CSC367 Parallel computing

Lecture 1: Introduction

Maryam Mehri Dehnavi
mmehride@cs.toronto.edu

Prepare to adjust to cluster downtimes and be ready to work in teams!

This course uses a Compute cluster called Scinet managed by ComputeCanada. The cluster will have downtimes! Which means:

- We might swap class and lab times if Scinet is down on a lab day!.
- The cluster might go down during assignment due dates! So plan ahead of time and do not leave submissions to last minute!
- You have to work in groups of two. We do not want to overload Scinet or CDF machines!

369 and 367! and APC!

I have never taken 369 and am not taking it this semester: You are probably at a disadvantage compared to your classmates that have taken or are taking 369!

So why isn't 369 a pre-req?! Because most of our top grades from previous years are students that took both courses at the same time!

Our class is a part of the APC online course directed by UC Berkeley with over 25 other schools involved. All members of this course use a similar course project. If our class best speedup for the project beats all other schools we will inform the entire class! We also have access to the XSEDE computing resources. If you receive a good grade and are interested to get access to that cluster for a parallel computing research project, come talk to me near the end of the semester!

Why the ~~Fastest~~ *all* Computers are Parallel Computers (since 2005)

Including laptops and handhelds

© Some of the slides in this lecture are adopted from slides from Charles Leiserson and James demmel.

Tunnel Vision by Experts

"I think there is a world market for maybe five computers."

Thomas Watson, chairman of IBM, 1943.

Slide source: Warfield et al.

Tunnel Vision by Experts

"I think there is a world market for maybe five computers."

Thomas Watson, chairman of IBM, 1943.

"There is no reason for any individual to have a computer in their home"

Ken Olson, president and founder of Digital Equipment Corporation, 1977.

Slide source: Warfield et al.

Tunnel Vision by Experts

"I think there is a world market for maybe five computers."
Thomas Watson, chairman of IBM, 1943.

"There is no reason for any individual to have a computer in their home"
Ken Olson, president and founder of Digital Equipment Corporation, 1977.

"640K [of memory] ought to be enough for anybody."
Bill Gates, chairman of Microsoft,1981.

Slide source: Warfield et al.

Tunnel Vision by Experts

"I think there is a world market for maybe five computers."
Thomas Watson, chairman of IBM, 1943.

"There is no reason for any individual to have a computer in their home"
Ken Olson, president and founder of Digital Equipment Corporation, 1977.

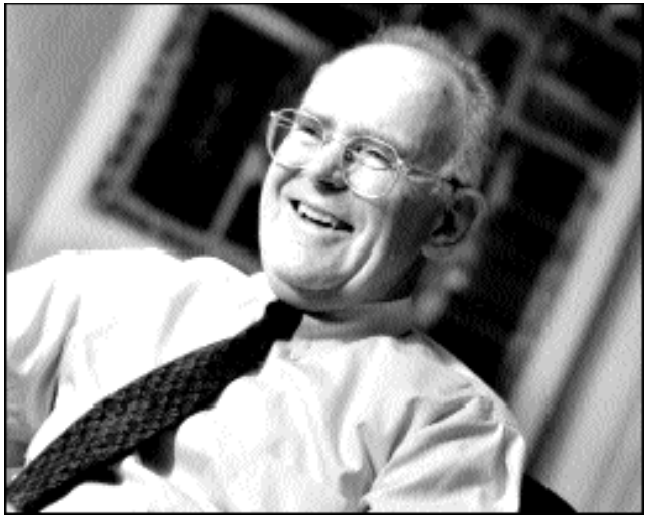
"640K [of memory] ought to be enough for anybody."
Bill Gates, chairman of Microsoft,1981.

On several recent occasions, I have been asked whether parallel computing will soon be relegated to the trash heap reserved for promising technologies that never quite make it."
Ken Kennedy, CRPC Directory, 1994

Slide source: Warfield et al.

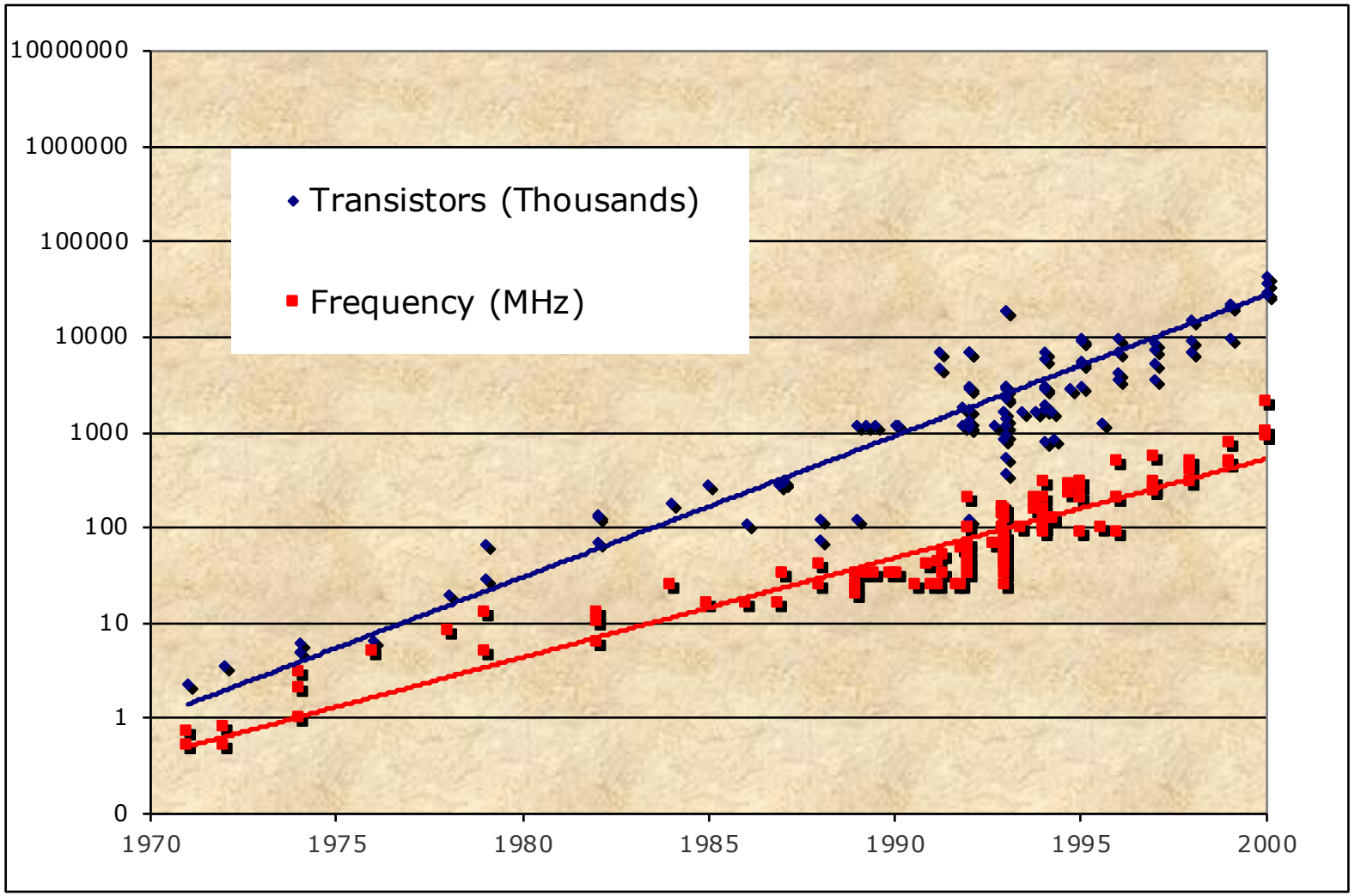
Technology Trends: Microprocessor Capacity

