



CS 575 Parallel Processing

Lecture one: Introduction

Wim Bohm

Colorado State University



Course Topics

- Introduction, Background
 - Orders of magnitude, Recurrences
- Parallel Models, communication
- Performance, Speedup, Efficiency
- Parallel Algorithms
 - Dense, sparse matrix algorithms
 - Sorting
 - Graph Algorithms
 - N body problem
 - Searching (Branch and bound, Backtrack, Dynamic Programming)
 - Fast Fourier Transform



Course Organization

- 2008 course **reorganization**
 - Unite 575, 575dl, GS511
 - Modernize: less models, more algorithms, apps
- Check the web page regularly
 - www.cs.colostate.edu/~cs575dl
- The distance learning course
 - uses RamCT, has TA: Daegon Kim
- Tests
 - Quizzes, One Midterm, One Final
- Project
 - Google/IBM data center



Intro: Parallel Computing

- Parallel Computing is cost effective
 - Off the shelf, commodity processors are very fast
 - Memory is very cheap
 - Building a processor that is a small factor faster costs an order of magnitude more
 - Clusters: NoW is the time!
 - Cheapest way to get more performance: multiprocessor
 - NoW: Networks of workstations
 - Workstation can be an SMP
 - SMP: Symmetric Multi Processor
 - Shared memory, Bus or Crossbar (eg. Cray)



Wile E. Coyote's Parallel Computer

- Get a lot of the fastest processors
- Get a lot of memory per processor
- Get the fastest network
- Hook it all together

- And then what ???



Now you gotta program it!

Parallel programming introduces:

- Task partitioning, task scheduling
- Data partitioning, distribution
- Synchronization
- Load balancing
- Latency issues
 - hiding
 - tolerance



Problem with Wile E. Coyote Architecture

Von Neumann Machines not built for //ism

- To get high speed, processors have **lots of state**
 - Cache, stack, global memory
- To tolerate latency, we need **fast context switch. WHY?**
- No free lunch: **can't have both**
 - Certainly not if the processor was not designed for both
- Memory wall: memory gets slower and slower
 - in terms of number of cycles it takes to access
- Memory hierarchy gets more and more complex
- Memory accesses block
 - No split phase memory access
- **BUT** that's still what we use, **WHY?**



Sequential vs Parallel Algorithms

- Efficient Parallel Algorithms
 - Use efficient sequential algorithms
 - Maximize parallelism
 - Minimize overhead
 - synchronization, remote accesses
 - Efficiency is Architecture Dependent
- Efficient Sequential Algorithms
 - Minimize time, space
 - Efficiency is portable
 - Efficient program on Pentium \sim Efficient program on Opteron



Speedup

- Ideal: n processors \rightarrow n fold speed up
 - Ideal not always possible. **WHY?**
 - Tasks are data dependent
 - Not all processors are always busy
 - Remote data needs communication
 - Memory wall PLUS Communication wall
- Linear speedup: α n speedup ($\alpha \leq 1$)
- Super linear speedup: $\alpha > 1$
 - Nonsense! Because we can execute the faster parallel program sequentially
 - No nonsense!! Because parallel computers do not just have more processors, they have more caches



Parallel Programming Paradigms

- Implicit parallel programming: Super Compilers
 - Compiler extracts parallelism from **sequential code**
 - Distributes data, creates and schedules tasks
 - Complication: **side effects:**
 - the sequential order of reads and writes to a memory location determines the program outcome
 - a parallelizing compiler must obey the sequential order of side effecting statements and still create //ism
 - pointers, aliases, indirect array reference make analyzing which statements access which locations hard or impossible
 - **35** years of compiler research has not brought much result.



Paradigms cont'

- Implicit parallel programming cont'
 - Simple, clean case: Functional Programming (FP)
 - Functions: **no side effects**, order of execution less constrained
 $F (P(x,y), Q(y,z))$ P and Q can be executed in parallel
 - Simple **single assignment** memory model: no pointers, no write after read or write after write hazards (dataflow semantics)
 - FP was long doomed too high level too inefficient, because the simple memory model causes lots of copies
 - FP is coming back: **MapReduce** approach in data centers (Google) is a data parallel functional paradigm



Explicit parallel programming

- Explicit parallel programming
 - Multithreading: OpenMP, Pthreads, Solaris threads
 - Message Passing: MPI
 - Data parallel programming (Niche work, but important)
- Explicit Parallelism complicates programming
 - creation, allocation, scheduling of processes
 - data partitioning
 - Synchronization (semaphores, locks, messages)



Example 1: Weather Prediction

- Area, segments
 - $3000*3000*11$ cubic miles
 - $.1*.1*.1$ cubic mile: $\sim 10^{11}$ segments
- Two day prediction
 - half hour time steps: ~ 100 time steps
- Computation per segment
 - Temp, Pressure, Humidity, Wind speed, Wind direction for each time step in each segment
 - Assume ~ 100 FLOPs per time step per segment



