High Performance Computing

Edmond Chow
School of Computational Science and Engineering
Georgia Institute of Technology

Why this course?

- Almost all computers today use parallelism
- As software developers, we need to think about parallelism *from the start* when we design algorithms and programs
- High performance in many applications is critical: we want to use our hardware as efficiently as possible

Forms of Parallelism

- A commodity cluster computer is composed of multiple nodes, connected via a network
  - Each node is composed of a system board, 1 or more chips, DRAM, coprocessors/accelerators, etc.
- Each chip contains multiple cores
  - Each core has its own L1 cache, but may share higher level caches with other cores
- Each core can
  - Execute multiple instructions simultaneously (instruction level parallelism)
  - Some instructions can execute the same instruction on multiple pieces of data simultaneously (SIMD parallelism)

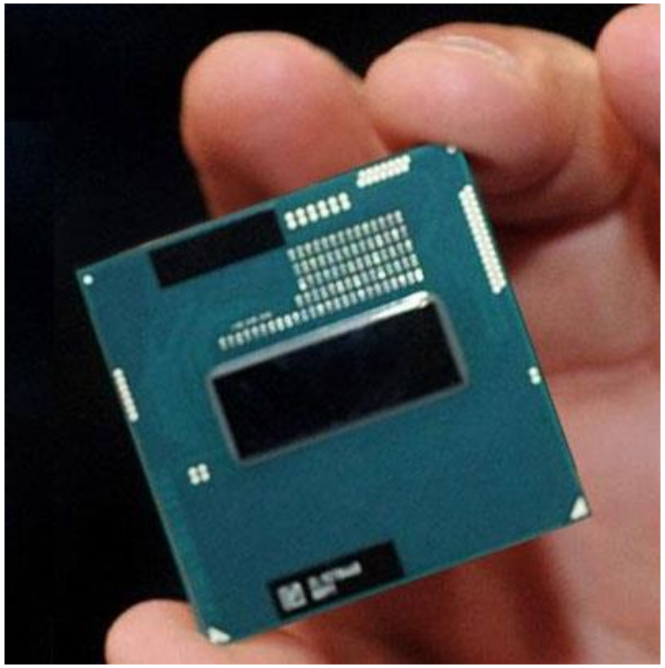
Tianhe-2: One of the fastest supercomputers in the world

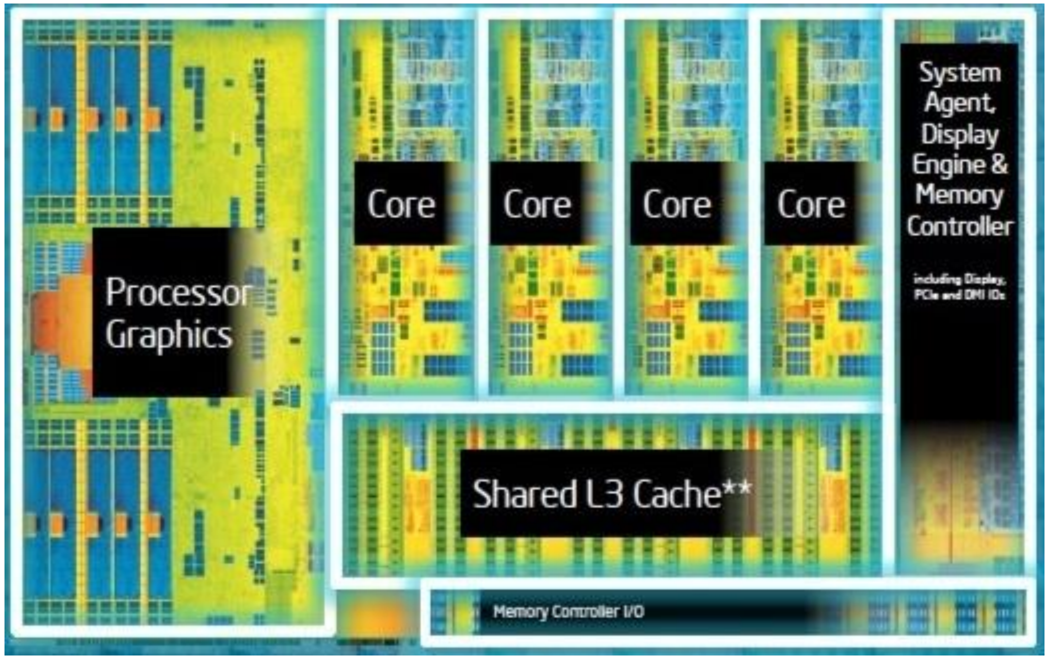
- 16000 nodes
- Each node contains two 12-core CPUs and three 57-core coprocessors
- Each node contains 64 GB DRAM + 3x8 GB DRAM on coprocessors

Tianhe-2

Georgia Tech graduate student, Xing Liu, with Tianhe-2 in China.

