History of Parallelism ...

HPC -- High Performance Computing

- 1970s: (1976/77 numbers, each a single CPU)
  - Big Iron: 1975-7 Cray-1, 5.5 Tons, 8MB memory, 160 MFLOPS, $8 Million (64 bit)
  - Mainframe: IBM System/370 Model 138, 1MB memory, ? FLOPS, $350 K (32 bit)
  - Mid/small: DEC pdp 11/70, 4 MB memory, .6 MIPS, $30-50K (16 bit)
  - Micro Processor: Intel 8086, 1MB memory max, .3 MIPS, $360 (16 bit)
- Reference to today:
  - $360 in 1975 is now about $2,010, $8 Mil is now about $44.6 Mil
  - iPhone 16 Pro: Apple A18 Pro chipset, 4040 Mhz max clock, $999 and up to $1500
  - 6 core CPU, 2 performance, 4 efficiency, 6 core GPU, 16 core neural engine
  - 35 TIPS, 2289 GFLOPS
  - 8 Gig Ram, Up to 1TB nvme storage, camers (not available in 1976)

Key idea:

- Cray-1: Big and expensive, "Elephant"
- Intel 8086: Small and "inexpensive", "Can an army of ants out perform the elephant?"

Decades starting in 1970s

- 1970
  - slow machines
  - super computers
  - appearance of microprocessor
- 1980
  - VLSI
  - availability of real parallel machines
  - lots of research/activity
- 1990
  - real parallel machines fall out of favor
  - network of workstations, small clusters
  - first GPUs for faster graphics
- 2000
  - very fast i386 machines
  - multiple threads per cpu ...
  - CPU speeds stopped doubling every 18 months
- 2005
  - multi-core CPUs as standard
  - CPU speeds stagnant or very slowly increasing
    - 3.2 GHz P4 -- Old?
- 2010
  - bigger clusters
  - more CPUs per box
  - more and more powerful GPUs

Decades starting in 1970s (Page 2)

2020

even bigger clusters

even more CPUs per box, lots of memory

GPUs for computation only, no graphics heads (AI needs)

Cluster Sizes today ... top500.org (November 2024)

- 1: El Capitan, HPE Cray EX255a, AMD 4th Gen EPYC 24C 1.8Ghz, ... 11,039,616 cores, 1,742.00 PFLOPS (US)
- 2: Frontier, HPE Cray EX235a, ... 9,066,176 cores, 1,353 PFLOPS (US)
- 3: Aurora, HPE Cray EX, ... 9,264,128 cores, 1,012 PFLOPS (US)
- 4: Microsoft DNv5, Xeon Platinum ... 2,073,600 cores, 561.2 PFLOPS (US, azure)
- 5: HPC6 - HPE Cray EX235a, ... 3,143,520 cores, 477.90 PFLOPS (Italy)
- ...

Moore’s "Law" --

1965: Gordon Earle Moore, co-founder of Intel

- capabilities of computing double every 18 months to 2 years ...
- speed -- quit doubling every 18 months to 2 years around 2000
- total computational power still doubling
- Now ... parallelism is the way to continue to double

Definitions:

- Thread -- "Thread of execution", requires an associated stack
- Processor -- "A CPU" -- physical central processor, may be hyper-threaded
- Pipelining -- "Typically a series of operations done in hardware"
- Parallel Machine (old) -- "A single machine with a number of processors where all processors are working together to solve a single problem. Each processor is running one thread or one thread per hardware thread."
- Distributed Computing -- "A collection of machines, possibly geographically distributed, each running an independent OS and running a variety of threads (processes), working together in some manner."
- (more recently) Parallel Computing: "A collection of cooperating threads working together to solve a single problem."

- Nelson’s Definitions:
  - Parallel: Geographically close, high speed connections, one or limited applications running
  - Distributed: Geographically distributed, "slow" connections, many applications

  - A machine with 8 CPUs running a collection of unrelated processes is not doing "parallel computation".
  - A machine with 8 CPUs running one process with 8 threads running at the same time may be doing "parallel computation".

  - HPC (High Performance Computing) -- Typically done on a cluster now and/or GPUs

Why parallel computing?

- Historically -- machines were slow
- 1 machines is not fast enough to compute results in "time budget"
- Sounded interesting ....
- VLSI -- how to use it ...
- Memory Speed and Disk speed ... vs processor speed
- Data communication advances ...