Performance: Weather Prediction

- Computational requirement: 10^{15} FLOPs
assume one FLOP per clock cycle
- Fast serial computer: 4 GHz
- Total serial time: 25×10^4 sec \sim 70 hours
- Not too good for 48 hour weather prediction



Parallel Weather Prediction

- 1 K workstations, grid connected
 - 10^8 segment computations per processor
 - 10^8 instructions per second
 - 100 instructions per segment computation
 - 100 time steps: 10^4 seconds = **~ 3 hours**
 - Much more acceptable
 - Assumption: **Communication not a problem here**
Why is this assumption reasonable?
- More workstations:
 - finer grid, better accuracy



Example 2: N body problem

- Astronomy: bodies in space
 - Attract each other: Gravitational force
Newtons law
 - $O(n*n)$ calculations per "snapshot"
 - Galaxy: $\sim 10^{11}$ bodies $\rightarrow \sim 10^{22}$ calculations
 - Calculation 1 micro sec
 - Snapshot: 10^{16} secs = $\sim 10^{11}$ days = $\sim 3*10^8$ years
 - Is parallelism going to help us? NO
 - What **does** help? Better algorithm: Barnes Hut
 - Divides the space in "quad tree"
 - Treats "far' away quads as one body



Other Challenging Applications

- Satellite data acquisition: billions of bits / sec
- Satellite data processing
 - Pollution levels, Remote sensing of materials
 - Image recognition
- Discrete optimization problems
 - Planning, Scheduling, VLSI design
- Material, Weapons modeling
- Airplane/Satellite/Vehicle design
- Internet (Google search)



Application Specific Architectures

- Mapping an algorithm directly onto hardware
 - Embedded systems: CS460, High Performance: CS560
- ASICs: Application Specific Integrated Circuits
- Levels of 'specificity'
 - Full custom ASICs
 - Standard cell ASICs
 - Field programmable gate arrays
- Computational models
 - Dataflow graphs
 - Systolic arrays
- Promising orders of magnitude better performance, lower power



ASICs cont'

- How much faster than General purpose?
 - Example: 1D 1024 FFT
 - General purpose machine (G4): 25 micro secs
 - ASIC device (MIT Lincoln Labs): 32 nano secs
 - ASIC device uses 20 milliwatts (100 * less power)
- Future designs:
 - 2 tera ops in small (< cubic ft) device
 - Target applications
 - FFT
 - Finite Impulse Response (FIR) Filters
 - Matrix multiply
 - QR decomposition



Background

- If you do not have necessary background in analysis of algorithms
 - See the book
 - Introduction to Algorithms by
 - Cormen, Leiserson, Rivest and Stein
 - Or go online
 - Topics to study
 - Introduction
 - Growth of functions
 - Summations
 - Recurrences



Recurrences

■ Fibonacci

- rabbit pairs become fertile after 1 month, and produce 1 rabbit pair every month
- month 0: 1 pair, month 1: 1 pair, month 2: 1+1 pairs, month 3: 2+1 pairs
(these are very happy rabbits, why? :)
- month n: old pairs + new born pairs
 - $F_n = F_{n-1} + F_{n-2}$, $F_0 = F_1 = 1$
 - 1,1,2,3,5,8,13,21,34 ... exponential growth, why?



Linear homogeneous recurrences

$$A_n = C_1 A_{n-1} + C_2 A_{n-2} + \dots + C_R A_{n-R}$$

- A general solution involves a linear combination of individual solutions $A_n = \alpha^n$
- Given r initial values this general solution can be made specific
- Back to bugs bunny: $F_n = F_{n-1} + F_{n-2}$, $F_0 = F_1 = 1$

$$F_k = \alpha^k$$

$$\alpha^n = \alpha^{n-1} + \alpha^{n-2} \rightarrow \alpha^2 = \alpha + 1 \quad \text{the characteristic equation}$$

$$\alpha_{1,2} = \frac{1}{2} \pm \frac{1}{2} \sqrt{5}$$



Bugs bunny cont'

- **General solution:**

$$F_n = k_1\left(\frac{1}{2} + \frac{1}{2} \sqrt{5}\right)^n + k_2\left(\frac{1}{2} - \frac{1}{2} \sqrt{5}\right)^n$$

- Initial values $F_0 = F_1 = 1$ determine **specific solution:**

$$F_n = \frac{1}{\sqrt{5}} \left(\frac{1}{2} + \frac{1}{2} \sqrt{5}\right)^{n+1} - \frac{1}{\sqrt{5}} \left(\frac{1}{2} - \frac{1}{2} \sqrt{5}\right)^{n+1}$$

