PROCEEDINGS OF SCIENCE

Performance evaluation and tuning of lattice QCD on the next generation Blue Gene

Jun Doi
IBM Research, Tokyo Research Laboratory
1623-14 Shimo-tsuruma Yamato-shi Kanagawa-ken, Japan
E-mail: doichan@jp.ibm.com

Blue Gene/P is a next generation supercomputer following the technology of Blue Gene/L that supports ultra-scalability and efficient performance for many applications, including lattice QCD. We had experience optimizing lattice QCD problems on Blue Gene/L and we reported on the good performance and scalability. In this paper we describe how we are porting and tuning the lattice QCD code on Blue Gene/P and report on the initial code tuning efforts that take advantage of the new architectural features.

The XXV International Symposium on Lattice Field Theory
Regensburg, Germany
July 30th - August 4th 2007

PoS(LATTICE 2007)032

1. Introduction

IBM® System Blue Gene®/P Solution [1] has been announced as the next generation of Blue Gene, and it upgrades the architecture of Blue Gene/L [2] to allow ultra-scalability and efficient performance on a number of applications such as lattice QCD computations. Blue Gene/P has additional features that can potentially improve the performance of applications. The performance results for lattice QCD on Blue Gene/L have been reported earlier [3][4] and lattice QCD is also a very suitable and scalable application. We have developed a kernel library for lattice QCD on Blue Gene/L [4] and the sustained performance was over 30% of the peak performance. In this paper, we describe how we tuned the lattice QCD on Blue Gene/P using its new features and show the performance. In Section 2, we give an overview of Blue Gene/P. In Section 3, we describe how we tuned the lattice QCD work for computation and communication. In Section 4 we present the measured performance on Blue Gene/P.

2. The next generation Blue Gene, Blue Gene/P

Blue Gene/P follows the Blue Gene/L approach in its design and extends the performance into petaflops. Table 1 shows the comparison between Blue Gene/L and Blue Gene/P.

	Blue Gene/L	Blue Gene/P
Processor	2x PPC 440 (700 MHz) no SMP	4x PPC 450 (850 MHz) with SMP
Main memory	512 MB/1 GB DDR / 5.6 GB/s	2 GB DDR2 /13.6 GB/s
L3 cache	4 MB	8 MB
Peak performance / node	5.6 GFLOPS	13.6 GFLOPS
Torus network	175 MB/s x12 = 2.1 GB/s	425 MB/s x12 = 5.1 GB/s with DMA

Table 1: Major differences between Blue Gene/L and Blue Gene/P

One of the changes is the processor. Blue Gene/L has a dual core PowerPC 440 on each compute nodes with a clock speed of 700 MHz. A Blue Gene/P node uses a quad core PowerPC 450 and the clock speed has been increased to 850 MHz. The quad core processor also supports 4-way SMP, so the performance per node is increased about 2.4 times over Blue Gene/L. Blue Gene/L has two run time modes, a virtual node (VN) mode and a coprocessor (CO) mode. Blue Gene/P has an SMP mode instead of a CO mode and with the SMP mode, we can use 2 GB of memory, which makes it possible to run more applications on Blue Gene/P. Since the processors per core are doubled, the size of the L3 cache, the memory bandwidth, and the network bandwidth are also doubled. Blue Gene/P also has a 3-dimensional torus network for communication and a DMA is used to transfer data directly to and from memory and between different nodes using that torus network. Because of the DMA engine, the cores are freed from managing data communications and communication and computation are easily overlapped.

3. Tuning lattice QCD computations on Blue Gene/P

We are developing kernel code for lattice QCD computations for Blue Gene/P, with support for the Wilson Dirac operator and the even-odd preconditioned Wilson Dirac operator. The equation of the Wilson Dirac operator is

$$D(n,m)=\delta(n,m)-\kappa\sum_{\mu=1}^{4}\left\{(1-\gamma_\mu)U_\mu(n)\delta(n+\hat{\mu},m)+(1+\gamma_\mu)U_\mu^\dagger(n-\hat{\mu})\delta(n-\hat{\mu},m)\right\}, \quad (1)$$

and the gamma matrices that our lattice QCD kernel library originally supported are:

$$\gamma_1=\begin{pmatrix}0&0&0&-i\\0&0&-i&0\\0&i&0&0\\i&0&0&0\end{pmatrix}\quad \gamma_2=\begin{pmatrix}0&0&0&-1\\0&0&1&0\\0&1&0&0\\-1&0&0&0\end{pmatrix}\quad \gamma_3=\begin{pmatrix}0&0&-i&0\\0&0&0&i\\i&0&0&0\\0&-i&0&0\end{pmatrix}\quad \gamma_4=\begin{pmatrix}0&0&-1&0\\0&0&0&-1\\-1&0&0&0\\0&-1&0&0\end{pmatrix} \quad (2)$$

