Parallel Computing Concepts

Le Yan

Scientific Computing Consultant

LONI/LSU HPC

CENTER FOR COMPUTATION & TECHNOLOGY

Outline

- Introduction
- Parallel programming models
- Parallel programming hurdles
- Heterogeneous computing

CENTER FOR COMPUTATION
& TECHNOLOGY

Why parallel computing

- Parallel computing might be the only way to achieve certain goals
  - Problem size (memory, disk etc.)
  - Time needed to solve problems
- Parallel computing allows us to take advantage of ever-growing parallelism at all levels
  - Multi-core, many-core, cluster, grid, cloud…

CENTER FOR COMPUTATION & TECHNOLOGY

Latest Top 500 List

- Released on Monday (6/20/11)
- Japan claims the top spot, again
  - Built by Fujitsu
  - 8 PetaFLOPS (10^{15}) sustained
  - More than half million cores
  - Power close to 10 MW
- Only one US machine in the top 5 for the first time in 5 year (in history?)

CENTER FOR COMPUTATION & TECHNOLOGY

Supercomputing on a cell phone?

- Quad-core processors are coming to your phone
  - Nvidia, TI, QualComm…
  - Processing power in the neighborhood of 10 GigaFLOPS
  - Would make the top 500 list 15 years ago

CENTER FOR COMPUTATION
& TECHNOLOGY

What is parallel computing

- Multiple processing units work together to solve a task
  - The processing units can be tightly or loosely coupled
  - Not every part of the task is parallelizable
  - In most cases, communication among processing units is necessary for the purpose of coordination
- Embarrassingly Parallel
  - Subtasks are independent, therefore communication is unnecessary

CENTER FOR COMPUTATION
& TECHNOLOGY

An example of parallel computing (not really)

- A group of people move a pile of boxes from location A to location B
- The benefit of going parallel: for a fixed number of boxes, more workers mean less time

CENTER FOR COMPUTATION & TECHNOLOGY

Evaluating parallel programs (1)

- Speedup
  - Probably the most import metric (that matters)
  - Let N_{proc} be the number of parallel processes
  - Speedup (N_{proc}) = $\frac{Time\ used\ by\ best\ serial\ program}{Time\ used\ by\ parallel\ program}$
  - Between 0 and N_{proc} (for most cases)
- Efficiency
  - Efficiency(N_{proc})=Speedup/N_{proc}
  - Between 0 and 1

CENTER FOR COMPUTATION & TECHNOLOGY

Evaluating parallel programs (2)

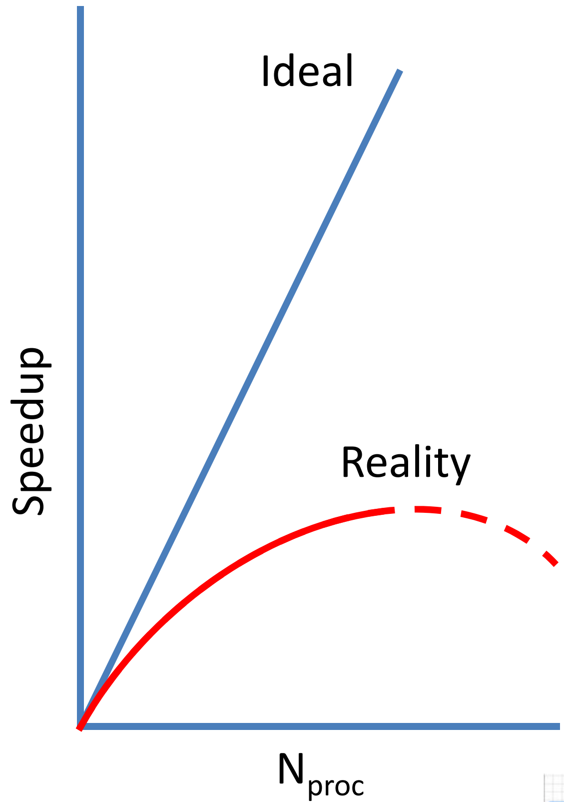
- For our box moving example
  - Assuming we have 20 boxes total and it takes 1 minute for 1 worker to move 1 box, ideally we will see:

Number of workers	Time used (minutes)	Speedup	Efficiency
1	20	1	1
2	10	2	1
5	4	5	1
10	2	10	1
20	1	20	1
40	0.5? 1?	?	?
…	?	?	?