How to use parallel computation

- Engineering & Design
  - e.g. airfoil design, modeling and simulation
- Scientific Applications
  - e.g. computational physics, biological work, chemistry ..
- Commercial Applications (often distributed not parallel)
  - e.g. distributed databases, distributed web servers, machine learning, ...
  - google ...
  - Computational finance (10% of top 500 as reported about 5 years ago.)
- Government
  - e.g. weather prediction, military applications, ...
- Computer Science Applications?
  - e.g. cryptography ... SHA-3 competition
  - Deep learning, AI, ...
- Eijkhout HPC intro has a good section on Applications

Architectures ... (Historical view)

Eijkhout HPC Intro section 1, Pacheco Chapter 2, Grama Chapter 2

- Sequential Computer
  - CPU
  - Memory
  - Connection between the two

- Parallel Computer (early ones built, some current ones with NUMA, non-uniform memory access)
  - n CPUs
  - m Memories
  - Connection from n to m ...
  - variety in kind of CPU

- Speed up attempts on sequential computers
  - Pipelining, data & instruction
  - Multi pipelines, "super scalar"
  - Very Long Instruction Word processors

Speed ups -- Memory caching

- 1 GHz CPU (1ns cycle time) vs 100ns memory
- memory access is bottle neck. (100 cycles/mem op)
  - 3.4GHz CPU, 800 MHz memory (2007 CPU, PC6400 mem)
    - .294 ns CPU clock time, 3.75 - 10 ns memory cycle
  - 4.8GHz-5.2GHz CPU, 3600 MHz memory (2025 AMD Ryzen 7, 9800x3D, DDR5)
    - .2 ns CPU clock time, 14 ns memory latency, higher bandwidth, 4ns prefetch
  - Add multiple CPUs accessing same memory
- Cache memory between CPU & memory
  - smaller than main memory
  - faster than main memory (e.g. 0.2ns)
  - much more expensive
- Cache fetch & hit ratio
- Memory bound programs depend on hit ratio
  - memory layout makes a difference!
  - Row major layout, column first access
- Multi-threading & prefetching ...

Classes of Parallel computers (Flynn’s Taxonomy)

- Single Instruction, Single Data (SISD) -- no parallelism

- Single Instruction, Multiple Data (SIMD)
  - all cpus execute same instruction at same time
  - parallel done by different data
    - for all i in [0 ...999] c[i] = a[i] + b[i]
  - mask to turn off some data element ...
    - for all i in [1 .. 100] if (i % 2) c[i-1] += c[i]

- Multiple Instruction, Multiple Data (MIMD)
  - each cpu has own program and data
  - synchronization by communication
  - no lock step execution
  - CPU access to memory

- Single Program, Multiple Data (SPMD) ... not really a parallel computer or in Flynn’s Taxonomy
  - Usually run MIMD machines
  - Single program not in sync like SIMD
  - Runs well on cluster or multi-core CPU or both

Communication ... Global Memory

- One memory, Many CPUs. Synchronization for access
- Uniform Memory Cost (unrealistic)
- Log time Memory Cost (NUMA architectures)
- cache coherence problems

- Typical theory models ... PRAM (Casanova Chapter 1)
  - concurrent read, concurrent write (CRCW)
  - exclusive read, exclusive write (EREW)
  - concurrent read, exclusive write (CREW PRAM)
  - Concurrent write: (CRCW)
    - common (or "consistent mode") -- all write same value
    - arbitrary - arbitrary PU writes
    - priority - PU with lowest number wins
    - fusion - some operation, eg sum, and, or, ...
  - All PUs execute the same "program" syncronously.
  - Can have some processors not execute during a step

- Lots of theory done using this model

Communication with Non-shared memory

- CPU with private memory
- Communication Network (message passing)
  - Bus based: broadcast, limited bandwidth, cheap
  - Crossbar switches: full bandwidth, expensive
  - multi-stage crossover: omega network, log p depth
    - moderate bandwidth, moderate expense
    - blocking access
  - completely connected
  - star
  - mesh
    - linear
    - 2 d mesh
    - 3 d mesh (weather modeling ...)
    - hypercube n-d, 2 node mesh
  - trees
  - fat tree

Network of Workstations

- Common "Parallel Machine" today
- Connections ...
  - High speed "switch" (Inifiniband is a common one now)
    - our old ones, 100GB/s
    - our new ones, 400GB/s
  - High speed ether (not quite 400GB/s)
  - May require more than one switch

Network of Custom Build Nodes -- e.g. Silicon Mechanics Clusters

- WWU compute cluster:
  - 1 head node: 6 Cores
  - compute nodes: 128 cores (256 hardware threads)
  - GPU nodes: 88 cores, 165,308 CUDA cores, 1368 tensor cores

Static Interconnection Networks ...

