

# Lecture 6: Scientific computing in C++

- A review of the work you completed in ACSE-3, specifically lecture 3 would be helpful over the next few weeks
- We're going to be writing versions of algorithms you should know
- But we're going to spend a lot of time thinking about how we might write them in C++
- C++ standard is big!
- There are lots of different ways to write C++ programs
- We're going to talk about some of things important for scientific computing/computational mathematics

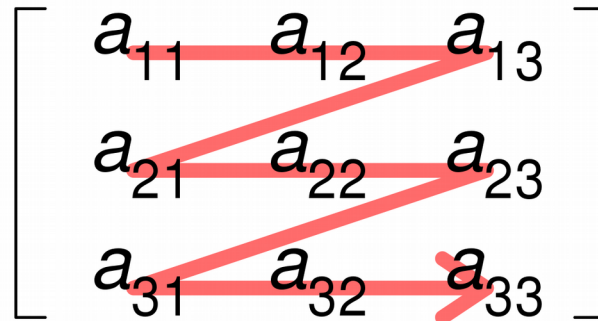
# Time to write some code!

- Name your C++ file “Matrix.cpp” and “Matrix.h”
- What are some things a “Matrix” class might need to do?

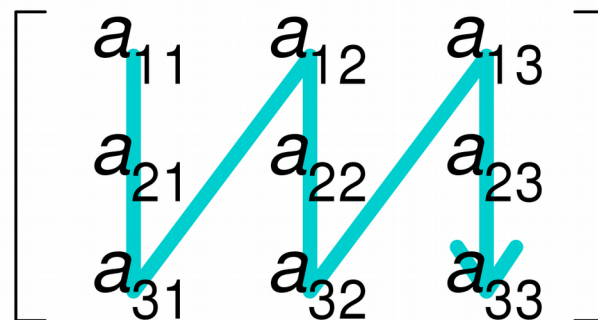
# Column/row major ordering

- How to store a dense matrix in a 1D array (remember 2D array are just 1D arrays under the hood)