The data layout of the array of a gauge field array and a quark field array in our lattice QCD kernel are represented as follows in FORTRAN array definitions:

```
COMPLEX*16 Gauge_field(row, col, x, y, z, t, mu)
COMPLEX*16 Quark_field(color, spin, x, y, z, t)
```

The data layouts for even-odd preconditioned are:

```
COMPLEX*16 Gauge_field_eo(row, col, x, y, z, t, parity, mu)
COMPLEX*16 Quark_field_eo(color, spin, x, y, z, t, parity)
```

We put μ in the outermost of the data layout of the gauge field array. This data layout is good to calculate the Wilson Dirac operator separately for each direction, X, Y, Z, and T.

3.1 Tuning the calculations of the Wilson Dirac operator

Blue Gene has 2 floating point processor units in each core, which together are called a "double FPU". The double FPU has an SIMD instruction set that can calculate two double precision floating point operations with one instruction. There are some special instructions for complex number arithmetic, so we can use double FPU instructions for all of the calculations of the Wilson Dirac operator. We used inline assembly code to put applicable double FPU instructions at the appropriate points in the instruction pipeline.

The Wilson Dirac operator can be separated into 8 calculations, since there are calculations for 4 directions and each direction has a calculation for forward and backward. Each calculation consists of a multiplication of a gamma matrix and a multiplication of a 4x3 quark spinor and 3x3 gluon matrix. The gamma matrix has symmetry and multiplying a gamma matrix by the 4x3 quark spinor eliminates half of the calculations for the multiplications of quark spinor and gluon matrix. Here are examples for the cases of Gamma 1 and 2 showing how we can eliminate half of the calculations.

$$(1-\gamma_1)\begin{bmatrix}U_1\cdot s_1\\U_1\cdot s_2\\U_1\cdot s_3\\U_1\cdot s_4\end{bmatrix}=\begin{bmatrix}U_1\cdot(s_1+is_4)\\U_1\cdot(s_2+is_3)\\-iU_1\cdot(s_2+is_3)\\-iU_1\cdot(s_1+is_4)\end{bmatrix},\quad (1-\gamma_2)\begin{bmatrix}U_2\cdot s_1\\U_2\cdot s_2\\U_2\cdot s_3\\U_2\cdot s_4\end{bmatrix}=\begin{bmatrix}U_2\cdot(s_1-s_4)\\U_2\cdot(s_2+s_3)\\U_2\cdot(s_2+s_3)\\-U_2\cdot(s_1-s_4)\end{bmatrix} \quad (3)$$

Therefore we can calculate the operator in 3 steps, (i) making a 2x3 spinor by adding 2 spins, (ii) multiplying the 2x3 spinor and a 3x3 gluon matrix, and (iii) multiplying by -κ and adding to the output of a 4x3 spinor.

3.1.1 Making a 2x3 spinor

We add 2 pairs of spins to make a 2x3 spinor, and while adding the spins one of the spins is multiplied by ±1 or ±*i* in accord with the gamma matrix shown in Table 2.

gamma 1 and 3		gamma 2 and 4	
calculation	double FPU instruction	calculation	double FPU instruction
$t = s_a + is_b$	t = FXCXNPMA(s_a, s_b, 1.0)	$t = s_a + s_b$	t = FXCPMADD(s_a, s_b, 1.0)
$t = s_a - is_b$	t = FXCXNSMA(s_a, s_b, 1.0)	$t = s_a - s_b$	t = FXCPNMSUB(s_a, s_b, 1.0)

Table 2: Double FPU instructions to make a 2x3 spinor

3.1.2 Multiplying the 2x3 spinor and a 3x3 matrix

Multiplying a spin and a 3x3 matrix consists of 9 complex multiplications and 6 complex additions, but we can do the complex multiplications with 2 double FPU instructions. By using floating multiply-add (FMA) instructions, the complex addition can be included in the complex multiplication calculation as shown in Table 3, so we need only 18 instructions to calculate the product of a spin and a 3x3 matrix. For backward calculation, a Hermitian matrix is multiplied by a spin. Each element of our Hermitian matrix is a conjugate complex number. We can use another set of double FPU instructions to do the conjugate complex multiplications without changing the values of the matrix.

