Survey of Parallel Computation

Larry Snyder
University of Washington, Seattle

Course Logistics

- Teaching Assistant: Carlson
- No Textbook
- Take lecture notes -- the slides will be online sometime after lecture
- Occasional homework problems, including programming in ZPL
- Final Exam

Please ask questions when they arise

Topic Overview

Goal: To give a good idea of parallelism

- Concepts -- looking at problems with "parallel eyes"
- Algorithms -- different resources to work with
- Languages -- reduce control flow; increase independence
- Hardware -- the challenge is communication, not instruction execution
- Programming -- describe the computation without saying it sequentially

Why study parallelism?

- For most user applications sequential computers are fast enough
- For sophisticated scientific modeling, engineering design, drug design, data mining, etc. sequential computers are too slow
- Except for fundamental breakthroughs in algorithms, parallelism is the only known method for greater speed beyond baseline sequential computation

Parallelism is interesting, too!

The Worlds Fastest Computers

- Check the site

  www.netlib.org/benchmark/top500.html
- Question: what computation is used to measure the world's fastest computers?

Three levels of performance …

- Manufacturers claimed performance
- Benchmark measured performance
- Observed performance on interesting programs

Types of Parallelism

Parallelism is a standard technique in computer science … think of examples

- Pipelining In Processor Design
  - The next instruction is fetched before the current instruction is completed
  - In modern processors there can be 7-10 pipeline stages or more, and usually many instructions are being executed simultaneously

Pipelining is a powerful form of parallelism

Types of Parallelism (continued)

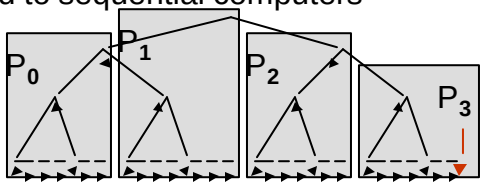
- Overlapping computation and communication
  - Web browsers construct the page and display it in incomplete form while the images are being downloaded
  - Operating systems switch tasks while pages are being fetched in response to a page fault

Overlapping comp/comm is a powerful technique

Types of Parallelism (continued)

- Partition task into many independent subproblems
  - Factoring and Monte Carlo simulation can be partitioned into many independent subproblems
  - Solve each subproblem, report results to Master task, which records results
  - "Searching" problems can be especially successful compared to sequential computers

Partitioning into subproblems is a powerful technique



Summary of Parallelism Types

- Most parallelism is a complex combination of these and similar techniques …
  - Algorithms are not easy to classify
  - We will learn algorithms to illustrate these ideas
  - In designing algorithms we seek scalable parallelism ...

Scalable parallelism means that an algorithm is designed so it can use "any" number of processors

Non-Parallel Techniques

- Distributed computing, e.g. client/server structure, is not usually parallel
- Divide-and-conquer is usually limited by "sending and receiving" the data instances
- Techniques that assume n^c processors for n size problem
- Almost all techniques that focus on reducing operation counts and building complex data structures

A Sample Computation: Global Add

- Adding a sequence of numbers x_1, x_2, …, x_n
- The standard sequential solution …

```
sum = 0;
for (i=0; i<n; i++) {
     sum = sum + x[i+1];
}
```

- The solution specifies an order of summation that is not essential to the task

An Alternative ...

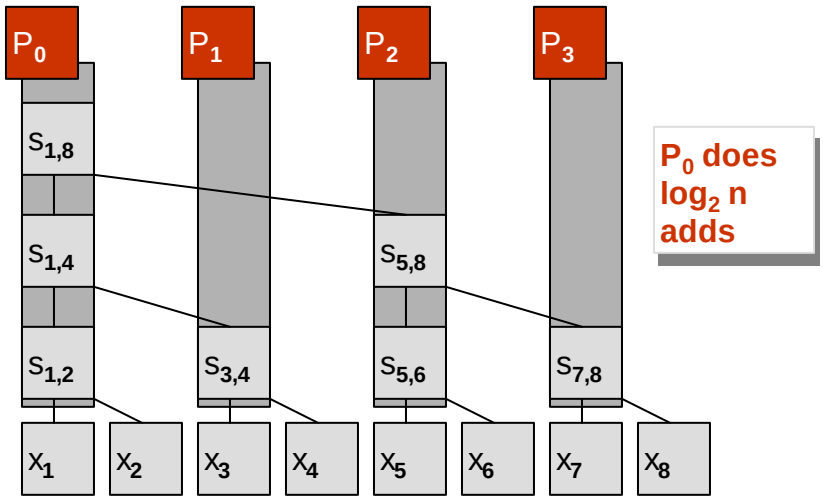
- The “summation tree”
- Exploit associativity of addition …
  - Number the processors 0 to n/2 - 1
  - Processor P_i adds x_{2P+1} and $x_{2(P+1)}$

