PROCEEDINGS OF SCIENCE

Multi-block/multi-core SSOR preconditioner for the QCD quark solver for K computer

T. Boku[a], **K.-I. Ishikawa**[*b,c], **Y. Kuramashi**[a,c,f], **K. Minami**[c], **Y. Nakamura**[c], **F. Shoji**[c], **D. Takahashi**[a], **M. Terai**[c], **A. Ukawa**[a], **T. Yoshié**[a,f]

[a] *Center for Computational Sciences, University of Tsukuba, Tsukuba, Ibaraki 305-8577, Japan*
[b] *Graduate School of Science, Hiroshima University, Higashi-Hiroshima, Hiroshima 739-8526, Japan*
[c] *RIKEN Advanced Institute for Computational Science, Kobe, Hyogo 650-0047, Japan*
[f] *Faculty of Pure and Applied Sciences, University of Tsukuba, Tsukuba, Ibaraki 305-8571, Japan*
E-mail: ishikawa@theo.phys.sci.hiroshima-u.ac.jp

We study the algorithmic optimization and performance tuning of the Lattice QCD clover-fermion solver for the K computer. We implement the Lüscher's SAP preconditioner with sub-blocking in which the lattice block in a node is further divided to several sub-blocks to extract enough parallelism for the 8-core CPU SPARC64$^{\rm TM}$ VIIIfx of the K computer. To achieve a better convergence property we use the symmetric successive over-relaxation (SSOR) iteration with *locally-lexicographical* ordering for the sub-blocks in obtaining the block inverse. The SAP preconditioner is included in the single precision BiCGStab solver of the nested BiCGStab solver. The single precision part of the computational kernel is solely written with the SIMD oriented intrinsics to achieve the best performance of the SPARC64$^{\rm TM}$ VIIIfx on the K computer. We benchmark the single precision BiCGStab solver on the three lattice sizes: $12^3 \times 24$, $24^3 \times 48$ and $48^3 \times 96$, with fixing the local lattice size in a node at $6^3 \times 12$. We observe an ideal weak-scaling performance from 16 nodes to 4096 nodes. The performance of a computational kernel exceeds 50% efficiency, and the single precision BiCGstab has $\sim$26% susutained efficiency.

The 30th International Symposium on Lattice Field Theory
June 24 – 29, 2012
Cairns, Australia

*Speaker.

PoS(Lattice 2012)188

1. Lattice QCD on the K computer

The K computer has been developed by RIKEN and Fujitsu since 2006 as the national leadership computer of Japan to promote science and technology [1]. The construction has been finished on July 2012 and the system is now provided to the strategic programs of the High Performance Computing Infrastructure (HPCI) and to the research users those who apply the computer resource through the HPCI [2]. The strategic programs consist of five fields: (i) Predictable life science, healthcare and drug discovery foundation, (ii) New Materials and Energy Creation, (iii) Projection of Planet Earth Variations for Mitigating Natural Disasters, (iv) Next-generation manufacturing technology, and (v) The origin of matter and the universe. Lattice QCD is contained in the fifth strategic program [3].

The K computer consists of over 80,000 computational nodes connected by the so called "Tofu" network. The Tofu network topology is six dimensional topology with 3D-mesh times 3D-torus shape, with which robustness against the node failure and high node flexibility and availability are ensured. Each node has a single CPU chip called "SPARC64$^{\mathrm{TM}}$ VIIIfx ". Equipping 8 cores with SIMD enabled 256 registers and 6MB shared L2 cache, the CPU can achieve a high efficiency for scientific applications (for the detailed system structure see [4]). Lattice QCD is one of the suitable applications for this kind of massively parallel system architecture.

In this paper we present the algorithmic and performance tuning of the $O(a)$ improved Wilson quark solver for the K computer. We focus on the solver algorithm preconditioned by the domain-decomposed Schwartz alternating procedure (SAP) with mixed precision [5]. In the next section we briefly introduce the SAP preconditioner and the nested BiCGStab algorithm [6]. The optimization of the algorithm and tuning methods for the computational kernels of the SAP are presented in section 3. We give the performance benchmark results in section 4 and summarize the paper in the last section.

