Vectorization and LArSoft

James Amundson

Vectorization and LArSoft

20 June, 2017

What is vectorization?

SIMD

- **S** ingle
- **I** nstruction
- **M** ultiple
- **D** ata
  - A single instruction performs the same operation on multiple data items
- Less important is FMA
  - **F** used
  - **M** ultiply
  - **A** dd
    - A single instructions performs a multiplication and add, i.e., $y = mx + b$
      - particularly useful in linear algebra operations

SIMD in the wild

We will just consider the x86 family (and gloss over many details)

- **SSE2**
  - "**SSE2** (**Streaming SIMD Extensions 2**), is one of the Intel SIMD (Single Instruction, Multiple Data) processor supplementary instruction sets first introduced by Intel with the initial version of the Pentium 4 in 2001." – *Wikipedia*
  - 128 bits: 2 doubles or 4 floats
- **AVX**
  - "**Advanced Vector Extensions** (**AVX**) are extensions to the x86 instruction set architecture for microprocessors from Intel and AMD proposed by Intel in March 2008 and first supported by Intel with the Sandy Bridge[1] processor shipping in Q1 2011 and later on by AMD with the Bulldozer[2] processor shipping in Q3 2011." – *Wikipedia*
  - 256 bits: 4 doubles or 8 floats
- **AVX2**
  - "**AVX2** expands most integer commands to 256 bits and introduces FMA."
- **AVX-512**
  - "supported in Intel's Xeon Phi x200 (Knights Landing) processor"
  - 512 bits: 8 doubles or 16 floats

Uh… how about my machine?

- Linux: `cat /proc/cpuinfo | grep flags | head -n 1`

On my old Dell laptop:

```
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36
clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx rdtscp lm constant_tsc arch_perfmon
pebs bts rep_good nopl xtopology nonstop_tsc aperfmperf eagerfpu pni pclmulqdq dtes64 monitor
ds_cpl vmx smx est tm2 ssse3 cx16 xtpr pdcm pcid sse4_1 sse4_2 x2apic popcnt
tsc_deadline_timer aes xsave avx lahf_lm epb tpr_shadow vnmi flexpriority ept vpid xsaveopt
dtherm ida arat pln pts
```

On woof (SSI development machine):

```
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36
clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm constant_tsc
arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc aperfmperf eagerfpu pni pclmulqdq
dtes64 monitor ds_cpl vmx smx est tm2 ssse3 cx16 xtpr pdcm pcid dca sse4_1 sse4_2 x2apic
popcnt tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm ida arat epb pln pts dtherm
tpr_shadow vnmi flexpriority ept vpid fsgsbase smep erms xsaveopt
```

Uh… I have a Mac? (and I like upspeak?)

- OSX: `sysctl -a | grep machdep.cpu.features`

On my Mac:

```
machdep.cpu.features: FPU VME DE PSE TSC MSR PAE MCE CX8 APIC SEP MTRR PGE MCA CMOV PAT PSE36
CLFSH DS ACPI MMX FXSR SSE SSE2 SS HTT TM PBE SSE3 PCLMULQDQ DTES64 MON DSCPL VMX SMX EST TM2
SSSE3 FMA CX16 TPR PDCM SSE4.1 SSE4.2 x2APIC MOVBE POPCNT AES PCID XSAVE OSXSAVE SEGLIM64
TSCTMR AVX1.0 RDRAND F16C
```

