Supercomputing in Plain English

Part VIII: Multicore Madness

Henry Neeman, Director

OU Supercomputing Center for Education & Research

University of Oklahoma Information Technology

Tuesday April 14 2009

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 toll free phone bridge to fall back on.

Access Grid

This week's Access Grid (AG) venue: Cactus.
If you aren't sure whether you have AG, you probably don't.

Tue Apr 14	Cactus
Tue Apr 21	Verlet
Tue Apr 28	Cactus
Tue May 5	Titan

Many thanks to John Chapman of U Arkansas for setting these up for us.

H.323 (Polycom etc)

If you want to use H.323 videoconferencing – for example, Polycom – then dial

`69.77.7.203##12345`

any time after 2:00pm. Please connect early, at least today.

For assistance, contact Andy Fleming of KanREN/Kan-ed (afleming@kanren.net or 785-230-2513).

KanREN/Kan-ed's H.323 system can handle up to 40 simultaneous H.323 connections. If you cannot connect, it may be that all 40 are already in use.

Many thanks to Andy and KanREN/Kan-ed for providing H.323 access.

iLinc

We have unlimited simultaneous iLinc connections available.

If you're already on the SiPE e-mail list, then you should receive an e-mail about iLinc before each session begins.

If you want to use iLinc, please follow the directions in the iLinc e-mail.

For iLinc, you **MUST** use either Windows (XP strongly preferred) or MacOS X with Internet Explorer.

To use iLinc, you'll need to download a client program to your PC. It's free, and setup should take only a few minutes.

Many thanks to Katherine Kantardjieff of California State U Fullerton for providing the iLinc licenses.

QuickTime Broadcaster

If you cannot connect via the Access Grid, H.323 or iLinc, then you can connect via QuickTime:

`rtsp://129.15.254.141/test_hpc09.sdp`

We recommend using QuickTime Player for this, because we've tested it successfully.

We recommend upgrading to the latest version at:

`http://www.apple.com/quicktime/`

When you run QuickTime Player, traverse the menus

File -> Open URL

Then paste in the rstp URL into the textbox, and click OK.

Many thanks to Kevin Blake of OU for setting up QuickTime Broadcaster for us.

Phone Bridge

If all else fails, you can call into our toll free phone bridge:

1-866-285-7778, access code 6483137#

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 is charged per connection per minute, so our preference is to minimize the number of connections.

Many thanks to Amy Apon and U Arkansas for providing the toll free phone bridge.

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 some kind of text.

Also, if you're on iLinc: **SIT ON YOUR HANDS!**

Please DON'T touch ANYTHING!


Questions via Text: iLinc or E-mail

Ask questions via text, using one of the following:

- iLinc's text messaging facility;
- e-mail to **sipe2009@gmail.com**.

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

Thanks for helping!

- OSCER operations staff (Brandon George, Dave Akin, Brett Zimmerman, Josh Alexander)
- OU Research Campus staff (Patrick Calhoun, Josh Maxey, Gabe Wingfield)
- Kevin Blake, OU IT (videographer)
- Katherine Kantardjieff, CSU Fullerton
- John Chapman and Amy Apon, U Arkansas
- Andy Fleming, KanREN/Kan-ed
- This material is based upon work supported by the National Science Foundation under Grant No. OCI-0636427, "CI-TEAM Demonstration: Cyberinfrastructure Education for Bioinformatics and Beyond."

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 toll free phone bridge to fall back on.

Supercomputing Exercises

Want to do the "Supercomputing in Plain English" exercises?

- The first several exercises are already posted at:

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

- If you don't yet have a supercomputer account, you can get a temporary account, just for the "Supercomputing in Plain English" exercises, by sending e-mail to:

  hneeman@ou.edu

Please note that this account is for doing the **exercises only**, and will be shut down at the end of the series.

OK Supercomputing Symposium 2009

2003 Keynote:
Peter Freeman
NSF
Computer & Information Science & Engineering
Assistant Director

2004 Keynote:
Sangtae Kim
NSF Shared Cyberinfrastructure Division Director

2005 Keynote:
Walt Brooks
NASA Advanced Supercomputing Division Director

2006 Keynote:
Dan Atkins
Head of NSF's Office of Cyber-infrastructure

2007 Keynote:
Jay Boisseau
Director
Texas Advanced Computing Center
U. Texas Austin

2008 Keynote:
José Munoz
Deputy Office Director/ Senior Scientific Advisor
Office of Cyber-infrastructure
National Science Foundation

2009 Keynote:
Ed Seidel
Director
NSF Office of Cyber-infrastructure

FREE! Wed Oct 7 2009 @ OU

http://symposium2009.oscer.ou.edu/

Parallel Programming Workshop
FREE! Tue Oct 6 2009 @ OU
Sponsored by SC09 Education Program
FREE! Symposium Wed Oct 7 2009 @ OU

SC09 Summer Workshops

This coming summer, the SC09 Education Program, part of the SC09 (Supercomputing 2009) conference, is planning to hold two weeklong supercomputing-related workshops in Oklahoma, for **FREE** (except you pay your own transport):

- **At OSU Sun May 17 – the May 23**:
  FREE Computational Chemistry for Chemistry Educators (2010 TENTATIVE: Computational Biology)
- **At OU Sun Aug 9 – Sat Aug 15**:
  FREE Parallel Programming & Cluster Computing

We'll alert everyone when the details have been ironed out and the registration webpage opens.

Please note that you must **apply** for a seat, and acceptance **CANNOT** be guaranteed.

SC09 Summer Workshops

1. May 17-23: Oklahoma State U: Computational Chemistry
2. May 25-30: Calvin Coll (MI): Intro to Computational Thinking
3. June 7-13: U Cal Merced: Computational Biology
4. June 7-13: Kean U (NJ): Parallel Progrmg & Cluster Comp
5. June 14-20: Widener U (PA): Computational Physics
6. July 5-11: Atlanta U Ctr: Intro to Computational Thinking
7. July 5-11: Louisiana State U: Parallel Progrmg & Cluster Comp
8. July 12-18: U Florida: Computational Thinking Grades 6-12
9. July 12-18: Ohio Supercomp Ctr: Computational Engineering
10. Aug 2- 8: U Arkansas: Intro to Computational Thinking
11. Aug 9-15: U Oklahoma: Parallel Progrmg & Cluster Comp


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

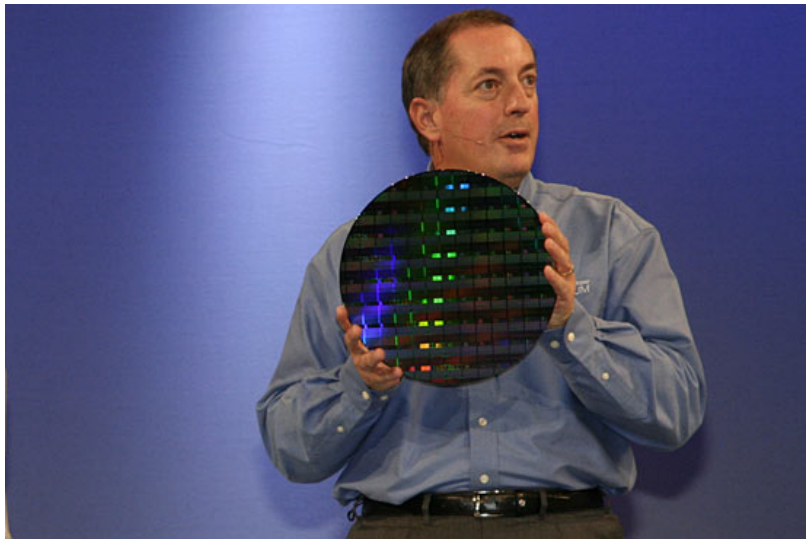
OS: Red Hat Linux

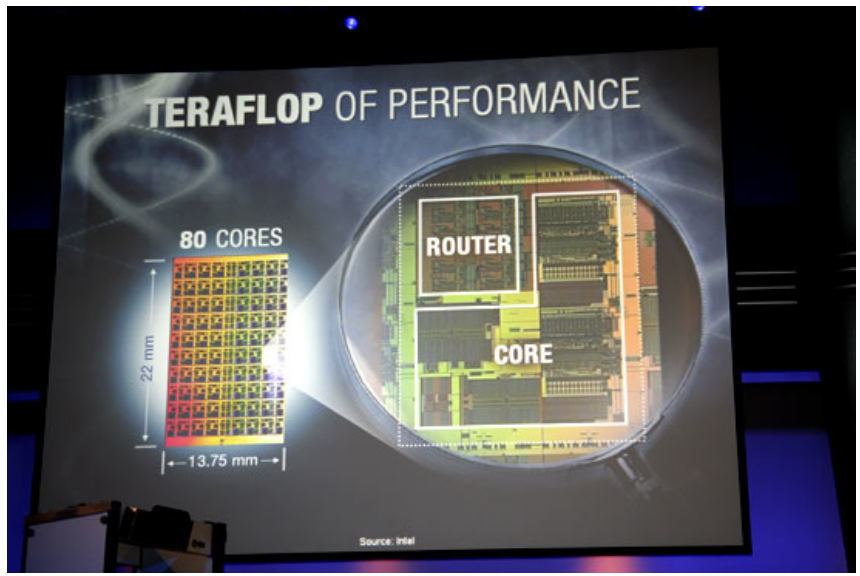
Peak speed: 1.08 TFLOPs
(1.08 trillion calculations per second)

One of the first Pentium4 clusters!

boomer.oscer.ou.edu

TeraFLOP, Prototype 2006, Sale 2011





9 years from room to chip!

