Lecture 6: Scientific computing in C++

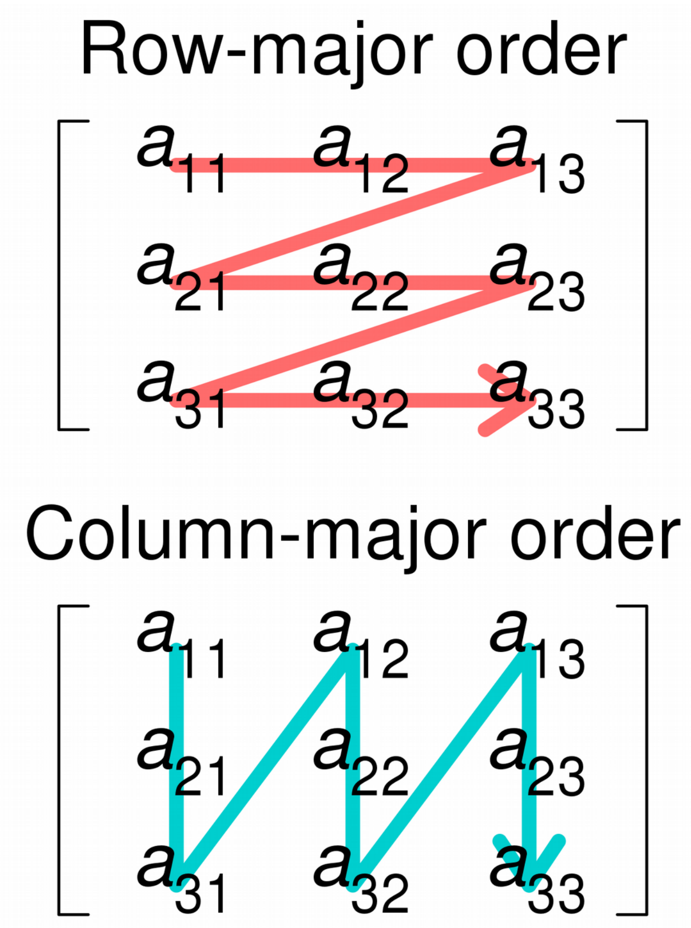
- A review of the work you completed in ACSE-3, specifically lecture 3 would be helpful over the next few weeks
- We’re going to be writing versions of algorithms you should know
- But we’re going to spend a lot of time thinking about how we might write them in C++
- C++ standard is big!
- There are 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)

