PROCEEDINGS OF SCIENCE

AVX-512 extension to OpenQCD 1.6

Jarno Rantaharju*
Swansea Academy of Advanced Computing, Swansea University, SA1 8EP, UK
E-mail: j.m.o.rantaharju@swansea.ac.uk

Ed Bennett
Swansea Academy of Advanced Computing, Swansea University, SA1 8EP, UK
E-mail: e.j.bennett@swansea.ac.uk

Mark Dawson
Swansea Academy of Advanced Computing, Swansea University, SA1 8EP, UK
E-mail: mark.dawson@swansea.ac.uk

Michele Mesiti
Swansea Academy of Advanced Computing, Swansea University, SA1 8EP, UK
E-mail: michele.mesiti@swansea.ac.uk

We publish an extension of openQCD-1.6 with AVX-512 vector instructions using Intel intrinsics. Recent Intel processors support extended instruction sets with operations on 512-bit wide vectors, increasing both the capacity for floating point operations and register memory. Optimal use of the new capabilities requires reorganising data and floating point operations into these wider vector units. We report on the implementation and performance of the AVX-512 OpenQCD extension on clusters using Intel Knights Landing and Xeon Scalable (Skylake) CPUs. In complete HMC trajectories with physically relevant parameters we observe a performance increase of 5% to 10%.

The 36th Annual International Symposium on Lattice Field Theory - LATTICE2018
22-28 July, 2018
Michigan State University, East Lansing, Michigan, USA.

*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(LATTICE2018)039

1. Introduction

The newest generations of Intel processors, Xeon Scalable (Skylake) and Xeon Phi (Knights Landing), extend the current standard of AVX2 vector instructions with the 512-bit wide AVX-512 instruction sets [1]. The width of registers is similarly increased and, in addition, the number of floating point registers is doubled to 32. Compilers can make use of the new instructions, but targeted code is required to reach optimal performance. Efficient vectorisation can lead to a doubling of the available register memory and floating point capability

OpenQCD-1.6 [2] already includes optional AVX2 and SSE targeted implementations and an extension targeting BlueGene/Q also exists [3]. Following the logic of these extensions we have reimplemented several performance critical functions using Intel intrinsic instructions for AVX-512 vectors. These are mainly the Dirac operator and the vector operations necessary for the conjugate gradient algorithm.

Here we publish the extension to openQCD-1.6 [4]. In addition, we report scaling studies performed for OpenQCD-FASTSUM, the FASTSUM collaboration's extension of openQCD [5], with the AVX-512 implementation.

2. Implementation

The implementation of the extension is guided by the expectation that lattice QCD simulations are memory bandwidth bound. The performance of the application is limited by the memory bandwidth between the processor and different cache levels rather than the capacity for floating point operations. The same assumption guides the existing AVX2, SSE and BlueGene/Q targeted implementations. Gauge matrices and spinors of the Wilson formulation are stored in memory as structures and SIMD vectors are constructed out of spinor degrees of freedom. Our implementation combines spinor and direction indices to construct the 512-bit vectors required.

The construction of a vector does not require any rearrangement of the data before loading into registers. However, different Dirac and directional indices are often handled differently, increasing the number of floating point instructions required. Since the primary objective is to optimise memory use, we consider this acceptable.

In the SSE and AVX2 extensions, direct insertions of assembly code is used to achieve full control over the compiled code. We use Intel intrinsic instructions, a library of C functions whos implementation is handled by the compiler. Compilers generally replace these routines with assembly instructions in a one-to-one correspondence. Intrinsic functions offer more freedom in choosing the optimal compiler and allows easier porting to different processor types. They leave the compiler with the task of choosing the optimal instruction for each operation and assigning data to registers. This is especially important since the number of registers is increased in Xeon Scalable CPUs. Using prewritten assembly code would confine the extension only to future processors with the higher register count.

We implement several core functions, including the Dirac operator, the application of the Sheikholeslami-Wohlert term, and several linear algebra functions. The extension is activated using the `AVX512` preprocessor flag, similarly to the existing `AVX2` and `SSE` preprocessor flags. When the flags are combined, the AVX-512 implementation is used when available.

