Supercomputing in Plain English

Multicore Madness

Henry Neeman, University of Oklahoma

Director, OU Supercomputing Center for Education & Research (OSCER)
Assistant Vice President, Information Technology – Research Strategy Advisor
Associate Professor, Gallogly College of Engineering
Adjunct Associate Professor, School of Computer Science

Tuesday April 3 2018

This is an experiment!

It’s the nature of these kinds of videoconferences that **FAILURES ARE GUARANTEED TO HAPPEN! NO PROMISES!**

So, please bear with us. Hopefully everything will work out well enough.

If you lose your connection, you can retry the same kind of connection, or try connecting another way.

Remember, if all else fails, you always have the phone bridge to fall back on.

PLEASE MUTE YOURSELF.

PLEASE MUTE YOURSELF.

PLEASE MUTE YOURSELF.

PLEASE MUTE YOURSELF

No matter how you connect, **PLEASE MUTE YOURSELF**, so that we cannot hear you.

At OU, we will turn off the sound on all conferencing technologies.

That way, we won't have problems with **echo cancellation**.

Of course, that means we cannot hear questions.

So for questions, you'll need to send e-mail:

`supercomputinginplainenglish@gmail.com`

PLEASE MUTE YOURSELF.

PLEASE MUTE YOURSELF.

PLEASE MUTE YOURSELF.

Download the Slides Beforehand

Before the start of the session, please download the slides from the Supercomputing in Plain English website:

`http://www.oscer.ou.edu/education/`

That way, if anything goes wrong, you can still follow along with just audio.

PLEASE MUTE YOURSELF.

PLEASE MUTE YOURSELF.

PLEASE MUTE YOURSELF.

Zoom

Go to:

http://zoom.us/j/979158478

Many thanks Eddie Huebsch, OU CIO, for providing this.

PLEASE MUTE YOURSELF.

PLEASE MUTE YOURSELF.

PLEASE MUTE YOURSELF.

YouTube

You can watch from a Windows, MacOS or Linux laptop or an Android or iOS handheld using YouTube.

Go to YouTube via your preferred web browser or app, and then search for:

`Supercomputing InPlainEnglish`

(**`InPlainEnglish`** is all one word.)

Many thanks to Skyler Donahue of OneNet for providing this.

PLEASE MUTE YOURSELF.

PLEASE MUTE YOURSELF.

PLEASE MUTE YOURSELF.

Twitch

You can watch from a Windows, MacOS or Linux laptop or an Android or iOS handheld using Twitch.

Go to:

http://www.twitch.tv/sipe2018

Many thanks to Skyler Donahue of OneNet for providing this.

PLEASE MUTE YOURSELF.

PLEASE MUTE YOURSELF.

PLEASE MUTE YOURSELF.

Wowza #1

You can watch from a Windows, MacOS or Linux laptop using Wowza from the following URL:

http://jwplayer.onenet.net/streams/sipe.html

If that URL fails, then go to:

http://jwplayer.onenet.net/streams/sipebackup.html

Many thanks to Skyler Donahue of OneNet for providing this.

PLEASE MUTE YOURSELF.

PLEASE MUTE YOURSELF.

PLEASE MUTE YOURSELF.

Wowza #2

Wowza has been tested on multiple browsers on each of:

- Windows 10: IE, Firefox, Chrome, Opera, Safari
- MacOS: Safari, Firefox
- Linux: Firefox, Opera

We’ve also successfully tested it via apps on devices with:

- Android
- iOS

Many thanks to Skyler Donahue of OneNet for providing this.

PLEASE MUTE YOURSELF.

PLEASE MUTE YOURSELF.

PLEASE MUTE YOURSELF.

Toll Free Phone Bridge

IF ALL ELSE FAILS, you can use our US TOLL phone bridge:

405-325-6688

684 684 #

NOTE: This is for **US** call-ins **ONLY**.

PLEASE MUTE YOURSELF and use the phone to listen.

Don't worry, we'll call out slide numbers as we go.

Please use the phone bridge **ONLY IF** you cannot connect any other way: the phone bridge can handle only 100 simultaneous connections, and we have over 1000 participants.

Many thanks to OU CIO Eddie Huebsch for providing the phone bridge..

Please Mute Yourself

No matter how you connect, **PLEASE MUTE YOURSELF**, so that we cannot hear you.

(For YouTube, Twitch and Wowza, you don’t need to do that, because the information only goes from us to you, not from you to us.)

At OU, we will turn off the sound on all conferencing technologies.

That way, we won’t have problems with **echo cancellation**.

Of course, that means we cannot hear questions.

So for questions, you’ll need to send e-mail.

PLEASE MUTE YOURSELF.

Questions via E-mail Only

Ask questions by sending e-mail to:

supercomputinginplainenglish@gmail.com

All questions will be read out loud and then answered out loud.

DON’T USE CHAT OR VOICE FOR QUESTIONS!

No one will be monitoring any of the chats, and if we can hear your question, you’re creating an **echo cancellation** problem.

PLEASE MUTE YOURSELF.
PLEASE MUTE YOURSELF.

Onsite: Talent Release Form

If you’re attending onsite, you **MUST** do one of the following:

- complete and sign the Talent Release Form,

OR

- sit behind the cameras (where you can’t be seen) and don’t talk at all.

If you aren’t onsite, then **PLEASE MUTE YOURSELF.**

TENTATIVE Schedule

Tue Jan 23: Storage: What the Heck is Supercomputing?
Tue Jan 30: The Tyranny of the Storage Hierarchy Part I
Tue Feb 6: The Tyranny of the Storage Hierarchy Part II
Tue Feb 13: Instruction Level Parallelism
Tue Feb 20: Stupid Compiler Tricks
Tue Feb 27: Multicore Multithreading
Tue March 6: Distributed Multiprocessing
Tue March 13: **NO SESSION** (Henry business travel)
Tue March 20: **NO SESSION** (OU's Spring Break)
Tue March 27: Applications and Types of Parallelism
Tue Apr 3: Multicore Madness
Tue Apr 10: **NO SESSION** (Henry business travel)
Tue Apr 17: High Throughput Computing
Tue Apr 24: GPGPU: Number Crunching in Your Graphics Card
Tue May 1: Grab Bag: Scientific Libraries, I/O Libraries, Visualization

Thanks for helping!

- OU IT
  - OSCER operations staff (Dave Akin, Patrick Calhoun, Kali McLennan, Jason Speckman, Brett Zimmerman)
  - OSCER Research Computing Facilitators (Jim Ferguson, Horst Severini)
  - Debi Gentis, OSCER Coordinator
  - Kyle Dudgeon, OSCER Manager of Operations
  - Ashish Pai, Managing Director for Research IT Services
  - The OU IT network team
  - OU CIO Eddie Huebsch
- OneNet: Skyler Donahue
- Oklahoma State U: Dana Brunson

This is an experiment!

It’s the nature of these kinds of videoconferences that **FAILURES ARE GUARANTEED TO HAPPEN! NO PROMISES!**

So, please bear with us. Hopefully everything will work out well enough.

If you lose your connection, you can retry the same kind of connection, or try connecting another way.

Remember, if all else fails, you always have the phone bridge to fall back on.

PLEASE MUTE YOURSELF.

PLEASE MUTE YOURSELF.

PLEASE MUTE YOURSELF.

Coming in 2018!

- Coalition for Advancing Digital Research & Education (CADRE) Conference: Apr 17-18 2018 @ Oklahoma State U, Stillwater OK USA
  https://hpcc.okstate.edu/cadre-conference
- Linux Clusters Institute workshops
  http://www.linuxclustersinstitute.org/workshops/
  - Introductory HPC Cluster System Administration: May 14-18 2018 @ U Nebraska, Lincoln NE USA
  - Intermediate HPC Cluster System Administration: Aug 13-17 2018 @ Yale U, New Haven CT USA
- Great Plains Network Annual Meeting: details coming soon
- Advanced Cyberinfrastructure Research & Education Facilitators (ACI-REF) Virtual Residency Aug 5-10 2018, U Oklahoma, Norman OK USA
- PEARC 2018, July 22-27, Pittsburgh PA USA
  https://www.pearc18.pearc.org/
- IEEE Cluster 2018, Sep 10-13, Belfast UK
  https://cluster2018.github.io
- **OKLAHOMA SUPERCOMPUTING SYMPOSIUM 2018, Sep 25-26 2018 @ OU**
- SC18 supercomputing conference, Nov 11-16 2018, Dallas TX USA
  http://sc18.supercomputing.org/

Outline

- The March of Progress
- Multicore/Many-core Basics
- Software Strategies for Multicore/Many-core
- A Concrete Example: Weather Forecasting

The March of Progress

OU’s TeraFLOP Cluster, 2002

10 racks @ 1000 lbs per rack

270 Pentium4 Xeon CPUs,
2.0 GHz, 512 KB L2 cache

270 GB RAM, 400 MHz FSB

8 TB disk

Myrinet2000 Interconnect

100 Mbps Ethernet Interconnect

OS: Red Hat Linux

Peak speed: 1.08 TFLOPs
(1.08 trillion calculations per second)

One of the first Pentium4 clusters!

Aspen Systems inc.

boomer.oscer.ou.edu

What does 1 TFLOPs Look Like?

1 TFLOPs: trillion calculations per second

1997: Room

ASCI RED[14]
Sandia National Lab

2002: Row

boomer.oscer.ou.edu
In service 2002-5: 11 racks

2012: Card

AMD FirePro W9000[15]

NVIDIA Kepler K20[16]

Intel MIC Xeon PHI[17]

CPU Chip 2017

AMD EPYC

Intel Skylake

https://www.top500.org/static/media/uploads/.thumbnails/epyc-vs-xeon.jpg/epyc-vs-xeon-742x382.jpg

Moore’s Law

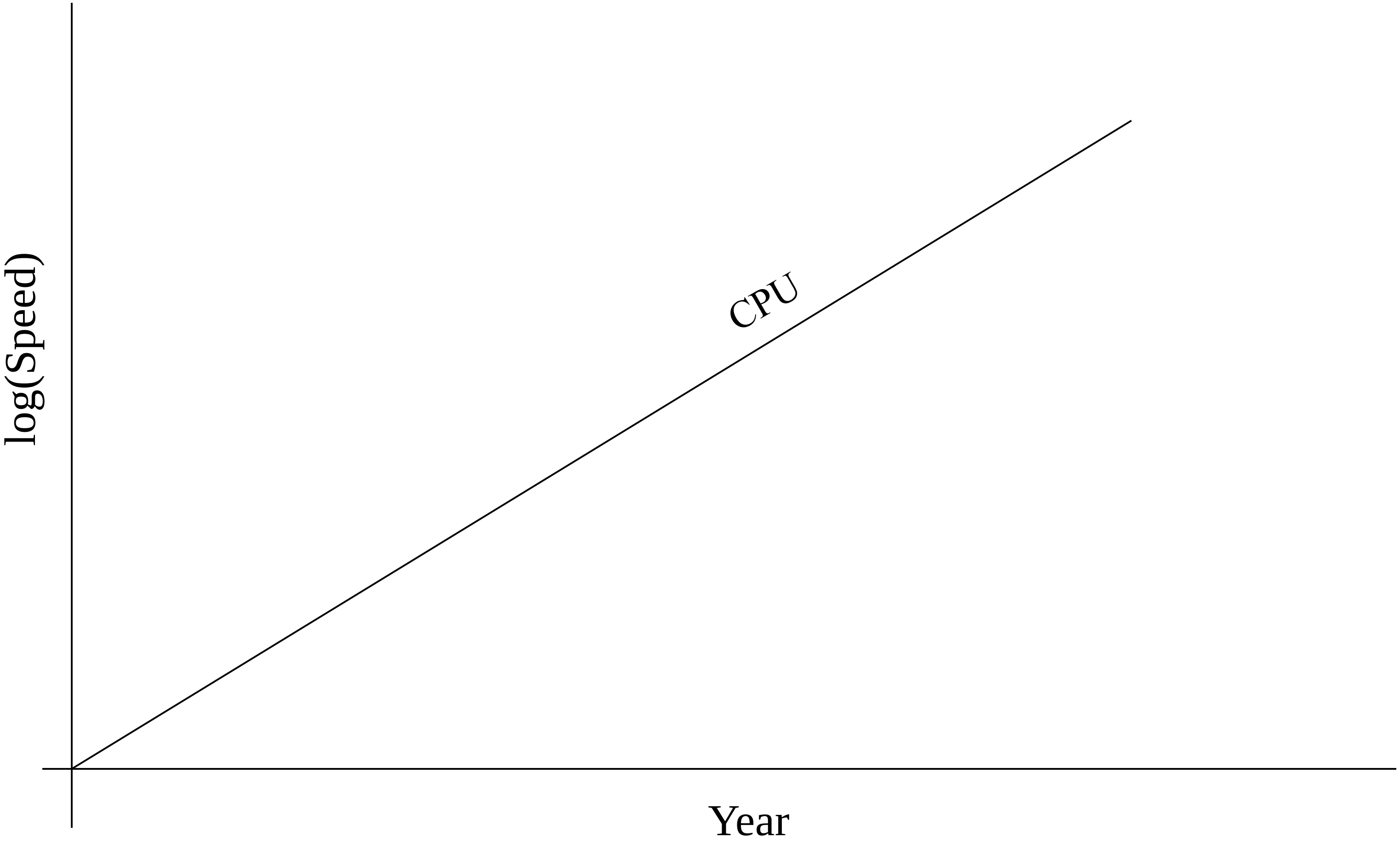
In 1965, Gordon Moore was an engineer at Fairchild Semiconductor.

He noticed that the number of transistors that could be squeezed onto a chip was doubling about every 18 months.

