CSE596 Parallel Computation

Lawrence Snyder

Details: http://www.cs.washington.edu/education/courses/cse596/CurrentQtr/

CSE596 Parallel Computation, Organization

- There is a survey to fill out on the web page
- Text book: *Parallel Computer Architecture: A hardware/software approach*, D. E. Culler and J. P. Singh, Morgan Kaufmann, 1999 -- Chapters 5-7 will be assigned, everything else is optional
- There will be occasional homework assignments and an Exam on March 19th
- Topics by week (revisons possible):

1: Concepts of Parallelism	6: Snooping MP
2: ZPL Programming	7: Scalable MP
3: Assessing Performance	8: Routing, Latency Hiding, etc
4: Parallelism Panorama	9: Programming Paradigms
5: Shared Memory MP	10: Algorithms and applications

Thinking In Parallel

Usually when we formulate a computation, we think of a sequential solution. Good parallel computations rarely result from transforming a sequential solution. A paradigm shift is required. So, it is essential to acquire a "parallel point of view" to produce good parallel computations from the start

A Sample Computation

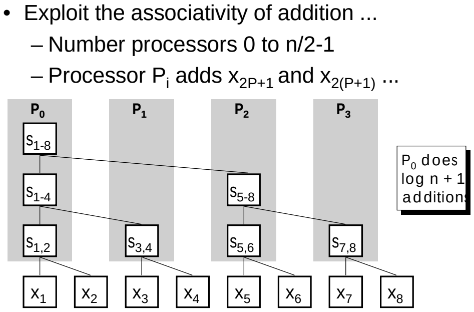
- Consider the problem of summing a sequence of numbers, $x_1, x_2, x_3, ..., x_n$: Σx_i
- Standard solution:

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

- The solution specifies a specific order for the summation, which is not essential

A More Parallel Solution

- Exploit the associativity of addition ...
  - Number processors 0 to n/2-1
  - Processor P_i adds x_{2P+1} and $x_{2(P+1)}$ ...



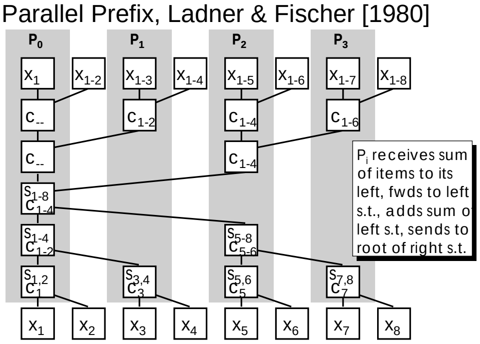
Prefix Sums ...

- Sum the prefixes of a sequence of numbers, $x_1, x_2, x_3, ..., x_n$, such that $y_i = \Sigma_{i<n} x_i$
- Each y_i result seems to depend on computing the previous item
- One solution is to apply the binary tree summation to compute each y_i in parallel ... this would take 1+1+2+2+3+3+...+n/2+n/2

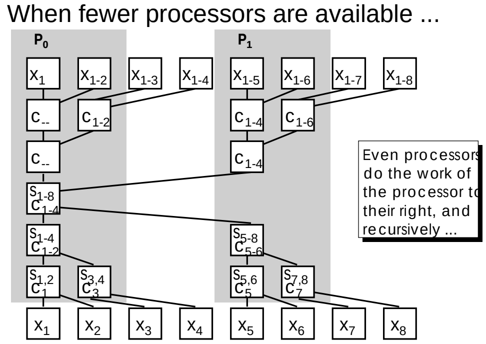
  = n(n+1)/4 processors

  and a lot of data communication

Parallel Prefix, Ladner & Fischer [1980]



When fewer processors are available ...



Essential Features of the Example

- Arbitrary ordering constraints removed by exploiting associativity -- focus on problem characteristics
- Chose direct solution rather than “reducing to an earlier solution” that “over-parallelizes” -- too parallel is no more useful than sequential
- Ladner & Fischer solution can use any number of processors in the range 1 - n/2 -- scalable parallelism is essential in practice

These Guidelines Will Be Elaborated Further

Consider Another Example ...