http://news.com.com/2300-1006_3-6119652.html

Moore's Law

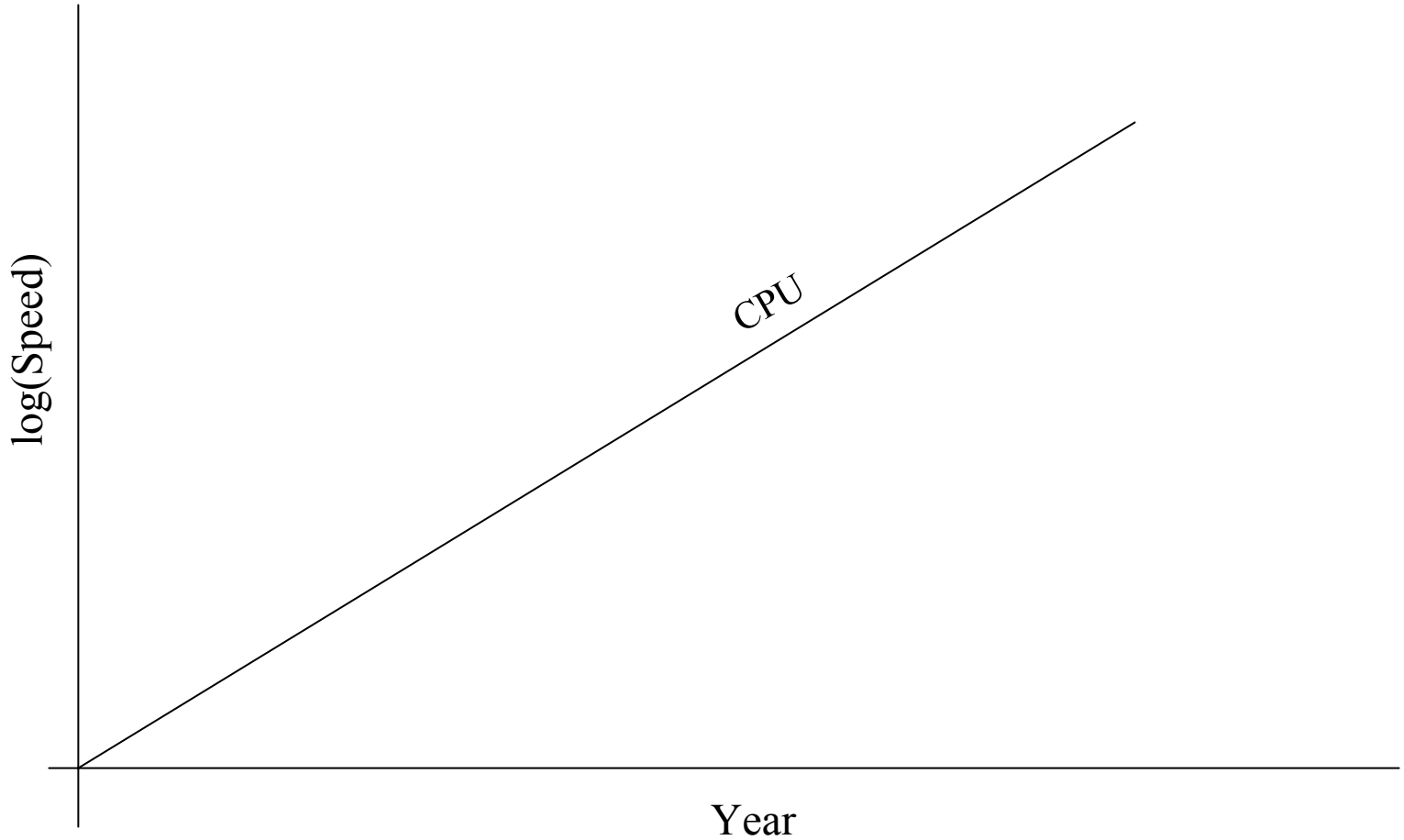
In 1965, Gordon Moore was an engineer at Fairchild Semiconductor.

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

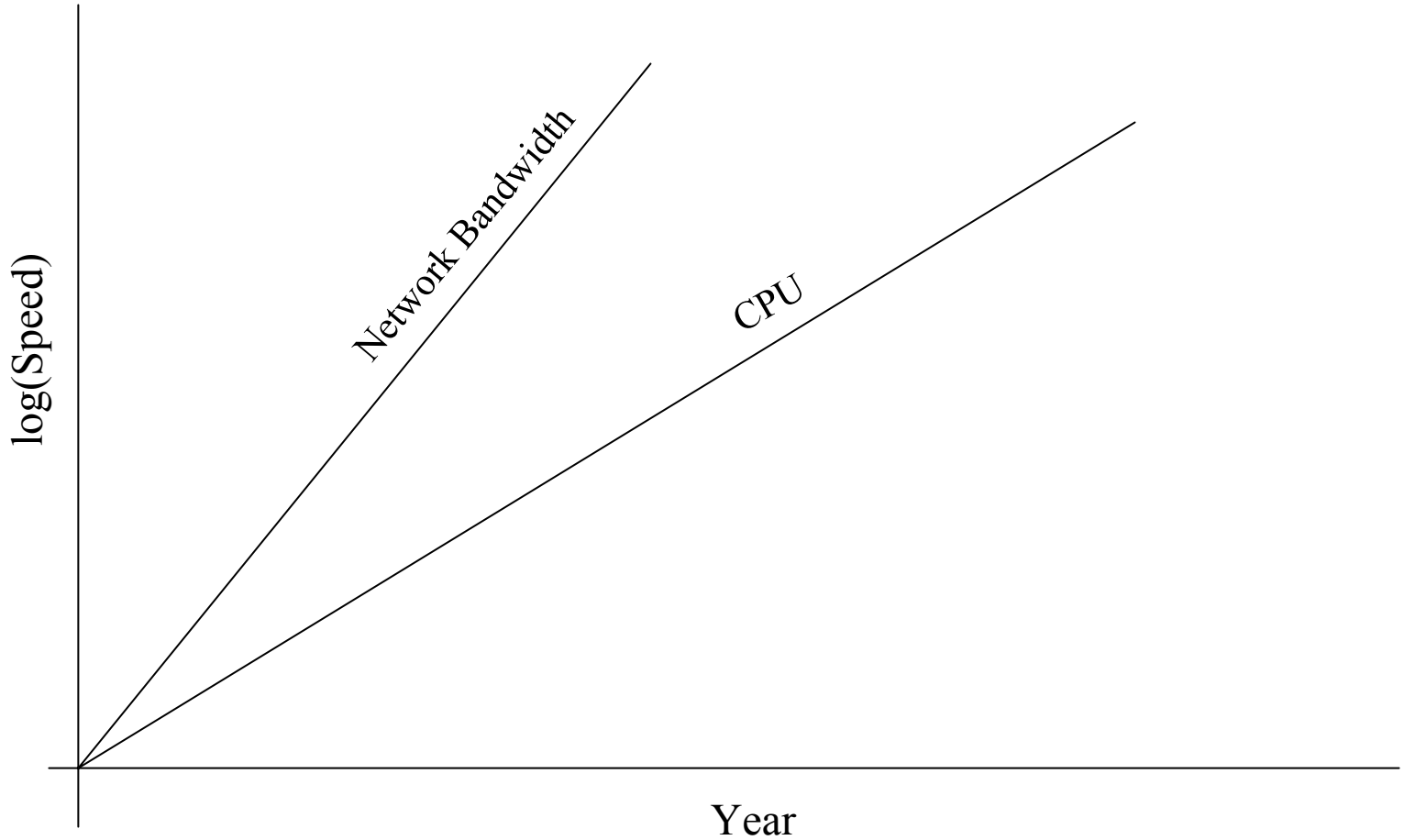
It turns out that computer speed is roughly proportional to the number of transistors per unit area.

Moore wrote a paper about this concept, which became known as ***"Moore's Law."***