- Let us do some more e.g-s
- Read up on linear **inhomogeneous** recurrences:

$$A_n = C_1 A_{n-1} + C_2 A_{n-2} + \dots + C_R A_{n-R} + f(n)$$



Inhomogeneous example

$$A_n = C A_{n-1} + f(n)$$

- Find solution for homogeneous part: $= kC^n$
- Suppose P_n is a particular solution:

$$P_n = C P_{n-1} + f(n)$$

then

$$A_n = kC^n + P_n \text{ is the general solution}$$

General solution = general homogeneous + particular

Check this by substitution

Some particular solutions for

$$A_n = C A_{n-1} + f(n)$$

F(n)	P _n
a (constant)	b (constant)
a n	b n + c
a n ²	b n ² + c n + d
a ⁿ	b a ⁿ

linear → linear, quadratic → quadratic, etc.

Let's try it on $A_n = 2 A_{n-1} + 1, A_1 = 1$



Orders of Magnitude

- O, Ω, Θ

- $f(x) = O(g(x))$ iff $\exists c, n_0 : f(x) < c.g(x) \quad \forall n > n_0$
 - used for upper bound of algorithm complexity
- $f(x) = \Omega(g(x))$ iff $\exists c, n_0 : f(x) > c.g(x) \quad \forall n > n_0$
 - used for lower bound of problem complexity
- $f(x) = \Theta(g(x))$ iff $f(x)=O(g(x))$ and $f(x)=\Omega(g(x))$
 - “Tight” bound

Divide and conquer recurrence

$$A_n = C A_{n/d} + f(n)$$

Cormen, Leiserson et.al. master method is complex. An easier version comes from Rosen (1st yr discrete math book): $A_n = C A_{n/d} + kn^p$

- $A_n = O(n^p)$ if $C < d^p$ eg. $A_n = 3 A_{n/2} + n^2$
- $A_n = O(n^p \log(n))$ if $C = d^p$ eg. $A_n = 2 A_{n/2} + n$
- $A_n = O(n^{\log_d C})$ if $C > d^p$ eg. $A_n = 3 A_{n/2} + k$

Discuss **binary search** and **merge sort**



Background: Closed problems

- Closed problem P:
 - \exists algorithm X with $O(X) = \Omega(P)$
eg. Sort has lower and upperbound of $n \log(n)$
- Problem P has algorithmic gap:
 - P is not closed, eg., all NP Complete problems (problems with polynomial lower bound but exponential upper bound, such as TSP)



Concluding Recurrence Relations

- <http://www.math.uvic.ca/faculty/gmacgill/guide/solvingrec.pdf>
- Rosen (discrete math book)
- Cormen Leiserson, et.al.

- Algorithmic complexity often described using recurrence relations: $f(n) = R(f(1) .. f(n-1))$
- Three important types of recurrence relations
 - Linear homogeneous
 - Simple linear inhomogeneous
 - Divide and Conquer



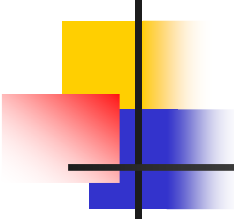
Linear, Homogeneous Rec. Relations

- $a_n = c_1 \cdot a_{n-1} + c_2 \cdot a_{n-2} + \dots + c_r \cdot a_{n-r}$
 $a_0 = i_0, \dots, a_{r-1} = i_{r-1}$: initial values
- Individual solutions of form α^n
 - give rise to characteristic equation
 - solve to obtain “roots”
- General solution
 - linear combination of roots
- Specific solution
 - find coefficients of general solution using initial values



Linear Inhomogeneous

- $a_n = c_1 \cdot a_{n-1} + f(n)$
 - Special case: $c_1 = 1$
 - $a_n = a_0 + f(1) + \dots + f(n)$
 - General case:
 - Solve homogeneous part: $h_n = A \cdot c_1^n$
 - if α_n^* is a particular solution then $a_n = A \cdot c_1^n + \alpha_n^*$ is the general solution
 - Particular solution "mimics" $f(n)$
 $f(n)$: $c, d \cdot n, d \cdot n^2, d^n$
 α_n^* : $b, a \cdot n + b, a \cdot n^2 + b \cdot n + c, b \cdot d^n$ or $b \cdot n \cdot d^n$
($d \neq c_1$) or ($d = c_1$)



Div & Co: $A_n = C A_{n/d} + kn^p$

C	P	A_n
1	0	$O(\log n)$
$c < d$	1	$O(n)$
$c = d$	0	$O(n)$
$c = d$	1	$O(n \log n)$
$c > d$	$p > \log_d c$	$O(n^p)$
$c > d$	$p < \log_d c$	$O(n^{\log_d c})$