Conf-9409269--9

ANL-HEP-CP-94-92

Lattice QCD calculation using VPP500

Seyong Kim[a]* and Shigemi Ohta[b]†

[a]HEP Division, Argonne National Laboratory, 9700 South Cass Avenue, Argonne, Illinois 60439, USA

[b]Institute of Physical and Chemical Research (RIKEN), Wako-shi, Saitama 351-01, Japan

A new vector parallel supercomputer, Fujitsu VPP500, was installed at RIKEN earlier this year. It consists of 30 vector computers, each with 1.6 GFLOPS peak speed and 256 MB memory, connected by a crossbar switch with 400 MB/s peak data transfer rate each way between any pair of nodes. The authors developed a Fortran lattice QCD simulation code for it. It runs at about 1.1 GFLOPS sustained per node for Metropolis pure-gauge update, and about 0.8 GFLOPS sustained per node for conjugate gradient inversion of staggered fermion matrix.

1. INTRODUCTION

A new super-computing system was installed at Institute of Physical and Chemical Research (RIKEN) at the end of March, 1994. It serves various researches in physics, chemistry and biology at RIKEN, and a significant amount of its cpu time is allotted for numerical calculations in lattice quantum chromodynamics (QCD).

2. HARDWARE

The core of this system is a new "vector parallel" supercomputer, Fujitsu VPP500. It consists of two main-frame cpu's and 30 "processor elements (PE's)", all connected by a crossbar switch. In addition, there are a 8 GB semiconductor disk, 40 GB magnetic disk to store the system and user files, 60 GB RAID3 and 80 GB RAID7 disks for data storage, and 10 TB tape archive.

One of the main-frame cpu's acts as a front-end for the computer. The other, the "control processor (CP)", controls I/O between the PE's and the front-end, disks, and other peripheral devices. It has a 100 MHz scalar cpu with LIW architecture and 128 MB memory. The cpu is made of GaAs and ECL chips and the memory consists of 1 Mbit SRAMs with 18 ns latency.

A single PE consists of the same 100 MHz scalar cpu as the CP, a vector processor with 1.6 GFLOPS peak speed, and 256 MB memory. The processors are again made of GaAs and ECL chips. The scalar cpu has 32 32-bit general purpose registers and 32 64-bit floating-point registers. The vector processor has 128 KB vector registers, 2 KB mask registers, and pipelines for multiplication, add/logic, division, mask, load and store operations. All the floating point operations are done in IEEE 64-bit format. The memory consists of the same kind of SRAMs as in the CP.

The crossbar switch is again a combination of GaAs and ECL chips. It provides data communication at the rate of 400 MB/s each way between any pair of PE's. It also provides data transfer to the outside through the 8 GB semiconductor disk connected to the CP, at more than 50 MB/s to the fastest RAID3 disk, about 20 MB/s to the 40 GB system disk, about 10 MB/s to the 80 GB RAID7, and a few MB/s to the tape archive which is connected through another crossbar switch that connects the entire super-computing system to the local area network.

3. SYSTEM SOFTWARE

The front-end is run by an Unix SVR4 system. The CP and PE's are run by modified Unix systems with enhancement for communications through the crossbar switch. This enhancement

*S.K. is supported by the U.S. Department of Energy, Division of High Energy Physics, Contract W-31-109-ENG-38. He also thanks RIKEN for its support in the early stage of this work.
†Talk presented by S.O.

allows arbitrary partitioning of the PE's. At the moment the 30 PE's are divided into 28-PE and 2-PE partitions during the night, and 16-PE, 8-PE, 4-PE and 2-PE partitions during the day.

Users edit their application programs on the front-end which is reachable from within the local area network. Compilation of the programs and production run are done only in the batch mode handled by a modified NQS systems running in cooperation on both the front-end and CP.

Three different programming languages, Fortran, C, and assembler, are provided. Fortran is based on the Fortran 77 standard with enhancement of vector and parallel directives as comment lines. C is based on its ANSI standard with vector and parallel directives as pragma statements. A vectorized numerical processing library is available for both languages. C is also augmented by P4 and PVM message passing libraries.

4. BENCHMARK TESTS

Procurement of this super-computing system started in late April, 1993, and took until the middle of December that year. Among the various benchmark tests in the process, the following are worthy to note: a) LINPACK speed, without any restriction for the array sizes, exceeded 30 GFLOPS. b) A CR(1) conjugate residual solver for staggered quarks of lattice QCD on a 32^4 lattice runs at more than 30 GFLOPS with 28 PE's. c) A fast Fourier transformation program for one-dimensional data of real numbers runs at about 20 GFLOPS for up to 2 GB of data.

5. INSTALLATION

The hardware installation was completed by the end of March, 1994, and test operations started immediately. By the end of April single-PE operations stabilized. Unfortunately parallel operations at first had some software problems, especially for larger partitions, but finally stabilized by the end of July. The system has been running quite well since then.

6. LATTICE QCD CALCULATION

We use Fortran for lattice QCD coding because it is more efficient than C at the moment. In Fortran, parallel directives are given by comment lines starting with "`!xocl`".