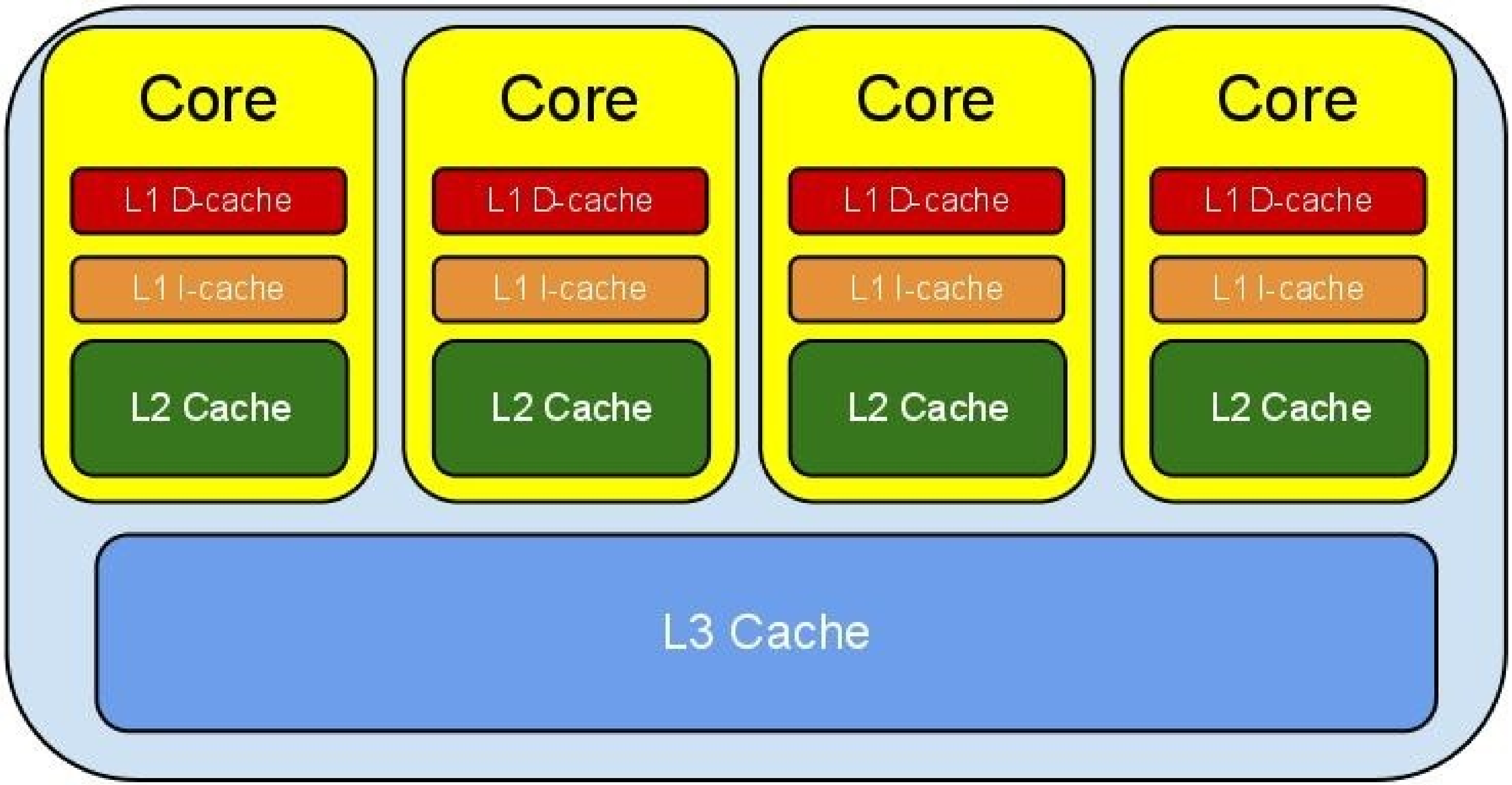


Time to write some code!

