A generic parallel architecture

Interconnection Network

Memory

A general parallel architecture

Where is the memory physically located?

Parallel Programming Models

• Control
  • How is parallelism created?
  • What orderings exist between operations?
  • How do different threads of control synchronize?

• Data
  • What data is private vs. shared?
  • How is logically shared data accessed or communicated?

• Operations
  • What are the atomic (indivisible) operations?

• Cost
  • How do we account for the cost of each of the above?

Simple Example

Consider a sum of an array function: $\sum_{i=0}^{n-1} f(A[i])$

• Parallel Decomposition:
  • Each evaluation and each partial sum is a task.
  • Assign $n/p$ numbers to each of $p$ procs
  • Each computes independent "private" results and partial sum.
  • One (or all) collects the $p$ partial sums and computes the global sum.

Two Classes of Data:

• Logically Shared
  • The original $n$ numbers, the global sum.

• Logically Private
  • The individual function evaluations.
  • What about the individual partial sums?

Programming Model 1: Shared Memory

• Program is a collection of threads of control.
  • Can be created dynamically, mid-execution, in some languages
  • Each thread has a set of private variables, e.g., local stack variables
  • Also a set of shared variables, e.g., static variables, shared common blocks, or global heap.

  • Threads communicate implicitly by writing and reading shared variables.
  • Threads coordinate by synchronizing on shared variables.

Outline

• Overview of parallel machines and programming models
  • Shared memory
  • Shared address space
  • Message passing
  • Data parallel
  • Clusters of SMPs
  • Trends in real machines
• A race condition or data race occurs when:
  - two processors (or two threads) access the same variable, and at least one does a write.
  - The accesses are concurrent (not synchronized) so they could happen simultaneously

• Problem is a race condition on variable s in the program
• A race condition or data race occurs when:

Improved Code for Computing a Sum

- Since addition is associative, it’s OK to rearrange order
- Most computation is on private variables
  - Sharing frequency is also reduced, which might improve speed
  - But there is still a race condition on the update of shared s
- The race condition can be fixed by adding locks (only one thread can hold a lock at a time; others wait for it)

Problems Scaling Shared Memory

- Why not put more processors on (with larger memory)?
  - The memory bus becomes a bottleneck
- Example from a Parallel Spectral Transform Shallow Water Model (PSTSWM) demonstrates the problem
  - Experimental results (and slide) from Pat Worley at ORNL
  - This is an important kernel in atmospheric models
  - 99% of the floating point operations are multiplies or adds, which generally run well on all processors
  - But it does sweeps through memory with little reuse of operands, which exercises the memory system
  - These experiments show serial performance, with one “copy” of the code running independently on varying numbers of processors
    - The best case for shared memory: no sharing
    - But the data doesn’t all fit in the registers/cache

Machine Model 1a: Shared Memory

- Processors all connected to a large shared memory.
  - Typically called Symmetric Multiprocessors (SMPs)
  - Sun, HP, Intel, IBM SMPs (nodes of Millennium, SP)
  - “Local” memory is not (usually) part of the hardware abstraction.
  - Difficulty scaling to large numbers of processors
    - <32 processors typical
  - Advantage: uniform memory access (UMA)
    - Cost: much cheaper to access data in cache than main memory.
Machine Model 1b: Distributed Shared Memory

- Memory is logically shared, but physically distributed
- Any processor can access any address in memory
- Cache lines (or pages) are passed around machine
- SGI Origin is canonical example (+ research machines)
- Scales to 100s
- Limitation is cache coherent protocols – need to keep cached copies of the same address consistent

Programming Model 2: Message Passing

- Program consists of a collection of named processes.
- Usually fixed at program startup time
- Thread of control plus local address space – NO shared data.
- Logically shared data is partitioned over local processes.
- Processes communicate by explicit send/receive pairs
  - Coordination is implicit in every communication event.
  - MPI is the most common example


- First possible solution – what could go wrong?
  - Processor 1
    - $x_{local} = A[1]$  
    - send $x_{local}$, proc2
    - receive $x_{remote}$, proc2
    - $s = x_{local} + x_{remote}$

- If send/receive acts like the telephone system? The post office?
- Second possible solution
  - Processor 2
    - $x_{loadl} = A[2]$  
    - send $x_{loadl}$, proc1
    - receive $x_{remote}$, proc1
    - $s = x_{loadl} + x_{remote}$

Machine Model 2a: Distributed Memory

- Cray T3E, NOW, IBM SP2
- IBM SP-3, Millennium, CITRIS are distributed memory machines, but the nodes are SMPs.
- Each processor has its own memory and cache but cannot directly access another processor’s memory.
- Each “node” has a network interface (NI) for all communication and synchronization.

