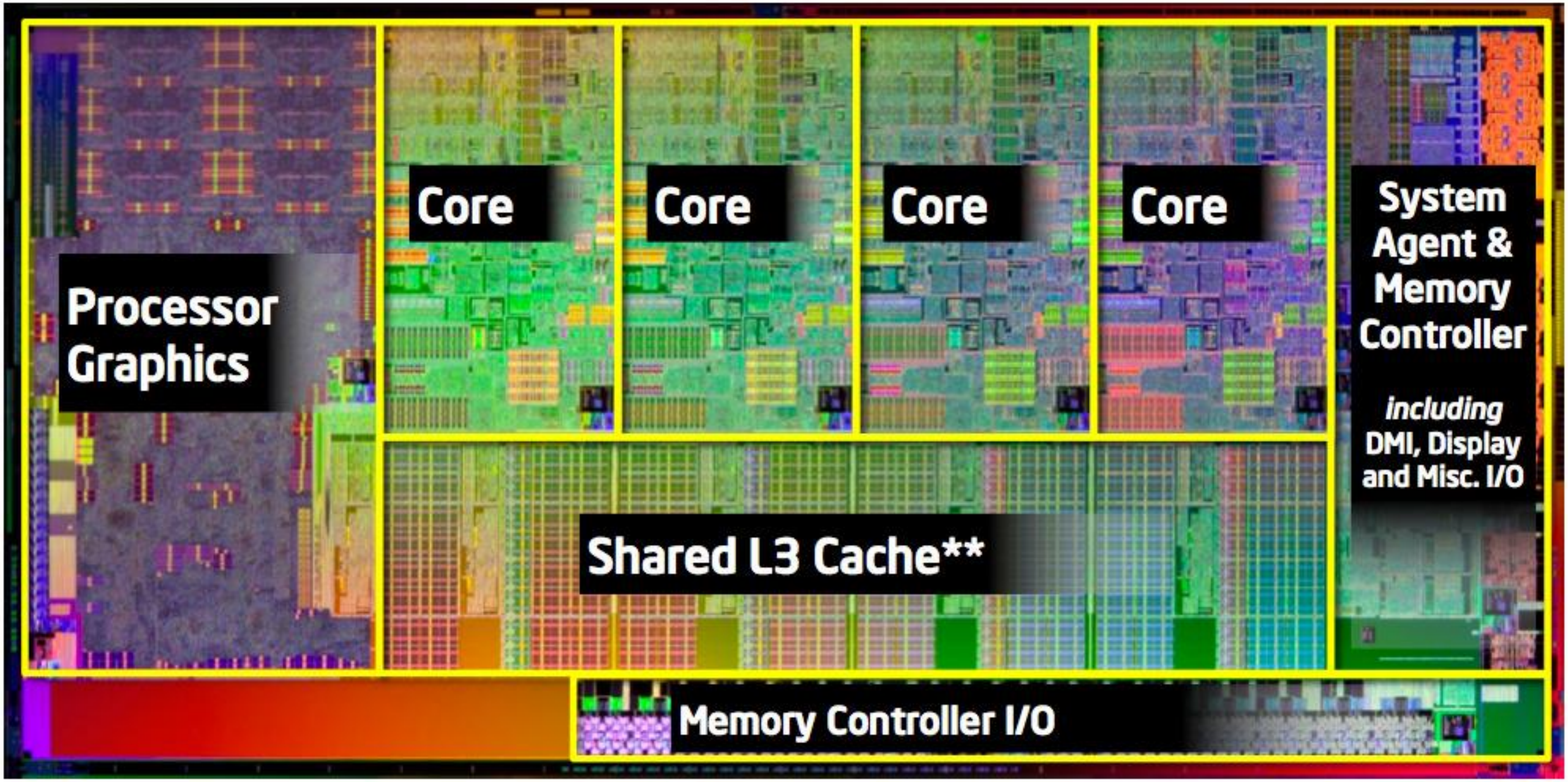
SIMD in C++

Presented
January, 2017

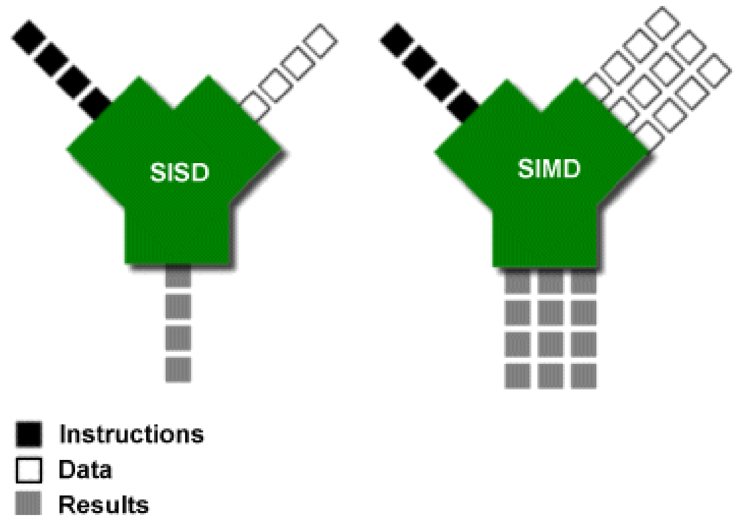
BrightSource

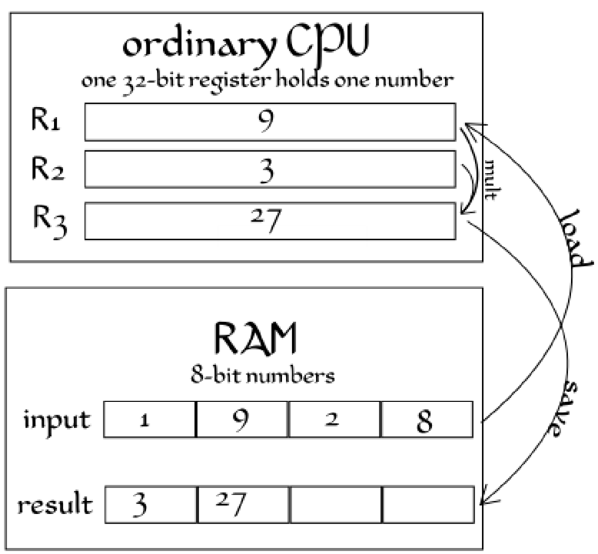
Doc ID:

Why C++?

