



Parallel Algorithms

Benny Wong

Outline

- Overview
- Parallel Models of Computation
- Algorithmic Techniques
- Designing Parallel Algorithms
- Coding in Parallel
- Parallel Coding Techniques
- Current Work
- Future Work

Overview

- Multiple operations performed simultaneously
- Major developments:
 - Models of computation
 - Algorithmic techniques

Parallel Models of Computation

- Developing a standard model is difficult as parallel computers vary in their organizations
- Work efficiency
 - Efficiency (total operations) more important than speed
 - Sequential machines, $W = t$
 - Parallel machines, $W = P \cdot t$
 - E.g. sort algorithm that takes $O(n^{1/2} \log n)$ with $n^{1/2}$ processors is better than an algorithm that takes $O(\log n)$ with n^2 processors

Parallel Models of Computation (continued)

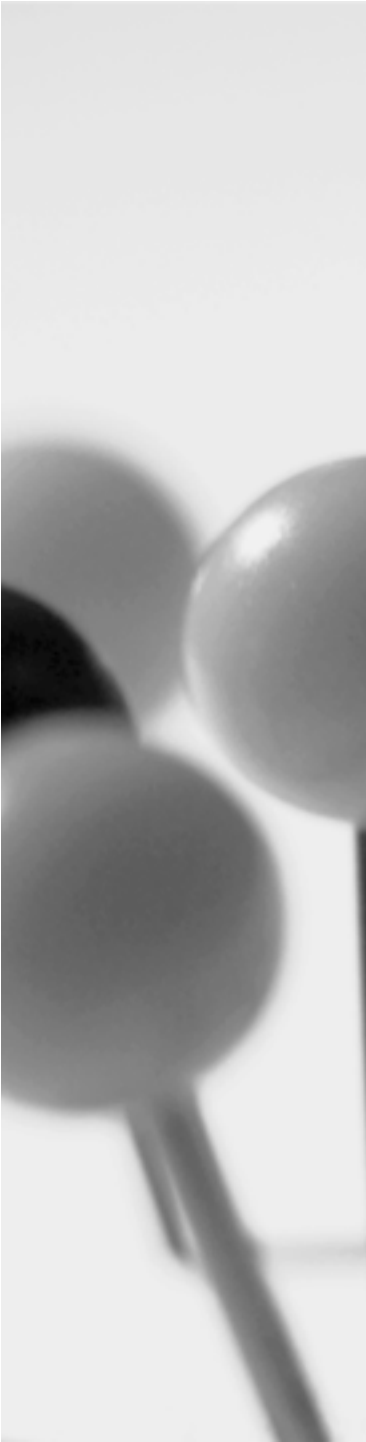
- Emulation
 - Parallel random-access machine (PRAM) model is often criticized
 - Importance lies in the fact that these models can be mapped to algorithms that are efficient on realistic machines
 - E.g. P/L processor machine with L being latency is as efficient as the latency is hidden and using factor of L fewer processors
- Modeling Communication
 - It is important to model the communication bandwidth for best performance
 - Difficult to hide insufficient bandwidth than large latency
 - Synchronization and memory access modeled as well
 - Shared memory or message passing processors

Algorithmic Techniques

- Divide and Conquer
 - Solving divided sub-problems in parallel
 - Using recursion
 - E.g. merge-sort
- Data Partitioning
 - Master and slave algorithm
 - Master splits the list of numbers
 - Slave gets allocated the subset of numbers

Algorithmic Techniques (continued)

- Randomization
 - Local decisions adds up to good global decisions
 - E.g. sorting collection of integer keys
- Parallel Pointer Manipulations
 - Pointer jumping, Euler-tour, ear decomposition, and graph contraction techniques
 - E.g. node labeling of a tree