Common Notation:

n* is used for *problem size

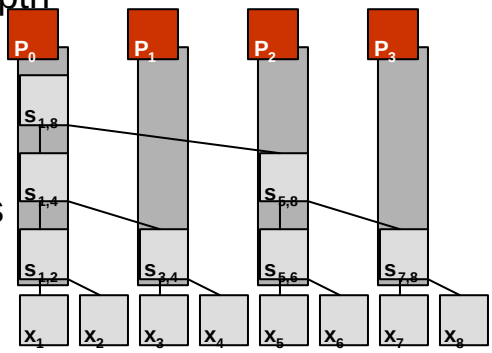
P* is used for *number of processors

Summation tree on 4 processors



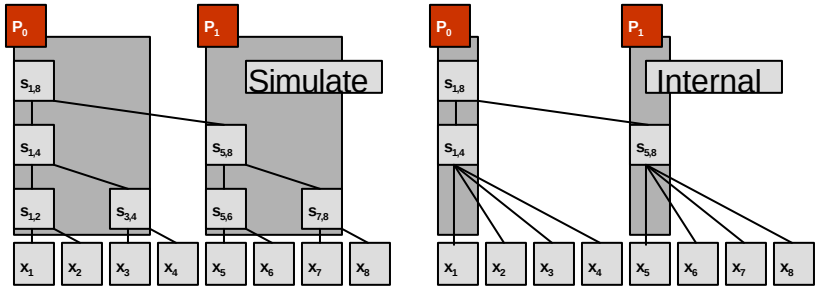
Analysis ...

- Should consider time, processors, comm, etc.
- Time : operation depth
  = $\log_2 n$
- Processors :
  P = n/2
- Space: P temp cells
- Comm = P-1 sends



How would solution scale?

Problems are (almost) always much larger than the available processors, so scaling concerns solving problem w/ fewer processors



Recognize internal structure in the algorithm rather than "simulate" full processor solution

Prefix Sum

Add the prefixes of a sequence of *n* numbers, $x_1, x_2, \ldots, x_n$. That is, $y_k = \Sigma_{i \le k}\ x_i$

$x_1 = 3 \quad x_2 = 1 \quad x_3 = 4 \quad x_4 = 1 \quad x_5 = 5 \quad x_6 = 9$

$y_1 = 3 \qquad\qquad = 3$

$y_2 = 3+1 \qquad\qquad = 4$

$y_3 = 3+1+4 \qquad\qquad = 8$

$y_4 = 3+1+4+1 \qquad\qquad = 9$

$y_5 = 3+1+4+1+5 \qquad\qquad = 14$

$y_6 = 3+1+4+1+5+9 \qquad\qquad = 23$

A Problem with Parallelism ...

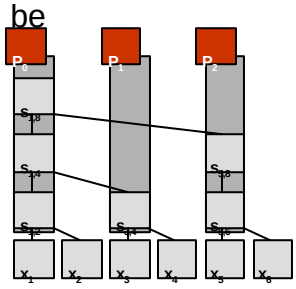
- Each y_i seems to depend on computing the previous item -- it looks very sequential
- An important challenge in parallel computation is to discover how to solve problems when it appears that a "sequential solution is necessary"

One (Not So Good) Solution

- One solution is to apply the summation tree to compute each y_i in parallel …
- This is a "reduce to previous solution" approach
  - Time would be maximum depth, I.e. $\log_2 n$
  - Processor requirements would be