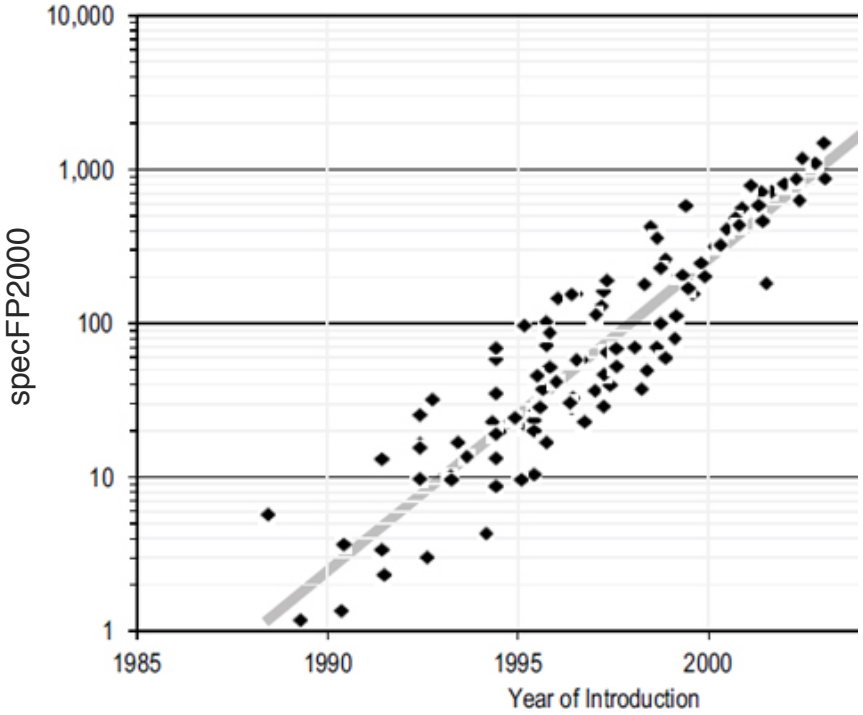
Why?

Why bother with this additional level of complication?

…Some context…

Computing is not what it used to be

- Computing in the good old days



taken from

THE FUTURE OF
COMPUTING PERFORMANCE

Game Over or Next Level?

Samuel H. Fuller and Lynette I. Millett, *Editors*

Committee on Sustaining Growth in Computing Performance

Computer Science and Telecommunications Board

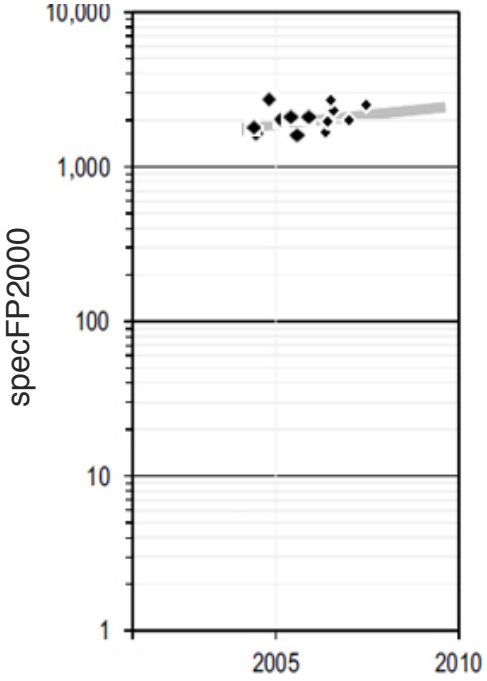
Division on Engineering and Physical Science