Row-major order



Column-major order

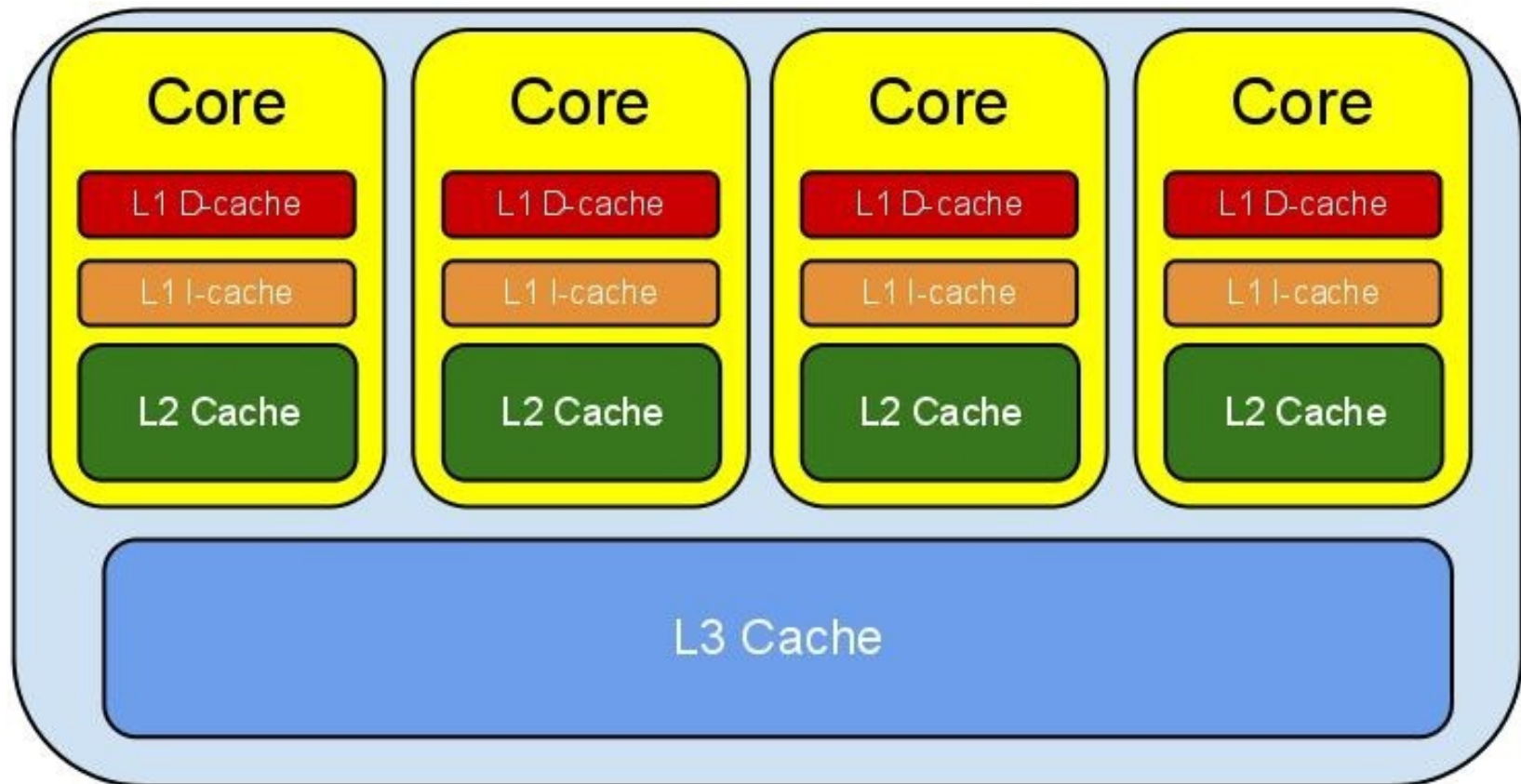


# Time to write some code!

- Continue writing the new Matrix class

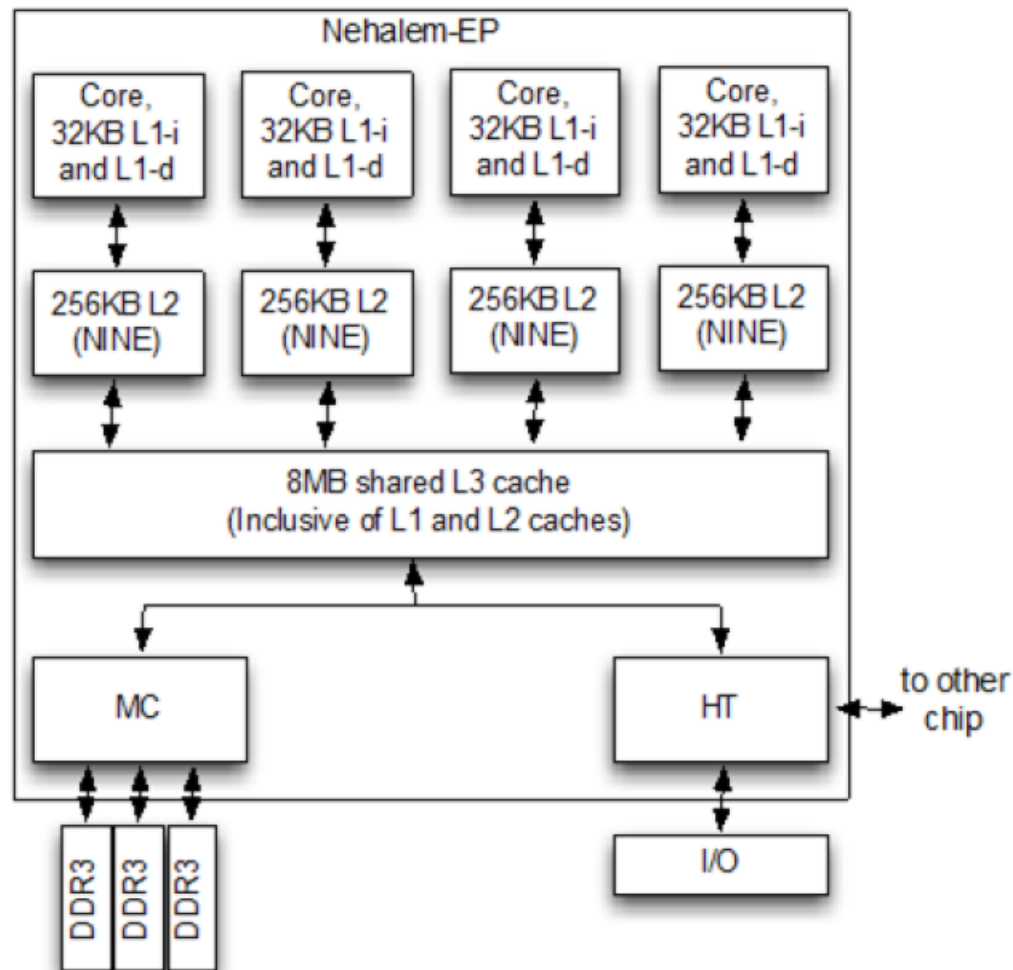
# Column/row major ordering

- Why do we care?
- Modern CPUs have a hierarchy of caches



# Caches

- Intel Nehalem architecture (2008 - 2011)



# Caches

- Caches near to the “cores” are small but close (and quick)
- The further away, the more expensive accessing memory is

Core i7 Xeon 5500 Series Data Source Latency (approximate)

local L1 CACHE hit,	~4 cycles ( 2.1 - 1.2 ns )
local L2 CACHE hit,	~10 cycles ( 5.3 - 3.0 ns )
local L3 CACHE hit, line unshared	~40 cycles ( 21.4 - 12.0 ns )
local L3 CACHE hit, shared line in another core	~65 cycles ( 34.8 - 19.5 ns )
local L3 CACHE hit, modified in another core	~75 cycles ( 40.2 - 22.5 ns )
local DRAM	~60 ns

# Time to write some code!

- Now that we are “cache aware” let’s use that in our Matrix class



# Caches

- Again why do we care? Caches are completely transparent to the user, so how can we benefit from thinking about them?
- When you access memory, the CPU automatically grabs neighbouring memory and stores it in the cache
- If you try to order your loops (etc) in your algorithm to use data close in memory, you will be rewarded!