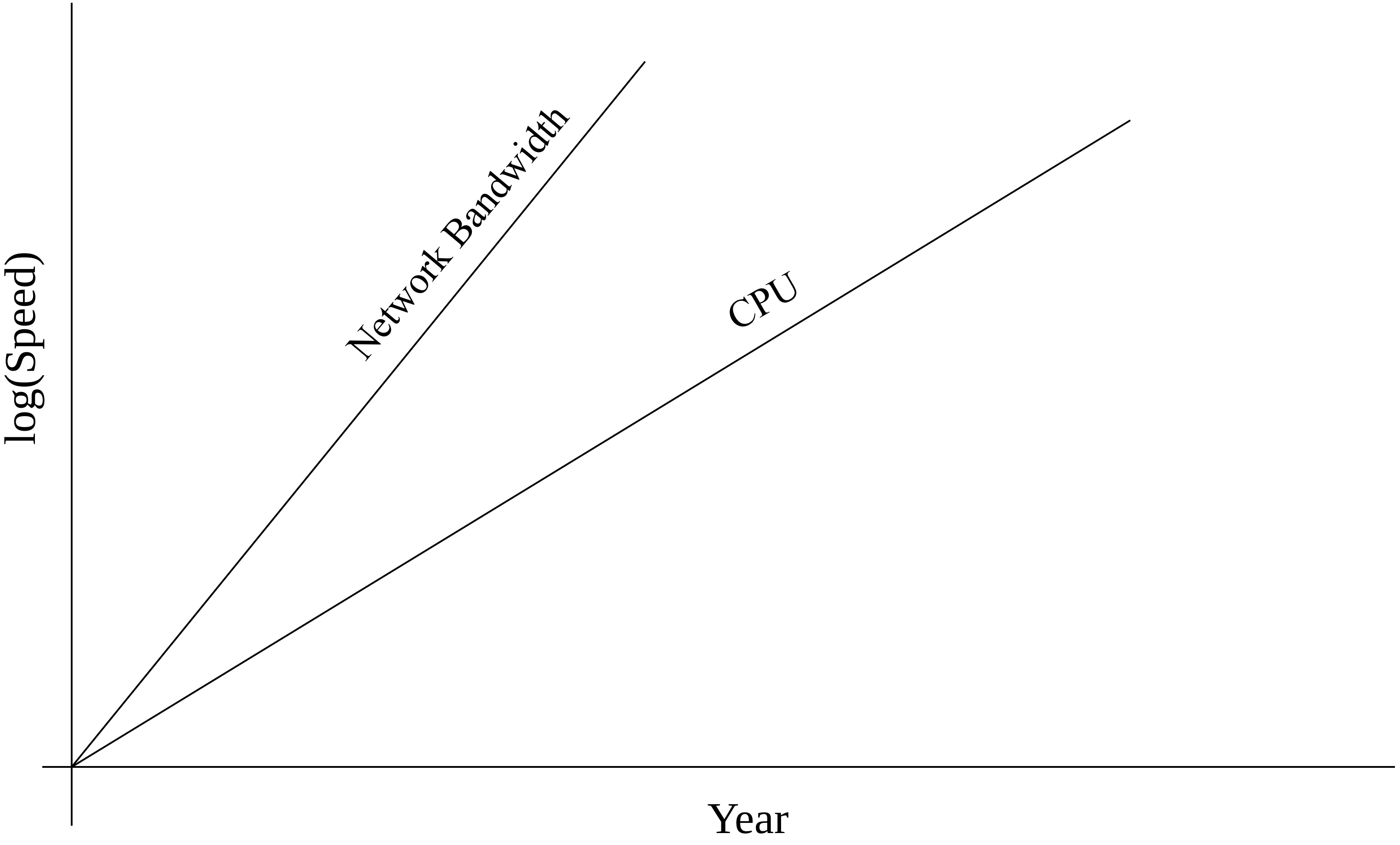
It turns out that computer speed is roughly proportional to the number of transistors per unit area.

Moore wrote a paper about this concept, which became known as ***“Moore’s Law.”***

