Parallel Programming Concepts

Tom Logan

Parallel Background

"Why Bother?"

What is Parallel Programming?

- Simultaneous use of multiple ‘processors’ to solve a single problem
- Use of a group (team) of ‘processes’ to cooperatively solve a problem
- A demanding challenge that requires programmers to “think in parallel”

Parallel Programming Problems

- **Difficult to design, develop, and debug**
- **Still not entirely portable, but getting better all the time**
- **Open to more latent bugs based on order of execution and bulk of code**
  - *"Correct execution of a parallel program once is no guarantee that it will ever execute correctly again."* --Jim McGraw, LLNL
- **Architectures, and, thus, 'tricks of the trade', are constantly in flux**

So Why Do It?

- **Scientific Demand**
  - CERN Large Hadron Collider generates 1 Pbyte of raw data per second. It is filtered to 100Mbyte/sec, but, this leaves 1 Pbyte/year to be saved.
  - Boeing's idea of a supercomputer is one that will do a complex Airliner flight simulation in 6-7 hours
  - ECMWF - European Centre for Medium Range Weather Forecasting would like a 10km weather forecast, but they will need 50 Tflops sustained to do it.

More Science Demands...

- Estimates for genomics:
  - Biology is big business ($50 billion/yr)
  - Modeling a prokaryotics cell (no nucleus) requires tracking ~30 million structures
  - 40 Tflops sustained to simulate an entire organ
  - Ab initio protein folding requires 1 Pflop/s
  - Director of NCSA estimated biology to be an Exaflop level challenge
- Human brain is a 100 Pflop machine - currently are barely past the *mouse brain* level of computing (100 Tflops)

The Demands Quantified

Demand	Result	Computational Increase
Better Remote Sensors	10x Resolution	100 - 1000
Increased resolution for models	10x Resolution	1000 - 10000
Coupled Models	Better Accuracy	2 - 5
Improved Processes	Better Accuracy	2 - 5
Longer Simulations	10-100 times longer	10 - 100
ALL OF THE ABOVE	**100x res, 100x longer, more accurate physics**	**10^{10}**

Aside - On Magnitudes

Magnitude	Number of Words
Mega (M)	Small novel
Giga (G)	Pick-up full of paper or 1 DVD
Tera (T)	1 million books (US library of congress in ~10 Tbytes)
Peda (P)	1-2 Pbytes is all academic research libraries combined
Exa (E?)	Probably less than 5 Ebytes of words spoken in human history

Update for 2010

CENTER	SYSTEM	VENDOR	NCPUS	Max TFLOPS
ORNL	Jaguar	Cray XT5	224162	1,759 (1.76 PF)
NSCS	Nebulae	Dawning/GPU	120640	1,271
LANL	Roadrunner	Bladecenter	122400	1,042
NICS	Kraken	Cray XT5	98,928	831.7
...				
ARSC	Pingo	Cray XT5	3456	26.21

Things change quick!

- **11/2008 Pingo debuted at #109**
- **06/2009 Pingo was #203**
- **06/2010 Pingo is currently #435**

Selected Talks At SC10

- **Petascale data analytics**
- **190 TF Astrophysical N-body Simulation on a cluster of GPUs**
- **Scalable Earthquake Simulation on Petascale Supercomputers**
- **Multiscale Simulation of Cardiovascular flows**
- **Multi-scale Heart Simulation**
- **Petascale Direct Numerical Simulation of Blood Flow on 200K cores**
- **Building Exascale GPU-Based Supercomputers**
- **Exatran: A language for Exascale Computing**
- **Scaling of a Multimillion-Atom MD Simulation**
- **Panasas: The Road to Exascale Storage**

Update for 2013

CENTER	SYSTEM	VENDOR	NCPUS	Max PFLOPS*
NUDT	Tainhe-2	Xeon CPUs	3.12M	33.86
ORNL	Titan	NVIDIA	261632	17.59
LLNL	Sequoia	IBM	1.57M	17.17
RIKEN	K	SPARC64	705024	10.51
ANL	Mira	IBM	786432	8.59

#1 is Chinese
#2, #3, and #5 are DOE machines
#4 is Japanese

Note the change to PFLOPS!