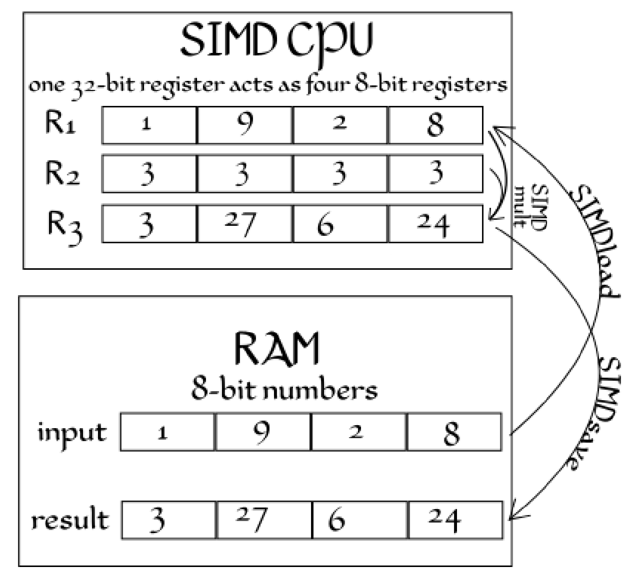

Processor Graphics
Core
Core
Core
Core
System Agent & Memory Controller
including DMI, Display and Misc. I/O
Shared L3 Cache**
Memory Controller I/O