- Matrix multiplication is a common operation in scientific computing
- The C code for multiplying an mxn matrix A times an nxp matrix B and to produce an mxp matrix C is ...

```
for (i=0; i<m; i++){
   for (j=0; j<p; j++){
      C[i][j] = 0;
      for (k=0; k<n; k++){
          C[i][j] += A[i][k]* B[k][j];
      }
    }
 }
```


Properties of the Computation ...

- Addition and multiplication are associative
- Each position c_{ij} in the result is the sum of the i^{th} row times the j^{th} column ... all of them could be computed simultaneously
- Each position admits plenty of parallelsim ...
  - All multiplys in row i×j column are independent
  - Sum of products could use binary addition tree

Notice that ideas from sequential complexity theory such as Strassen’s Algorithm, that reduces the number of multiplications from $O(n^3)$ to $O(n^{2.81})$, do not apply. Concurrency counts here!

A Very Parallel Solution ...

- Each c_{ij} is computed in parallel such that
  - One processor dedicated to each a[i][k]*b[k][j]
  - Addition tree computes sum of those products
- How many steps?
- How many processors running concurrently?
- Is this solution even remotely practical?
  - Data access -- conflicts/transit time/resources
  - Computation time vs communication time
  - Processor demands -- n^3 procs for n^2 results

Hello?

Realities of Parallel Computers ...

Dissiderata

- Every computer has a fixed number of processors
- Present large computers have a few hundred processors up to a few thousand
- Using all available processors (usually) gives the best performance
- Processors can be very simple, but as first approximation, assume Pentium, PowerPC, MIPS
- The transmission of data from processor to processor is a significant (often the most signficant) cost

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 maybe asynchroneity
- Guaranteeing portability among platforms
- Balancing work load across processors
- Maximizing programming convenience
- Avoiding races, guaranteeing determinacy
- Improve SoftEng... robust, maintain, debug, etc

My Answers ...

NA	Maximizing number of processors used
1	Minimizing execution time
NA	Minimizing the amount of work performed
--	Reducing size of memory footprint
--	Maximizing (minimizing) degree of data sharing
1	Reducing data motion (interprocessor comm.)
--	Maximizing synchroneity or maybe asynchroneity
1	Guaranteeing portability among platforms
7	Balancing work load across processors
4	Maximizing programming convenience
4	Avoiding races, guaranteeing determinacy
4	Improve SoftEng... robust, maintain, debug, etc

These answers are in conflict ...

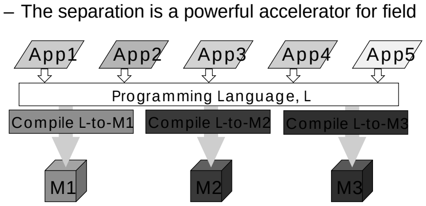
- No. 1 Goals Conflict --
  - Minimizing execution time ==> code close to the hardware
  - Portability ==> keep distance from hardware because machines differ
- No. 1 Goal Conflicts with No. 4 Goal
  - Convenience ==> ignore data motion
  - Minimizing data motion ==> attend to data motion

How are these conflicts solved in the sequential world?

Reason by Analogy to Sequential Case

Sequential languages separate applications development from computers:

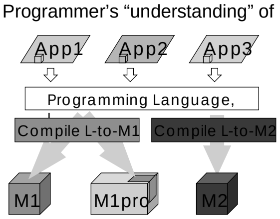
- Architects build machines that run the language well
- Programmers need not worry about machine specifics
- The separation is a powerful accelerator for field



Enabling Technologies

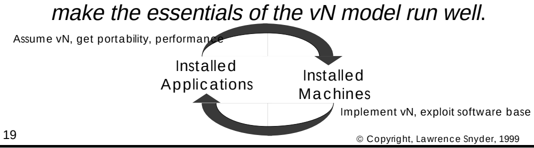
What makes this separation work?