NATIONAL RESEARCH COUNCIL
OF THE NATIONAL ACADEMIES

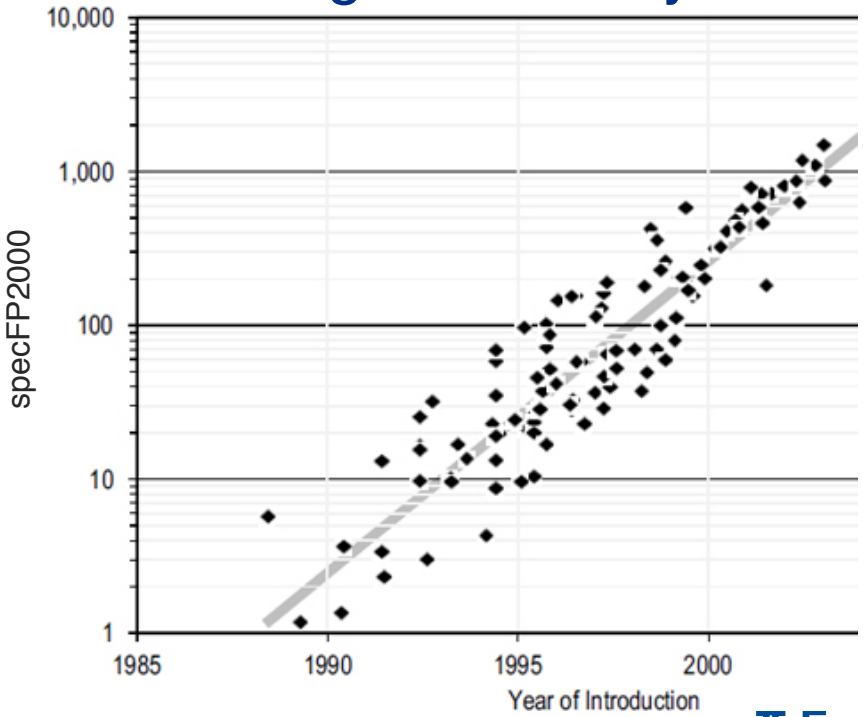
Things have changed already

- Computing is changing for reasons based on both physics and economics

recent times

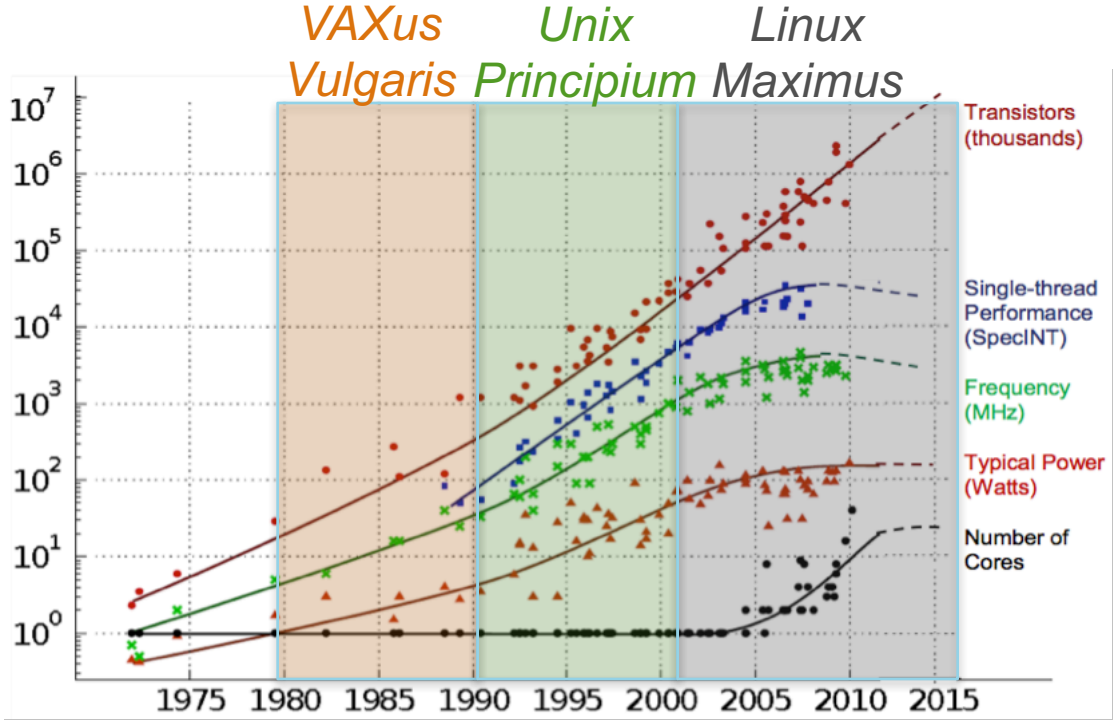


good old days



Fermilab

Hardware Trends are Changing



"Data Processing in Exascale-Class Computing Systems", Chuck Moore, AMD Corporate Fellow and CTO of Technology Group, presented at the 2011 Salishan Conference on High-speed Computing, Original data collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten, dotted line extrapolations by C. Moore

Fermilab

P5 Recognizes the Need for Evolution

- Strategic Plan for U.S. Particle Physics (P5 Report)