CENTER FOR COMPUTATION & TECHNOLOGY

Speedup as a function of N_{proc}

- Ideally
  - The speedup will be linear
- Even better
  - (in very rare cases) we can have superlinear speedup
- But in reality
  - Efficiency decreases with increasing number of processes

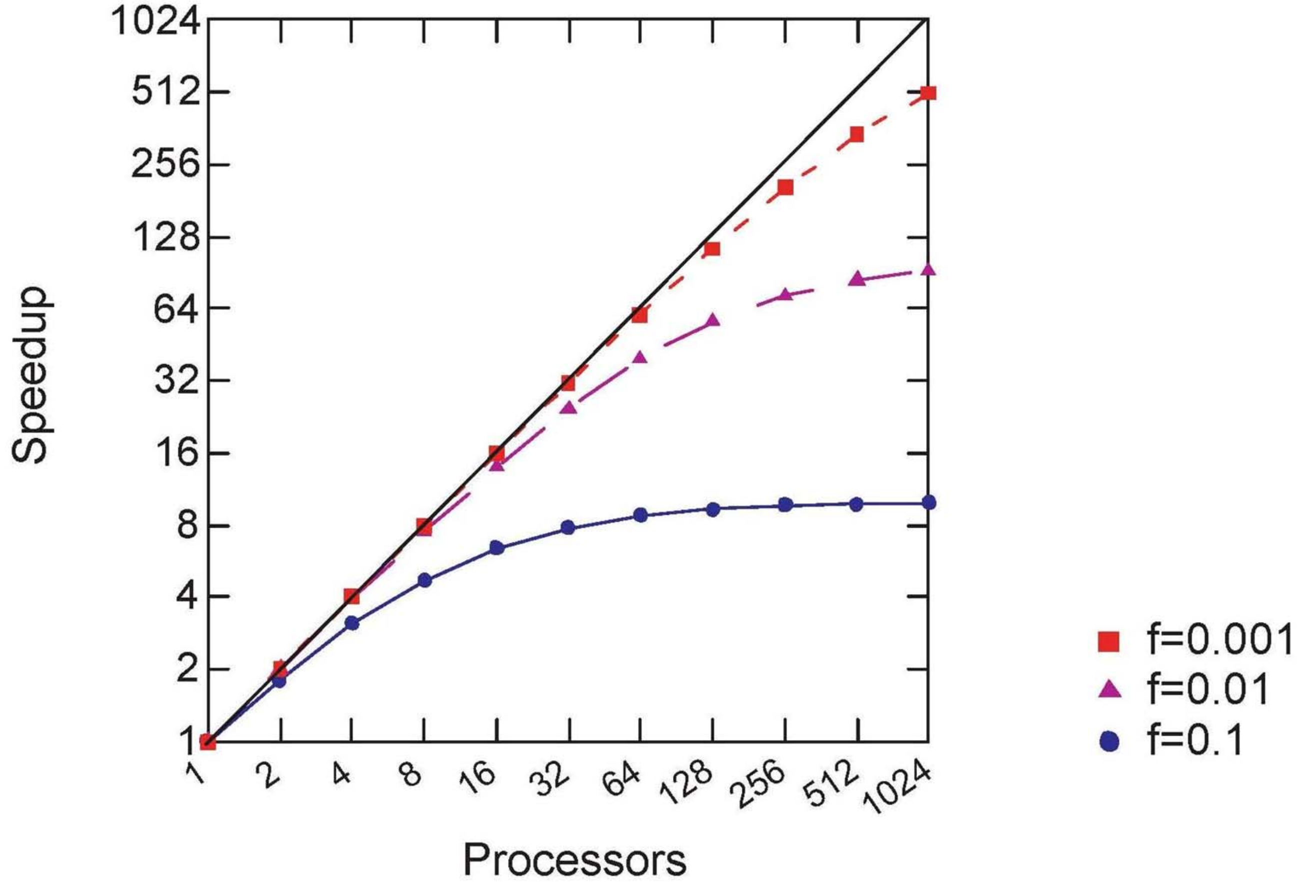


Amdahl's law (1)

- Let f be the fraction of the serial program that cannot be parallelized
- Assume that the rest of the serial program can be perfectly parallelized (linear speedup)
- Then
  - $Time_{parallel} = Time_{serial} \cdot (f + \frac{1-f}{N_{proc}})$
- Or
  - $Speedup = \frac{1}{f+\frac{1-f}{N_{proc}}} \leq \frac{1}{f}$

CENTER FOR COMPUTATION & TECHNOLOGY

Maximal Possible Speedup



Source: Stout & Jablonowski, Parallel computing 101, SC10

CENTER FOR COMPUTATION
& TECHNOLOGY

Amdahl's law (2)

