PROCEEDINGS OF SCIENCE

Development of QCD code on a CELL Machine

Shinji Motoki
Graduate School of Bio-Sphere Science, Hiroshima University, 1-7-1 Kagamiyama, Higashi-Hiroshima 739-8521, Japan
E-mail: motoki-shinji@hiroshima-u.ac.jp

Atsushi Nakamura
Research Institute for Information Science and Education (RIISE), Hiroshima University, 1-7-1 Kagamiyama, Higashi-Hiroshima 739-8521, Japan
E-mail: nakamura@riise.hiroshima-u.ac.jp

We report our experience of developing a QCD code on a CELL BE machine. First we describe what CELL BE is, and why it is worthwhile studying the possibility of simulating lattice QCD on this new multicore processor. Then we discuss our code development process and the performance of fermion $matrix \times vector$ calculations, which appear in a standard conjugate-gradient-type solver. Our first result, 20 GFLOPS, is far from the theoretical peak speed of 400 GFLOPS. We discuss the cause of this low value and a possible remedy.

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

PoS(LATTICE 2007)040

1. Introduction – What is CELL BE ?

The simulation of lattice QCD requires high performance computer resources. Future calculations will be performed on next-generation Peta-FLOPS machines, and together with recent developments in algorithms, we can more realistically describe the quark-gluon world, i.e., near the real u, d quark masses with chiral fermions and dynamical quarks. In addition to state-of-the-art computations by big collaborations, it is desirable to have a machine with a performance of several hundred Giga- or Tera-FLOPS at each of our laboratories to achieve breakthrough studies based on new ideas.

In this report we discuss the CELL BE (Broadband Engine) as one such candidate machine, and present our first attempt to develop a QCD code on the machine. A similar study is reported in Ref. [1].

CELL BE is a new multicore processor developed by SONY, IBM and Toshiba for the PS3 game machine. One CELL consists of one PPE (PowerPC processor element) and 8 SPEs (synergistic processing elements). Its theoretical peak speed is 200 GFLOPS. Figure 1 shows a schematic diagram of one CELL.

The CELL is available as a personal computer including

- PS3 itself (Sony) [1]
- CELL Reference Set (Toshiba)
- QS20 (IBM) [2]
- others.

A CELL BE software development kit (SDK) is available from the IBM website [2]. A useful instruction material can be found in Ref. [3].

The QS20 has two CELLs, and therefore its peak speed is 400 GFLOPS. See Fig. 2. It is desirable to employ a machine with this processor for lattice QCD calculations. We are developing a QCD code (Quench and HMC) using a CELL in order to accumulate experience and to study the potential of the CELL BE as a machine for lattice QCD calculations.

1.1 Limitations

The following points are known as disadvantages or limitations of using the present CELL as a high-performance machine for numerical simulations.

- Only single precision is supported, i.e., double-precision calculations are performed by software and are slow. We found that the calculations take ca. 7 times longer than the single-precision case.
- Only C++ is available; Fortran is not available.

[1] One can construct a development and computational environment using PS3 by installing Linux. Currently, Fedora Core is usually used. Yellow Dog Linux and other distributions will be available for CELL in future.

[2] QS21 has come recently.