What is SIMD





Operation Count:
4 loads, 4 multiplies, and 4 saves



Operation Count:
1 load, 1 multiply, and 1 save

History

- Intel
  - SSE – Streaming SIMD Extensions
  - 128 bit registers, XMM
    - 1999 – Version1
    - 2001 – Version2
    - 2003 – Version3
    - 2006 – Version4
  - AVX - Advanced Vector Extensions
  - 256 bit registers, YMM
    - 2011 – Version1
    - 2013 – Version2
  - AVX-512
    - 2015 – AVX 512
  - …

AVX-512 register scheme as extension from the AVX (YMM0-YMM15) and SSE (XMM0-XMM15) registers

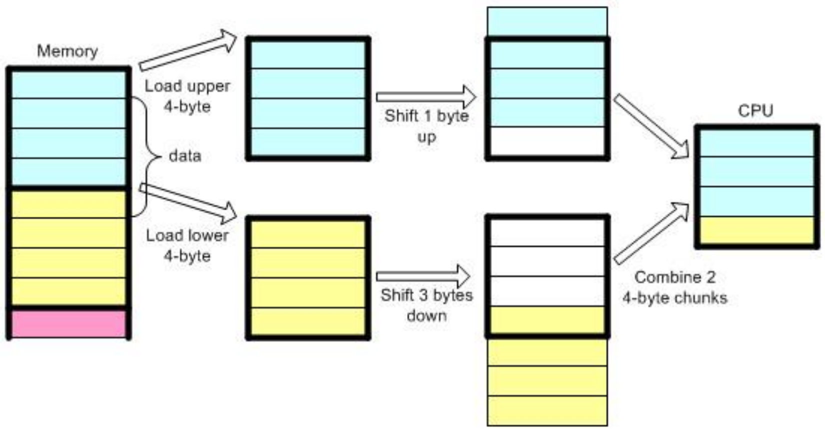
511 256	255 128	127 0
ZMM0	YMM0	XMM0
ZMM1	YMM1	XMM1
ZMM2	YMM2	XMM2
ZMM3	YMM3	XMM3
ZMM4	YMM4	XMM4
ZMM5	YMM5	XMM5
ZMM6	YMM6	XMM6
ZMM7	YMM7	XMM7
ZMM8	YMM8	XMM8
ZMM9	YMM9	XMM9
ZMM10	YMM10	XMM10
ZMM11	YMM11	XMM11
ZMM12	YMM12	XMM12
ZMM13	YMM13	XMM13
ZMM14	YMM14	XMM14
ZMM15	YMM15	XMM15
ZMM16	YMM16	XMM16
ZMM17	YMM17	XMM17
ZMM18	YMM18	XMM18
ZMM19	YMM19	XMM19
ZMM20	YMM20	XMM20
ZMM21	YMM21	XMM21
ZMM22	YMM22	XMM22
ZMM23	YMM23	XMM23
ZMM24	YMM24	XMM24
ZMM25	YMM25	XMM25
ZMM26	YMM26	XMM26
ZMM27	YMM27	XMM27
ZMM28	YMM28	XMM28
ZMM29	YMM29	XMM29
ZMM30	YMM30	XMM30
ZMM31	YMM31	XMM31