Measures

- Diameter -- max distance between 2 nodes
  - ring of p -- p/2
  - mesh -- 2 * (sqrt(p) - 1)
  - hypercube -- log (p)
- Connectivity -- multiplicity of paths between nodes
- Arc connectivity -- minimum number of arcs to remove to disconnect net
  - 1 -- linear array, tree, star
  - 2 -- rings & 2-d meshes (4 on 2-d meshes with wraparound)
  - log p -- hypercube

Measures (page 2)

- Bisection Width -- number of links to remove to equally partition nodes
  - mesh - 2 sqrt(p)
  - tree - 1, star - ?
  - hypercube -- do it in class ...
- Bisection Bandwidth -- number of bits / unit time
  - channel width -- number of bits at same time
  - channel rate -- peak rate per "wire"
  - channel bandwidth -- peak rate per connection
  - Bisection Bandwidth = channel bandwidth * bisection width
- Cost -- total number of links
  - liner array, trees -- p-1
  - d-dim wraparound mesh -- dp
  - hypercube -- (p log p) / 2 links
- How about our machine?

Cache Coherence in MP Systems

- shared memory
- each processor has own cache
- reads and writes to same memory locations by >1 processors
- hardware protocols to invalidate cache entries ...
- Books do more detail ... I’m not as interested ... (Grama and Pacheco)

Communication Costs -- message passing

- Used to talk about communication costs on various "networks"
- Now, primarily infiniband or ether
  - ether can be switched, bisection width?
  - infiniband can do TCP/IP over it!   Again, bisection width

Other "problems"

Algorithm connection graph vs hardware graph

- hypercube in a 2-d mesh
- 2-d mesh in linear
- ... in NOW

Tradeoffs -- parallel computing is full of them

- cost - performance
- algorithm to match hardware ?
- keep looking ...

Parallel Algorithms & Design (Ch 3)

Sequential:

- Data & program

Parallel:

- What can be done in parallel?
- mapping concurrent work -> multiple threads
- distribution of data to threads
- "shared data" management
- Synchronizing the threads

- Approach
  - from serial program?
  - from problem?

Jacobi Iterations

Code:

```
double x [1..1000, 1..1000], nv, diff;  int i, j;

/* initialize  X: all to zero except fixed locations ...*/
repeat
  diff = 0;
  for i <- 1 to 1000 do
    for j <- 1 to 1000 do
      if (not a fixed location) { // edge code ignored
        nv <- (x[i-1,j] + x[i+1,j] + x[i,j-1] + x[i,j+1])/4
        diff <- max (diff, abs(x[i,j]-nv))
        x[i,j] <- nv
      }
until diff < tolerance
```

- solution of a ODE.
- loop parallelism?
- data dependency in original code not necessary
- could calculate all into a new array
- communication?

How to design a parallel algorithm

Ignore current state of automatic tools to do:

sequential code -> tool -> parallel solution

Best to start again with parallel in mind

- Start with problem
- Look for parallelism
  - dependency graph (e.g dense matrix multiply)
  - amount of parallelism?
  - expected speed of graphs?
- Consider available parallelism
  - granularity
    - cluster vs multi-processor
  - degree of concurrency
  - communication bandwidth

Other considerations

- Data & Thread placement
- Thread interaction
- Processes vs Processors

"Decomposition Techniques"

- Divide-and-conquer (recursive)
- Data Decomposition (book)
  - partition data first ... then look
- Exploratory Decomposition
  - search tree decompositions
- Speculative Decomposition
  - Parallel discrete event simulation
  - (e.g. evaluating more than one switch ...)
- Hybrid Decompositions

Other issues

Let the problem help design the solution!