2. Nested BiCGStab with Lüscher's SAP preconditioner

Our target problem is solving the following linear equation:

$$Dx = b, \tag{2.1}$$

where D is the clover term preconditioned $O(a)$-improved Wilson-Dirac operator:

$$D^{a,b}_{\alpha,\beta}(n,m) = \delta^{a,b}\delta_{\alpha,\beta}\delta(n,m) - \kappa F^{a,c}_{\alpha,\gamma}(n)\sum_{\mu=1}^{4}\Big[(1-\gamma_\mu)_{\gamma,\beta}(U_\mu(n))^{c,b}\delta(n+\hat{\mu},m) + (1+\gamma_\mu)_{\gamma,\beta}((U_\mu(m))^{b,c})^*\delta(n-\hat{\mu},m)\Big], \tag{2.2}$$

where $F(n)$ is the inverse clover term $(1-(c_{\mathrm{SW}}\kappa/2)\sigma_{\mu\nu}F_{\mu\nu}(n))^{-1}$, (n,m) are lattice site, a,b color, α,β are spin indexes.

Lüscher has introduced the Schwartz alternating procedure (SAP) to further preconditioning the Eq. (2.1) [5]. By dividing the whole lattice into two colored blocks in checkerboard manner, Eq. (2.1) can be rewritten in the following 2×2 block form:

$$\begin{pmatrix} D_{EE} & D_{EO} \\ D_{OE} & D_{OO} \end{pmatrix}\begin{pmatrix} x_E \\ x_O \end{pmatrix} = \begin{pmatrix} b_E \\ b_O \end{pmatrix}, \tag{2.3}$$

where D_{EE} (D_{OO}) is a block restricted operator in even-domain (odd-domain), while D_{EO} (D_{OE}) contains hopping operations from odd-domain to even-domain (and vice versa). The SAP practitioner M_{SAP} is introduced as

$$M_{\mathrm{SAP}} = K \sum_{j=0}^{N_{\mathrm{SAP}}-1} (1-DK)^j, \qquad \text{with} \qquad K = \begin{pmatrix} A_{EE} & 0 \\ -A_{OO}D_{OE}A_{EO} & A_{OO} \end{pmatrix}, \tag{2.4}$$

where A_{EE} (A_{OO}) can be any approximation for $(D_{EE})^{-1}$ ($(D_{OO})^{-1}$). When A_{EE} and A_{OO} are exact, DK becomes block triangular and is expected to be well preconditioned. In the case $|DK| < 1$, M_{SAP} converges to D^{-1} when $N_{\mathrm{SAP}} \to \infty$.

The nested BiCGStab solver is designed to be flexible against the preconditioner changing iteration by iteration [6] using an inner-outer strategy. The outer BiCGStab contains the inner BiCGStab solver as the flexible preconditioner. The flexibility ensures the double precision accuracy of the solution vector even if we use the single precision for the inner BiCGStab solver [7]. We apply the SAP preconditioner M_{SAP} to the inner single precision BiCGStab solver. The use of single precision has a merit as it requires less system resources than double precision. We target the single precision part of the solver as the tuning part for the K computer. In the following section we focus on the tuning of the single precision DK of M_{SAP}.

3. Performance tuning for the K computer

Inverse of block operator and OpenMP threading The approximate inverse of the block operator A_{EE} (A_{OO}) are the important part of the SAP. The exactness is not required for the SAP, however, the better approximation with less computational cost is preferred for A_{EE} (A_{OO}). The even-odd site preconditioning has been used in [5] and the SSOR preconditioning has been used in [8]. The latter has a better performance than the former at the same computational cost. The SSOR preconditioning is derived by decomposing the original operator into the sum of an upper and a lower triangular matrices. The preconditioning is achieved by solving the upper and lower triangular parts through forward and backward substitutions. The decomposition and the efficiency of the SSOR depend on the site ordering. It is observed that the SSOR with natural ordering have a better performance [8]. However the SSOR with natural ordering is not suitable for the multi-core CPU architecture because the natural ordering has a global data recurrence pattern and less parallelism in the forward and backward substitutions. To extract 8 core parallelism for the K computer, we further divide the block into 16 sub-blocks via the *locally-lexicographical* ordering (*ll*-ordering) described in [9].