- Continue writing the new Matrix class

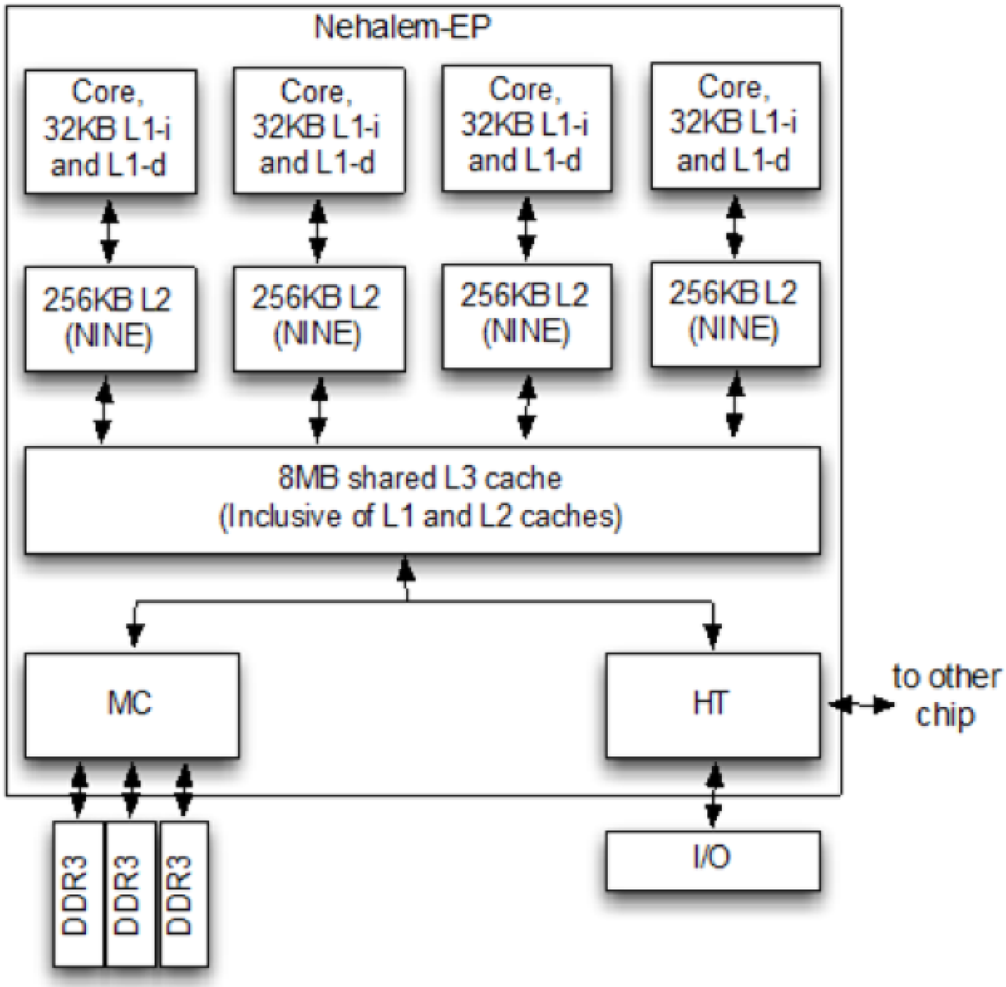
Column/row major ordering

- 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

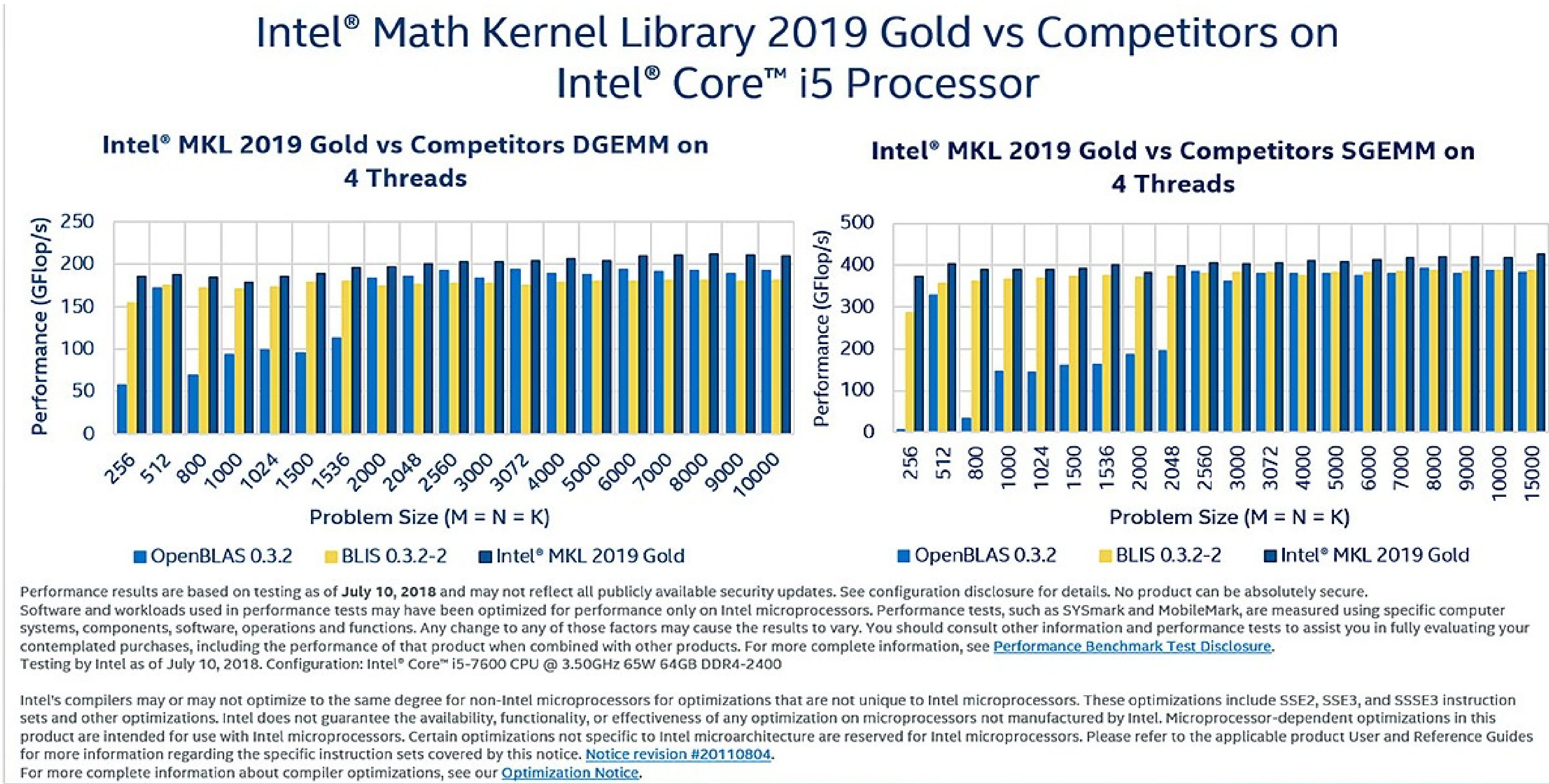
[in]	**N**	N is INTEGER number of elements in input vector(s)
[in]	**DA**	DA is DOUBLE PRECISION On entry, DA specifies the scalar alpha.
[in]	**DX**	DX is DOUBLE PRECISION array, dimension (1 + (N - 1)*abs(INCX))

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



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