Konrad-Zuse-Zentrum
für Informationstechnik Berlin

Takustraße 7
D-14195 Berlin-Dahlem
Germany

Mohammed Allalen[1], Matthias Brehm[1],
Hinnerk Stüben

Performance of Quantum Chromodynamics (QCD) Simulations on the SGI Altix

[1]Leibniz Supercomputing Centre (LRZ), Garching, Germany

Contribution to the SGI User Group 2008 Conference, 9–12 June 2008, Los Angeles, California, USA

ZIB-Report 08–54 (October 2008)

Performance of quantum chromodynamics (QCD) simulations on the SGI Altix 4700

Mohammed Allalen†‖, Matthias Brehm†, and Hinnerk Stüben‡

† Leibniz Supercomputing Centre (LRZ), Garching, Germany

‡ Konrad-Zuse-Zentrum für Informationstechnik Berlin (ZIB), Berlin, Germany

Abstract. We study performance and scaling of the Berlin Quantum Chromodynamics Program (BQCD) on the SGI Altix 4700 at Leibniz Supercomputing Centre (LRZ). We employ different communication methods (MPI, MPI with two OpenMP threads per process, as well as the *shmem* library) and run the MPI version on the two types of nodes of that machine. For comparison with other machines we made performance measurements on an IBM p690 cluster and a Cray XT4.

1. Introduction

The SGI Altix 4700 at Leibniz Supercomputing Centre (LRZ) is one of the most powerful computers in Germany. It has 9728 cores, 39 TByte of main memory, and delivers a peak performance of 62.3 Tflop/s. A prominent feature of an Altix is its ccNUMA architecture. The machine at LRZ has two other special features. It has two types of nodes and a two-dimensional torus network connecting the nodes. There are 13 so-called high-bandwidth nodes in which two cores are connected to one memory channel and there are six high-density nodes in which four cores are connected to one memory channel, i.e. in the high-density nodes the memory bandwidth per core is halved. Each node has 512 cores out of which two (or four) are reserved for the operating system on the high-bandwidth (or high-density) nodes.

The torus network is sketched in Figure 1. The network of the machine has a hierarchical structure. Within a node the bisection bandwidth per blade is 2×0.8 GByte/s (one blade comprises eight cores in the high-bandwidth nodes and 16 cores in the high-density nodes). Between two 'vertical' nodes it is 2×0.4 GByte/s (bisection indicated by cut *a* in Figure 1), between 'horizontal' nodes it is 2×0.2 GByte/s (bisection indicated by cut *b* in Figure 1), and for the whole system it is further reduced to 2×0.1 GByte/s.

In this paper we try to get some deeper understanding of the machine. One aspect is to use different parallelisation strategies on the ccNUMA architecture. The second aspect is the influence of the different types of nodes. In addition we compare the Altix 4700 with two other supercomputers, an IBM p690 cluster and a Cray

‖ Send offprint request to: M. Allalen, allalen@lrz.de

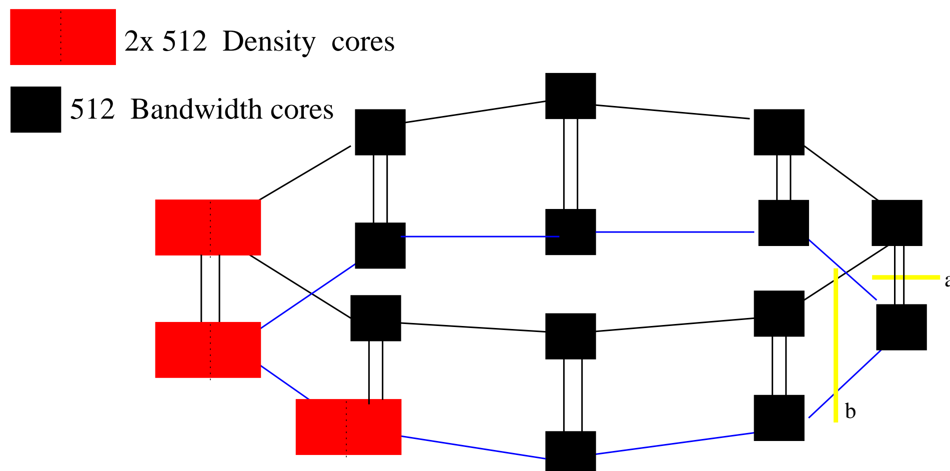

Figure 1. Two-dimensional torus network of the SGI Altix 4700 nodes at LRZ. Bisections of two nodes with different network bandwidths are indicated by cut *a* and cut *b* (see text).