forward		backward (Hermitian matrix multiply)	
calculation	double FPU instruction	calculation	double FPU instruction
$v = u_0 * t_0$	v = FXPMUL(t_0,u_0) v = FXCXNPMA(v, t_0, u_0)	$v = \sim u_0 * t_0$	v = FXPMUL(t_0,u_0) v = FXCXNSMA(v, t_0, u_0)
$v = v + u_1 * t_1$	v = FXCPMADD(v, t_1, u_1) v = FXCXNPMA(v, t_1, u_1)	$v = v + \sim u_1 * t_1$	v = FXCPMADD(v, t_1, u_1) v = FXCXNSMA(v, t_1, u_1)

Table 3: Double FPU instructions for complex multiplication

3.1.3 Multiplying by kappa

Finally we multiply the constant value $-\kappa$ by the 2x3 spinor and add the result to the output 4x3 spinor. We can use the same instructions in Table 2 and multiply by $-\kappa$ instead of the 1.0.

3.2 Tuning the communications of the Wilson Dirac operator

3.2.1 Parallelizing lattice QCD on Blue Gene/P

Parallelizing the lattice QCD computation is simple because each lattice point affects all of the neighboring nodes, so if we simply divide the original lattice into small lattices, then the boundary data should be exchanged between the neighboring small lattices as shown in Figure 1.

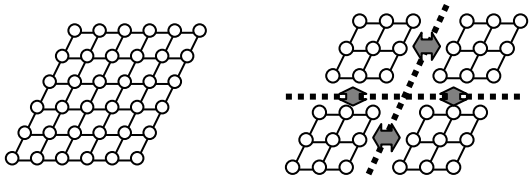

Figure 1: Parallelizing the lattice QCD computation

Blue Gene has a 3D torus network, so each node has 6 neighboring nodes in a 3D space. Blue Gene/P has 4 cores in each node and core to core data communications can be treated as a virtual 4th dimensional torus network in the VN mode. Then the topology of a 4D lattice and a virtual 4D torus are the same, so we can simply divide the lattice by the size of the torus in the X, Y, Z, and T directions and the boundary data only needs to be exchanged between

neighboring nodes. We map the in-node communications in the X direction of the lattice because the X direction is the innermost of the gauge and quark spinor arrays and this results in better performance. In the SMP mode, we divide the Y, Z, and T directions of the lattice by the size of the 3D torus and we use OpenMP to parallelize the computations in each node. In Figure 2, there are 2 examples of the parallelization of loops in the Y direction.

```
!$omp parallel do private(x)
do i=1,Nt*Nz
   do x=1,Nx
!    computation for Y direction
   end do
end do
```

(a) Outermost loop parallelization

```
!$omp parallel private(i,x)
dx=Nx/omp_get_num_threads()
sx=dx*omp_get_thread_num()
do i=1,Nt*Nz
   do x=sx,sx+dx-1
!    computation for Y direction
   end do
end do
!$omp end parallel
```

(b) Innermost loop parallelization

Figure 2: 2 ways of parallelization using OpenMP in FORTRAN

The outermost loop parallelization is the easiest way to parallelize using OpenMP, since we only put OpenMP directives before the loops in the serial code. Therefore we can use the same code as for the VN mode, but there are problems: (i) If Nt*Nz in Figure 2 cannot be divided by 4, then the performance does not scale, and (ii) The patterns of the data access are different among the loops for each of the directions, which may decrease the reusability of the data in cache.

The innermost loop parallelization is another method for parallelization. We have to modify the VN mode code, but the pattern of data access is the same as for the VN mode and is unchanged for the calculations in all directions. This is good for reusing the data in cache.

3.2.2 Boundary data exchange using DMA engine

Using the new DMA engine in Blue Gene/P, we can send data from the local memory of one node directly to the local memory of another node. We can also overlap computations or other procedures while the DMA is sending the data to the destination. The DMA engine can also handle multiple data exchange at the same time. The performance of the DMA is better if data is in larger packets, so we prepare boundary data for each direction from each buffer and send all of the data at the same time to each destination.

