# Architecture independent SIMD Library for Lattice QCD

Patrick Steinbrecher[1,2], F. Karsch[1,2], S. Mukherjee[2], H. Ohno[2,3], P. Petreczky[2], S. Sharma[2] and M. Wagner[4]

[1] Universität Bielefeld - Germany, [2] Brookhaven National Laboratory - USA, [3] University of Tsukuba - Japan, [4] Indiana University - USA

## Introduction

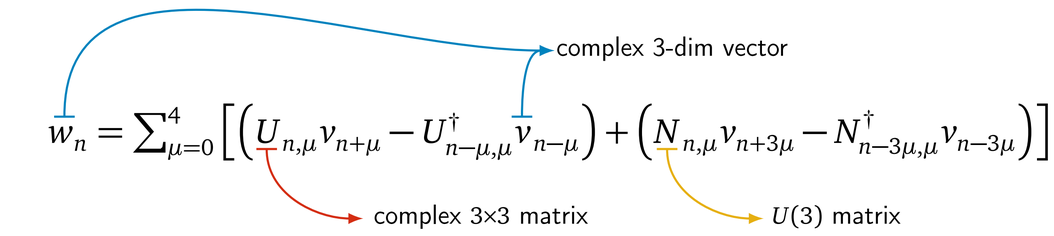
During the past few years single core performance was only driven by improving Single Instruction Multiple Data (SIMD) capabilities as introduced by Intel in 1996 with the x86 Multi Media Extension (MMX). It enabled to process two floating-point operations in a single instruction. As this vector register length increased over time to 512-bit, compiler technologies could not catch up to take advantage of large vector instructions, hence, leaving a huge gap for possible optimizations. At present, this gap has to be closed by hand using low-level architecture specific languages.

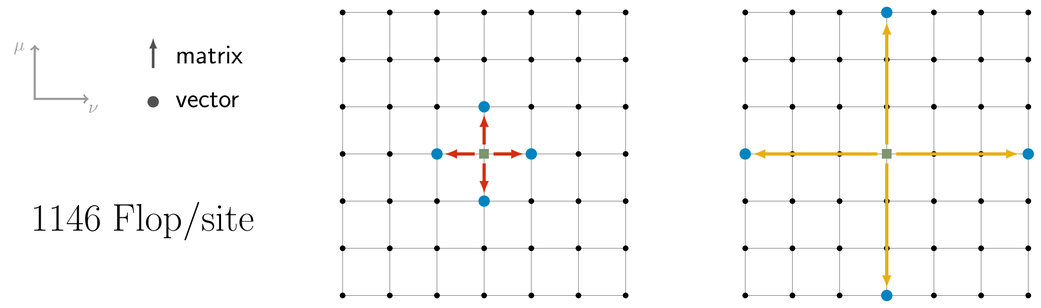
We have developed a C++ template based library for Lattice Quantum Chromodynamics (QCD) simulations using Staggered Fermions. It takes advantage of so-called low-level compiler intrinsics. We support nearly all available CPU instruction sets e.g. SSE, AVX, AVX512 and can easily support future architectures with vector registers larger than 512-bit.

In the following, we describe our implementation of the Highly Improved Staggered Quark (HISQ) formulation as well as specific optimizations [1]. A set of programs described here have been used in a calculation of conserved charge fluctuations (see neighboring poster).

