduced the time for structure input to 93 seconds.

Output was also written with the SIONlib library. The output time increased strongly with the number of processes. However, this was also due to the limited bandwidth of the I/O subsystem. For the unsaturated large scenario with 294'849 cores for the output of the potential 2.3 Terrabyte were written in 123 seconds, which corresponds to 19 GB/s which is close to the system limit. For the groundwater large scenario with 287'496 cores 8.8 Terrabyte of data was written (additionally including the flux field, the RT0-coefficients of the flux field and the volumetric water content) in 568 seconds (corresponding to 15 GB/s).

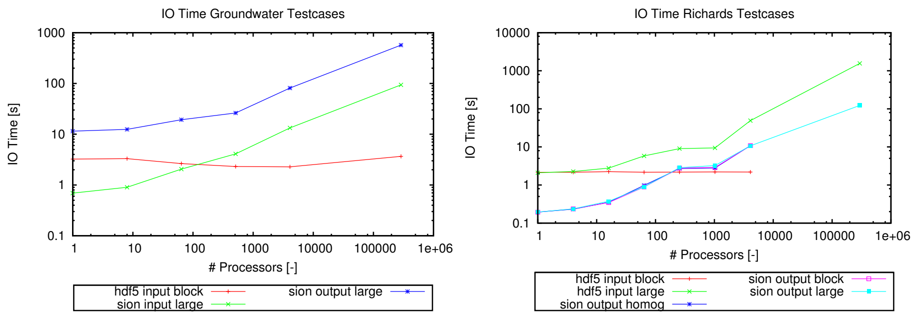


Figure 3: Time for input of structures with and the output of the results for the groundwater (left) and unsaturated (right) testcases. Please note the double-logarithmic scaling.

Crucial for the good scalability of our model and solver are

- a domain decomposition resulting in a communication pattern, which requires mostly local communications with few neighbour processes
- accumulation of communication so that only few large messages are sent instead of many small ones
- the good scalability of the algebraic multigrid solver with the number of unknowns
- the use of IO libraries really exploiting the features of the parallel file system (GPFS)

A minor bottle neck remains the sequential computation of the new data decomposition with ParMETIS as the parallel version needs too much memory.

Acknowledgements

We thank the Jülich Supercomputing Center for the possibility to take part in the workshop. This enabled us to test our code on more processors than ever and to discover and fix some scalability bottle necks.

References

[1] M. Blatt. *A Parallel Algebraic Multigrid Method for Elliptic Problems with Highly Discontinuous Coefficients*. PhD thesis, Ruprecht–Karls–Universität Heidelberg, 2010. `http://www.ub.uni-heidelberg.de/archiv/10856/`.

[2] M. Blatt and P. Bastian. On the generic parallelisation of iterative solvers for the finite element method. *Int. J. Comput. Sci. Engrg.*, 4(1):56–69, 2008.

[3] J. W. Demmel, J. R. Gilbert, and X. S. Li. An asynchronous parallel supernodal algorithm for sparse gaussian elimination. *SIAM J. Matrix Anal. Appl.*, 20:915–952, July 1999.

[4] W. Frings. Sionlib: Scalable I/O library for parallel access to task-local files. `http://www2.fz-juelich.de/jsc/sionlib`.

[5] G. Karypis and V. Kumar. A parallel algorithm for multilevel graph partitioning and sparse matrix ordering. *J. Parallel Distrib. Comput.*, 48(1):71–95, 1998.

[6] E. E. Miller and R. D. Miller. Physical theory for capillary flow phenomena. *J. Appl. Phys.*, 27:324–332, 1956.

[7] H. A. van der Vorst. BI-CGSTAB: a fast and smoothly converging variant of bi-cg for the solution of nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput.*, 13:631–644, March 1992.

[8] H.-J. Vogel. Quantim4, C/C++ library for scientific image processing. `http://www.quantim.ufz.de`.

[9] U. M. Yang. On the use of relaxation parameters in hybrid smoothers. *Numer. Linear Algebra Appl.*, 11(2–3):155–172, 2004.

Scaling of Eigenvalue Solver Dominated Simulations

Rainer Johanni, Andreas Marek and Hermann Lederer
Rechenzentrum der Max-Planck Gesellschaft, Garching

Volker Blum
Fritz-Haber Institut, Berlin

Description of the Code

FHI-aims, developed at the Fritz-Haber Institute, is an "ab initio molecular simulations package" (see `http://www.fhi-berlin.mpg.de/aims/`) which has been designed to compute physical and chemical properties of molecules and condensed matter from quantum-mechanical first principles. With numerically tabulated atom-centered orbitals as the quantum mechanical basis set it employs the density functional theory to calculate the electronic structure of the atoms. This ansatz allows for calculating with moderate computational costs an accurate all-electron/full-potential model without introducing approximations and simplifications as e.g. pseudopotentials, see [1].