  $1+1+2+2+3+3+ \ldots +n/2+n/2$

  $= \mathbf{n}(n/2+1) = O(n^2)$

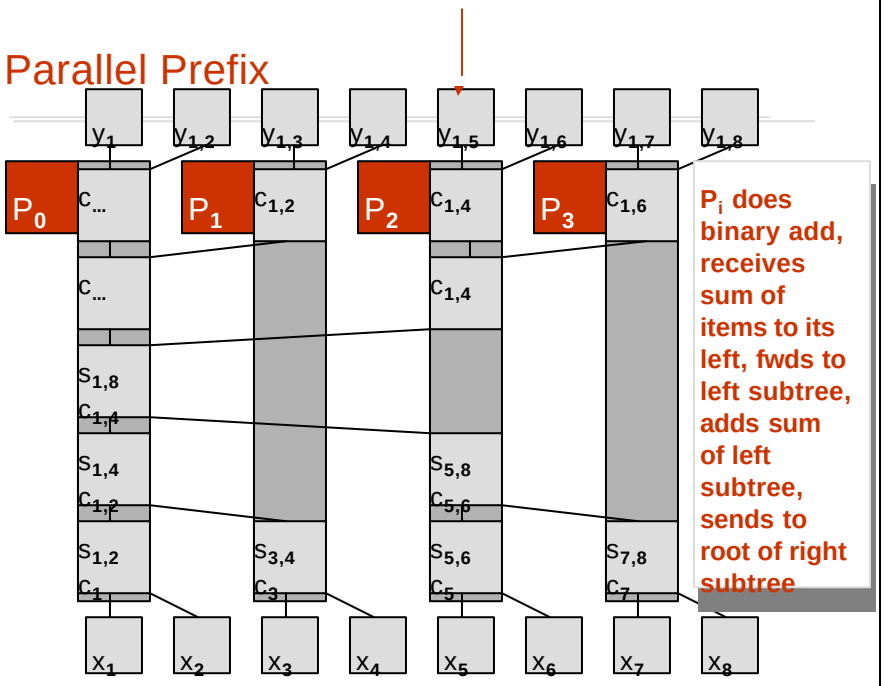


The k = 6 solution

Ladner Fischer [1980] Better Solution

- Parallel Prefix algorithm uses idea like “carry propagation” to reduce work
- In first half of algorithm, each processor adds as in “global sum” technique, but it also passes to its parent the sum of its left subtree
- In second half of algorithm, each processor
  - receives sum of items to its left
  - forwards that to its left subtree
  - adds in the sum of its left subtree
  - sends result to its right subtree
- Every processor i gets data to produce y_i

Parallel Prefix



Analysis

- What resources does the algorithm use?
- Time = 2 $\log_2$ n
- Processors P = n/2
- Space = P memory cells for local sums
- Communications 2 (P-1) sends

The Ladner-Fischer algorithm requires twice as much time as standard global sum

A Bonus ...

- Ladner - Fischer can solve a larger problem in "same" time! [Works for other algorithms]
  - Suppose there are P$\log_2$ P values, stored $\log_2$ P per processor
  - Ask each processor to add the $\log_2$ P items locally, and then solve the problem as stated
  - On completion, compute the prefix for each of the $\log_2$ P elements

Though the problem is $\log_2$ P times larger, the execution time is roughly three times as long

Citation

R. E. Ladner and M. J. Fischer
Parallel Prefix Computation
Journal of the ACM 27(4):831-838

Break Question

- What is the “best” parallel matrix multiplication?
- Problem: For nxn arrays A and B, compute their product C=AB. That is, $c_{rs} = \Sigma_{1 \le k \le n} a_{rk} b_{ks}$
- Assume the A and B arrays are (initially) stored in the parallel processors as blocks, strips or panels (your choice)

Break Problem (continued)