Haswell: Recent Intel microarchitecture

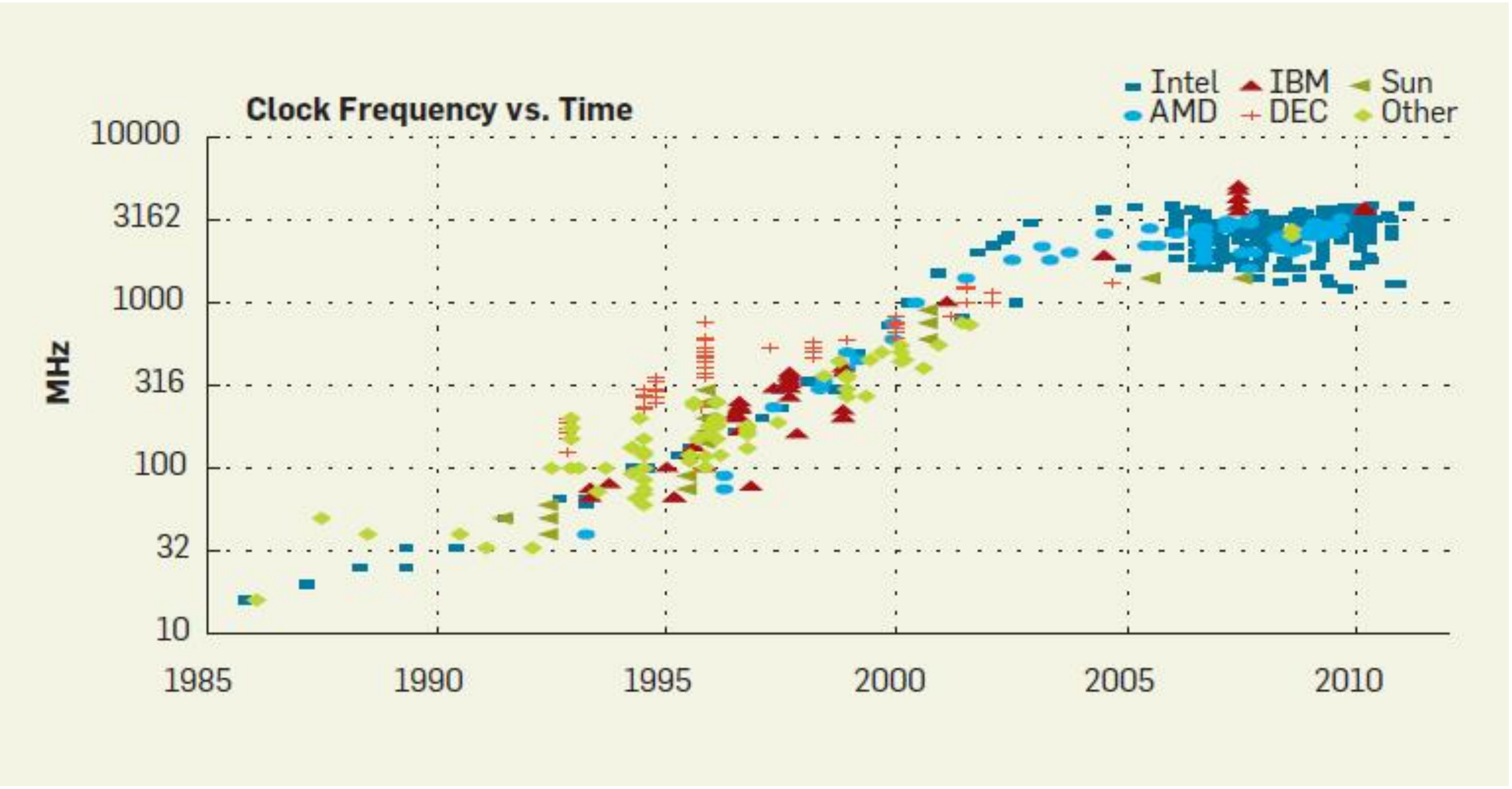



Processor Graphics
Core
Core
Core
Core
System Agent, Display Engine & Memory Controller
including Display, PCIe and DMI IOs
Shared L3 Cache**
Memory Controller I/O

Intel Xeon Microarchitectures

Core (2006) 65 nm, tock	**Penryn** (2007) 45 nm, tick	
Nehalem (2008) 45 nm, tock	**Westmere** (2010) 32 nm, tick	
Sandy Bridge (2011) 32 nm, tock	**Ivy Bridge** (2012) 22 nm, tick	
Haswell (2013) 22 nm, tock	**Broadwell** (2014) 14 nm, tick	
Skylake (2015) 14 nm, tock	**Kaby Lake** 14 nm, optimiz	**Cannonlake** 10 nm, process
Icelake 10 nm, architecture	**Tigerlake** 10 nm, optimiz	**???**

Why multiple cores?



Coprocessors and Accelerators

Intel Xeon Phi

NVIDIA K20 GPU

Important concept: shared memory vs distributed memory

- Shared memory: multiple threads of a process run on a single node
  - All the data can be accessed by all the threads in the regular way, i.e., serial or sequential program
  - Need mechanisms to coordinate cooperation of the threads (e.g. locks)
  - Minor point: Data may be physically distributed (multiple nodes), but software is used to make it look "logically shared"

- Distributed memory: multiple processes run on multiple nodes (e.g., one node per process)
  - Processes only have access to data on the node
  - Use a "communication library" to access data on other nodes
  - Minor point: Processes themselves can have multiple threads
  - Minor point: Could run multiple processes per node

Course Topics

- Aspects of computer architecture and networks
- Parallel algorithms
  - how to partition a problem for parallel computing
- Performance modeling
  - computation and communication
- Parallel scientific applications
  - molecular simulations, quantum chemistry
- Distributed memory (MPI) programming
- Multithreaded programming (OpenMP, etc.)
- Coprocessor/accelerator programming

Textbooks

- G. Hager and G. Wellein, *Introduction to High Performance Computing for Scientists and Engineers*, CRC Press, 2010
- A. Vladimirov and V. Karpusenko, *Parallel Programming and Optimization with Intel Xeon Phi Coprocessors*, Colfax International, 2014

What you need in order to succeed in this course

- Desire to learn how to make programs run fast
- Curiosity to investigate performance anomalies (required for making programs run fast)
- Engage and participate in class discussions and activities. You also need to bring a laptop computer for many classes.
- Expertise in C/C++ programming
- Familiarity with using the Linux command line
- Not afraid of matrix operations (the bread and butter of high performance computing)

Some Linux concepts you will need

- In addition to ssh, scp, editing (nano), compiling, moving files, /tmp file system……
- Understanding PATH
- Setting environment variables in general, e.g., LD_LIBRARY_PATH
- Writing shell scripts
- Shell startup file, e.g., .bashrc
- Note differences between different shells