Update for 2015

CENTER	SYSTEM	VENDOR	NCPUS	Max PFLOPS*
NUDT	Tainhe-2	Xeon CPUs	3.12M	33.86
ORNL	Titan	NVIDIA	261632	17.59
LLNL	Sequoia	IBM	1.57M	17.17
RIKEN	K	SPARC64	705024	10.51
ANL	Mira	IBM	786432	8.59

#1 is Chinese
#2, #3, and #5 are DOE machines
#4 is Japanese

I'm really surprised it is the same as 2013!

Ok - So Why Parallel?

- **That's the only way left to go**
  - Clock speeds are approaching that of light
  - Machine level instructions are optimized
  - Pipeline technology has limited scope
  - Vector processing has scalability limits
- **Since they can't build them faster, they're making multi-cpu chips**
  - that go into boards,
    - that go into nodes,
      - that go into clusters,
        - » that go into …

Parallel Architectures

"UMA, NUMA, NORMA?"

Parallel Architecture Models

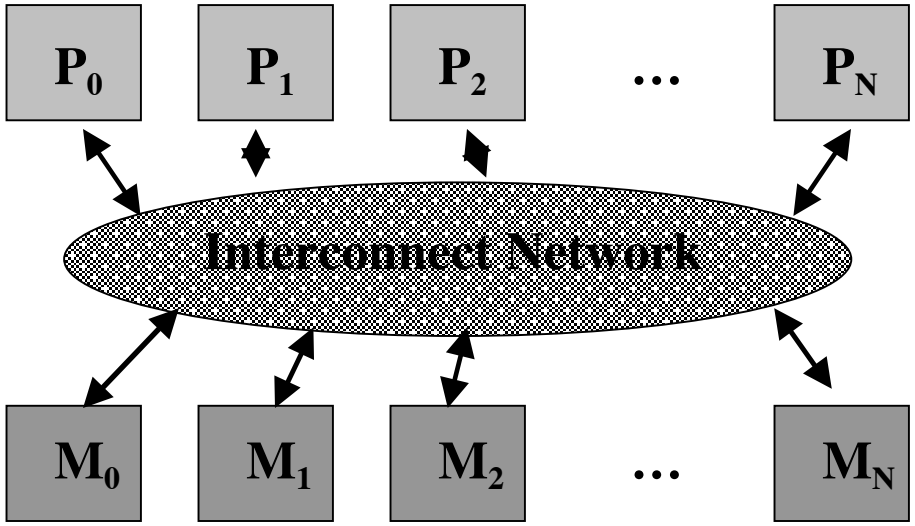
- **Shared Memory Multiprocessor**
  - N processors share a common memory
  - Ideal is UMA (Uniform Memory Access)
  - Reality is NUMA (Non-Uniform Memory Access)
  - To program this machine, use OpenMP
- **Distributed Memory Multicomputer**
  - N computers interconnected by a network
  - NORMA (NO-Remote Memory Access)
  - To program this machine, use MPI
- **Distributed Shared Memory**
  - To program , use OpenMP and/or MPI

Acronyms

- **UMA - Uniform Memory Access**
  - physical memory equidistant from all PEs
- **NUMA - Non-Uniform Memory Access**
  - physical memory is distributed to processors, thus access time varies with the location of the word
- **NORMA - No-Remote-Memory-Access**
  - physical memory is distributed to processors