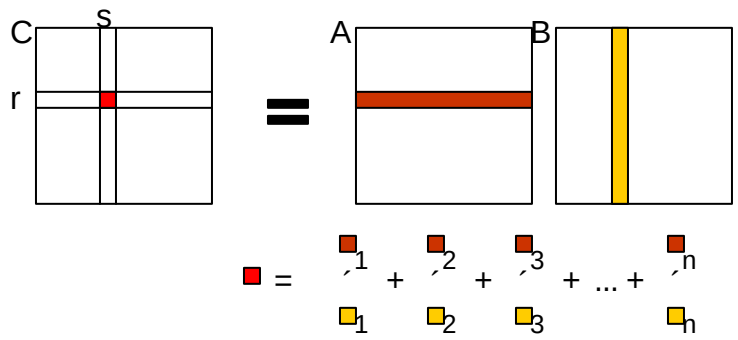
- Goal: My goal is for you to think about how to solve this problem, and to understand what makes it difficult
- Matrix Multiplication is the *most studied* parallel computation, and there have been *many answers* to this problem
- Everyone should try to solve it
  - With an unlimited number of processors
- If you're successful, try to solve it
  - With a limited number of processors P < n

Plan For Discussing MM

- Matrix Multiplication -- easiest solution
- A systolic solution
- A row/column solution + improvements
  - More work per "step"
  - More communication at a time
  - Improve locality
  - Overlap communication with computation
  - Reorient computation
- Discover "best practical" MM
- Review Role of Computation Model

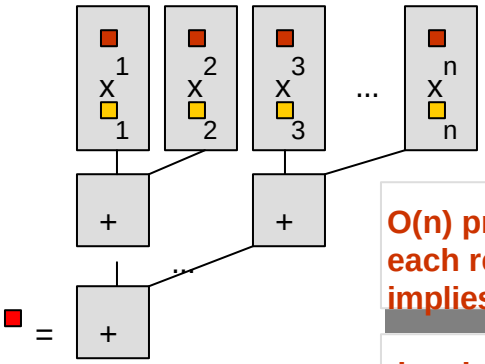
Recall Matrix Multiplication (MM) Definition

- For n ´ n arrays A and B, compute C = AB where $c_{rs} = \sum_{1 \le k \le n} a_{rk} b_{ks}$



Features of MM Computation

- Multiplications are independent, additions can each use global sum tree



O(n) processors for each result element implies $O(n^3)$ total

time is $\log_2 n$

Evaluation of Most Parallel MM

Good properties

- Extremely parallel … shows limit of concurrency
- Very fast -- $\log_2 n$ is a good bound … faster?

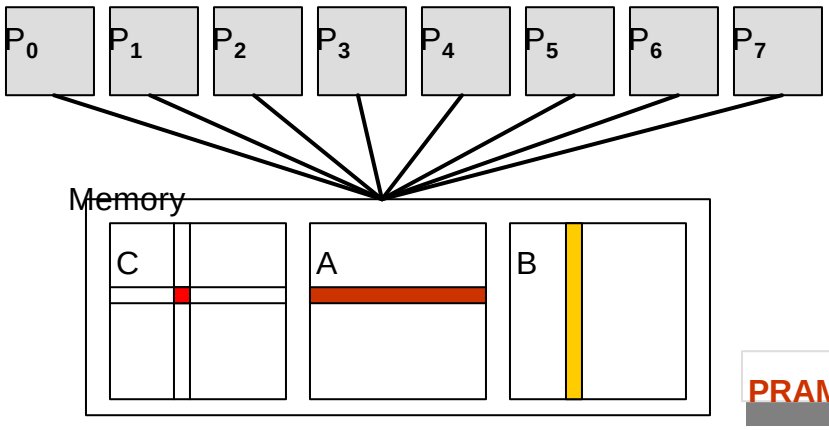
Bad properties

- Ignores memory structure and reference collisions
- Ignores data motion and communication
- Under-uses processors -- half of the processors do only 1 operation

Unrealistically parallel

Where Is The Data?

- Global shared memory is one possibility
- How should it work?