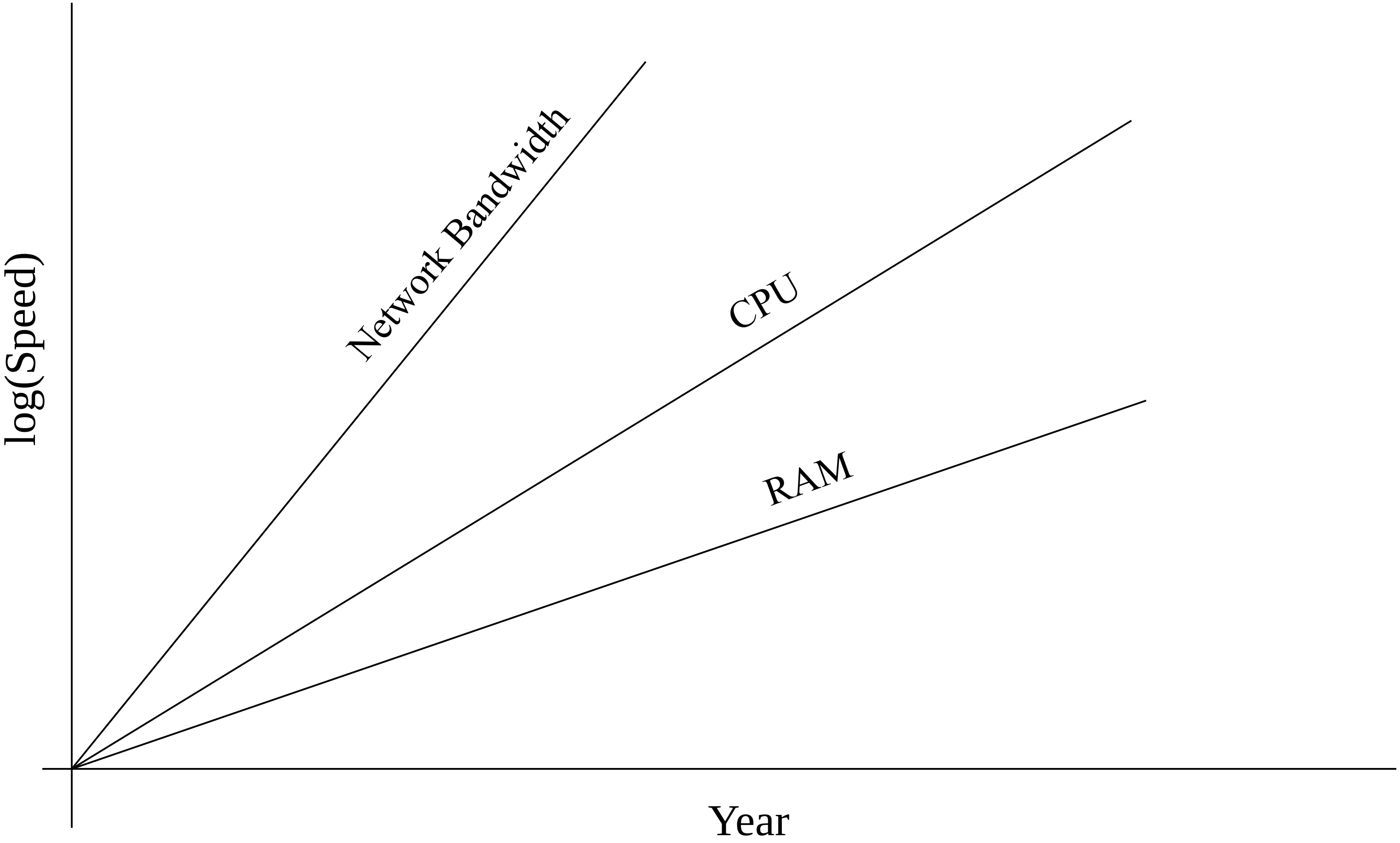
Moore’s Law in Practice



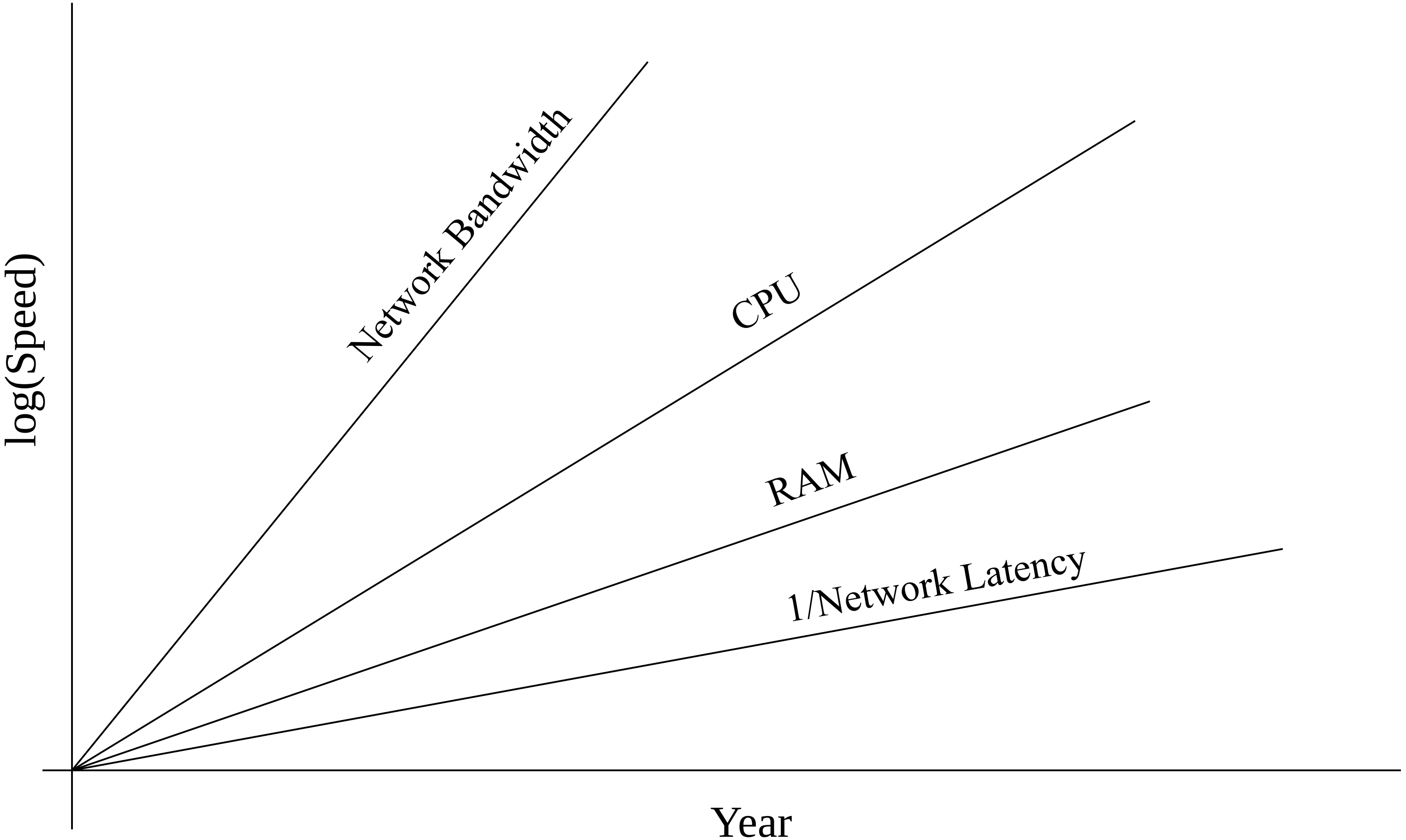
Moore’s Law in Practice



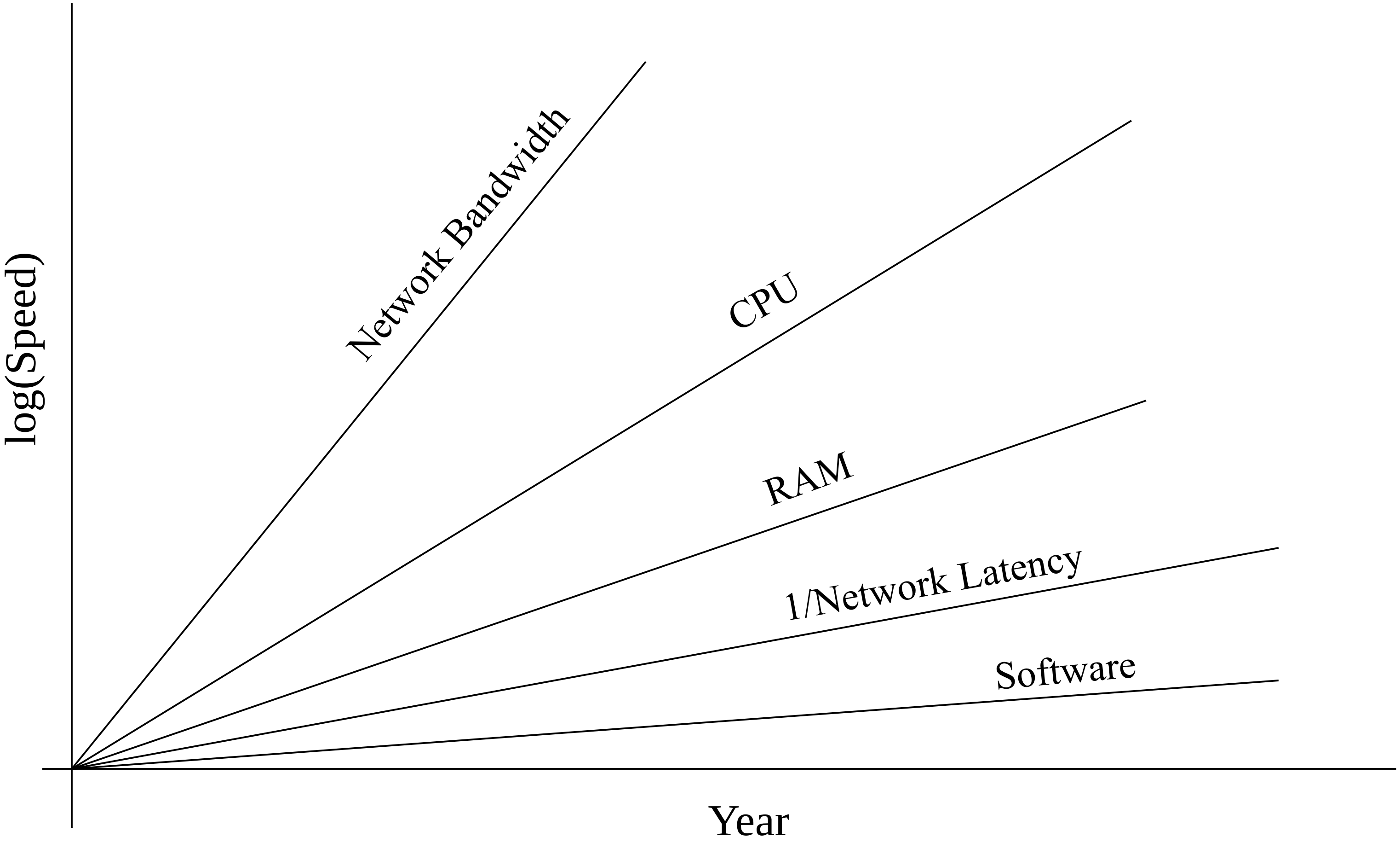
Moore's Law in Practice



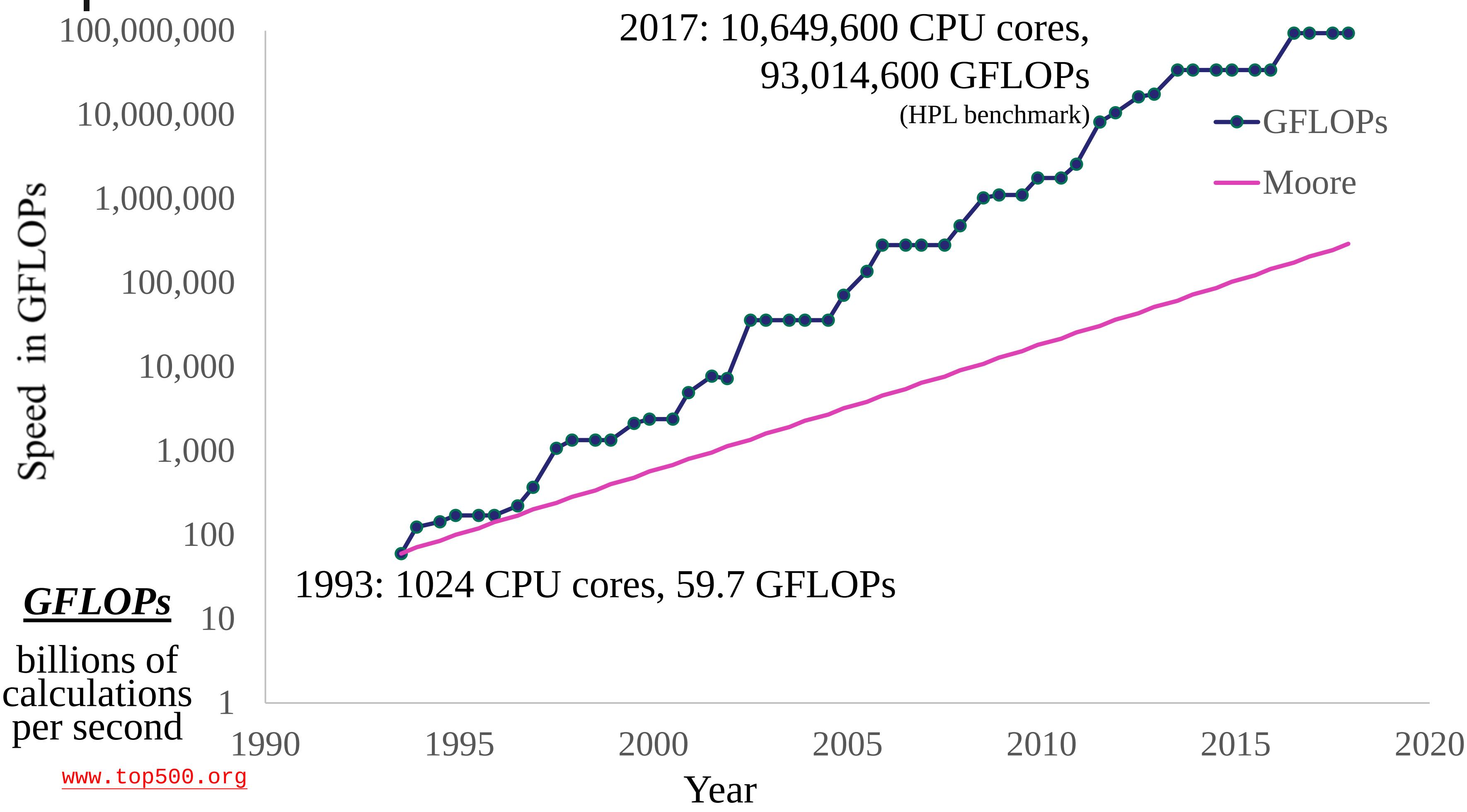
Moore’s Law in Practice



Moore’s Law in Practice



Fastest Supercomputer vs. Moore



2017: 10,649,600 CPU cores, 93,014,600 GFLOPs (HPL benchmark)

1993: 1024 CPU cores, 59.7 GFLOPs

GFLOPs

billions of calculations per second

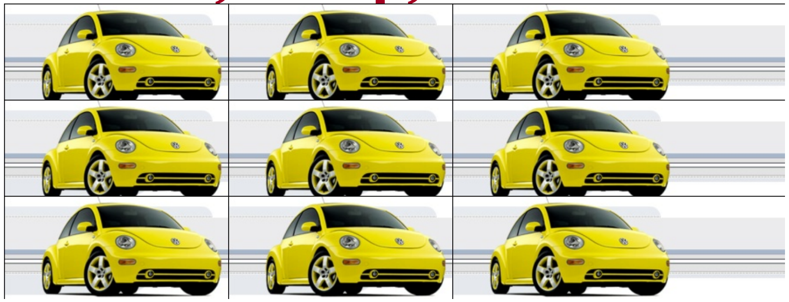
www.top500.org

The Tyranny of the Storage Hierarchy

The Storage Hierarchy

Fast, expensive, few

Slow, cheap, a lot

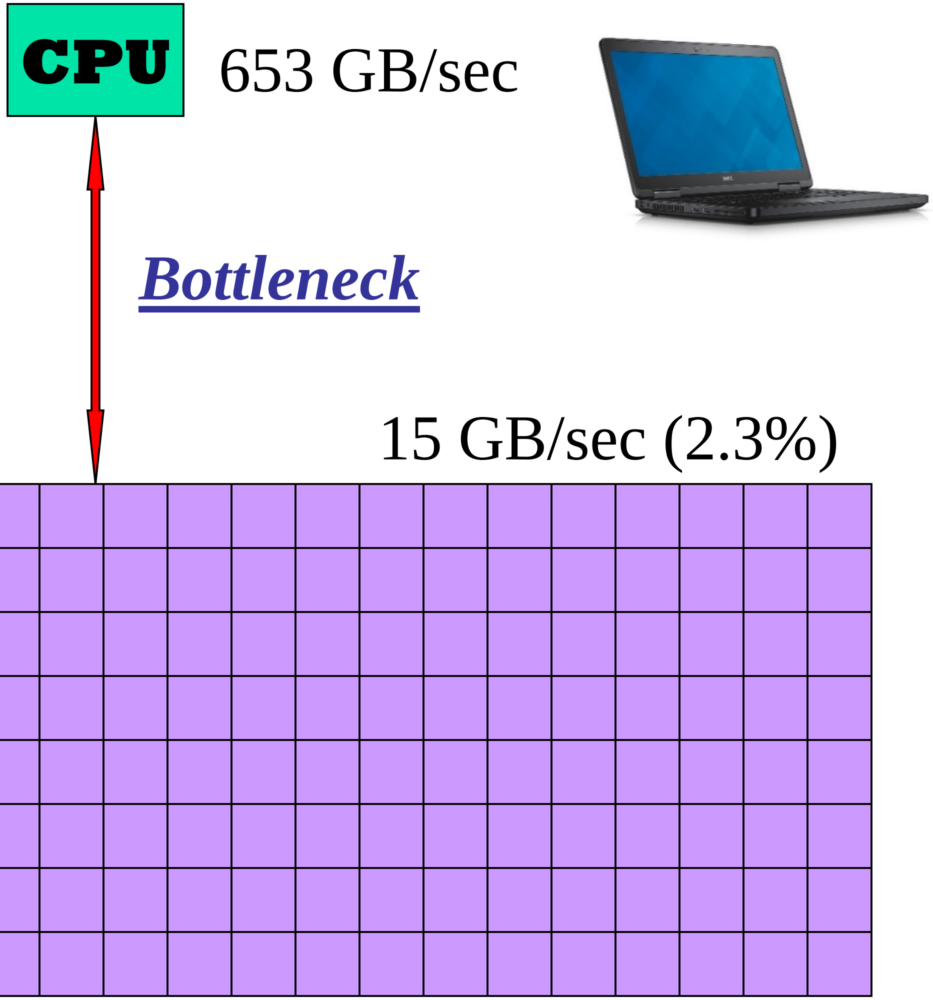


[5]

- Registers
- Cache memory
- Main memory (RAM)
- Hard disk
- Removable media (CD, DVD etc)
- Internet

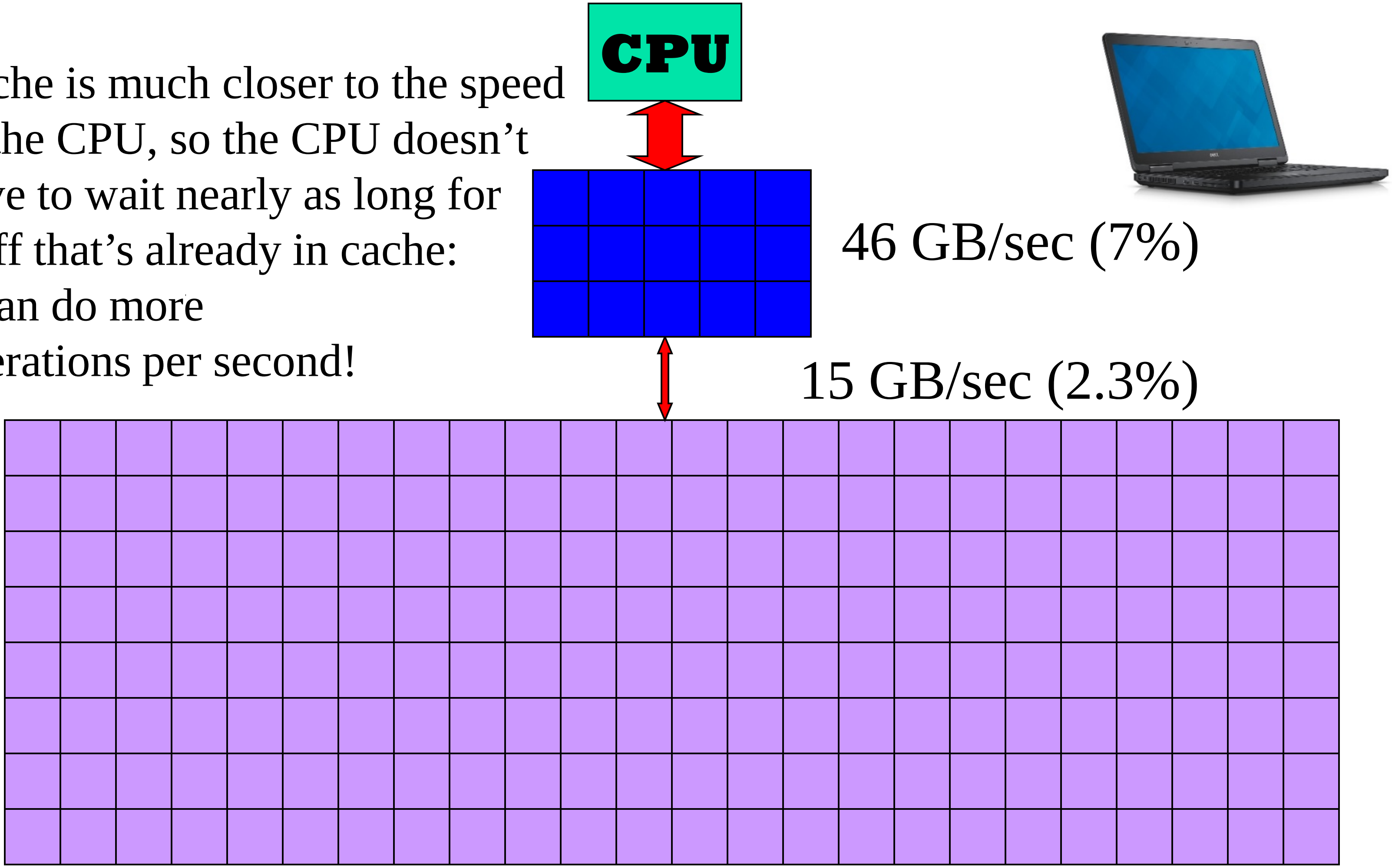
RAM is Slow

The speed of data transfer between Main Memory and the CPU is much slower than the speed of calculating, so the CPU spends most of its time waiting for data to come in or go out.



Why Have Cache?