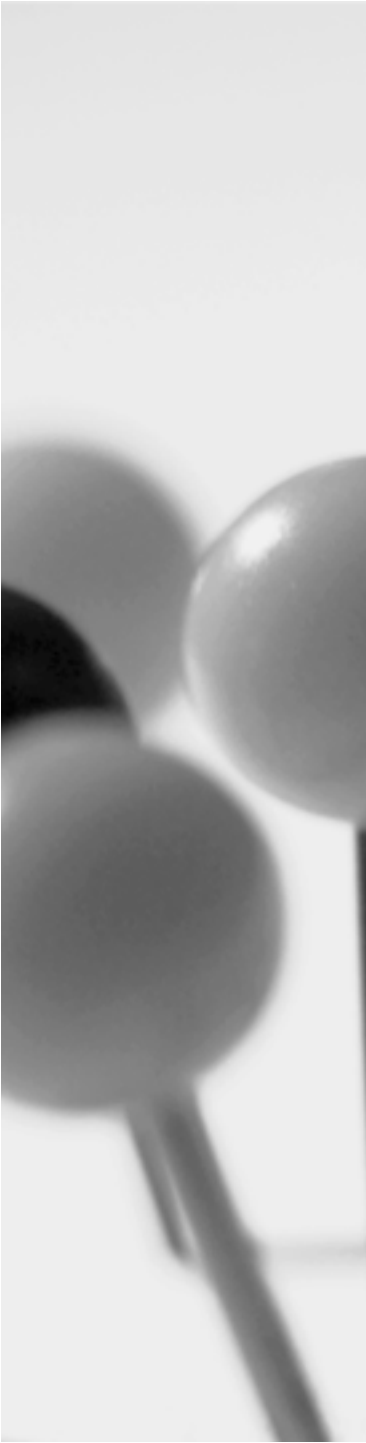


Algorithmic Techniques (continued)

- Hashing for balancing load and mapping addresses
 - Hash maps each job to one random bucket so no bucket gets many jobs
- Iterative techniques for solving linear systems
 - Separating independent sub-problems with one-way dissection and matrix modifications
- Finding graph separators to reduce communication
 - Finding common sub-problems

Designing Parallel Algorithms

- Design process can be viewed in four stages
 - Partitioning
 - E.g. grading n answer-scripts with m questions
 - Use n markers (Domain decomposition)
 - Use m markers (Functional decomposition)
 - Communication Analysis
 - If average needs to be computed given different physical locations
 - Proceed independently and communicate in the end
 - If time to communicate much greater than grading, reduce number of marker and increase work for each



Designing Parallel Algorithms (continued)

- Design process can be viewed in four stages
 - Granularity Control
 - Combining small tasks to produce larger more efficient ones
 - E.g. k markers could mark n/k scripts each
 - Mapping
 - Specifies where each task is to be executed
 - E.g. schedule jobs to maximize processor utility, not in this example as all jobs are uniform

Coding in Parallel

- Spot the possible errors!

- Original program

```
real a[m], b[m]
```

```
do i = 1, n
```

```
    a[i] = b[k] + a[i] + 10000.0
```

```
end do
```

- Transformed program

```
real a[m], b[m]
```

```
c = b[k] + 10000.0
```

```
do i = n, 1, -1
```

```
    a[i] = a[i] + c
```

```
end do
```

Coding in Parallel (continued)

- Possible errors
 - Overflow
 - If $b[k]$ is very large and $a[1]$ is negative
 - Floating number approximations
 - Order of addition has changed
 - Memory fault
 - If $k > m$ but $n < 1$, the reference is illegal
 - Aliasing
 - a and b might be aliased
 - If $k = n$, $b[k]$ will be changed when $i = 1$ because $b[n]$ is aliased with $a[1]$

Parallel Coding Techniques

- Data Layout

Original program:

```
do all j = 1, n
  do all i = 1, n
    a[i,j] = a[i,j]+c
  end do all
end do all
```

Transformed program:

```
call FORK(Pnum)
do j = (n/Pnum)*Pid+1, min((n/Pnum)*(Pid+1),n)
  do i = 1, n
    a[i,j] = a[i,j]+c
  end do
end do
call JOIN()
```

Parallel Coding Techniques (continued)

- Computation Partitioning

Original program:

```
do i = 1, n
    a[i] = a[i] + c
    b[i] = b[i] + c
end do
```

Transformed program:

```
LB = (n/Pnum)*Pid + 1;
UB = (n/Pnum)*(Pid + 1);
do i = LB, UB
    a[i] = a[i] + c
    b[i] = b[i] + c
end do
```



Parallel Coding Techniques (continued)

- Communication Optimization
 - Message coalescing
 - Message aggregation
 - Message pipelining
- Exposing Coarse-Grained Parallelism to avoid excessive communication
 - Procedure call parallelization
 - Split
 - Graph partitioning

Current Work

- Theoretical work will be complemented by extensive experimentation
- Data-parallel programming
 - Operation performed in parallel across a set of data, e.g. HPF
- Control-parallel programming
 - Multiple independent processes executed simultaneously, e.g. MPI, PVM, etc.



Future Directions

- Computational biology, astronomy, seismology, fluid dynamics, etc.
- Multiple processors began to appear on the commodity personal computers

A black and white photograph of several lollipops on sticks, arranged in a row. The lollipops are out of focus, creating a soft, bokeh effect. The background is a dark, solid grey color. The word "Questions?" is written in a large, white, sans-serif font across the center of the image.

Questions?



Thank You