PROCEEDINGS OF SCIENCE

Optimization of Lattice QCD with CG and multi-shift CG on Intel Xeon Phi Coprocessor

Hirokazu Kobayashi*
Intel K. K.
E-mail: hirokazu.kobayashi@intel.com

Yoshifumi Nakamura
RIKEN AICS
E-mail: nakamura@riken.jp

Shinji Takeda
Kanazawa University
E-mail: takeda@hep.s.kanazawa-u.ac.jp

Yoshinobu Kuramashi
RIKEN AICS
E-mail: kuramasi@riken.jp

We implemented lattice QCD on Intel Xeon Phi coprocessor using intrinsics as vectorization method, and OpenMP and MPI as parallelization method. Our implementation uses double precision conjugate gradient (CG) solver which also supports multi-shift CG. We present our optimization methodology and performance for key steps in CG algorithms.

The 33rd International Symposium on Lattice Field Theory
14 -18 July 2015
Kobe International Conference Center, Kobe, Japan

*Speaker.

© Copyright owned by the author(s) under the terms of the Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License (CC BY-NC-ND 4.0).

PoS(LATTICE 2015)029

1. Introduction

Recently, microprocessor becomes to have lots of cores and Intel released Xeon Phi™coprocessor which has up to 61 cores. Our motivation is to develop high performance lattice QCD implementation which runs natively on Xeon Phi. To develop it, efficient implementation of Wilson Clover fermion operator is necessary. The operator is described below.

$$D = 1 + C - \kappa \sum_{\mu=1}^{4} ((1-\gamma_\mu)U_{+\mu}(n)\delta_{n,m+\hat{\mu}} + (1+\gamma_\mu)U_{-\mu}(n)\delta_{n,m-\hat{\mu}}) \tag{1.1}$$

$$C = \frac{i}{2}\kappa c_{sw}\sigma_{\mu\upsilon}F_{\mu\upsilon}(n)\delta_{m,n} \tag{1.2}$$

C is called clover term and $\sum_{\mu=1}^{4}((1-\gamma_\mu)U_{+\mu}(n)\delta_{n,m+\hat{\mu}} + (1+\gamma_\mu)U_{-\mu}(n)\delta_{n,m-\hat{\mu}})$ is called hopping term.

Section 2 describes implementation details of our QCD, section 3 describes performance evaluation of our QCD implementation, and section 4 concludes our work.

2. Implementation details

Previous works[1, 2] use code generator, while our work uses direct implementation using intrinsics. It makes easier to control the generated assembly code, to reduce register allocation or register spill.

There are some usage models of Xeon Phi. Offloading from host or using native run mode. we choose native run mode because next generation of Xeon Phi will be released as a self bootable CPU which will only support native mode.

Our solver uses double precision and Conjugate Gradient(CG) to solve the equation. QCD kernel is fully written in intrinsics to vectorize the code and it utilizes MPI and OpenMP parallelism. Our implementation supports overlapping of MPI communication and computation to hide communication latency.

QCD kernel heavily uses memory bandwidth, therefore saving memory bandwidth usage is a key element of high performance QCD implementation. To save memory bandwidth, several features are introduced to our implementation. Our implementation supports both normal and compressed gauge representation. Compressed gauge uses 6 complex values to represent a link and reconstruct 9 complex values from them on the fly. It can save 33% of gauge access memory bandwidth at the expense of some floating point operations. Clover term computation in CG solver is fused with hopping term computation. This reduces quark field data access. Hopping term computation can run without clover term computation for benchmark purpose. Streaming stores are used to reduce cache footprint. It makes writing data go directly to memory without polluting cache. Our implementation uses software prefetch to hide memory latency. BLAS like linear algebra in CG is fused as much as possible to reduce memory access.

PoS(LATTICE 2015)029

2.1 Data layout

Utilizing Xeon Phi vector register is one of the key element of high performance implementation of QCD. Xeon Phi supports gather and scatter operation in its instruction set[3], but these instructions latency and throughput are not as good as load or store instructions. Therefore proper data layout which avoid using gather and scatter instructions should be used.

Our implementation uses Array of Structure of Array(AOSOA) Layout along X direction. Vector length of Xeon Phi is 512bit, therefore it can contain 8 double precision floating point numbers in a vector register. Storage of quark fields is `double SC[3][4][2][8]` where last 8 contains X direction data sequentially. Even-odd precondition is used in this CG solver, therefore X direction lattice size should be multiple of 16. Thus load and store instructions are used to move data between memory and register. No size constraints for other direction. Our implementation doesn't allow to cut X direction among MPI processes to avoid complex data rearrangement.

Gauge fields storage is rearranged to enable linear memory access pattern. Both forward and backward links are stored at each site. This increases memory locality and effectiveness of prefetching gauge data. As a side effect, double storage size is required for gauge field data.