MPI – the de facto standard

- In 2002 MPI has become the de facto standard for parallel computing
- The software challenge: overcoming the MPI barrier
  - MPI created finally a standard for applications development in the HPC community
  - Standards are always a barrier to further development
  - The MPI standard is a least common denominator building on mid-80s technology
- Programming Model reflects hardware!

"I am not sure how I will program a Petaflops computer, but I am sure that I will need MPI somewhere" – HDS 2001

PC Clusters: Contributions of Beowulf

- An experiment in parallel computing systems
- Established vision of low cost, high end computing
- Demonstrated effectiveness of PC clusters for some (not all) classes of applications
- Provided networking software
- Conveyed findings to broad community (great PR)
- Tutorials and book
- Design standard to rally community!
- Standards beget: books, trained people, software … virtuous cycle

Adapted from Gordon Bell, presentation at Salishan
Open Source Software Model for HPC

- Linus's law, named after Linus Torvalds, the creator of Linux, states that "given enough eyeballs, all bugs are shallow".
  - All source code is "open"
  - Everyone is a tester
  - Everything proceeds a lot faster when everyone works on one code (HPC: nothing gets done if resources are scattered)
- Software is or should be free (Stallman)
- Anyone can support and market the code for any price
- Zero cost software attracts users!
- Prevents community from losing HPC software (CM5, T3E)

Tflop/s Clusters

The following are examples of clusters configured out of separate networks and processor components

- Shell: largest engineering/scientific cluster
- NCSA: 1024 processor cluster (IA64)
- Univ. Heidelberg cluster
- PNNL: announced 8 Tflops (peak) IA64 cluster from HP with Quadrics interconnect
- DTF in US: announced 4 clusters for a total of 13 Teraflops (peak)

But make no mistake: Itanium and McKinley are not a commodity product

Internet Computing - SETI@home

- Running on 500,000 PCs, ~1000 CPU Years per Day
  - 465,821 CPU Years so far
- Sophisticated Data & Signal Processing Analysis
- Distributes Datasets from Arecibo Radio Telescope

Next Step - Allen Telescope Array

Programming Model 2b: Global Addr Space

- Program consists of a collection of named threads.
  - Usually fixed at program startup time
  - Local and shared data, as in shared memory model
  - But, shared data is partitioned over local processes
  - Cost models says remote data is expensive
- Examples: UPC, Titaniunm, Co-Array Fortran
- Global Address Space programming is an intermediate point between message passing and shared memory

Machine Model 2b: Global Address Space

- Cray T3D, T3E, X1, and HP Alphaserver cluster
- Clusters built with Quadrics, Myrinet, or Infiniband
- The network interface supports RDMA (Remote Direct Memory Access)
  - NI can directly access memory without interrupting the CPU
  - One processor can read/write memory with one-sided operations (put/get)
  - Not just a load-store as on a shared memory machine
- Remote data is typically not cached locally

Programming Model 3: Data Parallel

- Single thread of control consisting of parallel operations.
- Parallel operations applied to all (or a defined subset) of a data structure, usually an array
  - Communication is implicit in parallel operators
  - Elegant and easy to understand and reason about
  - Coordination is implicit – statements executed synchronously
  - Similar to Matlab language for array operations
- Drawbacks:
  - Not all problems fit this model
  - Difficult to map onto coarse-grained machines

\[
A = \text{array of all data} \\
IA = f(A) \\
s = \text{sum}(IA)
\]
### Machine Model 3a: SIMD System
- A large number of (usually) small processors.
- A single “control processor” issues each instruction.
- Each processor executes the same instruction.
- Some processors may be turned off on some instructions.
- Machines are very specialized to scientific computing, so they are not popular with vendors (CM2, Maspar).
- Programming model can be implemented in the compiler.
- Mapping n-fold parallelism to p processors, n >> p, but it’s hard (e.g., HPF).