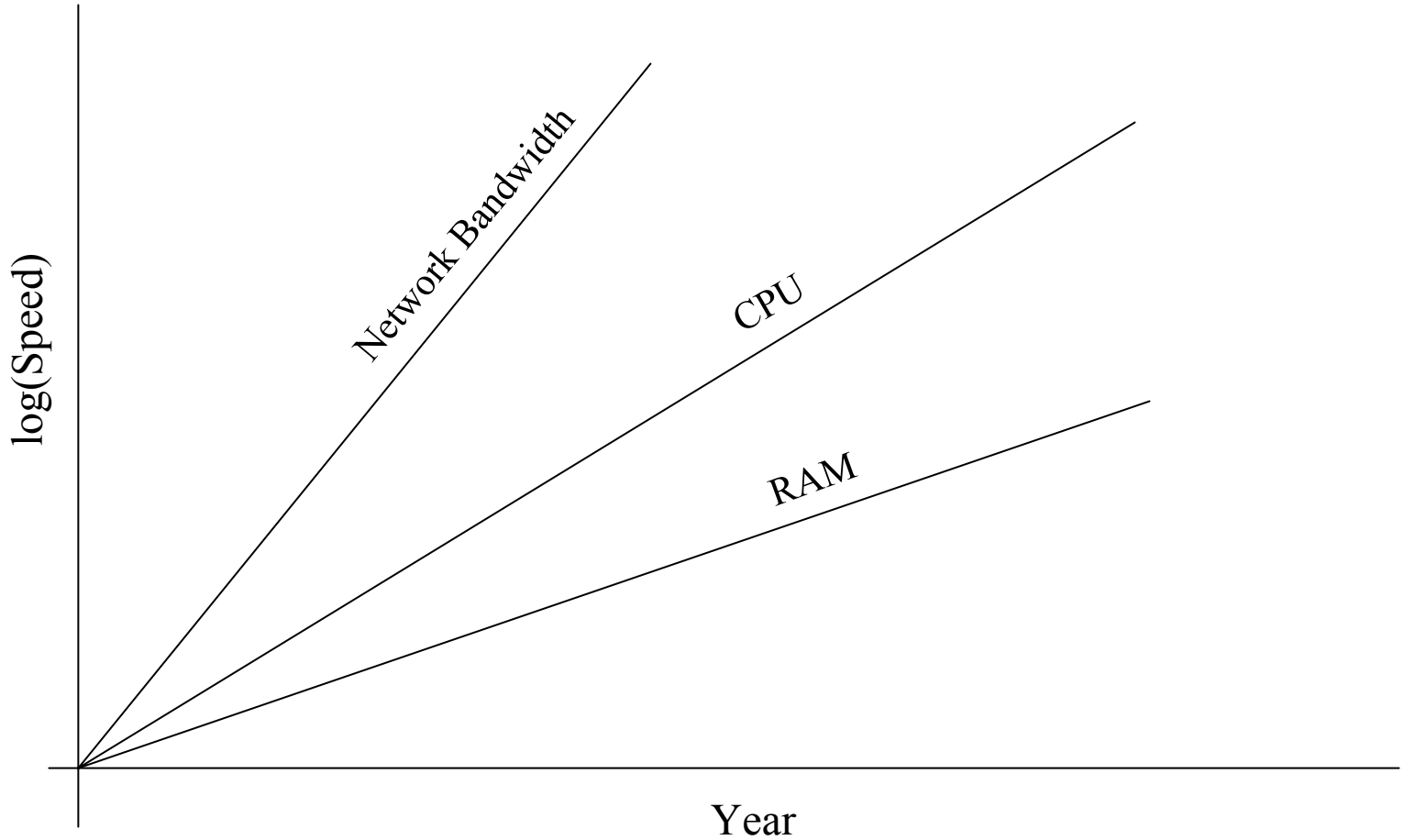
Moore's Law in Practice



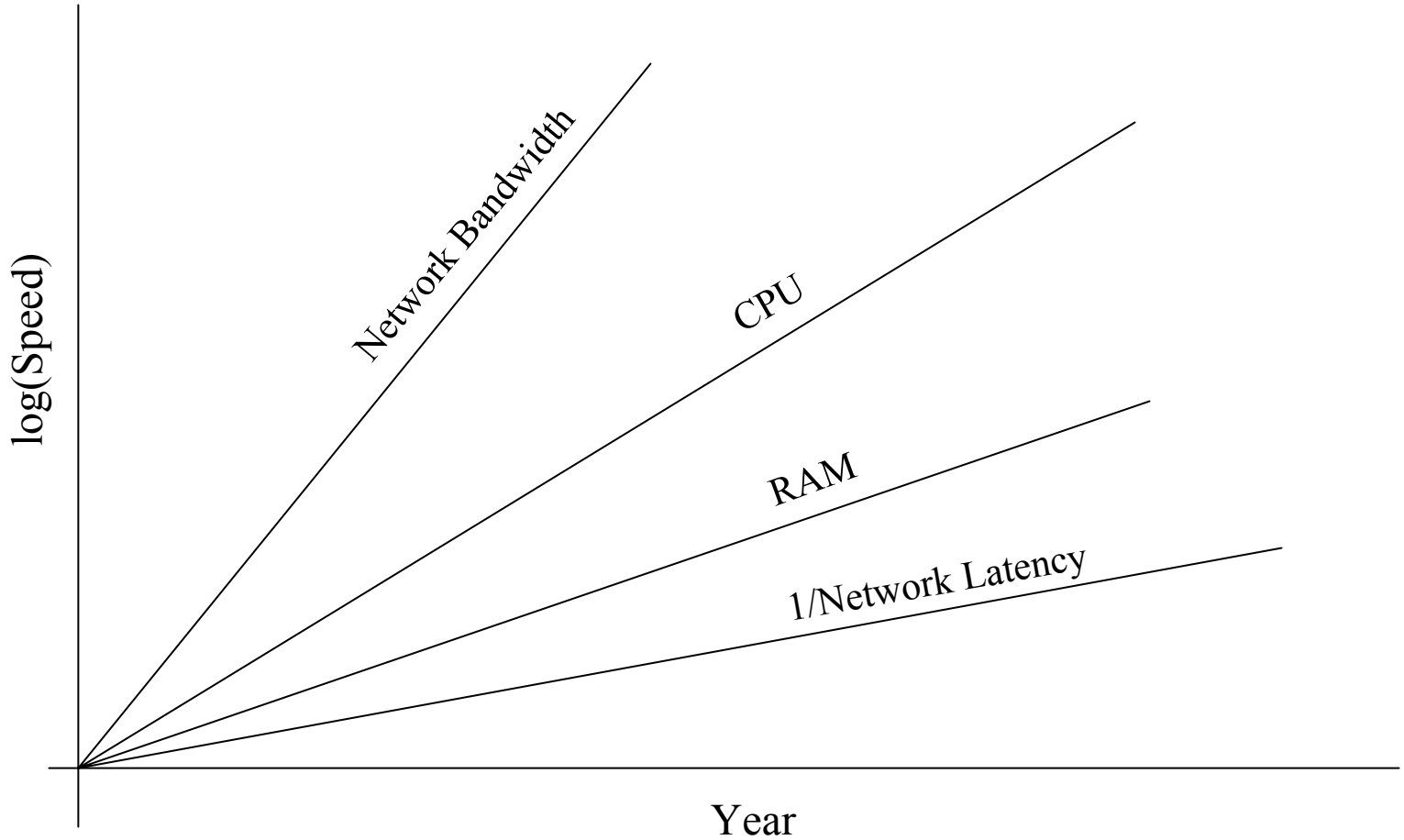
Moore's Law in Practice



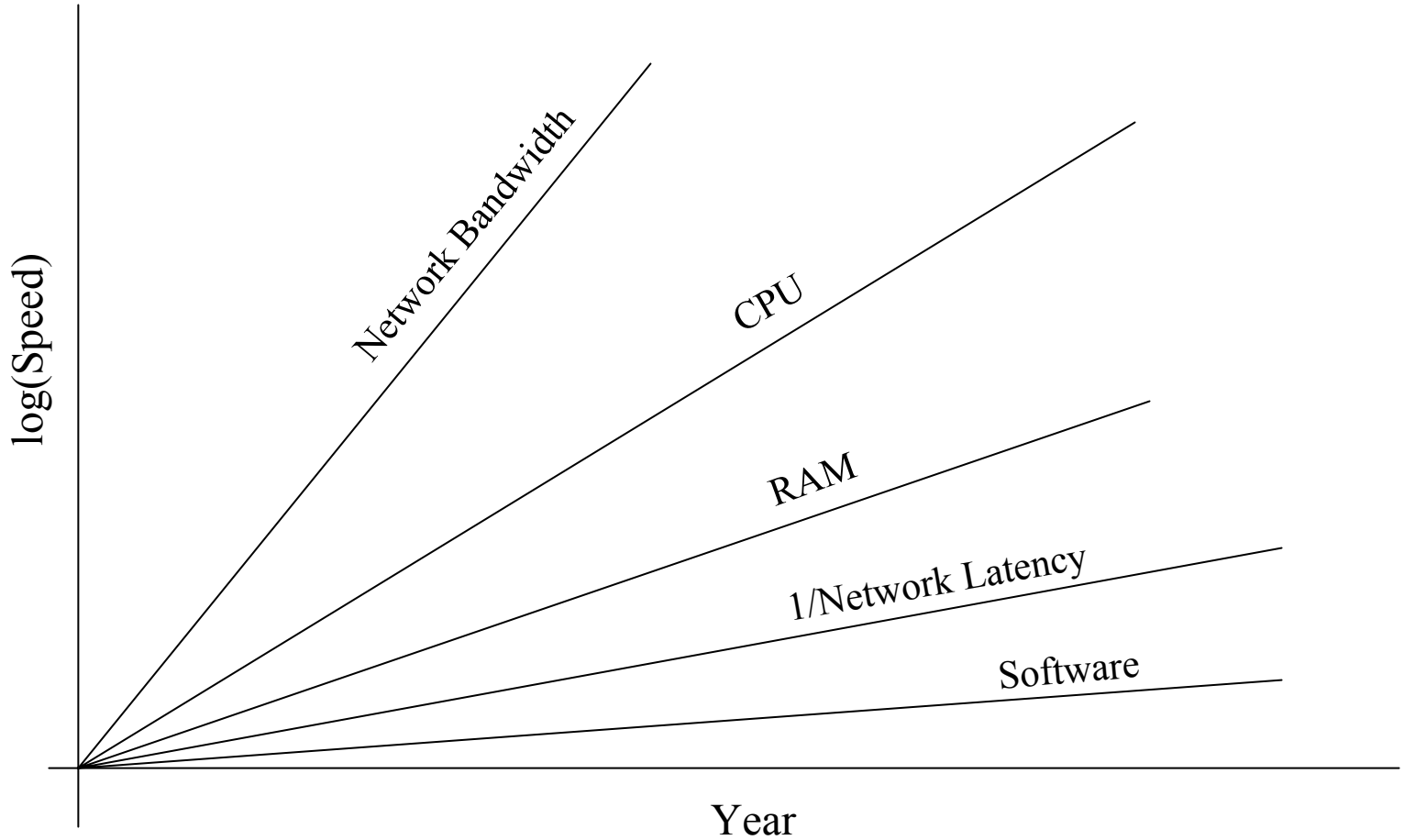
Moore's Law in Practice



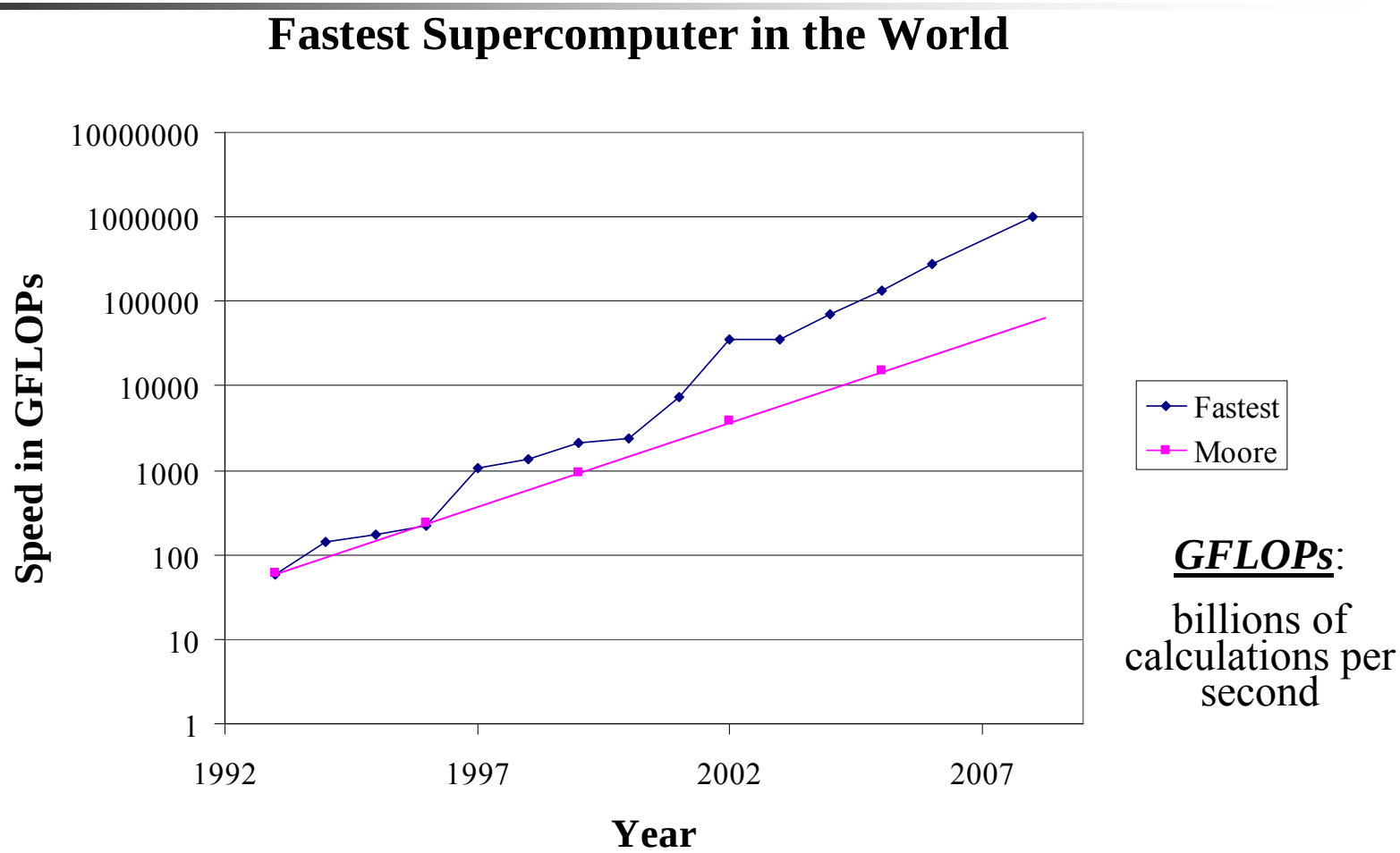
Moore's Law in Practice



Moore's Law in Practice



Fastest Supercomputer vs. Moore



GFLOPs:

billions of calculations per second

The Tyranny of the Storage Hierarchy

The Storage Hierarchy

Fast, expensive, few

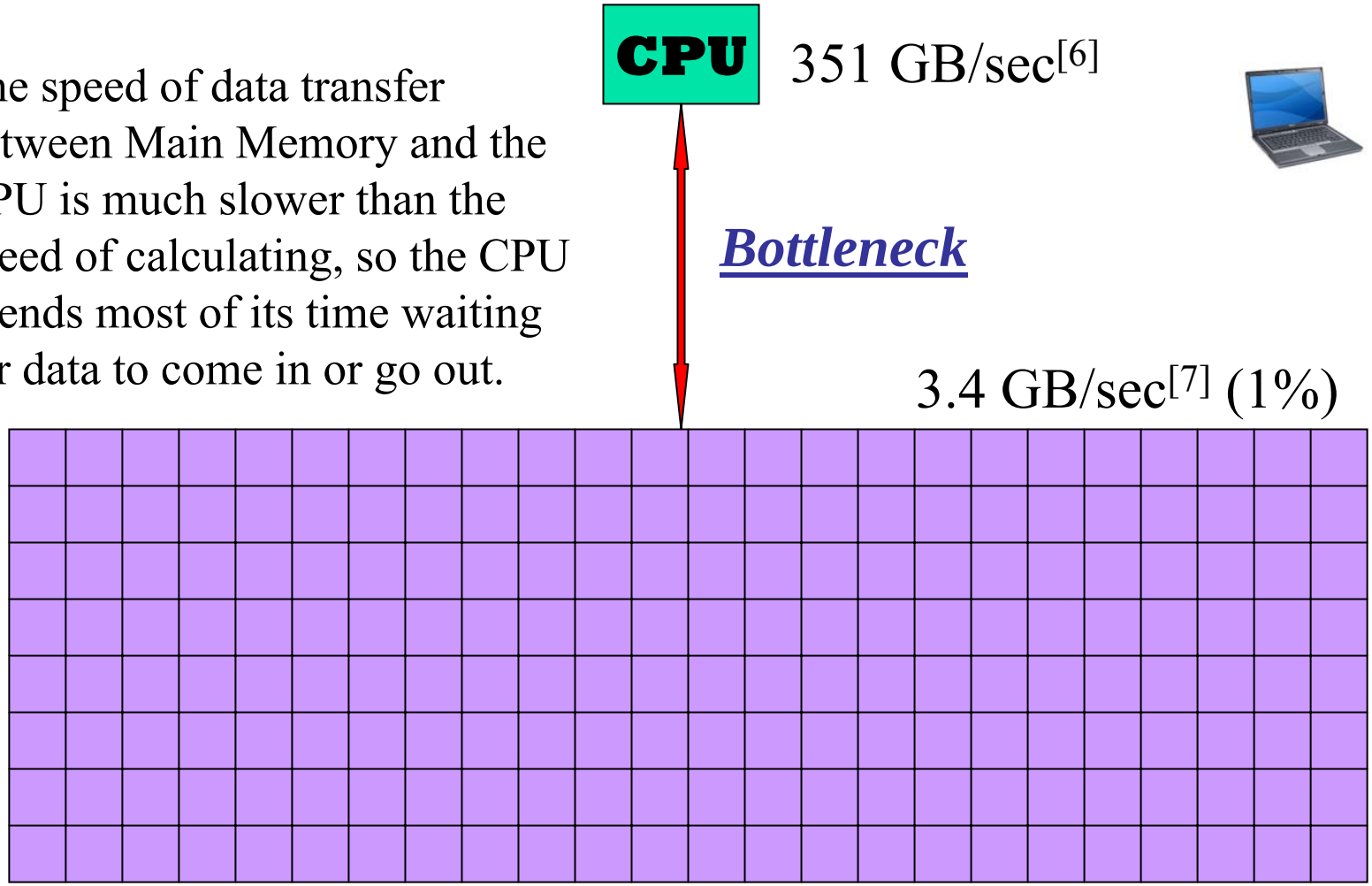
Slow, cheap, a lot

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

[5]

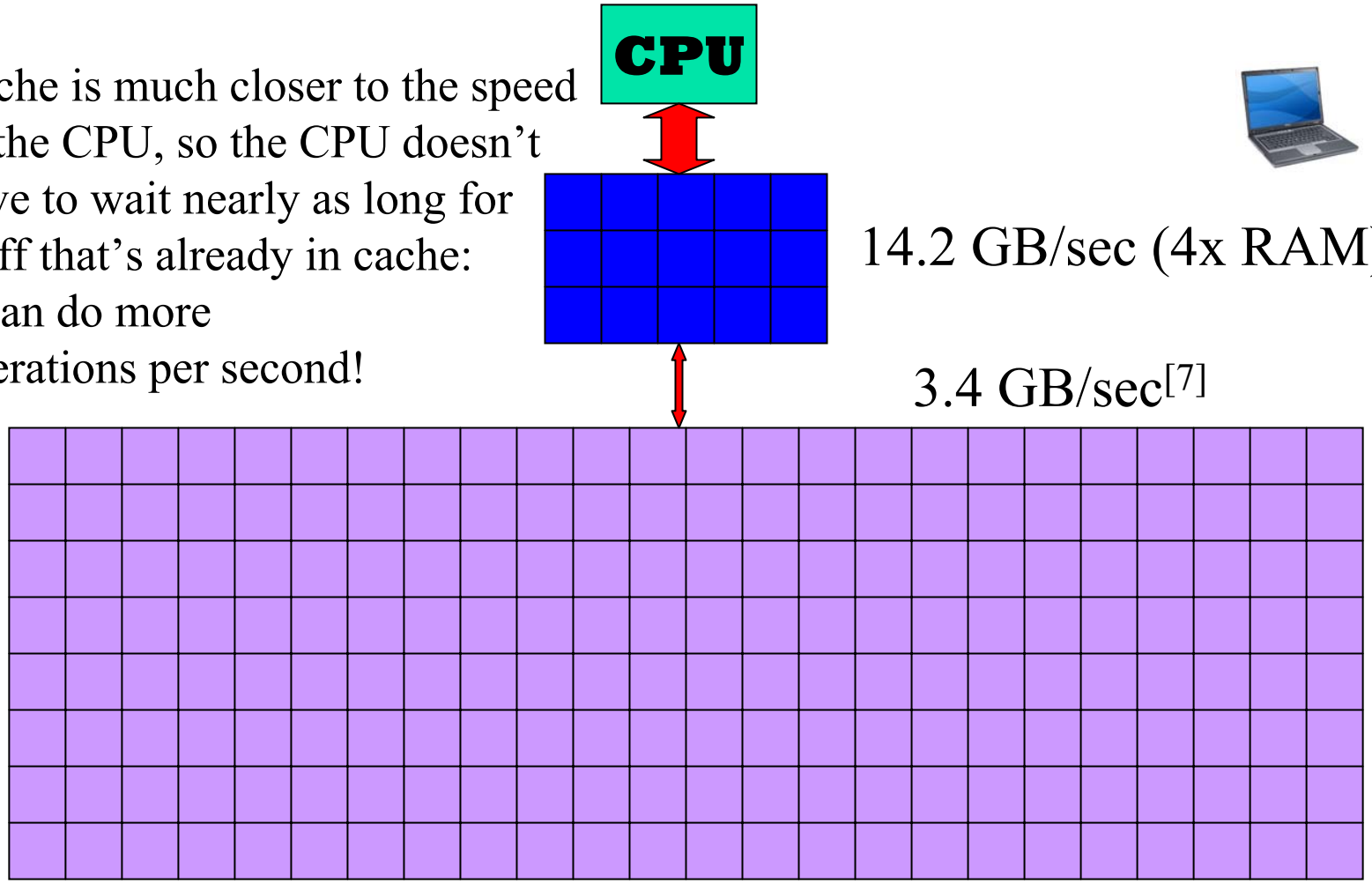
RAM is Slow

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 D620[4]

- Pentium 4 Core Duo T2400
  1.83 GHz w/2 MB L2 Cache
  ("Yonah")
- 2 GB (2048 MB)
  667 MHz DDR2 SDRAM
- 100 GB 7200 RPM SATA Hard Drive
- DVD±RW/CD-RW Drive (8x)
- 1 Gbps Ethernet Adapter
- 56 Kbps Phone Modem

Storage Speed, Size, Cost

Henry's Laptop	Registers (Pentium 4 Core Duo 1.83 GHz)	Cache Memory (L2)	Main Memory (667 MHz DDR2 SDRAM)	Hard Drive (SATA 7200 RPM)	Ethernet (1000 Mbps)	DVD±RW (8x)	Phone Modem (56 Kbps)
Speed (MB/sec) [peak]	359,792[6] (14,640 MFLOP/s*)	14,500 [7]	3400 [7]	100 [9]	125	10.8 [10]	0.007
Size (MB)	304 bytes** [11]	2	2048	100,000	unlimited	unlimited	unlimited
Cost ($/MB)	–	$5 [12]	$0.03 [12]	$0.0001 [12]	charged per month (typically)	$0.00003 [12]	charged per month (typically)

* MFLOP/s: millions of floating point operations per second

** 8 32-bit integer registers, 8 80-bit floating point registers, 8 64-bit MMX integer registers, 8 128-bit floating point XMM 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

- OSCER's big cluster, Sooner, has Harpertown CPUs: quad core, 2.0 GHz, 1333 MHz Front Side Bus.
- The theoretical peak CPU speed is 32 GFLOPs (double precision) per CPU, and in practice we've gotten as high as 93% of that. For a dual chip node, the peak is 64 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 transfer up to 1536 GB/sec.
- The theoretical peak RAM bandwidth is 21 GB/sec (but in practice we get about 3.4 GB/sec).
- So, even at theoretical peak, any code that does less than 73 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 $A = B \bullet C$ is