  *Rapidly evolving computer architectures and increasing data volumes require effective **crosscutting solutions that are being developed in other science disciplines and in industry**. Mechanisms are needed for the continued maintenance and development of major software frameworks and tools for particle physics and long-term data and software preservation, as well as investments to **exploit next-generation hardware and computing models**.*

Coming Epoch: Deus Ex Machina (Exascale Computing)

President Obama, July 29, 2015:

EXECUTIVE ORDER

CREATING A NATIONAL STRATEGIC COMPUTING INITIATIVE

By the authority vested in me as President by the Constitution and the laws of the United States of America, and to maximize benefits of high-performance computing (HPC) research, development, and deployment, it is hereby ordered as follows:

…

Sec. 2. Objectives. Executive departments, agencies, and offices (agencies) participating in the NSCI shall pursue five strategic objectives:

1. **Accelerating delivery of a capable exascale computing system that integrates hardware and software capability to deliver approximately 100 times the performance of current 10 petaflop systems across a range of applications representing government needs.**

…

Two reasons to care about Exascale

1. Supercomputer hardware is really just the leading edge of computing hardware.
   - Changes filter down to commodity hardware later.
2. HEP computing is likely to utilize Exascale hardware.
   - Another story.

Exascale Computing Challenges

http://science.energy.gov/ascr/research/scidac/exascale-challenges/

- Power. Power, power, power.
  - Naively scaling current supercomputers to exascale would require a dedicated nuclear power plant to operate.
    - ALCF's Mira: 4 MW, 0.01 exaflop
    - "The target is 20-40 MW in 2020 for 1 exaflop"
- *Many* more threads. Less memory and performance per thread.
- Memory bandwidth
  - "Memory bandwidth is not expected to scale with floating-point performance."
- I/O
  - "The I/O system at all levels – chip to memory, memory to I/O node, I/O node to disk—will be much harder to manage, as I/O bandwidth is unlikely to keep pace with machine speed."

How can I take advantage of SIMD instructions?

Intrinsics (*bad*)

```
01 void
02
03 dotmul_intrins2 (short A[], short B[], short &C, const int SIZE)
04
05 {
06
07 register int k;
08
09 short sarr[4];
10
11 register __m64 *a_ptr, *b_ptr, catch_mul, part_sum;
12
13 catch_mul = 0;
14
15 part_sum = 0;
16
17 for (k = 0; k < SIZE; k += 4)
18
19 {
20
21 a_ptr = (__m64*) &A[k];
22
23 b_ptr = (__m64*) &B[k];
24
25 catch_mul = _m64_pmpyshr2 (*a_ptr, *b_ptr, 0);
26
27 part_sum = _m_paddw (part_sum, catch_mul);
28
29 }
30
31
32 __st8_rel (&sarr [0], _m_to_int64 (part_sum));
33
34 C = sarr[0] + sarr[1] + sarr[2] + sarr[3];
35
36 }
```

- Not portable
- Hard to read
- Basically, disgusting

Vc (*better*)

Let's start from the code for calculating a 3D scalar product using builtin floats:

```
using Vec3D = std::array<float, 3>;
float scalar_product(Vec3D a, Vec3D b) {
  return a[0] * b[0] + a[1] * b[1] + a[2] * b[2];
}
```

Using Vc, we can easily vectorize the code using the `float_v` type:

```
using Vc::float_v
using Vec3D = std::array<float_v, 3>;
float_v scalar_product(Vec3D a, Vec3D b) {
  return a[0] * b[0] + a[1] * b[1] + a[2] * b[2];
}
```

https://github.com/VcDevel/Vc

- C++ class to abstract out SIMD commands
- Used in GeantV
- Does not yet have full support for AVX-512

- I tried it, but never liked it
  - Your mileage may vary

vectorclass (*even better for me*)

```
// Example 1a. Calculations on arrays
float a[8], b[8], c[8];       // declare arrays
...                           // put values into arrays
for (int i = 0; i < 8; i++) { // loop for 8 elements
    c[i] = a[i] + b[i]*1.5f;  // operations on each element
}
```

