PROCEEDINGS OF SCIENCE

Optimizing the domain wall fermion Dirac operator using the R-Stream source-to-source compiler

Meifeng Lin[*a], **Eric Papenhausen**[b], **M. Harper Langston**[c], **Benoit Meister**[c], **Muthu Baskaran**[c], **Taku Izubuchi**[d,e] **and Chulwoo Jung**[d]

[a] *Computational Science Center, Brookhaven National Laboratory, Upton, NY 11973, USA*
[b] *Department of Computer Science, State University of New York, Stony Brook, NY 11794, USA*
[c] *Reservoir Labs Inc., New York, NY 10012, USA*
[d] *Physics Department, Brookhaven National Laboratory, Upton, NY 11973, USA*
[e] *RIKEN-BNL Research Center, Brookhaven National Laboratory, Upton, NY 11973, USA*
E-mail: mlin@bnl.gov, epapenhausen@cs.stonybrook.edu, langston@reservoir.com, meister@reservoir.com, baskaran@reservoir.com, izubuchi@bnl.gov, chulwoo@bnl.gov

The application of the Dirac operator on a spinor field, the Dslash operation, is the most computation-intensive part of the lattice QCD simulations. It is often the key kernel to optimize to achieve maximum performance on various platforms. Here we report on a project to optimize the domain wall fermion Dirac operator in Columbia Physics System (CPS) using the R-Stream source-to-source compiler. Our initial target platform is the Intel PC clusters. We discuss the optimization strategies involved before and after the automatic code generation with R-Stream and present some preliminary benchmark results.

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

[*] Speaker.

1. Introduction

Lattice QCD (LQCD) community has traditionally produced very efficient home-grown software and is continuing to do so. With the advent of new hardware architectures, significant efforts are required to optimize the LQCD software for new systems. One way to deal with this is to rewrite the software to be *future-proof*, which inevitably needs significant initial investments to develop a software suite that is portable, flexible, adaptable and, most importantly, performant for different architectures. Another way, perhaps complementary to the first approach, is to develop or use existing high-quality automatic *code generators* that are capable of producing efficient codes for a target architecture from generic, high-level, user codes. The latter will help to port an existing application to a new architecture quickly, and ideally with good performance.

Numerical LQCD computations are dominated by the application of the Dirac operator matrix D on a fermion field vector ψ, the so-called Dslash operation. It is often the key kernel to optimize for maximum performance, as the Dslash operation typically accounts for more than 90% of the execution time in LQCD simulations. In this work we explore the feasibility and efficiency of using the R-Stream source-to-source compiler being developed by Reservoir Labs Inc to optimize the Dslash code in the Columbia Physics System (CPS) [1]. This report is organized as following. After a brief introduction to R-Stream in Section 2, we present the details of the application of R-Stream to the Wilson Dslash and the Domain Wall Fermion (DWF) Dslash in Section 3. Section 4 discusses SSE and AVX SIMD vectorizations on the R-Stream generated code. In Section 5 we show the performance benchmarks for the single-node Wilson and DWF Dslash. And we conclude in Section 6.

2. The R-Stream Source-to-Source Compiler

R-Stream [2] is a source-to-source compiler that generates target-specific optimized source codes based on the polyhedral mapping for computer programs [3, 4]. It takes a serial C code as input, generates the dependence graph using the polyhedral model, and can perform optimizations ranging from loop-level parallelism, tiling to memory management. The outputs of R-Stream may be C code threaded with OpenMP to run on CPUs, or CUDA C for NVIDIA GPUs. These outputs can then be processed by a low-level compiler such as `icc`, `gcc` or `nvcc`. The advantage of using a high-level code generator such as R-Stream is that the source code generated can be tuned and further optimized before sending it to a low-level compiler, allowing user-in-the-loop optimizations. For more details of the types of optimizations R-Stream can perform and the process involved, readers can refer to [2, 5, 6].