Figure 1 shows an example of 6^4 lattice in a node. The block in a node is divided into 2^4 sub-blocks. Each single core contains two sub-blocks (two sub-blocks with size 3^4 adjacent in temporal direction). The numbering on the sites is the ordering according to the *ll*-ordering. The arrows on the links represent the data recurrence direction. Most of the recurrences are limited to each sub-block and there are little data reference on the surface of sub-blocks. Based on this ordering we wrote the forward and backward solver to construct A_{EE} (A_{OO}) with the SSOR. The parallelization in a node is achieved by explicit OpenMP threading. 8 threads are invoked and two sub-blocks are assigned to each OpenMP thread. The spatial division is mapped to OpenMP 8 threads, and the

temporal division is dedicated to the loop unrolling in a single thread. To resolve the recurrence dependency among threads, we carefully insert explicit **omp barrier**'s at the sites that require data on other threads.

The sub-block size affects the performance of the SAP. It has been reported that the size of 2^4 for block still has a gain against the even-odd site ordering on a larger size lattice [9]. Although we apply the *ll*-ordering to a small lattice (corresponds to a block in a node), we observed that the use of the SSOR with sub-blocking via *ll*-ordering for A_{EE} still has a gain against the even-odd site ordering.

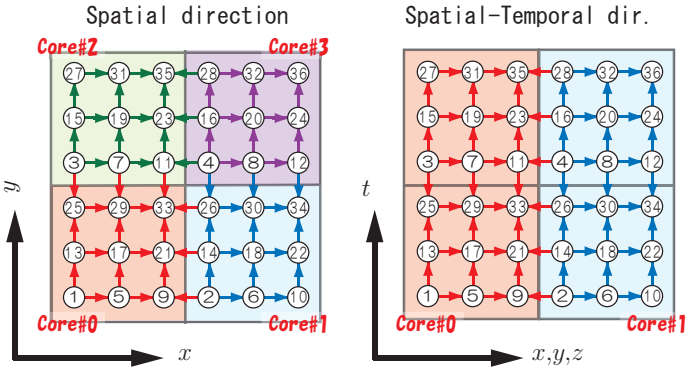

Figure 1: Site ordering in a CPU. This is an example with a 6^4 block in a node. The numbering of sites is limited in the 2-D plane for simplicity.

SIMDzation The SPARC64$^{\text{TM}}$ VIIIfx CPU has 256 double precision (64bit) FP registers per core. Each core can issue two independent fused multiply add (FMAD) on paired registers (SIMD) in a cycle resulting in 8 floating operations per cycle. To fully make use of these core infrastructure we have to extract enough independent data-computation stream in the kernel code D_{EE} and A_{EE}.

The SIMD FMAD operation can be explicitly invoked using intrinsic functions defined in the C/C++ language provided by Fujitsu. Although our entire program codes were developed in Fortran90 at first, we totally write the single precision inner BiCGStab solver using the SIMD intrinsic functions in the C language. The details of the intrinsics are not available publicly, while these have one-to-one correspondence to the mnemonic of the SPARC64$^{\text{TM}}$ VIIIfx [10]. The linkage between the Fortran90 and the C codes is realised through the `ISO_C_BINDING` feature of Fortran 2003 which is partly included in the Fujitsu Fortran. We use double precision intrinsic functions for the SIMD computation because the single precision FLOPS performance is identical to that in double precision in this CPU architecture. The data are converted from single precision to double precision (and vice verse) at the data loading (storing) from (to) memory. We employ the $SU(3)$ reconstruction method in the kernel code where the link variables are stored in compressed form (keeps only two-columns of the $SU(3)$ matrix) and the third column is reconstructed using the unitarity condition on the fly.

