

Lecture #42 – Parallel Computing

2005-05-09



Andy Carle
cs61c @inst

The California legislature is currently working on a bill to ban “remote hunting via the internet” after the incorporation of a Texas company specializing in a unique combination of robotics, web cameras, and weapons. Years of Counter Strike practice and I can’t even get a meal out of it...



Scientific Computing

- **Traditional Science**
 - 1) Produce theories and designs on “paper”
 - 2) Perform experiments or build systems
 - Has become difficult, expensive, slow, and dangerous for fields on the leading edge
- **Computational Science**
 - Use ultra-high performance computers to simulate the system we’re interested in
- **Acknowledgement**
 - Many of the concepts and some of the content of this lecture were drawn from Prof. Jim Demmel’s CS 267 lecture slides which can be found at http://www.cs.berkeley.edu/~demmel/cs267_Spr05/



Example Applications

- **Science**
 - Global climate modeling
 - Biology: genomics; protein folding; drug design
 - Astrophysical modeling
 - Computational Chemistry
 - Computational Material Sciences and Nanosciences
- **Engineering**
 - Semiconductor design
 - Earthquake and structural modeling
 - Computation fluid dynamics (airplane design)
 - Combustion (engine design)
 - Crash simulation
- **Business**
 - Financial and economic modeling
 - Transaction processing, web services and search engines
- **Defense**
 - Nuclear weapons – test by simulations
 - Cryptography



Performance Requirements

- **Terminology**
 - Flop – Floating point operation
 - Flops/second – standard metric for expressing the computing power of a system
- **Global Climate Modeling**
 - Divide the world into a grid (e.g. 10 km spacing)
 - Solve fluid dynamics equations to determine what the air has done at that point every minute
 - Requires about 100 Flops per grid point per minute
 - This is an extremely simplified view of how the atmosphere works, to be maximally effective you need to simulate many additional systems on a much finer grid



Performance Requirements (2)

- **Computational Requirements**
 - To keep up with real time (i.e. simulate one minute per wall clock minute): 8 Gflops/sec
 - Weather Prediction (7 days in 24 hours): 56 Gflops/sec
 - Climate Prediction (50 years in 30 days): 4.8 Tflops/sec
 - Climate Prediction Experimentation (50 years in 12 hours): 288 Tflops/sec
- **Perspective**
 - Pentium 4 1.4GHz, 1GB RAM, 4x100MHz FSB
 - ~320 Mflops/sec, effective
 - Climate Prediction would take ~1233 years



What Can We Do?

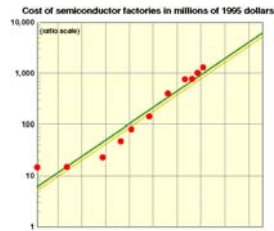
- **Wait**
 - Moore’s law tells us things are getting better; why not stall for the moment?
- **Parallel Computing!**



Prohibitive Costs

Rock's Law

- The cost of building a semiconductor chip fabrication plant that is capable of producing chips in line with Moore's law doubles every four years



CS61C L42 Parallel Computing (7)

Carle, Spring 2005 © UCB

How fast can a serial computer be?

- Consider a 1 Tflop/sec sequential machine:
 - Data must travel some distance, r , to get from memory to CPU
 - To get 1 data element per cycle, this means 10^{12} times per second at the speed of light, $c = 3 \times 10^8$ m/s. Thus $r < c/10^{12} = 0.3$ mm
 - So all of the data we want to process must be stored within 0.3 mm of the CPU
- Now put 1 Tbyte of storage in a 0.3 mm x 0.3 mm area:
 - Each word occupies about 3 square Angstroms, the size of a very small atom
 - Maybe someday, but it most certainly isn't going to involve transistors as we know them



CS61C L42 Parallel Computing (8)

Carle, Spring 2005 © UCB

What is Parallel Computing?

- Dividing a task among multiple processors to arrive at a unified (meaningful) solution
 - For today, we will focus on systems with many processors executing identical code
- How is this different from Multiprogramming (which we've touched on some in this course)?
- How is this different from Distributed Computing?



CS61C L42 Parallel Computing (9)

Carle, Spring 2005 © UCB

Recent History

- Parallel Computing as a field exploded in popularity in the mid-1990s
- This resulted in an "arms race" between universities, research labs, and governments to have the fastest supercomputer in the world



CS61C L42 Parallel Computing (10)

Carle, Spring 2005 © UCB

Current Champions



BlueGene/L – IBM/DOE
Rochester, United States
32768 Processors, 70.72 Tflops/sec
0.7 GHz PowerPC 440



Columbia – NASA/Ames
Mountain View, United States
10160 Processors, 51.87 Tflops/sec
1.5 GHz SGI Altix



Earth Simulator – Earth Simulator Ctr.
Yokohama, Japan
5120 Processors, 35.86 Tflops/sec
SX6 Vector



Data Source: top500.org
CS61C L42 Parallel Computing (11)

Carle, Spring 2005 © UCB

Parallel Programming

- Processes and Synchronization
- Processor Layout
- Other Challenges
 - Locality
 - Finding parallelism
 - Parallel Overhead
 - Load Balance



CS61C L42 Parallel Computing (13)

Carle, Spring 2005 © UCB

Processes

◦ We need a mechanism to intelligently split the execution of a program

◦ Fork:

```
int main(...){
    int pid = fork();
    if (pid == 0) printf("I am the child.");
    if (pid != 0) printf("I am the parent.");
    return 0;
}
```

◦ What will this print?