The vector class library allows you to write this code as vectors:

```
// Example 1b. Same code using vectors
#include "vectorclass.h"      // use vector class library
float a[8], b[8], c[8];       // declare arrays
...                           // put values into arrays
Vec8f avec, bvec, cvec;       // define vectors
avec.load(a);                 // load array a into vector
bvec.load(b);                 // load array b into vector
cvec = avec + bvec * 1.5f;    // do operations on vectors
cvec.store(c);                // save result in array c
```

http://www.agner.org/optimize/

- an ugly web site that is the best source of low-level optimization information I know of

- Another C++ class to abstract out SIMD commands
- Used in Synergia

- I tried it, and it immediately made sense
  - Your mileage may vary

Automatic vectorization by compiler (best, hands down, but…)

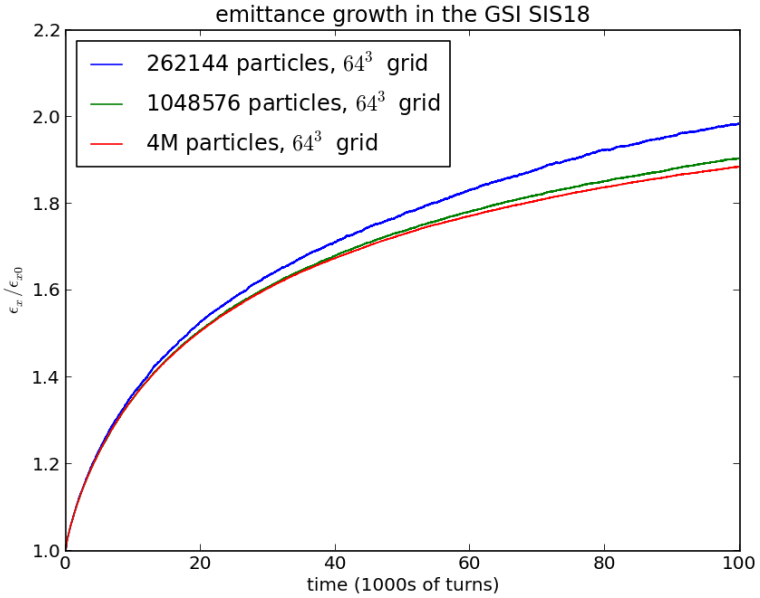
- Most modern compilers can do some automatic vectorization
- Ideal
- Usually requires –fast-math, or some equivalent
- icc (Intel)
  - Best at vectorization I know of
  - Still can't get everything
  - When icc beats other compilers by a significant margin, it's probably vectorizing
- gcc
  - I have trouble even constructing an example that will vectorize
- clang
  - Similar to gcc, possibly a little better
    - For some reason, much better on BlueGene/Q
      - Not much help, though…

Cheat (better than best)

- Many widely-available math libraries include SIMD intrinsic support
  - Most linear algebra libraries
    - Eigen and other modern C++ libraries
    - Atlas
  - FFTW
  - Intel Math Kernel Library
  - Others
- How are SIMD intrinsics implemented in these libraries?
    - Don’t know
    - Don’t care

Real world example: Synergia accelerator simulation

- Study emittance growth over 100,000 revolutions in GSI SIS18 accelerator
  - Effects of statistical noise are important
- Largest beam dynamics simulation ever



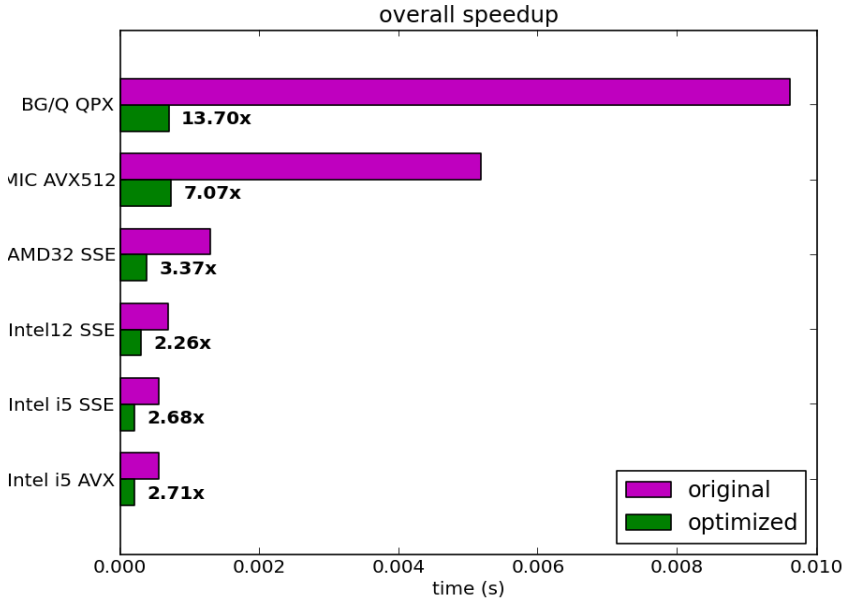
71 steps/turn
7,100,000 steps
4,194,304 particles
29,779,558,400,000 particle-steps
1,238,158,540,800,000 calls to “drift”