# BLAS/LAPACK

- Thankfully a lot of these algorithms are commonly used in scientific computing
- BLAS – Basic Linear Algebra Subroutines (originally 1979)
- LAPACK – Linear Algebra PACKage (originally 1992)
- These are standard API that define common operations
- BLAS level 1 – vector operations, like  $ax + y$
- BLAS level 2 – Matrix vector operations, like  $A * x$
- BLAS level 3 – Matrix matrix operations, like  $A * B$
- LAPACK includes things as diverse as eigenvalue decompositions, etc

# BLAS/LAPACK

Here are descriptions of the BLAS/LAPACK standards:

<http://www.netlib.org/blas/>

<http://www.netlib.org/lapack/>

## ◆ daxpy()

```
subroutine daxpy ( integer          N,  
                  double precision  DA,  
                  double precision, dimension(*) DX,  
                  integer           INCX,  
                  double precision, dimension(*) DY,  
                  integer           INCY  
)
```

### DAXPY

#### Purpose:

DAXPY constant times a vector plus a vector.  
uses unrolled loops for increments equal to one.

#### Parameters

[in]	<b>N</b>	N is INTEGER number of elements in input vector(s)
[in]	<b>DA</b>	DA is DOUBLE PRECISION On entry, DA specifies the scalar alpha.
[in]	<b>DX</b>	DX is DOUBLE PRECISION array, dimension ( 1 + ( N - 1 ) * abs( INCX ) )