Intel Xeon Phi Servers

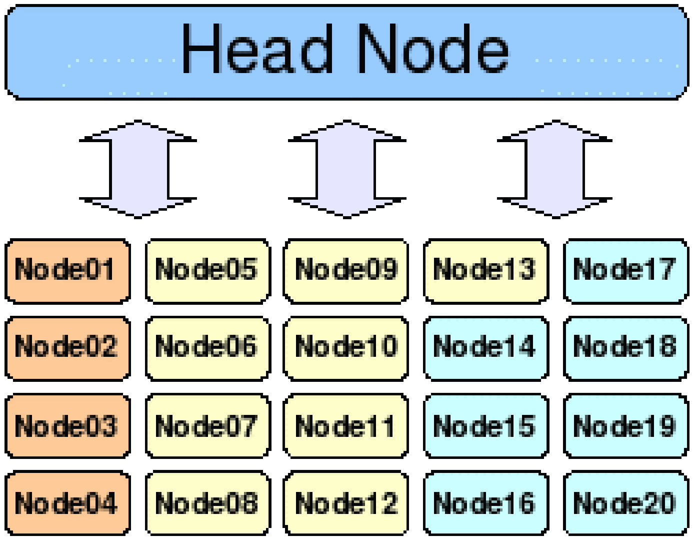
Accounts will be created for you on the following machines:

- joker
  - 8 Intel Xeon Phi cards (KNC)
  - dual 10-core Haswell
- gotham
  - 8 Intel Xeon Phi cards (KNC)
  - dual 16-core Haswell

Jinx cluster

- ssh yourgtid@jinx-login.cc.gatech.edu logs you onto the head node
  - 2 Intel Xeon E5520 (4-core), 12 GB mem
- 30 nodes
  - 24 nodes: 2 Intel Xeon X5650 (6-core), 24 GB mem, 2 GPU
  - 6 nodes: 2 Intel Xeon X5570 (4-core), 48 GB mem
- Commands to know:

```
– qstat -a
– qsub -I -q class
         -l nodes=1:sixcore
         -l walltime=30:00
```



jinx cluster (in CCB 247)

qstat command

- qstat % see just your jobs
- qstat –a
- qstat –f <jobid>
- qstat –q % list queues and their limits
- pbsnodes % check which nodes are down

Requesting node attributes

- qsub –l nodes=1 % request 1 node
- qsub –l nodes=1:sixcore
- qsub –l nodes=jinx1 % request specific node
- jinx node attributes
  - sixcore, fourcore, bigmem, gpu, m2070, m2090

Interactive jobs vs. Batch jobs

- Usual practice at supercomputer centers is to submit a batch script
- Interactive jobs are useful for debugging
- Cluster etiquette
  - Log out of interactive jobs when you are not using them
  - Use batch jobs if possible

Interactive jobs

- If you are using multithreaded parallelism, you will usually want to request an entire node for yourself; otherwise you can share a node with others
- When you are allocated a node, you can also ssh into that node

git revision control

- We are going to use Georgia Tech's github installation to manage class materials and allow you to collaborate. Please create an account for yourself at:

- https://github.gatech.edu/   (Use your GT credentials)

- Clone (fork) the repo called **HPC-course**

- For more info: https://support.cc.gatech.edu/support-tools/faq/what-gt-github-enterprise

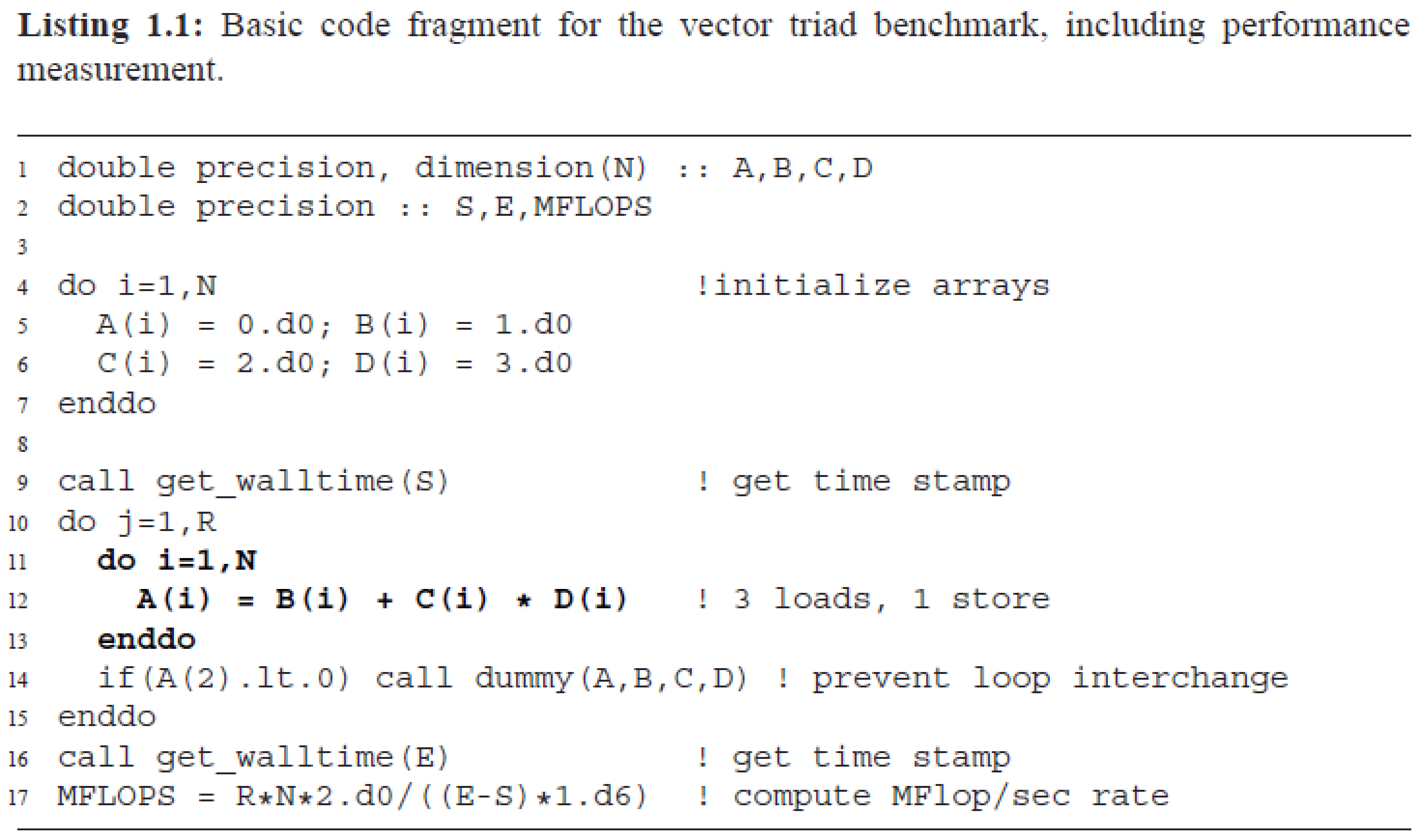
Why Performance Modeling?