- What Amdahl's law says
  - It puts an upper bound on speedup (for a given *f*), no matter how many processes are thrown at it
- Beyond Amdahl's law
  - Parallelization adds overhead (communication)
  - *f* could be a variable too
    - It may drop when problem size and N_{proc} increase
  - Parallel algorithm is different from the serial one

CENTER FOR COMPUTATION & TECHNOLOGY

Writing a parallel program step by step

- Step 1. Start from serial programs as a baseline
  - Something to check correctness and efficiency against
- Step 2. Analyze and profile the serial program
  - Identify the "hotspot"
  - Identify the parts that can be parallelized
- Step 3. Parallelize code incrementally
- Step 4. Check correctness of the parallel code
- Step 5. Iterate step 3 and 4

CENTER FOR COMPUTATION & TECHNOLOGY

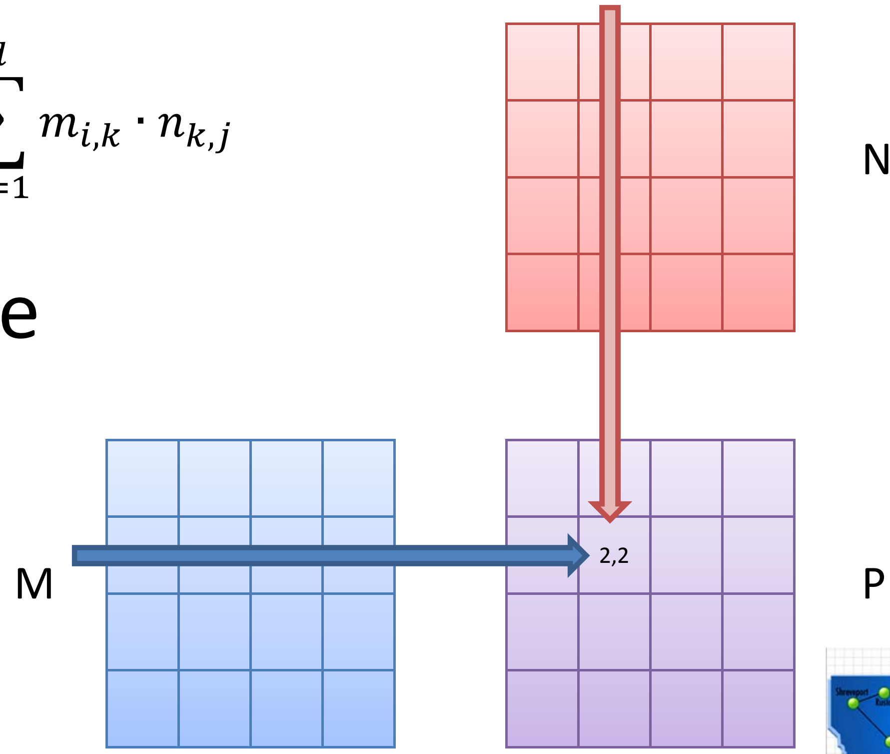
An REAL example of parallel computing

- Dense matrix multiplication $M_{md} x N_{dn} = P_{mn}$
- Formula

$$p_{i,j} = \sum_{k=1}^{d} m_{i,k} \cdot n_{k,j}$$

- For our 4x4 example

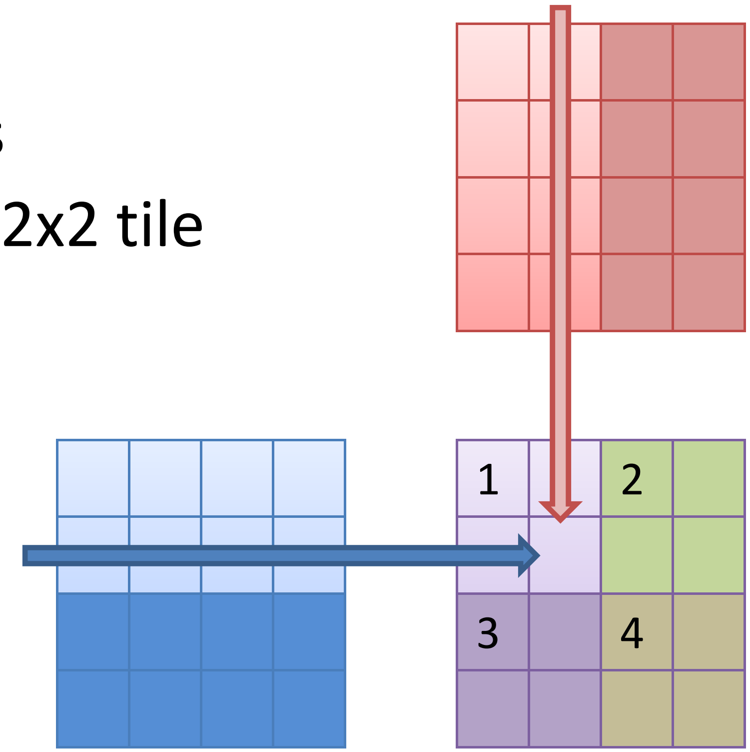
$$\begin{aligned} p_{2,2} = {} & m_{2,1} * n_{1,2} + \\ & m_{2,2} * n_{2,2} + \\ & m_{2,3} * n_{3,2} + \\ & m_{2,4} * n_{4,2} \end{aligned}$$



