FORSCHUNGSZENTRUM JÜLICH GmbH
Zentralinstitut für Angewandte Mathematik
D-52425 Jülich, Tel. (02461) 61-6402

Technical Report

Report on the Jülich Blue Gene/L Scaling Workshop 2006

Wolfgang Frings, Marc-André Hermanns, Bernd Mohr, Boris Orth (Editors)

FZJ-ZAM-IB-2007-02

February 2007

(last change: 16.2.2007)

Report on the Jülich Blue Gene/L Scaling Workshop 2006

Wolfgang Frings, Marc-André Hermanns,
Bernd Mohr, Boris Orth (Eds.)

John von Neumann Institute for Computing (NIC),
Research Centre Jülich

31. January 2007

Abstract

The John von Neumann Institute for Computing (NIC), IBM, and the Blue Gene Consortium have jointly sponsored the first "Blue Gene/L Scaling Workshop" in Jülich, Germany, on December 5-7, 2006. The purpose of the workshop was to provide participants the chance to scale their codes across an 8 rack Blue Gene/L system. Besides the hardware, appropriate software and support personnel were provided to accomplish this task. Jülich provided about 800.000 CPU hours on the 8-rack Blue Gene system JUBL (`http://www.fz-juelich.de/zam/ibm-bgl`) over a three day period for the scaling runs.

The attendees of the workshop were selected by a peer review team. Selection criteria were the confidence that the code would scale across 8 racks, if the JUBL infrastructure (OS, compilers, libraries) support the user request, and the scientific impact that the code could produce. Members selected were paired up with assigned advisors from Argonne National Laboratory, IBM, and NIC, who assisted in administrative issues (log on, moving data, storing data), and scaling support.

This report presents the outcome of the workshop as a collection of the results of running, scaling and optimizing the following seven application on Blue Gene/L:

1. **Molecular Dynamics Studies of Radiation Hard Materials** (*page 2*)
   Ian J. Bush, Ilian T. Todorov, CCLRC Daresbury Laboratory, United Kingdom
2. **Thermonuclear Supernovae: Simulations of Delayed Detonation with Adaptive Mesh Refinement and Lagrangian Tracer Particles** (*page 5*)
   Anshu Dubey, Center for Astrophysical Thermonuclear Flashes, University of Chicago, USA
3. **Large scale ab initio calculations of functional materials** (*page 7*)
   Markus E. Gruner, Sanjeev K. Nayak, and Peter Entel,
   Theoretical Physics, University of Duisburg-Essen, Germany
4. **Parallel Stabilized Finite Element Methods for Aero-, Hemo- and Hydrodynamics** (*page 9*)
   Mike Nicolai, Markus Probst and Marek Behr,
   Computational Analysis of Technical Systems, RWTH Aachen University
5. **Development of Scalable Software Infrastructure on Blue Gene Systems** (*page 13*)
   Akira Nishida, Chuo University, Japan
   Hisashi Kotakemori, Akira Nukada, Japan Science and Technology Agency, Japan
   Akihiro Fujii, Kogakuin University, Japan
6. **Turbulent convection for very large aspect ratios** (*page 15*)
   Jörg Schumacher, Department of Mechanical Engineering, Technische Universität Ilmenau,
   Matthias Pütz, Deep Computing – Strategic Growth Business, IBM
7. **Numerical simulations of QCD** (*page 19*)
   Hinnerk Stüben, Konrad-Zuse-Zentrum für Informationstechnik Berlin, Germany,
   Thomas Streuer, Department of Physics and Astronomy, University of Kentucky, Lexington, USA

1 Molecular Dynamics Studies of Radiation Hard Materials

Ian J. Bush, Ilian T. Todorov
CCLRC Daresbury Laboratory, Keckwick Lane, Daresbury, Cheshire, WA4 4AD, UK

The portability and performance of DL_POLY3 on Blue Gene (BG/L) has been examined at the Scaling Workshop. DL_POLY3 is a classical molecular dynamics package developed at CCLRC Daresbury Laboratory [1]. It is a very widely used application, with a few thousand licenses being held worldwide, and may be used to study a very wide range of systems due to the flexibility of the force field that it supports. However, the code has never been run before on systems with appreciably more than 1000 processors.

The system chosen for the workshop was a model of radiation damage in a fluoritized Zirconium pyrochlore. One of the native Gadolinium ions in the system was replaced by Uranium, which was then given a velocity consistent with a 100 keV recoil after an alpha decay. Due to the very high velocity of the Uranium ion it is necessary to study very large supercells, and the total system size we use is approximately 14.6 million particles. It should be noted that this is the first attempt to model this system with a realistic recoil, previous work having been carried out at appreciably lower energies [2,3,4]. Since both the required size of system and the number of timesteps increases with the recoil energy it is only on machines with power comparable to the Jülich BG/L that these calculations may be performed.

It was hoped that production runs could be performed as well as benchmarks. However, due to time constraints and unforeseen problems around the large I/O demands of the code for systems of such size, this proved not to be possible.

1.1 DL_POLY3

DL_POLY3 is a totally distributed memory code. The scaling of the time to solution depends slightly on the force field employed, but it is always approximately O(N). To achieve both the time and memory scaling a link-cell algorithm [5] is used, which is essentially a domain decomposition method. The force field used for the above simulations is relatively simple, but is not trivial. The various terms can be generalised as

1. Short range repulsion
2. Van Der Waal's (VDW) attraction
3. Coulomb forces

1 and 2 are both short range terms, and are handled together in DL_POLY3. As such subsequent references to VDW terms should be understood to include both these terms. Due to the short range of these forces they should scale very well with processor count due to their spatial locality (compare halo exchange algorithms). Previous experience suggested that for the test case the scaling of the VDW terms should begin to fall away from perfect only when using the full size of the machine (16384 processors).

On the other hand Coulomb forces are long range terms, and have to be handled differently. As is standard, the Ewald sum technique is used in DL_POLY3. This splits the evaluation into two terms – one short ranged, one long ranged. The former can be handled in a very similar way to the VDW terms, and hence is evaluated in real space. The long range term, however, has to be handled differently. DL_POLY3 uses the Smooth Particle Mesh Ewald (SPME) algorithm [6], the key feature of which is a Fast Fourier Transform (FFT). For this DaFT is used, a package written at Daresbury. This has some features in common with the various volumetric transforms that have been developed, but is novel in that it avoids performing 'all-to-all' operations by parallelising the individual 1D FFTs [7].

The use of an FFT implies a considerable amount of communication, so one would expect that the scaling is ultimately controlled by this portion of the code. One of the main reasons for our attendance at the workshop was to examine this.

1.2 Porting DL_POLY3 to BG/L

As the code is written in standard conforming Fortran and MPI no major problems were experienced in porting DL_POLY3 to BG/L. The only issue that stopped the code running "as is" was that at certain points in the code very large numbers of messages were outstanding, causing the machine to run out of memory due to the number of buffers required. This was very easily solved by introducing some handshaking to cut down on the number of outstanding messages. This problem had not been observed before due to the smaller number of processors on which the code had been run.

1.3 Results

Once ported, excepting one problem, described below, the code ran and scaled very well "out of the box". The scaling for MD of the test system described above is shown below. The figure 1 shows speed-up values which are calculated relative to 2048 processors as below that insufficient memory is available to run the simulation. Note that the use of speed-up obscures the absolute time - clearly some components are more important than others - for this see the table below. All jobs were run in virtual node mode.

