SHARCNET™

High Performance Computing

David McCaughan, *HPC Analyst*
SHARCNET, University of Guelph
`dbm@sharcnet.ca`

SHARCNET™

The Need for Parallelism

- Scientific investigation traditionally takes two forms
  - theoretical
  - empirical
- The increasing speed and accessibility of computers facilitates a *computational* approach
  - use computer models to study phenomena too irregular for theoretical treatment and too large or cumbersome for an experimental approach

HPC Resources

SHARCNET™

The Need for Parallelism

- So, just build faster hardware (non-trivial exercise)
  - "to pull a bigger wagon it is easier to add more oxen then to find (or build) a bigger ox" (Gropp et al.)
  - issues:
    - complexity
    - availability
    - cost

HPC Resources

SHARCNET™

Example

- Something to think about (adapted from Pacheco, 1997)
- *Problem*:
  - predict the weather over Canada and the US
  - assume:
    - model atmosphere from sea level to altitude of 20km
    - need to make prediction at each hour for the next 2 days
- *Standard approach*:
  - predict weather at each vertex of a grid covering the region
  - use a cubical grid with each cube 100m (0.1km) on each side
    - area of Canada + US ≈ 20 million km^2
    - 2.0×10^7 km^2 x 20km x 10^3 cubes/km^3 = **4×10^{11} grid points**

HPC Resources

Example (cont.)

- Some numbers
  - assume 100 calculations to compute weather at each grid point
  - predicting the weather one hour from now requires about 4×10^{13} calculations
  - to predict weather at each hour for 48 hours:
    - 4×10^{13} calculations x 48 hours ≈ 2×10^{15} calculations
  - if we can execute 10^9 calculations per second it will take
    - 2×10^{15} calculations / 10^9 calcs per sec = 2×10^6 sec. ≈ 23 days!!
- But so what…computers keep getting faster, right?
  - assume 10^{12} calculations per second
  - result in around 30min.

HPC Resources

Example (cont.)

- What is implied by a computer that executes 10^{12} operations per second?
  - need to carry out 10^{12} copies from memory to registers in 1 sec.
  - assume data travels at speed of light (3×10^8 m/s)
  - if d is the avg. distance from a register to memory then
    - $d \cdot 10^{12}$ m= 3×10^8 m/s x 1 sec.
    - $d = 3 \times 10^{-4}$m
  - 10^{12} words of memory laid out in square grid of size *s* with CPU at the center --- avg. distance from memory to CPU is *s/2*
    - $s/2 = d = 3 \times 10^{-4}$ m
    - $s = 6 \times 10^{-4}$ m

HPC Resources

Example (cont.)

- A typical row of memory will contain $\sqrt{10^{12}} = 10^6$ words
- Thus
  - we need to fit a single word of memory into a square with side length

$$\frac{6 \times 10^{-4} m}{10^6} = 6 \times 10^{-10} m$$

- This is a measurement on the atomic scale!
  - unless we figure out how to represent a 32 or 64 bit word with a single atom we aren't building this serial computer

HPC Resources

High Performance Computing

- Definition is nebulous
  - resource (processing) intensive computation
  - computing where the need for speed is compelling
    - computing nearer the limit of what is feasible
  - parallel computing (this is too strict)
- In reality, HPC is concerned with varied issues involving:
  - hardware
    - pipelining, instruction sets, multi-processors, inter-connects
  - algorithms
    - efficiency, techniques for concurrency
  - software
    - compilers (optimization/parallelization), libraries

HPC Resources

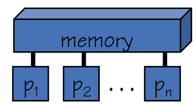
Problems Faced

- Hardware
  - in order to facilitate processors working together they must be able to communicate
  - interconnect hardware is complex
    - sharing memory is easy to say, harder to realize as system scales
    - communication over any kind of network is still painfully slow compared to bus speed --- overhead can be significant
- Software
  - parallel algorithms are actually fairly well understood
  - the realization of algorithms in software is non-trivial
  - compilers
    - automated parallelism is difficult
  - design
    - portability and power are typically at odds with each other

HPC Resources

Building Parallel Machines

- Symmetric Multiprocessors (SMP)
  - shared memory with uniform memory access (UMA)
  - issues of scalability
    - accessing memory is complex with multiple processors
    - crossbar switches can only get so big in practice
- Non-Uniform Memory Access (NUMA)
  - processors can see all of memory, but cannot access it all at the same speed
  - some memory is local (fast), other memory is global (slower)

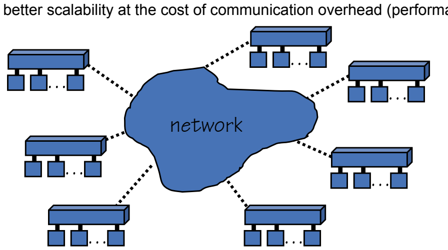


Adapted from "Using MPI: Portable Parallel Programming with the Message Passing Interface (2e), Gropp, Lusk and Skjellum, 1999.

HPC Resources

Building Parallel Machines (cont.)