CENTER FOR COMPUTATION
& TECHNOLOGY

Parallelizing matrix multiplication

- Divide work among processors
- In our 4x4 example
  - Assuming 4 processors
  - Each responsible for a 2x2 tile (submatrix)
  - Can we do 4x1 or 1x4?



Pseudo code

Serial

```
for i = 1 to 4
  for j = 1 to 4
    for k = 1 to 4
      P(i,j) += M(I,k)*N(I,k);
```

Parallel

Each process figures out its own starting and ending indices;

for i = i_{start} to i_{end}

  for j = j_{start} to j_{end}

    for k = 1 to 4

      P(i,j) += M(I,k)*N(I,k);

CENTER FOR COMPUTATION & TECHNOLOGY

Outline

- Introduction
- Parallel programming models
- Parallel programming hurdles
- Heterogeneous computing

CENTER FOR COMPUTATION
& TECHNOLOGY

Single Program Multiple Data (SPMD)

- All program instances execute same program
- Data parallel - Each instance works on different part of the data
- The majority of parallel programs are of this type
- Can also have
  – SPSD: serial program
  – MPSD: rare
  – MPMD

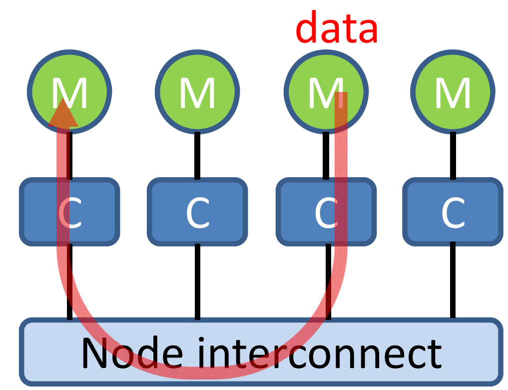
Memory system models

- Different ways of sharing data among processors
  - Distributed Memory
  - Shared Memory
  - Other memory models
    - Hybrid model
    - PGAS (Partitioned Global Address Space)

CENTER FOR COMPUTATION
& TECHNOLOGY

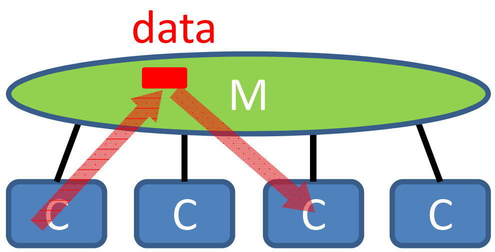
Distributed memory model

- Each process has its own address space
  - Data is local to each process
- Data sharing achieved via explicit message passing (through network)
- Example: MPI (Message Passing Interface)



Shared memory model

- All threads can access the global address space
- Data sharing achieved via writing to/reading from the same memory location
- Example: OpenMP



CENTER FOR COMPUTATION & TECHNOLOGY

Distributed vs. shared memory

Distributed

- Pro
  - Memory amount is scalable
  - Cheaper to build
- Con
  - Slow data sharing
    - Hard to balance the load
- Pro and con?
  - Explicit data transfer

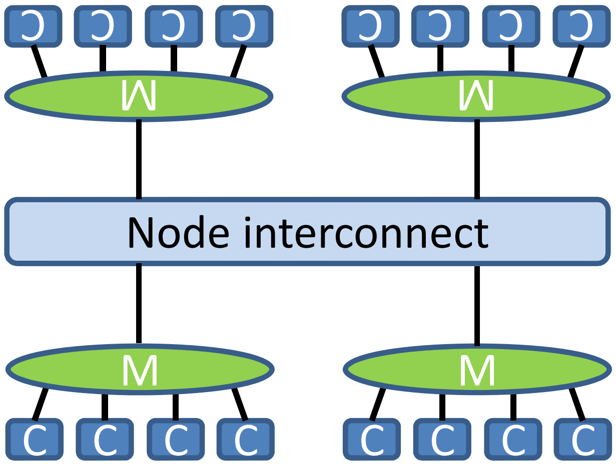
Shared

- Pro
  - Easy to use
  - Fast data sharing
- Con
  - Memory amount is not scalable
  - Expensive to build
- Pro and con?
  - Implicit data transfer