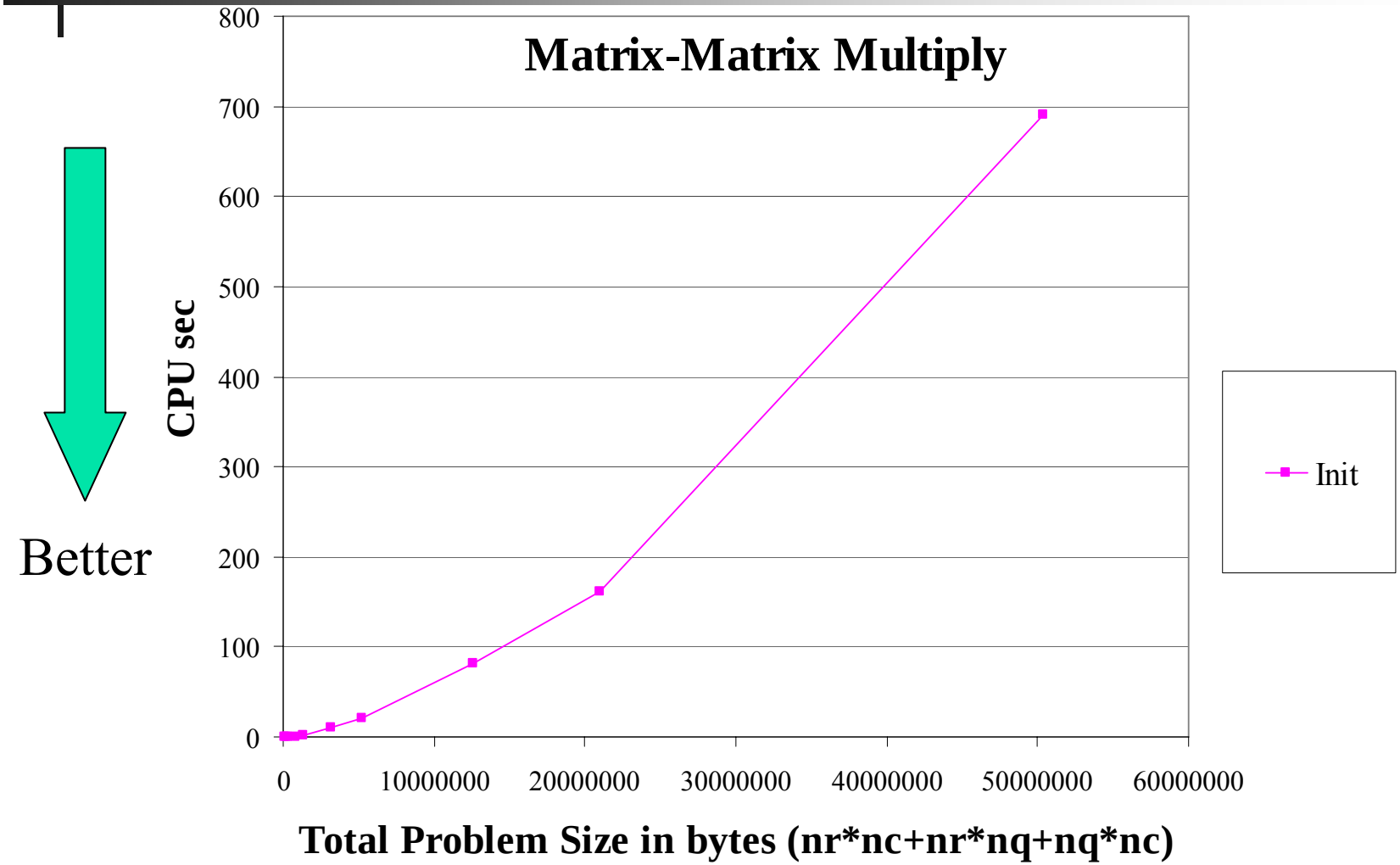
$$a_{r,c} = \sum_{k=1}^{nk} b_{r,k} \cdot c_{k,c} = b_{r,1} \cdot c_{1,c} + b_{r,2} \cdot c_{2,c} + b_{r,3} \cdot c_{3,c} + \ldots + b_{r,nk} \cdot c_{nk,c}$$

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

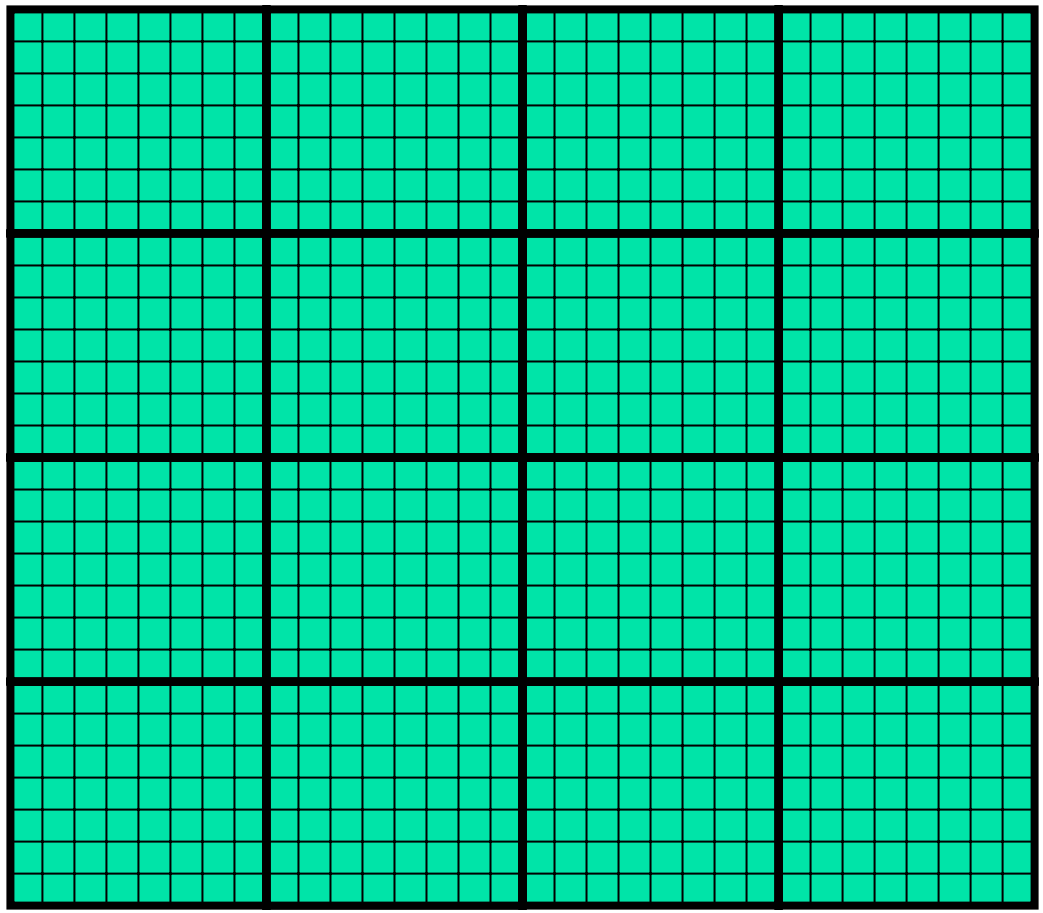
Matrix Multiply: Naïve Version

```
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

```
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
    END DO
  END DO
END SUBROUTINE matrix_matrix_mult_by_tiling
```

Multiplying Within a Tile

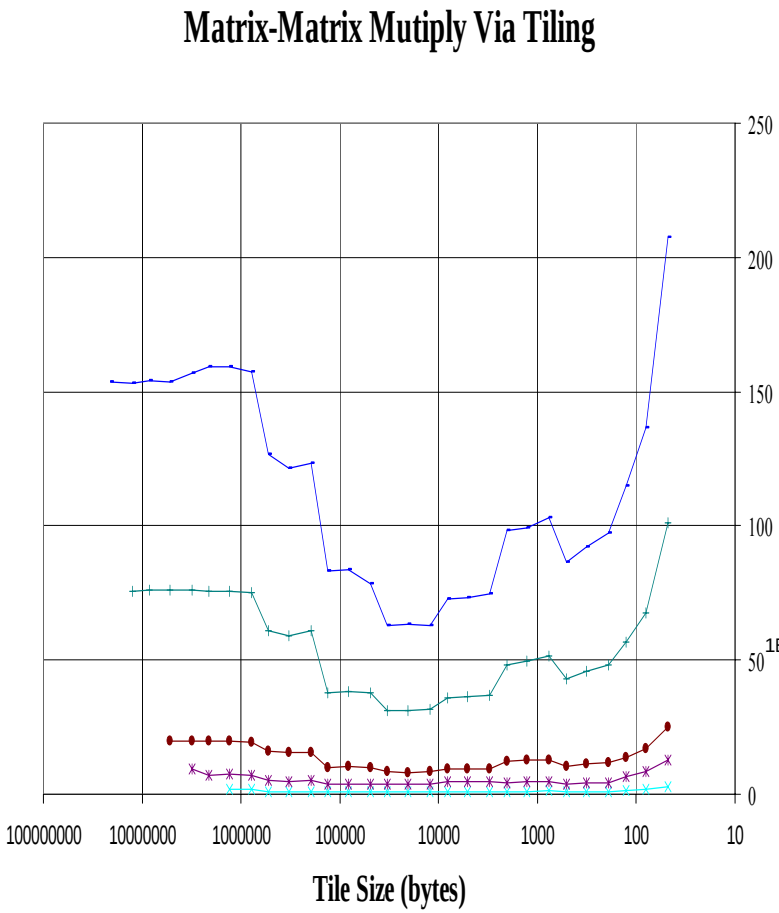
```
SUBROUTINE matrix_matrix_mult_tile (dst, src1, src2, nr, nc, nq, &
 &             rstart, rend, cstart, cend, qstart, qend)
  IMPLICIT NONE
  INTEGER,INTENT(IN) :: nr, nc, nq
  REAL,DIMENSION(nr,nc),INTENT(OUT) :: dst
  REAL,DIMENSION(nr,nq),INTENT(IN)  :: src1
  REAL,DIMENSION(nq,nc),INTENT(IN)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