- Clusters
  - simple (relatively) nodes connected by a network
  - inter-process communication is explicit (e.g. message passing)
  - much better scalability at the cost of communication overhead (performance)



HPC Resources

Adapted from "Using MPI: Portable Parallel Programming with the Message Passing Interface (2e), Gropp, Lusk and Skjellum, 1999.

Sequential Computing

- Traditional model of computing
  - von Neumann architecture
  - one processor + one memory + bus
  - processor executes one instruction at a time
    - where does the notion of pipelining fit in to this?
- Characteristics of ***sequential algorithms***
  - statement of a step-wise solution to a problem using simple instructions and the following three types of operation:
    1. sequence
    2. iteration
    3. conditional branching

HPC Resources

SHARCNET™

Parallel Computing

- The general idea is if one processor is good, many processors will be better
- Parallel programming is not generally trivial
  - tools for automated parallelism are either highly specialized or absent
- Many issues need to be considered, many of which don't have an analog in serial computing
  - data vs. task parallelism
  - problem structure
  - parallel granularity

HPC Resources

SHARCNET™

Data Parallelism

- Data is distributed (blocked) across processors
  - easily automated with compiler hints (e.g. OpenMP, HPF)
  - code otherwise looks fairly sequential
  - benefits from minimal communication overhead
- e.g. Vector Machines
  - a single CPU with a number of subordinate ALUs each with its own memory
  - operate on an array of similar data items during a single operation
    - each cycle: load in parallel from all memories into ALU and perform same instruction on their local data item

HPC Resources

SHARCNET™

Task Parallelism

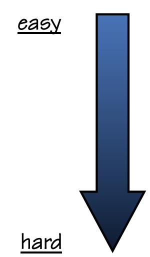
- Work to be done is decomposed across processors
  - e.g. divide and conquer, recursive doubling, etc.
  - each processor responsible for some part of the algorithm
  - communication mechanism is significant
- Must be possible for different processors to be performing different tasks
  - shared memory
    - multi-threading, SMP
  - distributed memory
    - network of workstations, message passing, remote memory operations

HPC Resources

SHARCNET™

Problem Structure

- Structure of the problem dictates the ease with which we can implement parallel solutions



perfect parallelism
- independent calculations

pipeline parallelism
- overlap otherwise sequential work

synchronous parallelism
- parallel work is well synchronized

asynchronous parallelism
-dependent calculations
-parallel work is loosely synchronized

HPC Resources

SHARCNET™

Parallel Granularity

- A measure of the amount of processing performed before communication between processes is required
- Fine grained parallelism
  - constant communication necessary
  - best suited to shared memory environments
- Course grained parallelism
  - significant computation performed before communication is necessary
  - ideally suited to message-passing environments
- Perfect parallelism
  - no communication necessary

HPC Resources

SHARCNET™

Theory of Parallel Computation

- Theoreticians often see themselves as cleaning up after the preliminary empirical work is done
- In computational science the theory can provide significant insight into the nature of a problem
  - design guide; aid to decision making
  - perspective and understanding
  - many practical hints for parallel problem solving
- Amdahl's Law
  - limits to parallel speedup
- P-completeness
  - some problems are inherently sequential

HPC Resources

SHARCNET™

Amdahl's Law

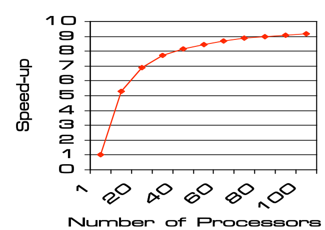
- How well can an application make use of parallel processing resources?
  - Gene Amdahl (IBM 360 architect)
- Consider: every program has some serial component (e.g. set-up)
  - define *speedup* as the ratio of serial to parallel run-time
    - e.g. t_s = 1000s, t_p = 50s
    - speed-up = t_s / t_p = 20
  - note: speed-up = 1 for p = 1
  - consider speed-up as number of processors grows
- Amdahl's Law suggests there is an upper limit on speed-up from parallelism imposed by the serial component of a program

HPC Resources

SHARCNET™

Amdahl's Law (cont.)

- e.g. consider a program that is 90% parallel
  - in the limit, with infinite processors, parallel time would be 0
  - speed-up = 10



% parallel	speed-up (limit)
10	1.11
25	1.33
50	2.00
75	4.00
90	10.00
95	20.00

HPC Resources

SHARCNET™

Amdahl's Law in Perspective

- This suggests diminishing return for a given parallel application
  - there would be no point to massive parallelism
- This isn't necessarily accurate
  - the proportion of serial to parallel work is rarely constant as the problem size scales up
    - e.g. double the problem size
      - amount of serial time may double
      - amount of parallel time may multiply many times over
    - for large problems the serial time may be effectively insignificant, in which case there is the potential for vast pay-offs from massive parallelism

HPC Resources

SHARCNET™

P-completeness

- Parallel analog of NP-completeness
  - study of asymptotic computation (as problem size grows)
  - problems are not equally difficult to solve
  - complexity classes P, NP, NP-complete
- P-complete problems are the hardest problems in P
  - at least as difficult as all problems in P
  - parallel complexity theory asks "does every problem in P have an efficient parallel solution"
  - the answer is most likely no
  - P-complete problems are said to be *inherently sequential*
  - techniques for parallel algorithm design and analysis

HPC Resources