PRAM

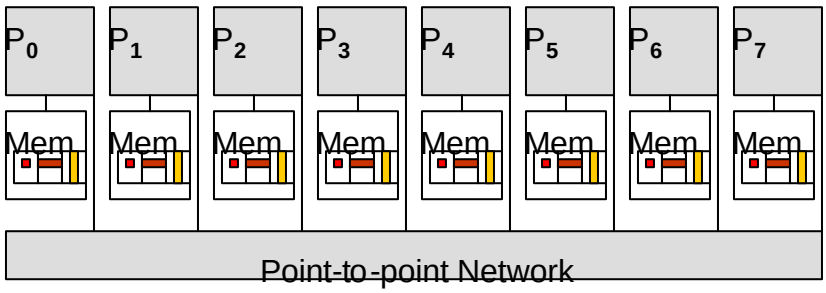
Parallel Random-Access Machine (PRAM)

- Any number of processors, including n^c
- Any processor can reference any memory in "unit time," that is, with no delay
- Memory contention must be solved
  - Reference Collisions -- many processors are allowed to read the same location at one time
  - Write Collisions -- what happens when many processors write to the same memory location?
    - Allowed, but must all write the same value
    - Allowed, but value from highest indexed processor wins
    - Allowed, but a random value wins
    - Prohibited

PRAM well studied

Where Else Could The Data Be?

- Local Memory Distributed Among Processors
- Alternative to Global Shared Memory

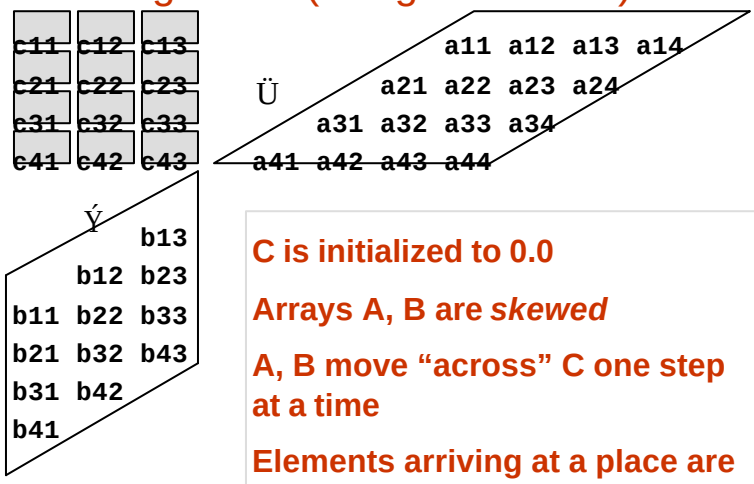


Send/Receive make data motion cost explicit

Alternative Organizations

- Parallelism is perfect for VLSI circuit implementations since many processors could be fabricated on a chip or wafer
- Planar constraints are significant, however
- Reading an *n* ´ *n* array takes *n* time steps
- Data motion is critical to a successful algorithm

Cannon's Algorithm (Kung/Leiserson)



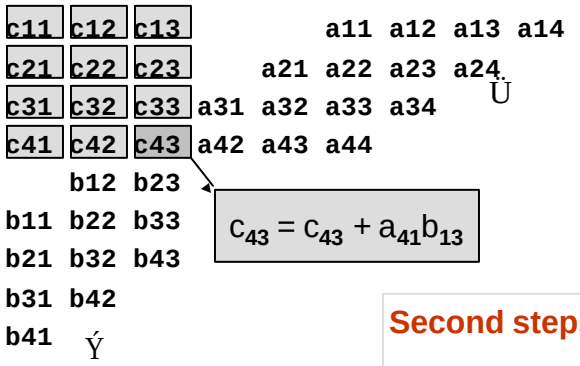
C is initialized to 0.0

Arrays A, B are *skewed*

A, B move "across" C one step at a time

Elements arriving at a place are multiplied, added in

Motion of Cannon's Algorithm



Second steps ...