Yes, that's over a quadrillion

Fermilab

Explicit vectorization of Synergia

- C++ template-based
  - vectorclass
    - http://www.agner.org
  - GSVector
    - Generalized SIMD Vector
    - Part of Synergia
    - Compile-time vectorization model choice
      - double to AVX512
  - Rearranged data to facilitate vectorization



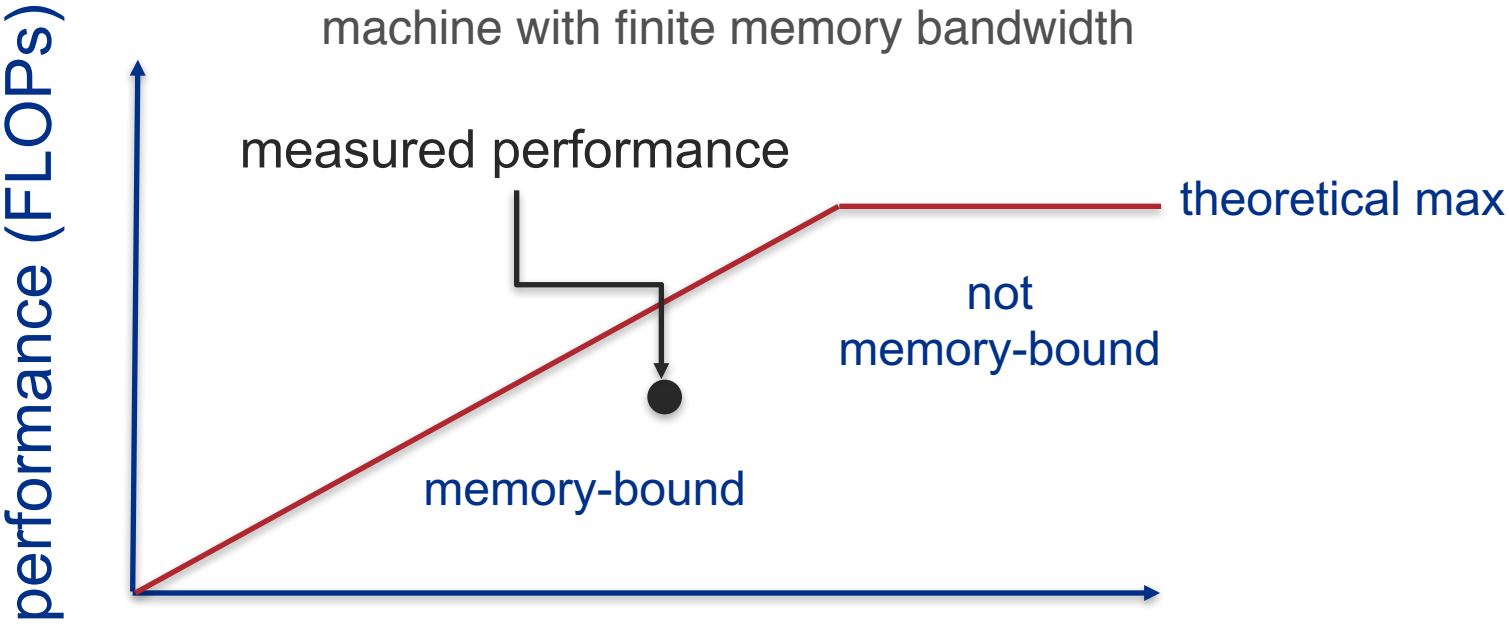
Fermilab

Does vectorization always help?

