Parallel Programming Models (Chapter 3)

Sequential Programming Model
- Von Neumann model
- Programming languages for Von Neumann model

Parallel Programming Model (Heywood, Ranka, JPDC 1992)
- Machine Model -- "node models"
- Architectural Model -- "full parallel machine model"

Computational Model
- "Complexity of an algorithm should reflect the performance on a real computer"

Programming Model
- How programming exploits hardware
- How is parallelism exhibited
  - Implicit or explicit
- Execution mode (SIMD, SPMD, ...)
- Communication methods
- Synchronization methods
- Memory organization
  - Global Address Space
  - Distributed Address Space
Candidate Type Architecture


Linux: /home/phil/public/cs515/TypeArchitectures.pdf

Fundamental Law:
- A parallel solution utilizing p processors can improve the best sequential solution by at most a factor of p.

Type Architecture: Model of a parallel computer

Typical problems that can use parallelism ...
- compute bound
- typically polynomial in n (n size of problem)
- often n^4. x, y, z, time
- time bound, parallel -> larger problem
- t = cn^x
- increase by factor of m
Larger problem (cont)

\[
\begin{align*}
\text{t} &= c (\text{n}m)^x / p \quad \text{(Best speedup!)} \\
\text{m} &= p^{(1/x)}, \quad \text{or} \quad p = m^x \\
x=4, \text{ m}=100 \Rightarrow p = 100,000,000 \\
x=4, \text{ p}=64 \Rightarrow m = 2.828427 \\
x=4, \text{ p}=300,000 \Rightarrow m = 23.403473 \\
x=4, \text{ p}=10,649,600 \Rightarrow m = 57.12593 \\
\end{align*}
\]

Corollary of modest potential

Parallelism doesn’t buy us much ... don’t waste it!

Common attitude I’ve found:

Effeciency doesn’t matter, computers will get faster! See Moore’s Law!

My typical answer:

Yes, computers were getting faster, and now with more parallelism, they are still getting "faster", but think how much better things would be with efficient programs!
Language: Medium is message

Sequential language => sequential solution!
  - hard to get parallelism out of sequential code

Language mapping?
  - sequential -> easy to any sequential machine
  - parallel -> how to translate?
    - Model?
    - PRAM?
      - shared memory
      - constant time access to memory
    - Other?
      - P processors
      - fixed number of edges
      - communication net
Evaluate PRAM (eg. paracomputer)

Problem: maximum

• algorithm?

• Valiant, time \( O(\log \log n) \)

• Stages, \( n(s) \) number of items

• partition \( n(s) \) items into \( r \) sets of equal size (+/-1)

• \( \sum_{i=1}^{r} \binom{|s_i|}{2} \leq p \)

• for set \( s_i \), \( \binom{|s_i|}{2} \) processors assigned

• each processor sets bit \( b_i \) to 1 (initially 0) for looser \( X_i \)

• requires at least common model concurrent write.

• \( X_i \) with corresponding \( b_i \) as 0 is largest

• next round has \( r \) elements in computation

• total number of rounds .... \( \log \log n \).

• Each round constant time.
Example -- 1000 elements

332 sets of 3, 2 of 2, 996+2 = 998 processors => 334 winners

46 sets of 7, 2 of 6, (7:2)=21, (6:2)=15, 46*21+2*15 = 996
  => 48 winners

2 sets of 24, (24:2) = 276, 552 => 2 winners

1 processor chooses ultimate winner

4 stages --- but must charge for concurrent write!

Real hardware would cost \( \log n \) for concurrent write

  => true time \( O(\log n \times \log \log n) \)

Straight forward tree algorithm is \( O(\log n) \)
PRAM based => sub-optimal!
Machine model (type architecture) must accurately represent costs

PRAM can not be realized with constant time concurrent write

Snyders type architecture:
- P processors with local memory
- fixed number of edges
- communication net (fixed degree graph)
- global controller

Hypercube?
- not fixed number of edges

NOW
- fixed edges (1 or 2)
- net fixed ... X is degree of graph

Most computers on Top 500 are essentially CTA architectures.
Parallelization of programs (3.2)

Book’s approach is to take serial program or algorithm and parallelize it.
- Decomposition of the computations
  - granularity -- size of "tasks"
  - data dependencies need to be preserved
- Assignment of tasks to processes or threads
- Mapping of processes or threads to physical processors/cores

Nelson’s approach
- Understand the algorithm
- Find data dependencies required by problem, not just sequential solution
- Look for a parallel algorithm
- Implement the parallel algorithm
Levels of parallelism (3.3)

- Instruction level
  - Tends to be easier to exploit by compilers
- Data parallelism
  - Array processing
  - Big data may have this
  - Problem helps define this
  - SIMD works well for some kind of data parallelism
- Loop parallelism
  - forall i in 1..n do ...
  - body at i can not depend on body at i-1
- Functional parallelism
  - e.g. Divide and conquer algorithms
  - sub problems not in divide and conquer
How to express parallelism in a Programming Language