2.2 OpenMP and MPI implementation of hopping term

Xeon Phi has up to 61 cores and thread synchronization on Xeon Phi is relatively slow compared to other CPUs, therefor reducing thread synchronization is one of the key element to get high performance implementation on Xeon Phi. Our implementation overlaps MPI communication and processing of internal(non-boundary) part of hopping term while reducing thread synchronization. Figure1 illustrates this mechanism.

When hopping term processing starts, all threads start to process the boundary parts first to prepare the transmit data. After processing boundary parts, all threads are synchronized to ensure all the transmit data preparation is finished. Then master thread starts MPI communication and all the other threads start processing internal parts concurrently. After MPI communication and internal processing are done, all threads are synchronized. Because All data is received at this time and internal processing is finished, all threads are used to process the boundary data. After finishing to process the boundary data, all threads are synchronized, to ensure all hopping term processing is finished. Thus our implementation uses three synchronizations in one iteration of hopping term.

If only one thread is used to preprocess the boundary data and MPI communication is initiated on the thread, synchronization after boundary data preprocessing could be skipped. This method is tested on our implementation and the performance is not good because the boundary data preprocessing takes longer time. Therefore our implementation doesn't use this method.

3. Performance evaluation

Table 1 describes hardware and software configuration for performance evaluation. Each node has one Xeon Phi coprocessor card. Xeon Phi card and Infiniband network card are attached to the same CPU socket's PCIe slots. If Xeon Phi and Infiniband network card use different socket's PCIe slots, the multi card performance would be worse because of network latency increase.

PoS(LATTICE 2015)029

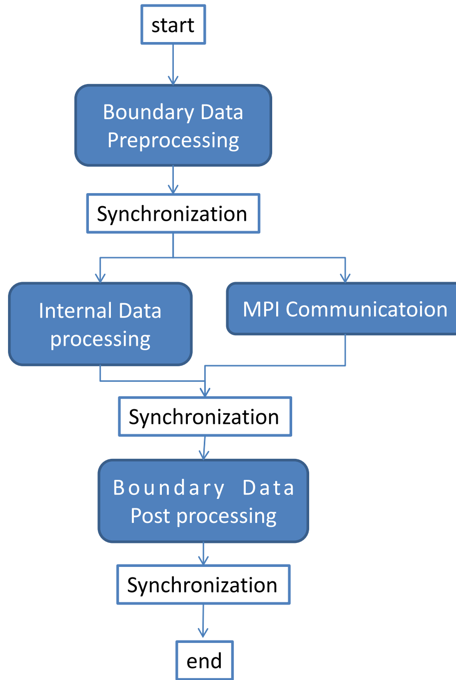

Figure 1: This figure describes one iteration of hopping term processing

Element	Configuration
Host	Xeon E5-2697 v3(Haswell) 2.6GHz 14core* 2sockets
Coprocessor	Xeon Phi 7120A(1.238Ghz, 61core)
HCA	Mellanox FDB IB
MPSS	Version 3.3.3
Compiler	Intel Compiler 15.0.2
MPI	Intel MPI 5.0.3

Table 1: Hardware and software configuration for performance evaluation

3.1 Hopping term performance on 1 card

In this section, we evaluate some key optimization features that affect performance on hopping term. Hopping term is fused with clover term computation in CG, but performances in this section addresses only hopping term computation which doesn't include clover term computation.

Table 2 shows hopping term performance with gauge compression and without gauge compression which means normal gauge. Hopping term performance depends on lattice size. Small lattice size performance cannot get peak performance, because the thread synchronization overhead and thread load imbalance have significant impact on performance. To get peak performance on one card, lattice size larger than 32x32x32x24 is required in this implementation. Gauge compression saves memory traffic, its effect on hopping term performance is about 13-14%.

Table 3 shows hopping term performance with software prefetch instructions and without soft-

PoS(LATTICE 2015)029

Lattice size	normal gauge	compressed gauge
32x32x32x12	79 GFLOPS (0.87)	86 GFLOPS (1.00)
32x32x32x24	85 GFLOPS (0.86)	94 GFLOPS (1.00)
32x32x32x32	88 GFLOPS (0.86)	93 GFLOPS (1.00)

Table 2: Comparison of hopping term performance between normal gauge and compressed gauge on 1 Xeon Phi

Lattice size	without prefetch	with prefetch
32x32x32x12	63 GFLOPS (0.87)	86 GFLOPS (1.00)
32x32x32x24	69 GFLOPS (0.86)	94 GFLOPS (1.00)
32x32x32x32	69 GFLOPS (0.86)	93 GFLOPS (1.00)

Table 3: Evaluation of software prefetch effect on hopping term with gauge compression on 1 Xeon Phi