Cache is much closer to the speed of the CPU, so the CPU doesn’t have to wait nearly as long for stuff that’s already in cache: it can do more operations per second!



Henry's Laptop

Dell Latitude E5540[4]

http://content.hwigroup.net/images/products/xl/204419/dell_latitude_e5540_55405115.jpg

- Intel Core i3-4010U
  dual core, 1.7 GHz, 3 MB L3 Cache
- 12 GB 1600 MHz DDR3L SDRAM
- 340 GB SATA 5400 RPM Hard Drive
- DVD±RW/CD-RW Drive
- 1 Gbps Ethernet Adapter

Storage Speed, Size, Cost

Henry's Laptop	Registers (Intel Core2 Duo 1.6 GHz)	Cache Memory (L3)	Main Memory (1600MHz DDR3L SDRAM)	Hard Drive	Ethernet (1000 Mbps)	DVD±R (16x)	Phone Modem (56 Kbps)
Speed (MB/sec) [peak]	668,672[6] (16 GFLOP/s*)	46,000	15,000 [7]	100[9]	125	32 [10]	0.007
Size (MB)	464 bytes** [11]	3	12,288 4096 times as much as cache	340,000	unlimited	unlimited	unlimited
Cost ($/MB)	–	$38 [12]	$0.0084 [12] ~1/4500 as much as cache	$0.00003 [12]	charged per month (typically)	$0.000045 [12]	charged per month (typically)

* GFLOP/s: billions of floating point operations per second

** 16 64-bit general purpose registers, 8 80-bit floating point registers, 16 128-bit floating point vector registers

Storage Use Strategies

- ***Register reuse***: Do a lot of work on the same data before working on new data.
- ***Cache reuse***: The program is much more efficient if all of the data and instructions fit in cache; if not, try to use what’s in cache a lot before using anything that isn’t in cache.
- ***Data locality***: Try to access data that are near each other in memory before data that are far.
- **I/O efficiency**: Do a bunch of I/O all at once rather than a little bit at a time; don’t mix calculations and I/O.

A Concrete Example

- Consider a cluster with Intel Xeon “Skylake” CPUs, model 6138: 20-core, 2.0 GHz (1.3 GHz AVX-512 base frequency), 2666 MHz 6-channel RAM.
- The theoretical peak CPU speed is 832 GFLOPs (double precision) per CPU chip, so for a dual chip node, the peak is 1664 GFLOPs.
- Each double precision calculation is 2 8-byte operands and one 8-byte result, so 24 bytes get moved between RAM and CPU.
- So, in theory each node could consume up to 39,936 GB/sec.
- The sustained RAM bandwidth is around 190 GB/sec.
- So, even at theoretical peak, any code that does less than around 210 calculations **per byte** transferred between RAM and cache has speed limited by RAM bandwidth.

Good Cache Reuse Example

A Sample Application

Matrix-Matrix Multiply

Let A, B and C be matrices of sizes
$nr \times nc$, $nr \times nk$ and $nk \times nc$, respectively:

$$\mathbf{A} = \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,nc} \\ a_{2,1} & a_{2,2} & a_{2,3} & \cdots & a_{2,nc} \\ a_{3,1} & a_{3,2} & a_{3,3} & \cdots & a_{3,nc} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{nr,1} & a_{nr,2} & a_{nr,3} & \cdots & a_{nr,nc} \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} b_{1,1} & b_{1,2} & b_{1,3} & \cdots & b_{1,nk} \\ b_{2,1} & b_{2,2} & b_{2,3} & \cdots & b_{2,nk} \\ b_{3,1} & b_{3,2} & b_{3,3} & \cdots & b_{3,nk} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ b_{nr,1} & b_{nr,2} & b_{nr,3} & \cdots & b_{nr,nk} \end{bmatrix} \quad \mathbf{C} = \begin{bmatrix} c_{1,1} & c_{1,2} & c_{1,3} & \cdots & c_{1,nc} \\ c_{2,1} & c_{2,2} & c_{2,3} & \cdots & c_{2,nc} \\ c_{3,1} & c_{3,2} & c_{3,3} & \cdots & c_{3,nc} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_{nk,1} & c_{nk,2} & c_{nk,3} & \cdots & c_{nk,nc} \end{bmatrix}$$

The definition of A = B • C is

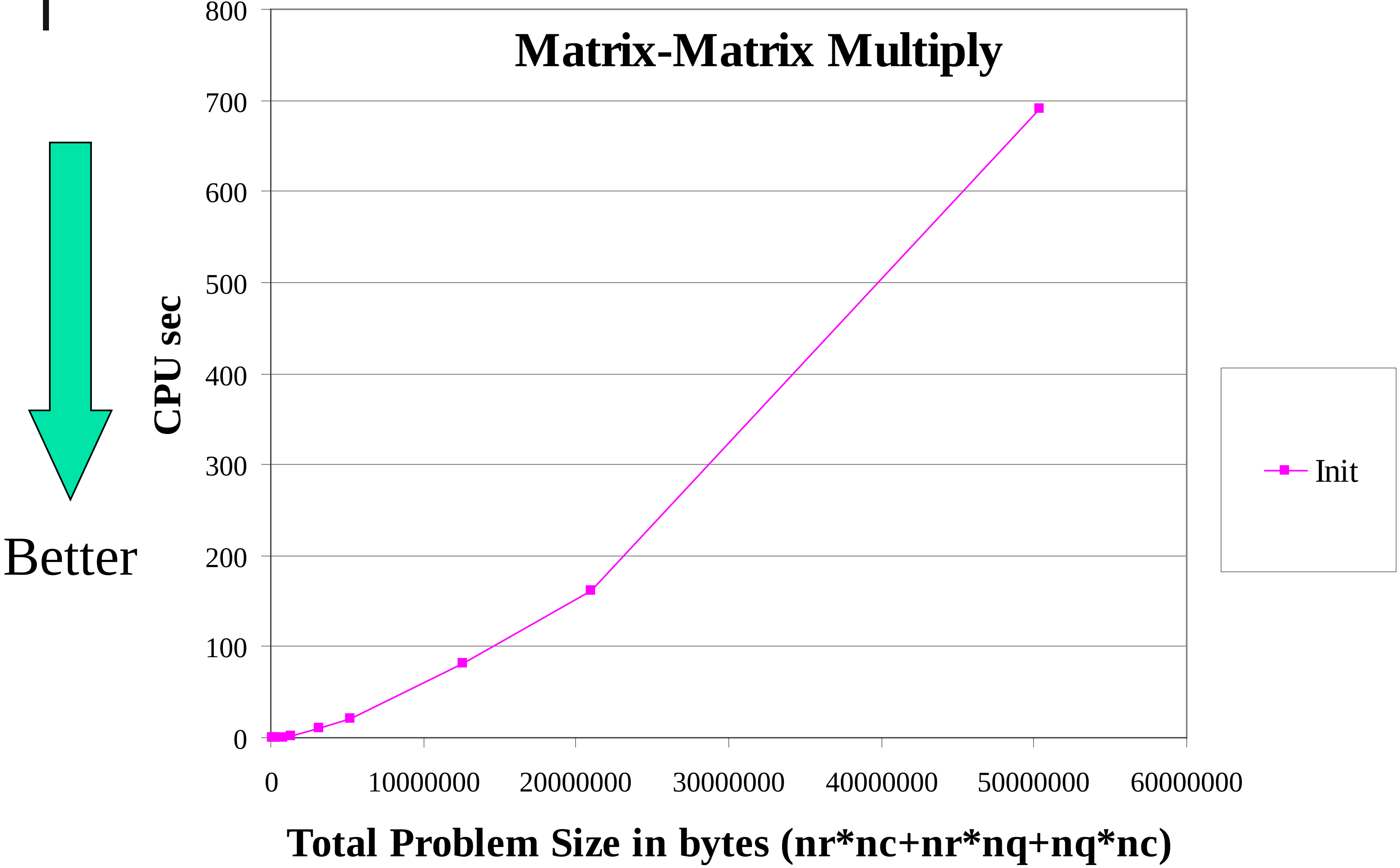
$$a_{r,c} = \sum_{k=1}^{nk} b_{r,k} \cdot c_{k,c} = b_{r,1} \cdot c_{1,c} + b_{r,2} \cdot c_{2,c} + b_{r,3} \cdot c_{3,c} + \ldots + b_{r,nk} \cdot c_{nk,c}$$

for $r \in \{1, nr\}$, $c \in \{1, nc\}$.

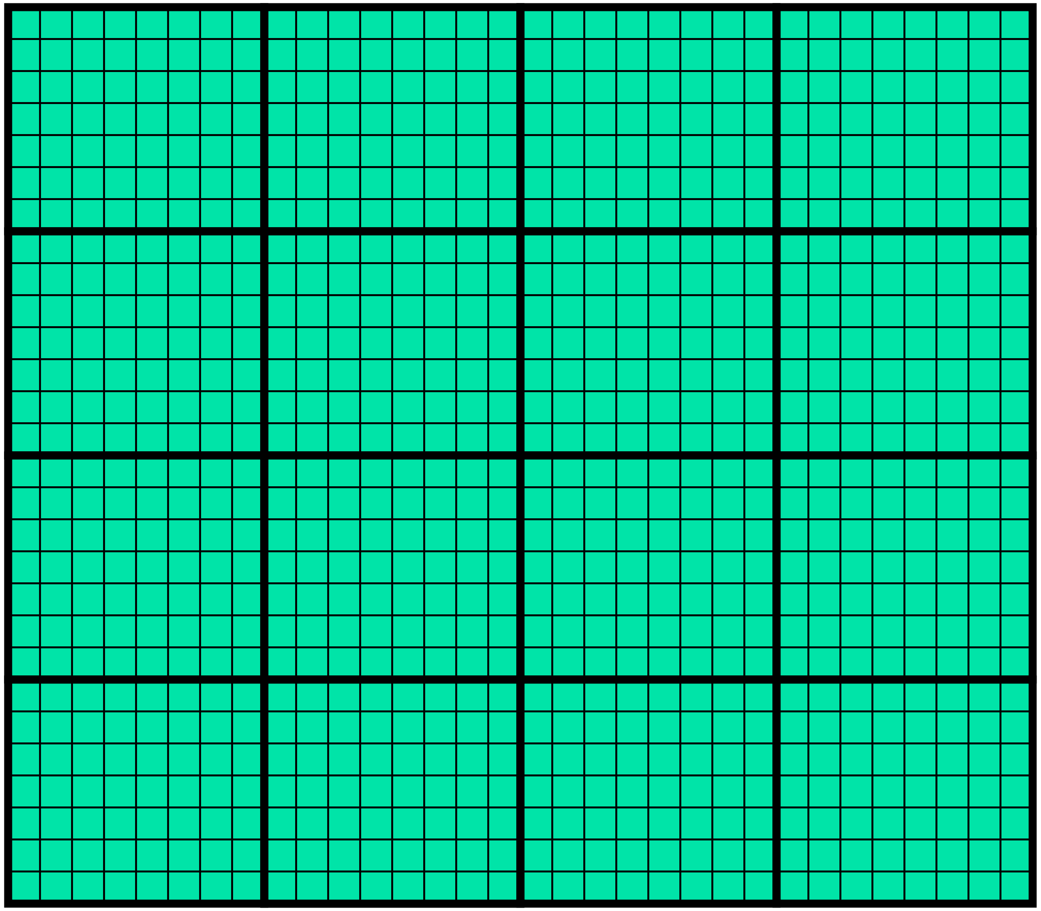
Matrix Multiply: Naïve Version

```
SUBROUTINE matrix_matrix_mult_naive (dst, src1, src2, &
 &                                   nr, nc, nq)
  IMPLICIT NONE
  INTEGER,INTENT(IN) :: nr, nc, nq
  REAL,DIMENSION(nr,nc),INTENT(OUT) :: dst
  REAL,DIMENSION(nr,nq),INTENT(IN)  :: src1
  REAL,DIMENSION(nq,nc),INTENT(IN)  :: src2

  INTEGER :: r, c, q

  DO c = 1, nc
    DO r = 1, nr
      dst(r,c) = 0.0
      DO q = 1, nq
        dst(r,c) = dst(r,c) + src1(r,q) * src2(q,c)
      END DO
    END DO
  END DO
END SUBROUTINE matrix_matrix_mult_naive
```

Performance of Matrix Multiply



Tiling



Tiling

- ***Tile***: a small rectangular subdomain of a problem domain. Sometimes called a ***block*** or a ***chunk***.
- ***Tiling***: breaking the domain into tiles.
- Tiling strategy: operate on each tile to completion, then move to the next tile.
- Tile size can be set at runtime, according to what's best for the machine that you're running on.