- Task generation
  - static situation
  - dynamic situation
- Task size
  - Very small to Very Large
- Task data
- Task communication
  - static vs dynamic
  - regular vs irregular
  - read-only vs read/write
  - one-way vs two-way
- Load Balancing
  - static mapping
  - dynamic mapping

Other issues (page 2)

Array Distribution Schemes (distributed memory systems)

- Block distributions
  - Row-wise
  - Column-wise
  - Sub-matrix
- Cyclic Distributions
- Block Cyclic Distributions
- Randomized block distribution
- Graph Partitioning
- Mappings based on task partitioning (NP complete)
- Hierarchical Mappings
- Dynamic Mappings
  - Centralized (master/slave)
  - Distributed Schemes

Other issues (page 3)

- Reducing interaction overheads
  - maximize data locality
  - minimize data exchange
  - less frequent interaction
  - Minimize Hot spots & contention
  - Data replication (data locality)
  - Overlapping computation and interactions
  - Overlap interactions

- Avaliable libraries
  - LINPACK -- linear algebra package
    - BLAS - basic linear algebra subprograms
    - started in 1970
    - now has parallel LINPACK
  - may be others

Parallel Algorithm Models (CH 3.6)

- Data-Parallel
  - Compute-Aggregate-Broadcast
- Task Graph Model (tasks usually large)
- Work pool model
- Master-slave model
- Pipeline (producer consumer)
- Hybrid

Communication operations (Ch 4)

- Broadcast: one to all
  - Jacobi: delta value, do we need to continue
- Reduction: all to one (aka aggregation)
  - Jacobi: doing a minimum across all computed values
- All to All broadcast
  - all nodes do "1 to all broadcast"
  - limiting factor on speed?
- All to All personalized -- unique messages
  - n x n matrix on n processors, transpose
- All to All reduce
  - different "all to 1 reductions" at same time
- all-reduce
  - all to 1 reduce, 1 to all broadcast
- Scatter operation: aka one to all personalized communication
- Gather operation: all to one personalized communication

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_i

prefix sums on a CREW PRAM

PrefixSum(array x, array s, int n)

Input: x[1] - x[n]

Output: s[1] - s[n]

Temp: y[1] - y[n/2], z[1] - z[n/2]

1) if n=1, set s[1] <- x[1], exit

2) for each i, 1 <= i <= n/2 in parallel do

   y[i] <- x[2i-1] + x[2i]

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

4) for each i, 1 <= i <= n in parallel do

   i = 1: s[1] <- x[1]

   i even: s[i] <- z[i/2]

   i odd > 1: s[i] <- z[(i-1)/2] + x[i]

Prefix sums

non-recursive on a EREW PRAM?

```
PrefixSum(array x, array s, int n)
   input: x[0] - x[n-1]   output s[0] - s[n-1]
   temp:  z[0] - z[n-1], t[0] - t[n-1], d

   1) for all i in [ 0 .. n-1] in parallel do
         s[i] <- x[i], z[i] <-x[i]
   2) if i = 1, exit
   3) d <- 0
   4) while d < log n
        for all i in [0 .. n-1] in parallel do
           j <- i XOR 2^d
           if (j < n)
              t[i] <- z[j]
              z[i] <- z[i] + t[i]
              if (i > j)
                 s[i] <- s[i] + t[i]
        d = d + 1
```

"Communication" pattern?