Lattice size	without streaming store	with streaming store
32x32x32x12	79 GFLOPS (0.91)	86 GFLOPS (1.00)
32x32x32x24	85 GFLOPS (0.91)	94 GFLOPS (1.00)
32x32x32x32	85 GFLOPS (0.91)	93 GFLOPS (1.00)

Table 4: Evaluation of streaming store effect on hopping term with gauge compression on 1 Xeon Phi

Lattice size	normal gauge	compressed gauge
32x32x32x12	68 GFLOPS (0.91)	75 GFLOPS (1.00)
32x32x32x24	75 GFLOPS (0.90)	83 GFLOPS (1.00)
32x32x32x32	77 GFLOPS (0.93)	83 GFLOPS (1.00)

Table 5: CG Performance of both normal gauge and compressed gauge on 1 Xeon Phi

ware prefetch instructions. All other conditions except software prefetch is the same, therefore we can evaluate software prefetch effect on hopping term performance. Software prefetch effect on hopping term performance is about 13-14%.

Table 4 shows hopping term performance with streaming stores and without streaming store instructions. Streaming store effect on hopping term is about 9%.

3.2 CG performance on 1 card

Table 5 shows CG performance with compressed gauge and without compressed gauge. Compressed gauge field saves memory traffic only in hopping term, therefore its effect on CG performance is 9% and less than hopping term.

3.3 Multi card performance

Figure 2 shows multi-card performance of hopping term of several lattice sizes. To achieve peak performance on one card, lattice size larger than 32x32x32x24 is required. Multi card run

PoS(LATTICE 2015)029

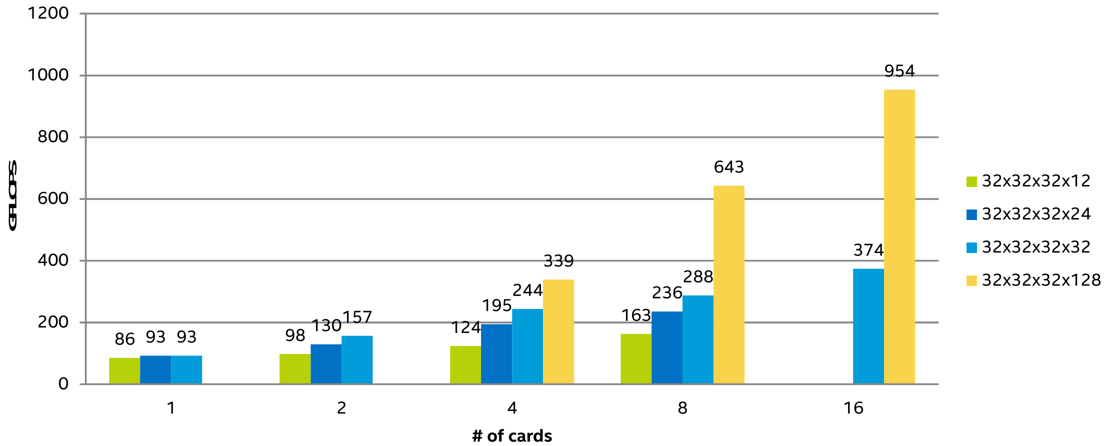

Figure 2: This figure describes hopping term performance with compressed gauge using multi-nodes

needs more larger lattice size to achieve good scalability. To achieve good scalability up to 16 cards, lattice size larger than 32x32x32x128 is required. Smaller lattice size limits the multi card scalability. For example, lattice size 32x32x32x24 can achieve 93 GFLOPS on one card, but it can only achieve 236 GFLOPS using 8 cards which is 2.5x of 1 card performance. There is two reasons for worse scalability of relatively smaller lattice size.

1. MPI communication bandwidth is better with large MPI message size. Small lattice size uses small MPI message size and it cannot use network bandwidth effectively.

2. Internal part processing time of small lattice size is shorter than MPI communication time. Therefor it cannot hide MPI communication latency.

Figure 3 shows multi-card performance of CG of several lattice sizes. Small lattice size scalability is still worse as same as hopping term. For example 32x32x32x24 CG performance on 1 card is 83 GFLOPS and the performance of 8 cards is 271 GFLOPS which is 3.2x of one card performance. Its scalability is better than hopping term because linear algebra and clover term need less communication bandwidth compared to hopping term, and thus their scalability is better than hopping term.

Figure 4 shows multi card performance of multi-shift CG. This graph shows nshift=10 which means 10 equation are solved simultaneously. Multishift CG scales in small lattice size because the linear algebra weights is bigger than normal CG.