- Instruction set architectures (ISAs)
- Effective compilers that "place the program directly on the iron" with little or no overhead
- Programmer's "understanding" of idealized machine



Machine Model Is The Interface

- The von Neumann machine is the conceptual computer, “running” Fortran or C code
- Imagining the vN machine running the code lets programmer make rough estimates of how alternative solutions will perform.
  - Linear search vs logarithmic search?
- The program runs well *because architects make the essentials of the vN model run well*.

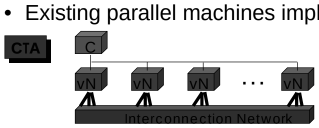


Selecting a Machine Model

- Picking the machine model is subtle
- Like porriage, the model has to be just right
  - Too absract implies performance critical aspects of the computation will not be included
  - Too specific implies the model over-constrains the implementation in a way that may not match physical machines well
- Also, the model must be both intuitive and workable

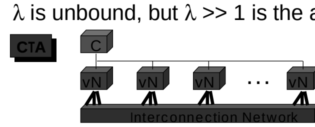
CTA: A Parallel Machine Model

- First practical and general parallel model [‘86]
- Properties emphasize concurrency, locality
  - P = number of processors
  - λ = off processor latency, large
  - Communication network = unspecified, fixed low degree
  - “Thin” global communication capability
- Existing parallel machines implement CTA



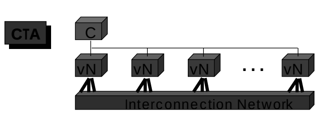
Implications of the CTA

- The processors are von Neumann processors
  - Each has a program counter ==> MIMD
  - Memory local to the processor has fast access
  - Implements sequential thread of execution, but may have multiple processors, memory hierarchy, etc.
- Interconnect’s unbound -- cannot program to it
- λ is unbound, but $\lambda >> 1$ is the assumption



Further Implications of CTA

- The memory is physically distributed (it must be), but there is no mention of for shared address space or shared memory
- Since λ is large, programs exploit locality run faster, i.e. try to compute on data in the local memory
- Fixed degree (usually 1) limits burst rate



Reconsider the Matrix Multiplication

- If every processor had a copy of the A,B matrices, each could compute a rectangular subarray
  - Memory footprint would be huge, $P(mn+np) + C_r$
  - Transfer time of arrays to each memory would be $\lambda(mn+np)$, also huge
  - Optimization -- C[i..i+x,j..j+y] requires rows i..i+x and columns j..j+y
  - Total numeric operations would be O(mpn) which should benefit from a P-way speedup
- Alternatives?

Cannon's Algorithm

Hall of Fame

One of the all time great MM algorithms

Abstractly ...

```
c11 c12 c13              a11 a12 a13 a14
c21 c22 c23  <-      a21 a22 a23 a24
c31 c32 c33      a31 a32 a33 a34
c41 c42 c43  a41 a42 a43 a44
     ^
     |
        b13
    b12 b23
b11 b22 b33
b21 b32 b43
b31 b42
b41
```

A and B are skewed and conceptually "pass across" the result array C that is initialized to 0. As aik and bkj pass over cij, they are multiplied and the result is added into the cij.

Properties of Cannon's Algorithm

- The communication is included in the computation -- compute on the move
- Communication is "nearest neighbor"
- Time is O(n)
- Processors are fully utilized only in the middle of the computation
- Scaling is possible by grouping elements of C
- Skewing and staging data is a complication

Further Reading

- L. F. Cannon [1969] *A Cellular Computer to Implement the (Kalman) Filter Algorithm*, Ph.D. Thesis, Montana State University
- R. E. Ladner & M. J. Fischer [1980] Parallel Prefix Computation, *Journal of the ACM* 27(4):831-838
- L. Snyder [1995] Experimental Validation of Models of Parallel Computation, A. Hofmann & J. van Leeuwen (eds), *Lecture Notes in Computer Science, Special Volume 1000*, Springer, pp. 78-100
- L. Snyder [1986] Type Architecture, Shared Memory and the Corollary of Modest Potential, Annual Review of Computer Science 1, pp. 289-318