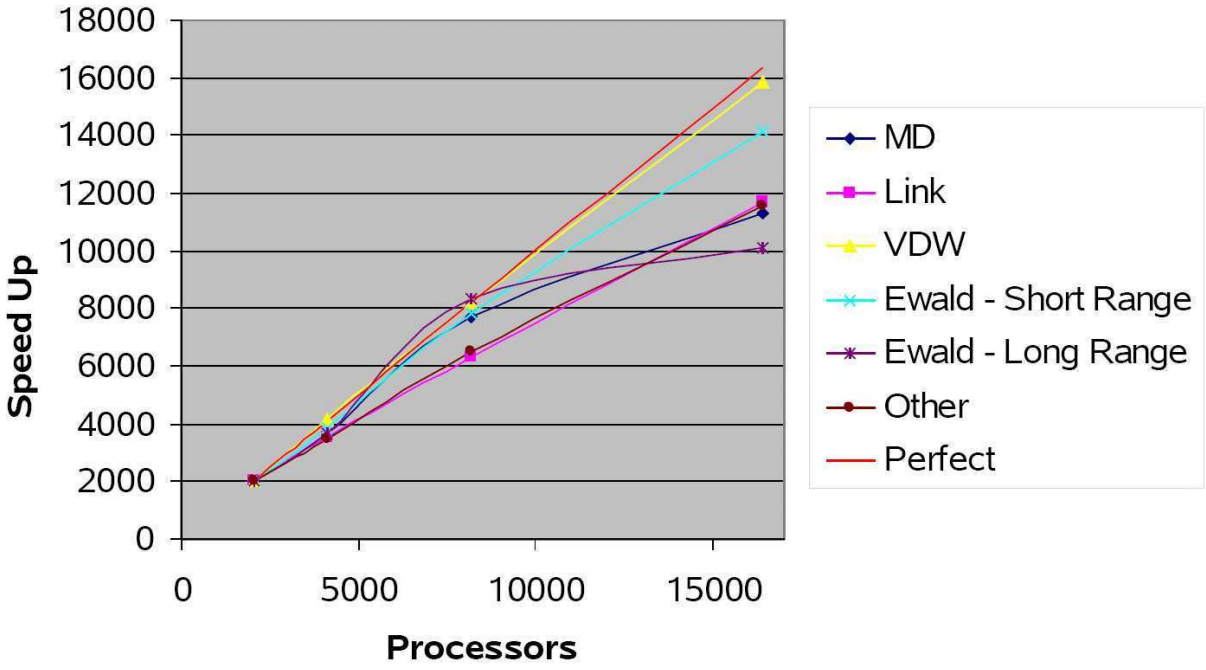

Figure 1: ***Scaling of DL_POLY3 on BG/L***: "MD" shows the scaling of the total computation time, "Link" refers the time taken at each time step to implement the link-cell algorithm (and build the Verlet neighbour list) and the remainder are the components of the force field terms outlined above.

It can be seen that the scaling for the various elements of the force field is good. The VDW and short range Ewald terms both scale almost perfectly, and at least for VDW terms the expected deviation from ideal behaviour at 16384 processors is not very apparent.

As expected the long range Ewald terms, i.e. those terms that require an FFT, scale less well. However, given the comparatively small size of the FFT grid, 512*512*512, the scaling is still good.

The scaling of the time spent in implementing the link-cells and the other parts of the calculation is curious. There is very little deviation from straight line behaviour, but this line is less than perfect. The reason for this requires more investigation.

Number of Processors	MD	Link	VDW	Ewald Short Range	Ewald Long Range	Other
2048	2.70	0.383	0.333	0.256	1.333	0.389
4096	1.50	0.220	0.163	0.136	0.751	0.230
8192	0.72	0.124	0.083	0.067	0.327	0.123
16384	0.49	0.067	0.043	0.037	0.270	0.069

The table above reports the time per timestep for each of the previously mentioned components of the execution. It can be seen that, at these processor counts and for this system, the dominant term is the long range component of the Ewald summation. Whilst not totally unexpected the margin by which this dominates is surprising, as for smaller systems it was found that the time taken for this term was roughly comparable with that for the VDW and short ranged Ewald terms. The behaviour is probably a reflection of the system size scaling of the long range Ewald term being O(NLog(N)), as compared to O(N) for the other sections.

The most important time in the above table is that for an MD timestep on 16384 processors. This is sufficiently small to allow full simulations to be performed in a realistic amount of time, or to put it another way, the code runs fast enough to allow science to be done. This must be the ultimate criterion of performance!

Xprofiler and explicit timings were used to examine the performance of the code that implements the long range Ewald terms. It was found that most of the time taken is in just two areas:
1. The loop nest that interpolates the ionic charges onto the regular grid
2. The message passing time in the FFT
The latter may be extracted by taking the difference between the measured times and those reported by Xprofiler, since this tool reports only CPU time. These are two very short sections of code and are obvious candidates for optimisation.

The one major problem that was experienced was I/O. Reading the input file took 10 minutes, and dumping the final results $1/2$ hour. While these are not too bad, as each has only to be done once, it was found impossible to perform periodic dumping of the atomic coordinates. To give an idea of how bad this problem is it takes about 4 minutes to perform 500 time steps, and 10 minutes to dump the coordinates. As a full simulation would take over 70,000 timesteps and require coordinate dumping every 500 timesteps it is clear that the total time taken would be prohibitive. As such in all the above figures the periodic dumping of coordinates has not been included, and the time reported is only for the MD steps, not the initialization and finalization.

The reason for this bottleneck is probably in the way I/O is implemented in `DL_POLY3` rather than the Blue Gene I/O subsystem, at least at present. In `DL_POLY3` all I/O is performed
1. in serial, i.e. all through one processor
2. to/from formatted files
The reasons that it is done in such a simple way are simplicity and portability of the files, and up until now the time taken for I/O has just not been an issue in the development of the code as the time taken has been small compared to the compute time. However it is now very clear that if such system sizes are to be regularly simulated the I/O performance must be investigated.

1.4 Summary and Future Work

DL_POLY3 has been shown to scale well out to 16384 processors on BG/L. Porting was very straightforward, and the code scaled very well "out of the box". It was shown that the code runs fast enough on 16384 processors to allow a detailed scientific study of the system were time permitting. The one major problem was I/O. Using a mixture of code instrumentation and Xprofiler the three main areas for future work to improve the scaling and performance are

1. As mentioned above, I/O
2. The interpolation of the ionic charges onto the regular grid
3. The message passing in the FFT

1.5 References

[1] Todorov I.T. and Smith W, 2004, Phil Trans R Soc Lond, A 362, 1835

[2] Todorov I.T., Purton, JA, Allan, NL, Dove M.T 2006 J. Phys. Cond. Mat. 18, 2217

[3] E.g., Trachenko K 2004 J. Phys. Condens. Matter 16 R1491 and references therein

[4] Trachenko K, Pruneda J M, Artacho E and Dove M T 2005 Phys. Rev. B 71 184104

[5] M.R.S. Pinches, D. Tildesley, W. Smith, 1991, Mol Simulation, 6, 51

[6] U. Essmann, L. Perera, M.L. Berkowtz, T. Darden, H. Lee and L.G. Pedersen, 1995, J Chem Phys, 103, 8577

[7] Bush I.J., Todorov I.T., Smith W, Comp. Phys. Commun., 175, 323

2 Thermonuclear Supernovae: Simulations of Delayed Detonation with Adaptive Mesh Refinement and Lagrangian Tracer Particles

Anshu Dubey
Center for Astrophysical Thermonuclear Flashes,
The University of Chicago, Chicago, IL 60637

The FLASH code was one of the early applications on the BG/L architecture, so there is a relatively long history of the Flash center's interaction with the BG/L machines. A couple of production runs had been done with FLASH on large installations prior to the workshop. One of them was to investigate a denotation mechanism in Type Ia Supernovae in two dimensions, called Gravitationally Confined Detonation (GCD) proposed by the Flash Center. This run used Adaptive Mesh Refinement (AMR) to optimize computing and memory resources, and was found to scale very well on the machine. The second production run was that of a three dimensional driven turbulence problem, using the Uniform Grid. This simulation also scaled very well, but that is expected from Uniform Grids. However, neither of the runs had exploited the full complexity of the FLASH code as applied to problem of primary interest in the center. The scientists at the center are currently most interested in investigating the GCD mechanism in three dimensions, which adds several layers of complexity to the two runs described above. AMR is more expensive, and potentially less scalable than the Uniform Grid, and the memory requirements in 3-D are significantly higher. Since the Flash Center has applied for a significant amount of computing time on the next generation Blue Gene machine, we were highly motivated to participate in the scaling workshop. It would either verify for us that the code scales in more demanding situations, or give us early warning of potential problems.

2.1 Experience

The workshop proved to be extremely useful for the Flash Center, even though the code did not scale as expected. The workshop helped identify roadblocks and bottlenecks that FLASH can expect to face in future BGL machines. We found out that the our current AMR package PARAMESH needs significant optimization in its transient memory usage, otherwise it severely limits the problem size with small memory. We also identified an IO performance bottleneck with the parallel IO library Parallel-NetCDF; one of two parallel IO libraries supported by FLASH. The cause for the performance bottleneck was found during the course of the workshop, and a temporary workaround proved to be very successful. We were able to do several experiments to test the scaling of individual components of FLASH using simpler problem setups. Some of these components included the cost of regridding the mesh when the refinement changes, global all-reduce operations in some of the physics modules in the code, and performance comparison between managing the mesh with or without permanent guard cells. Most of these experiments were instrumented with TAU.

The early detection of potential bottlenecks is extremely useful to us, since that gives us time to address them without the pressure of an imminent production schedule. The workshop also helped us connect with the research efforts at Argonne National Laboratory in single PE performance optimization, which is an important concern in a few of our physical solvers. Since FLASH is a public domain code with a fairly large user base, platform specific optimizations cannot be applied to it. The current trend in using code transformation tools to optimize for a specific platform is the best option for FLASH.