Tiling Code: F90

```
SUBROUTINE matrix_matrix_mult_by_tiling (dst, src1, src2, nr, nc, nq, &
 &           rtilesize, ctilesize, qtilesize)
  IMPLICIT NONE
  INTEGER,INTENT(IN) :: nr, nc, nq
  REAL,DIMENSION(nr,nc),INTENT(OUT) :: dst
  REAL,DIMENSION(nr,nq),INTENT(IN)  :: src1
  REAL,DIMENSION(nq,nc),INTENT(IN)  :: src2
  INTEGER,INTENT(IN) :: rtilesize, ctilesize, qtilesize

  INTEGER :: rstart, rend, cstart, cend, qstart, qend

  DO cstart = 1, nc, ctilesize
    cend = cstart + ctilesize - 1
    IF (cend > nc) cend = nc
    DO rstart = 1, nr, rtilesize
      rend = rstart + rtilesize - 1
      IF (rend > nr) rend = nr
      DO qstart = 1, nq, qtilesize
        qend = qstart + qtilesize - 1
        IF (qend > nq) qend = nq
        CALL matrix_matrix_mult_tile(dst, src1, src2, nr, nc, nq, &
 &                                   rstart, rend, cstart, cend, qstart, qend)
      END DO !! qstart
    END DO !! rstart
  END DO !! cstart
END SUBROUTINE matrix_matrix_mult_by_tiling
```

Tiling Code: C

```
void matrix_matrix_mult_by_tiling (
         float** dst, float** src1, float** src2,
         int nr, int nc, int nq,
         int rtilesize, int ctilesize, int qtilesize)
{ /* matrix_matrix_mult_by_tiling */
  int rstart, rend, cstart, cend, qstart, qend;

  for (rstart = 0; rstart < nr; rstart += rtilesize) {
    rend = rstart + rtilesize – 1;
    if (rend >= nr) rend = nr - 1;
    for (cstart = 0; cstart < nc; cstart += ctilesize) {
      cend = cstart + ctilesize – 1;
      if (cend >= nc) cend = nc - 1;
      for (qstart = 0; qstart < nq; qstart += qtilesize) {
        qend = qstart + qtilesize – 1;
        if (qend >= nq) qend = nq - 1;
        matrix_matrix_mult_tile(dst, src1, src2, nr, nc, nq,
                                rstart, rend, cstart, cend, qstart, qend);
      } /* for qstart */
    } /* for cstart */
  } /* for rstart */
} /* matrix_matrix_mult_by_tiling */
```

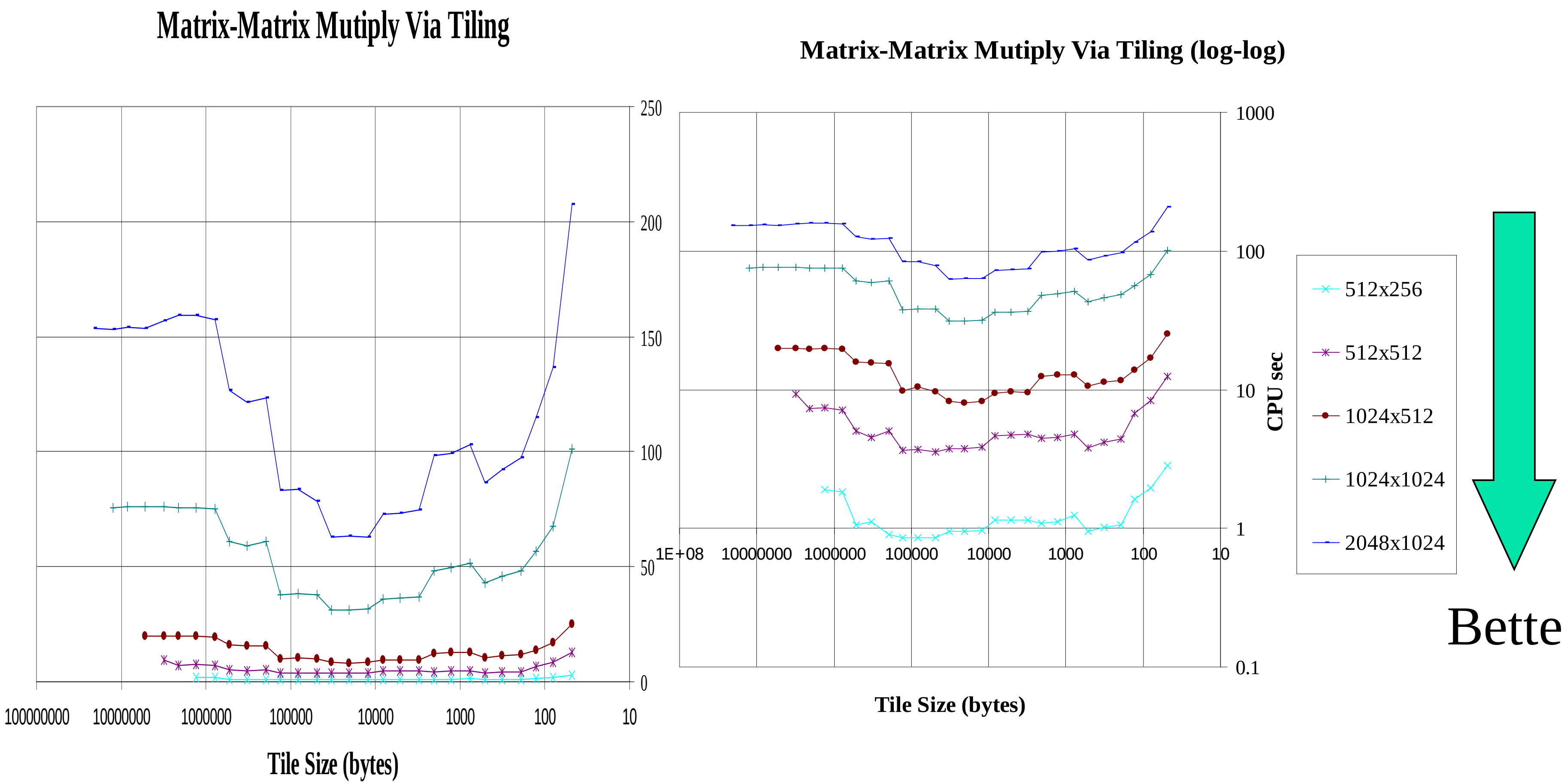
Multiplying Within a Tile: F90

```
SUBROUTINE matrix_matrix_mult_tile (dst, src1, src2, nr, nc, nq, &
 &             rstart, rend, cstart, cend, qstart, qend)
  IMPLICIT NONE
  INTEGER,INTENT(IN) :: nr, nc, nq
  REAL,DIMENSION(nr,nc),INTENT(OUT) :: dst
  REAL,DIMENSION(nr,nq),INTENT(IN)  :: src1
  REAL,DIMENSION(nq,nc),INTENT(IN)  :: src2
  INTEGER,INTENT(IN) :: rstart, rend, cstart, cend, qstart, qend

  INTEGER :: r, c, q

  DO c = cstart, cend
    DO r = rstart, rend
      IF (qstart == 1) dst(r,c) = 0.0
      DO q = qstart, qend
        dst(r,c) = dst(r,c) + src1(r,q) * src2(q,c)
      END DO !! q
    END DO !! r
  END DO !! c
END SUBROUTINE matrix_matrix_mult_tile
```

Multiplying Within a Tile: C

```
void matrix_matrix_mult_tile (
         float** dst, float** src1, float** src2,
         int nr, int nc, int nq,
         int rstart, int rend, int cstart, int cend,
         int qstart, int qend)
{ /* matrix_matrix_mult_tile */
  int r, c, q;

  for (r = rstart; r <= rend; r++) {
    for (c = cstart; c <= cend; c++) {
      if (qstart == 0) dst[r][c] = 0.0;
      for (q = qstart; q <= qend; q++) {
        dst[r][c] = dst[r][c] + src1[r][q] * src2[q][c];
      } /* for q */
    } /* for c */
  } /* for r */
} /* matrix_matrix_mult_tile */
```

Performance with Tiling



Matrix-Matrix Mutiply Via Tiling

Matrix-Matrix Mutiply Via Tiling (log-log)

Better

The Advantages of Tiling

- It allows your code to **exploit data locality** better, to get much more cache reuse: your code runs faster!
- It's a relatively **modest amount of extra coding** (typically a few wrapper functions and some changes to loop bounds).
- **If you don't need** tiling – because of the hardware, the compiler or the problem size – then you can **turn it off by simply** setting the tile size equal to the problem size.

Will Tiling Always Work?

Tiling **WON’T** always work. Why?

Well, tiling works well when:

- the order in which calculations occur doesn’t matter much, AND
- there are lots and lots of calculations to do for each memory movement.

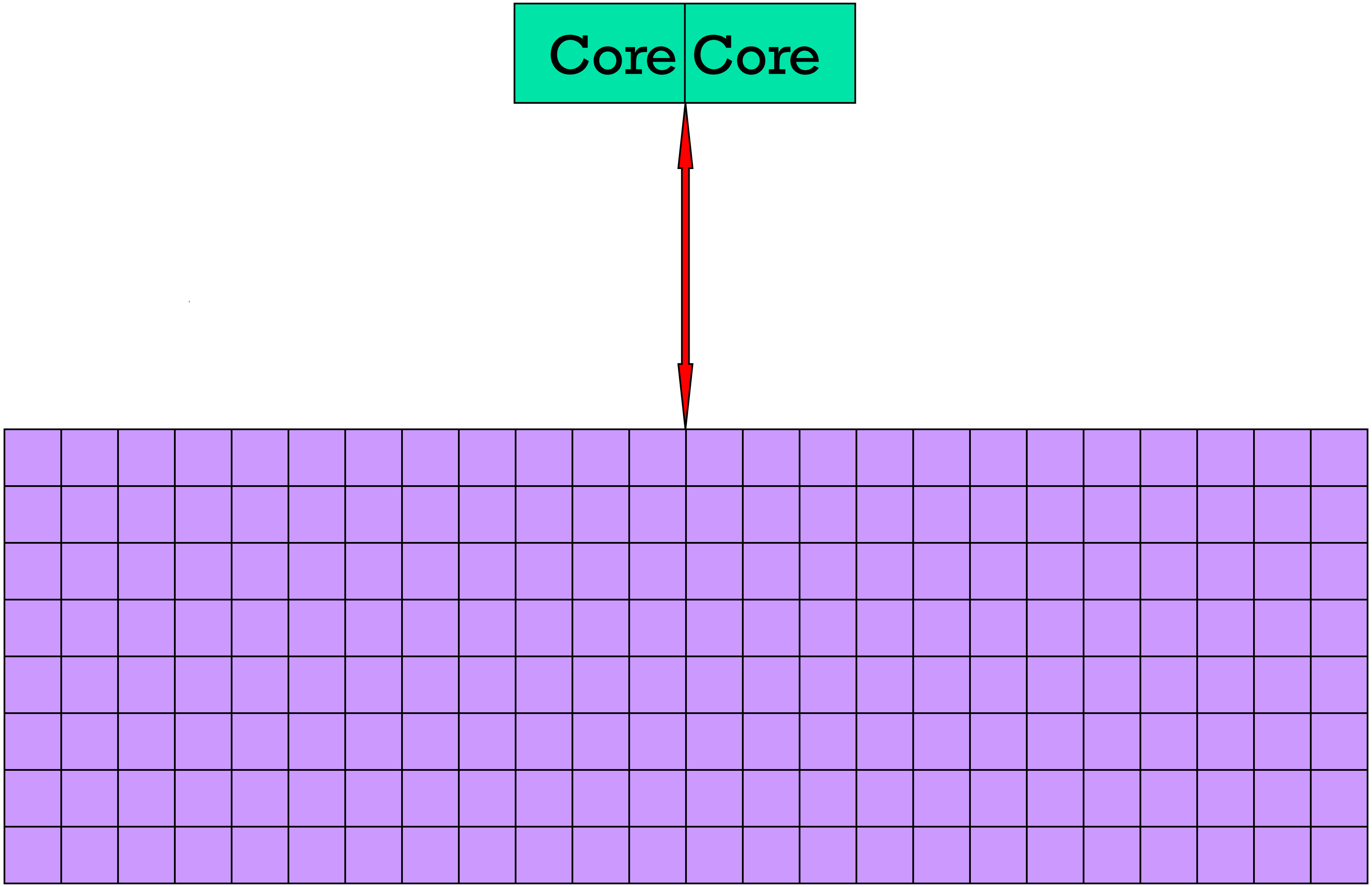
If either condition is absent, then tiling won’t help.

Multicore/Many-core Basics

What is Multicore?

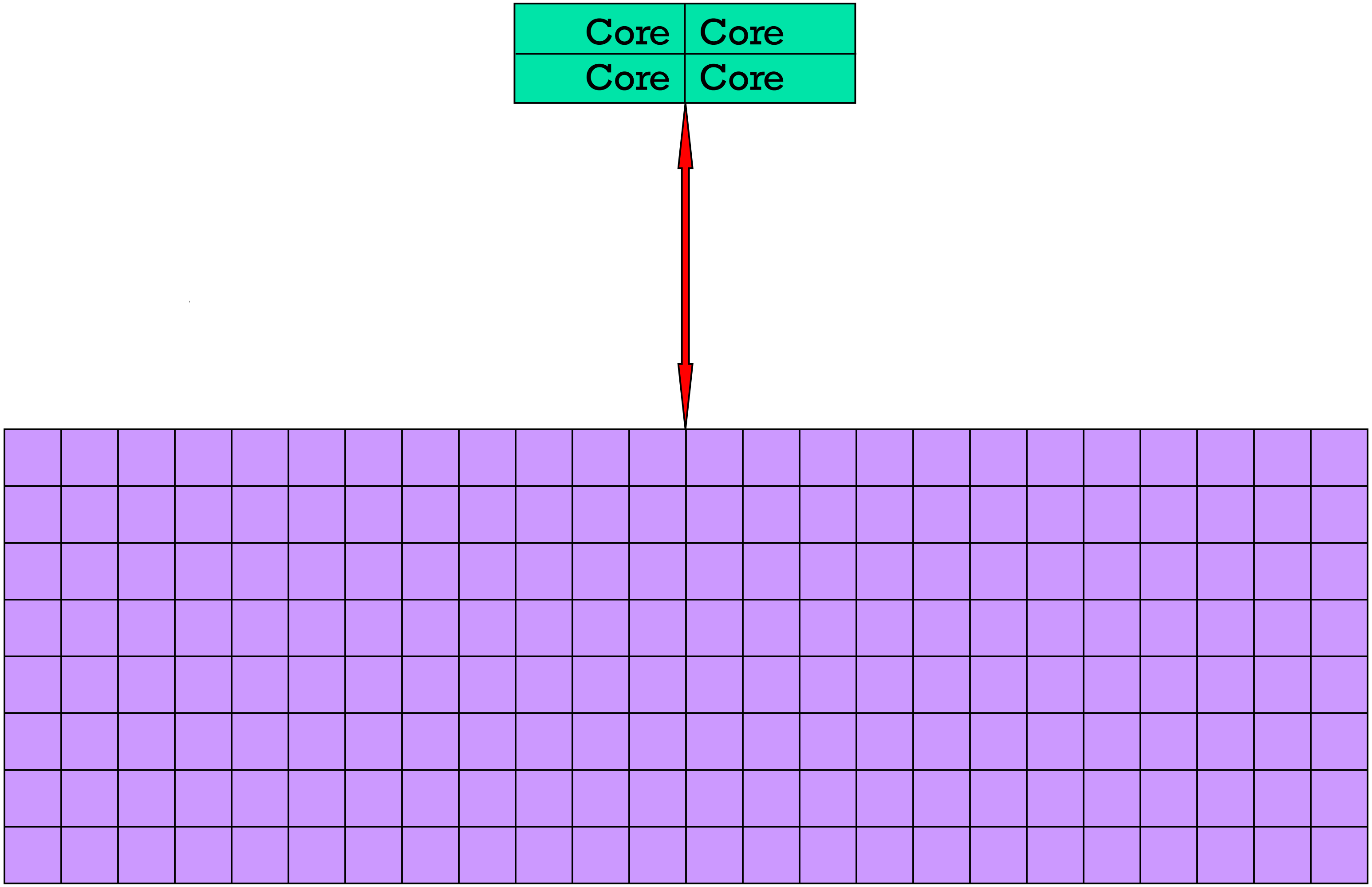
- In the olden days (that is, through the first half of 2005), each CPU chip had one "brain" in it.
- Starting the second half of 2005, each CPU chip could have up to 2 ***cores*** (brains); starting in late 2006, 4 cores; starting in late 2008, 6 cores; in early 2010, 8 cores; in mid 2010, 12 cores; in 2011, 16 cores (AMD); by 2017, 32 cores (AMD).
- **Jargon**: Each CPU chip plugs into a ***socket***, so these days, to avoid confusion, people refer to **sockets** and **cores**, rather than CPUs or processors.
- Each core is just like a full blown CPU, except that it shares its socket (and maybe some of its cache) with one or more other cores – and therefore shares its bandwidth to RAM with them.