- implicit parallelism
  - programmer writes code, compiler finds parallelism
  - compiler/system places data at processors as required
  - compiler assigns computation to processors/threads

- explicit parallelism
  - language/system has explicit constructs for parallelism
  - may have "hints" to compiler that a construct is parallel
  - programmer has a way to place data
  - programmer assigns computation to processors/threads

- Synchronization
  - e.g. quicksort: partition, in parallel quicksort both, sync
  - Possible problem: synchronization "wastes time"
  - may be necessary for correct solutions
Execution of a collection of tasks ... by a process (possibly with threads)

- Thread or process creation
- fork/join
- parbegin/parend constructs
- SIMD vs SPMD (remote process creation)
- client/server (not often used)
- pipelining (hardware and software)
  - producer/consumer ...

(skip 3.4 for now (SIMD))

Data Distribution:

- SMP/Shared memory -- all data in shared memory
- Distributed memory -- data distributions are important
  - can have speed implications
  - may need data duplicated
Data Distributions

- **block data distributions**
  - $n$ data, $p$ processors, $n/p$ data blocks to each processor
  - 2-d: rows, column, or sub 2-d structures (checkerboard)

- **cyclic data distributions**
  - $n$ data, $p$ processors, $n/p$ data cyclicly: $i$, $p+i$, $2p+i$, $3p+i$, ..., processor $i$
  - 2-d: rows, columns, cyclic checkerboard

Information exchange (communication)

- **SMP/Shared memory**
  - race conditions, critical sections, synchronization ... (CSCI 322)

- **Distributed memory -> communication between processors**
  - message passing
    - point-to-point: $p_i \rightarrow p_j$
    - broadcast: $p_i \rightarrow \text{all}$ (hardware supported)
communication operations

- single transfer: $p_i \rightarrow p_j$
- single broadcast: $p_i \rightarrow \text{all}$
- single accumulate: $\text{all} \rightarrow p_i$ (reduction, single value result)
- single gather: $\text{all} \rightarrow p_i$ (results in an $n$ long vector)
- single scatter: $p_i$ with vector, individual to all (one-to-all personalized)
- permutation: $\text{all} \rightarrow \text{all}$ 1 message, unique destinations (not in book)
- multi-broadcast: all broadcast: each ends up with a vector (all-to-all broadcast)
- multi-accumulation: each $P$ ends up with a different accumulation (all-to-all reduction)
- total exchange: vector $\rightarrow$ all, all $\rightarrow$ vector (all-to-all personalized)
- parallel prefix sums: $v_i$ in processor $i$, at end of sum, $p_i$ holds sum $j=0,i v_j$ (not in book)

Spanning tree implementations:
- single broadcast / single accumulate (dual) both use spanning tree
- messages relayed at each node
- consider the mesh algorithm: send on $x$, then on $y$. 
Prefix Sums

Prefix sums (aka scan, aka parallel prefix)
P_i, i=0 ... n-1, has V_i
Result -- S_i,  S_i = Sum (k=0, k<i) of V_k

On Hypercube

procedure prefix_sum (id, d, data, result)
{
  result <- data;
  msg <- data;
  for i <- 0 to d-1
    other <- id XOR 2^i
    send msg to other;
    receive newdata from other;
    msg <- msg + newdata;
  if (other < id) result <- result + newdata
}
PrefixSum(array x, array s, int n)

Input:  x1 - xn
Output: s1 - sn

1) if n=1, set s1 <- x1, exit

2) for each i, 1 <= i <= n/2 in parallel do
   yi <- x_(2i-1) + x_2i

3) PrefixSum(y,z, n/2)

4) for each i, 1 <= i <= n in parallel do
   i even:  si <- z_(i/2)
   i = 1:  s1 <- x1
   i odd > 1: si <- z_((i-1)/2) + xi
Parallel matrix-vector product

- $\mathbf{A}\mathbf{b} = \mathbf{c}$, $\mathbf{A}$ is a matrix, $\mathbf{b}$, $\mathbf{c}$ vectors. $\mathbf{A}$ size $n \times m$, $\mathbf{b}$ size $m$, $\mathbf{c}$ size $n$
- Seq: $C_i = \sum_{j=1}^{m} a_{ij} \times b_j$

- Parallel solutions?
  - Shared Memory?
    - $P = m$? $m \log n$?
  - Distributed Memory?
    - $\mathbf{A}$, $\mathbf{b}$ and $\mathbf{c}$ distributions?
    - $n \times m$ mesh?
    - hypercube?
Implementation Strategies

- Standard programming language and a Library
  - C + Pthreads
  - C + OpenMP
  - C + MPI
  - C + Cuda (with new pragmas, modified compiler)

Design a new programming language
- Unified parallel C
- ZPL (Z-level Programming Language)
- DARPA HPCS language challenge
  - Fortress
  - X10
  - Chapel