Processes (2)

◦ We don't know! Two potential orderings:

- I am the child. I am the parent.
- I am the parent. I am the child.
- This situation is a simple race condition. This type of problem can get far more complicated...

◦ Modern parallel compilers and runtime environments hide the details of actually calling `fork()` and moving the processes to individual processors, but the complexity of synchronization remains



Synchronization

◦ How do processors communicate with each other?

◦ How do processors know when to communicate with each other?

◦ How do processors know which other processor has the information they need?

◦ When you are done computing, which processor, or processors, have the answer?



Synchronization (2)

◦ Some of the logistical complexity of these operations is reduced by standard communication frameworks

- Message Passing Interface (MPI)

◦ Sorting out the issue of who holds what data can be made easier with the use of explicitly parallel languages

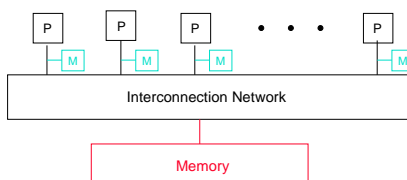
- Unified Parallel C (UPC)
- Titanium (Parallel Java Variant)

◦ Even with these tools, much of the skill and challenge of parallel programming is in resolving these problems



Processor Layout

Generalized View

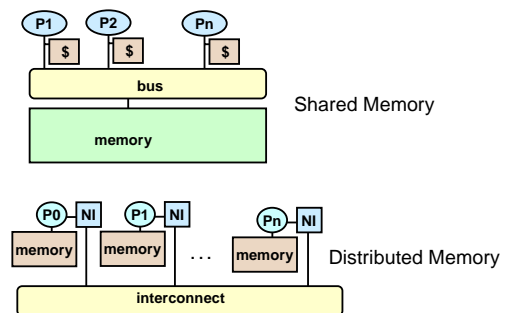


M = Memory local to one processor

Memory = Memory local to all other processors



Processor Layout (2)



Processor Layout (3)

- Clusters of SMPs
 - n of the N total processors share one memory
 - Simple shared memory communication within one cluster of n processors
 - Explicit network type calls to communicate from one group of n to another
- Understanding the processor layout that your application will be running on is crucial!

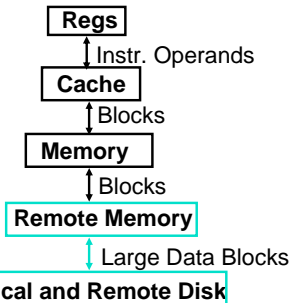


CS61C L42 Parallel Computing (20)

Carle, Spring 2005 © UCB

Parallel Locality

- We now have to expand our view of the memory hierarchy to include remote machines
- Remote memory behaves like a very fast network
 - Bandwidth vs. Latency becomes important



CS61C L42 Parallel Computing (21)

Carle, Spring 2005 © UCB

Amdahl's Law

- Applications can almost never be completely parallelized
 - Let s be the fraction of work done sequentially, so (1-s) is fraction parallelizable, and P = number of processors
- $$\text{Speedup}(P) = \text{Time}(1)/\text{Time}(P)$$
- $$\leq 1/(s + (1-s)/P)$$
- $$\leq 1/s$$
- Even if the parallel portion of your application speeds up perfectly, your performance may be limited by the sequential portion



CS61C L42 Parallel Computing (22)

Carle, Spring 2005 © UCB

Parallel Overhead

- Given enough parallel work, these are the biggest barriers to getting desired speedup
- Parallelism overheads include:
 - cost of starting a thread or process
 - cost of communicating shared data
 - cost of synchronizing
 - extra (redundant) computation
- Each of these can be in the range of milliseconds (many millions of flops) on some systems
- Tradeoff: Algorithm needs sufficiently large units of work to run fast in parallel (i.e. large granularity), but not so large that there is not enough parallel work



CS61C L42 Parallel Computing (23)

Carle, Spring 2005 © UCB

Load Balance

- Load imbalance is the time that some processors in the system are idle due to
 - insufficient parallelism (during that phase)
 - unequal size tasks
- Examples of the latter
 - adapting to "interesting parts of a domain"
 - tree-structured computations
 - fundamentally unstructured problems
- Algorithms need to carefully balance load



CS61C L42 Parallel Computing (24)

Carle, Spring 2005 © UCB

Summary

- Parallel Computing is a multi-billion dollar industry driven by interesting and useful scientific computing applications
- It is extremely unlikely that sequential computing will ever again catch up with the processing power of parallel systems
- Programming parallel systems can be extremely challenging, but is built upon many of the concepts you've learned this semester in 61c



CS61C L42 Parallel Computing (25)

Carle, Spring 2005 © UCB