Prefix sums on a 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
  }
```

Prefix sums on a cluster where n is much larger than p?

- distribute the data?
- what needs to be communicated?
  - Pattern?

Analysis of a parallel algorithm

Sources of "overhead"

- lack of parallelism (idling)
- communication
- excess computation
- synchronization time
- poor algorithm

Measures

- Time -- sequential
  - CPU
  - clock
- Time -- parallel
  - clock
  - total -- Sum of CPU over all processors
- Overhead time
  - $T_o = p * T_p - T_s$

Speedup

Performance gain

- $S = T_s / T_p$
- Typically use O-notation
- Example - sum: n numbers, n processors
  - T_s is $O(n)$
  - T_p is $O(\log n)$
  - S is $O(n / \log n)$
- Use best know sequential algorithm
- Limits to speedup?
  - At best P!
  - <P => what?
  - >P => what?
  - cache effects ?
  - search tree effects?

Efficiency & Cost

Efficiency

- $E = S / p$
- Often may give a measure of use of processors.
- e.g. tree addition: $E = O(1/\log n)$

Cost

- $C = p * T_p$ (total time?)
- Cost-optimal
  - C & T_s has same growth
  - efficiency of Theta(1)
  - example: sorting $(\log n)^2$ for n PEs, $n \log n$ for seq
    - $S = n / \log n$, $E = \log n$, $C = n (\log n)^2$

Mapping Algorithm -> Machine

In practice, n != p.

- choose less processors
- choose less data
- n/p data items per processor?
- simulate n processes vs change algorithm
- adding n numbers on p processors
  - n on n: time T(log n)
  - n on p: (mapping wrap) time T((n/p) log p)
  - n on p: (mapping block) time T(n/p + log p)

TERMS

- Course grain, Fine grain
- Scalability, how does it work on a variety of n&p
- Overhead! (T_o total overhead)

Scalability of Parallel Systems

Speedup vs number of processing elements

- for various problem sizes
- "results"
  - increase p, efficiency goes down
  - increase n, efficiency goes up

Type Architecture & Corollary ...

Lawrence Snyder:

"Type Architectures, Shared Memory, and the Corollary of Modest Potential"

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)

- $t = c (nm) ^ x / p$ (Best speedup!)
- $m = p ^ (1/x)$, or $p = m ^ x$
- x=4, m=100 => p = 100,000,000
- x=4, p=64 => m = 2.828427
- x=4, p=300,000 => m = 23.403473
- x=4, m=57.5902 => p = 11,000,042
- processor speed: sequential vs parallel

Corollary of modest potential

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

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) for items X_i, 1 <= i <= n
- Stages, n(s) number of items, p processors, p = original n
- Stage:
  - partition n(s) items into r sets of equal size (+/-1)
    - where sum (i=1 to r) (|s_i| choose 2) <= p
  - set b_i, 1 <= i <= n(s), to 0
  - for set s_i, (|s_i| choose 2) processors assigned
  - each processor compares two elements of s_i,
  - each processor sets bit b_i to 1 for looser X_i in comparison
    - 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.
  - There is a way to do the following in constant time
    - compute r
    - assign processors for comparisons
    - move the r winners to the "bottom of the array"

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 * log log n)

Straight forward tree algorithm is O(log n)
PRAM based => sub-optimal!

Costs

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 / Clusters

- fixed edges (1+, fixed hardware)
- net fixed ... X is degree of graph
- HPE CRAY El Capitan -- top of Nov 24 top 500.
  - 11,039,616 cores, AMD 4th gen EPYC 24C 1.8GHz
  - interconnect: slingshot-11
    - fixed degree interconnect, GPUs connected (one connection) to slingshot
    - Based on the Rosetta chip, 64 ports running at 200Gb/sec
    - clearly need a network of these chips for 11M cores
    - Aggressive adaptive routing, advanced congestion control, very low average and tail latency, ...

Parallel Algorithm Design (Foster Ch 2)

Approach to designing parallel algorithms

- Many problems have several possible parallel solutions
  - SMP vs Cluster algorithms
  - GPU vs CPU
- Foster proposes 4 stages
  - Partitioning -- problem to "small tasks"
  - Communication -- communications to allow task execution
  - Agglomeration -- possible combining of tasks for improved performance ...
  - Mapping -- assigning tasks to processors
- Design patterns (Not in Foster)
  - Divide and conquer / Recursion
  - Others talked about in previous slices (CAB, pipeline, task graph, work pool...)
- Partitioning
  - Domain Decomposition (Data parallel)
  - Functional Decomposition (Pipeline, task group, ...)