![Diagram of control processor and interconnect](image)

### Model 3b: Vector Machines
- Vector architectures are based on a single processor.
- Multiple functional units.
- All performing the same operation.
- Instructions may specify large amounts of parallelism (e.g., 64-way) but hardware executes only a subset in parallel.
- Historically important.
- Overtaken by MPPs in the 90s.
- Re-emerging in recent years.
- At a large scale in the Earth Simulator (NEC SX6) and Cray X1.
- At a small scale in SIMD media extensions to microprocessors.
- SSE, SSE2 (Intel: Pentium/IA64).
- Altivec (IBM/Motorola/Apple: PowerPC).
- VIS (Sun: Sparc).
- Key idea: Compiler does some of the difficult work of finding parallelism, so the hardware doesn’t have to.

### Vector Processors
- Vector instructions operate on a vector of elements.
- These are specified as operations on vector registers.
- A supercomputer vector register holds ~32-64 elts.
- The number of elements is larger than the amount of parallel hardware, called vector pipes or lanes, say 2-4.
- The hardware performs a full vector operation in
- `#elements-per-vector-register / #pipes`

![Diagram of vector register operations](image)

### Cray X1 Node
- Cray X1 builds a larger “virtual vector”, called an MSP.
- 4 SSPs (each a 2-pipe vector processor) make up an MSP.
- Compiler will (try to) vectorize/parallelize across the MSP.

![Diagram of Cray X1 node](image)

### Earth Simulator Architecture
- Parallel Vector Architecture.
- High speed (vector) processors.
- High memory bandwidth (vector architecture).
- Fast network (new crossbar switch).
- Rearranging commodity parts can’t match this performance.

![Diagram of Earth Simulator](image)
Machine Model 4: Clusters of SMPs

- SMPs are the fastest commodity machine, so use them as a building block for a larger machine with a network
- Common names:
  - CLUMP = Cluster of SMPs
  - Hierarchical machines, constellations
- Most modern machines look like this:
  - Millennium, IBM SPs, (not the t3e)...
- What is an appropriate programming model #4 ???
  - Treat machine as "flat", always use message passing, even within SMP (simple, but ignores an important part of memory hierarchy).
  - Shared memory within one SMP, but message passing outside of an SMP.

Cluster of SMP Approach

- A supercomputer is a stretched high-end server
- Parallel system is built by assembling nodes that are modest size, commercial, SMP servers – just put more of them together

Outline

- Overview of parallel machines and programming models
  - Shared memory
  - Shared address space
  - Message passing
  - Data parallel
  - Clusters of SMPs
- Trends in real machines

TOP500

- Listing of the 500 most powerful Computers in the World
- Yardstick: $R_{max}$ from Linpack
  \[ Ax = b, \text{ dense problem} \]
- Updated twice a year:
  - ISC’xy in Germany, June xy
  - SC’xy in USA, November xy
- All data available from www.top500.org

TOP500 list - Data shown

- Manufacturer: Manufacturer or vendor
- Computer Type: indicated by manufacturer or vendor
- Installation Site: Customer
- Location: Location and country
- Year: Year of installation/last major update
- # Processors: Number of processors
- $R_{max}$: Maximal LINPACK performance achieved
- $R_{peak}$: Theoretical peak performance
- $N_{max}$: Problem size for achieving $R_{max}$
- $N_{1/2}$: Problem size for achieving half of $R_{max}$
- $N_{world}$: Position within the TOP500 ranking

22nd List: The TOP10
Analysis of TOP500 Data

• Annual performance growth about a factor of 1.82
• Two factors contribute almost equally to the annual total performance growth
• Processor number grows per year on the average by a factor of 1.30 and the
• Processor performance grows by 1.40 compared to 1.58 of Moore’s Law


Summary

• Historically, each parallel machine was unique, along with its programming model and programming language.
• It was necessary to throw away software and start over with each new kind of machine.
• Now we distinguish the programming model from the underlying machine, so we can write portably correct codes that run on many machines.
• MPI now the most portable option, but can be tedious.
• Writing portably fast code requires tuning for the architecture.
  • Algorithm design challenge is to make this process easy.
  • Example: picking a blocksize, not rewriting whole algorithm.

Reading Assignment

• Extra reading for today
  • Cray X1
  • Clusters
    http://www.mirror.ac.uk/sites/www.beowulf.org/papers/ICPP95/
  • “Parallel Computer Architecture: A Hardware/Software Approach” by Culler, Singh, and Gupta, Chapter 1.
• Next week: Current high performance architectures
  • Shared memory (for Monday)
    • Or read about the Altix system on the web (www.sgi.com)
  • Blue Gene L (for Wednesday)
    • http://sc-2002.org/paperpdfs/pap.pap207.pdf