4. Conclusion

One card performance of our implementation can achieve the peak of 94 GFLOPS for hopping term and 83 GFLOPS for CG lattice size larger than 32x32x32x24. Our implementation scales up to 16 Xeon Phi cards for 32x32x32x128 lattice size. And scalability depends on lattice size because effective network bandwidth for small data size is not good as large data size. SW prefetching, streaming stores and compressed gauge data is a key element to increase performance.

PoS(LATTICE 2015)029

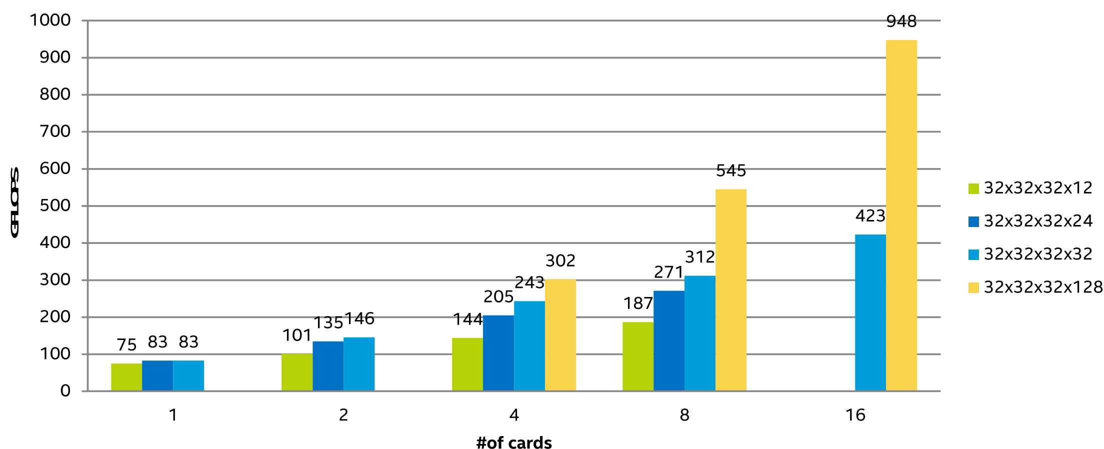

Figure 3: This figure describes CG performance using multi-nodes

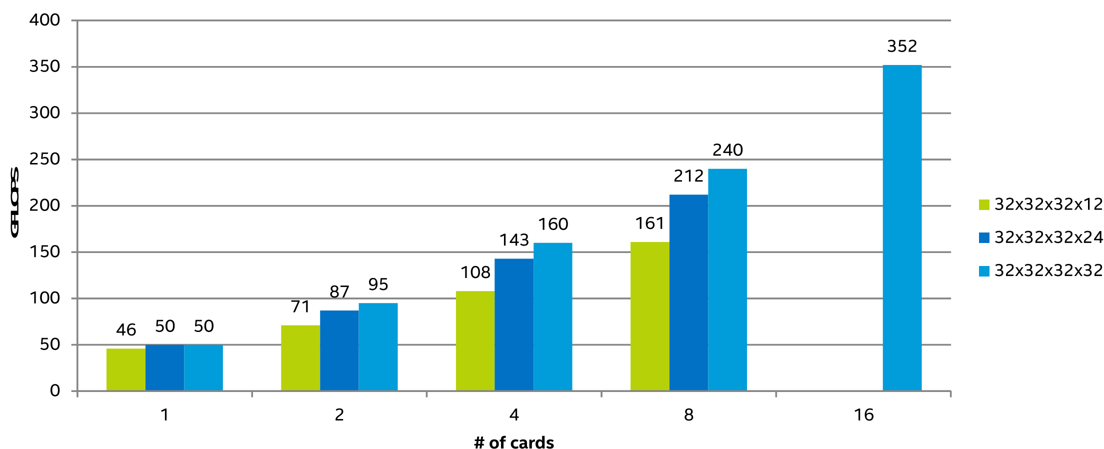

Figure 4: This figure describes Multishift-CG performance(nshift=10) using multi-nodes

Acknowledgments

This work was supported by JSPS KAKENHI Grant Number 25871116.

References

[1] B. Joó, D. Kalamkar, K. Vaidyanathan, M. Smelyanskiy, K. Pamnany, V. Lee, P. Dubey, and W. Watson, *Lattice QCD on Intel® Xeon Phi™ Coprocessors*, in *Supercomputing*, vol. 7905 of *Lecture Notes in Computer Science*, pp. 40–54. Springer Berlin Heidelberg, 2013.

[2] R. Li and S. Gottlieb, *Staggered Dslash Performance on Intel Xeon Phi Architecture*, in *Proceedings of 32nd International Symposium on Lattice Field Theory,* PoS(LATTICE2014)034.

[3] Intel, *Intel® Xeon Phi™ Coprocessor Instruction Set Architecture Reference Manual.*

PoS(LATTICE 2015)029