CENTER FOR COMPUTATION & TECHNOLOGY

Hybrid model

- Clusters of SMP (symmetric multi-processing) nodes dominate nowadays
- Hybrid model matches the physical structure of SMP clusters
  - OpenMP within nodes
  - MPI between nodes



LSU
CENTER FOR COMPUTATION & TECHNOLOGY

Potential benefits of hybrid model

- Message-passing within nodes (loopback) is eliminated
- Number of MPI processes is reduced, which means
  - Message size increases
  - Message number decreases
- Memory usage could be reduced
  - Eliminate replicated data
- Those are good, but in reality, (most) pure MPI programs run as fast (sometimes faster than) as hybrid ones…

Reasons why NOT using hybrid model

- Some (most?) MPI libraries already use internally different protocols
  - Shared memory data exchange within SMP nodes
  - Network communication between SMP nodes
- Overhead associated with thread management
  - Thread fork/join
  - Additional synchronization with hybrid programs

CENTER FOR COMPUTATION & TECHNOLOGY

Partitioned Global Address Space (PGAS)

- PGAS languages present programmers a global address space, regardless the type of the underlying system
  – Simulates hardware with software
  – Logically shared, physically distributed
- Examples
  – Unified Parallel C (UPC), CoArray Fortran (CAF), Fortress, Chapel, X10…
- Limitation
  – Lack of standard

CENTER FOR COMPUTATION
& TECHNOLOGY

Outline

- Introduction
- Parallel programming models
- Parallel programming hurdles
- Heterogeneous computing

CENTER FOR COMPUTATION
& TECHNOLOGY

Parallel Programming Hurdles

- Hidden serializations
- Overhead caused by parallelization
- Load balancing
- Synchronization issues

CENTER FOR COMPUTATION
& TECHNOLOGY

Hidden Serialization (1)

- Back to our box moving example
- What if there is a very long corridor that allows only one work to pass at a time between Location A and B?

Hidden Serialization (2)

- It is not the part in serial programs that is hard or impossible to parallelize
  - Intrinsic serialization (the *f* in Amdahl's law)
- Examples of hidden serialization:
  - System resources contention, e.g. I/O hotspot
  - Internal serialization, e.g. library functions that cannot be executed in parallel for correctness