3. Application of R-Stream to Dslash Operators in CPS

Our first application of R-Stream is the Wilson Dslash operator, which is the kernel operator in many lattice fermion formulations, including the domain wall fermion formulation used in many of the lattice simulations performed by the RBC and UKQCD collaborations. The Wilson quark Dirac matrix can be written as

$$M = (N_d + m_q) - \frac{1}{2}D, \tag{3.1}$$

where N_d is the space-time dimension and m_q is the input quark mass. The Wilson Dslash operator, D, in four space-time dimensions is defined as

$$D^{ij}_{\alpha\beta}(x,y) = \sum_{\mu=1}^{4}\left[(1-\gamma_\mu)_{\alpha\beta}{U_\mu}^{ij}(x)\delta_{x+\hat{\mu},y} + (1+\gamma_\mu)_{\alpha\beta}U^{\dagger ij}_\mu(x+\hat{\mu})\delta_{x-\hat{\mu},y}\right], \tag{3.2}$$

where x and y are the coordinates of the lattice sites, α,β are spin indices, and i, j are color indices. $U_\mu(x)$ is the gluon field variable and is an SU(3) matrix. In CPS, the *complex* fermion fields are represented by one-dimensional arrays with size $L_X L_Y L_Z L_T \times SPINS \times COLORS \times 2$ for the unpreconditioned Dirac operator, where L_X, L_Y, L_Z and L_T are the numbers of lattice sites in the x, y, z and t directions, respectively. *SPINS* and *COLORS* are the numbers of spin and color degrees of freedom, typically 4 and 3 respectively. With even-odd preconditioning, the array sizes are cut in half.

The domain wall fermion (DWF) Dirac matrix $M_{x,s;x',s'}$ is defined as

$$M_{x,s;x',s'} = \delta_{s,s'}M^{\parallel}_{x,x'} + \delta_{x,x'}M^{\perp}_{s,s'}, \tag{3.3}$$

where s, s' label the extra fifth dimension, and the 4D DWF Dslash is defined as

$$M^{\parallel}_{x,x'} = -\frac{1}{2}\sum_{\mu=1}^{4}\left[(1-\gamma_\mu)U_{x,\mu}\delta_{x+\hat{\mu},x'} + (1+\gamma_\mu)U^{\dagger}_{x',\mu}\delta_{x-\hat{\mu},x'}\right] + (4-M_5)\delta_{x,x'}. \tag{3.4}$$

Note that Eq.(3.4) is just the Wilson Dirac operator in Eq.(3.1) with a negative mass $m_q \equiv -M_5$ and is independent of s. The hopping term in the fifth dimension is defined as

$$\begin{aligned} M^{\perp}_{s,s'} &= -\frac{1}{2}\left[(1-\gamma_5)\delta_{s+1,s'} + (1+\gamma_5)\delta_{s-1,s'} - 2\delta_{s,s'}\right] \\ &+ \frac{m_f}{2}\left[(1-\gamma_5)\delta_{s,L_s-1}\delta_{0,s'} + (1+\gamma_5)\delta_{s,0}\delta_{L_s-1,s'}\right]. \end{aligned} \tag{3.5}$$

As the 4D DWF Dslash dominates DWF calculations, we first focused on its optimization.

The input we gave to R-Stream was the `noarch` version of the Wilson Dslash operator in CPS, which is an unoptimized serial C code. For the polyhedral mapping to work, the array accesses have to be *affine* functions of the outer loop indices so that the dependence can be constructed as a system of linear equations. The array access in CPS breaks affinity, as the array index for the lattice site (x,y,z,t) has to be calculated as products of the array indices. To overcome this, we cast the linear arrays as multi-dimensional pointers in C, which gives us the performance of linear arrays but gives R-Stream the readability of multi-dimensional arrays.