The calculation of molecules as well as periodic geometries (like metallic surfaces) with FHI-aims requires the computation of eigenvalues and eigenvectors. For very large problems (e.g. of matrices of the size 67000 for a slab of 1000 platinum atoms) this can become computationally dominant, and also a parallel scalability limitation of the FHI-aims package.

Driven by this bottleneck of FHI-aims a new eigensolver was developed within the ELPA project (see [2] and `http://elpa.rzg.mpg.de`), which is designed to achieve good scaling on several thousands cores. For details the reader is referred to [2].

During the Bluegene/P scaling workshop we intended on the one hand to evaluate the principal scaling capabilities of the isolated ELPA eigenvalue solver for large sparse matrices as they are used in the simulations of biological and technical networks. On the other hand we wanted to investigate the scalability of the complete FHI-aims application with the ELPA eigenvalue solver incorporated on real scientific problems from material and life sciences.

Results

In the following we will describe the measurements done during the Bluegene/P Scaling Workshop and the conclusions we draw from these experiments.

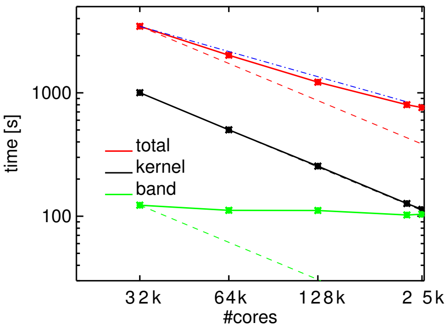


Figure 1: The strong scaling of the isolated eigenvalue solver for a matrix of size 260k. Shown are the total solver time (red), the time of the kernel (black), and the time for transformation of a band matrix to tri-diagonal form (green). The dashed lines give the theoretical perfect linear scaling. Note that the kernel part scales perfectly. As a guideline the blue dashed-dotted line shows the limit for a scaling of $t \propto 1/(1.6)^n$.

Eigenvalue solver in biological and technical networks

As already mentioned, we planned to investigate the principal scaling capabilities of the newly developed eigenvalue solver. For this purpose we did not use matrices as they are used in FHI-aims, but instead we used matrices from graph theory: For the simulation of biological and technical networks based on graph theory sparse matrices of size 1 million and larger will play a role. To be able to measure strong scaling with at least four data points, the matrix size was reduced to 260000, in order to fit into the memory of the Bluegene/P. This fixed problem size was measured with 32k, 64k, 128k, 256k, and 294k cores (full machine) and we obtained up to 0.3 Petaflop/s on the full Bluegene/P system.

Figure 1 displays the scaling of essential parts of the solver. The kernel scales excellent up to the full machine, while the transformation from banded to tri-diagonal form of the matrix does not improve, since the scalability limit for this step is given by matrix size (here 260k) divided by the matrix bandwidth (here 64) which is in this case 4000 cores. As a result the total solver scalability is limited to $t_{n+1} = t_n/1.6$.

Scalability of FHI-aims

Scalability of the full FHI-aims application was tested with real science cases from biology and solid state physics: a polyalanine molecule with 200 atoms (referred as Ala200) and a platinum surface with 1442 atoms in seven layers (referred as Pt1442).

Both science cases need relatively little main memory and can thus already be calculated on 1000cores (case Ala200) or 8000 cores (case Pt1442), respectively.

Small core numbers (up to 16k cores) were measured at the Bluegene/P of RZG, Garching. The data points from 32k cores on during the workshop at FZJ. All runs were done in "virtual node mode", since FHI-aims does not support hybrid parallelization.