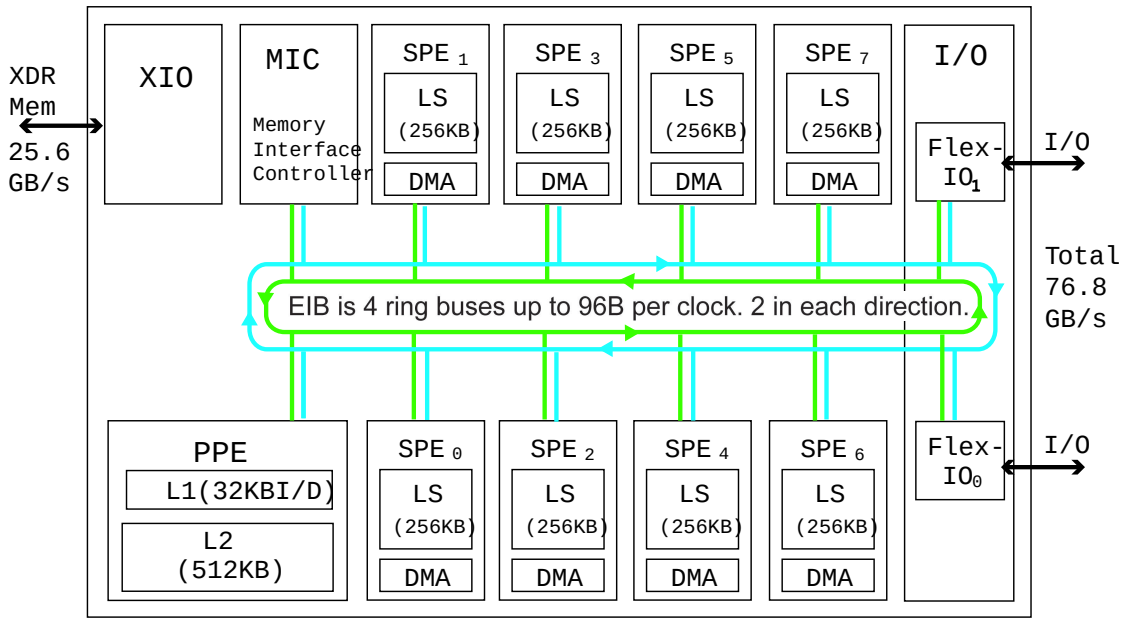

Figure 1: CELL BE Structure

- SPEs have a very small memory, LS (local storage), i.e., 256 KByte/SPE. Consequently, programming is difficult.

Some of these disadvantage will be overcome in the near future.

2. Code Development

We start from our previous QCD code in Fortran 90, LTKf90 [4]. This code is written using the Fortran 90 *module*. We replaced the *module* part of Fortran 90 with C++ *class*, and then made a few modifications to most computational parts. Examples of modifications are given below.

Fortran90

```
   DO nu = 1, 4

   if(nu==mu) cycle

c      x+nu temp2
c        .----------.
c        I          I
c  temp1 I          I
c        I          I
c        .          .
c        x         x+mu
```

C++

```
for( int nu = 0; nu < 4; nu++)
{
  if(nu == mu){continue;}

  //        x+nu temp2
  //          .----------.
  //          I          I
  //    temp1 I          I
  //          I          I
  //          .          .
  //          x         x+mu
```

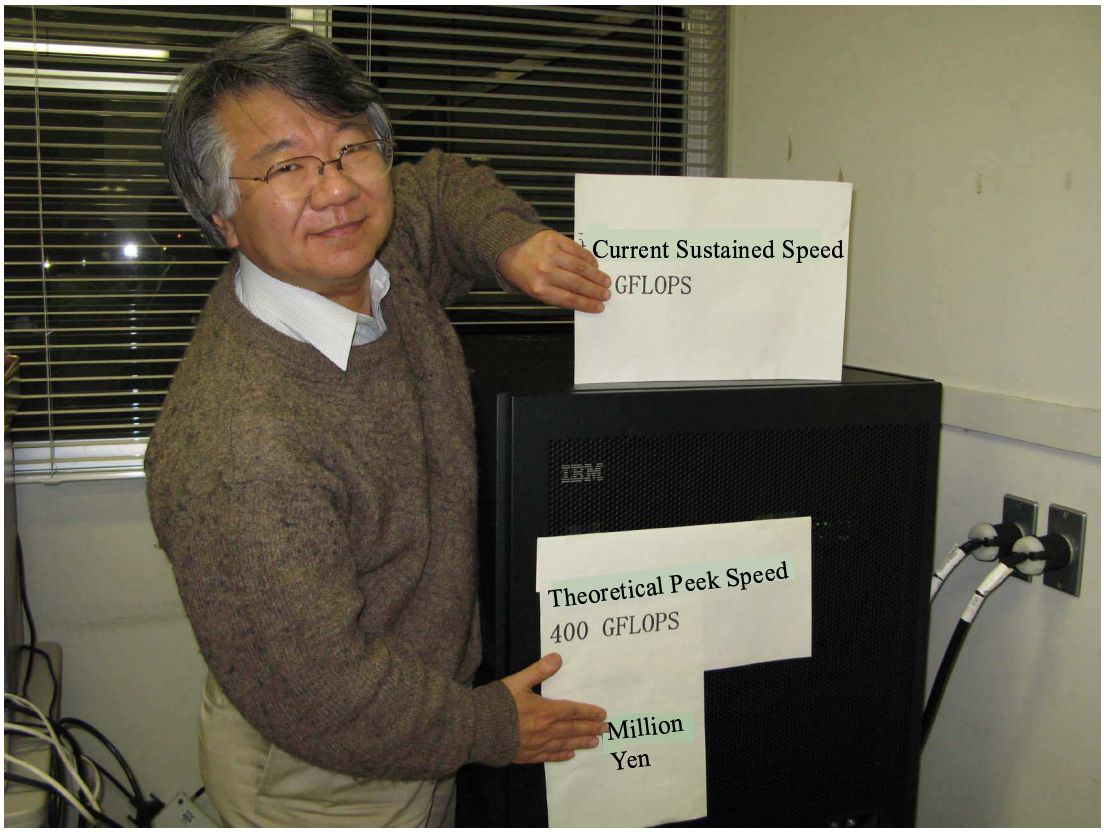

Figure 2: QS20 (IBM) in our laboratory

```
      temp1 = u(nu)                    temp1 = u[nu];
      temp2 = nu.gshift.u(mu)          temp2 = gshift1(nu, u[mu]);
      temp3 = temp1 * temp2            temp3 = temp1 * temp2;
      temp1 = mu.gshift.u(nu)          temp1 = gshift1(mu, u[nu]);
      staple = staple                  staple = staple
&           + (temp3.prodAD.temp1)           + prodAD(temp3,temp1);
 ENDDO                                  }
```