- Use models to choose among different implementation options
- Use models to determine if you have achieved the best possible performance possible on your hardware
  - Otherwise, you get a timing result, but how do you know if it is any good?

Measuring Performance

- Performance is measured by execution time to accomplish a task
  - If we use n processors, we ideally expect time to be n times smaller
- Also now trendy to measure performance in terms of energy used
  - Which uses less energy: a computation on your laptop, or the same computation on your cell phone? What energy should be counted?

Fortran Example: Note loop R times and call to a dummy function



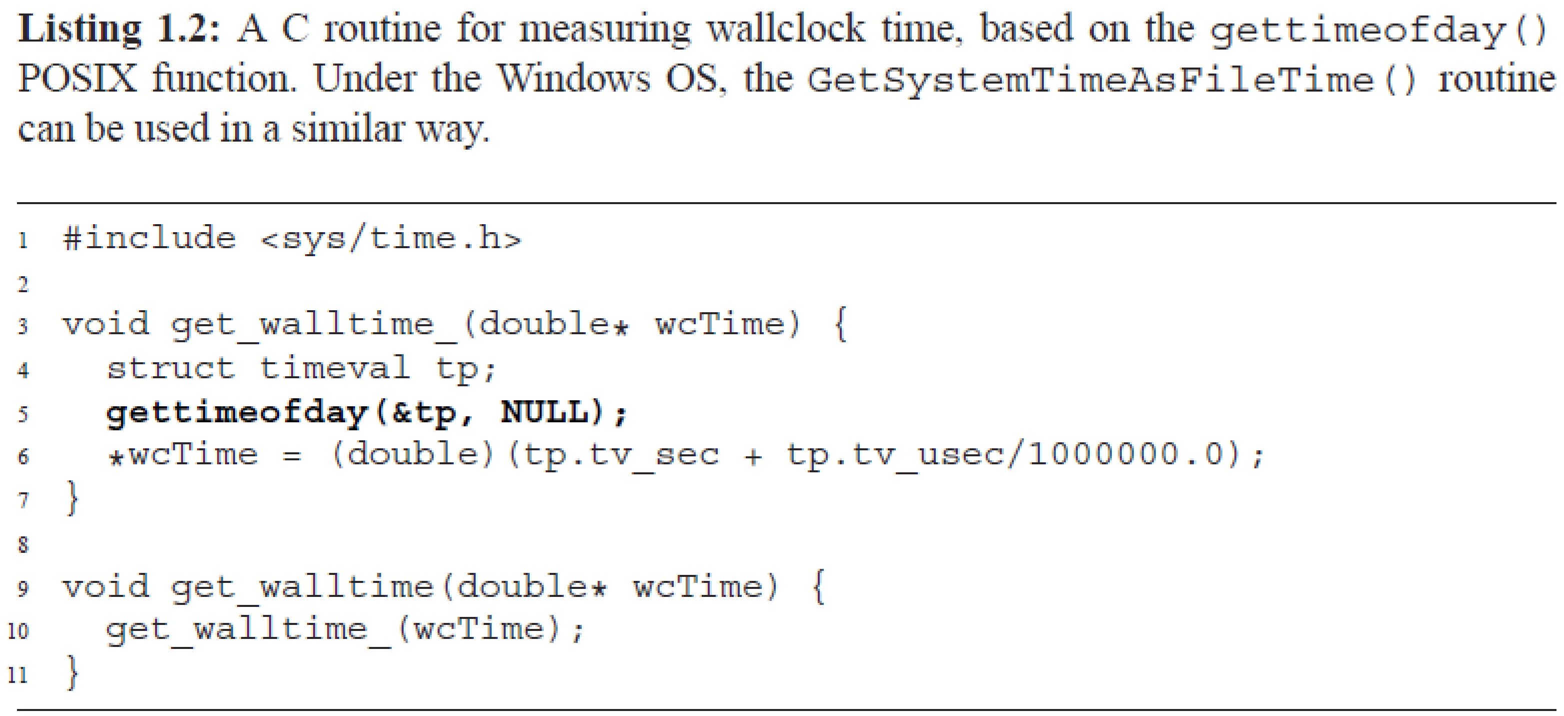
Listing 1.1: Basic code fragment for the vector triad benchmark, including performance measurement.

```fortran
double precision, dimension(N) :: A,B,C,D
double precision :: S,E,MFLOPS

do i=1,N                              !initialize arrays
  A(i) = 0.d0; B(i) = 1.d0
  C(i) = 2.d0; D(i) = 3.d0
enddo

call get_walltime(S)                  ! get time stamp
do j=1,R
  do i=1,N
    A(i) = B(i) + C(i) * D(i)         ! 3 loads, 1 store
  enddo
  if(A(2).lt.0) call dummy(A,B,C,D) ! prevent loop interchange
enddo
call get_walltime(E)                  ! get time stamp
MFLOPS = R*N*2.d0/((E-S)*1.d6)        ! compute MFlop/sec rate
```