PE configuration: the parallel directive allows from one-dimensional to seven-dimensional configuration of the available PE's,

```
!xocl processor PE(N1)
!xocl processor PE(N1,N2)
```

where `N1`, `N2`, ..., denote the number of PE's in each dimension. The first example shows how to allocate `N1` PE's configured on a one-dimensional line segment, while the second shows `N1` × `N2` PE's configured on a rectangle, etc. Since we have only 30 PE's at most, we are practically limited to one-, two-, three- or four-dimensional configuration. We use the one-dimensional for simplicity.

Arrays: we define how to distribute array data by the following directives and definitions,

```
!xocl index partition&
!xocl IP=(PE,index=0:NP-1,part=band)
      complex
     *ul(0:TXYZ-1,0:3,3,3,0:NP-1)
!xocl local ul(:,:,:,:,/IP)
      complex
     *ug(0:TXYZ-1,0:3,3,3,0:NP-1)
!xocl global ug(:,:,:,:,/IP)
      equivalence (ul,ug)
      common/work/ug
```

An "`index partition`", `IP`, defines that any array index running from `0` through `NP-1` is distributed over the `NP` PE's configured in one dimension. The symbol "`&`" means the `!xocl` directive is continued to the next line. Here a `TXYZ` × 4 × 9 byte contiguous partition of a "`local`" partitioned array `ul` is stored on each PE. A `local` partitioned array is very quickly accessed by the local processors, but cannot be accessed from nonlocal ones. A "`global`" partitioned array can be accessed by either local or nonlocal processors, but less quickly. As a result of the `equivalence` statement, the `local` partitioned array `ul` and the `global` one `ug` share the same memory area on each PE. We use `local` partitioned arrays for most of the calculations and global ones for only inter-PE communications. The common statement is used for sub-procedure interface because a partitioned array name cannot appear as an argument of sub-procedures.

Data structure: again we tried from one- through four-dimensional mapping of the lattice. One-dimensional mapping gives too heavy load for data communication. Three- and four-dimensional ones are not flexible enough in changing the lattice size. We use two-dimensional mapping. Since we use one-dimensional PE configuration, we explicitly calculate the two-dimensional PE coordinates, say n1 and n2, from the one-dimensional PE identification number n0, $0 \leq$ n0 $\leq$ NP-1, like n0 = n1 + NZ * n2.

Data transfer: we move around some data from one PE to another using a combination of parallel directives "spread do" and "spread move",

```
!xocl spread do /IP
      do n=0,NP-1
        ......
!xocl spread move
        do i=0,TXYZ-1
          ......
          ug(ii,:,:,:,nn)
     *    =ul(i,:,:,:,n)
        enddo
!xocl end spread(TAG)
      enddo
!xocl end spread
      ......
!xocl movewait(TAG)
```

Here nn defines a PE to which the data are sent from the PE n, while ii defines a nonlocal array coordinate as a function of the local coordinate i. The "end spread(TAG)" directive means the data have been sent out of the local PE, but may not have reached the destination yet. The other directive "movewait(TAG)" tells the PE to wait until the sent-out data have been completely stored in the memory of the destination PE. In between these two directives each PE is free to do other tasks such as those do not affect or are not affected by the data being transferred. This way we can hide the cost of data transfer behind numerical calculations. Actual data transfer speed reaches about 330 MB/s sustained per PE one way, out of 400 MB/s peak, if we transfer more than a couple of MB at a time which is common for lattice QCD.

Numerical calculations are distributed to the PE's easily by the "spread do" directive:

```
!xocl spread do /IP
      do n=0,NP-1
        do i=0,TXYZ-1
          sl(i,:,:,:,n)
     *    =tl(i,:,:,:,n)
     *     *ul(i,:,:,:,n)
        enddo
      enddo
!xocl end spread
```

By using suitable buffer arrays, most of the calculations are done using only local partitioned array. Such calculations for SU(3) matrix and vector operations can easily reach a speed of more than 1.3 GFLOPS sustained per PE out of the 1.6 GFLOPS peak. This is achieved by the Fortran compiler alone and does not require any assembler-level optimization by hand.

Unfortunately, the inter-PE data transfer and local calculations interfere with each other in accessing the memory. This interference often reduces the overall efficiency to about 1.0 GF per PE. The interference decreases if we use higher dimensional mapping of the lattice than the current two-dimensional one.

Our test runs on the 8-PE partition of the machine for a 64×16^3 lattice achieved the speed of about 6 μs per Metropolis link update per PE and 25 ms per CG iteration for the entire lattice. These numbers should translate into about 1.1 and 0.8 GFLOPS sustained respectively. We are preparing for production runs on larger lattices using partitions with up to 28 PE's [1]. It should be faster by up to 20 %. We do not see any problem in moving up to still larger partitions, like the ones with 64 PE's or more planned at KEK, since the crossbar switch has been proved to work for as many as 128 PE's.

REFERENCES

1. S. Kim and S. Ohta, in preparation.