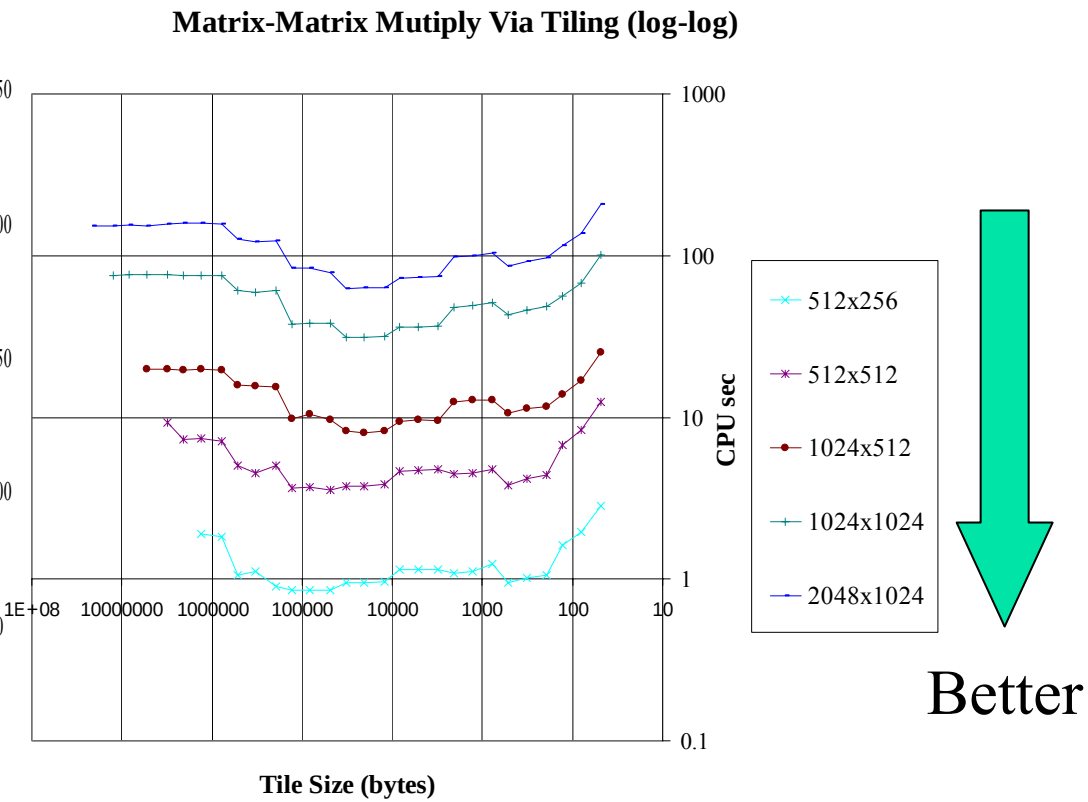


Reminder: Naïve Version, Again

```
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 with Tiling





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.

Why Does Tiling Work Here?

Cache optimization works best when the number of calculations per byte is large.

For example, with matrix-matrix multiply on an $n \times n$ matrix, there are $\mathbf{O}(n^3)$ calculations (on the order of n^3), but only $\mathbf{O}(n^2)$ bytes of data.

So, for large n, there are a huge number of calculations per byte transferred between RAM and cache.

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.

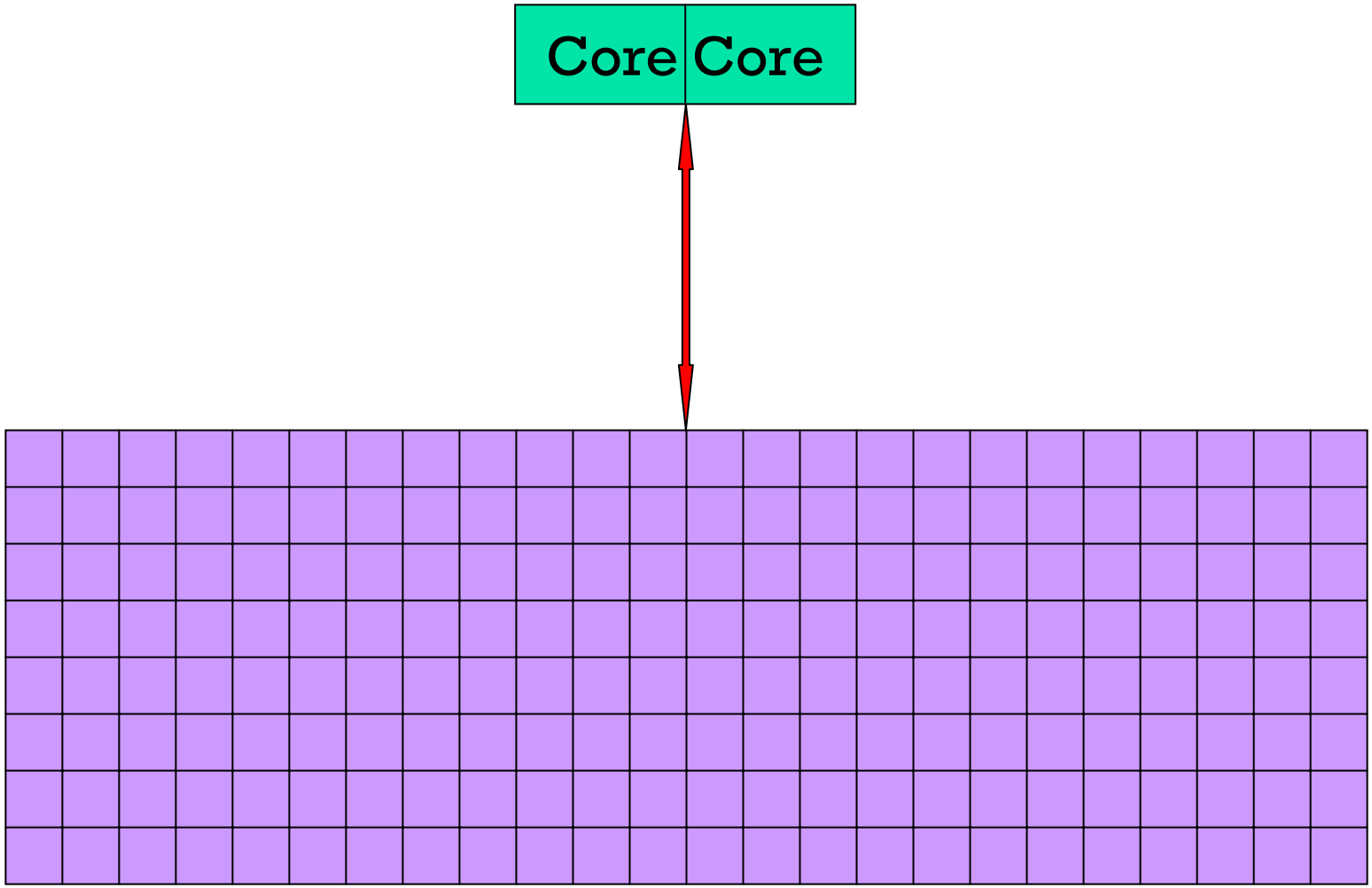
If either condition is absent, then tiling won't help.

Multicore/Many-core Basics

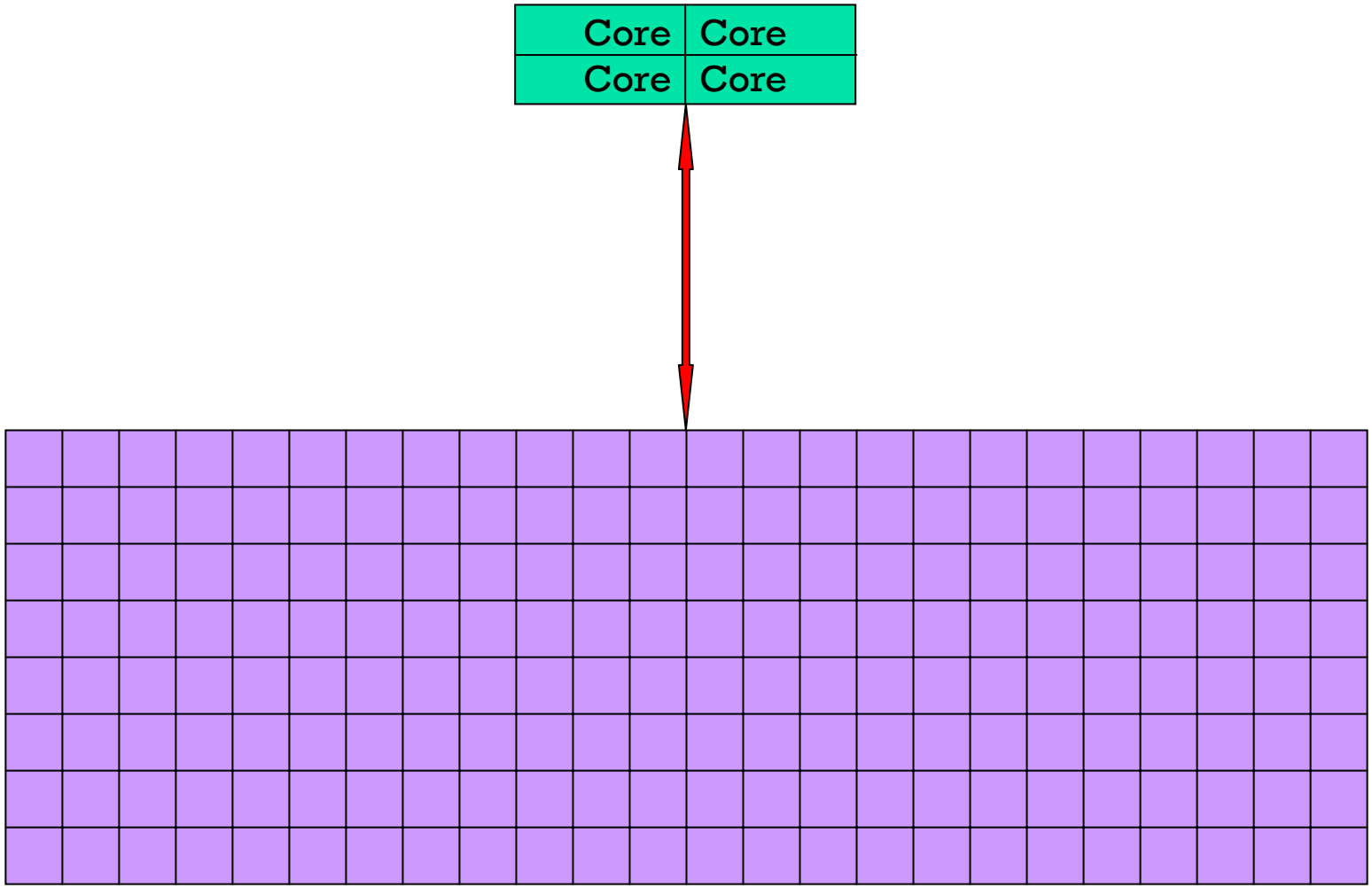
What is Multicore?

- In the olden days (that is, the first half of 2005), each CPU chip had one "brain" in it.
- Starting the second half of 2005, each CPU chip has 2 ***cores*** (brains); starting in late 2006, 4 cores; starting in late 2008, 6 cores; expected in late 2009, 8 cores.
- **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 with one or more other cores – and therefore shares its bandwidth to RAM.

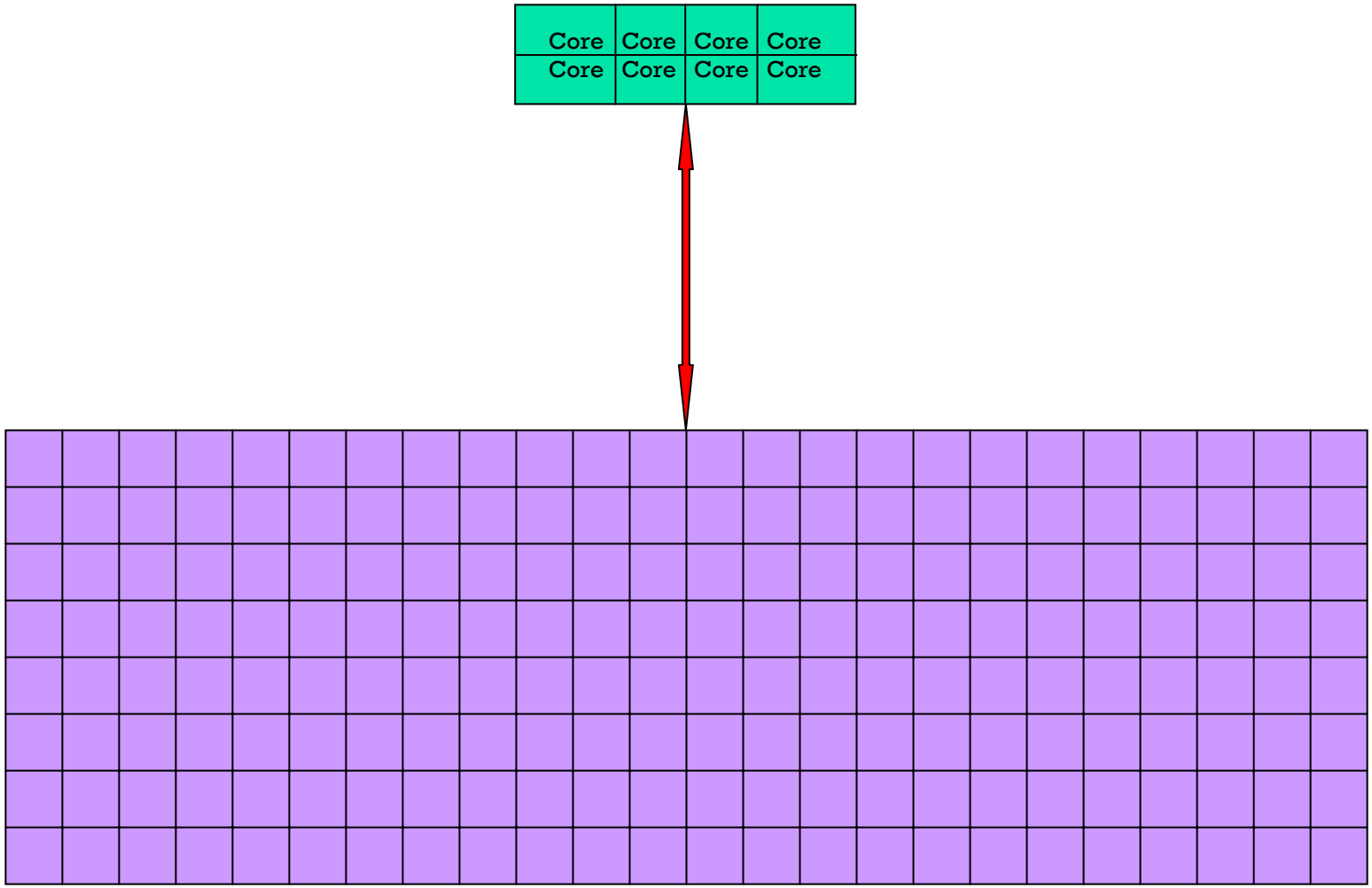
Dual Core



Quad Core



Oct Core



The Challenge of Multicore: RAM

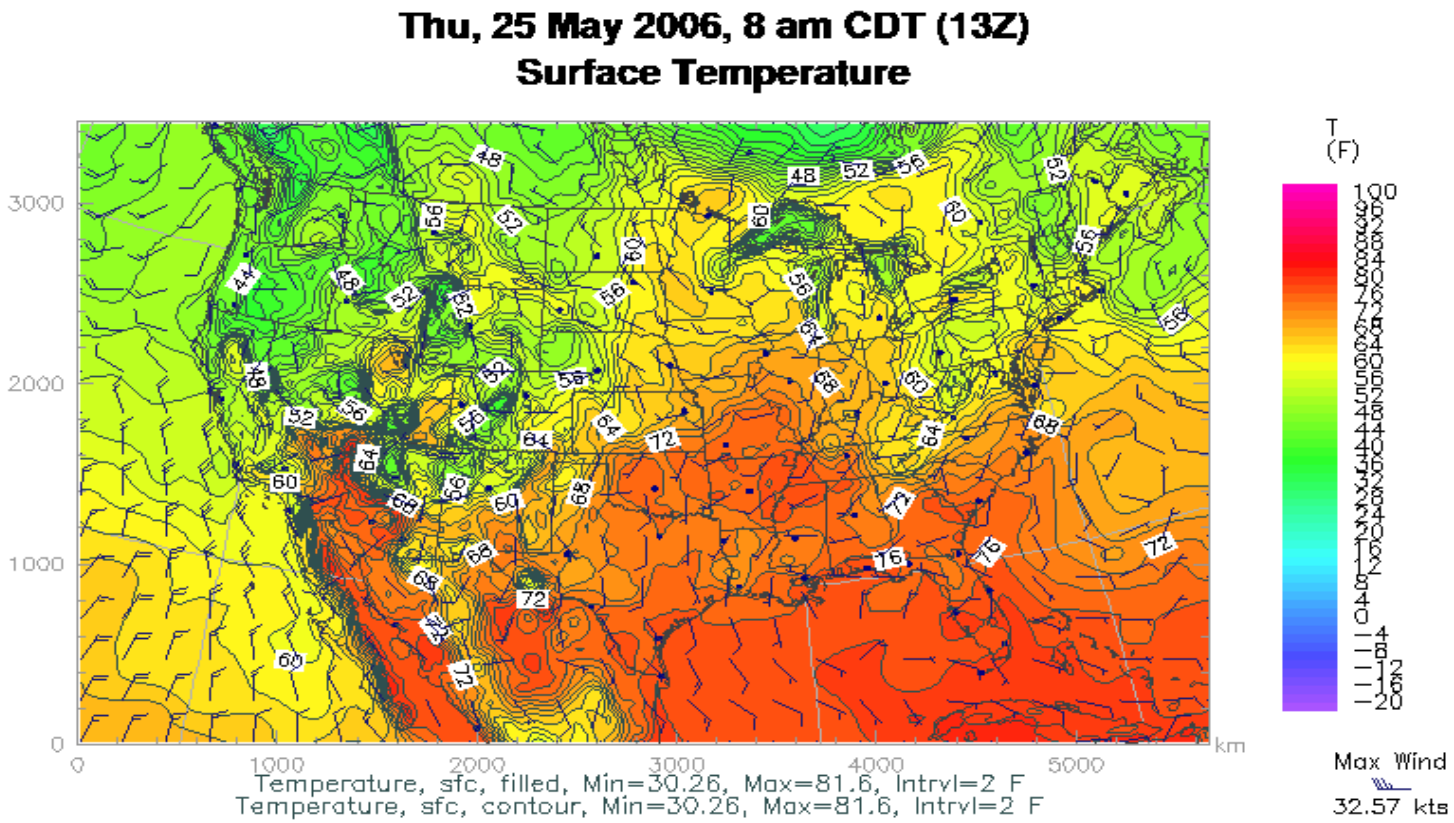
- Each socket has access to a certain amount of RAM, at a **fixed RAM bandwidth per SOCKET** – or even per node.