get_walltime



Listing 1.2: A C routine for measuring wallclock time, based on the `gettimeofday()` POSIX function. Under the Windows OS, the `GetSystemTimeAsFileTime()` routine can be used in a similar way.

```c
#include <sys/time.h>

void get_walltime_(double* wcTime) {
  struct timeval tp;
  gettimeofday(&tp, NULL);
  *wcTime = (double)(tp.tv_sec + tp.tv_usec/1000000.0);
}

void get_walltime(double* wcTime) {
  get_walltime_(wcTime);
}
```

Tips for measuring execution time

- Can we measure the execution time of

  ```
  a = b + c;
  ```

- What is going on if we measure the time twice and the time is not the same?
  - Should we take the average?
  - Should we take the lowest one?

- What is good practice when measuring execution time?

What takes more time?

- Computing: a=b+c (b and c in registers)
- Reading memory: a=array[1234];

Answer

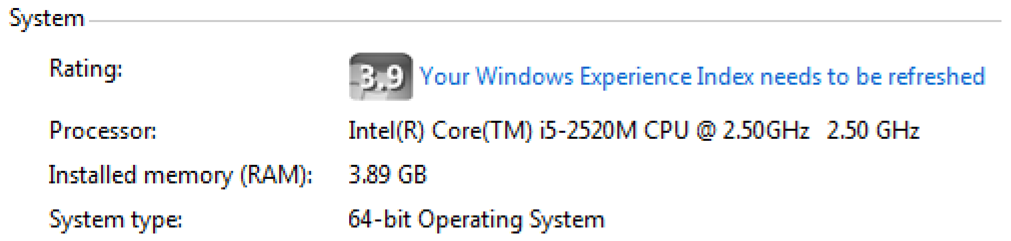
- Reading memory may take 100-1000 times more time (and energy). This is memory latency.
- Overall performance may be memory-latency or memory-bandwidth bound rather than compute-bound.
- In the olden days (1980s), data movement was not more expensive than computation, and performance could be reliably measured by counting "flops" -- floating point operations.

How fast is your computer?

- How many floating point operations can be performed per second (flops/s) by your laptop?

- What is the peak flops/s for Tianhe-2? (CPUs are 2.2 GHz and accelerators are 1 GHz)

Example: How fast is your computer?

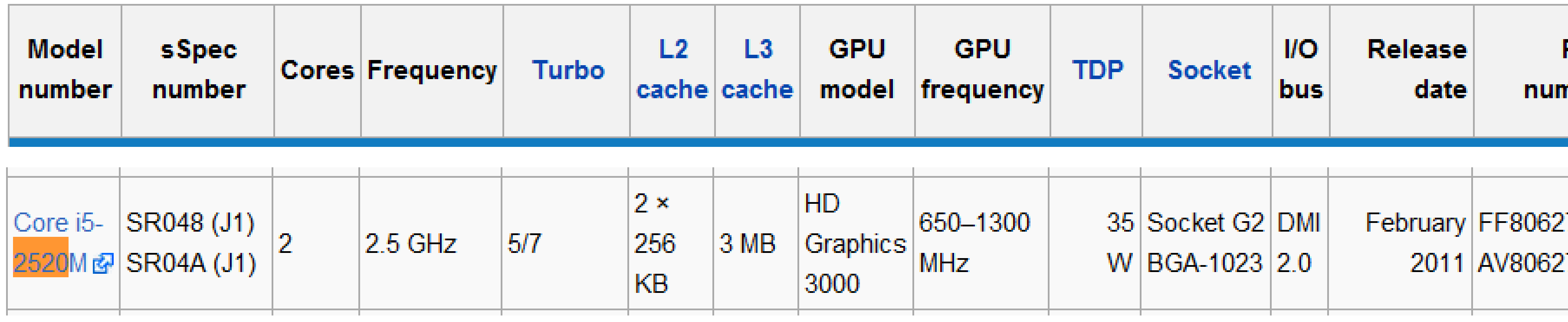

System

Rating:	3.9 Your Windows Experience Index needs to be refreshed
Processor:	Intel(R) Core(TM) i5-2520M CPU @ 2.50GHz 2.50 GHz
Installed memory (RAM):	3.89 GB
System type:	64-bit Operating System

Intel Sandy Bridge i5-2520M at 2.5 GHz (2 cores)

AVX (advanced vector instructions) 256 bits (4 double precision numbers)

No FMA (fused multiply add), 2 FP ports → 40 Gflops/s



Model number	sSpec number	Cores	Frequency	Turbo	L2 cache	L3 cache	GPU model	GPU frequency	TDP	Socket	I/O bus	Release date	F num
Core i5-2520M	SR048 (J1) SR04A (J1)	2	2.5 GHz	5/7	2 × 256 KB	3 MB	HD Graphics 3000	650–1300 MHz	35 W	Socket G2 BGA-1023	DMI 2.0	February 2011	FF8062 AV8062

Data access can be the bottleneck

- Data access across nodes and to main memory
- Data access time measured in terms of bandwidth and latency

- How fast can the CPU read main memory?
  - Stream benchmark for bandwidth http://www.cs.virginia.edu/stream
  - Based on measuring the performance of vector operations, such as A = B + C