Moore's Law: Gordon Moore predicted in 1965 that the transistor density would double roughly every 18 months.



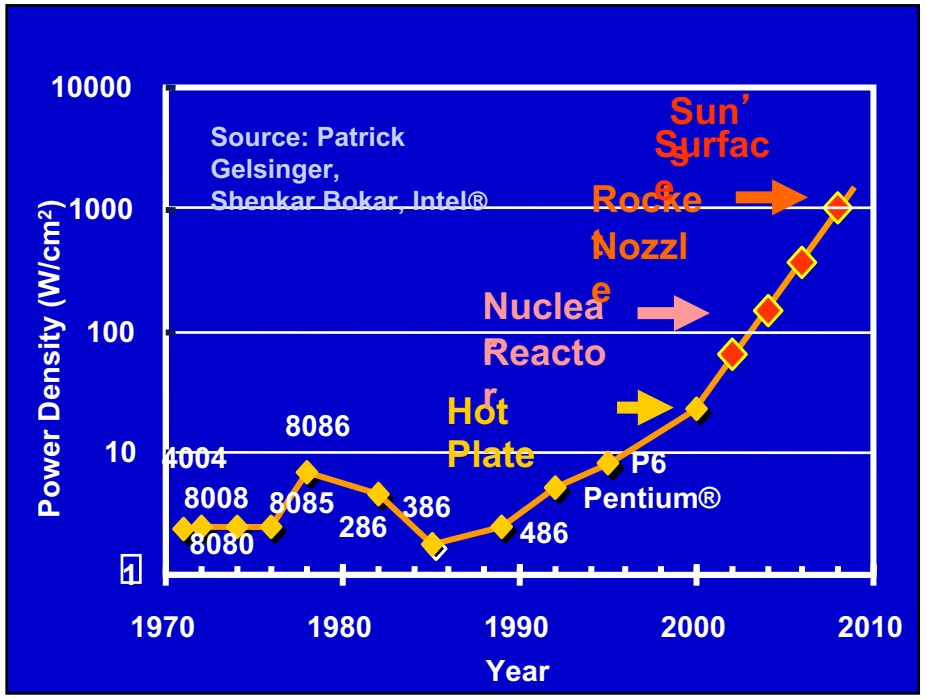
Gordon Moore

Microprocessor Transistors / Clock (1970-2000)



Power Density Limits Serial Performance

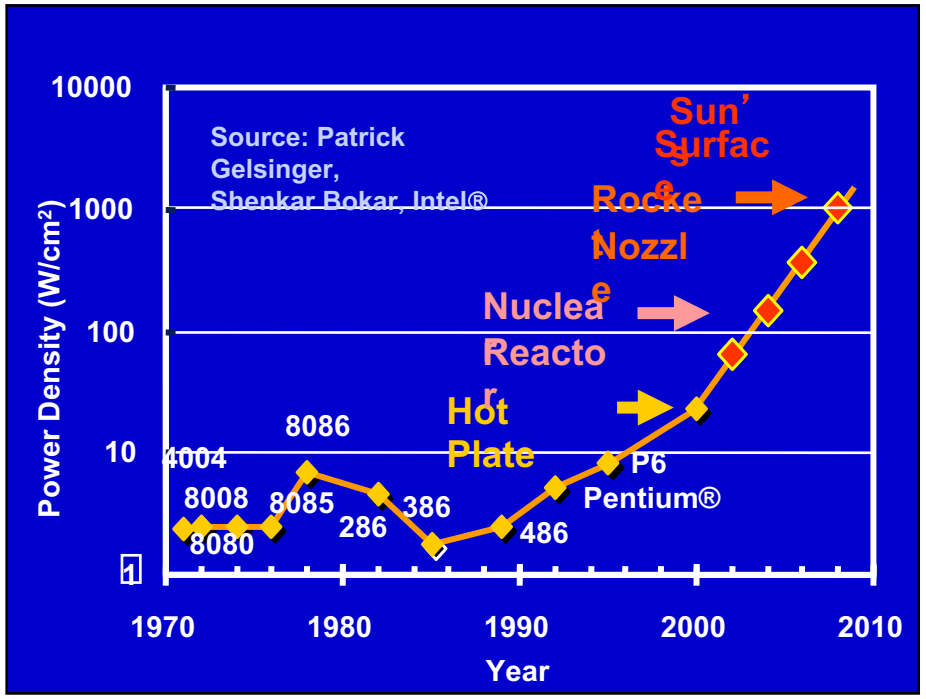
- Concurrent systems are more power efficient
  - Dynamic power is proportional to