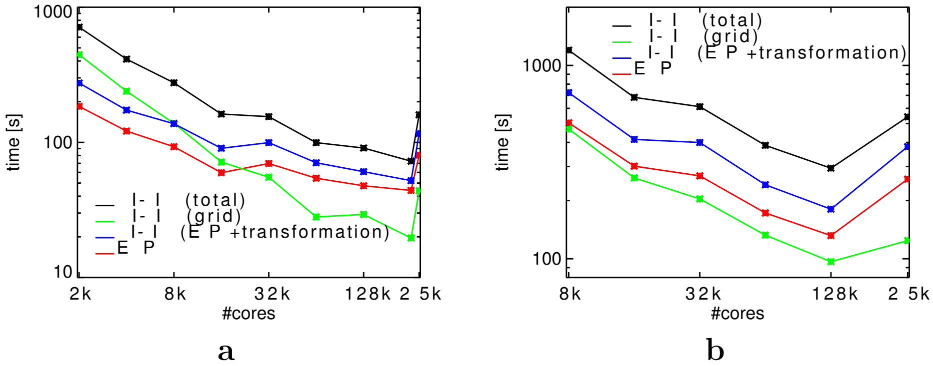


Figure 2: The scaling behaviour of the FHI-aims code for the total time (black), the grid-based part (green), and the eigenvalue solver with the Cholesky transformation (blue) for the computation of different setups (left and right panel). The scaling behaviour of the ELPA solver (red lines) is also shown. **a**: The case of the Ala200 setup. **b**: The case of the Pt1442 setup.

The participation in the workshop allowed us to identify some bottlenecks in the initialization of the FHI-aims code which needed $\mathcal{O}(\text{ncpus}^2)$ work and showed up only when the number of employed cores exceeded more than 16000. These bottlenecks could be corrected during the workshop and then we were able to run all setups to the full machine size of 72 racks (i.e. 294912 cores), although the runs on 72 racks cannot be taken seriously because of machine problems. For the Ala200 as well the Pt1442 setup we could also obtain results for the intermediate core numbers (16k, 32k, 64k, 128k). Unfortunately we were able to obtain results on 256k cores for the Ala200 case only but not for the Pt1442 setup due to an unexpected lack of machine availability.

The total cycle time of FHI-aims scales fairly well from 1k to 16k cores for Ala200 and up to 128k cores for Pt1442. The eigenvalue solver behaves similarly, its potential for larger matrices has already been demonstrated in Section . The data points for 72 racks should be disregarded since these results can only be explained with machine problems. For the Ala200 case execution time still reduces from 128k to 256k cores, what we also have to assume for the Pt1442 case with much larger problem size.

Based on the timing results achieved during the workshop, the analysis of the other parts of FHI-aims has started and the optimization potential is being evaluated. This new level of the performance data of the FHI-aims code would not have been possible without the participation at the workshop.

Conclusions

The scaling workshop has enabled us to demonstrate the scalability potential of the newly developed ELPA eigenvalue solver both with the electron structure code FHI-aims with biological and solid state science cases, as well as for very large matrices needed for the simulation of biological and technical networks.

Acknowledgments

We thank the Jülich Supercomputing Center and IBM for hosting this project during the Scaling Workshop and for their competent support. The ELPA solver tested during the workshop was developed by support of the BMBF grant 01IH08007.

References

[1] Blum V., Gehrke R., Hanke F., Havu P., Havu V., Ren X., Reuter K. & Scheffler, M.: "Ab initio molecular simulations with numeric atom-centered orbitals", *Computer Physics Communications* **180**, page 2175-2196, (2009)

[2] Auckenthaler T., Blum V., Bungartz H.-J., Huckle T., Johanni R., Krämer L., Lang B., Lederer H. & Willems P. R.: "Parallel solution of partial symmetric eigenvalue problems from electronic structure calculations", (preprint available at `http://www.fhi-berlin.mpg.de/th/publications/`)

Parallel Simulations of Quantum and Frustration Effects in Molecular-based Nanomagnets

Michał Antkowiak, Łukasz Kucharski, Piotr Kozłowski, Grzegorz Kamieniarz
Adam Mickiewicz University, Poznań, Poland

Ryszard Matysiak
University of Zielona Góra, Poland

Description of the Code

Our code has been designed to calculate thermodynamic properties of spin rings and chains, which can be modeled by an anisotropic quantum Heisenberg model. In particular we use this code to simulate molecular nanomagnets. The thermodynamic quantities are calculated by means of Quantum Transfer Matrix (QTM) method. Then the Transfer Matrix (TM) method is used to deal with the resulting classical system [1, 2].

We use such a version of QTM, which does not require diagonalisation of large matrices. The computation finally reduces to calculation of a number of traces calculated by subsequent matrix vector multiplications. The problem is expressed in such way that the parallelization is trivial. Each diagonal element of the final matrix can be calculated separately. The communication takes place only at the beginning and the end of the process. The only limiting factors to the efficiency are maintaining even distribution of the work among the computing cores and the size of the problem itself.