PoS(LATTICE2018)039

Volume	Knights Landing			Skylake		
	AVX-512	Vanilla	Speedup	AVX-512	AVX2	Speedup
Single Precision						
4×4×4×4	3377	5659	1.69	36177	29839	1.21
8×4×4×4	5528	3360	1.65	34649	27769	1.25
8×8×4×4	5463	2043	2.67	36167	29289	1.23
8×8×8×4	4973	2361	2.11	34894	28476	1.23
8×8×8×8	3377	5659	1.69	26408	21617	1.22
16×8×8×8	3377	5659	1.69	25300	19180	1.32
Double precision						
4×4×4×4	6087	3346	1.82	26737	24681	1.08
8×4×4×4	5509	3223	1.71	26690	24609	1.08
8×8×4×4	4912	1747	2.81	26521	19687	1.35
8×8×8×4	4321	1630	2.65	25267	19312	1.31
8×8×8×8	3210	1873	1.71	18772	14471	1.29
16×8×8×8	2853	1670	1.71	15513	15125	1.03

Table 1: The performance of the functions Dw and Dw_dble (single and double precision respectively) in Mflops on single Knights Landing and Skylake cores.

3. Benchmarking

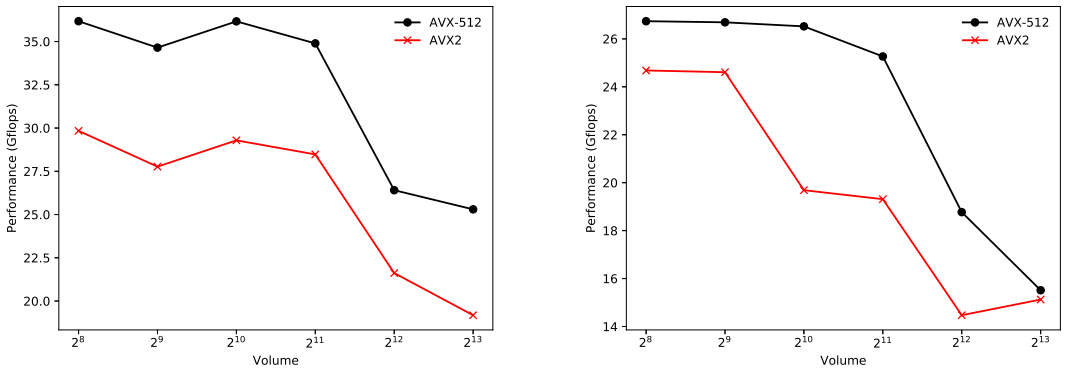

Figure 1: The performance of Dw() (left), Dw_dble() (right) performance measures run on a single Skylake core using the AVX-512 and AVX2 implementations.

We run several performance tests on the FASTSUM extension of OpenQCD 1.6 with and without the AVX-512 implementation on Cineca Marconi A2 cluster Intel Knights Landing nodes and the Supercomputing Wales Sunbird cluster with Intel Skylake nodes. On the Skylake cluster we use the Intel C Compiler to build the AVX-512 implementation using the compiler flags

```
-std=c89 -xCORE-AVX512 -mtune=skylake -O3 -DAVX512 -DAVX -DFMA3 -DPM.
```

The original AVX2 version is compiled with

PoS(LATTICE2018)039

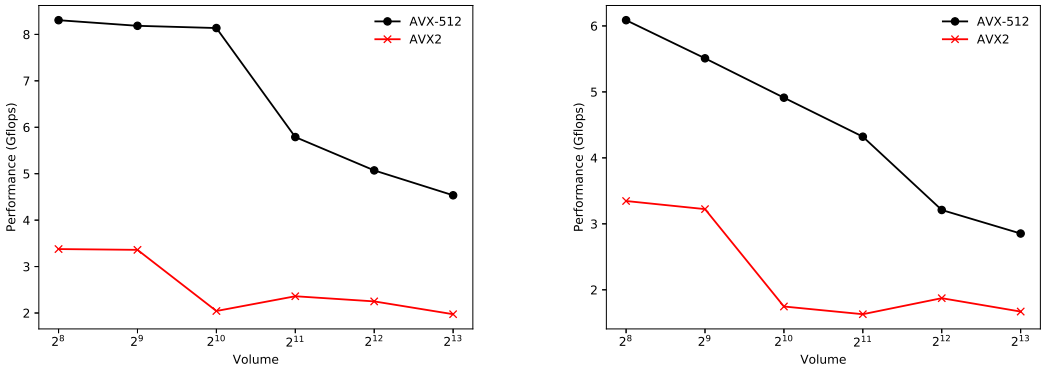

Figure 2: The Dw() (left), Dw_dble() (right) performance measures on a single KNL core using the AVX-512 and Vanilla implementations.