Source Demmel

Power Density Limits Serial Performance

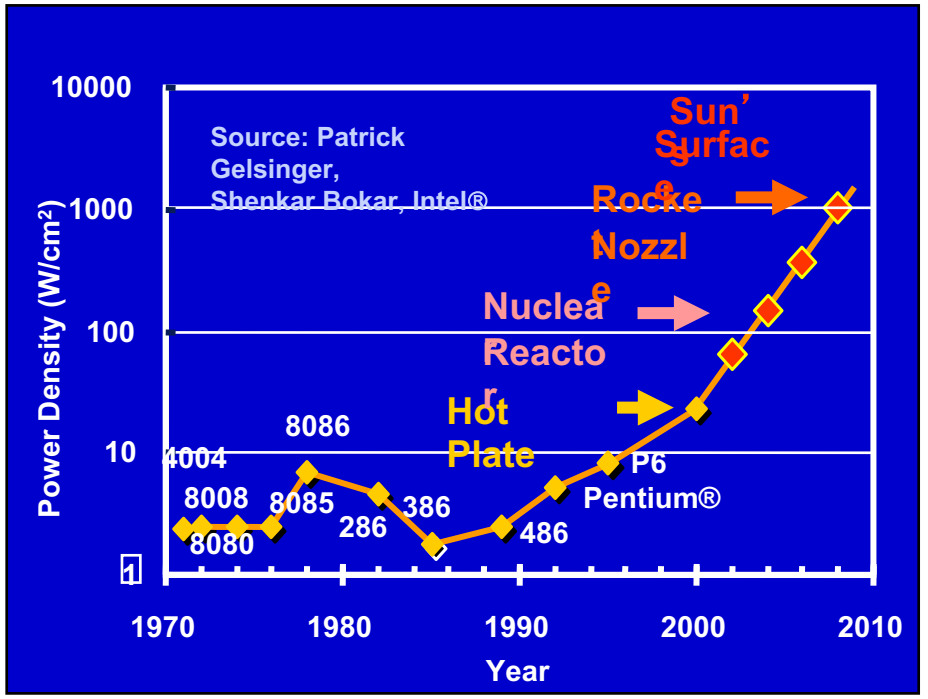
- Concurrent systems are more power efficient
  - Dynamic power is proportional to V^2fC
  - Increasing frequency (f) also increases supply voltage (V) →



Source Demmel

Power Density Limits Serial Performance

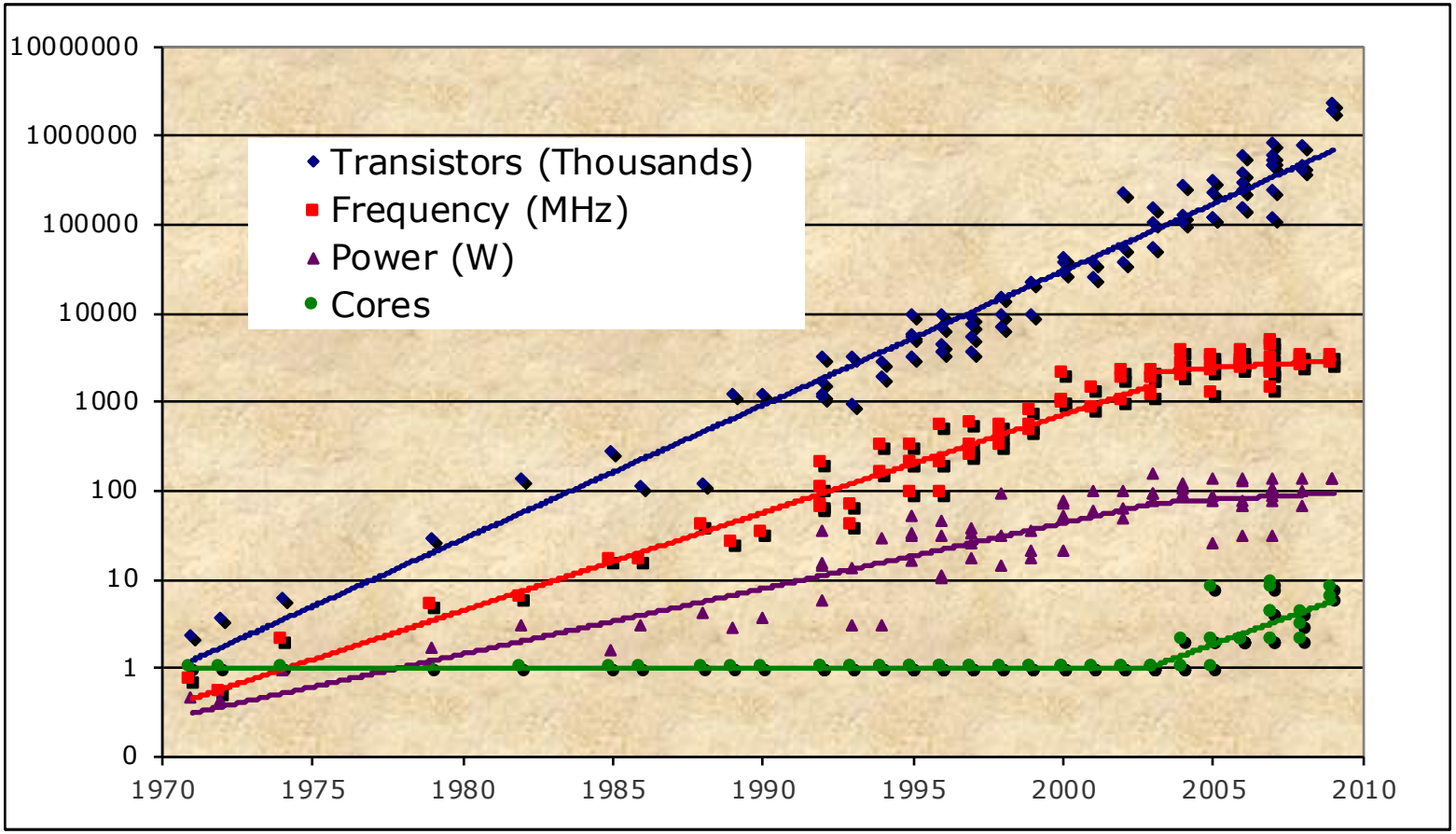
- Concurrent systems are more power efficient
  - Dynamic power is proportional to V^2fC
  - Increasing frequency (f) also increases supply voltage (V) → cubic effect
  - Increasing cores increases capacitance (C) but only linearly
  - Save power by lowering clock speed



Source Demmel

- High performance serial processors waste power
  - Speculation, dynamic dependence checking, etc. burn power
  - Implicit parallelism discovery
- More transistors, but not faster serial processors

Revolution in Processors



- Chip density is continuing increase ~2x every 2 years
- Clock speed is not
- Number of processor cores may double instead
- Power is under control, no longer growing

Some of the World's Fastest Computer

The Top500 List

Units of Measure for HPC

- **High Performance Computing (HPC) units are:**
  - Flop: floating point operation, usually double precision unless noted
  - Flop/s: floating point operations per second
  - Bytes: size of data (a double precision floating point number is 8 bytes)
- **Typical sizes are millions, billions, trillions…**