3. Test Code on CELL

The most time-consuming part of the current standard QCD code is the *matrix* $\times$ *vector* calculations appearing in fermion CG (conjugate gradient).

```
  Code A

//------------------------//
   void function WxVect()
```

```
//------------------------//
/* for x = 1, Nx          */
   for y = 1, Ny
   for z = 1, Nz
   for t = 1, Nt
     for c = 1, 3

     for alpha = 1, 4
```

$$Y_{x,y,z,t,c,alpha} = \sum W^{x',y',z',t',c',alpha'}_{x,y,z,t,c,alpha} X_{x',y',z',t',c',alpha'}$$

```
//---------------------------//
  void function para_WxVect()
//---------------------------//
  for x = 1, Nx
    para_WxVect(W,X,Y);
```

We rewrote this part as a function (see the above), and then the function *para_WxVect* was transferred to the SPEs. PPE and SPE codes generally have the following structure:

PPE code

1. Set pointers that point to data regions of the SPE output.
2. Create SPE threads.
3. Start SPE threads.
4. Wait until the end of SPE threads.

SPE code

1. DMA (Direct Memory Access) transportation from main memory to LS.
2. Calculations.
3. DMA transportation of the data in LS to the main memory in PPE.

It is practical to construct a *structure* for storing DMA transferred data.

```
// One constructs a structure for storing DMA transferred data.

typedef struct
{
    float   a[N];
```

```
    volatile float *result_p;
    volatile int flag;
} __attribute__ ((aligned(128))) dma_data_t;

        // Prepare the transfer.
        prepare(Nspe);

        // Transfer the data
        dma_trans(Nspe,a);
        // Calculation on SPE;
        //   At the first state we call a function serially.
        //   When the function is checked and ready, we move
        //   it to SPE.
        for(int ispe=0; ispe<Nspe; ispe++){
                execute(ispe);
        }

        // Post process
        dma_done();
```

4. Performance

We measured the time required for the fermion *matrix* $\times$ *vector* calculations. See Table 1. From this data, it is clear that the communication overhead particularly for the 1. time is too large. If we convert this data to the floating point operation, it is 20 GFLOPS, i.e., 5% of the theoretical peak speed.

	DMA_GET	Calculation	DMA_PUT
1. time	1072	36	215
2. time	519	36	215
3. time	514	36	215
4. time	338	36	215

Table 1: Time required for data transfer and calculation in μ second. $N_y = 4, N_z = 2, N_t = 2$ The data transferred is 10.4 KByte for DMA_GET and 9.2 KByte for DMA_PUT. We repeated this transfer/calculation 1000 times and took the average of the results.

5. Concluding Remarks

In this study, we used a master-slave parallel strategy, where the PPE is the master and the SPEs are slaves. The communication overhead is very large. Communication here is in broadcasting

form, and the EIB with 4 ring buses in Fig. 1 is not fully used. We are now studying MPI parallel programing.

It is also necessary to consider the following.

1. The overlap of communication and calculation.

2. Finding a good granularity of parallelization for realistic lattice sizes.

Concerning the second point, consider a typical lattice size, $N_x \times N_y \times N_z \times N_t = 32 \times 32 \times 32 \times 32$. We need sufficient memory for a gauge configuration and at least two fermion vectors in a CG calculation, i.e., 350 MByte is required for the single-precision case. A PPE has sufficient memory, but the total LS of eight SPEs, two MByte/CELL, is far from sufficient. Thus, we should divide the lattice (i.e., decrease the granularity). On the other hand, one DMA transfer can provide 16 KByte, and therefore we need many DMA transfers. The overlap of the communications and the calculations is essential.

Acknowledgments

We thank Y. Tsuchimoto for his invaluable support in the development of the system environment for our CELL machine. This work was supported by Grants-in-Aid for Scientific Research from Monbu-Kagaku-sho (the Ministry of Education, Culture, Sports, Science and Technology), No. 17340080. We used SR11000 at Hiroshima University and SX-8 at RCNP, Osaka University to obtain the data used for comparison.

References

[1] N. Meyer et al., “QCD on the Cell processor", PoS(Lattice 2007) 039.

[2] `http://www-128.ibm.com/developerworks/power/cell/`

[3] Lecture on CELL at MIT: `http://cag.csail.mit.edu/ps3/index.shtml`

[4] S. Choe, S. Muroya, A. Nakamura, C. Nonaka, T. Saito and F. Shoji, “Lattice Tool Kit in Fortran 90”, Nucl. Phys. B (PS) 106 (2002) 1037-1039. http://nio-mon.riise.hiroshima-u.ac.jp/ LTK/
Version 2.0 will soon appear.