`-std=c89 -O3 -march=skylake-avx512 -DPM -DAVX -DFMA3 -DPM.`

On the Knights Landing cluster the AVX2 assembly code is not efficient and we compile the AVX-512 version with

`-std=c89 -xCORE-AVX512 -mtune=skylake -O3 -DAVX512`

and a vanilla OpenQCD 1.6 implementation using

`-std=c89 -xCORE-AVX512 -mtune=skylake -O3.`

Firstly, we have measured the single core performance of the Dirac operator itself using the timing tools provided in the openQCD code. These tests do not account for memory dependence, but only measure floating point performance. The performance in Mflops per second is shown in Tab. 1 and in Fig. 1 and 2. With small lattice sizes we observe a significant improvement, exceeding a factor of two with certain cases. Naturally the improvement is smaller in a realistic test case due to data dependencies and MPI communication.

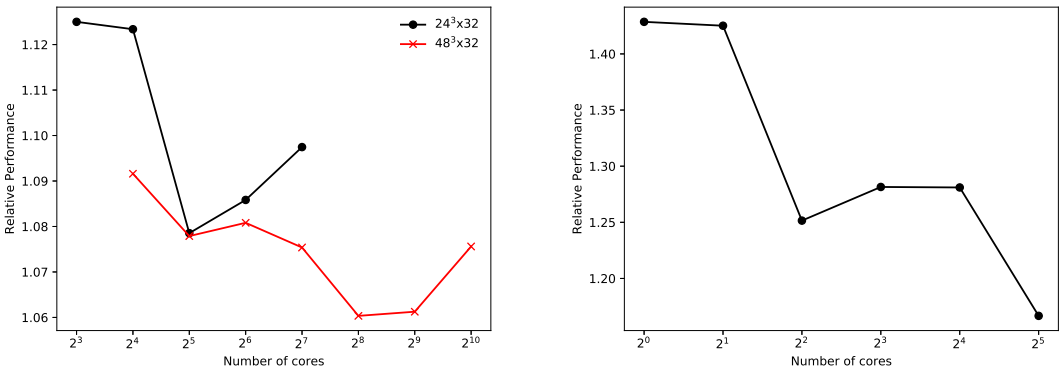

Figure 3: Left: Strong scaling performance on the Sunbird Skylake cluster measured against the AVX2 implementation. Right: Strong scaling performance on the Marconi Knights Landing cluster measured against the vanilla implementation

PoS(LATTICE2018)039

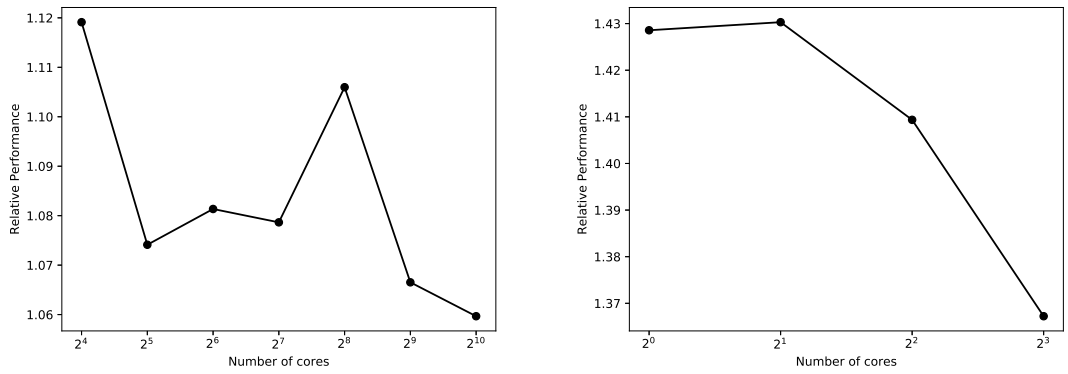

Figure 4: Left: Weak scaling performance on the Sunbird Skylake cluster using n cores and the lattice size $V = 48^3 \times n$ measured against the AVX2 implementation. Right: Weak scaling performance on the Marconi Knights Landing cluster using n cores the lattice size $V = 32^3 \times n$ measured against the vanilla implementation.

L	T	N	Time (s) / trajectory		Speedup
			AVX-512	Vanilla	
32	32	1	1.61e3	2.30e3	1.42
32	32	2	8.28e2	1.18e3	1.42
32	32	4	4.97e2	6.22e2	1.25
32	32	8	2.70e2	3.46e2	1.28
32	32	16	1.53e2	1.96e2	1.28
32	32	32	1.08e2	1.26e2	1.17
32	32	1	1.61e3	2.30e3	1.42
32	64	2	1.65e3	2.36e3	1.43
32	128	4	1.71e3	2.41e3	1.41
32	256	8	1.77e3	2.42e3	1.37

Table 2: Average timings per trajectory using N Knights Landing nodes with the volume $T \times L^3$. Speedup is measured against the vanilla implementation.