XT4. In all cases we are interested in achieving high scalability. In our study we use the Berlin Quantum Chromodynamics Program (BQCD). BQCD has various communication methods implemented: MPI, OpenMP, a combination of both, as well as *shmem* (single sided communication) in the hopping matrix multiplication (see Sect. 3).

BQCD is used in benchmarks for supercomputer procurements at our centres. The benchmark version implicitly measures two important aspects of supercomputer applications: effective network- and memory and bandwidth. In addition QCD is an application that scales very well and reliable performance measurements on large number of cores can be obtained within minutes. Production versions of BQCD and other QCD programs are highly tuned including assembler parts in the kernel [1, 2]. Here we employ the high level Fortran90 version as an example of a typical supercomputer application.

In the following we start by giving a short overview of numerical simulations of QCD and a short introduction to the computational aspects of QCD simulations. Then we present and discuss our results.

2. Overview of numerical simulations of QCD

QCD is the theory of strongly interacting elementary particles. The theory describes particle properties like masses and decay constants from first principles. The starting point of QCD is an infinite-dimensional integral. To deal with the theory on the computer space-time continuum is replaced by a four-dimensional regular finite lattice with (anti-) periodic boundary conditions. After this discretisation, the integral is finite-dimensional but still rather high-dimensional. The high-dimensional integral is solved by Monte-Carlo methods. BQCD is a program that simulates QCD with the Hybrid Monte-Carlo algorithm [3].

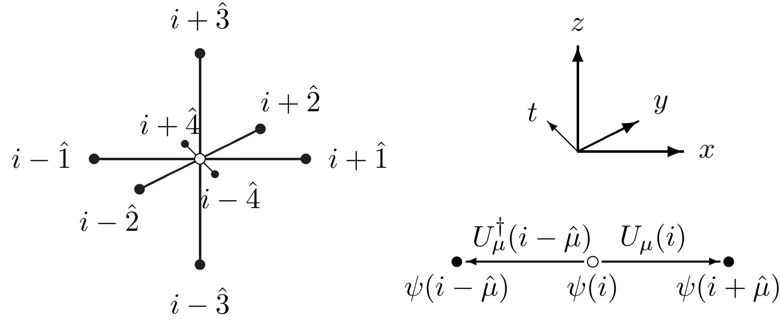

Figure 2. Nearest neighbour stencil underlying the hopping matrix. The central point is i, where i is a short cut for a point given by four coordinates (x, y, z, t). On the right-hand side the corresponding Cartesian coordinate system and the variables involved are indicated for one dimension. U is called the *gauge field* which is defined on the links of the lattice. The field ψ is defined on the lattice sites. The index μ stands for a direction and $\hat{\mu}$ is a unit vector in μ-direction.

The basic building blocks of QCD are called quarks (matter particles) and gluons (particles mediating the interaction of quarks). The quark fields cannot be represented directly on a computer. In the computations they appear as large sparse matrices which describe systems of linear equations. QCD programs spend most of their execution time in solving these systems of linear equations. In the solver and an overall QCD program the multiplication of the so-called hopping matrix with a vector is the dominant operation. The hopping matrix multiplication is communication intensive.

3. Computational aspects

To go a little bit more into detail let us describe the structure of the hopping matrix and the systems of linear equations.

QCD is defined on a four-dimensional Cartesian lattice. The lattice has three spatial and one time direction. Its size is denoted by $L_s^3 \times L_t$. On the links of the lattice the field U is defined which represents the gluons. U is a function of the four directions $\mu = 1, 2, 3, 4$ and the lattice sites denoted by i (see right-hand side of Figure 2). $U_\mu(i)$ is a 3×3 complex matrix. The U field is part of the hopping matrix. It is constant in the solver. On the sites of the lattice the field ψ is defined which represents the quarks. $\psi(i)$ is a 4×3 complex matrix. These kinds of fields are the vectors in our systems of linear equations. In our Fortran program U and ψ have the following data structure:

```
complex(8) u(3, 3, Ls, Ls, Ls, Lt)
complex(8) psi(4, 3, Ls, Ls, Ls, Lt)
```