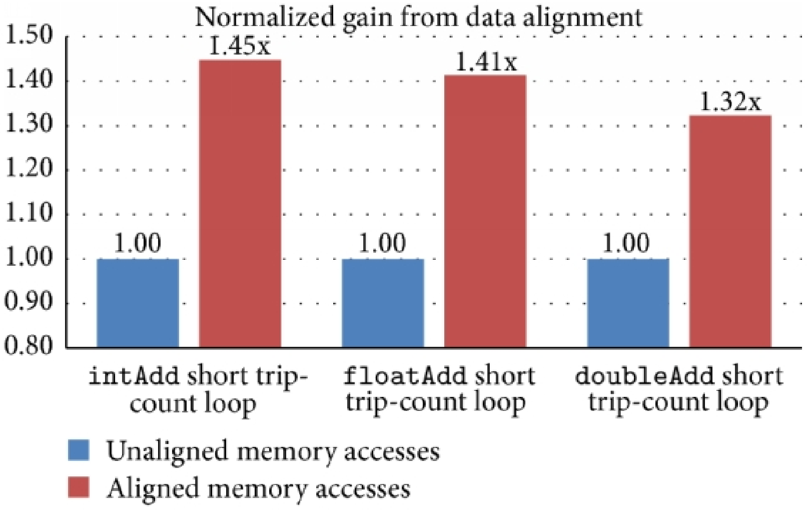
Intro


simd_intro.cpp

Issues

- Data alignment
  - For optimal performance SIMD data needs to be aligned in memory
  - In order to load to a register of size 128, the data needs to be aligned to 128
  - It’s possible to load/store unaligned data, you might pay performance penalty, depending on hardware
  - http://en.cppreference.com/w/cpp/language/alignof
  - http://en.cppreference.com/w/cpp/language/alignas





Issues - continued

- Data arranging, reorg
  - In order for SIMD to be efficient you might want to arrange the data “by properties”
  - Not so good (Array of Structures):
    - std::vector<Point3> points
  - Better (Structure of Arrays)
    - struct Points { std::vector<float> Xs, Ys, Zs; }
  - Though there are operations you can do within the register…

R	G	B

Point Structure with Data in AoS Arrangement

```
struct Point{
    float r;
    float g;
    float b;
}
```

R	G	B	R	G	B	R	G	B

Points Structure with Data in SoA Arrangement

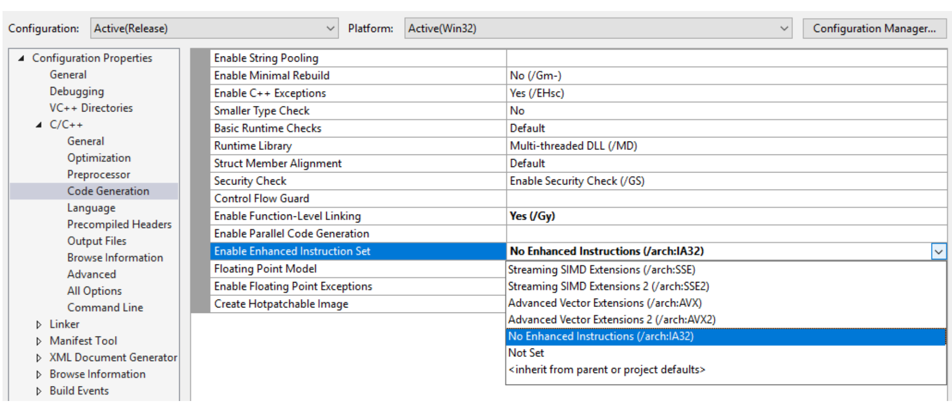
```
struct Points{
    float* x;
    float* y;
    float* z;
}
```

R	R	R	G	G	G	B	B	B

Don’t do it yourself

- We have libraries!
  - Some are Low level:
    - https://github.com/NumScale/boost.simd
    - http://ermig1979.github.io/Simd/help/index.html
    - https://software.intel.com/en-us/intel-ipp
  - Some are High level
    - http://eigen.tuxfamily.org/
    - https://opencv.org/
  - Compilers usually do some of it for you, familiarize yourself with it:

```cpp
#include <boost/simd/pack.hpp>
#include <iostream>

namespace bs = boost::simd;

int main()
{
   bs::pack<float,4> p{1.f,2.f,3.f,4.f};
   std::cout << p + 10*p << "\n";

   return 0;
}
```



Is your software compute or data bound?