Another change we had to make to the input Dslash code was to remove the modulo operations when the lattice boundaries are involved. For example, in the x direction, the Dslash operation for the site $x = LX - 1$ requires the access to the fermion vector at site $x = 0$, which is done by `x % LX`. Similarly, access to $x = LX - 1$ is needed when Dslash operates on $x = 0$. To preserve affinity, we padded each boundary with the value at the opposite boundary. The new fermion arrays have a size $LX + 2$ in the x direction, with `psi[0]` containing the value of $x = LX - 1$, and `psi[LX+1]` containing the value of $x = 0$. Similarly for the other directions. For the gauge field, since only the neighbor in the forward direction is needed, we only need to introduce padding for one side of

the boundaries, resulting in a new array length of $L+1$ in each of the space-time directions. The downside of the data padding is that the memory footprint is increased substantially, especially when local lattice volume is small. However, when the memory increases the most (small local volume) is also when the total memory footprint is small, even though the percentage increase may be large. So having padding does not affect the problem size we are able to simulate greatly.

With the above two modifications, R-Stream was able to analyze the input code and generate outputs based on the user guidance. Details of how to use R-Stream and how the user can affect the optimization strategies are beyond the scope of these proceedings, and we refer the readers to Refs. [2, 6] for further information. The R-Stream generated code included automatic loop tiling and unrolling. While the performance of the R-Stream output was a bit better than the input unimproved serial code, it was much inferior to a version of a hand-optimized CPS code which has SSE intrinsics. To fully exploit the performance offered by the modern computer architectures, it is imperative for us to take advantage of their SIMD capabilities. We thus further optimized the code using both the Intel SSE and AVX intrinsics, which we will discuss in the next section.

4. SIMD Implementation

4.1 SSE

The Intel SSE instruction set extension allows to perform two double-precision or four single-precision floating point operations at each clock cycle. It requires the data in such operations to be vectorized. As discussed in Section 3, we use the multi-dimensional arrays for the fermion vectors. For the Wilson Dslash in 4D, it goes as

```
double psi[LT][LZ][LY][LX][4][3][2];
```

where the real/imaginary index runs the fastest. For double precision, this data structure is already vectorized in a way suitable for the SSE instruction. While using this vectorization does not require a complete data structure transformation, the cross term in the complex multiplication requires permutation of the complex vector, which may affect the performance. We used this implementation for the performance benchmark shown in Section 5, as our final goal is to implement the AVX instruction for the DWF Dslash operator.

4.2 AVX

The AVX instruction set extension can perform four double-precision or eight single-precision floating point operations at one clock cycle, provided that the data are vectorized accordingly. A data structure transformation is required to implement the AVX vectorization. There are various ways to do this. For the Wilson Dslash, we chose to vectorize using the four spinor degrees of freedom. The new data layout becomes

```
double psi[LT][LZ][LY][LX][3][2][4];
```

For DWF, we tried different data layouts, including one similar to the Wilson Dslash. But the best performance was obtained when the data layout vectorized along the fifth dimension:

```
double psi[LT][LZ][LY][LX][3][2][4][LS];
```

as the 4D DWF Dslash in Eq.(3.3) is inherently data parallel in the fifth dimension. The disadvantage of this data layout is, it requires `LS` to be multiples of 4 (8) when double (single) precision is used in the computation. A more flexible data structure is discussed in [7], which we will look into in the future.

5. Performance

5.1 Compile Environment and Test Platforms

For the performance benchmarks reported in this section, double precision arithmetics were used. OpenMP was used for threading, while SSE or AVX intrinsics were used for SIMD. In the tests shown below, we used the GNU C compiler `gcc` version 5.1.0. Similar results were obtained with `gcc` version 4.9.2. We ran the tests on three clusters: (i) The `pi0` cluster at Fermilab with dual-socket Intel "Sandy Bridge" Xeon CPU E5-2650 v2 at 2.60GHz, referred to as `SNB-pi0`. (ii) The `hpc1` cluster at Brookhaven National Lab with dual-socket Intel "Sandy Bridge" Xeon CPU E5-2670 at 2.60 GHz, referred to as `SNB-hpc1`. (iii) The `lired` cluster at Stony Brook University with dual-socket Intel "Haswell" Xeon CPU E5-2690 v3 at 2.60 GHz, referred to as `HSW-lired`. Both `SNB-pi0` and `SNB-hpc1` support AVX extension, while `HSW-lired` supports AVX2.