2.2 Conclusions

Participation in the Scaling Workshop has proven to be of immense value to the FLASH center, not in terms of demonstrating the scaling of the code, which it does for simple problems, but for early identification of potential trouple spots. The memory bottleneck in 3D GCD simulations persuaded our PARAMESH colloborators to re-examine the communication machinery in their package. They have, since then, significantly reduced the memory footprint of PARAMESH, which has also proven to be useful on other platforms. During the workshop there was quick and easy access to expertise, and the staff was exceptionally helpful. Several smaller hurdles in IO, and using profilers, were solved very rapidly and we were able to collect meaningful data about the general code performance. The experiments helped identify one design issue in Parallel-NetCDF, which was reported to the developers of the library. It also lead to FLASH developers using Parallel-NetCDF more effectively.

3 Large scale ab initio calculations of functional materials

Markus E. Gruner, Sanjeev K. Nayak, and Peter Entel
Theoretical Physics, University of Duisburg-Essen

3.1 Overview

Aim of this project was to explore the limits for the employment of Density Functional Theory (DFT) codes on massively parallel computers. Ab initio DFT codes have become a widely used tool for the explanation of unusual materials properties and the prediction of novel functional materials. Unlike empirical and semi-empirical methods their accuracy is not hampered by simplifying assumptions about the interatomic interactions, so that their use becomes inevitable where electronic and structural properties become closely interrelated as it is often the case for modern functional materials. Off-the-shelf DFT codes like VASP (Vienna Ab initio Simulation Package, `http://cms.mpi.univie.ac.at/vasp`) were historically not designed to run large problems on a vast number of processors, but a redevelopment with this goal in mind appears infeasible due to their complexity. So, the readiness of existing codes for modern massively parallel supercomputer architectures is pivotal for the question whether *ab initio* materials science on the mesoscopic scale will become possible in near future.

3.2 Test Cases

In previous tests, we could show that the code scales well on up to 1024 nodes. However, further scaling was hampered by parts of the code which make heavy use of the scalapack eigensolver `pdsyevx` and the parallel 3d fast Fourier transforms (FFT). One strategy to circumvent this limitation is the tackling of larger systems, where the relative amount of time spent for interprocess communication will be reduced. The option to optimize the communication structure in the code was not projected at this stage, but may become a target for future efforts. Therefore, three questions were at the center of interest during the scaling workshop:

1. Is the VASP code capable of running on 8k nodes?
2. How severe is the limitation of 512MB/node for larger problems on several thousand nodes?
3. And connected to the last point: What is the maximum system size that can be tackled on the Blue Gene/L?

To answer these questions, we set up test calculations for realistic test systems. The first consisted of super-cells of the magnetic shape alloy Ni_2MnGa of various sizes (576, 672, 720, 768, 800, 896 and 1024 atoms) large enough to contain a martensitic twin boundary. These twin boundaries can be shifted in realistic magnetic fields which gives rise to the so called ferromagnetic shape memory effect, which makes these alloys interesting for a new class of magnetomechanical actuators. We also performed calculations of large super-cells of the dilute magnetic semiconductor ZnO:Co and GaN:Gd. Three different system sizes were categorized consisting of a total number of 432, 864 and 1296 atoms. In both cases, due to the large super-cells, integration in k-space was restricted to the Γ-point.

3.3 Results

During our tests it became evident that the constrained memory of only 512MB/node is a severe limitation and that much better scaling may be achieved on Blue Gene/L installations with 1GB/node. Consequently, we were forced to use the Coprocessor mode throughout all of our calculations. Another drawback was the lengthy initialization procedure of the VASP code, which takes up to 30min for the largest problems. This does not affect production runs, since the current scheduling system allows runs for up to 24 h, but is inconvenient for test runs, which yield appropriate timing information (and maximum memory allocation) only after the initialization stage.

The largest system we could run on one and two racks was a Ni_2MnGa super-cell consisting of 672 atoms (6720 spin-polarized valence electrons). Here, we could achieve a speedup of 1.73 between one and two racks, indicating that this system can run on two racks with reasonable efficiency.

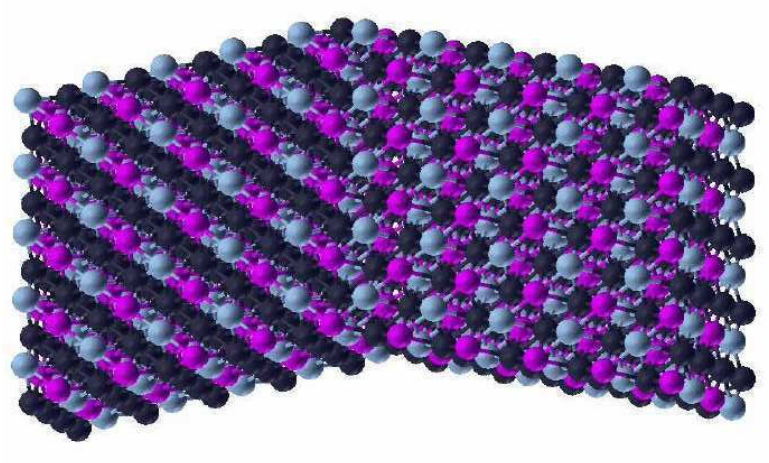

Figure 2: ***Ni_2MnGa super-cell with 800 atoms and a twin boundary as used in the calculations (black for Ni, blue and magenta for Ga and Mn, respectively).***

The largest system that could be successfully tested on 8 racks was an 800 atom Ni_2MnGa supercell comprising 8000 spin-polarized valence electrons (see figure). Here, we achieved a few selfconsistency steps, which gave enough data for a timing analysis. On a cube (2 x 2 racks) this problem was brought to complete self-consistency during the night testing time, proving that the VASP code can be used for scientific calculations on several thousands of processors. The measured speedup between four and eight racks, however, was only 1.22. Reasonable timing data could not be obtained for two racks and less due to memory restrictions. However, for a previous test case (561 iron atoms), which was not hampered by memory limitations down to 128 nodes, we achieved a speedup of 1.31 between one and two racks, while measuring on 1024 nodes an efficiency of 70% of the ideal performance extrapolated from 128 nodes. So, it appears plausible to expect an overall efficiency of above 50% for the 800 atom super-cell on the cube. For the doped ZnO the largest system calculated consisted of 864 (7725 valence electrons) atoms which could be computed on one rack.

Based on our experience during the week we conclude that on a Blue Gene/L system with 512MB/node the maximum partition that can be used efficiently consists of four racks. The largest problems that were manageable with VASP on such an installation contained 800- 900 atoms and up to 8000 (spin-polarized) valence electrons. On Blue Gene/L systems with 1GB/node larger systems can be treated and efficient scaling will probably be achieved even on 8 or 16 racks.

From the results of the scaling workshop two concepts for further improvement of the scaling behavior of VASP emerged:

- Implementation of an improved memory monitoring system to find out whether the VASP code performs significant memory allocations that are not distributed efficiently over the nodes. In this case the memory consumption per processor may grow with the size of the system setting an upper limit to the system sizes which can be handled on the machine. A simple monitoring scheme, keeping track of the minimum free memory using the `sbrk(0)` system call has already been implemented and will be extended to return subroutine specific information. This may help to localize possible unbalanced memory allocations in the VASP code.
- Detailed monitoring of the communication and improved timing of the FFT and linear algebra calls. Here, it is planned in close cooperation with Dr. Pascal Vezolle of IBM to find out whether improvements in the FFT communication scheme may result in a significantly improved performance.

4 Parallel Stabilized Finite Element Methods for Aero-, Hemo- and Hydrodynamics

Mike Nicolai, Markus Probst and Marek Behr
Chair for Computational Analysis of Technical Systems
RWTH Aachen University

For our group the workshop was very successful. We could significantly improve the performance of our CFD code on JUBL for runs on 2048 and 4096 processors. The application shows an acceptable scaling up to 4096 processors now and good scaling for a larger amount of processors can be expected. This was achieved by splitting the communication into sends and receives.

4.1 The Testcase

For measuring the performance of the XNS CFD solver, a 3D space-time simulation of the MicroMed DeBakey axial blood pump (see Figure 3) was run with the MPI version of the code. The mesh for the pump consists of almost 4 million elements and is divided into subdomains that are then assigned to a single processor. This partitioning is carried out by the METIS graph partitioning package.

The equation system resulting from the finite-element (FE) formulation of the incompressible Navier-Stokes (NS) equations that describe the flow of the blood through the pump is solved with a GMRES method. We typically carried out 3 timesteps with 4 Newton-Raphson iterations in the solver.