Kilo	Kflop/s = 10^3 flop/sec	Kbyte = $10^3 \sim 2^{10}$ = 1024 bytes (KiB)
Mega	Mflop/s = 10^6 flop/sec	Mbyte = $10^6 \sim 2^{20}$ bytes (MiB)
Giga	Gflop/s = 10^9 flop/sec	Gbyte = $10^9 \sim 2^{30}$ bytes (GiB)
Tera	Tflop/s = 10^{12} flop/sec	Tbyte = $10^{12} \sim 2^{40}$ bytes (TiB)
Peta	Pflop/s = 10^{15} flop/sec	Pbyte = $10^{15} \sim 2^{50}$ bytes (PiB)
Exa	Eflop/s = 10^{18} flop/sec	Ebyte = $10^{18} \sim 2^{60}$ bytes (EiB)
Zetta	Zflop/s = 10^{21} flop/sec	Zbyte = $10^{21} \sim 2^{70}$ bytes (ZiB)
Yotta	Yflop/s = 10^{24} flop/sec	Ybyte = $10^{24} \sim 2^{80}$ bytes (YiB)

- **Current fastest (public) machines are petaflop systems**
  - Up-to-date list at www.top500.org

The TOP10 of the Top500, November 2017

#	Site	Vendor	Computer	Country	Cores	Rmax [Pflops]	Rpeak (Pflops)	Power [MW]
1	National Supercomputing Center in Wuxi	NRCPC	Sunway TaihuLight NRPC Sunway SW26010, 260C 1.45GHz	China	10,649,600	93.0	125.4	15.4
2	National University of Defense Technology	NUDT	Tianhe-2 NUDT TH-IVB-FEP, Xeon 12C 2.2GHz, IntelXeon Phi	China	3,120,000	33.9	54.9	17.8
3	Swiss National Supercomputing Centre (CSCS)	Cray	Piz Daint Cray XC50, Xeon E5 12C 2.6GHz, Aries, NVIDIA Tesla P100	Switzerland	361,760	19.6	25.3	2.27
4	Japan Agency for Marine-Earth Science and Technology	Exa-Scaler	Gyoukou ZettaScaler-2.2 HPC System, Xeon 16C 1.3GHz, IB-EDR, PEZY-SC2 700Mhz	Japan	19,860,000	19.1	28.2	1.35
5	Oak Ridge National Laboratory	Cray	Titan Cray XK7, Opteron 16C 2.2GHz, Gemini, NVIDIA K20x	USA	560,640	17.6	27.1	8.21
6	Lawrence Livermore National Laboratory	IBM	Sequoia BlueGene/Q, Power BQC 16C 1.6GHz, Custom	USA	1,572,864	17.2	20.1	7.89
7	Los Alamos NL / Sandia NL	Cray	Trinity Cray XC40, Intel Xeon Phi 7250 68C 1.4GHz, Aries	USA	979,968	14.1	23.9	3.84
8	Lawrence Berkeley National Laboratory / NERSC	Cray	Cori Cray XC40, Intel Xeons Phi 7250 68C 1.4 GHz, Aries	USA	622,336	14.0	27.9	3.94
9	JCAHPC Joint Center for Advanced HPC	Fujitsu	Oakforest-PACS PRIMERGY CX1640 M1, Intel Xeons Phi 7250 68C 1.4 GHz, OmniPath	Japan	556,104	13.6	24.9	2.72

SciNet (Scinet)

We will use SciNet for all non-GPU related tasks in this class!

SciNet is Canada's largest supercomputing center, providing Canadian researchers with computational resources and expertise necessary to perform their research on scales not previously possible in Canada.

Scinet Teach Specs!

Teach Cluster

Installed	(orig Feb 2013), Oct 2018
Operating System	Linux (Centos 7.4)
Number of Nodes	42
Interconnect	Infiniband (QDR)
Ram/Node	64 Gb
Cores/Node	16
Login/Devel Node	teach01 (from `teach.scinet`)
Vendor Compilers	icc/gcc
Queue Submission	slurm

Important Action Item and Notes on Scinet!

You have the read the Intro to Scinet document before coming to the lab. We will let you know if things change!

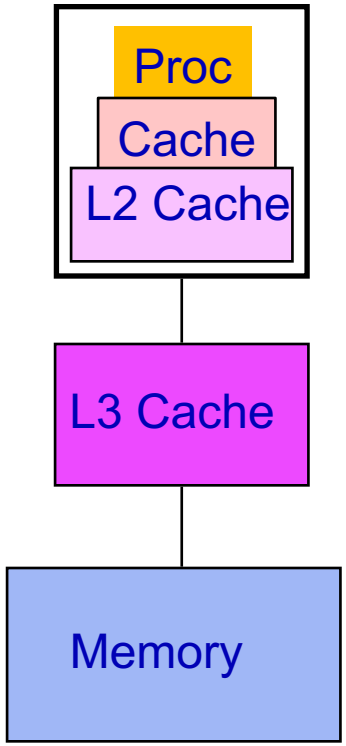
Be prepared for downtimes. We have backup plans but you need to check Quercus and class discussions frequently to be uptodate.