Dual Core

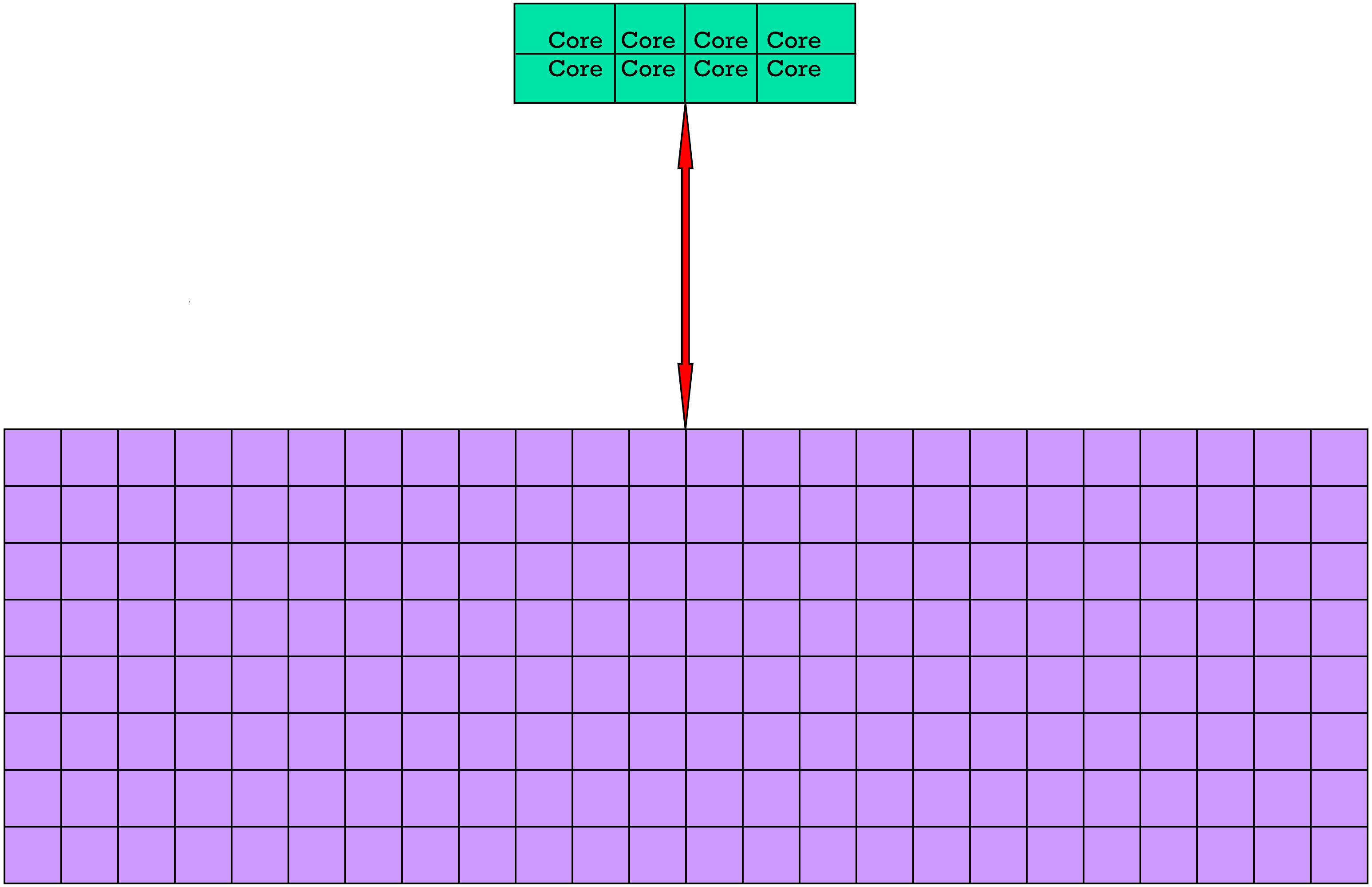


Core Core

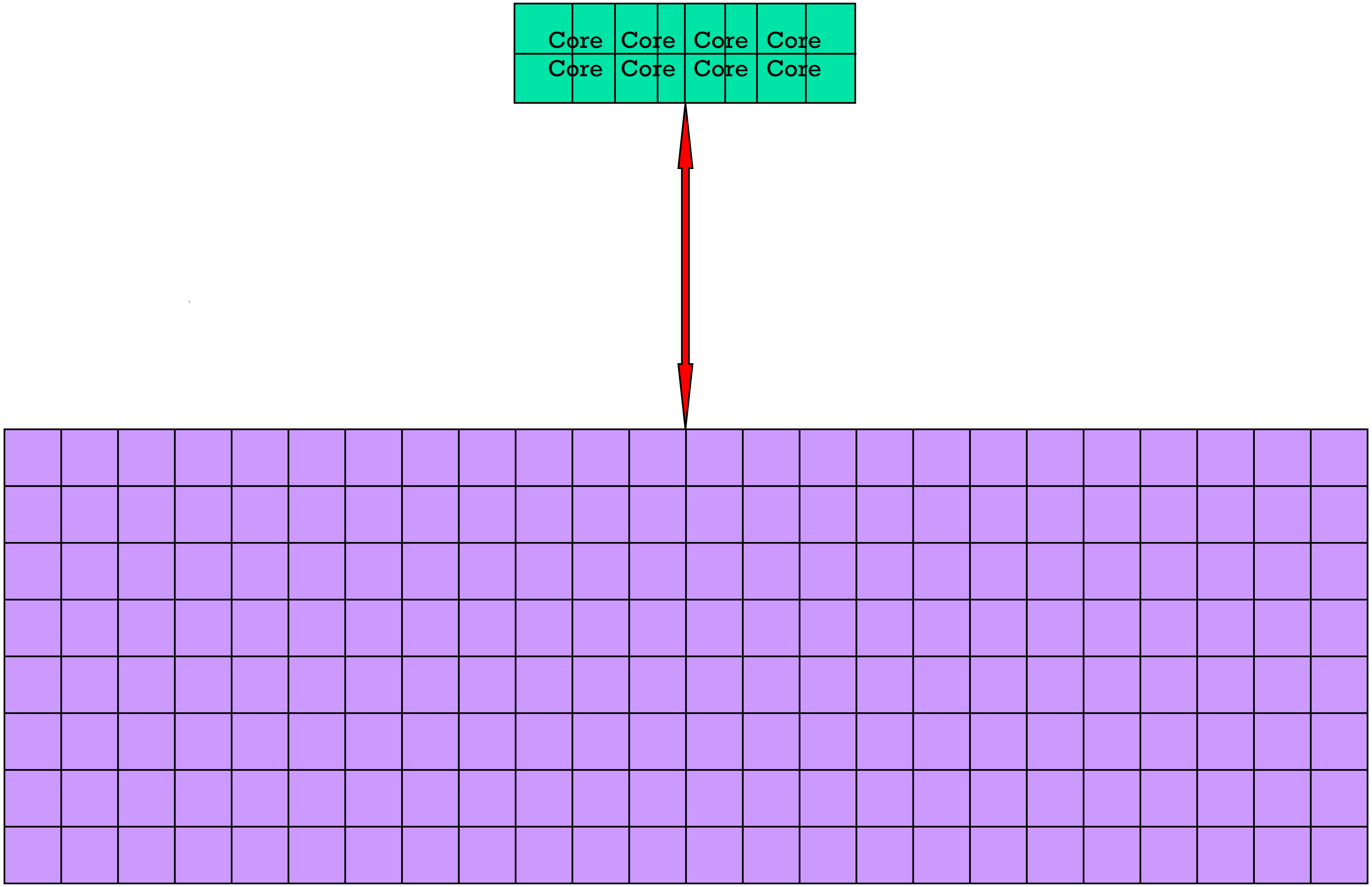
Quad Core



Oct Core



16-Core



The Challenge of Multicore: RAM

- Each socket has access to a certain amount of RAM, at a **fixed RAM bandwidth per SOCKET** – or even per node.
- As the number of cores per socket increases, the **contention for RAM bandwidth increases** too.
- At 2 or even 4 cores in a socket, this problem isn’t too bad. But at 16 or 32 or 64 cores, it can be **a huge problem**.
- So, applications that **are cache optimized** will get **big speedups**.
- But, applications whose performance is **limited by RAM bandwidth** are going to speed up only as fast as RAM bandwidth speeds up.
- RAM bandwidth **speeds up much slower** than CPU speeds up.

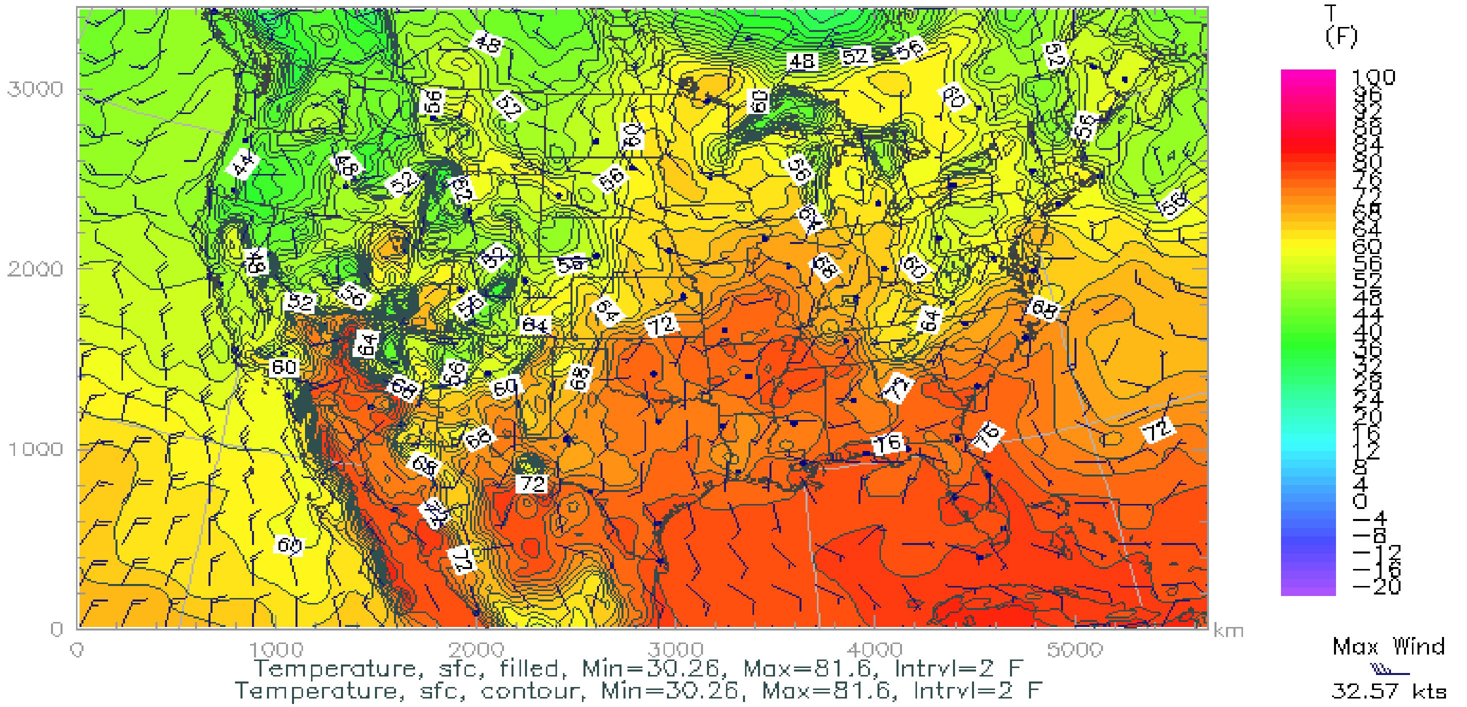
The Challenge of Multicore: Network

- Each node has access to a certain number of network ports, at a **fixed number of network ports per NODE**.
- As the number of cores per node increases, the **contention for network ports increases** too.
- At 2 or 4 cores in a socket, this problem isn't too bad. But at 16 or 32 or 64 cores, it can be **a huge problem**.
- So, applications that **do minimal communication** will get **big speedups**.
- But, applications whose performance is **limited by the number of MPI messages per second** are going to speed up very very little – and may even crash the node.