$c_{43} = c_{43} + a_{42}b_{23}$
$c_{33} = c_{33} + a_{31}b_{13}$
$c_{42} = c_{42} + a_{41}b_{12}$

Analysis of Cannon's Algorithm

- Pipelined algorithm that computes as it goes

  Systolic
- For *n* ´ *n* result, 2*n-1* steps to fill pipeline, 2*n-1* steps to empty it
- Computation, communication are balanced
- Suitable for signal processing -- Cannon's reason for developing it -- where there is a pipeline of MMs

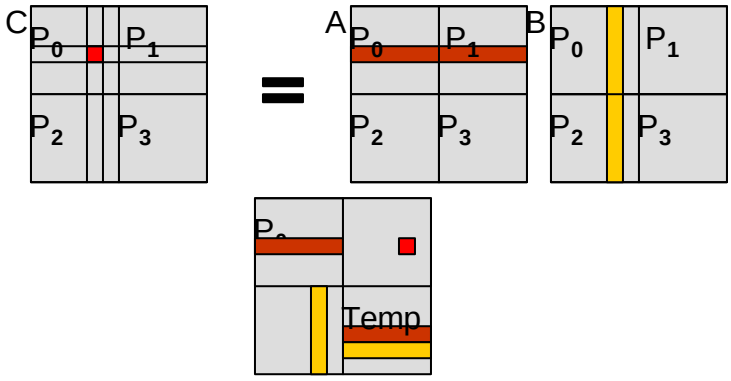
Processors And Memories

Realistically, parallel computers must be made from commodity parts, which means that the processors will be RISC machines with large caches, large memories, support for VM, etc.

- One idea is to use a RISC design as a base and modify it -- it didn't work
- From scratch processors are never fast enough
- Best plan -- try to use the RISC as it was designed to be used

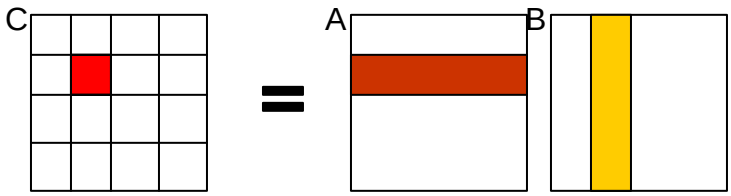
Send Larger Units of Information

- Communication is often pipelined ...
- Send a whole row -- or as much as fits into a packet -- in one operation



Use Processors More Effectively

Assign t rows and the corresponding t columns to each processor so it can compute a t × t subarray



Each element requires n iterations

Modern pipelined processors benefit from large block of work

Computing t × t block

- What is the logic for computing a t × t block?

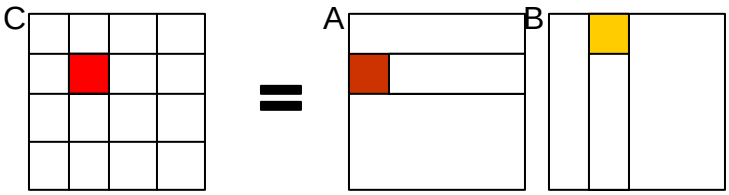
```
for (r=0; r < t; r++){
  for (s=0; s < t; s++){
      c[r][s] = 0.0;
      for (k=0; k < n; k++){
         c[r][s] += a[r][k]*b[k][s];
      }
  }
}
```

Loop is easy to analyze and "unroll"
Branch prediction should work well
This code may be near "optimal"

Locality Can Be Improved

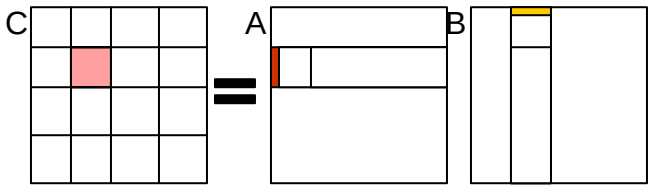
- Improve cache behavior by “blocking”

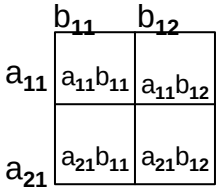
```
for (p=0; p < n; p=p+t){ //block count
  for (r=0; r < t; r++){
    for (s=0; s < t; s++){
      for (k=0; k < t; k++){
        c[r][s] += a[p+r][p+k]*b[p+k][p+s];
      }
    }
  }
)
```