Will it always work like that?

Understanding CPU constraints: the Roofline Model

Theoretical limits of floating point performance in a simple machine with finite memory bandwidth

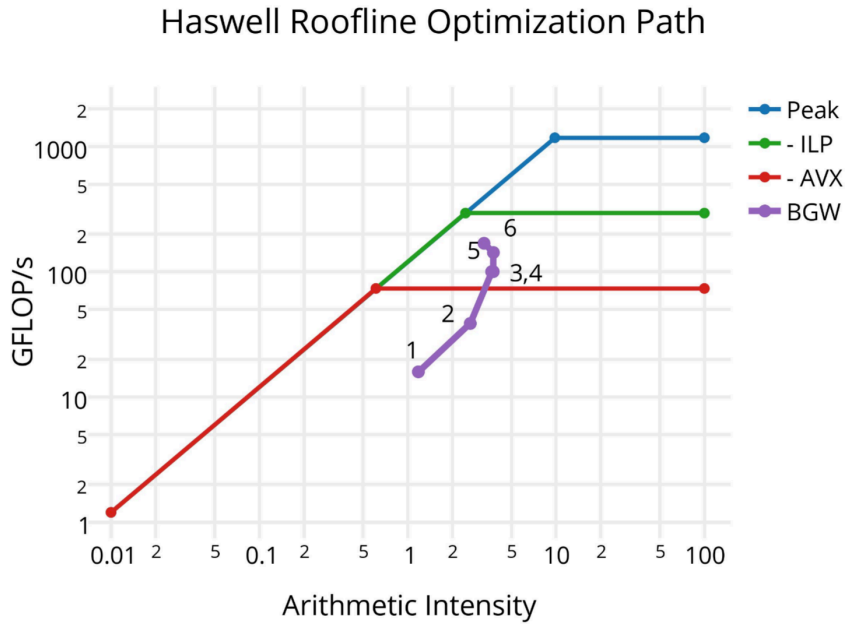


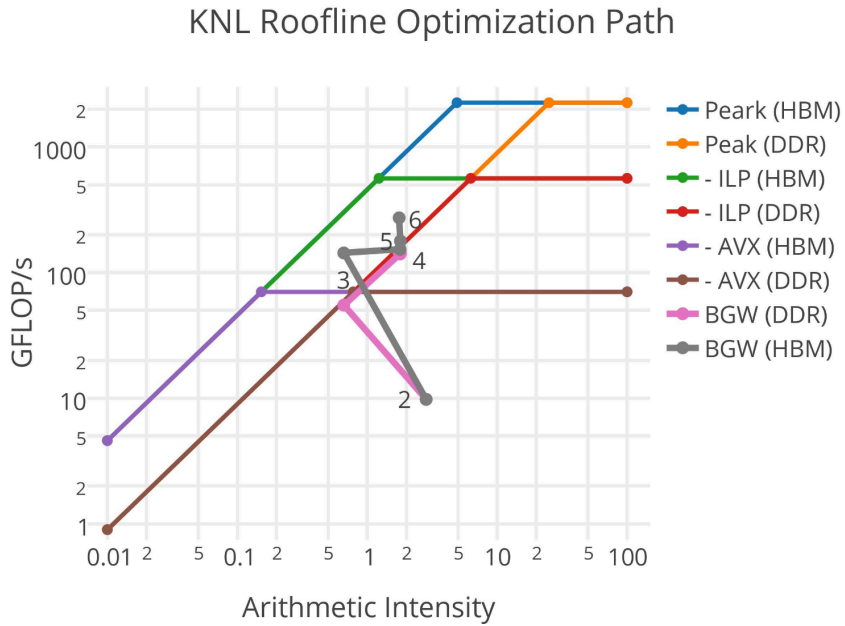
Real-world roofline

https://anl.app.box.com/v/IXPUG2016-presentation-29

Optimizing the BerkeleyGW code.