- As the number of cores per socket increases, the **contention for RAM bandwidth increases** too.
- At 2 or even 4 cores in a socket, this problem isn't too bad. But at 16 or 32 or 80 cores, it's **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 80 cores, it's **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** are going to speed up very very little – and may even crash the node.

A Concrete Example: Weather Forecasting

Weather Forecasting

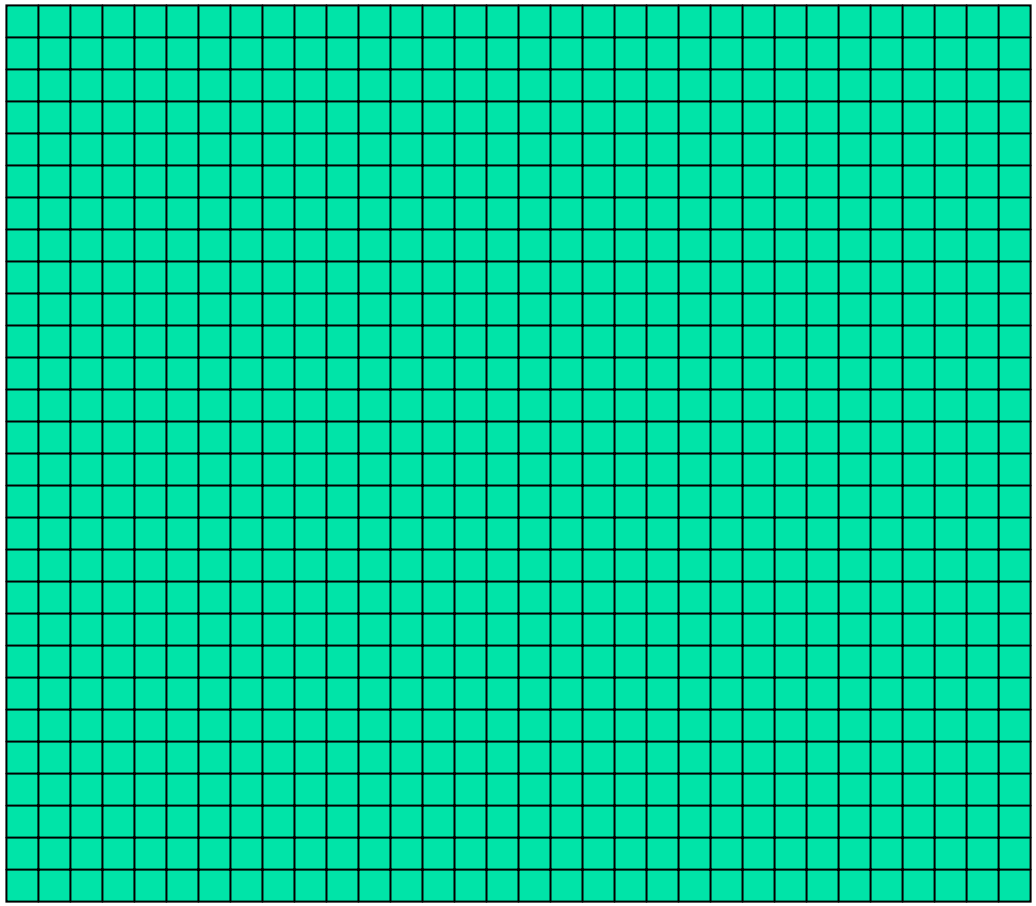


http://www.caps.ou.edu/wx/p/r/conus/fcst/

Weather Forecasting

- Weather forecasting is a **transport** 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.

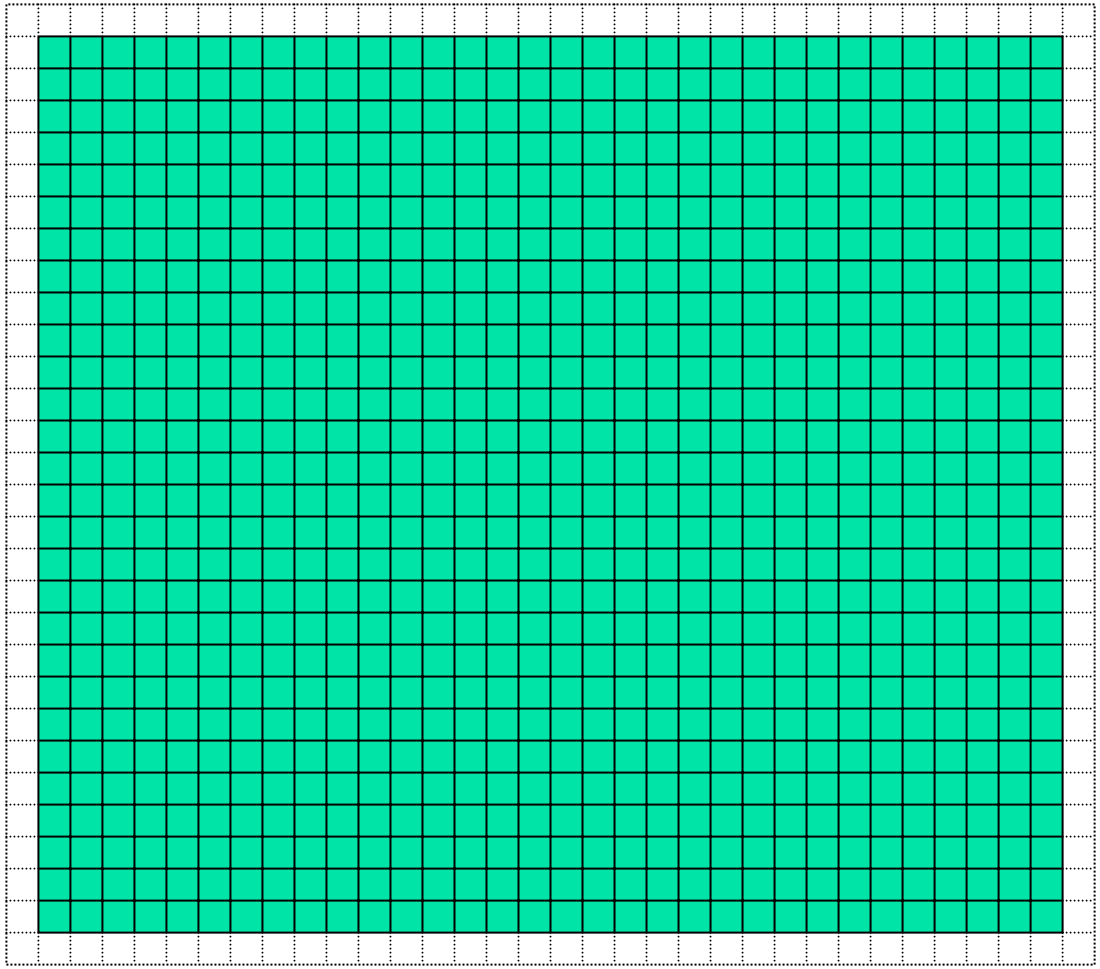
Cartesian Mesh



Finite Difference

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

Ghost Boundary Zones



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 73 to 1 needed (on mid-2008 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?

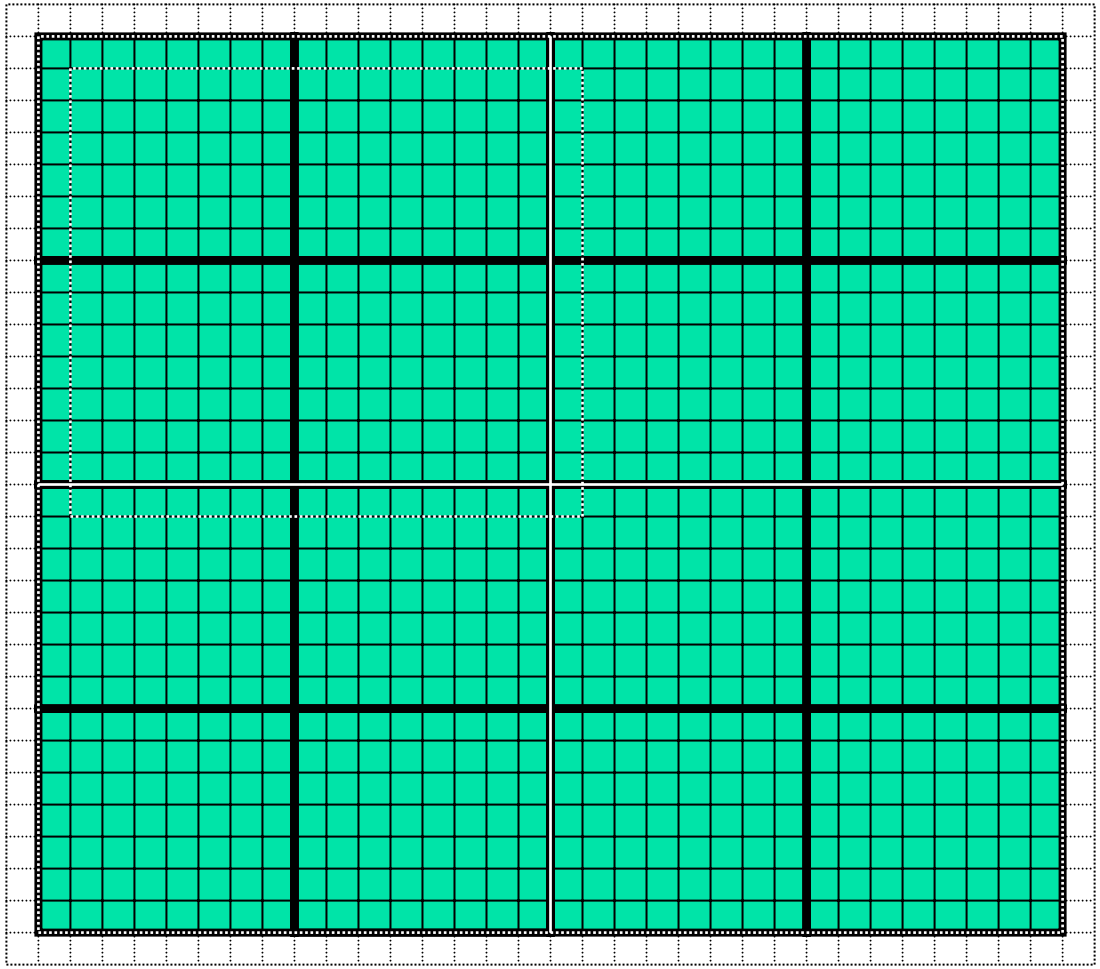
- Multiple independent subdomains per processor.
- Each subdomain fits entirely in L2 cache.
- Each subdomain's page table entries fit entirely in the TLB.
- Expanded ghost zone stencil allows multiple timesteps before communicating with neighboring subdomains.
- Parallelize along the Z-axis as well 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 15000 or 20000.
- So, you can better amortize the cost of transferring the data between RAM and cache.

New Algorithm