A Concrete Example: Weather Forecasting

Weather Forecasting

Thu, 25 May 2006, 8 am CDT (13Z)
Surface Temperature



Temperature, sfc, filled, Min=30.26, Max=81.6, Intrvl=2 F
Temperature, sfc, contour, Min=30.26, Max=81.6, Intrvl=2 F

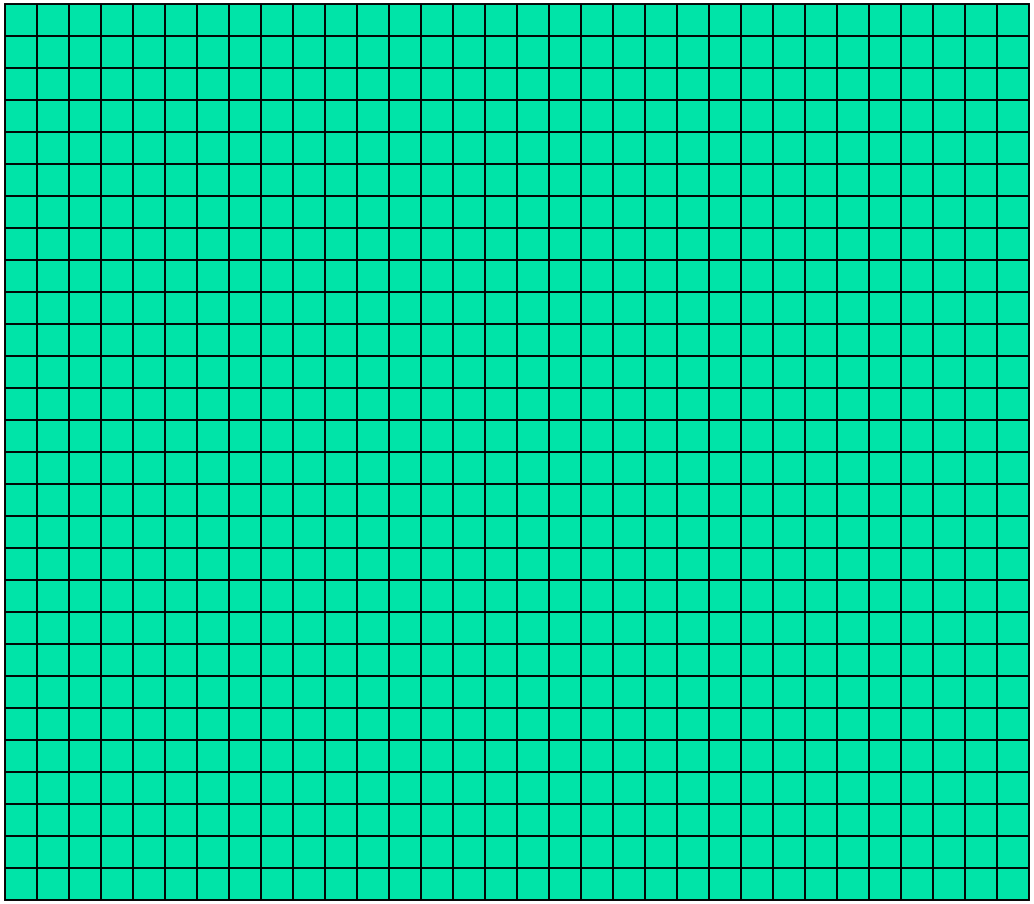
http://www.caps.ou.edu/wx/p/r/conus/fcst/

CAPS/OU Experimental ADAS Anlys CONUS, 210x128x50, dx=27 km 05/25/06 08:45 CDT

Weather Forecasting

- Weather forecasting is a **<u>transport</u>** problem.
- The goal is to predict future weather conditions by simulating the movement of fluids in Earth’s atmosphere.
- The physics is the Navier-Stokes Equations.
- The numerical method is Finite Difference.

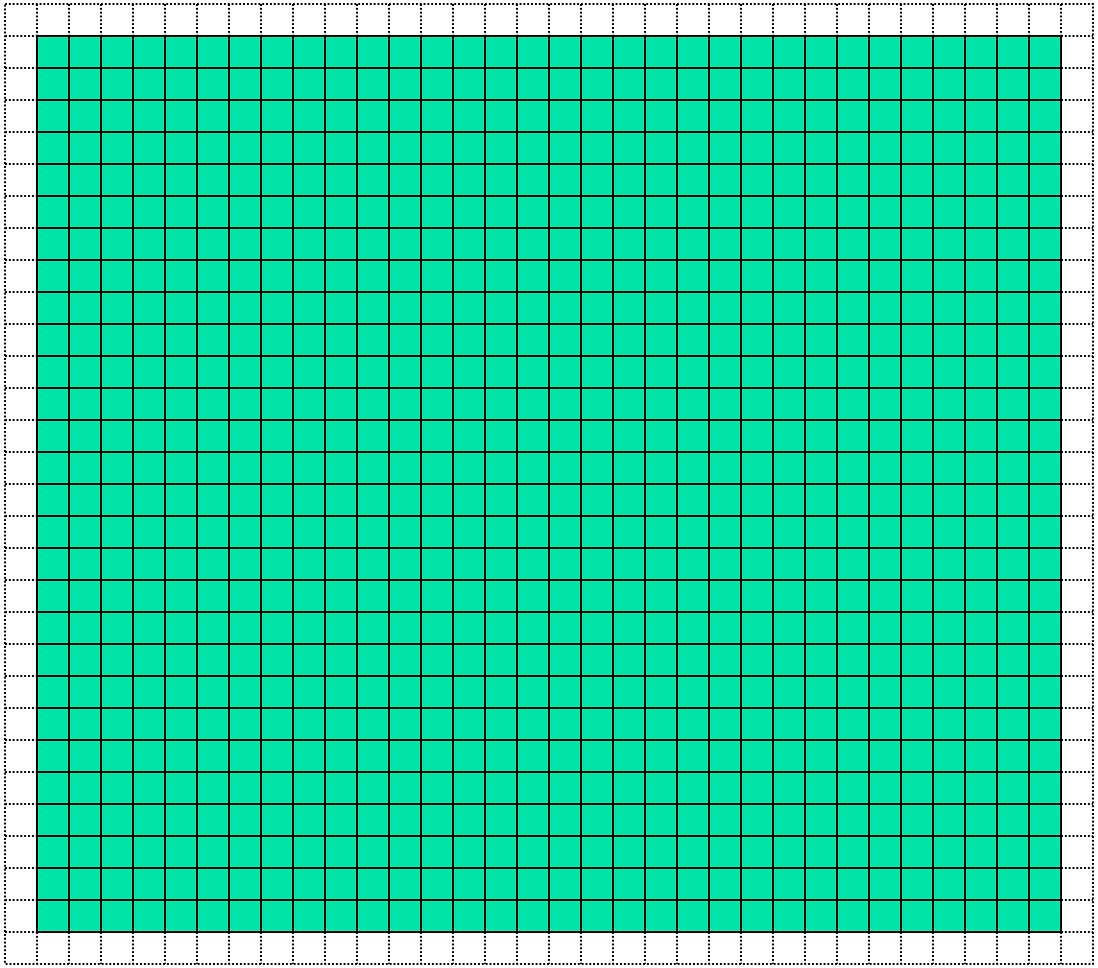
Cartesian Mesh



Finite Difference

$$
\begin{aligned}
unew(i,j,k) = \mathrm{F}(uold,\ i,\ j,\ k,\ \Delta t) = \\
\mathrm{F}(&uold(i,j,k), \\
&uold(i\text{-}1,j,k),\ uold(i+1,j,k), \\
&uold(i,j\text{-}1,k),\ uold(i,j+1,k), \\
&uold(i,j,k\text{-}1),\ uold(i,j,k+1),\ \Delta t)
\end{aligned}
$$

Ghost Boundary Zones



Virtual Memory

Virtual Memory

- Typically, the amount of main memory (RAM) that a CPU can ***address*** is larger than the amount of data physically present in the computer.
- For example, consider a laptop that can address 16 GB of main memory (roughly 16 billion bytes), but only contains 2 GB (roughly 2 billion bytes).

Virtual Memory (cont'd)

- **Locality**: Most programs don't jump all over the memory that they use; instead, they work in a particular area of memory for a while, then move to another area.
- So, you can offload onto hard disk much of the ***memory image*** of a program that's running.

Virtual Memory (cont'd)

- Memory is chopped up into many ***pages*** of modest size (for example, 1 KB – 32 KB; typically 4 KB).
- Only pages that have been recently used actually reside in memory; the rest are stored on hard disk.
- Hard disk is typically 0.1% as fast as main memory, so you get better performance if you rarely get a ***page fault***, which forces a read from (and maybe a write to) hard disk: **exploit data locality!**

Cache vs. Virtual Memory

- Lines (cache) vs. pages (VM)
- Cache faster than RAM (cache) vs.
  RAM faster than disk (VM)

Virtual Memory

- Every CPU family today uses ***virtual memory***, in which disk pretends to be a bigger RAM.
- Virtual memory capability **can't be turned off** (though you can turn off the ability to swap to disk).
- RAM is split up into ***pages***, typically **4 KB** each.
- Each page is either **in RAM** or **out on disk**.
- To keep track of the pages, a ***page table*** notes whether each table is in RAM, where it is in RAM (that is, physical address and virtual address are different), and some other information.
- So, a 4 GB physical RAM would need over a million ***page table entries*** – and a 32 GB physical RAM as on Schooner would need over 32M page table entries.

Why Virtual Memory is Slow

- When you want to access a byte of memory, you have to find out whether it’s in physical memory (RAM) or virtual memory (disk) – and the page table is in RAM!
- A page table of a 32 million entries can’t fit in, for example, the 25 MB cache L3 cache on Schooner CPU chips – and even if it could, that wouldn’t leave much cache for application data.
- So, each memory access (load or store) is actually 2 memory accesses: the first for the page table entry, and the second for the data itself.
- This is slow!
- And notice, this is assuming that you don’t need more memory than your physical RAM.

The Notorious T.L.B.

- To speed up memory accesses, CPUs today have a special cache just for page table entries, known as the ***Translation Lookaside Buffer*** (TLB).
- The size of TLBs varies from 64 entries to 1024 entries, depending on chip families. At 4 KB pages, this means that the size of cache covered by the TLB varies from 256 KB to 4 MB.
- Some TLBs allow large pages (1 MB to a few GB) – but the operating systems often provide poor or no support.

The T.L.B. on a Current Chip

On Intel Haswell, specifically E5-2650 v3 (e.g., on Schooner):

- L3 cache size is 25 MB, shared among 10 cores.
- L2 cache size is 256 KB per core (dedicated to that core).
- Page size can be 4 KB or 2 MB/4 MB or 1 GB.
- Data TLB (DTLB) per core is:
  - 64 entries for 4 KB pages, covering 256 KB per core, **OR**
  - 32 entries for 2 MB/4 MB pages, covering 128 MB per core, **OR**
  - 4 entries for 1 GB pages, covering 4 GB per core.
- DTLB is 4-way set associative.
- Shared TLB (STLB), containing a second level of both instruction and data TLB, for the whole chip is:
  - 1024 entries for 4 KB or 2 MB pages, covering 4 MB or 2 GB.
- A page table failure can cause a delay of hundreds of cycles.

(This information is from [13].)

The T.L.B. on a Recent Chip

On Intel Haswell, specifically E5-2650 v3 (e.g., on Schooner):

- L3 cache size is 25 MB, shared among 10 cores.
- L2 cache size is 256 KB per core (dedicated to that core).
- Page size can be 4 KB or 2 MB/4 MB or 1 GB.
- Data TLB (DTLB) per core is:
  - 64 entries for 4 KB pages, covering 256 KB per core, **OR**
  - 32 entries for 2 MB/4 MB pages, covering 128 MB per core, **OR**
  - 4 entries for 1 GB pages, covering 4 GB per core.
- Mesh: At 100 vertical levels of 150 single precision variables, 2 MB is a 5 x 5 horizontal domain – **almost nothing but ghost zones**!
- If your OS and compiler support 2 MB pages, then your TLB can support a 186 x 186 horizontal domain -- but your L3 cache can only support 20 x 20.

Software Strategies for Weather Forecasting on Multicore/Many-core

Tiling NOT Good for Weather Codes

- Weather codes typically have on the order of 150 3D arrays used in each timestep (some transferred multiple times in the same timestep, but let's ignore that for simplicity).
- These arrays typically are single precision (4 bytes per floating point value).
- So, a typical weather code uses about 600 bytes per mesh zone per timestep.
- Weather codes typically do 5,000 to 10,000 calculations per mesh zone per timestep.
- So, the ratio of calculations to data is less than 20 to 1 – much less than the ~210 to 1 needed (on 2017 hardware).

Weather Forecasting and Cache

- On current weather codes, data decomposition is per process. That is, each process gets one subdomain.
- As CPUs speed up and RAM sizes grow,
  the size of each processor’s subdomain grows too.
- However, given RAM bandwidth limitations, this means that performance can only grow with RAM speed –
  which increases slower than CPU speed.
- If the codes were optimized for cache, would they speed up more?
- First: How to optimize for cache?