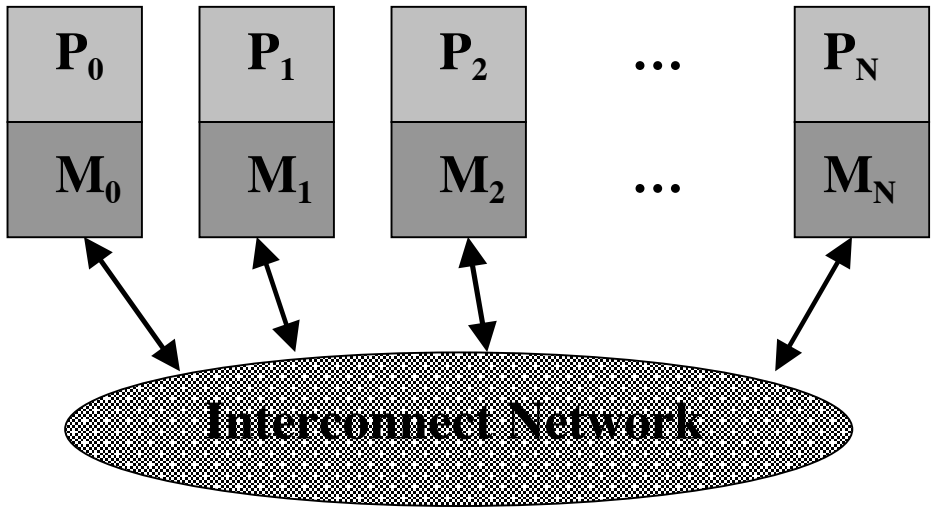
Shared Memory

- **Large globally shared memory space**
- **These are single nodes in modern systems**
- **Examples include SGI O2000, CRAY SV1, IBM Regatta, Cray/NEC SX-6, IBM P690+**



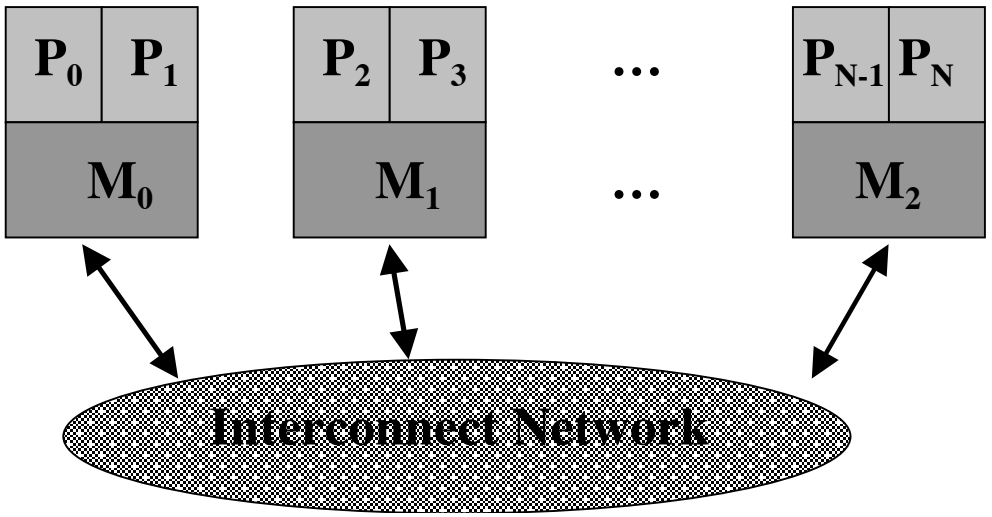
Distributed Memory

- **Each node contains one processor with its own memory.**
- **Nodes are interconnected by message passing network (switches or routers)**
- **Examples include typical linux clusters**



Distributed Shared Memory

- **Modest number of processors share memory within a single node**
- **Distributed memory among nodes**
- **Example is IBM Winterhawk, p655, p690**



Architecture Comparison

- **Memory Latency**
  - Time to access a memory location
  - Local memory access is low latency (SM, DSM)
  - Remote memory access is high latency and may be possible only through message passing (DM, DSM)