# BLAS/LAPACK

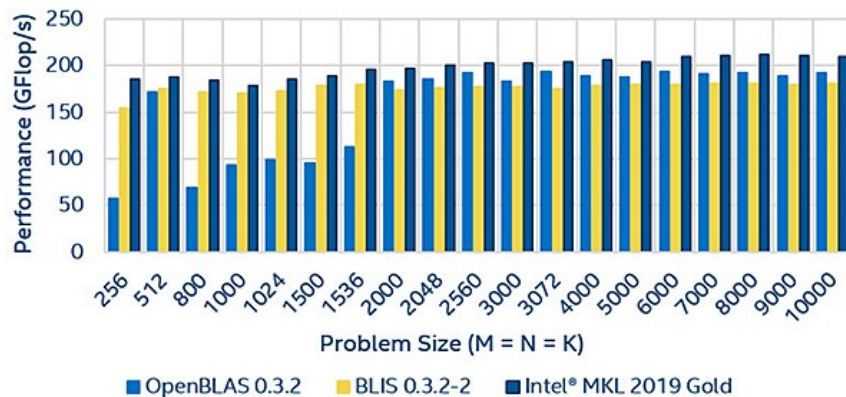
- You should always use BLAS/LAPACK libraries to do basic linear algebra
- Many different people have produced their own BLAS/LAPACK libraries, including Intel, most supercomputer vendors, GPU vendors, etc, which are very high performance/threaded/thread safe/deterministic on different hardware
- You can pay a lot of money for BLAS/LAPACK implementations
- MATLAB for example started as basically a nice interface for BLAS/LAPACK under the hood

# BLAS/LAPACK

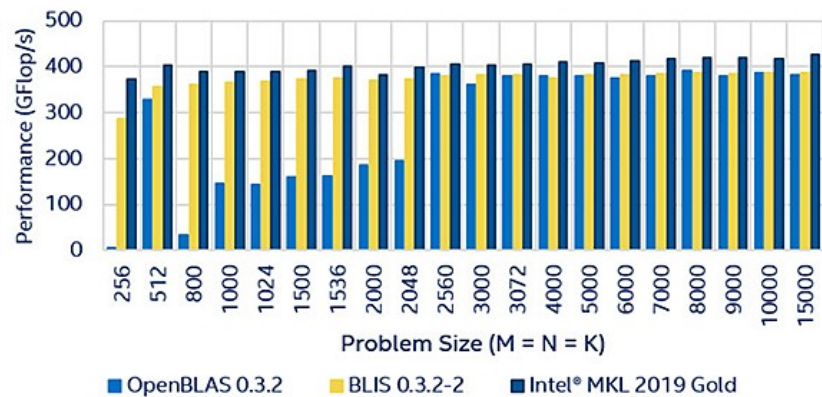
- For example, Intel sell a BLAS/LAPACK library they call they Intel MKL and they market it aggressively
- Dgemm: BLAS routine does dense double matmatmult

## Intel® Math Kernel Library 2019 Gold vs Competitors on Intel® Core™ i5 Processor

Intel® MKL 2019 Gold vs Competitors DGEMM on 4 Threads



Intel® MKL 2019 Gold vs Competitors SGEMM on 4 Threads



Performance results are based on testing as of July 10, 2018 and may not reflect all publicly available security updates. See configuration disclosure for details. No product can be absolutely secure. Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information, see [Performance Benchmark Test Disclosure](#). Testing by Intel as of July 10, 2018. Configuration: Intel® Core™ i5-7600 CPU @ 3.50GHz 65W 64GB DDR4-2400

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice. [Notice revision #20110804](#). For more complete information about compiler optimizations, see our [Optimization Notice](#).

# Homework

- Implement a “cache aware” matrix-vector product
- Change the Matrix class to use column-major storage (and change your matmatmult routines, etc to respect this)
- Overload the + operator for our Matrix class
- ADVANCED – Build an LU factorisation for your Matrix class

VERY ADVANCED – Build an LU factorisation with partial pivoting

SUPER ADVANCED – Read

[http://www.metal.agh.edu.pl/~banas/OWW/sc11\\_dgemm.pdf](http://www.metal.agh.edu.pl/~banas/OWW/sc11_dgemm.pdf)

for an overview of the complexity of building a performant dgemm routine on modern hardware