In practice the four dimensions `(Ls, Ls, Ls, Lt)` are collapsed to a single one and there is one array `u` for each of the four dimensions. In a pseudo code notation the matrix multiplication reads:

```
psi_out := hopping_matrix[u] * psi_in
```

The entries of the hopping matrix are given by a four-dimensional nearest neighbour stencil as indicated in Figure 2, i.e. the hopping matrix has nine entries per row. The entries are the $U_\mu(i)$ matrices.

At the single CPU level QCD programs benefit from the fact that the basic operations involve the small complex matrices $U_\mu(i)$ and $\psi(i)$. One can perform at the order of ten floating point operations per memory access. As a rule of thumb, the resulting performance is about 20–25% of peak when programming in Fortran or C. The single CPU performance can be considerably improved by employing low level programming techniques like assembler or multimedia streaming functions.

QCD programs are parallelised by domain decomposition. The nearest neighbour structure of the hopping matrix implies that the boundary values (surfaces) of `psi_in` have to be exchanged between neighbouring processes in every iteration of the solver. In production runs where one aims at sustained performance in the Tflop/s range the domains become so small that their surface to volume ratio is at the order of one or even larger. In Table 1 we give that ratio for the lattices and numbers of processes we consider here. The ratio depends on the actual decomposition. For example, if the lattice is decomposed into sub-lattices of size $l_s^3 \times l_t$ the surface to volume ratio is $(2 \times l_s^3 + 6 \times l_s^2 \times l_t)/(l_s^3 \times l_t)$. In general the four dimensions of the local lattice can have different extension l_x, l_y, l_z, and l_t.

Table 1. Surface to volume ratios

number of processes	64	128	256	512	1024	2048	3072	4096
$24^3 \times 48$ lattice	0.833	1.000	1.167	1.333				
$48^3 \times 96$ lattice	0.417	0.500	0.583	0.667	0.833	1.000	1.083	1.167

Decomposing the lattice for a large number of processes has two effects. First, at some stage a domain might completely fit into the data cache. Second, the data from the relatively large surface of the small domains has to be communicated to eight nearest neighbour processes. The communication becomes dominant for large numbers of processes. It requires an excellent network. For lattice sizes that are used in actual simulations the network is the challenge. In our examples we mainly use the $48^3 \times 96$ lattice which is relatively large for today's supercomputers. For the comparison with other machines we use the $24^3 \times 48$ lattice which is 16 times smaller but which has similar surface to volume ratios as the $48^3 \times 96$ lattice and thus similar communication requirements as that lattice but on fewer processes (see Table 1).

4. Results

All results we present are for the entire conjugate gradient (*cg*) solver of BQCD. This is essentially the overall performance of the program in practical simulations. The performance measurement is based on manually instrumented code and manually