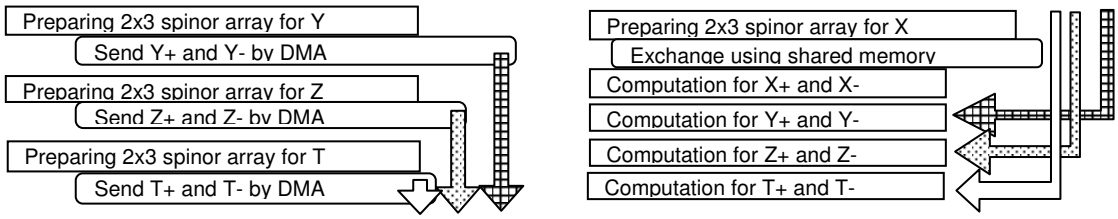

Figure 3: Overlapping communication and computation

Figure 3 shows how we overlap computation and communication using DMA for the communications. For example, while sending the boundary data for Y+ and Y- we can do the preparations and sending for Z, T, and do the computation for X. First we prepare the boundary data array separately for each direction. As described in Section 3.1, we have to calculate only half of the multiplications for the gauge matrix and the 4x3 spinor. For the boundary data

exchange, we also have to exchange only a 2x3 spinor for each lattice point. We accumulate all of the 2x3 spinors into an array. For the forward direction, we multiply a gauge matrix by a 2x3 spinor and store the result in the array. For the backward direction, we multiply a gauge matrix after receiving it at the destination. For the X direction we do not send any data outside of the node, but we exchange the boundary data between the cores using shared memory in the VN mode, and we do not exchange any boundary data for the X direction in the SMP mode.

4. The performance of lattice QCD on Blue Gene/P

We ran the benchmark program for lattice QCD on a half-rack with 512 nodes of Blue Gene/P. In the benchmark program, the gauge matrix array is filled with random values between -1.0 and 1.0 and the CG inverter performs the Wilson Dirac operator as long as the spinor is well converged. We used the BiCGStab method in our benchmark program.

	Blue Gene/L		Blue Gene/P		
Lattice size	MFLOPS (vs peak)	w/ CG	MFLOPS (vs peak)	w/CG	speed up /node
16x16x16x16	776.7 (27.74%)	602.7 (21.52%)	596.3 (17.54%)	463.5 (13.63%)	x 1.54
16x16x16x32	856.6 (30.59%)	672.4 (24.01%)	709.9 (20.79%)	575.7 (16.93%)	x 1.66
24x24x24x24	933.4 (33.34%)	749.3 (26.76%)	928.7 (27.31%)	744.9 (21.91%)	x 1.99
24x24x24x48	940.1 (33.58%)	769.3 (27.47%)	1037.4 (30.51%)	827.3 (24.33%)	x 2.21
32x32x32x32	1015.9 (36.28%)	807.6 (28.84%)	1132.3 (33.30%)	889.6 (26.16%)	x 2.23
32x32x32x64	1019.1 (36.40%)	609.0 (21.75%)	1192.4 (35.07%)	918.4 (27.01%)	x 2.34

Table 4: Performance of Wilson Dirac operator on 512 nodes in VN mode

	Blue Gene/L		Blue Gene/P		
Lattice size	MFLOPS (vs peak)	w/ CG	MFLOPS (vs peak)	w/CG	speed up /node
16x16x16x16	691.6 (24.70%)	496.9 (17.75%)	397.8 (11.70%)	313.3 (9.21%)	x 1.15
16x16x16x32	749.7 (26.77%)	568.6 (20.31%)	524.4 (15.42%)	431.5 (12.69%)	x 1.40
24x24x24x24	873.4 (31.19%)	668.0 (23.86%)	735.4 (21.63%)	597.5 (17.57%)	x 1.68
24x24x24x48	872.6 (31.17%)	692.6 (24.74%)	834.0 (24.53%)	670.9 (19.73%)	x 1.91
32x32x32x32	961.0 (34.32%)	751.7 (26.84%)	910.5 (26.78%)	730.8 (21.49%)	x 1.89
32x32x32x64	968.2 (34.58%)	634.7 (22.67%)	974.4 (28.66%)	777.7 (22.87%)	x 2.07

Table 5: Performance of preconditioned Wilson Dirac operator on 512 nodes in VN mode