```
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
```

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, 2 MB) – and this problem seems likely to remain.
- Small TLBs make the problem worse: 512 KB per core rather than 3 MB.
- So, to get good cache reuse, you need subdomains of no more than 512 KB.
- If you have 150 3D variables at single precision, and 100 zones in Z, then your horizontal size will be 3 x 3 zones – just enough for your stencil!

What Do We Need?

- We need much bigger caches!
  - 16 MB cache ➔ 16 x 16 horizontal including stencil
  - 32 MB cache ➔ 23 x 23 horizontal including stencil
- 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!

OK Supercomputing Symposium 2009

2003 Keynote:
Peter Freeman
NSF
Computer & Information Science & Engineering
Assistant Director

2004 Keynote:
Sangtae Kim
NSF Shared Cyberinfrastructure Division Director

2005 Keynote:
Walt Brooks
NASA Advanced Supercomputing Division Director

2006 Keynote:
Dan Atkins
Head of NSF's Office of Cyber-infrastructure

2007 Keynote:
Jay Boisseau
Director
Texas Advanced Computing Center
U. Texas Austin

2008 Keynote:
José Munoz
Deputy Office Director/ Senior Scientific Advisor
Office of Cyber-infrastructure
National Science Foundation

2009 Keynote:
Ed Seidel
Director
NSF Office of Cyber-infrastructure

FREE! Wed Oct 7 2009 @ OU

http://symposium2009.oscer.ou.edu/

Parallel Programming Workshop
FREE! Tue Oct 6 2009 @ OU
Sponsored by SC09 Education Program
FREE! Symposium Wed Oct 7 2009 @ OU

SC09 Summer Workshops

This coming summer, the SC09 Education Program, part of the SC09 (Supercomputing 2009) conference, is planning to hold two weeklong supercomputing-related workshops in Oklahoma, for **FREE** (except you pay your own transport):

- **At OSU Sun May 17 – the May 23**:
  FREE Computational Chemistry for Chemistry Educators (2010 TENTATIVE: Computational Biology)
- **At OU Sun Aug 9 – Sat Aug 15**:
  FREE Parallel Programming & Cluster Computing

We'll alert everyone when the details have been ironed out and the registration webpage opens.

Please note that you must **apply** for a seat, and acceptance **CANNOT** be guaranteed.

SC09 Summer Workshops

1. May 17-23: Oklahoma State U: Computational Chemistry
2. May 25-30: Calvin Coll (MI): Intro to Computational Thinking
3. June 7-13: U Cal Merced: Computational Biology
4. June 7-13: Kean U (NJ): Parallel, Distributed & Grid
5. June 14-20: Widener U (PA): Computational Physics
6. July 5-11: Atlanta U Ctr: Intro to Computational Thinking
7. July 5-11: Louisiana State U: Parallel, Distributed & Grid
8. July 12-18: U Florida: Computational Thinking Pre-college
9. July 12-18: Ohio Supercomp Ctr: Computational Engineering
10. Aug 2- 8: U Arkansas: Intro to Computational Thinking
11. Aug 9-15: U Oklahoma: Parallel, Distributed & Grid

To Learn More Supercomputing

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

Thanks for helping!

- OSCER operations staff (Brandon George, Dave Akin, Brett Zimmerman, Josh Alexander)
- OU Research Campus staff (Patrick Calhoun, Josh Maxey, Gabe Wingfield)
- Kevin Blake, OU IT (videographer)
- Katherine Kantardjieff, CSU Fullerton
- John Chapman and Amy Apon, U Arkansas
- Andy Fleming, KanREN/Kan-ed
- This material is based upon work supported by the National Science Foundation under Grant No. OCI-0636427, "CI-TEAM Demonstration: Cyberinfrastructure Education for Bioinformatics and Beyond."

Thanks for your attention!

Questions?

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/