To get a more complete picture we measure the average time taken to generate a HMC trajectory. We have produced 6 trajectories starting from a random gauge configuration and report the average time per trajectory. We use a Wilson-Yukawa gauge action with $\beta = 1.5$, $c_0 = 5/3$ and $c_1 = -1/12$ and include two fermions with $\kappa = 0.278000465$ and 0.276509194 and $c_{sw} = 1$. Two levels of smearing are enabled in the fermion actions. Domain Deflation and blocking are enabled.

The timings for several lattice sizes and configurations of nodes are given in Tabs. 2 and 3 and shown in Fig. 3 and 4. Full compilation and runtime parameters and the simulation output on the Skylake system are published in [6]. The two version of the code scale similarly, with the AVX-512 version remaining faster in each case. In a full trajectory the improvement is reduced to approximately between 6% and 13%. On the Sunbird Skylake machine, each node has 40 cores and a minimal number allocation of nodes is used in each case. On the Marconi KNL system 64

PoS(LATTICE2018)039

L	T	N	n	Time (s) / trajectory		Relative
				AVX-512	AVX2	Speedup
32	48	1	16	1.31e3	1.43e3	1.09
32	48	1	32	7.96e2	8.58e2	1.08
32	48	2	64	3.96e2	4.28e2	1.08
32	48	4	128	1.99e2	2.14e2	1.08
32	48	7	256	1.16e2	1.23e2	1.06
32	48	13	512	6.04e1	6.41e1	1.06
32	48	26	1024	2.91e1	3.13e1	1.08
32	24	1	8	2.48e2	2.79e2	1.13
32	24	1	16	1.54e2	1.73e2	1.12
32	24	1	32	9.55e1	1.03e2	1.08
32	24	2	64	4.66e1	5.06e1	1.09
32	24	4	128	2.36e1	2.59e1	1.10
48	16	1	16	6.38e2	7.146e2	1.12
48	32	1	32	7.96e2	8.55e2	1.07
48	64	2	64	7.99e2	8.64e2	1.08
48	128	4	128	8.01e2	8.64e2	1.08
48	256	7	256	8.87e2	9.81e2	1.11
48	512	13	512	9.32e2	9.94e2	1.07
48	1024	26	1024	9.72e2	1.03e3	1.06

Table 3: Average timings per trajectory using N Skylake nodes with n cores and the volume $T \times L^3$. Speedup is measured relative to the AVX2 implementation.

L	T	N	n	Time (s) / trajectory		speedup
				AVX-512	AVX2	
Skylake						
32	24	1	32	2.28e3	2.47e3	1.08
32	24	2	64	1.12e3	1.21e3	1.08
32	24	4	128	5.68e2	6.22e2	1.10

Table 4: Average timings starting from a thermalised configuration per trajectory with the volume $T \times L^3$ with N nodes using n cores.

cores are used per node. No clear dependence on the lattice size or number of nodes can be deduced from the data.

Finally, we perform the same test using a thermalised starting configuration and a light quark. Two fermions are included $\kappa = 0.27831$ and 0.276509. The results are reported in Table 4. The speedup achieved is similar to the previous tests, between 8% and 10%.

4. Conclusion

We announce an open source implementation of the Dirac operator in OpenQCD 1.6 using

PoS(LATTICE2018)039

extended AVX-512 vector operations using Intel's intrinsic operations. These operations allow the application to make full use of the wider, 512-bit registers, reducing the total number of memory request, in particular those to L1 cache, and the number of floating point operations.

The implementation assumes that memory bandwidth is the main bottleneck in the application. Tradeoffs that reduce memory use at the cost of floating point operations and vector shuffles are considered acceptable. The application performs significantly better than the existing AVX2 implementation on Knights Landing and Skylake processors. In realistic benchmarking cases the improvement factor is between 6% and 12%.

5. Acknowledgements

We acknowledge the support of the Supercomputing Wales project, which is part-funded by the European Regional Development Fund (ERDF) via Welsh Government.

References

[1] "Intel Architecture Instruction Set Extensions and Future Features Programming Reference," January 2018, Ref. # 319433-032

[2] `luscher.web.cern.ch/luscher/openQCD/`

[3] `hpc.desy.de/simlab/codes/openqcd_bgopt/`

[4] `github.com/sa2c/OpenQCD-AVX512`
DOI: 10.5281/zenodo.1451764

[5] `fastsum.gitlab.io`

[6] "Sunbird Performance Data for Fastsum OpenQCD 1.0,"
DOI: 10.5281/zenodo.1475136

PoS(LATTICE2018)039