To further improve the kernel performance we unroll the temporal direction loop, which is the innermost site loop, by three times for D_{EE}. For the forward/backward solver in A_{EE} the two independent sub-blocking in temporal direction is used as unrolling. This is based on the technique of the register blocking which hide the data load latency. The unrolling also contains branch elimination by condition merging. This loop unrolling with huge loop body is possible thanks to the many registers of the SPARC64$^{\text{TM}}$ VIIIfx .

$SU(3)$ reconst.	Flop count	Load/Store count (real)	Real/Flop
Yes	2232	372	0.1667
No	1896	420	0.2215

Table 1: Flop count and load/store count for the clover hopping matrix per site.

In table 1 we summarize the total flop and load/store count per site for the hopping matrix of D.

These numbers are estimated for bulk sites in a block (not for the surface sites). We use the Dirac representation for γ_μ (γ_4 is diagonal) in the spin projection, and use the chiral representation for the inverse clover term $F(n)$. When we multiply $F(n)$, it is converted to the Dirac representation on the fly. The required byte/flop is 0.667 with the $SU(3)$ reconstruction method in single precision. The theoretical system byte/flop is 64 [GByte/s]/128 [GFlops] = 0.5. Thus our computation is still limited by the memory bandwidth. The maximum theoretical performance is estimated to be ~ 96 GFlops from the system memory bandwidth of 64 GByte/s. Excluding the redundant flop count caused by the $SU(3)$ reconstruction, the effective performance is ~ 82 GFlops, which is still better than that without the $SU(3)$ reconstruction (~ 72 GFlops). This performance number could be reduced or enhanced by the cache system, site loop control, conditional branch for site location, thread controlling etc. We test and benchmark the D_{EE} and A_{EE} kernels.

Communication hiding The internode communication in the SAP kernel DK is arranged in the D_{EO} and D_{OE} kernels. The structure of D_{EO} (D_{OE}) is basically organized as follows: (i) spin projection, link $U^\dagger_\mu(m)$ multiplication and data packing, (ii) sending data, (iii) receiving data, (iv) link $U_\mu(n)$ multiplication, spin reconstruction and accumulation. Other computation is possible during the step (ii) and (iii).

Algorithm 1 $w = DKv$ computation with communication hiding. f_E and f_O are working vectors.

1: $x_E = A_{EE} v_E$
2: **MPI_Isend** and **MPI_Irecv** for $w_O = D_{OE} x_E$.
3: $w_E = D_{EE} x_E$
4: **MPI_Wait** for $w_O = D_{OE} x_E$.
5: $f_O = v_O - w_O$
6: $x_O = A_{OO} f_O$
7: **MPI_Isend** and **MPI_Irecv** for $f_E = D_{EO} x_O$.
8: $f_O = D_{OO} x_O$
9: $w_O = w_O + f_O$
10: **MPI_Wait** for $f_E = D_{EO} x_O$.
11: $w_E = w_E + f_E$

We organize the SAP kernel DK to hide the communication time of D_{EO} (D_{OE}) behind the computation of D_{EE} (D_{OO}) as shown in Alg. 1. To hide the communication we employ the non-blocking MPIs: **MPI_Isend**, **MPI_Irecv**, and **MPI_Wait**. The steps (i)-(ii) are done at the lines 2 and 7, and the steps (iii)-(iv) are at the lines 4 and 10. We benchmark the communication performance in the weak-scaling test by comparing the performance of D_{EE} and DM_{SAP}, where the latter contains the internode communication while the former does not.

4. Results

We benchmark the single precision BiCGStab with the SAP preconditioner on the K computer. The lattice sizes benchmarked are $12^3 \times 24$, $24^3 \times 48$, and $48^3 \times 96$. The block size of the SAP is kept fixed at 6^4 and the local lattice size in a node is $6^3 \times 12$. The sub-block size for the SSOR is thus 3^4. The number of nodes used for the benchmark is 16, 256 and 4096 nodes. The performance is measured using the profiler provided by the Fujitsu's compiler system.