Size of the problem is the number N of elements to calculate and is defined directly by the size of the matrix. It is determined by the number and the type of atoms the simulated molecule consists of. The two application we have tested calculated problems of sizes $N = 4^{11}$ for Application 1 and $N = 12^6$ for Application 2. Of course such large matrices would not fit into memory limits for Blue Gene/P systems. The memory footprint was reduced to acceptable values by exploiting the sparseness of the matrices. In fact only a few vectors of size N are needed to remember. Furthermore, using the same mechanisms the computational complexity for calculating a single element was also reduced from matrix-vector multiplications to vector-vector operations.

Results

We have performed a number of successful large scale runs and proven our initial predictions that they will scale very well. Application 1 was tested thoroughly. Runs were performed from 8 racks up to the full machine (details in Table 1). One element of the matrix was calculated for a fixed time and the more cores computed whole task, the less each had to calculate on its own. Given this it is possible to theoretically predict speed up and efficiency as a function of computing tasks. This was done for Application 1 and presented in Figure 1. This application maintained perfect linear strong scalability when the work distribution was optimal.

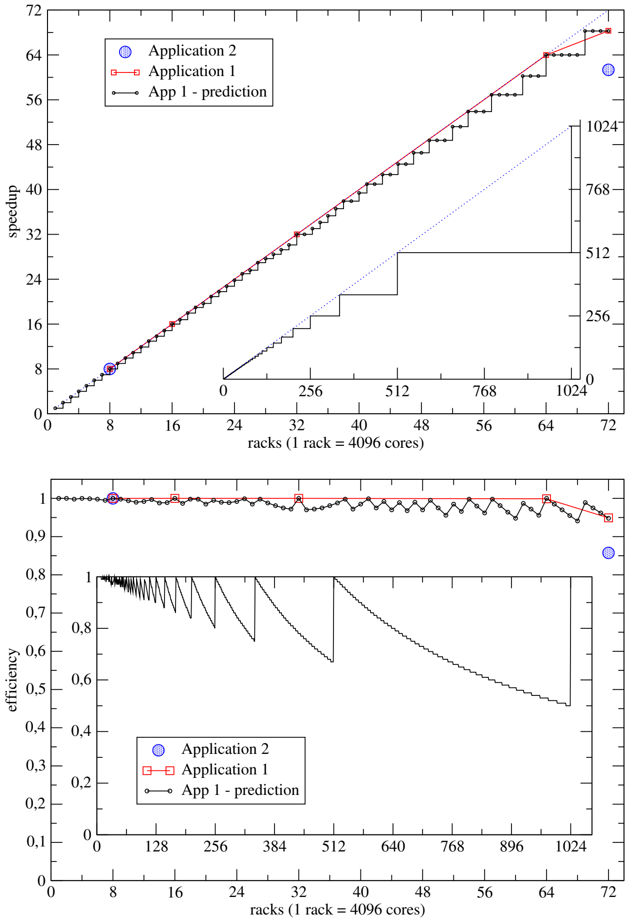


Figure 1: Strong scaling plots for both applications. Theoretical prediction made for Application 1 match the actual data very well. The insets picture the expected values calculated up to 1024 racks (4 194 304 cores).

Since Application 2 is a younger piece of work, encouraged by the success of Application 1 we had tried to compile Application 2 with performance measurement mechanisms to obtain FLOPS data. The idea backfired due to technical problems with running dynamically linked applications experienced on JUEGENE. The program failed 3 out of 5 runs. However, we managed to perform two runs on 8 racks and the full machine. With the 8 racks as a reference point the full machine run had efficiency of 86% cutting the total run time from 804s to 104s. Both points are presented on the Figure 1. Because both applications are based on the same mathematical framework we expected the same scaling

Racks	Cores	Elements/core	Time [s]
8	32768	128	7428
16	65536	64	3713
32	131072	32	1857
64	262144	16	929
72	294912	14-15	870

Table 1: Total running time of Application 1 for different number of racks/cores. In case of 72 racks the number of elements calculated by a single core could not be distributed equally.

results qualitatively. The unexpectedly low result comes from the fact that we did not pay enough attention to the even distribution of the work among the cores. The largest amount of cores that divides 12^6 available on JUGENE is 248 832 which is exactly 60.75 racks.

Conclusions

We have proven that our codes can be scaled up to whole JUGENE. We can maintain from very good to perfect strong scalability.

The theoretical predictions matched the actual running time for all calculated cases.

For the purpose of testing only the scalability we have simplified the physical problem. To obtain real physical values the total running time would increase up to hours. Clearly we could use several times more computing nodes.

Acknowledgments

A partial support from the MNiSW grant Nr 579138 is acknowledged.

References

[1] G. Kamieniarz et al., Inorganica Chimica Acta **361**, 3690-3696 (2008)

[2] P. Kozłowski et al., Polyhedron **28**, 1852 (2009)