  - Avoid redundant computation if possible / redundant storage
  - Solution scale?

Parallel Algorith Design (Pg 2)

- Communication
  - local communication with small tasks
  - global communication
  - looking for balance of communication / computation
  - structured, static vs unstructured, dynamic
    - Foster’s example: finite element methods, irregular objects or high resolution in areas
  - asyncronous vs syncronous
- Agglomeration
  - Review previous two stages
  - May need to consider specific hardware here
  - Different groupings provide different communication patterns
    - e.g mapping trees to processing elements
  - Increase of granularity / placement of data in tasks
  - Avoiding communication

Parallel Algorithm Design (Pg 3)

- Mapping -- "tasks" to processors
  - quite "hardware" dependent ... although with Clusters as our primary tool now ...
    - Big question is GPU vs CPU now
  - load balancing
    - recursive bisection (finite element work)
    - local algorithms
    - cyclic vs block -- best is algorithm dependent
  - task scheduling
    - not needed in some algorithms
    - Manager/Worker -- various versions
    - Task pool
    - Termination detection
- Foster has a number of examples
  - Atmosphere Model
    - collection of PDEs and other equations
    - 9 point horizontal, 3 point vertical stencils
  - Floorplan Optimization, VLSI
  - Computational Chemistry
    - 4 deep nested loops (n^4!)

Algorithms: (Grama Ch 8, Eijkhout Ch 6) Dense Matrix

Some of these algorithms are scattered around the books ..

Mapping of data (From Ch 3.4 ...)

Partitioning methods (n x n array to PEs)

- Striped: row or column to one PE
  - block: contiguous rows or columns
  - cyclic: row1 -> pe1, row2 -> pe2, ...
  - hybrid: combination of two
- Checkerboard:
  - block: rectangular sub matrix
  - cyclic: pe0 has (0,0), (0,4), (4,0), (4,4) ...

Transpose

Given A, n x n, A^T[i,j] = A[j,i]

- Data placement ?
- Communication pattern?
  - May be all-to-all personal communication!
- Cluster vs SMP?
- How about p < n^2?
- Cost, speedup, efficiency, ... ?

Matrix - vector multiplication (Special case of matrix matrix multiply!)

A: n x n times B: n x 1 -> C: n x 1

Sequential algorithm?

- Time O(n^2)

Parallel?

- PRAM?
- n processors
  - A: row striped
  - B: in every processor / one per P -> broadcast
  - C: one per P
  - Time is O(n), O(n) processors (broadcast O(n))
  - Speedup O(n). Cost O(n^2)
- < n processors?
  - Store multiple rows per processor (n/p?)

Matrix - vector (page 2)

p = n^2 ?

- An element of A to each processor
- B and C placement?
  - On the first row or column?
  - On the diagonal?
- Column broadcast of B.
- Row summing for C.
- Times?
  - mesh: O(n) cost: O(n^3)
  - Hypercube: O(log n) cost : O(n^2 log n)
- On fewer than n^2 processors?

Matrix - Matrix multiplication

A: n x n times B: n x n => C: n x n

Sequential algorithm?

- Time O(n^3)

Simple Algorithm

- p=n^2
- Each processor has a single element of A and B
- All-to-all broadcast of Matrix A in each row
- All-to-all broadcast of Matrix B in each column
- P_i,j has A_i,0 ... A_i,sqrt(p) and B_0,j ... B_sqrt(p),j
- Can calculate C from that data.
- Issues
  - Communication ?
  - Computation
    - n Multiplies, n-1 additions => n
  - Storage:
    - n^3 total across all processors
  - Total time?

$p < n^2$

Block decomposition $n/\sqrt{p} \times n/\sqrt{p}$ per processor

- Issues
  - Communication?
  - Computation times?
    - n^3/p
  - Total time?

Cannon’s Algorithm

A Memory efficent algorithm for matrix multiply

Basic did all communicate, then all compute

Cannons does communicate & compute at same time

Does mesh communication pattersn ... assumes an nxn mesh with end-around-connections

Cannons basic algorithm:

- a) shift .. A in row left, B in column up
  - row/col 0 by 0 places, row/col 1 by 1 place, ...
- b) for i gets 1 to n do
  - compute C += A * B
  - send A on row, B in col (right & down)
- All Cs are now complete.
- Time on a mesh?
  - n^3/p + 2 * sqrt(p) * (t_s + t_w n^2/p)
- Memory?
  - No extra memory required (may use a copy)
  - Needs n^2/sqrt(p) memory for data

Nelson's Algorithm (Paper on Ubuntu systems in public/cs515)

2x2 Matrix multiply -- Notice pattern

Algorithm:

- Decomposition of nxn into 2x2 problems.
- Recursively solve 8 sub-problems

Example:

- 8 x 8 example
- time?

Expansion to n^3 processors

- time?

Reduction to less than n^2 processors

DNS algorithm (Dekel, Nassimi, and Sahni)

- O(log n) time, O(n^3 / log n) processors
- Read it.