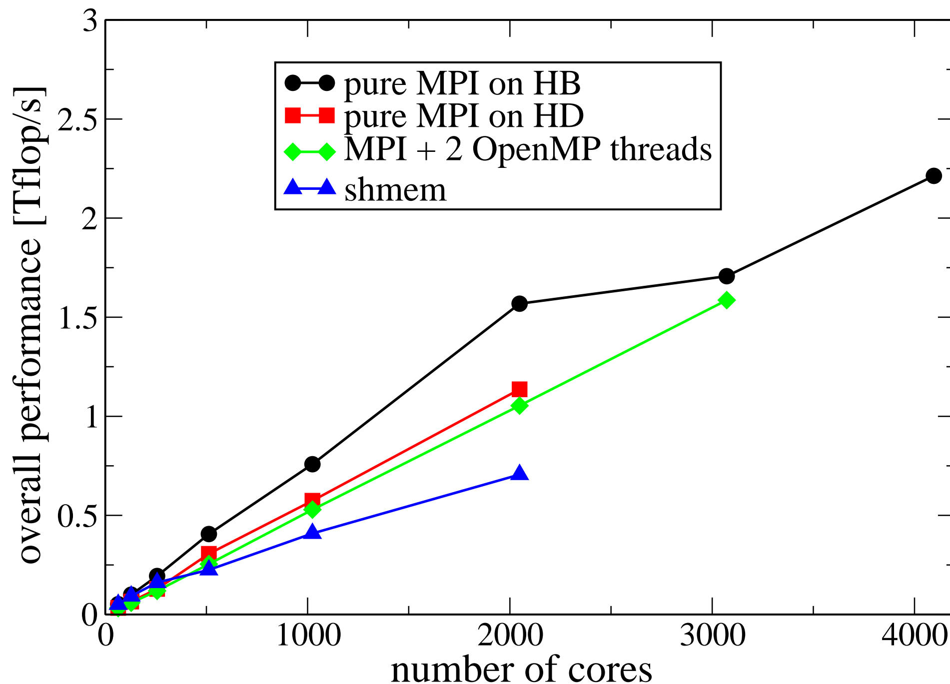

Figure 3. Strong scaling of BQCD for the $48^3 \times 96$ lattice on SGI Altix 4700 using different communication setups (see text).

counted operations in the source code. In all tables we give four results. First, we give the overall performance including communication overhead in Gflop/s. The overall performance is plotted in Figures 3, 4 and 5. Second, we give the compute performance per core, i.e. the performance that was measured in program regions outside MPI (or *shmem*) functions. This quantity indicates whether one is in a memory bound region, where the quantity would be constant, or one can profit from data caching, where the quantity would grow. Third, we give the effective MPI (or *shmem*) bandwidth per process. In an ideal network this quantity should be constant for any number of processes. Fourth, we give the MPI overhead, i.e. the fraction of time spent in communication routines. Typically the overhead grows with increasing numbers of processes because the surface to volume ratio increases.

We study strong scaling of simulations on the $48^3 \times 96$ lattice. On the Altix we look into four setups, namely running with

1. MPI on high-bandwidth nodes,
2. MPI on high-density nodes,
3. MPI plus two OpenMP threads per process on high-bandwidth nodes,
4. *shmem* on high-bandwidth nodes,

followed by

5. a comparison with the other platforms.

4.1. MPI on high-bandwidth nodes

Results for this setup are given in Table 2. The overall performance scales very well up to 2048 cores and becomes worse for higher numbers of cores.

Table 2. Performance on high-bandwidth nodes

number of cores	64	128	256	512	1024	2048	3072	4096
overall performance [Gflop/s]	52	100	194	406	758	1568	1707	2213
compute perf. per core [Mflop/s]	954	947	977	1106	1171	1607	1450	1535
MPI perf. per proc. [MByte/s]	464	450	379	350	306	262	178	171
MPI overhead [%]	14	17	22	28	37	52	62	65

The main reason for the good scaling is the utilisation of the data cache. The compute performance increases from about 950 up to 1600 Mflop/s per core. At the same time the MPI overhead stems not only from the increasing surface to volume ratio but also from decreasing effective MPI bandwidth. This effect is quite pronounced. Up to 2048 cores this can be compensated by data caching. In that case the communication loss is already 52 %.

4.2. MPI on high-density nodes

Results for this setup are given in Table 3. Again the overall performance scales very well up to 2048 cores which in this case is the largest job possible in the system configuration.

Table 3. Performance on high-density nodes

number of cores	64	128	256	512	1024	2048
overall performance [Gflop/s]	33	65	129	306	574	1136
compute performance per core [Mflop/s]	611	611	643	839	925	1130
MPI performance per process [MByte/s]	305	313	262	262	218	192
MPI overhead [%]	14	16	22	29	39	51

Up to 256 cores the compute performance per core is roughly constant. This is the memory bound region. In these cases the compute performance is about 65 % of the performance obtained on high-bandwidth nodes which shows a clear dependency on the memory bandwidth. For higher numbers of cores the data caches come into play and the compute performance grows up to 79 % of the value from high-bandwidth nodes. The overall performance behaves similarly. In the high-density partition the MPI bandwidth varies less than in the high-bandwidth partition. On 2048 cores the MPI bandwidth is 63 % of the bandwidth measured on 64 cores. For the high-bandwidth partition the corresponding value is 56 %.