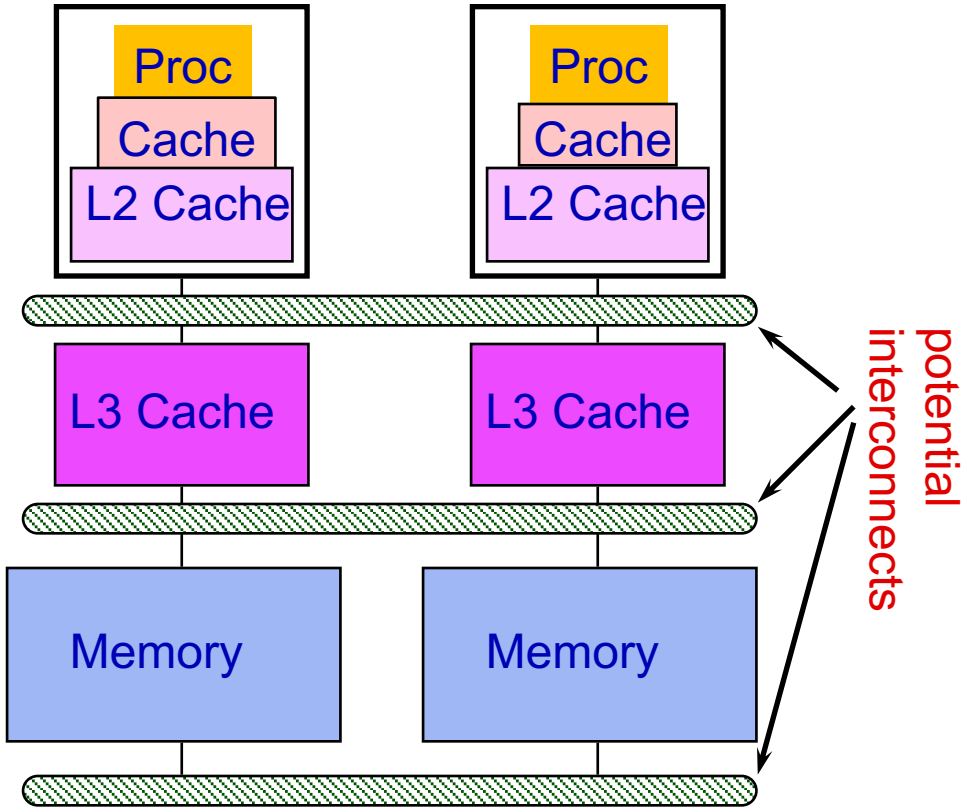
You have to work in groups of two to reduce job loads on Scinet.

If you drop the class at any point, let us know to delete your Scinet account. Using Scinet outside of this class has serious penalties. You might have to pay for compute hours!

What does a parallel computer look like?

Conventional
Storage
Hierarchy





Does performance engineering really matter?!

Example: The Dense Matrix Multiplication

Example from MIT 6.172, © Charles Leiserson

Square-Matrix Multiplication

$$\begin{pmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{nn} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix} \cdot \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nn} \end{pmatrix}$$

$C \qquad A \qquad B$

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

Assume for simplicity that $n = 2^k$.

Intel Haswell Computer System

Feature	Specification
Microarchitecture	Haswell (Intel Xeon E5-2666 v3)
Clock frequency	2.9 GHz
Processor chips	2
Processing cores	9 per processor chip
Hyperthreading	2 way
Floating-point unit	8 double-precision operations, including fused-multiply-add, per core per cycle
Cache-line size	64 B
L1-icache	32 KB private 8-way set associative
L1-dcache	32 KB private 8-way set associative
L2-cache	256 KB private 8-way set associative
L3-cache (LLC)	25 MB shared 20-way set associative
DRAM	60 GB

$$\text{Peak} = (2.9 \times 10^9) \times 2 \times 9 \times 16 = 836 \text{ GFLOPS}$$

Intel Haswell Computer System

Feature	Specification
Microarchitecture	Haswell (Intel Xeon E5-2666 v3)
Clock frequency	2.9 GHz
Processor chips	2
Processing cores	9 per processor chip
Hyperthreading	2 way
Floating-point unit	8 double-precision operations, including fused-multiply-add, per core per cycle
Cache-line size	64 B
L1-icache	32 KB private 8-way set associative
L1-dcache	32 KB private 8-way set associative
L2-cache	256 KB private 8-way set associative
L3-cache (LLC)	25 MB shared 20-way set associative
DRAM	60 GB

$$\text{Peak} = (2.9 \times 10^9) \times 2 \times 9 \times 16 = 836 \text{ GFLOPS}$$

GFlops = (CPU speed in GHz) x (number of CPUs) x (number of CPU cores) x (CPU instruction per cycle)

1. Triply Nested Loops in Python

```
import sys, random
from time import *

n = 4096

A = [[1.0*random.random()
      for row in xrange(n)]
     for col in xrange(n)]
B =


C =


star
for


end

print '%0.6f' % (end - start)
```

Running time
= 21042 seconds
≈ 6 hours

Is this fast?

Back-of-the-envelope calculation

$2n^3 = 2^{37}$ floating-point operations
Running time ≈ 21042 seconds
∴ Python gets 2^{37}/ 21042 = 6.25 MFLOPS
Peak = 836 GFLOPS
Python gets ≈ 0.0075% of peak

2. Let's Try Java

```
mport java.util.Random;

public class mm_java {
  static int n = 4096;
  static double[][] A = new double[n][n];
  static double[][] B = new double[n][n];
  static double[][] C = new double[n][n];

  public static void main(String[] args) {
    Random r = new Random();

    for (int i=0; i<n; i++) {
      for (int j=0; j<n; j++) {
        A[i][j] = r.nextDouble();
        B[i][j] = r.nextDouble();
        C[i][j] = 0;
      }
    }

    long start = System.nanoTime();

    for (int i=0; i<n; i++) {
      for (int j=0; j<n; j++) {
        for (int k=0; k<n; k++) {
          C[i][j] += A[i][k] * B[k][j];
        }
      }
    }

    long stop = System.nanoTime();

    double tdiff = (stop - start) * 1e-9;
    System.out.println(tdiff);
  }
}
```

Running time = 2,738 seconds
≈ 46 minutes
… about 8.8X faster than Python!

```
for (int i=0; i<n; ++i) {
  for (int j=0; j<n; ++j) {
    for (int k=0; k<n; ++k) {
      C[i][j] += A[i][k] * B[k][j];
    }
  }
}
```

3. Why Not C?

```
#include <stdlib.h>
#include <stdio.h>
#include <sys/time.h>
#include <assert.h>

typedef unsigned long long uint64_t;

#define n 4096
double A[n][n];
double B[n][n];
double C[n][n];

float tdiff (struct timeval *start,
             struct timeval *end) {
  return (end->tv_sec-start->tv_sec)
          + 1e-6*(end->tv_usec-start->tv_usec);
}

int main(int argc, const char *argv[]) {
  for (int i=0; i<n; ++i) {
    for (int j=0; j<n; ++j) {
      A[i][j] = (double)rand() / (double)RAND_MAX;
      B[i][j] = (double)rand() / (double)RAND_MAX;
      C[i][j] = 0;
    }
  }

  struct timeval start, end;
  gettimeofday(&start, NULL);

  for (int i=0; i<n; ++i) {
    for (int j=0; j<n; ++j) {
      for (int k=0; k<n; ++k) {
        C[i][j] += A[i][k] * B[k][j];
      }
    }
  }

  gettimeofday(&end, NULL);
  printf("%0.6f\n", tdiff(&start, &end));
  return 0;
}
```

Using the Clang/LLVM compiler

Running time = 1,156 seconds
≈ 19 minutes

… about 2X faster than Java and 18X faster than Python.

```
for (int i=0; i<n; ++i) {
  for (int j=0; j<n; ++j) {
    for (int k=0; k<n; ++k) {
      C[i][j] += A[i][k] * B[k][j];
    }
  }
}
```

Where We Stand So Far

Version	Implementation	Times (s)	GFLOPS	Absolute speedup	Relative speedup	Fraction of peak
1	Python	21,041,67	0.007	1	1	0.00%
2	Java	2,387.32	0.058	9	8.81	0.00%
3	C	1,155.77	0.119	18	2.07	0.01%

Why is Python so slow and C so fast?