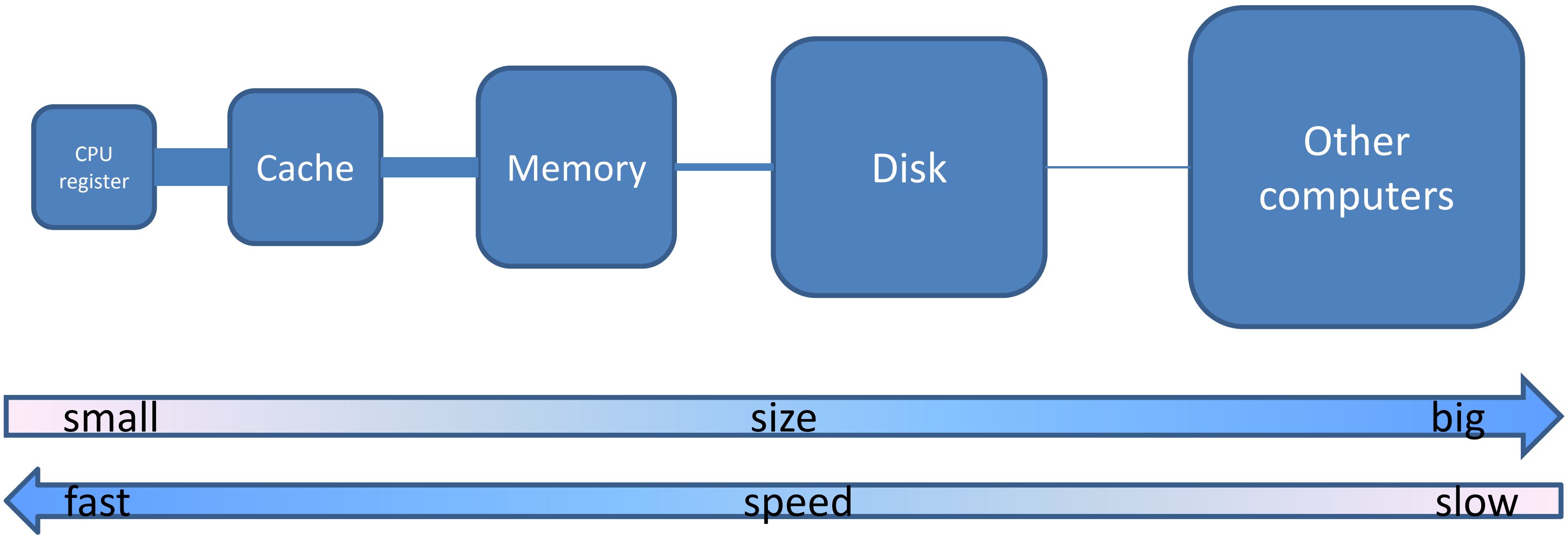
CENTER FOR COMPUTATION
& TECHNOLOGY

Communication overhead

- Sharing data across network is slow
  - Mainly a problem for distributed memory systems
- There are two parts of it
  - Latency: startup cost for each transfer
  - Bandwidth: extra cost for each byte
- Reduce communication overhead
  - Avoid unnecessary message passing
  - Reduce number of messages by combining them

CENTER FOR COMPUTATION & TECHNOLOGY

Memory Hierarchy

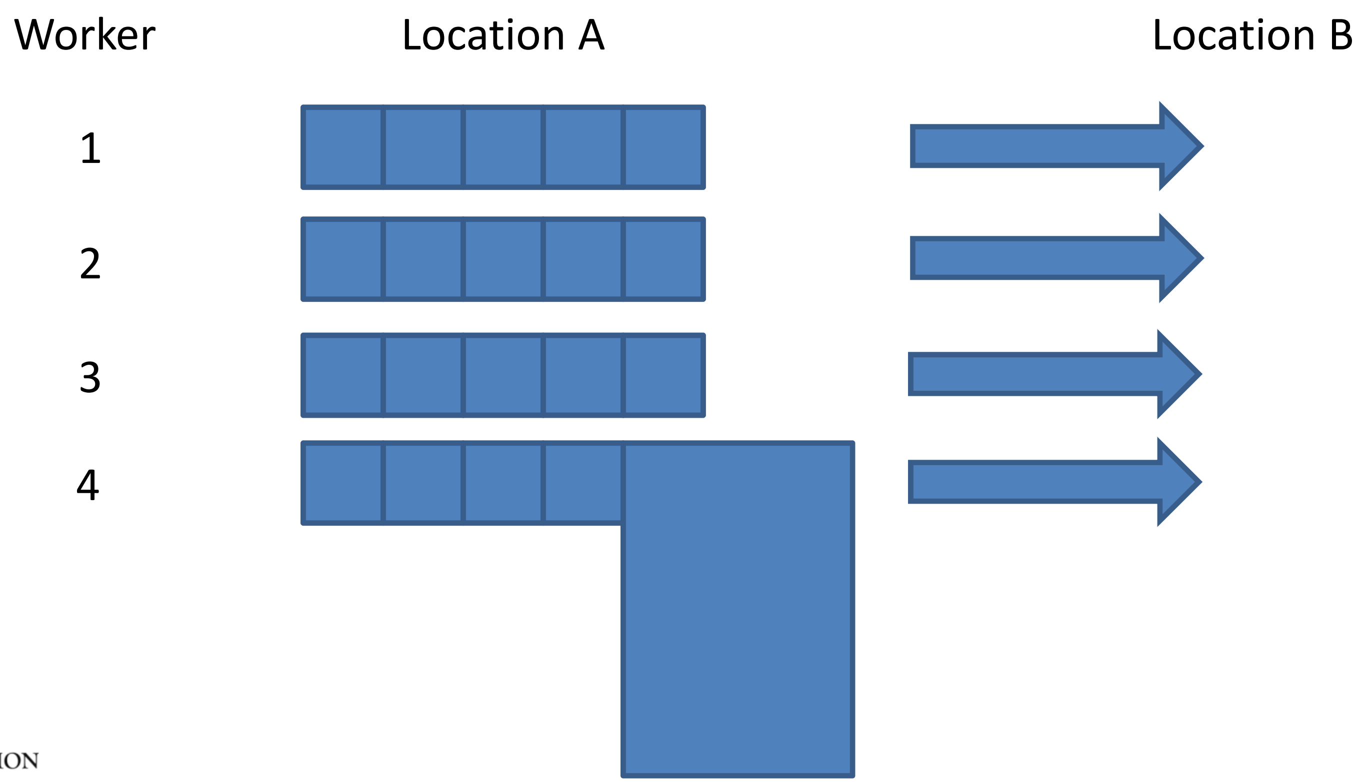


- Avoid unnecessary data transfer
- Load data in blocks (spatial locality)
- Reuse loaded data (temporal locality)
- All these apply to serial programs as well

CENTER FOR COMPUTATION & TECHNOLOGY

Load balancing (1)

- Back to our box moving example, again
- Anyone sees a problem?