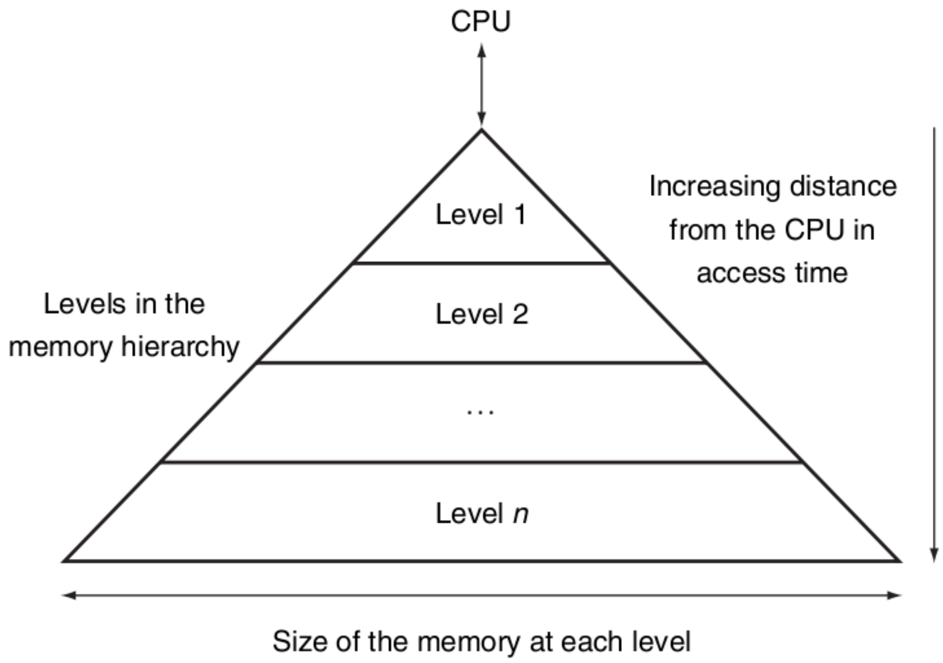
Memory Latency and Bandwidth

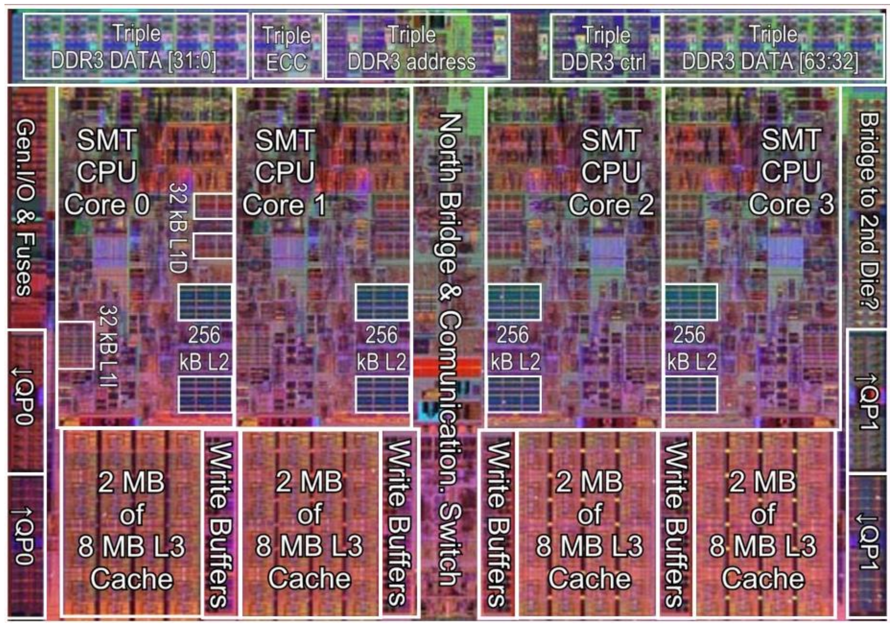
- Latency is the “startup” time for DRAM memory access: hundreds of cycles
- Compare to incremental time for accessing one word in terms of bandwidth: 1/bandwidth = 1/7.5 GB/s = 1.3e-10 sec = 3 cycles
- Latency be neglected for large memory transfers, but dominates memory access time for short memory transfers
- time = latency + length/bandwidth

Two ways to handle high memory latency and low memory bandwidth

- Cache memory
- Multithreading

Memory Hierarchy and Cache


CPU
Level 1
Level 2
...
Level n
Increasing distance from the CPU in access time
Levels in the memory hierarchy
Size of the memory at each level



