Introduction to Parallel Architectures and the World of MultiCores!

R. Govindarajan
SERC, IISc
Introduction

A parallel computer is a “collection of processing elements that communicate and cooperate to solve large problems fast”

- Can run a single parallel application
  - Speedup the application
- Can run multiple independent applications in multiprogramming environment
  - High throughput
- Combination of the two
Parallel Programming Model

- What programmer uses in coding applications
- How processes interact/communicate with each other
- Programming Models:
  - *Shared address space*: Uses common address space, like bulletin board
  - *Message passing*: Explicit messages, like letters
Architectural Models

- **Shared Memory Architecture**
  - Can be used for Shared Address Space or Message passing programs

- **Message Passing Architecture**
  - Typically good for message passing programs only!
Shared Memory Architectures

- Shared, global, address space, hence called *Shared Address Space*

- Any processor can **directly** reference any memory location
  - Interaction occurs implicitly as result of loads and stores

- Memory may be physically distributed among processors
Shared Memory Architecture

Centralized Shared Memory

Distributed Shared Memory

Network
Shared Memory Architecture

- **MultiCore!**

- **Centralized Shared Memory**
- **Distributed Shared Memory**

- **Shared Memory Systems**
  (e.g., Regatta, Altix, p575)
Distributed Memory Architecture

- Message Passing Architecture
  - Memory is private to each node
  - Processes communicate by messages
Distributed Memory Architecture

Cluster
A Look at the MultiCores

- **Multiple Processors (cores) in a single Chip!**
  - Prev. generation had multiple processors on the same motherboard

- **Confusing nomenclature**
  - Core, CPU, Chip, die, socket, ...
MultiCore Structure

Socket0
Core
Core
Core
Core

Socket1
Core
Core
Core
Core

Memory
Using MultiCores to Build Cluster

Node 0
- Memory
- NIC

Node 1
- Memory
- NIC

Node 3
- Memory
- NIC

Node 2
- Memory
- NIC

N/W Switch
Building My Cluster

- How many nodes?
- How many Sockets/node?
- How many Cores/sockets
- How much memory per node?
- What interconnect?
  - Gigabit, Infiniband, ...
Details of MultiCore Structure (Intel Harpertown)

- L1$ L1$ L1$ L1$
- L2-Cache L2-Cache

Memory
Ping-Pong Latency
(One pair of Send-Recv)

- Intranode – diff sockets
- Internode
- Intranode – same die
Ping-Pong Multiple Send-Recvs

MPI Ping pong latency between parallel pairs (Internode)

Time in microseconds
0 5000000 10000000 15000000 20000000 25000000

Bytes
0 1E+07 2E+07 3E+07 4E+07

- 2 pairs in parallel
- 4 pairs in parallel
Memory Bandwidth

- **Memory Bandwidth on a single core**
- **Memory Bandwidth on a pair of cores in the**
  - Same die
  - Same socket
  - Different Socket
- **Memory Bandwidth on four cores**
  - Same socket
  - Different Socket
Memory Bandwidth (Intel Harpertown)

Memory Bandwidth (MB/s)

- Copy
- Scale
- Add
- Triad

1. 1 Socket
2. 2 (same die)
2. 2 (diff die)
4. 4 (Diff sockets, ...)
4. 4 (Single Socket)
Running Multiple Applns. (matmult) on MultiCores

Running Multiple Appln. Instances

<table>
<thead>
<tr>
<th>Execution Time (in Seconds)</th>
<th>1</th>
<th>2 (Diff socket)</th>
<th>2 (Diff die)</th>
<th>2 (Same die)</th>
<th>4 (Diff sockets &amp; diff. die)</th>
<th>4 (Different die)</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execution Time</td>
<td>63.8</td>
<td>63.8</td>
<td>64.5</td>
<td>64.6</td>
<td>64.7</td>
<td>72.2</td>
<td>74.5</td>
</tr>
</tbody>
</table>

© SERC/IISc
**HP Linpack**

- **Performance on 16 processes**

<table>
<thead>
<tr>
<th>Configuration</th>
<th>GFLOPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 nodes x 8 cores</td>
<td>87.2 GFLOPS</td>
</tr>
<tr>
<td>4 nodes x 4 cores</td>
<td>67.16 GFLOPS</td>
</tr>
<tr>
<td>8 nodes x 2 cores</td>
<td>73.0 GFLOPS</td>
</tr>
</tbody>
</table>

- 4 x 4 or 8 x 2 performs worse, even though they have more memory, and can tackle large matrices.
- Network b/w sharing may cause problem for non-local communication!
Intro. to Parallel Programming

- **Shared Memory Address**
  - OpenMP programming model
  - Some synchronization needed between processes

- **Message Passing model**
  - MPI or PVM
  - Explicit message between processes for data and coordination
Parallel Programming

- **Sum the elements of an array** $A[1000000]$
- **Different possibilities:**
  - Accumulate each element using a process
    - Synchronization overheads high – almost sequential!
  - Compute the sum of $1000000/P$ elements in each process, and compute the sum of $P$ elements in process 0
    - Okay if $P$ is not large!
  - Compute the sum of $1000000/P$ elements in each process; compute sum from a pair of processes, and proceed in a similar manner (tree-add - $\log P$ steps!)
Understanding Performance

(a) Serial

(b) Naïve Parallel

(c) Efficient Parallel
Parallelizing a Program

Given a sequential program/algorithim, how to go about producing a parallel version

Four steps in program parallelization

1. Decomposition
   Identifying parallel tasks with large extent of possible concurrent activity
   Domain decomposition

2. Assignment
   Grouping the tasks into processes with best load balancing

3. Orchestration
   Reducing synchronization and communication costs

4. Mapping
   Mapping of processes to processors (if possible)
Steps in Parallel Programming

Partitioning

Decomposition

Sequential computation

Tasks

Assignment

Processes

Orchestration

Parallel program

Mapping

Processors
Assignment

- Specifies how to group tasks together for a process
  - Balance workload, reduce communication and management cost

- Structured approaches usually work well
  - Code inspection (parallel loops) or understanding of application
  - Static versus dynamic assignment

- Both decomposition and assignment are independent of architecture or prog model
  - But cost and complexity of using primitives may affect decisions
Orchestration

- Naming data
- Structuring communication
- Synchronization
- Organizing data structures and scheduling tasks temporally
  - Usually architecture dependent
Orchestration

- **Goals**
  - Reduce cost of communication and synch.
  - Preserve locality of data reference
  - Schedule tasks to satisfy dependences early
  - Reduce overhead of parallelism management

- **Choices depend on Prog. Model., comm. abstraction, efficiency of primitives**

- **Architects should provide appropriate primitives efficiently**
Mapping

- **Two aspects:**
  - Which process runs on which particular processor?
  - Will multiple processes run on the same processor?

- **Space-sharing**
  - Machine divided into subsets, only one app. at a time in a subset
  - Processes can be pinned to processors, or left to OS

- **System allocation**
More on Parallel Programming and Intro to MPI

- Wednesday, March 11 at 5.30 p.m.
  by Dr. Sathish Vadhiyar
Thank you!