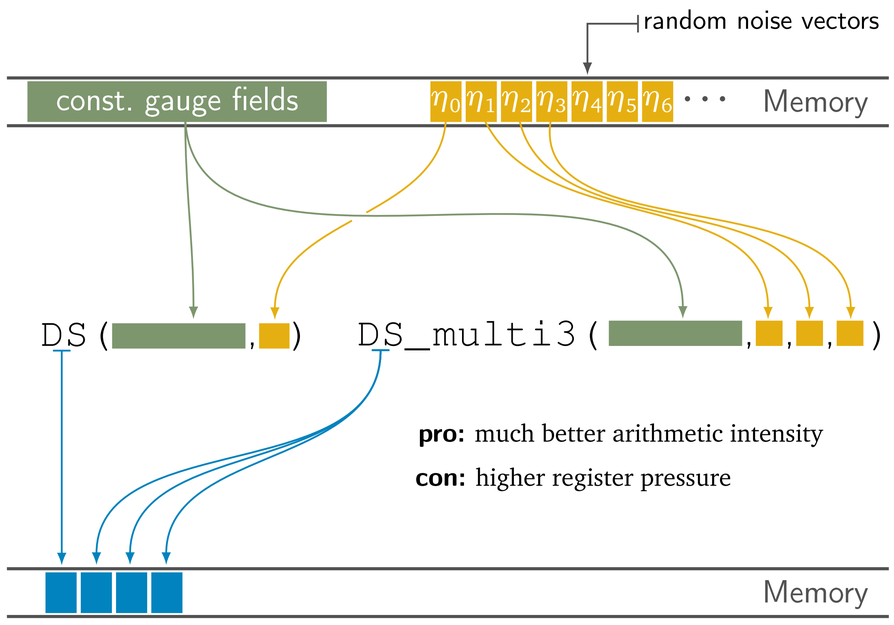
## 1. Dslash operator

The key operation in many Lattice QCD simulations is the inversion of the fermion matrix. It requires a 4-dimensional stencil which calculates the product of a vector $\nu$ by a sparse matrix known as the Dslash operator and stems from a discretized 4-dimensional derivative.



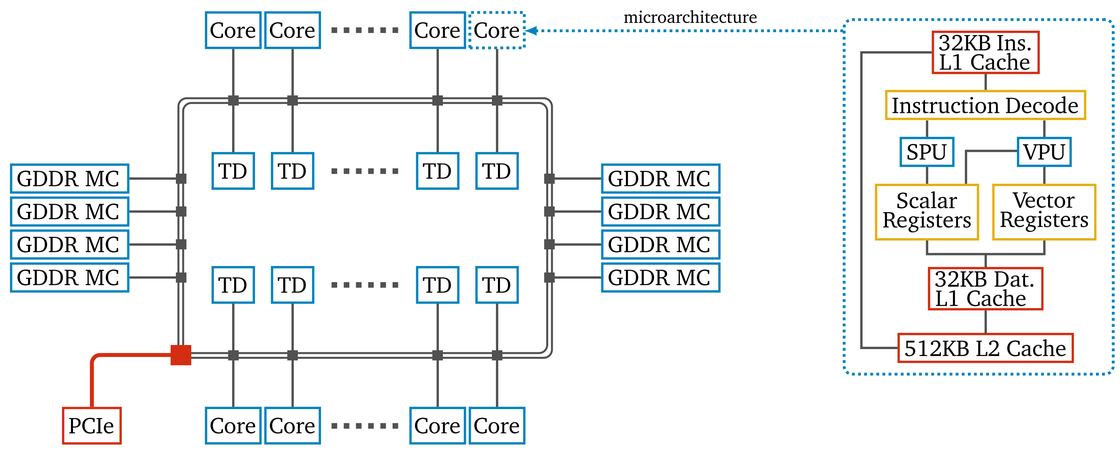




## 2. Solving multiple right-hand sides




For many Lattice QCD applications, a large number of fermion matrix inversions are performed on a single gauge field. In order to exploit reuse of these gauge fields, we can apply the Dslash operation for multiple right-hand sides (rhs) at once. Increasing the number of rhs from one to four more than doubles the arithmetic intensity (Flop/byte) of the Dslash operation.