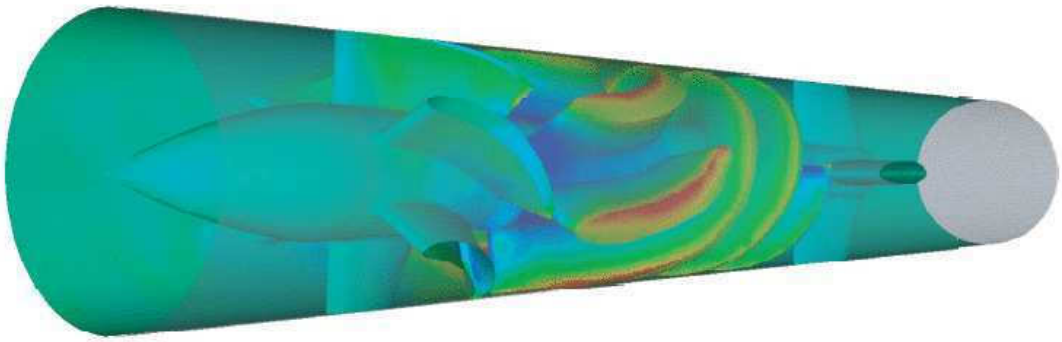

Figure 3: ***DeBakey blood pump***

Before the workshop one could observe an acceptable scaling of XNS up to 1024 processors while there was no significant speed-up above that. Here, the performance is always measured in time steps per hour to exclude the effects of time-consuming initialization or data output (see figure 5).

4.2 Analysis of the code

To find the bottleneck in the code the communication between the processes during the simulation run was analyzed, both with the XNS-internal option `debug comm` and the SCALASCA package of FZ Jülich. Later on, communication patterns were obtained by linking the `MPITRACE` library to XNS.

This analysis showed that the poor scalability resulted from an increasing number of calls to the communication routines in the EWD library and hence an increasing amount of time spent in those. More precisely, this concerned the functions `ewdgather1` and `ewdscatter2` which incorporate the transition from node-level (i.e., global) to partition-level (i.e., local on each processor), see figure 4a, and make use of `MPI_Sendrecv`.

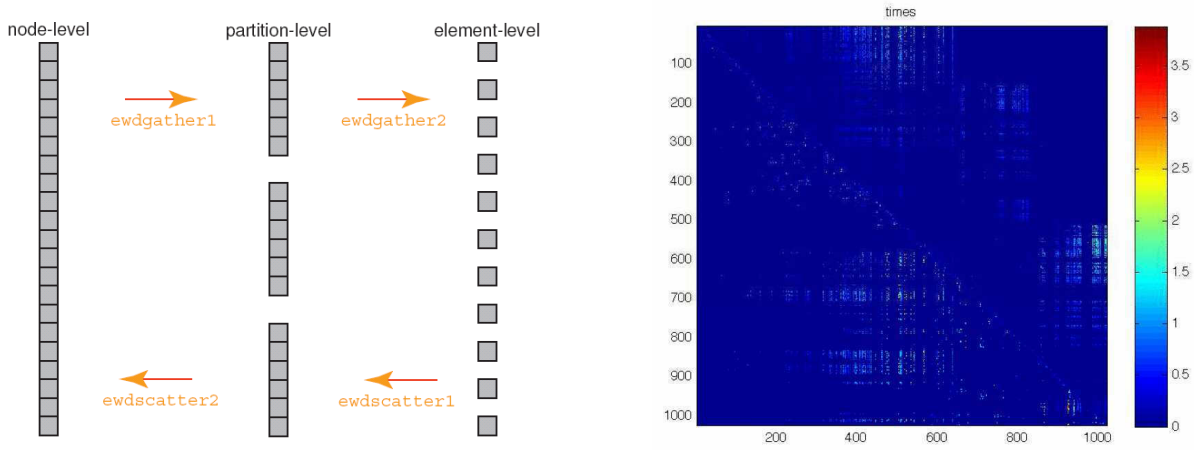

Figure 4: *a)* ***Interface between different levels*** *b)* ***Communication pattern as histogram***

A communication matrix between the processes was created with `debug comm` option revealing that there was communication from each process to each other process. The amount of communication was then investigated with the help of a script provided by Markus Geimer. One could observe that an increasing number of calls to `MPI_Sendrecv` involved zero-sized messages. Also, the amount of data actually transfered seemed to vary enormously from process to process.

To account for these variations in message size, XNS was built with `MPITRACE`. Running this version records the communication and produces a log file that is supposed to suggest a more efficient mapping of the partitions onto JUBL's topology. A reduction of communication time can be expected by this when processes that exchange bigger amounts of data are located physically close to each other. This idea could not be realized during the short time of the workshop but will be followed up later on with the help of Pascal Vezolle.

4.3 Modifications based on the analysis

The first observation suggested to simply eliminate zero-transfers from the communication routines. This was done by splitting the `MPI_Sendrecv`'s into seperate `MPI_Send`'s and `MPI_Recv`'s requiring that number of bytes of the message sent/received was non-zero. This (minor) change turned out to be very efficient reducing the time spent used in the `ewdgather`/`ewdscatter` routines enormously especially when using a large number of nodes. The comparison between the old and the modified version of XNS is shown in Table 1 where communication times according to the XNS log file are listed.

number of processes	XNS `ewdgather`	XNS `ewdscatter`	modified XNS `ewdgather`	modified XNS `ewdscatter`
256	11.51	11.69	5.25	6.16
512	12.99	13.55	4.33	6.18
1024	16.70	17.40	4.16	6.06
2048	27.99	29.09	3.73	7.01
4096	57.38	58.34	3.68	6.56

Table 1: ***Communication time in seconds***

While there was communication among all processes, i.e., also among partitions that do not have a common boundary in the mesh, Figure 4b shows a significantly reduced communication pattern for 1024 processes with the modified XNS version. The communication times were obtained using Markus Geimer's script (that also records the number of visits and the amount of bytes transfered).

The effects on the scaling of the code are documented in Figure 5. We can see that without the modifi-

cations there was nearly no speed-up any more above 1024; in fact, there was even a reduction of time steps per hour for 4096 processes due to the increasing amount of communication. The modified version of the code shows some speed-up for smaller amounts of nodes, but more importantly scales fine on both a rack and a row on JUBL. We can expect a good scaling also for 8192 processes; this is to be analyzed in future test runs.

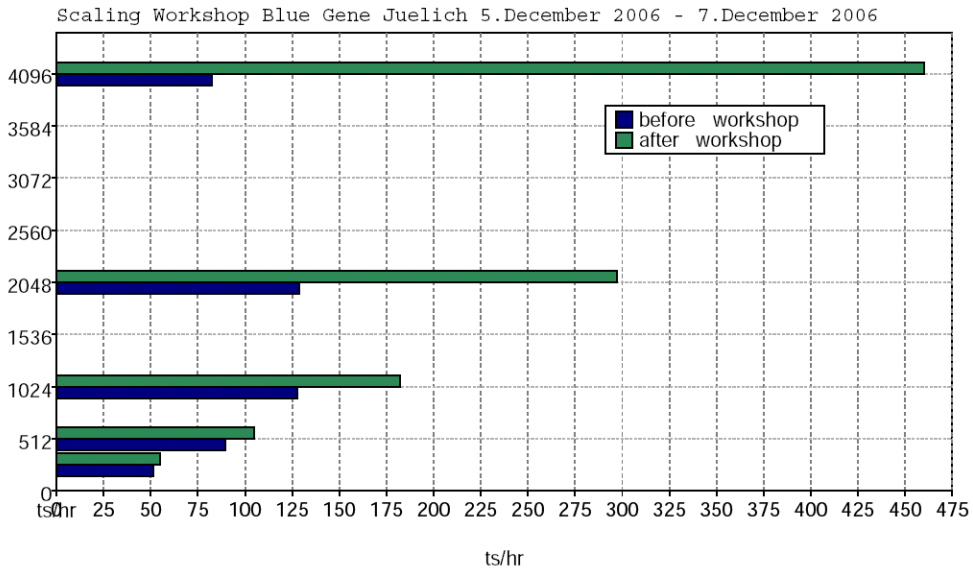

Figure 5: ***Scaling after the workshop***

Another modification in the communication routines that was tested involved non-blocking transfers. This was expected to avoid waste of time caused by late senders. Unfortunately, using `MPI_Irecv`, `MPI_Send` and `MPI_Wait` did not lead to an improved (or equal) performance for all partition sizes which should have been the case. For 4096 processors the performance was slightly worse while there was some speed-up for 2048, as seen in Figure 6. Brian Wylie will help us to follow this track and has already performed further tests (e.g., executing all the `MPI_Irecv`'s at once before any processor transfers data).