Load balancing (2)

- Work load not evenly distributed
  - Some are working while others are idle
  - The slowest worker dominates in extreme cases
- Solutions
  - Explore various decomposition techniques
  - Dynamic load balancing
    - Hard for distributed memory
    - Adds overhead

CENTER FOR COMPUTATION & TECHNOLOGY

Synchronization issues - deadlock

Source: Stout & Jablonowski, Parallel computing 101, SC10

CENTER FOR COMPUTATION
& TECHNOLOGY

Deadlock

- Often caused by “blocking” communication operations
  - “Blocking” means “I will not proceed until the current operation is over”
- Solution
  - Use “non-blocking” operations
  - Caution: tradeoff between data safety and performance

CENTER FOR COMPUTATION
& TECHNOLOGY

Outline

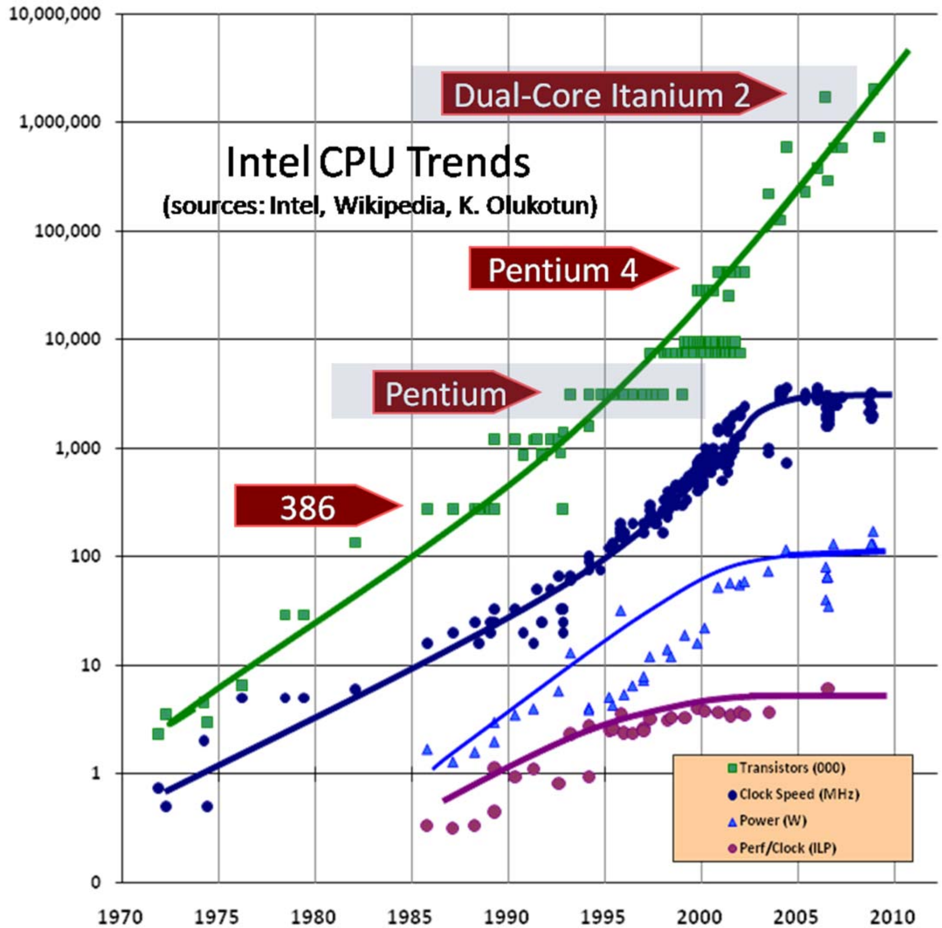
- Introduction
- Parallel programming models
- Parallel programming hurdles
- Heterogeneous computing

CENTER FOR COMPUTATION & TECHNOLOGY

Heterogeneous computing

- A heterogeneous system solves tasks using different types of processing units
  - CPUs
  - GPUs
  - DSPs
  - Co-processors
  - …
- As opposed to homogeneous systems, e.g. SMP nodes with CPUs only