4.3. MPI plus two OpenMP threads on high-bandwidth nodes

On a shared memory system or a system with shared memory properties it is tempting to reduce the communication overhead by working with more than one thread per MPI process. By doing this the domains per MPI process become larger and as a consequence less data has to be communicated for a given problem size. Therefore we tried to use two OpenMP threads per MPI process. The idea is that the two threads work on the two cores of the same (dual core) Itanium processor of the Altix. Results for this setup are given in Table 4. In this setup we find good scaling up to even 3072 cores.

Table 4. Performance on high-bandwidth nodes for MPI plus 2 OpenMP threads

number of cores	64	128	256	512	1024	2048	3072
overall performance [Gflop/s]	32	59	119	254	529	1054	1586
compute performance per core [Mflop/s]	543	540	559	685	704	893	1121
MPI performance per process [MByte/s]	905	520	516	387	544	373	315
MPI overhead [%]	7	14	17	28	27	47	54

However, the absolute performance is slightly lower than the performance in the high-density case. The MPI bandwidth per core is higher than in the high-bandwidth case but the compute performance per core is significantly lower. This effect can already be observed in small test cases where we put an $8^3 \times 16$ lattice on two cores using two MPI processes or two OpenMP threads. Using threads the performance was only 78 % of the MPI case. To get that reasonable OpenMP performance it is important to pin threads to processor cores and control page allocation. On the Altix this can be accomplished by employing the *omplace* command. In our tests OpenMP performance is roughly halved when *omplace* is not employed.

4.4. shmem on high-bandwidth nodes

The last setup we have tried on the Altix is replacing the `MPI_Sendrecv` in the hopping matrix multiplication by single sided communication functions from the *shmem* library (we used `shmem_put`). Results for this setup are given in Table 5.

Table 5. Performance on high-bandwidth nodes using *shmem*

number of cores	64	128	256	512	1024	2048
overall performance [Gflop/s]	51	93	161	225	409	706
compute performance per core [Mflop/s]	988	1062	1094	1122	1141	1247
shmem performance per process [MByte/s]	301	187	131	67	76	63
shmem overhead [%]	20	31	42	61	65	72

Up to 256 cores the overall performance is comparable to the MPI setup. For higher numbers of cores the scaling becomes worse. While the first three setups scale practically

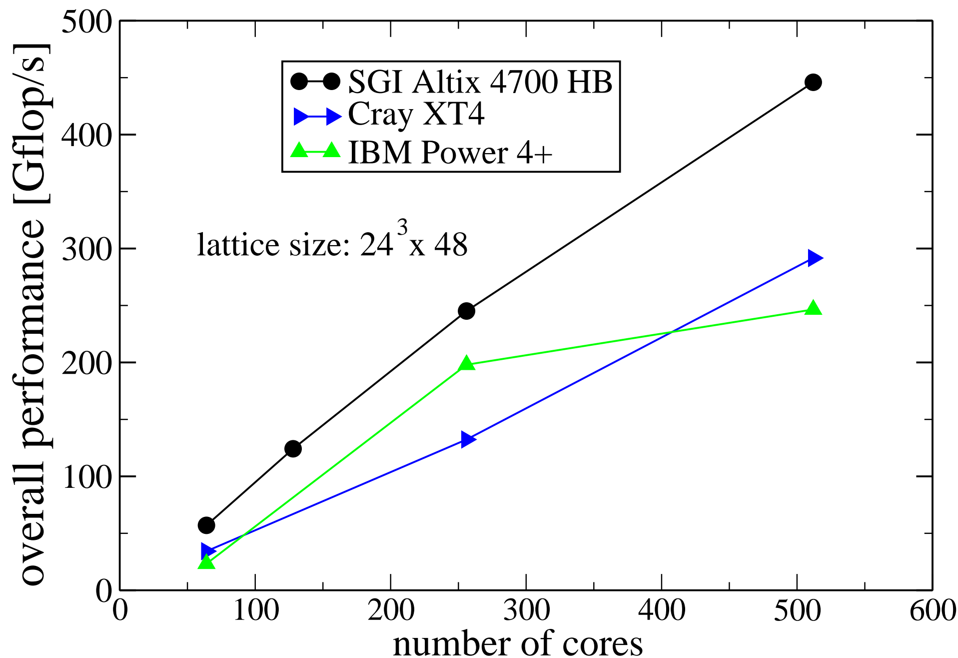

Figure 4. Performance comparison of three platforms (24×48 lattice).