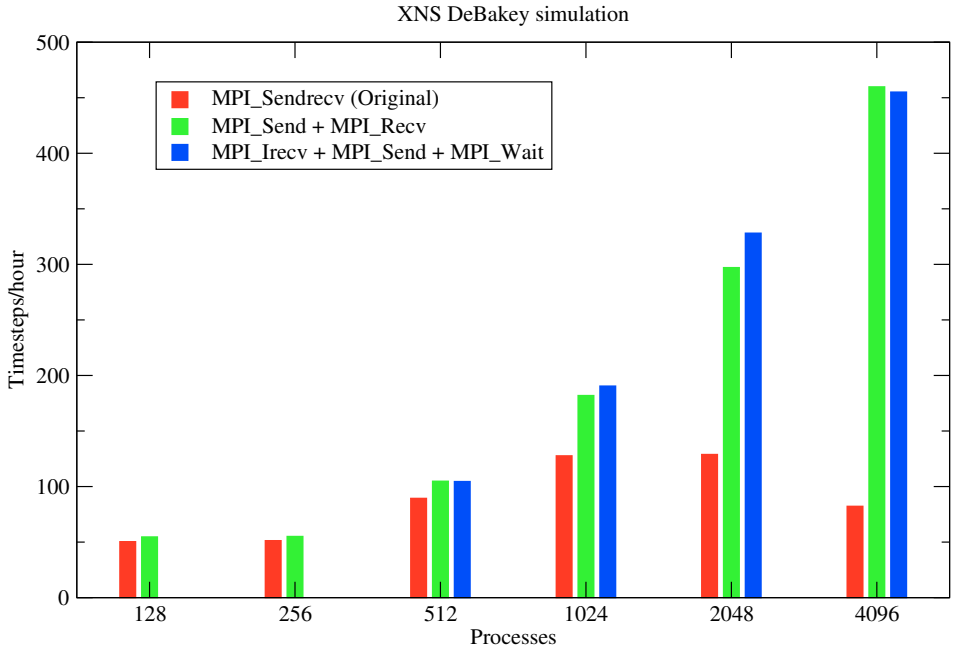

Figure 6: ***Comparison of modifications in ewdcomm.F***

4.4 Additional analysis

Since the main focus during the workshop had been on improvements in the communication routines, there was little time left to look at the other parts of the code. What could be found is a load imbalance in the ewdgmres routine, where the process with the highest rank seemed to be significantly overloaded. This is shown in Figure 7 for a test run of XNS with 1024 processes. The GMRES solver uses up to 13% of the total time spent (with 256 processes). Even if this percentage is decreasing to around 5% on 4096 processes, it could be worth investigating this imbalance and trying to reduce it.

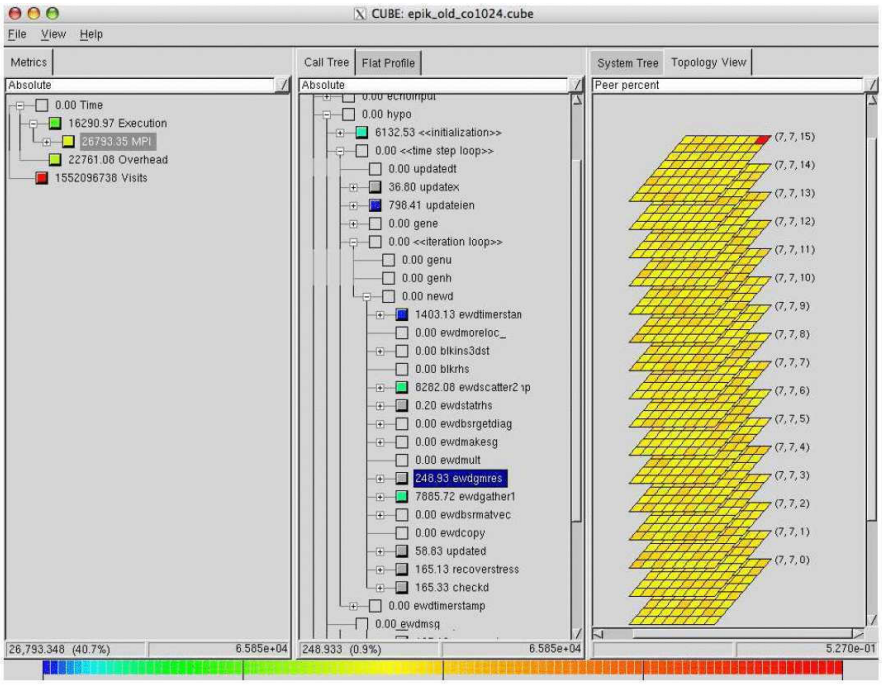

Figure 7: ***Load imbalance in ewdgmres***

5 Conclusions

As mentioned before, the workshop was very successful for our project. The performance was significatly improved by optimizing the communication routines and some other promising tracks were discovered that could lead to further improvements of the scaling. This refers particularly to reducing the load imbalance in ewdgmres and to applying an improved mapping of the mesh partitions onto JUBL that we are waiting to receive from Pascal Vezolle from IBM. Brian Wylie from FZJ has helped us a lot testing the different modifications in ewdcomm.F and we will continue to test the efficiency of using `MPI_Irecv`.

Thanks to the improved scaling, we could run the code on 4096 processors which had not been done before because of the bad performance for 2048 processors. Unfortunately, we also encountered some obstacles when running XNS for the first time on a higher number of processors. We had to spend quite some time creating the nprm file which holds information about the node renumbering on the different mesh partitions created with the METIS package. For some reason that is still to be analyzed we did not succeed in generating this file for 8192 processes which made it impossible to test the scaling of the modified version of XNS on a cube. Strangely enough, the nprm generation also crashed for 3072 processors while the file could successfully be created on 4096 processors. This is to be examined later on and can hopefully be cured.

Finally, we would like to thank the staff of Forschungszentrum Jülich, especially Brian Wylie (who had already performed some analysis prior to the workshop to give us a head start), and the IBM crew, for the nice atmosphere and the good cooperation.

6 Development of a Scalable Software Infrastructure on Blue Gene Systems

Akira Nishida
21st Century COE Program, Chuo University / Core Research for Evolution Science and Technology Program, Japan Science and Technology Agency
Hisashi Kotakemori
Core Research for Evolution Science and Technology Program, Japan Science and Technology Agency / Department of Computer Science, the University of Tokyo
Akihiro Fujii
Department of Computer Science and Communication Engineering, Kogakuin University
Akira Nukada
Core Research for Evolution Science and Technology Program, Japan Science and Technology Agency / Department of Computer Science, the University of Tokyo

6.1 Overview

Recent progress of science and technology has made numerical simulation an important approach for studies in various fields. The object of the Scalable Software Infrastructure (SSI) project, funded by the Japan Science and Technology Agency since 2002, is the development of a basic library of solutions and algorithms required for large scale scientific simulations, which have been developed separately in each field, and its integration into a scalable software infrastructure.

The components include a scalable iterative solvers library Lis, having a number of solvers, preconditioners, and matrix storage formats that are flexibly combinable, and a fast Fourier transform library FFTSS for various superscalar architectures with SIMD instructions, which outperforms some vendor-provided FFT libraries.

For this workshop, we have posed two problems:

1. As a preconditioner for iterative linear system solvers, is the algebraic multigrid scalable enough to beat the traditional preconditioners, such as the incomplete LU factorization?

2. For the scalable implementation of large scale fast Fourier transforms, is the volumetric 3D FFT the best choice? In Section 2 and 3, we describe the results of our tests on the Blue Gene systems.

6.2 Algebraic Multigrid Preconditioning

During the first run of our linear solvers library Lis on Blue Gene Watson in October 2005, we observed the linear scalability up to 8,192 nodes, and successfully run up to 16,384 nodes. The bottleneck which limited the scalability was its memory requirement to make the communication tables to manage the distributed sparse matrix data.

After the run, we implemented a new data structure to reduce the table size by keeping only the information on nonzero blocks, and a new parallel I/O methodology to dramatically reduce the data access to the global storage, which was the main bottleneck of the pre and post processes. In March 2006, we observed linear scalability up to 16,384 nodes on BGW with our new code.

Our group proposed the multigrid preconditioned conjugate gradient method (MGCG) in 1992, and the algebraic multigrid preconditioned conjugate gradient method (AMGCG), an expansion of the MGCG for unstructured meshes in 2001, which has shown to be the most effective general purpose iterative solver for symmetric linear systems. After the second run, we ported our parallel AMG preconditioner

into Lis, which dramatically boosts the performance of the conjugate gradient method on 512 nodes of the Blue Gene system at NIWS Corporation.