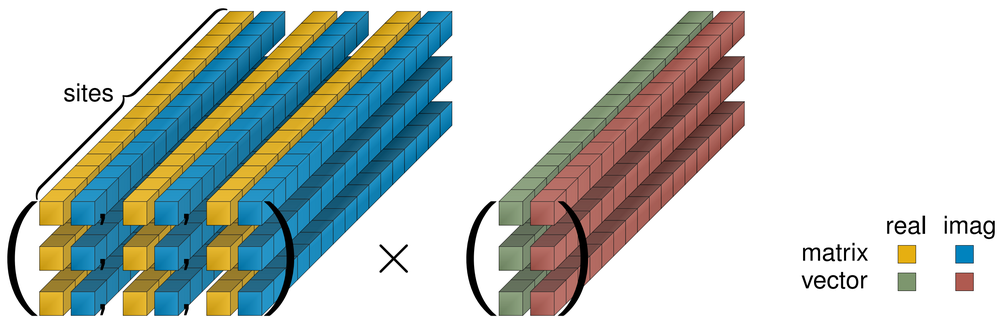
## 3. Intel Xeon Phi



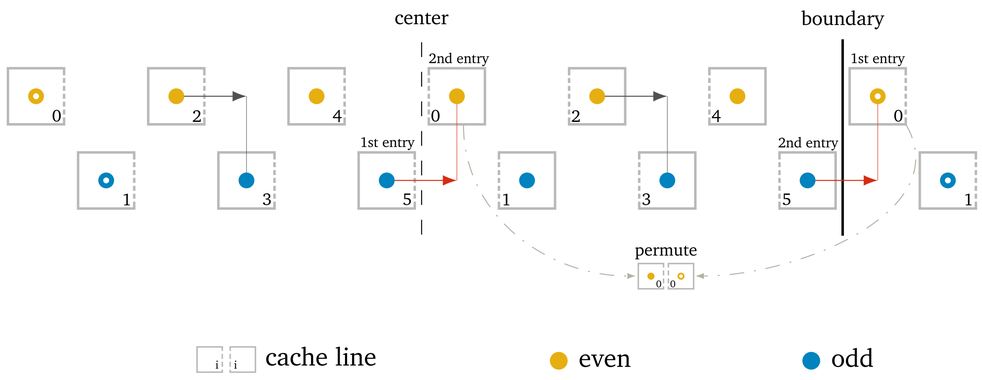

- x86 based many-core processor
  - runs a Linux $\mu$OS
  - in-order execution
  - up to 61 cores at 1.238 GHz
  - core boost for 7120 series
- parallelism through
  - large 512-bit SIMD registers
  - 4 hardware threads per core
- bandwidth benchmark


- peak fp32/fp64
  - 2.42/1.21 TFlop/s
- fully coherent cache

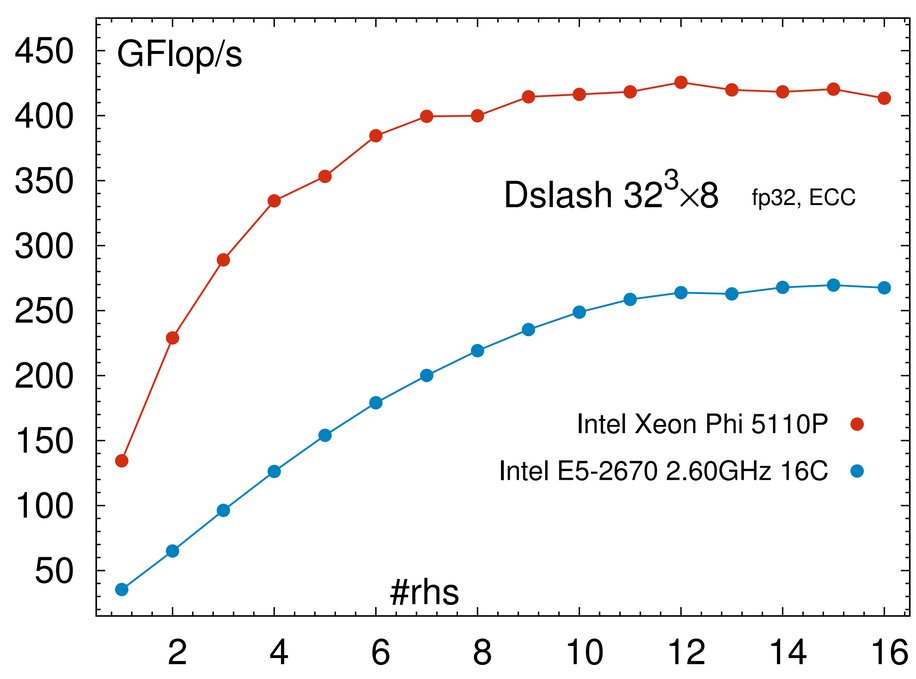
## 4. Single Instruction Multiple Data




On SIMD CPUs it is more efficient to process several matrix-vector products at the same time using a site-fusion method. A naive implementation could possibly be to create a "Struct of Arrays" (SoA) object for 16 matrices as well as for 16 vectors. One specific register then refers to the real or imaginary part of the same column gathered from all 16 vectors. However, we decided to store the real and imaginary in an alternating order because this is advantageous for the next Xeon Phi generation.




## 5. Dslash performance




| #rhs | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| Flop/byte | 0.80 | 1.25 | 1.53 | 1.73 | 1.87 | 1.98 |

[1] O. Kaczmarek et al., Conjugate gradient solvers on Intel Xeon Phi and NVIDIA GPUs. arXiv:1411.4439

SciDAC Scientific Discovery through Advanced Computing

NESAP

NERSC