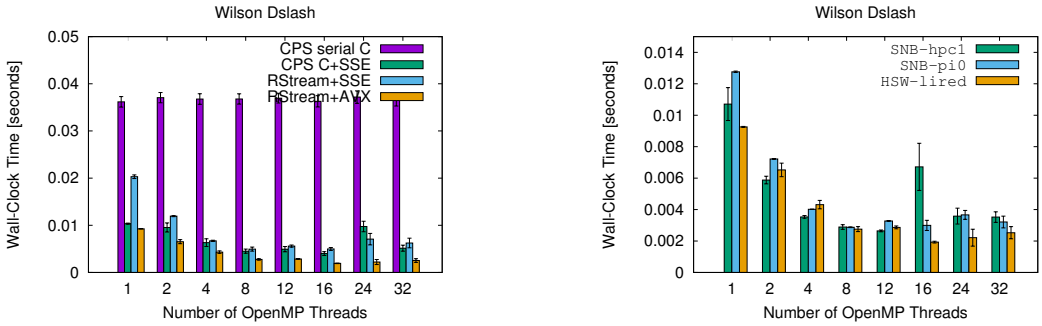
5.2 Single-Node Wilson Dslash

Figure 1(a) shows the comparison of wall-clock time for the even-odd preconditioned Wilson Dslash in several implementations on a 16^4 lattice using the `HSW-lired` cluster. "CPS serial C" refers to the `noarch` implementation in CPS. "CPS C+SSE" refers to the original hand-optimized version with SSE intrinsics. "RStream+SSE" refers to the R-Stream generated code with SSE intrinsics and "RStream+AVX" refers to the version with AVX intrinsics. The best performance was achieved with 16 threads for the parallelized versions, with the AVX implementation performing twice as well as the CPS SSE version, at about 22 GFlops per node. While the one-thread performance with AVX was quite good, at 4.6 GFlops, the performance suffered from poor scaling of our OpenMP implementation, which we expect to improve with further investigation.

We also show the performance for the RStream+AVX implementation on three different machines in Figure 1(b). The performances were quite similar. While the Haswell processors support fused multiply-add (FMA) that are available in AVX2, at the time of our implementation we did not have access to this type of CPUs and thus did not include AVX2 instructions. It is not surprising that our code performs similarly on these three machines. It is also worth noting that the total number of cores on one node is 16 for the Sandy Bridge clusters and 24 for the Haswell cluster. When the number of OpenMP threads is equal to the number of cores available, performance drops slightly, possibly due to the way the threads are scheduled.

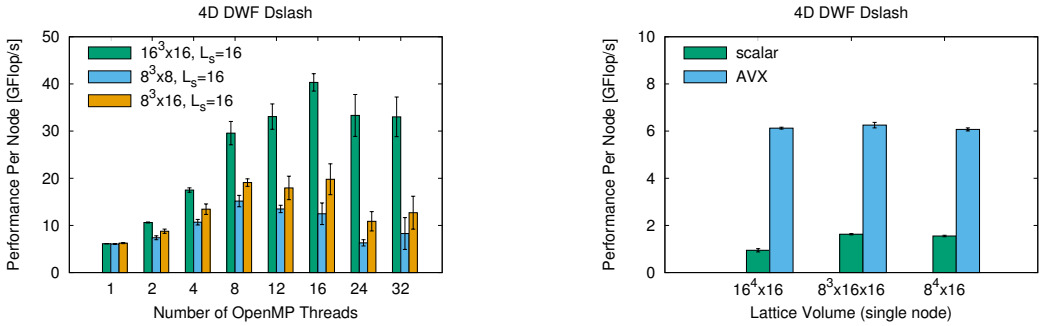
5.3 Single-Node 4D DWF Dslash

For DWF, the dominating computation is the 4D DWF Dslash as given in Eq.(3.4). We show the performance of the 4D DWF Dslash with "RStream+AVX" implementation in Figure 2. The tests were run on the `HSW-lired` cluster. Figure 2(a) shows the performance on one node with different lattice volumes. Since we are interested in scaling up to a large number of nodes, we



(a) (b)

Figure 1: Single-node performance for the Wilson Dslash on a 16^4 lattice. Figure 1(a) shows the comparison of wall-clock time for four implementations of the even-odd preconditioned Wilson Dslash on `HSW-lired`. Figure 1(b) shows the comparison of wall-clock time for RStream+AVX implementation on three different machines. Timing is averaged over 10 runs, and the error bars show its variance.