Figure 2 shows the SIMD rate in the instruction executed in benchmarking runs. We achieve 90% SIMD rate for the kernel D_{EE} and A_{EE} by explicitly using the SIMD intrinsic functions. At the solver level it reduces to 85% as it contains internode communication. Figs 3 and 4 represents the weak-scaling property of the flops performance. The efficiency (Fig. 3) is almost at constant for the kernels resulting an ideal weak-scaling (Fig. 4) from 16 nodes to 4096 nodes.

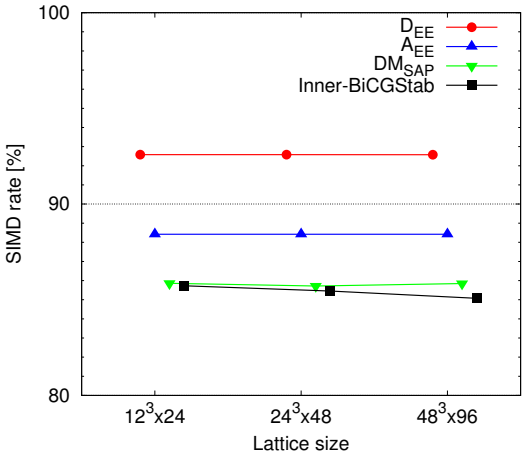

Figure 2: Weak-scaling of the SIMD rate of the single precision kernels in a node.

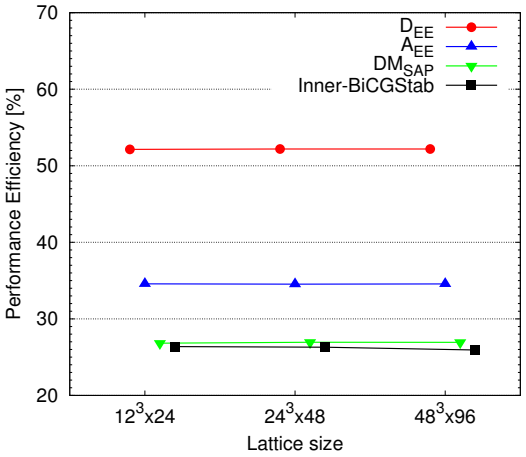

Figure 3: Same as Fig. 2, but for the performance efficiency in a node.

The efficiency reaches over 50% for the D_{EE} kernel, while the A_{EE} kernel has lower 35%.

There are two reasons for the lower performance for A_{EE}. One is the load imbalance of the computation among the threads in A_{EE}. For example, the number of red arrows is more than that of purple in the left panel of figure 1. This means that the core #0 has much computation than that of the core #3. The second is the less computational density in the loop body compared to that of D_{EE}. As mentioned in previous section the loop of D_{EE} is unrolled by three times, while A_{EE} is by two with sub-blocking. The forward/backward substitution of A_{EE} only has one sided hopping operation for bulk sites. This also reduces the computational density in the loop body. The theoretical peak efficiency for D_{EE} is estimated be 75%(=96 [GFlops]/128 [GFlops]), while the observed 50% efficiency does not reach the theoretical peak efficiency. A reason for this could be the conditional branch to distinguish the sites in the bulk or on the surface of the block. We still need a further investigation on the gap between the theoretical peak efficiency and the observed efficiency.

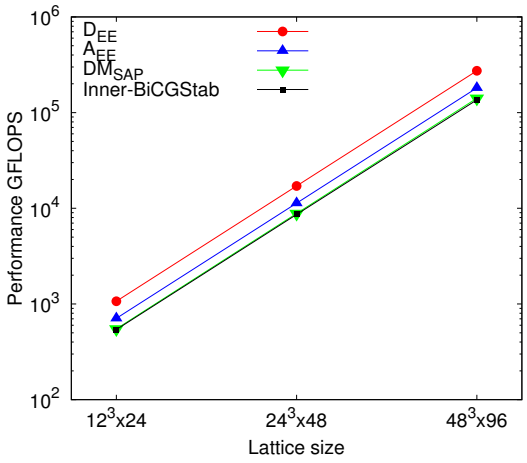

Figure 4: Weak-scaling of the total flops performance.