In this workshop, we had hard time to pinpoint the remaining bugs in AMGCG, which were finally found to be related to the notation of memory allocation, and we rewrote the Fortran 90 based AMG codes for the new version of IBM XL compiler. As of today, we can run both of the symmetric and nonsymmetric versions on Blue Gene systems, the result of which on 1,024 nodes is shown below. They are to be included in the next update of Lis. (Akira Nishida, Hisashi Kotakemori, and Akihiro Fujii)

6.3 Volumetric 3D FFT

In this workshop, we computed 3-D FFT on the Blue Gene system. Previously, Eleftheriou, et al. showed the efficiency of the volumetric FFT using 3-D decomposition, for small data sizes. We examined its performance with large data.

In our project, we have developed a high performance FFT library 'FFTSS'. It supports many kinds of processor architectures including SIMOMD instructions of PowerPC 440 FP2. In this time, 1-D FFT routine was integrated into the volumetric FFT. The volumetric FFT requires all-to-all communications for each direction of the 3-D Torus network. In our implementation, MPI communicators are created and are simply passed to MPI_Alltoall() functions. The all-to-all communications are essentially not scalable on the Torus network. For this reason, the communications occupy large percentages of the total execution time.

In computing 3-D FFT of 4096^3, we have achieved the performance of about 850 GFLOPS with eight racks. But this is only about four times the performance with one rack.

The ratio is theoretically correct. The number of nodes is doubled to each direction. The average distance of the all-to-all communication is also doubled. As a result, the throughput of the communication becomes half. The message size becomes 1/8, therefore the execution time becomes 1/4.

The performance with 64 racks is estimated to be about 3 TFLOPS. (*Akira Nukada*)

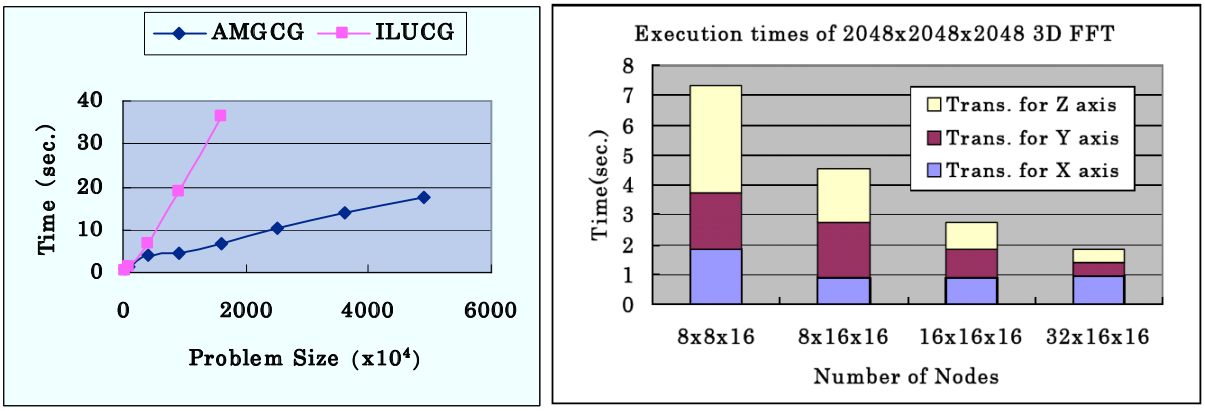

Figure 8: ***Execution time of symmetric and nonsymmetric version of AMG (left) and execution time of 3D FFT (right) on Blue Gene/L***

7 Turbulent convection for very large aspect ratios

Jörg Schumacher
Department of Mechanical Engineering, Technische Universität, D-98684 Ilmenau, Germany

Matthias Pütz
Deep Computing – Strategic Growth Business, IBM Deutschland GmbH, D-55131 Mainz, Germany

Scaling and runtime tests of a pseudospectral code on up to 16384 CPUs of the Blue Gene/L system JUBL of the John von Neumann-Institute for Computing Jülich are reported.

7.1 Physical motivation

Turbulence appears frequently in geometries for which the lateral dimensions exceed the vertical dimension by orders of magnitude. Examples appear in planetary and stellar astrophysics or in atmospheric and oceanographic science. For example, atmospheric mesoscale layers typically have lateral extensions of up to 1000 km and are characterized by aspect ratios $L : W : H = 1000 : 1000 : 1$. It is clear that the turbulent transport in the lateral dimensions will differ from that in the remaining vertical dimension. Which aspect ratios of such systems are necessary to observe features that are characteristic for two-dimensional turbulence as frequently used in geophysical fluid dynamics? When do we observe the formation of an inverse cascade for which smaller vortices feed their kinetic energy into larger structures? In addition, the turbulent motion of the air is affected by rotation, stratification, and thermal convection. Radiation and moisture play an important role as well. A bigger uncertainty for larger scale atmospheric models arises from strongly variable cloud coverage. The rapid time scales of cloud formation are still not completely understood. Recent studies suggest that the coalescence and clustering of small sub-Kolmogorov scale sized droplets is triggered by the smallest turbulent flow scales at which the inertial particles are advected. In summary, it is very challenging and practically (almost) impossible to include all effects within a single numerical simulation.

In the following, we want to study one particular process in such mesoscale layers, turbulent Rayleigh-Bénard convection. The three-dimensional pseudospectral simulation code advances the Boussinesq equations for an incompressible fluid in time by means of a second-order predictor-corrector scheme. Lateral boundary conditions are periodic; vertical boundary conditions are free-slip. The program runs in single precision and is implemented on the Blue Gene/L system.

7.2 3D-FFT packages

One of the main building blocks of the numerical method is the fast Fourier transform (FFT). The classical parallel implementation of three-dimensional FFTs uses a slabwise decomposition of the simulation domain. For a simulation with N^3 grid points, the method allows a parallelization on up to N processors. Due to the rather small memory size per core, the Blue Gene/L requires a *volumetric* FFT which decomposes the three-dimensional volume into cuboid-like rods and hence allows a parallelization degree of N^2. The prime requirement for being able to run a simulation with a large grid is that the domain fractions of the grid (including buffers and temporary storage) fit into the 512 MB of memory on a single Blue Gene/L node. At the same time, of course, the FFT algorithm should also be scalable, i.e. increasing the number of CPUs to solve the problem should also substantially decrease the time-to-answer.

In order to check this, we compared three FFT packages on a small test case: (i) the old slabwise method, (ii) the BGL3DFFT package by M. Eleftheriou *et al.* (IBM J. Res. & Dev. **49**, (2005)), (iii) the P3DFFT package by D. Pekurovsky (http://www.sdsc.edu/user_services/applications/fft3d.html). Package (ii) is written in C++ and contains complex-to-complex FFTs only (Fortran- and C-bindings are available from

IBM). Packages (i) and (iii) are available in Fortran. The results on strong scaling are summarized in Figure 9. For the latter two packages we have varied the mapping of the MPI tasks onto the torus grid network of the Blue Gene/L machine. By default, Blue Gene/L maps MPI tasks on all primary cores of a partition first (BGLMPI_MAPPING = XYZT. Beside the fact that the P3DFFT interface and implementation supports our needs in an optimal way (real-to-complex/complex-to-real), it also turned out to be the best solution in terms of performance. For example, the symmetry $\mathbf{u}(\mathbf{k}) = \mathbf{u}^*(-\mathbf{k})$ for real-to-complex transforms is explicitely implemented already and they allow for the storage of physical space and Fourier space fields into the same array (in-place transformation).