How to Get Good Cache Reuse?

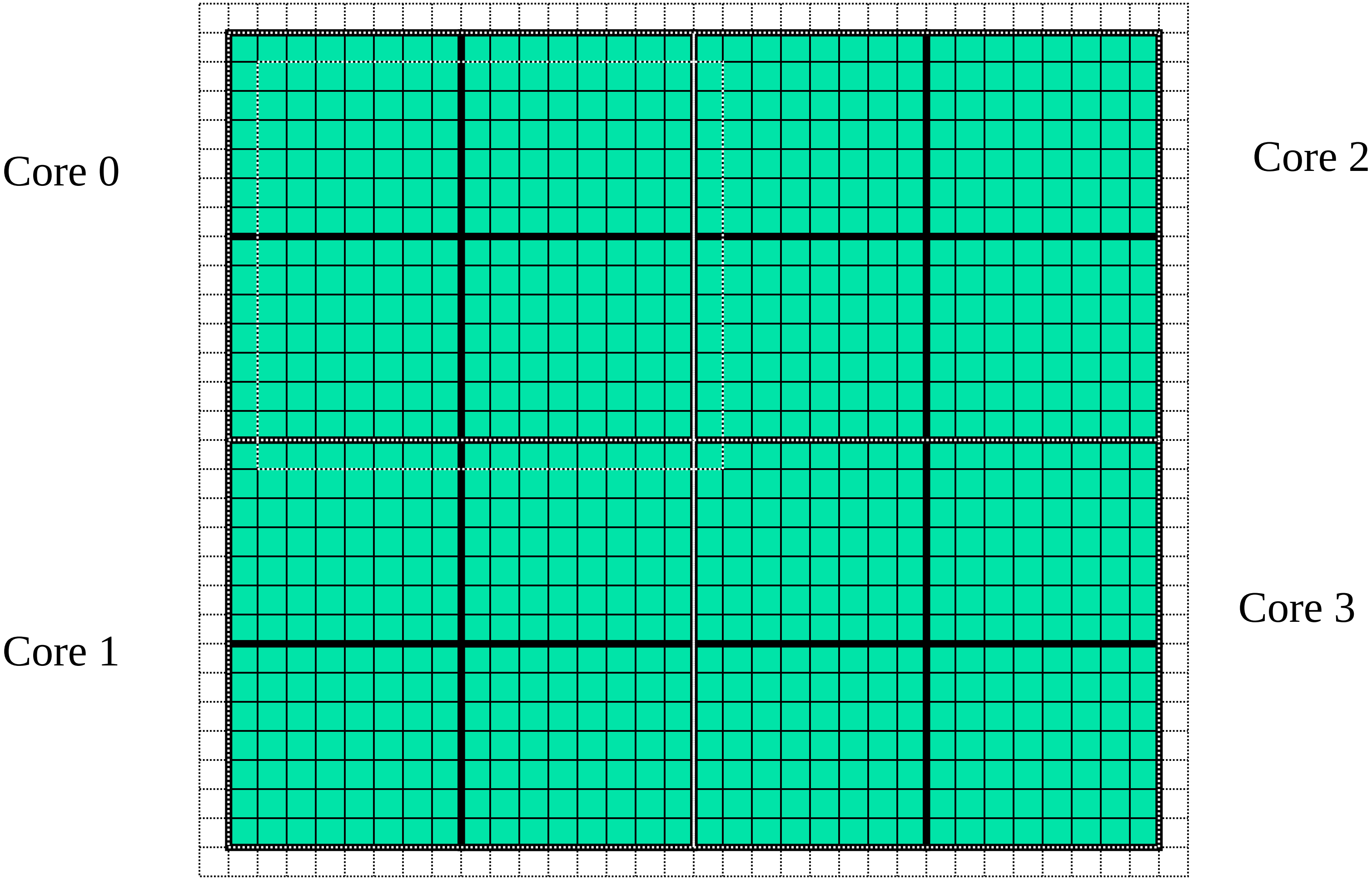
- Multiple independent subdomains per processor.
- Each subdomain fits entirely in L3 cache.
- Each subdomain’s page table entries fit entirely in the TLB.
- Expanded ghost zone ***stencil*** (ghost zones on each side) allows multiple timesteps before communicating with neighboring subdomains.
- Parallelize along the Z-axis as well as X and Y.
- Use higher order numerical schemes.
- Reduce the memory footprint as much as possible.

Coincidentally, this also reduces communication cost.

Cache Optimization Strategy: Tiling?

Would tiling work as a cache optimization strategy for weather forecasting codes?

Multiple Subdomains Per Core



Why Multiple Subdomains?

- If each subdomain fits in cache, then the CPU can bring all the data of a subdomain into cache, chew on it for a while, then move on to the next subdomain: lots of cache reuse!
- Oh, wait, what about the TLB? Better make the subdomains smaller! (So more of them.)
- But, doesn’t tiling have the same effect?

Why Independent Subdomains?

- Originally, the point of this strategy was to hide the cost of communication.
- When you finish chewing up a subdomain, send its data to its neighbors non-blocking (**`MPI_Isend`**).
- While the subdomain's data is flying through the interconnect, work on other subdomains,
  which hides the communication cost.
- When it's time to work on this subdomain again,
  collect its data (**`MPI_Waitall`**).
- If you've done enough work, then the communication cost is zero.

Expand the Array Stencil

- If you expand the array stencil of each subdomain beyond the numerical stencil, then you don’t have to communicate as often.
- When you communicate, instead of sending a slice along each face, send a slab, with extra stencil levels.
- In the first timestep after communicating, do extra calculations out to just inside the numerical stencil.
- In subsequent timesteps, calculate fewer and fewer stencil levels, until it’s time to communicate again – less total communication, and more calculations to hide the communication cost underneath!

An Extra Win!

- If you do all this, there's an amazing side effect: you get better cache reuse, because you stick with the same subdomain for a longer period of time.
- So, instead of doing, say, 5000 calculations per zone per timestep, you can do 15,000 or 20,000.
- So, you can better amortize the cost of transferring the data between RAM and cache.
- Downside: ratio of ghost zone RAM use to computed zone RAM use gets worse.
  - But RAM is cheap.

Old Algorithm (F90)

```
DO timestep = 1, number_of_timesteps
  CALL receive_messages_nonblocking(subdomain, timestep)
  CALL calculate_entire_timestep(subdomain, timestep)
  CALL send_messages_nonblocking(subdomain, timestep)
END DO
```

Old Algorithm (C)

```
for (timestep = 0;
     timestep < number_of_timesteps; timestep++) {
  receive_messages_nonblocking(subdomain, timestep);
  calculate_entire_timestep(subdomain, timestep);
  send_messages_nonblocking(subdomain, timestep);
} /* for timestep */
```

New Algorithm (F90)

```
DO timestep = 1, number_of_timesteps, extra_stencil_levels
  DO subdomain = 1, number_of_local_subdomains
    CALL receive_messages_nonblocking(subdomain, timestep)
    DO extra_stencil_level = 0, extra_stencil_levels - 1
      CALL calculate_entire_timestep(subdomain,
               timestep + extra_stencil_level)
    END DO
    CALL send_messages_nonblocking(subdomain,
             timestep + extra_stencil_levels)
`  END DO
END DO
```

New Algorithm (C)

```
for (timestep = 0;
     timestep < number_of_timesteps;
     timestep += extra_stencil_levels) {
  for (subdomain = 0;
       subdomain < number_of_local_subdomains; subdomain++) {
    receive_messages_nonblocking(subdomain, timestep);
    for (extra_stencil_level = 0;
         extra_stencil_level < extra_stencil_levels;
         extra_stencil_level++) {
      calculate_entire_timestep(subdomain,
        timestep + extra_stencil_level);
    } /* for extra_stencil_level */
    send_messages_nonblocking(subdomain,
      timestep + extra_stencil_levels);
  } /* for subdomain */
} /* for timestep */
```

Higher Order Numerical Schemes

- Higher order numerical schemes are great, because they require more calculations per mesh zone per timestep, which you need to amortize the cost of transferring data between RAM and cache. Might as well!
- Plus, they allow you to use a larger time interval per timestep (**`dt`**), so you can do fewer total timesteps for the same accuracy – or you can get higher accuracy for the same number of timesteps.

Parallelize in Z

- Most weather forecast codes parallelize in X and Y, but not in Z, because gravity makes the calculations along Z more complicated than X and Y.
- But, that means that each subdomain has a high number of zones in Z, compared to X and Y.
- For example, a 1 km CONUS run will probably have 100 zones in Z (25 km at 0.25 km resolution).

Multicore/Many-core Problem

- Most multicore chip families have relatively small cache per core (for example, 1 - 4 MB per core at the highest/slowest cache level) – and this problem seems likely to remain.
- Small TLBs make the problem worse: a few MB per core rather than 10-30 MB, for small page size VM.
- So, to get good cache reuse, you need subdomains of no more than a few MB.
- If you have 150 3D variables at single precision, and 100 zones in Z, then your horizontal size will be 5 x 5 zones – just enough for your stencil!

What Do We Need?

- We need much bigger caches!
  - But that depends on transistor siz, which is becoming harder and harder to shrink.
- TLB must be big enough to cover the entire cache.
- It’d be nice to have RAM speed increase as fast as core counts increase, but let’s not kid ourselves.

Keep this in mind when we get to GPGPU!

TENTATIVE Schedule

Tue Jan 23: Storage: What the Heck is Supercomputing?
Tue Jan 30: The Tyranny of the Storage Hierarchy Part I
Tue Feb 6: The Tyranny of the Storage Hierarchy Part II
Tue Feb 13: Instruction Level Parallelism
Tue Feb 20: Stupid Compiler Tricks
Tue Feb 27: Multicore Multithreading
Tue March 6: Distributed Multiprocessing
Tue March 13: **NO SESSION** (Henry business travel)
Tue March 20: **NO SESSION** (OU's Spring Break)
Tue March 27: Applications and Types of Parallelism
Tue Apr 3: Multicore Madness
Tue Apr 10: **NO SESSION** (Henry business travel)
Tue Apr 17: High Throughput Computing
Tue Apr 24: GPGPU: Number Crunching in Your Graphics Card
Tue May 1: Grab Bag: Scientific Libraries, I/O Libraries, Visualization

Thanks for helping!

- OU IT
  - OSCER operations staff (Dave Akin, Patrick Calhoun, Kali McLennan, Jason Speckman, Brett Zimmerman)
  - OSCER Research Computing Facilitators (Jim Ferguson, Horst Severini)
  - Debi Gentis, OSCER Coordinator
  - Kyle Dudgeon, OSCER Manager of Operations
  - Ashish Pai, Managing Director for Research IT Services
  - The OU IT network team
  - OU CIO Eddie Huebsch
- OneNet: Skyler Donahue
- Oklahoma State U: Dana Brunson

This is an experiment!

It’s the nature of these kinds of videoconferences that **FAILURES ARE GUARANTEED TO HAPPEN! NO PROMISES!**

So, please bear with us. Hopefully everything will work out well enough.

If you lose your connection, you can retry the same kind of connection, or try connecting another way.

Remember, if all else fails, you always have the phone bridge to fall back on.

PLEASE MUTE YOURSELF.

PLEASE MUTE YOURSELF.

PLEASE MUTE YOURSELF.

Coming in 2018!

- Coalition for Advancing Digital Research & Education (CADRE) Conference: Apr 17-18 2018 @ Oklahoma State U, Stillwater OK USA
  https://hpcc.okstate.edu/cadre-conference
- Linux Clusters Institute workshops
  http://www.linuxclustersinstitute.org/workshops/
  - Introductory HPC Cluster System Administration: May 14-18 2018 @ U Nebraska, Lincoln NE USA
  - Intermediate HPC Cluster System Administration: Aug 13-17 2018 @ Yale U, New Haven CT USA
- Great Plains Network Annual Meeting: details coming soon
- Advanced Cyberinfrastructure Research & Education Facilitators (ACI-REF) Virtual Residency Aug 5-10 2018, U Oklahoma, Norman OK USA
- PEARC 2018, July 22-27, Pittsburgh PA USA
  https://www.pearc18.pearc.org/
- IEEE Cluster 2018, Sep 10-13, Belfast UK
  https://cluster2018.github.io
- **OKLAHOMA SUPERCOMPUTING SYMPOSIUM 2018, Sep 25-26 2018 @ OU**
- SC18 supercomputing conference, Nov 11-16 2018, Dallas TX USA
  http://sc18.supercomputing.org/

Thanks for your attention!

Questions?

www.oscer.ou.edu

References

[1] Image by Greg Bryan, Columbia U.

[2] "Update on the Collaborative Radar Acquisition Field Test (CRAFT): Planning for the Next Steps." Presented to NWS Headquarters August 30 2001.

[3] See http://hneeman.oscer.ou.edu/hamr.html for details.

[4] http://www.dell.com/

[5] http://www.vw.com/newbeetle/

[6] Richard Gerber, The Software Optimization Cookbook: High-performance Recipes for the Intel Architecture. Intel Press, 2002, pp. 161-168.

[7] RightMark Memory Analyzer. http://cpu.rightmark.org/

[8] ftp://download.intel.com/design/Pentium4/papers/24943801.pdf

[9] http://www.seagate.com/cda/products/discsales/personal/family/0,1085,621,00.html

[10] http://www.samsung.com/Products/OpticalDiscDrive/SlimDrive/OpticalDiscDrive_SlimDrive_SN_S082D.asp?page=Specifications

[11] ftp://download.intel.com/design/Pentium4/manuals/24896606.pdf

[12] http://www.pricewatch.com/

[13] https://en.wikichip.org/wiki/intel/microarchitectures/haswell_(client)