The performance efficiency of the single precision inner solver is at ∼26% in all. We observe an ideal weak-scaling property as shown in Fig. 4. Note that the measured performance presented in the figures contains redundant floating operations coming from the $SU(3)$ reconstruction and the use of FMAD for the spin projection. We have to roughly multiply 0.8 on the Flops numbers in Fig. 4 to exclude the redundant operations and to obtain the effective performance.

5. Summary

We have benchmarked the single-precision BiCGStab solver for the $O(a)$-improved Wilson fermion on the K computer. The solver code has been successfully optimized and tuned for the

system. We have applied the SSOR with the *ll*-ordered sub-blocking for the approximate block inverse of the SAP preconditioner to enhance the parallelism for the multi-core architecture. Thanks to the science specific architecture of th K computer we could achieve the ideal weak-scaling performance and $\sim 26\%$ efficiency for the single-precision BiCGStab solver. It partly remains unclear the origin of the gap between the theoretical system performance and the measured performance. We expect a further improvement on the internode communication by using the "Tofu" specific optimization which is not covered in this study.

Acknowledgement This work has been done in the "collaborative research for the performance analysis of large scale simulations on next-generation supercomputers" between RIKEN and University of Tsukuba. We thank the members of the next-generation Technical Computing Unit of Fujitsu for giving us the technical advice and support, and for tuning the computational kernels. Part of the results is obtained by using the K computer at the RIKEN Advanced Institute for Computational Science (Proposal numbers hp120108, hp120153, hp120170, hp120281), T2K-Tsukuba System in Center for Computational Sciences, University of Tsukuba, and a PC-cluster of HPCI strategic program Field 5. This work is supported in part by Grants-in-Aid for Scientific Research from the Ministry of Education, Culture, Sports, Science and Technology (Nos. 22244018, 24540276).

References

[1] RIKEN Advanced Institute for Computational Science (AICS), [http://www.aics.riken.jp/en/].

[2] High Performance Computing Infrastructure (HPCI), [https://www.hpci-office.jp/index.html].

[3] HPCI Strategic Program Field 5, *The origin of matter and the universe*, [http://www.jicfus.jp/field5/en/].

[4] FUJITSU SCIENTIFIC & TECHNICAL JOURNAL (FSTJ), *The K computer*, 2012-7 (Vol.48, No.3) [http://www.fujitsu.com/global/news/publications/periodicals/fstj/].

[5] M. Lüscher, Comput. Phys. Commun. **165** (2005) 199 [arXiv:hep-lat/0409106]; JHEP **0305** (2003) 052 [hep-lat/0304007].

[6] J.A. Vogel, Appl. Math. Comput, **167** (2005) 1004-1025; H. Tadano and T. Sakurai, LSSC'07, Lec. Notes Comput. Sci. **4818** (2008) 721.

[7] A. Buttari, J. Dongarra, J. Kurzak, P. Luszczek, and S. Tomov, *Using mixed precision for sparse matrix computations to enhance the performance while achieving 64-bit accuracy*, ACM Trans. Math. Soft., **34** (2008) 1.

[8] S. Aoki *et al.* [PACS-CS Collaboration], Phys. Rev. D **81** (2010) 074503 [arXiv:0911.2561 [hep-lat]]; Phys. Rev. D **79** (2009) 034503 [arXiv:0807.1661 [hep-lat]].

[9] N. Eicker, W. Bietenholz, A. Frommer, T. Lippert, B. Medeke and K. Schilling, Nucl. Phys. Proc. Suppl. **73** (1999) 850 [arXiv:hep-lat/9809038]; S. Fischer, A. Frommer, U. Glassner, T. Lippert, G. Ritzenhofer and K. Schilling, Comput. Phys. Commun. **98** (1996) 20 [arXiv:hep-lat/9602019].

[10] Fujitsu Limited, "SPARC64$^{\mathrm{TM}}$ VIIIfx Extensions" Version 15, 2010.