(a) (b)

Figure 2: Single-node performance for the RStream+AVX implementation of the 4D DWF Dslash on `HSW-lired`. Figure 2(a) shows the performance with different OpenMP threads. Figure 1(b) shows the effect of AVX vectorization on different lattice volumes with one OpenMP thread. Timing is averaged over 10 runs, and the error bars show its variance.

also tested the small node-volumes of 8^4 and $8^3 \times 16$. The fifth dimension length L_s was fixed to 16. The best performance was achieved with the node-volume of 16^4 and with 16 OpenMP threads at about 40 GFlops. We also show the effect of vectorization in Figure 2(b), where we compare the performance of the scalar CPS `noarch` version with the R-Stream generated code further vectorized with AVX. As expected, at least a factor of 4 speedup was achieved.

6. Summary and Conclusions

We have presented our experience with using the R-Stream source-to-source, polyhedral-model-based, compiler to optimize the Dslash operator in CPS for both the Wilson and DWF fermions. With some modifications to the input sequential C code, we were able to use R-Stream to

analyze and generate C code with loop reorganizations that made it easier for us to parallelize with OpenMP for the single-node execution. With AVX SIMD instructions, we were able to achieve more than a factor of 16 speedup on a single node compared to the input serial code for the Wilson Dslash and more than a factor of 40 for the 4D DWF Dslash . The performance is twice that of the manually optimized CPS code with SSE intrinsics, suggesting that such high-level code generators have the potential to give performance on par with hand-written codes. More tests with different lattice volumes and different compile environments are ongoing. We are also working on a single-precision version that should offer even greater performance. Multi-node version with MPI communications is in progress. The performance for the full version with conjugate gradient, MPI communications and single precision arithmetics will be presented in a future publication.

Acknowledgments

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Nuclear Physics under Award Number DE-SC0009678. M.L., T.I. and C.J. are supported in part by the U.S. Department of Energy, Office of Science under Contract Number DE-SC0012704 through which Brookhaven National Laboratory is operated. T.I. is also supported by Grants-in-Aid for Scientific Research #26400261. We gratefully acknowledge the use of computing resources on the USQCD cluster at Fermilab, HPC Code Center cluster at Brookhaven National Laboratory and the LI-red cluster at Stony Brook University for testing and benchmarking.

References

[1] `http://qcdoc.phys.columbia.edu/cps.html`.

[2] B. Meister, N. Vasilache, D. Wohlford, M. M. Baskaran, A. Leung and R. Lethin, *R-stream compiler*, in *Encyclopedia of Parallel Computing*, pp. 1756–1765. 2011.

[3] P. Feautrier, *Some efficient solutions to the affine scheduling problem. Part I. One-dimensional time*, *International Journal of Parallel Programming* **21** (1992), no. 5 313–348.

[4] A. Darte, R. Schreiber and G. Villard, *Lattice-based memory allocation*, *IEEE Trans. Comput.* **54** (2005), no. 10 1242–1257.

[5] N. Vasilache, M. M. Baskaran, B. Meister and R. Lethin, *Memory reuse optimizations in the r-stream compiler*, in *Proceedings of the 6th Annual Workshop on General Purpose Processing with Graphics Processing Units (GPGPU), Houston, TX, USA, January*, 2013.

[6] E. Papenhausen, B. Wang, M. H. Langston, M. Baskaran, T. Henretty, T. Izubuchi, A. Johnson, C. Jung, M. Lin, B. Meister, K. Mueller and R. Lethin, *Polyhedral user mapping and assistant visualizer tool for the r-stream auto-parallelizing compiler*, in *Proc. VISSOFT*, pp. 180–184, IEEE, 2015.

[7] P. Boyle, G. Cossu, A. Yamaguchi and A. Portelli, *Grid: A next generation data parallel c++ qcd library*, *PoS* **LATTICE2015** (2015) 23.