linearly up to 2048 cores, the parallel efficiency on 2048 cores with *shmem* is only 0.43 (related to 64 cores). The striking observation for this setup is that the effective MPI bandwidth is much lower than in the other cases. We think that this effect is due to the latencies of the *shmem* communication that add up in many function calls. In contrast to MPI the *shmem* library does not contain a function for transferring block-strided data. There are only functions for contiguous blocks or strided data with block size one. In Fortran90 notation the array sections corresponding to surfaces are:

```
psi(:, :, :, :, :, 1)        psi(:, :, :, :, :, l_t)
psi(:, :, :, :, 1, :)        psi(:, :, :, :, l_z, :)
psi(:, :, :, 1, :, :)        psi(:, :, :, l_y, :, :)
psi(:, :, 1, :, :, :)        psi(:, :, l_x, :, :, :)
```

Only the array sections defined in the first line consist of one contiguous block each while all other array sections are block-strided. Hence *shmem* has to be called much more often than MPI and latencies add up.

4.5. Comparison with other platforms

For comparison we repeated some measurements using pure MPI communication on an IBM p690 cluster and a Cray XT4. On those platforms resource usage was limited to 512 cores. To challenge MPI communication we measured on a $24^3 \times 48$ lattice in addition to the $48^3 \times 96$ lattice (cf. Table 1). Simulating on the large lattice requires approximately 160 GByte of main memory. On the XT4 this was not available on 64

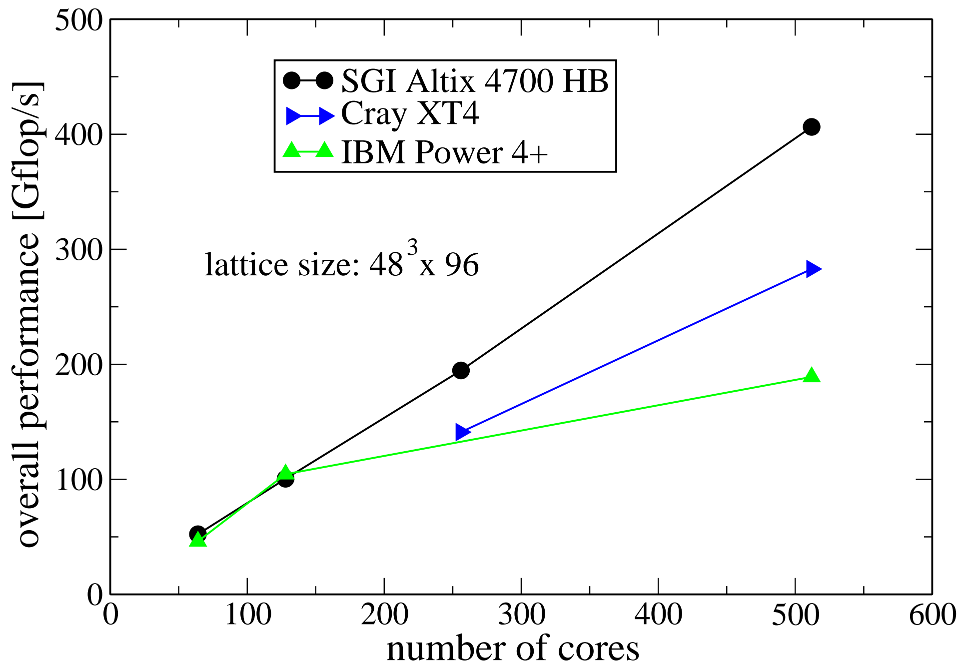

Figure 5. Performance comparison of three platforms (48×96 lattice).

and 128 cores. Results are compiled in Table 6. The overall performance is plotted in Figures 4 and 5.

On both lattices the Altix delivers the best overall performance except for the large lattice on 128 cores where the p690 performs slightly better.

On the Altix the performance figures from the small lattice behave similar to the ones from the large lattice (see Sect. 4.1). The role of the data cache is even more pronounced on the small lattice where the compute performance reaches up to 2.6 Gflop/s per core. On both lattices the MPI bandwidth decreases in a similar way when the number of cores is increased.