- Python is interpreted: Dynamic interpretation at the cost of performance.
- Java is compiled to byte-code, which is then interpreted and just-in-time (JIT) compiled. Frequent statements get compiled to machine code.
- C is compiled directly to machine code.

Loop order i,j,k

The order of the loops can be changed without changing correctness.

```
for (int i=0; i<n; ++i) {
  for (int j=0; j<n; ++j) {
    for (int k=0; k<n; ++k) {
      C[i][j] += A[i][k] * B[k][j];
    }
  }
}
```

Interchange these two loops.

Loop order i,k,j

The order of the loops can be changed without changing correctness.

```
for (int i=0; i<n; ++i) {
  for (int k=0; k<n; ++k) {
    for (int j=0; j<n; ++j) {
      C[i][j] += A[i][k] * B[k][j];
    }
  }
}
```

Does changing the loop order help with performance?

Performance of Different Orders

Loop order	Running time (s)
`i, j, k`	1155.77
`i, k, j`	177.68

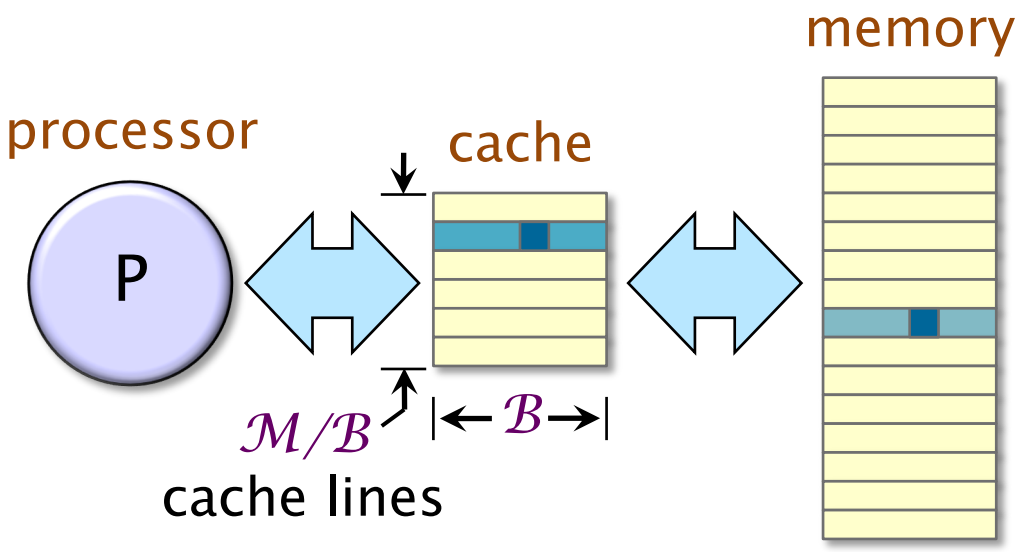
Loop order affects running time by a factor of 15!

What's going on!?

Caches

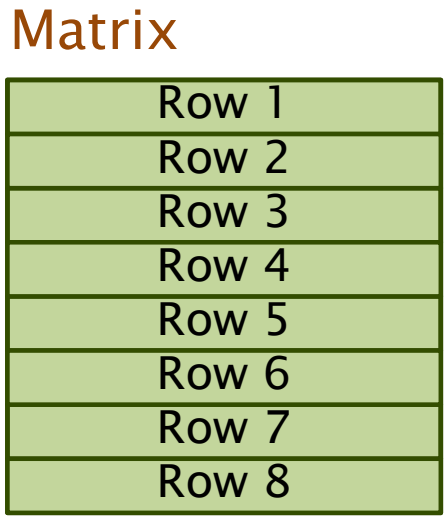
Each processor reads and writes from main memory in contiguous blocks, called cache lines.

- Previously accessed cache lines are stored in a smaller memory called the *cache*.
- If the data required by the processor resides in cache we get a *cache hit*, fast!
- Data accesses that are not in the cache lead to a cache miss, slow!

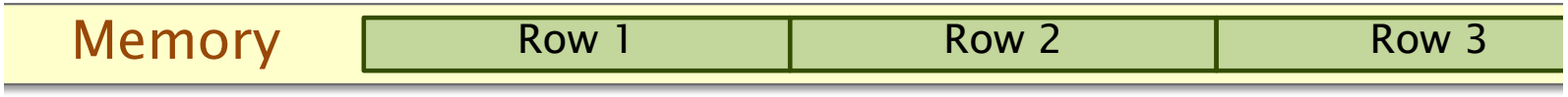


Row-Major Matrix Data Layout

In this example matrix-multiplication code, the matrices are stored in memory in *row-major order*.



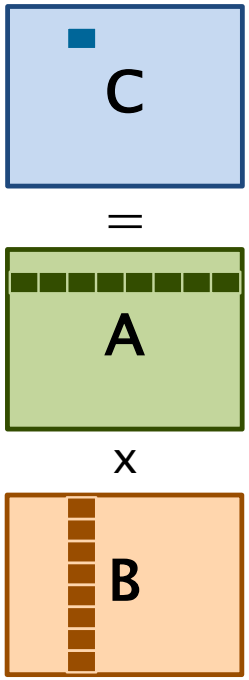
What does this layout imply about the performance of different loop orders?