The free (performance) lunch is over



Source: Herb Sutter, The Free Lunch is Over, http://www.gotw.ca/publications/concurrency-ddj.htm

CENTER FOR COMPUTATION & TECHNOLOGY

Power efficiency is the key

- We have been able to make computer run faster by adding more transistors
  - Moore's law
- Unfortunately, not any more
  - Power consumption/heat generation limits packing density
  - Power ~ speed^2
- Solution
  - Reduce each core's speed and use more cores – increased parallelism

CENTER FOR COMPUTATION & TECHNOLOGY

Graphic Processing Units (GPUs)

- Massively parallel many-core architecture
  - Thousands of cores capable of running millions of threads
  - Data parallelism
- GPUs are traditionally dedicated for graphic rendering, but become more versatile thanks to
  - Hardware: faster data transfer and more on-board memory
  - Software: libraries that provide more general purposed functions
- GPU vs CPU
  - GPUs are very effectively for certain type of tasks, but we still need the general purpose CPUs

CENTER FOR COMPUTATION
& TECHNOLOGY

GPUs and HPC

- Latest trend in HPC
  - SMP nodes with GPUs installed
  - 3 of the top 5 machines in the top 500 list are accelerated by GPUs
- Why people love them
  - Tremendous performance gain – single to double digit speedup compared to cpu-only versions
- Why people hate them (well, just a little bit)
  - Still (relatively) hard to program, even harder to optimize

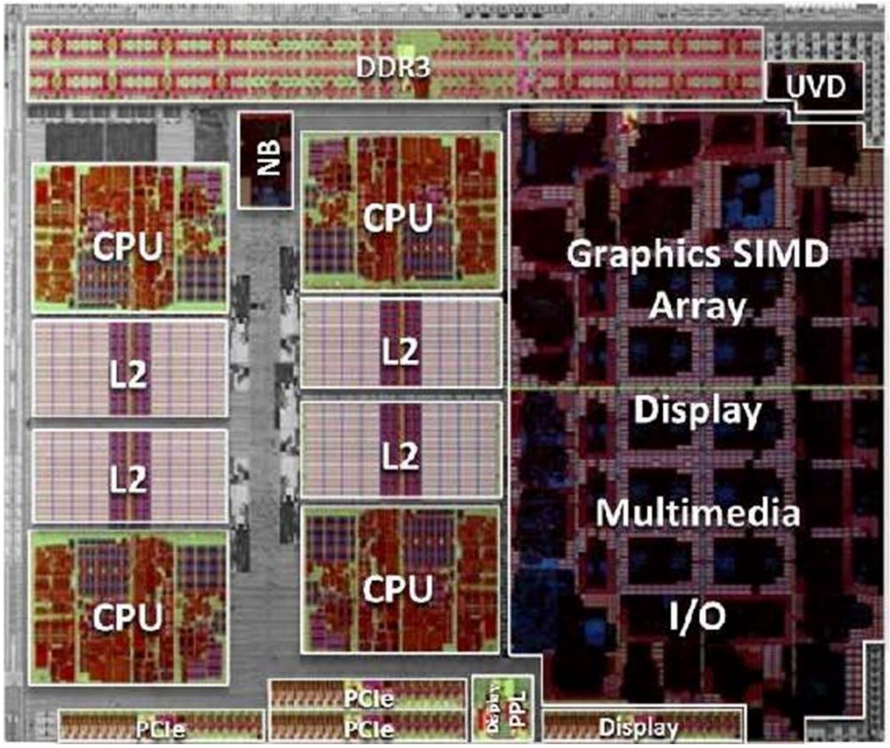
GPU programming strategies

- GPUs need to copy data from main memory to its on-board memory and copy them back
  - Data transfer over PCIe is the bottleneck, so one needs to
    - Avoid data transfer and reuse data
    - Overlap data transfer and computation
- Massively parallel, so it is a crime to do anything anti-parallel
  - Need to launch enough threads in parallel to keep the device busy
  - Threads need to access contiguous data
  - Thread divergence needs to be eliminated

CENTER FOR COMPUTATION & TECHNOLOGY

Fused processing unit

- CPU and GPU cores on the same die
- GPU cores can access main memory
  - Hence no PCIe bottleneck
- Much less GPU cores than a discrete graphic card can carry
  - Less processing power



AMD "Llano" Accelerated Processing Unit (APU)

CENTER FOR COMPUTATION & TECHNOLOGY