From Figures 4 and 5 one can see directly that scaling to 512 cores is not good on the p690. On the p690 we find a sweet spot for both lattices where the MPI performance is much better than in the other cases. On the small lattice the compute performance is significantly increased at the same time. The effect is super-linear scaling from 64 to 256 cores. In order to try to explain the drop in performance on 512 cores we have to come back to network latencies. We explained the poor *shmem* performance by latencies that add up. On the p690 we also see the effect that the MPI performance decreases when using the maximal number of cores. In addition we see that the compute performance decreases too. This effect can be explained by latencies as well because the global sum is not excluded in the measurement of the compute performance. Large network latencies lead to relatively slow global reduction functions what introduces additional communication overhead. The effect can also be noticed for the small lattice on the Altix. But there it is quite small.

The behaviour of the XT4 is much more constant in comparison to the other

Table 6. Comparison of performance results from three platforms. The columns contain the same kind of information as the rows in the other performance tables.

lattice	platform	number of cores	overall perf. [Gflop/s]	comp. perf. per core [Mflop/s]	MPI perf. per proc. [MByte/s]	MPI overhead [%]
$24^3 \times 48$	Cray XT4	64	34	621	631	14
		256	132	666	517	22
		512	291	721	693	21
	IBM p690	64	23	737	115	51
		256	197	1279	441	40
		512	246	955	247	50
	SGI Altix 4700	64	57	1202	541	26
		128	124	1687	424	43
		256	245	2641	321	64
		512	445	2443	330	64
$48^3 \times 96$	Cray XT4	256	141	639	455	14
		512	282	627	601	12
	IBM p690	64	46	911	281	21
		128	104	971	503	16
		512	189	530	177	30
	SGI Altix 4700	64	52	954	464	14
		128	100	947	450	17
		256	194	977	379	22
		512	406	1106	350	28

platforms. Both the compute performance per core and the MPI bandwidth vary much less. The machine has the smallest MPI overhead which also is quite constant. From this one would expect very high scalability what would have been interesting to check.

5. Conclusion

In this article we have used the BQCD simulation program to compare three communication modes and two node types on the SGI Altix at LRZ. In all cases we observed very good scaling up to 2048 cores except for *shmem* communication which scaled well up to 256 cores. According to our measurements pure MPI communication is the method of choice. Combining MPI with OpenMP or replacing it by *shmem* gave substantially lower performance. On high-density nodes 65–79 % of the high-bandwidth performance was achieved. In a strictly memory bound situation on would expect this value to be about 50 %.

A surprise to us was the discovery that data caches play such an important role

on the Altix. The cache size is 9 MByte per dual core chip. In our measurements better cache utilisation compensates decreasing network bandwidth when increasing the number of cores. The discussion of *shmem* as one communication method available on the Altix and the comparison with other machines led us to consider the effect of network latencies. Although network latencies were not measured directly we could in some cases indirectly see their effect on the effective network bandwidth and the duration of global reduction operations.

It is interesting to see the interplay of network bandwidth, network latency, and the memory hierarchy when studying strong scaling of a real world application on the Altix 4700 and other machines.

Acknowledgements

The computations were performed on the IBM p690 cluster at Jülich Supercomputer Centre (JSC), Jülich, Germany, on the Cray XT4 at Scientific Computing Ltd (CSC), Espoo, Finland, and on the SGI Altix 4700 at Leibniz Supercomputing Centre (LRZ), Garching, Germany. Computer time at JSC and CSC was provided by the DEISA Consortium.

References

[1] G. Schierholz and H. Stüben, *Optimizing the Hybrid Monte Carlo Algorithm on the Hitachi SR8000*, in S. Wagner, W. Hanke, A. Bode and F. Durst (Eds.), High Performance Computing in Science and Engineering, Munich 2004, Springer-Verlag, 385–393.

[2] T. Streuer and H. Stüben, *Simulations of QCD in the Era of Sustained Tflop/s Computing*, in C. Bischof, M. Brückner, P. Gibbon, G. Goubert, T. Lippert, B. Mohr, F. Peters (Eds.), Parallel Computing: Architectures, Algorithms and Applications, NIC Series, Vol. 38 (2007), 535–542.

[3] S. Duane, A. Kennedy, B. Pendleton and D. Roweth, Phys. Lett. B 195 (1987) 216.