Data Access Pattern for i,j,k

```
for (int i = 0; i < n; ++i)
  for (int j = 0; j < n; ++j)
    for (int k = 0; k < n; ++k)
      C[i][j] += A[i][k] * B[k][j];
```

Running time:
1155.77s

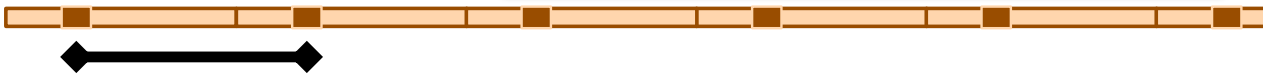


In-memory layout

Excellent spatial locality

Good spatial locality

Poor spatial locality

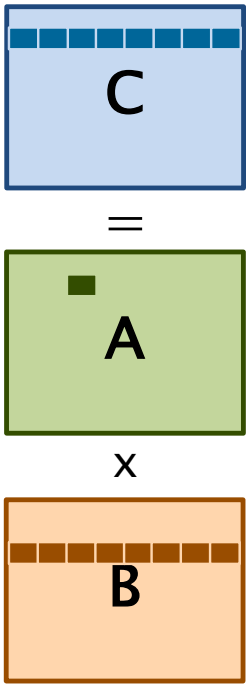


4096 elements apart

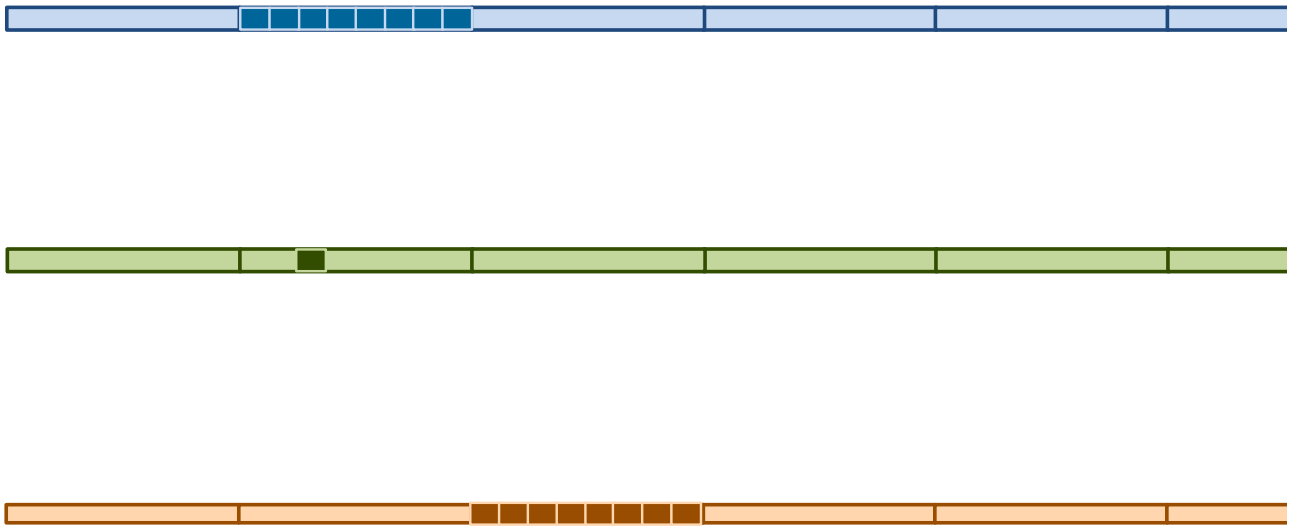
Data Access Pattern for i,k,j

```
for (int i = 0; i < n; ++i)
  for (int k = 0; k < n; ++k)
    for (int j = 0; j < n; ++j)
      C[i][j] += A[i][k] * B[k][j];
```

Running time:
177.68s



In-memory layout



Profiling

We can use valgrind to see the effects of different implementations.

```
$ valgrind --tool=cachegrind ./mm
```

Loop order	Running time (s)	Last-level-cache miss rate
i, j, k	1155.77	7.7%
i, k, j	177.68	1%

Profiling

We can use valgrind to see the effects of different implementations.

```
$ valgrind --tool=cachegrind ./mm
```

You can also use perf!

Loop order	Running time (s)	Last-level-cache miss rate
i, j, k	1155.77	7.7%
i, k, j	177.68	1%

Version 4: Interchange Loops

Version	Implementation	Times (s)	GFLOPS	Absolute speedup	Relative speedup	Fraction of peak
1	Python	21,041,67	0.007	1	1	0.00%
2	Java	2,387.32	0.058	9	8.81	0.00%
3	C	1,155.77	0.119	18	2.07	0.01%
4	+ interchange loops	177.68	0.774	118	6.50	0.09%

Loop interchange lead to 6.5X speedup.

5. Optimization Switches

Compilers provide a collection of optimization switches.

Opt. level	Meaning	Time (s)
`-O0`	Do not optimize	177.54
`-O1`	Optimize	66.24
`-O2`	Optimize even more	54.63
`-O3`	Optimize yet more	55.58

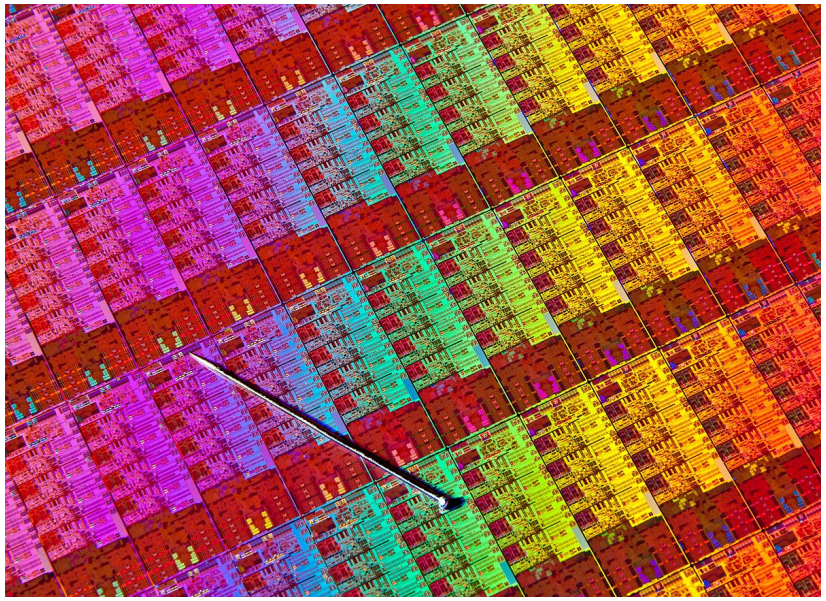
Version 5: Optimization flags

Version	Implementation	Times (s)	GFLOPS	Absolute speedup	Relative speedup	Fraction of peak
1	Python	21,041,67	0.007	1	1	0.00%
2	Java	2,387.32	0.058	9	8.81	0.00%
3	C	1,155.77	0.119	18	2.07	0.01%
4	+ interchange loops	177.68	0.774	118	6.50	0.09%
5	+ optimization flags	54.63	2.516	385	3.25	0. 3%

With simple optimizations to our serial implementation we can achieve 0.3% of the peak machine performance.