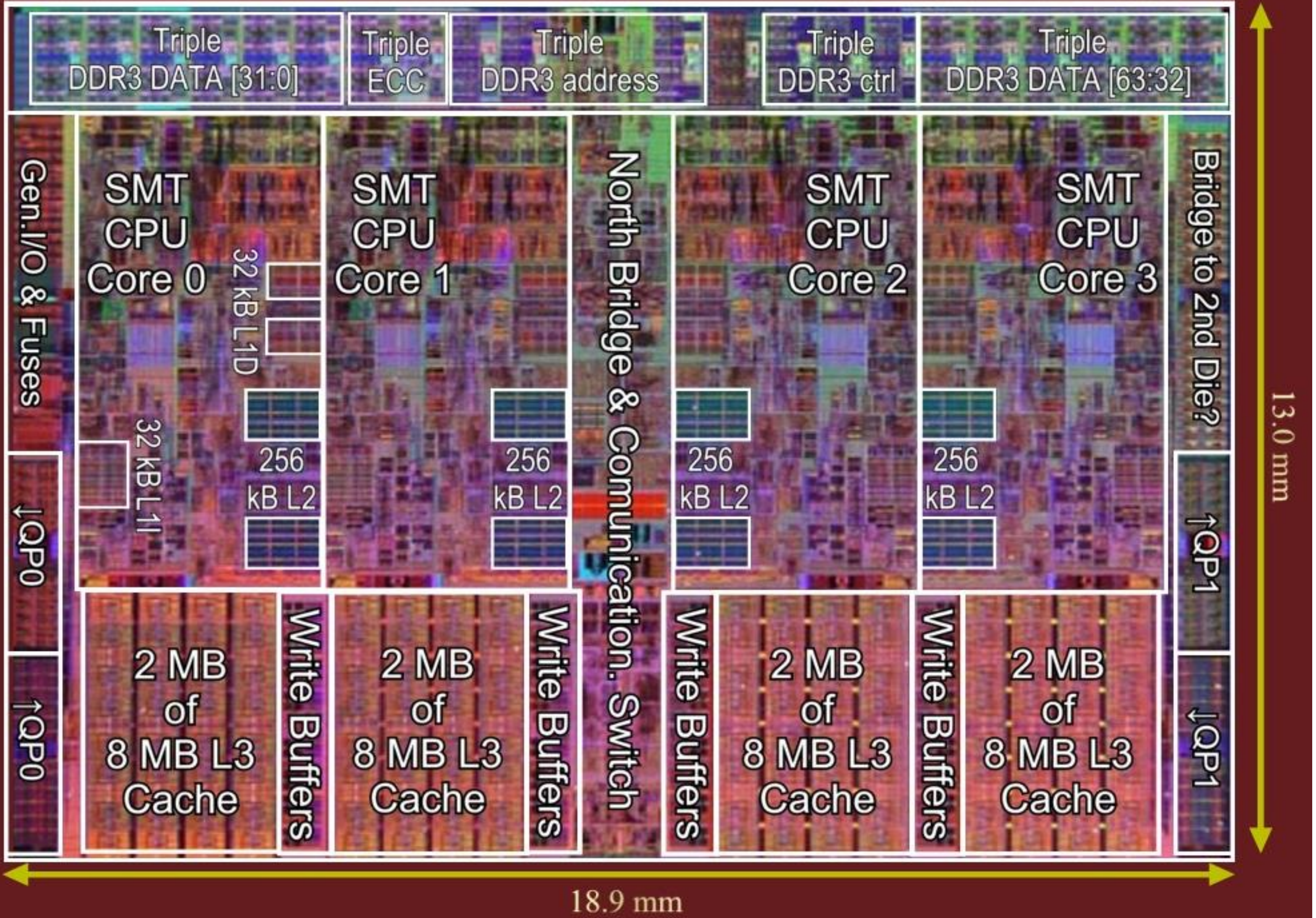
- Based on idea of spatial and temporal data locality
- Cache hit: data found in cache
- Cache miss: data not found in cache and must be copied from a lower level
- Compulsory miss: first reference miss
- Capacity miss: cache runs out of room for new data
- Conflict miss: many data items map to same location in cache

- Registers, latency 1 cycle (0.376 ns)
- L1 32 kB, latency 3 cycles
- L2 256 kB, latency 10 cycles
- L3 8 MB shared, latency 40 cycles
- DRAM, latency hundreds of cycles
- Disk, latency millions cycles