Locality Can Be Improved

- Put operands in registers, “strip mine”

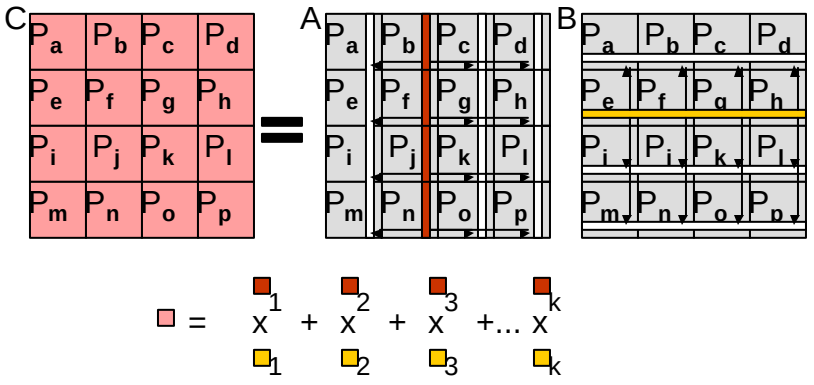




Switch Orientation -- By using a *column* of A and a *row* of B compute all of the “1” terms of the dot product, i.e. use 2t inputs to produce t^2 first terms

Communication Is Important

- So far the algorithm is “send-then-compute”
- Try overlapping and computing
- Use broadcast communication



Communication Pattern

- Architectures differ, but row and column broadcasts are often fast
- Transfer only the segment of row stored locally to the processors in the column
  - For 1 block P_{uv} is a sender
  - For $P^{1/2}$-1 blocks P_{uv} is a receiver
  - Space required is only 4t elements -- 2t for the segments being processed and 2t for the segments arriving

van de Geijn and Watts' SUMMA

- Scalable Universal MM Algorithm
- Claimed to be the best practical algorithm
- Uses overlap, pipelining, decomposition …

Initialize C, blocking all arrays the same

broadcast (segment of) 1st A column to processors in row

broadcast (segment of) 1st B row to processors in column

for i = 2 through n

broadcast (segment of) next A column to all processors in row

broadcast (segment of) next B row to all processors in column

compute i-1 term in dot product for all elements of block

compute last term for all elements of block

Use Groups of Rows and Columns

- For large machines both communication and processing capabilities can usually be helped by processing more than a single row and column
  - Sending more values amortizes start-up costs
  - Pipelines and caching favor large sequences of uninterrupted instructions
- Combine all of ideas of today!

What's Important?

- Maximizing number of processors used
- Minimizing execution time
- Minimizing the amount of work performed
- Reducing size of memory footprint
- Maximizing (minimizing) degree of data sharing
- Reducing data motion (interprocessor comm.)
- Maximizing synchroneity or asynchroneity
- Guaranteeing portability among platforms
- Balancing work load across processors
- Maximizing programming convenience
- Avoiding races, deadlocks, guaranteeing determinacy
- Better software engineering: robust, maintain, debugging

Knowing The Parallel Computer

Was it difficult to design a MM algorithm without a clear idea of how the processors interact with the memory?

- To be effective, programmers and algorithm designers must know *the characteristics* of the computer that will run their algorithms
  - Call this the machine model
  - We forget the importance of the machine model because the sequential model -- von Neumann -- is in our bones … we never think about it
  - Parallelism needs a different model from vN

Next Week ...

We discuss the best parallel machine model for programming parallel machines …

- Will it be the PRAM?
- Will it be VLSI?
- Will it be distributed memory machine with RISC processors?

Citations

- L. F. Cannon [1969] *A Cellular Computer to Implement the Kalman Filter Algorithm*, PhD Thesis, Montana State University
- H. T. Kung & C. E. Leiserson, Systolic Arrays, in Carver Mead and Lynn Conway, *Introduction to VLSI*, Addison-Wesley, 1980
- Robert van de Geijn & Jerrell Watts (to appear). "SUMMA: Scalable Universal Matrix Multiplication Algorithm," *Concurrency: Practice and Experience*, 1998