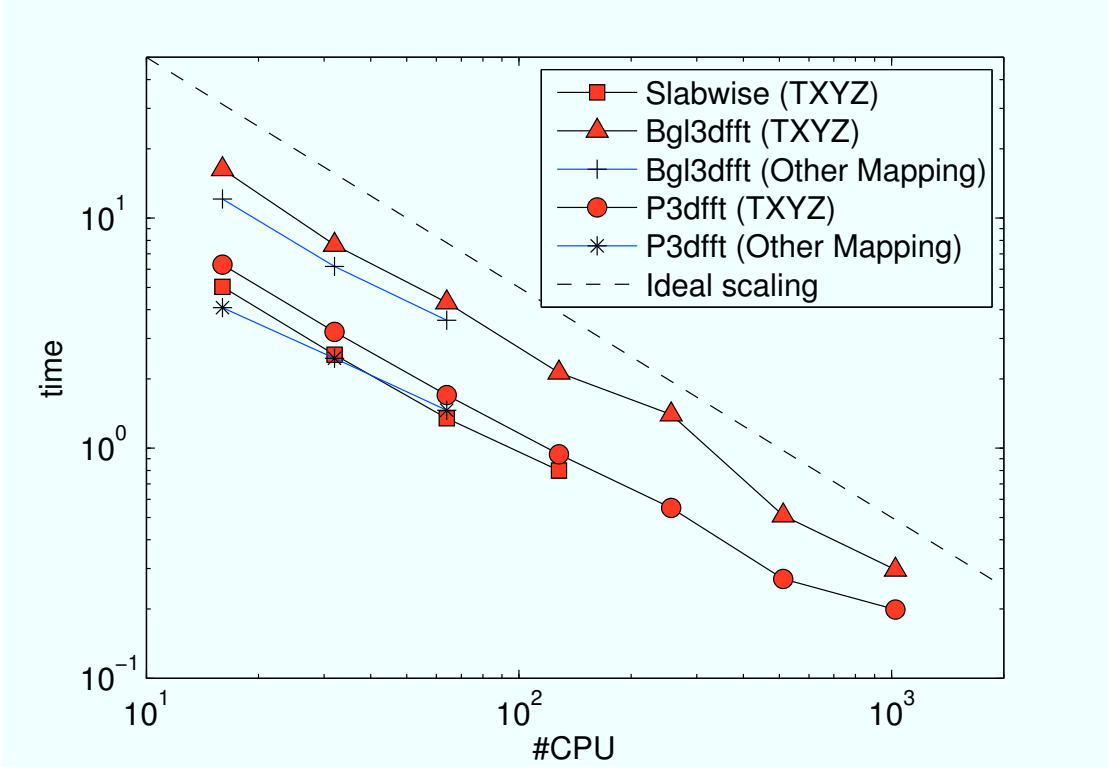

Figure 9: Strong scaling tests for up to 1024 CPUs. The test function is $f(x, y, z) = \sin(x)\cos(2y)\sin(z)$ which is resolved on an equidistant 128^3 cubic grid of sidelength 2π.

7.3 Scaling tests for the convection code

The inclusion of the P3DFFT package into the convection code gave the following scaling on one rack (2048 CPUs) for a $512 \times 512 \times 64$ system with an aspect ratio $\Gamma = L/H = 16$. Figure 10 shows the results. All tests have been done in the virtual node mode. The following improvements of the code performance should be mentioned:

- For numbers of CPUs ≥ 256 the mpirun option for the processor topology BGL_MAPPING = XYZT brought the shortest computation times.
- Splitting the loops for the calculation of the r.h.s. of the Boussinesq equations brought an acceleration of the code. This option avoids an if-then command for the $\mathbf{k} = 0$ case which would otherwise inhibit code vectorization.
- The use of non-blocking point-to-point communication instead of MPI_Reduce and MPI_Bcast caused a gain of 10% in total time.

The analysis of the code with the mpitracelib indicated that after all these improvements had been made almost all of the communication overhead remained in the MPI_Alltoallv communication task which is

required in the FFT algorithm for the transposition of the cuboid-like rods and is hence the only factor which limits strong scaling. Future improvements of the implemenation of MPI_Alltoallv will therefore have an immediate impact on the performance of our code.

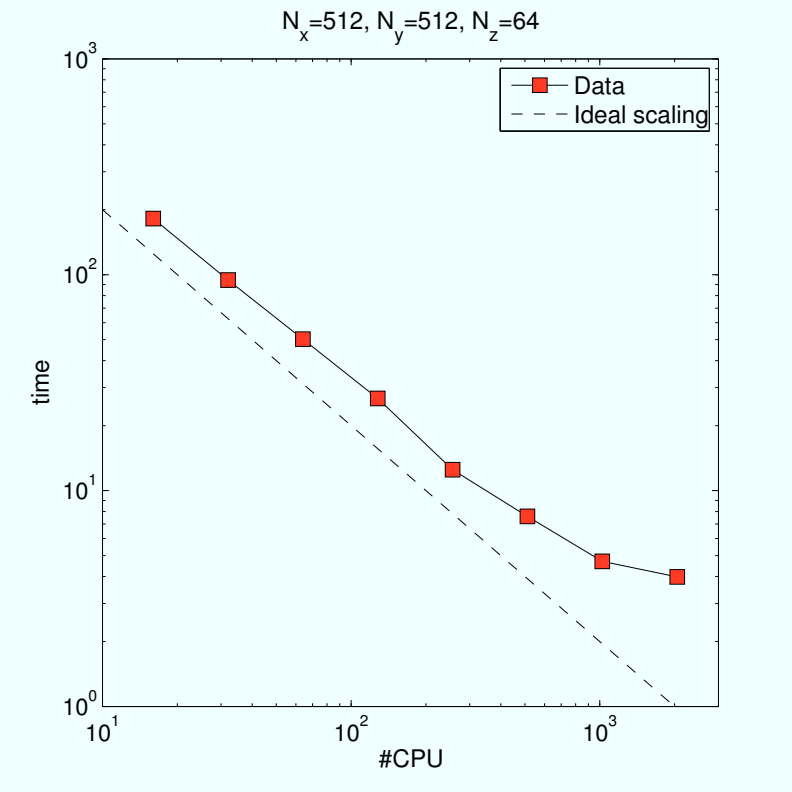

Figure 10: Strong scaling tests for the code on up to 2048 CPUs.

Table 7.3 summarizes our findings for grid resolutions that we intend to use for the production runs, i.e. $N_x \times N_y \times N_z = 4096 \times 4096 \times 256$ at $\Gamma = 32$ given the present ressources at the NIC Jülich. For such large grids it became necessary to implement some internal summation loops in double precision. All of these loops are used to calculate the ride-hand-side (RHS) of the equations of motion. The number of grid points per MPI task was of the order of 10^6 which is simply to large to sum over with 32-bit floating point precision.

We have studied the two modes in which one dual core CPU (which is denoted as one node) can be operated on Blue Gene/L, the virtual node mode (VN) and the co-processor mode (CO). The parameters $iproc$ and $jproc$ in the table represent the two dimensions of the processor grid that has to be generated for the volumetric FFTs. Note, that the aspect ratio of $iproc : jproc$ can be tuned as long as $iproc \times jproc < N_{cpu}$. In general, an aspect ratio of 1 would be best, but other choices can be better, if they improve the mapping of the MPI tasks onto the partition grid. For the large problem we observe that the differences between the VN and CO modes are small. This results from the fact that large local grid fractions do no longer fit into the L3 cache of a Blue Gene/L node and have to be streamed from memory. If the second core is used for computation (VN mode) the two cores are actually competing for the memory bandwidth of the node and the gain in computing time is fairly small. Additionally, the communication time in VN mode is bigger than in CO mode, such that these two effects almost cancel each other. The table indicates that for ≥ 8192 CPUs runtime starts to saturate. There is no chance to improve the cache locality of the code to avoid this, because the FFTs will always require the full grid. There may however be some further optimization potential by using explicit task mappings. The main problem here is to find an embedding of a 2d grid decomposition into a 3d torus communication grid, such that no communication hot spots occur on the point-to-point network of the Blue Gene/L. In general this is a hard problem.

CPUs	processor mode	$iproc \times jproc$	time
2048	vn	32×64	183.3 s
1024	co	32×32	205.7 s
4096	vn	64×64	118.6 s
4096	vn	128×32	109.6 s
8192	vn	128×64	77.8 s
4096	co	64×64	90.66 s
16384	vn	128×128	62.6 s
8192	co	128×64	60.5 s
8192	co	512×16	74.0 s

Table 2: Runtime tests for the aspect ratio $\Gamma = 32$. The CPU time was measured with MPI_Wtime() task for a loop of 5 time steps.

7.4 Summary and outlook

The use of the P3DFFT package and a number of further improvements in the communication overhead improved significantly the performance of our code on the Blue Gene/L system. These efforts allow us to run production jobs on at least two racks. Since the CPU time of our code is memory bandwidth limited for problem sizes of interest, VN mode does not give a significant speedup over CO mode.

8 Numerical simulations of QCD

Hinnerk Stüben
Konrad-Zuse-Zentrum für Informationstechnik Berlin, Berlin, Germany

Thomas Streuer
Department of Physics and Astronomy, University of Kentucky,Lexington, KY, USA
– QCDSF collaboration –

The scientific aim of current projects of the QCDSF collaboration is to better constrain the extrapolation of the lattice results to the chiral and continuum limits by performing simulations at more realistic quark masses and at smaller lattice spacings. This has become possible now due to substantial improvements of the Hybrid Monte-Carlo (HMC) algorithm and a significant increase in computing power.