Architecture Comparison

- **Memory Contention**
  - Occurs when multiple processors attempt to access a single memory location - particularly for update
  - No memory contention for distributed memory - only a single processor can access local memory
  - Potential for a high level of memory contention for shared memory systems

Example - Cache Effects on Shared Memory

- Sharing/False Sharing - 2 (or more) PEs updating variable(s) on the same cache line
  - PE 1 wants to write to cache line X, but PE 2 owns it
    - request PE 2 to flush cache line X to main memory
    - invalidate all other PEs cache line X
    - read cache line X from main memory
    - update local cache line with value
  - PE 2 wants to write to cache line X, but PE 1 owns it
    - Request …
  - PE 3 wants to write to …
  - And so on, and so on, …

Parallel Algorithms

"Thinking In Parallel"

Algorithm Requirements

- ***Concurrency*** **- ability to work on a problem with separate simultaneous tasks**
- ***Scalability*** **- ability to produce increased speed up as the number of processors is increased**
- ***Locality*** **- ability to minimize remote memory access by accessing local data as frequently as possible**

Parallel Programming Models

- **Shared Memory**
  - Communication via memory constructs
  - SPMD (single program multiple data) is the most common form
  - For our purposes, the majority of shared memory programming will be splitting main loops into the available PEs

(sounds simple enough, right?)

Parallel Programming Models

- **Distributed Mem/Message Passing**
  - Communication is explicit using messages
  - SPMD is most common form of code for a homogeneous environment
  - Adapts well to MPMD (multiple program multiple data) codes in heterogeneous computing systems

Example - Global Reduction

- ***Global Reduction* - parallel operation on an array**
  - commutative binary operations only
    - OK: SUM, PRODUCT(?), MIN, MAX, AVE
    - NOT OK: division, subtraction
  - usually defined in standard parallel programming libraries

Global Sum - Shared Memory

- **Shared Memory**

```
$OMP private(my_sum)
Do j=my_start,my_stop
    my_sum = my_sum+data[j]
End do
$OMP critical
    sum = sum + my_sum
$OMP end critical
```

- **Notes**
- Must declare variable scope
- Calculate global indices (usually automatically done)
- Global memory must be protected in order to avoid race conditions, i.e. EXPLICIT SYNCHRONIZATION is required.
- Each synchronization construct slows code and has potential to reduce performance to serial (or worse)

Global Sum - Message Passing

- **Message Passing**

```
Do j=start,stop
    my_sum = my_sum + data[j]
End do
If (my_pe==0) then
    sum = my_sum
    do j=1,NUM_PE-1
        Recv(j,remote_sum)
        sum = sum + remote_sum
    end do
Else
    Send(0,my_sum)
End if
```

- **Notes**
- Local memory access only
- Message passing can be orders of magnitude slower than memory access
- Synchronization is implicit
- Boss/Worker paradigm leads to load imbalance and, once again, has the potential to reduce performance to serial (or worse)
- How could this be done better?

Parallel Performance Issues

(Yep, they've got issues)

Speedup

- **Definition**

  speedup = unimproved run-time / improved run-time

  (Also called *application speedup*)

- **Example**
  - Program ported to X1 runs in 20 hours
  - Core loop is modified to vectorize, bringing run-time down to 5 hours
  - Therefore, speedup = 20 / 5 = 4

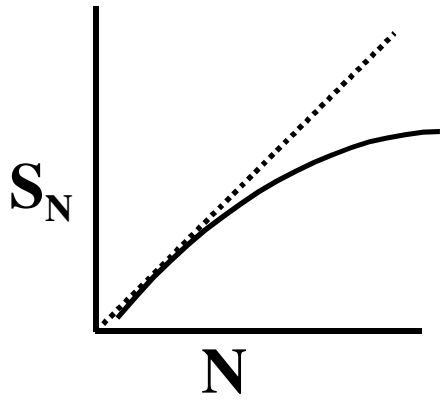
What's the Big Deal?