6. Parallel Loops

We're running on only one of our 18 cores, leaving 17 idle. Let's use all of them!



Intel Haswell E5 has 9 cores per chip. With 2 processor chips we have 18 cores.

Parallelize the Code!

Replace *with #pragma omp parallel for* or *clik_for* or any other parallel programming model you like.

```
for (int i=0; i<n; ++i) {
  for (int k=0; k<n; ++k) {
    for (int j=0; j<n; ++j) {
      C[i][j] += A[i][k] * B[k][j];
    }
  }
}
```

Version 6: Parallel Loops

Version	Implementation	Times (s)	GFLOPS	Absolute speedup	Relative speedup	Fraction of peak
1	Python	21,041,67	0.007	1	1	0.00%
2	Java	2,387.32	0.058	9	8.81	0.00%
3	C	1,155.77	0.119	18	2.07	0.01%
4	+ interchange loops	177.68	0.774	118	6.50	0.09%
5	+ optimization flags	54.63	2.516	385	3.25	0. 3%
6	Parallel loops	3.04	45.2	6,921	17.97	5.4%

With parallel loops we get almost 18X speedup on 18 cores. However, not all code is so easy to parallelize!

Why are we still at 5% peak!

Optimize More!

There are a lot of other optimizations we can do. *Tiling*, *vectorization*, *matrix transposition*, *data alignment*, *preprocessing*, and *AVX instructions* are just a few. We will cover some of these optimizations in the upcoming lectures.

Version 7: More Optimizations

Version	Implementation	Times (s)	GFLOPS	Absolute speedup	Relative speedup	Fraction of peak
1	Python	21,041,67	0.007	1	1	0.00%
2	Java	2,387.32	0.058	9	8.81	0.00%
3	C	1,155.77	0.119	18	2.07	0.01%
4	+ interchange loops	177.68	0.774	118	6.50	0.09%
5	+ optimization flags	54.63	2.516	385	3.25	0. 3%
6	Parallel loops	3.04	45.2	6,921	17.97	5.4%
7	+ tiling, vectorization+ AVX +more	0.39	352.4	53,292	1.76	41.67%
8	Intel MKL	0.41	335.2	51,497	0.97	40.01%

Our code and the professionally engineering Intel MKL code reach 40% of peak!

Version 5: Overall Speedup

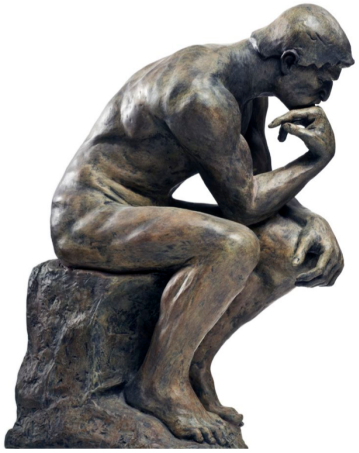
Version	Implementation	Times (s)	GFLOPS	Absolute speedup	Relative speedup	Fraction of peak
1	Python	21,041,67	0.007	1	1	0.00%
2	Java	2,387.32	0.058	9	[illegible]	0.00%
3	C	1,155.77	0.119	18	2.07	0.01%
4	+ interchange loops	177.68	0.774	[illegible]	6.50	0.09%
5	+ optimization flags	54.63	[illegible]	385	3.25	0. 3%
6	Parallel loops	[illegible]	45.2	6,921	17.97	5.4%
7	+ tiling, vectorization, AVX +more	0.39	352.4	53,292	1.76	41.67%
8	Intel MKL	0.41	335.2	51,497	0.97	40.01%

51,497x Improvement!

Our code and the professionally engineering Intel MKL code reach 40% of peak!

The Secret to Writing Fast Code

Think,



code,

run, run, run…

…to test and measure many different implementations

Source: Leiserson

ADMINISTRIVIA

Course Overview

Read the course Syllabus in Quercus very carefully. The class schedule along with all deadlines and policies are posted there.

Lets go over some of them!

Course Overview

Required Prerequisites:
CSC258H1: Computer Organization
CSC209H1: Software Tools and Systems Programming

Recommended Prerequisites:
CSC369H1: Operating Systems

Discussions:
Piazza for questions and discussions.
Quercus for all course material along with important announcements.

Course Overview: Evaluation

Four assignments: 40%

Project: 30%

Labs: 10%

In-tutorial exam: 20% on Friday, Nov 5

If you actively answering Piazza questions and helping your friends, you are a top contributor. At the end of class, 5 of the top contributors (across both sessions) will receive up to 5% bonus marks!

Tentative Topics

- Introduction to memory hierarchies
- Performance engineering on a single processor machines
- Shared memory parallelism with Pthreads and OpenMP
- Distributed memory parallelism with MPI
- Sources of locality and parallelism
- Parallel algorithms
- GPU programming with CUDA
- Introduction to cloud computing

Important Action Item and Notes on Scinet!

You will receive information on your Scinet login before our lab session on Friday Sep 17. You have the read the Intro to Scinet document before coming to the lab.

Be prepared for downtimes. We have backup plans but you need to check Quercus and class discussions frequently to be UpToDate.

You have to work in groups of two to reduce job loads on Scinet.

If you drop the class at any point, let us know to delete your Scinet account. Using Scinet outside of this class has serious penalties. You might have to pay for compute hours!

Start early!

- Scinet puts your job in a queue! So expect longer wait times as you get closer to the deadline.
- We will not provide extensions and with high probably Scinet will be very busy close to deadlines.
- Do not submit concurrent jobs in Scinet if the cluster is busy.
- Do not change the job timeouts in the provided batch files.

Clean code and correct submissions!

- Write well-structured and well-commended code.
- Your submitted code has to compile or you get 0.
- Make sure you only modify the files instructed in the handouts, also submit all relevant files. We can not grade files you never submitted!

Academic Integrity

- We will run plagiarism software on all your submitted code!
- Don’t use code from online resources.
- Don’t post your code in public places and repositories even after the class.
- Don’t search for solutions.
- You and your partner are both responsible for plagiarism found in your code.
- Also do not share lecture recordings.

Assignments and Labs as Partners

- To reduce the load on the Scinet Cluster we will enforce working in groups of two for all labs and assignments. Please do not take this class if you don't feel comfortable working in groups!

- Both partners have to fully understand and should have contributed in the code submitted by their group. You will be asked by TAs to explain your code.