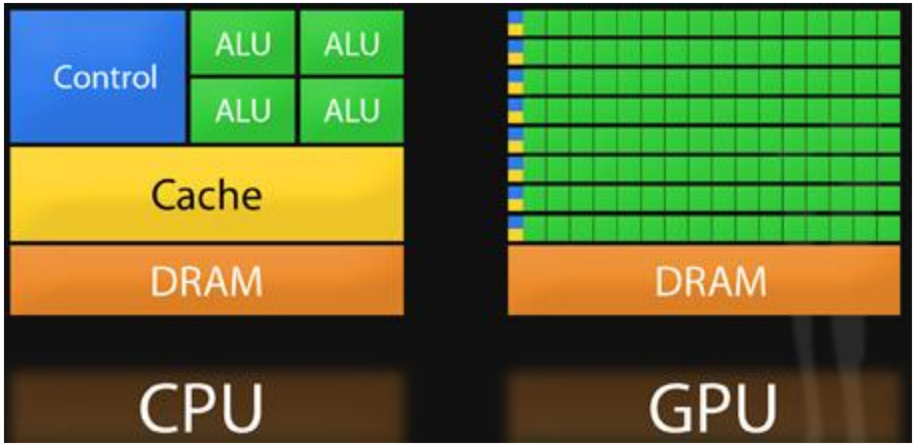
- http://www.overbyte.com.au/misc/Lesson3/CacheFun.html
- https://gist.github.com/jboner/2841832

```
Latency Comparison Numbers
--------------------------
L1 cache reference                           0.5 ns
Branch mispredict                            5   ns
L2 cache reference                           7   ns                      14x L1 cache
Mutex lock/unlock                           25   ns
Main memory reference                      100   ns                      20x L2 cache, 200x L1 cache
Compress 1K bytes with Zippy             3,000   ns        3 us
Send 1K bytes over 1 Gbps network       10,000   ns       10 us
Read 4K randomly from SSD*             150,000   ns      150 us          ~1GB/sec SSD
Read 1 MB sequentially from memory     250,000   ns      250 us
Round trip within same datacenter      500,000   ns      500 us
Read 1 MB sequentially from SSD*     1,000,000   ns    1,000 us    1 ms  ~1GB/sec SSD, 4X memory
Disk seek                           10,000,000   ns   10,000 us   10 ms  20x datacenter roundtrip
Read 1 MB sequentially from disk    20,000,000   ns   20,000 us   20 ms  80x memory, 20X SSD
Send packet CA->Netherlands->CA    150,000,000   ns  150,000 us  150 ms
```

GP-GPU Parallelization Alternative

- Another option to gain performance on dedicated hardware is using the GPU processor abilities for General Purpose

```
#include <vector>
#include <algorithm>
#include <boost/compute.hpp>

namespace compute = boost::compute;

int main()
{
    // get the default compute device
    compute::device gpu = compute::system::default_device();

    // create a compute context and command queue
    compute::context ctx(gpu);
    compute::command_queue queue(ctx, gpu);

    // generate random numbers on the host
    std::vector<float> host_vector(1000000);
    std::generate(host_vector.begin(), host_vector.end(), rand);

    // create vector on the device
    compute::vector<float> device_vector(1000000, ctx);

    // copy data to the device
    compute::copy(
        host_vector.begin(), host_vector.end(), device_vector.begin(), queue
    );

    // sort data on the device
    compute::sort(
        device_vector.begin(), device_vector.end(), queue
    );

    // copy data back to the host
    compute::copy(
        device_vector.begin(), device_vector.end(), host_vector.begin(), queue
    );

    return 0;
}
```



- Very low level, separate area of expertise
- High latency for data transfer to/from GPU
- Lots of primitive CPUs
- https://github.com/boostorg/compute

BrightSource

Limitless

brightsourceenergy.com