- **If we have a program that takes time T_1 to run on one processor, why don't we just use p processors to run it in time T_1/p ?**
  1. To use more than one processor requires coordination and communication between them, and
  2. Almost all "real" programs contain portions that cannot utilize multiple processors.

Speed Up Goal

- **Goal: Get N speed up for N processors**
- **Nearly always, Speed Up < N**
- **Fixed problem size has a fixed speed up**

$$S_N = T_1 / T_N$$



Speedup Revisited

- **If we can see a speedup of p for p processors, we call that a *linear speedup***
  - In general, that is the best we can hope for
  - However, there are pathological cases of *superlinear speedup*
- **Often though, speedup tails off so that adding more processors could even cause a slowdown**

Amdahl's Law

- **Speedup is limited by the fraction of the code that does not benefit**
- **E.g., suppose we have a parallel machine. Let**
  - F_s = fraction of code that is inherently serial
  - F_p = fraction of code that is parallelizable
  - So $F_s + F_p = 1$
  - T_1 = time to run on 1 processor
- **Therefore**

  speedup = $T_1 / (F_s T_1 + (F_p T_1 / N) = 1 / (F_s + ((1-F_s) / N))$

Amdahl's Law (cont.)

- **Implication of Amdahl's Law**

  speedup = 1 / (Fs + (Fp / N))

  – Suppose a program runs in 10 hours on one processor and 80% of the code can be executed in parallel on 1000 processors. Then
    - Tp = 0.2 (10) + (0.8 (10) / 1000) = 2.0 + 0.008 = 2.008 and
    - speedup = 1 / (0.2 + 0.8 / 1000) = 1 / 0.2008 = 4.98

- **What if**
  – Fp drops to 50% ?
  – N goes to infinity?

N \ Fs	20%	10%	5%	1%
4	2.5	3.1	3.5	3.9
16	4.0	6.4	9.1	13.9
64	4.7	8.8	15.4	39.3
256	4.9	9.7	18.6	72.1
∞	5.0	10.0	20.0	100.0

Efficiency

- **Ratio of actual speed up to perfect speed up**
- **Measure of a parallel program's ability to use multiple processors**
- **Goal is to achieve 100% efficiency**
- **Reality is that Efficiency < 1**
- **Remember Amdahl's Law? Here's the impact on efficiency:**

	$f = 0.2$	$f = 0.1$	$f = 0.05$	$f = 0.01$
N = 4	63%	78%	88%	98%
N = 16	25%	40%	57%	87%
N = 64	7%	14%	24%	61%
N = 256	2%	4%	7%	28%

Data Access Issues

- **Data Scoping: Private vs. Shared**
  - Refers to the accessibility of data to processes
  - Private data is accessible only to owning process
  - Shared data is accessible to all processes in group
    - ☺**allows work-sharing by multiple processes**
    - ☹**allows memory contention and data races**
  - DM architectures are all private memory
  - DSM & SM architectures allow both private & shared

Data Access Issues

- **Atomic operation: an operation that can be performed in a single step**
- **Few operations are truly atomic**

  Example: j = j + 1

  Step 1: Load contents of memory location j into register

  Step 2: Increment the contents of the register

  Step 3: Store contents of register to memory location j

Data Access Issues

- **Data races occur when multiple PEs attempt simultaneous modification of a shared variable using a non-atomic operation**

Time	PE1	PE2
0		
1	Load r1, j	
2	Incr r1	Load r2, j
3	Store j,r1	Incr r2
4		Store j,r2

j	PE1	PE2
3		
	r1=3	
	r1=4	r2=3
4	j=4	r2=4
4		j=4

More Parallel Terminology

- ***Granularity*** **- measure of the amount of computation involved in a software process. e.g. # of instructions, usually**
  - fine (instruction level),
  - medium (loop level), or
  - coarse (subroutine/program level)
- ***Load Balancing*** **- assignment of tasks to equalize the work load across multiple processors**