The BerkeleyGW Package is a set of computer codes that calculates the quasiparticle properties and the optical responses of a large variety of materials from bulk periodic crystals to nanostructures such as slabs, wires and molecules.

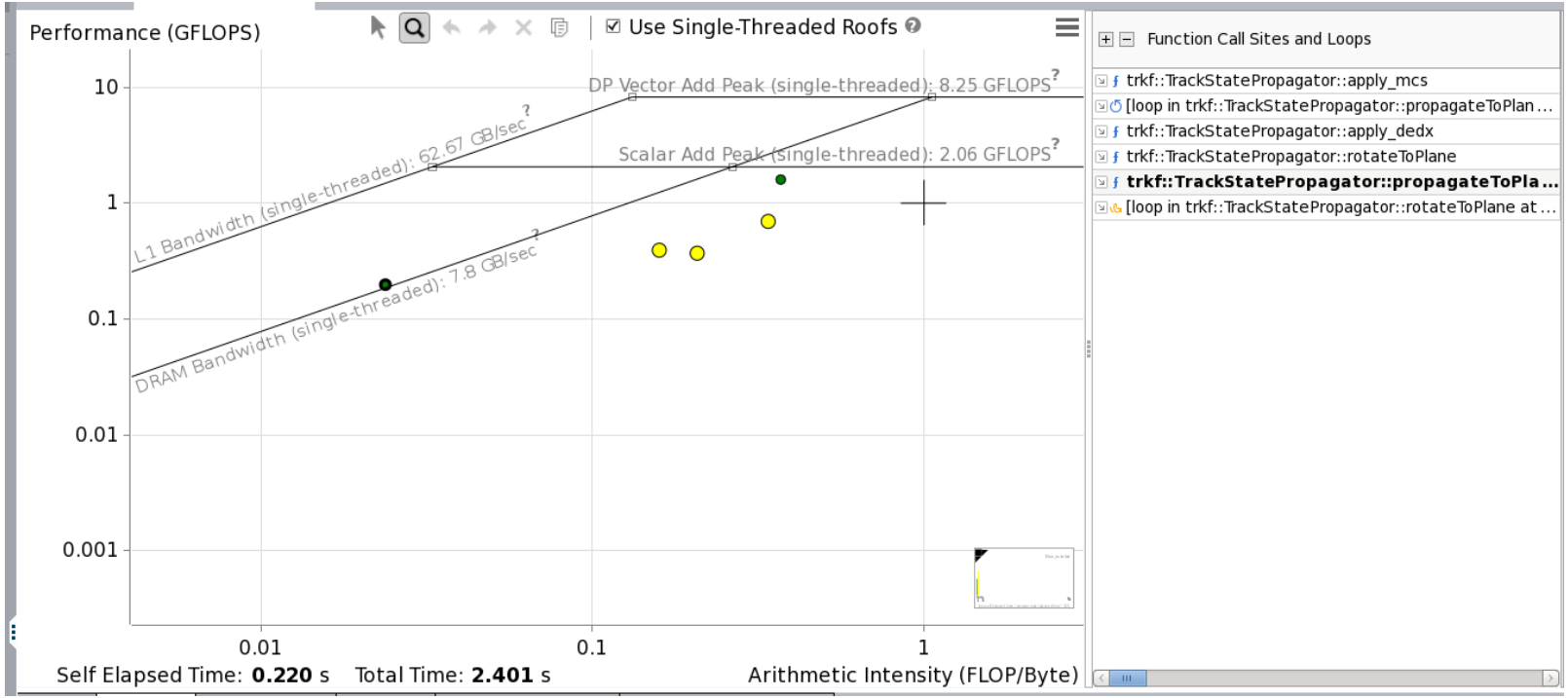




Fermilab

First LArSoft roofline analysis

- Performed by Giuseppe Cerati using Intel Advisor



Fermilab

Conclusions

- SIMD instructions have the potential to improve floating point performance 2x – 16x
- Future trends will make these types of instructions more prevalent
- Multiple libraries are available to help
  - Direct use of intrinsics is not recommended
- Memory bandwidth constraints can limit the usefulness of SIMD