At the BlueGene/L Scaling Workshop we studied the scaling of our production code BQCD (Berlin quantum chromodynamics programme) up to the full machine. BQCD is a Hybrid Monte-Carlo programme that simulates quantum chromodynamics (QCD) with Wilson gauge action and non-perturbatively $O(a)$ improved Wilson fermions. The overall programme was written in Fortran 90 and parallelised with MPI. For the BlueGene the most compute intensive part was implemented in assembler.

In computational QCD the theory is defined on a four-dimensional regular lattice with (anti-) periodic boundary conditions. The kernel of BQCD is a standard conjugate gradient solver with even/odd preconditioning. Typically, 80 % of the execution time is spent in that solver. The dominant operation in the solver is the matrix times vector multiplication. In the context of QCD the matrix is called *fermion matrix*. The fermion matrix is sparse. It has eight entries per row. The entries in row i are the nearest neighbours of entry i of the vector.

The entries of the fermion matrix are 3×3 complex matrices and the entries of the vector are 3×4 complex matrices. The experience on current machines is that the performance of the Fortran code for the matrix times vector multiplication is about 10 % of peak and that the performance approximately doubles for the assembler version. On the Hitachi SR8000-F1 which was one of the main production machine for the QCDSF collaboration from 2000–2006 the Fortran code ran at about 40 % of peak.

machine dimensions			total lattice volume	
			$32^3 \times 64$	$48^3 \times 96$
# racks	# cores	network torus	local volume	local volume
1	2048	$8 \times 8 \times 16 \times 2$	$16 \times 4 \times 4 \times 4$	$24 \times 6 \times 6 \times 6$
2	4096	$8 \times 16 \times 16 \times 2$	$16 \times 4 \times 4 \times 2$	$24 \times 6 \times 6 \times 3$
4	8192	$16 \times 16 \times 16 \times 2$	$16 \times 4 \times 2 \times 2$	$24 \times 6 \times 3 \times 3$
8	16384	$32 \times 16 \times 16 \times 2$	$16 \times 2 \times 2 \times 2$	$24 \times 3 \times 3 \times 3$

Table 3: Lattice volumes per core. The network torus dimension "2" corresponds to the two cores of a node.

The programme is parallelised by a domain decomposition. In the parallelised *cg*-kernel ghost cells of the input vector have to be exchanged before the vector can be multiplied with the fermion matrix. In order to scale QCD programmes to high numbers of processes an excellent communication network is required because the lattice volume per core becomes small (see Table 3). On the other hand small lattice volumes per core improve the utilisation of data caches and hence improve performance.

8.1 Details on the assembler code

For the floating point operations of the femion matrix multiplication we use the SIMD *(double hummer)* instructions of the floating point units, which can perform four floating point operations per clock cycle. The communication is done in parallel to the computation, so that in principle computation and communication can overlap. For the communication between nearest neighbours in three of the four directions we use the torus network. The network is accessed from the nodes via memory-mapped fifos. Each node has six injection fifos, which can receive outgoing packets in any direction. Each of the two cores in one node uses three of these fifos, so that packets going into forward and backward directions have to share one fifo. The packet size must be a multiple of 32 bytes, including an eight-byte header, so that we use packets of 128 bytes for one projected spinor. Incoming packets arrive in one of twelve reception fifos: there are two fifos for each of the six nearest neighbours of a node. These two fifos are assigned to the two cores in a node. Because of the simple communication pattern (of nearest neighbour data exchange only), no dynamical routing is necessary. Furthermore, each node receives the data packets in the order in which they are needed, so that no reordering of packets is necessary. Each node sends a packet to its neighbour one iteration before the neighbouring node needs the data, so that the network latency can be hidden.

In the fourth direction, which corresponds to the x-direction of the lattice, the lattice is split between the two CPUs in one node, so that the communication can be done via the shared memory. However, since the L1 caches are not coherent, the straightforward approach of simply reading data from the other CPUs memory does not work. At present, we use a memory region in the L3 cache (the scratchpad area), which is marked L1-caching inhibited for data exchange between the two cores.

implementation: Fortran/MPI		lattice: $48^3 \times 96$		
#racks	MFlop/s per core	overall TFlop/s	speed-up	efficiency
1	280	0.57	1.00	1.00
2	292	1.20	2.09	1.04
4	309	2.53	4.41	1.10
8	325	5.32	9.29	1.16
implementation: Fortran/MPI		lattice: $32^4 \times 64$		
#racks	MFlop/s per core	overall TFlop/s	speed-up	efficiency
1	337	0.69	1.00	1.00
2	321	1.32	1.91	0.95
4	280	2.30	3.33	0.83
8	222	3.65	5.28	0.66
implementation: assembler		lattice: $32^3 \times 64$		
#racks	MFlop/s per core	overall TFlop/s	speed-up	efficiency
1	535	1.10	1.00	1.00
2	537	2.20	2.01	1.00
8	491	8.05	7.34	0.92

Table 4: Performance of the *cg*-kernel for two implementations and two lattices.

8.2 Results

In scaling tests the performance of the *cg*-kernel was measured. For performance measurements the code was instrumented with timer calls and for the kernel all floating point operations were counted manually.

In order to get good performance it is important that the lattice fits the physical torus of the machine. In order to achieve this, MPI process ranks have to be assigned properly. On the BlueGene/L this can be accomplished by setting the environment variable `BGLMPI_MAPPING` appropriately. The settings of that variable were `TXYZ` on 1, 2, and 4 racks and `TYZX` on 8 racks.

Performance results are given in Table 4. In Figure 11 the results are shown on double logarithmic plots (the dotted lines indicate linear scaling).

One can see from the table and the plots that the Fortran/MPI version exposes super-linear scaling on the $48^3 \times 96$ lattice. Even the $32^3 \times 64$ lattice scales quite well given the fact that the lattice volumes per core are tiny.

The scaling of the assembler version is excellent. For the same tiny local lattices the scaling is considerably better than for the Fortran/MPI version. This means that in the assembler version computation and communication really overlap.

Conclusion. We found that our code scales up to the whole Blue Gene/L at NIC/ZAM. The highest performance measured was 8.05 TFlop/s on the whole machine. The high performance could be obtained by using *double hummer* instructions and techniques to overlap communication and computation.

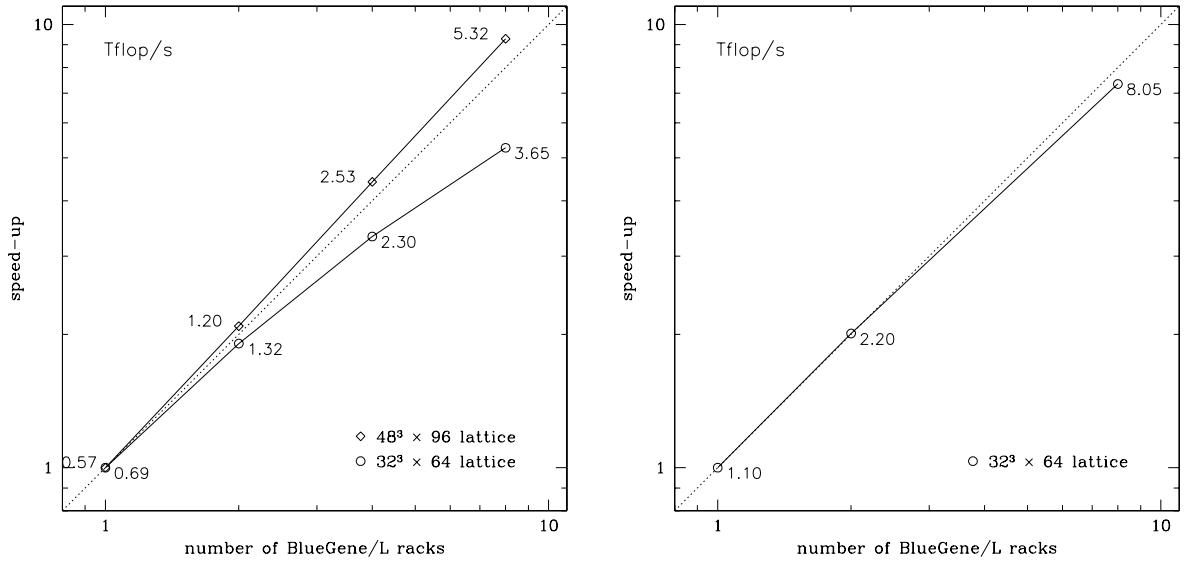

Figure 11: **Scaling of the *cg*-kernel of BQCD for the Fortran 90/MPI version (left) and for the assembler version (right).**