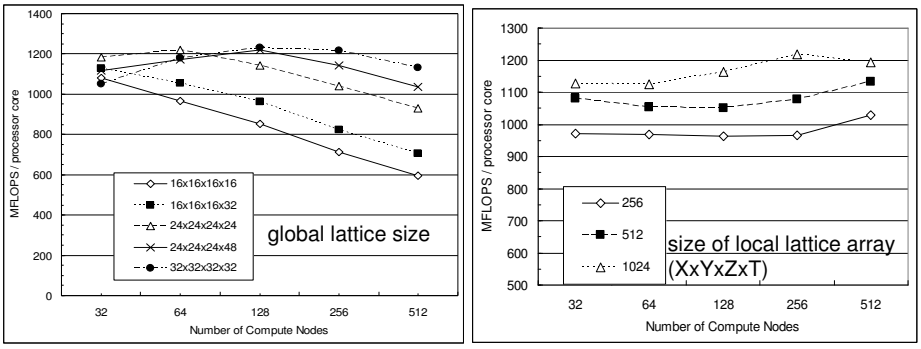

Figure 4: Strong and weak scaling of Wilson Dirac operator in VN mode on Blue Gene/P

Table 4 and Table 5 show the sustained performance per processor core in VN mode. Comparing to Blue Gene/L, if the lattice size is small, the performance speedup in Blue Gene/P is much smaller than 2.4. There are two reasons for this. One is the change of the L1 cache

mode. The L1 cache for Blue Gene/L is a write-back cache, but a write-through cache is used in Blue Gene/P. We can reuse the modified data in the L1 cache in write-back mode, which is good for small amounts of data. The second reason is the methods of communication. In Blue Gene/L, data is sent directly to and received directly by data the registers [4], but in Blue Gene/P the data is put into arrays and sent using DMA. The overhead is larger when the amount of data is small, and the performance of the DMA is also better for larger amounts of data.

Figure 4 shows the scalability of the Wilson Dirac operator on Blue Gene/P, where the left graph shows strong scaling and the right shows weak scaling. The Wilson Dirac operator performs very well for weak scaling and well for strong scaling for larger data.

	Outermost loop parallelization		Innermost loop parallelization	
Lattice size	MFLOPS (vs peak)	w/ CG	MFLOPS (vs peak)	w/CG
16x16x16x16	1620.3 (11.91%)	1048.1 (7.71%)	2426.0 (17.84%)	1434.9(10.55%)
16x16x16x32	2333.8 (17.16%)	2079.6 (15.29%)	2869.6 (21.10%)	2062.2(15.16%)
24x24x24x24	2613.9 (19.22%)	2079.6 (15.29%)	3622.2 (26.63%)	2595.3 (19.08%)
24x24x24x48	3275.0 (24.08%)	2619.6 (19.26%)	4037.2 (29.69%)	3065.1 (22.54%)
32x32x32x32	3715.48 (27.32%)	2872.1 (21.12%)	4268.5 (31.39%)	3339.3 (24.55%)
32x32x32x64	3925.5 (28.86%)	2985.7 (21.95%)	4631.4 (34.05%)	3515.0 (25.85%)

Table 6: Performance of Wilson Dirac operator on 512 nodes in SMP mode

Table 6 shows performance comparisons for the outermost loop parallelization and innermost loop parallelization of the OpenMP implementation in the SMP mode. The performance is better in the innermost loop parallelization.

5. Summary

We have optimized the lattice QCD code on Blue Gene/P and we achieved over 30% sustained performance even though the L1 cache mode uses write-through mode, which is a very good result compared to Blue Gene/L. We will optimize more for preconditioned Wilson and in SMP mode and we also hope to test our kernel in the actual lattice QCD applications.

Acknowledgements

We optimized and ran our code on a Blue Gene/P system installed at the IBM T.J.Watson Research Center. We would like to thank James Sexton for providing access to the machine and support, and the Deepcomputing team members of Tokyo Research Laboratory for their advice.

References

[1] *The IBM System Blue Gene Webpage*, http://www.ibm.com/servers/deepcomputing/bluegene.html

[2] A. Gara et al. , *Overview of the Blue Gene/L System Architecture,* IBM Journal of Research and Development, Vol. 49, No. 2/3, pp. 195-212, 2005.

[3] P. Vranas et al. , *The Blue Gene/L Supercomputer and Quantum ChromoDynamics*, Gordon Bell finalist, SCl06, 2006.

[4] J. Doi, H. Samukawa, H. Matsufuru, and S. Hashimoto, *Highly Parallelization of Lattice QCD program for Blue Gene*, IPSJ ACS, vol. 14, 2006.