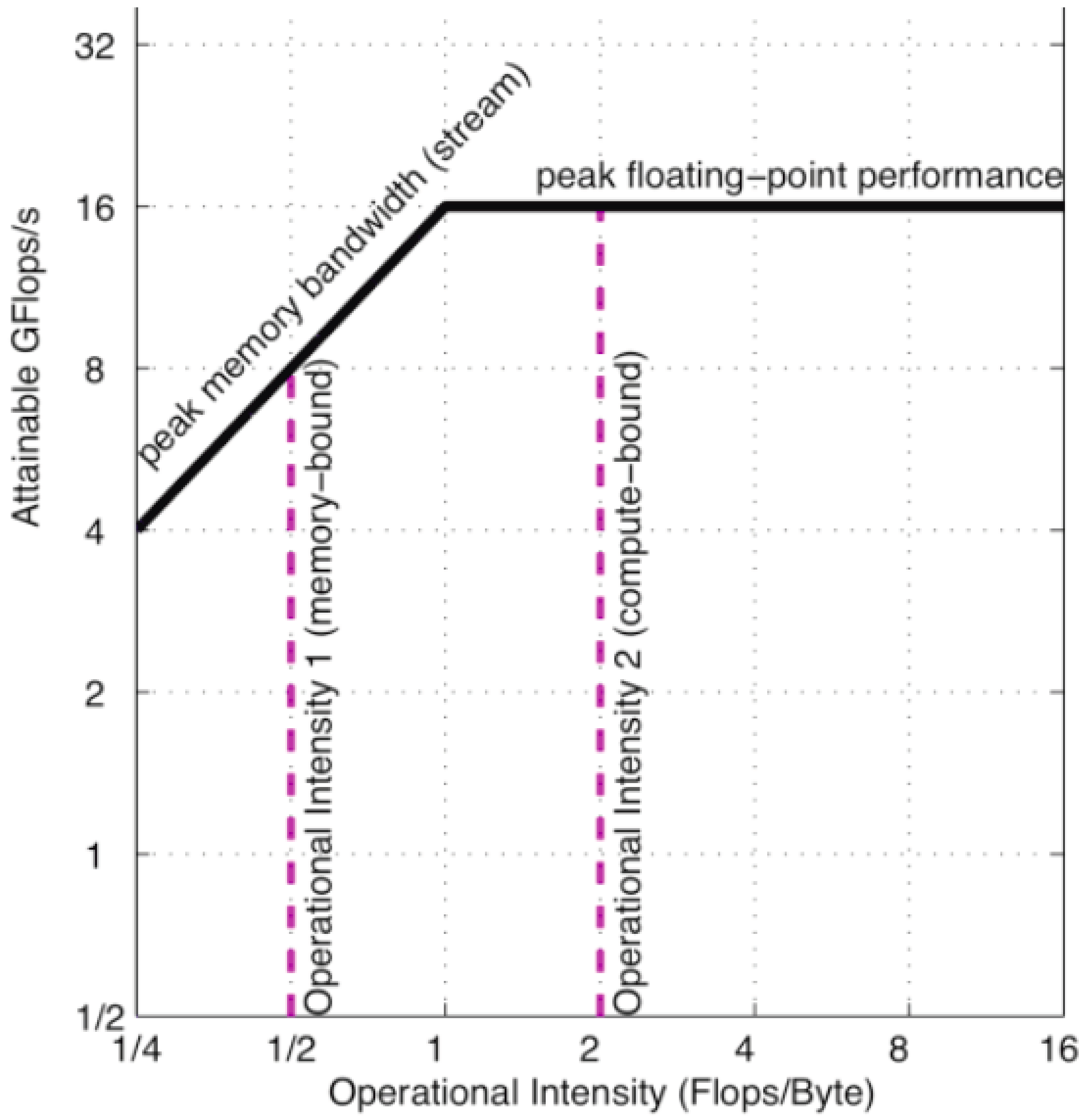
Intel Quad Core Nehalem

731 million transistors --- 8 MB L3 plus 4 x 256 kB L2 --- 3x64bit DDR3 bus
2x Quick path I/O --- Single core size: ~24.4 mm2 (excl L2)
L2 cache tiles: 7.1 mm2 / MB, L3 cache tiles: 5.7 mm2 / MB (excl.tags)

Die size 246 mm2 (incl. test circ.265 mm2)

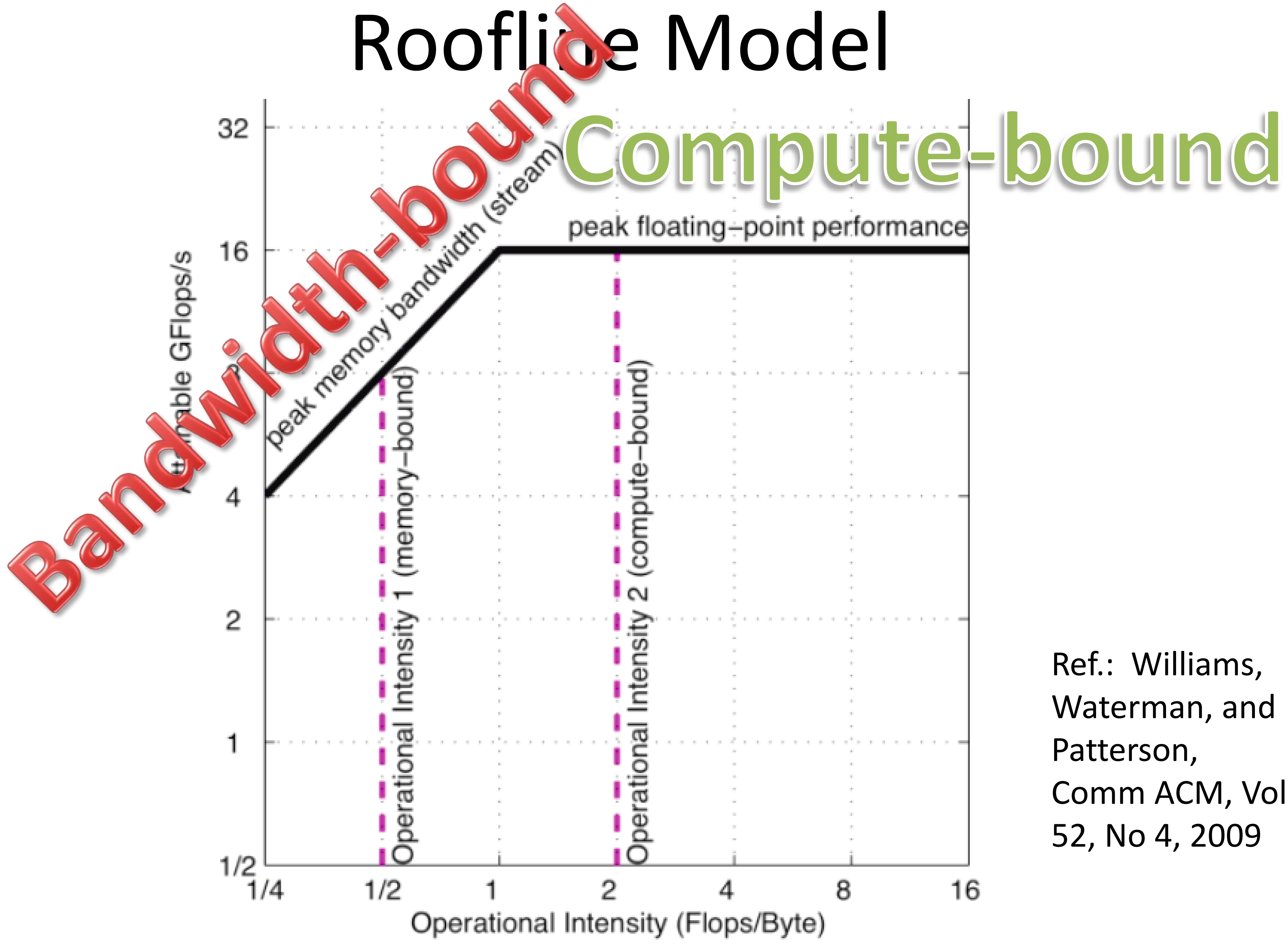


Roofline Model



Ref.: Williams, Waterman, and Patterson, Comm ACM, Vol 52, No 4, 2009

Roofline Model



Ref.: Williams, Waterman, and Patterson, Comm ACM, Vol 52, No 4, 2009