ELE 455/555
Computer System Engineering

Section 4 – Parallel Processing

Class 1 – Challenges

Parallel Processing
Introduction

- Motivation
  - Desire to provide more performance (processing)
  - Scaling a single processor is limited
    - Clock speeds
    - Power concerns
    - Cost (yield)
  - A group of multiple smaller processors used in parallel can resolve these concerns and provide additional flexibility
  - Requires effective software to succeed

Parallel Processing

Introduction

- Perspective
  - Run multiple independent programs on a group of processors
    - Independent single-threaded applications

    Task-level parallelism

  - Single program running on a group of processors simultaneously

    Parallel processing program

Parallel Processing
Introduction

- Definitions
  - Multiple discrete processors

    Clusters

  - Multiple processors in a single chip
    - Individual processors are called Cores

      Multi-Core Processor

    - These typically share a common physical memory

      Shared Memory Processor (SMP)

Parallel Processing

Introduction

- Definitions
  - Software that performs in a linear fashion
    - Compiler, Motor Controller

    Sequential Software

  - Software that can handle multiple tasks in parallel
    - OS, Circuit Simulators

    Concurrent Software

Parallel Processing
Introduction

- Hardware / Software Compatibility

  - Need sequential software to run on both serial and parallel hardware
    - The challenge with parallel hardware is to try to utilize the resources

  - Need concurrent software to run on serial and parallel hardware
    - The challenge with serial hardware is performance
    - The challenge with parallel hardware is to utilize all the resources
      - # of parallel processors varies from system to system

		Software	
		Sequential	Concurrent
Hardware	Serial	Matrix Multiply written in MatLab running on an Intel Pentium 4	Windows Vista Operating System running on an Intel Pentium 4
	Parallel	Matrix Multiply written in MATLAB running on an Intel Core i7	Windows Vista Operating System running on an Intel Core i7

Parallel Processing

Challenges

- It is difficult to create parallel processing programs
  - At the processor level (hardware) we have support via:
    - Sub-word parallelism
    - Superscalar hardware
    - Instruction level parallelism
      - Out-of-order, speculation
    - Cache coherence

Parallel Processing
Challenges

- It is difficult to create parallel processing programs
  - We need to create EFFICIENT parallel processing programs
    - If the solution isn't efficient – just use a single processor
    - If a single processor is not an option – still need efficiencies to offset cost, power, complexity
  - Need: Faster, Lower Power

Parallel Processing

Challenges

- It is difficult to create parallel processing programs
  - Many factors limit performance
    - Partitioning
      - Need equal size tasks, otherwise parts of the system are waiting
    - Coordination
      - Synchronizing between processors to share data
    - Communications overhead
      - The actual time to communicate
    - Portions of the program that must be run sequentially

Parallel Processing

Challenges

- It is difficult to create parallel processing programs
  - Example – desire a 90x speedup using 100 processors – what percentage of the program can be sequential

$$T_{new} = T_{parallelizable}/100 + T_{sequential}$$

$$\text{Speedup} = \frac{1}{(1 - F_{parallelizable}) + F_{parallelizable}/100} = 90$$

Solving: $F_{parallelizable} = 0.999$

  - Only 0.1% can be sequential

Parallel Processing

Challenges

- It is difficult to create parallel processing programs

  - Example – calculate 2 sums using 10 and 40 processors
    a) 10 scalars - must be sequential
    b) 10 x 10 matrix - parallelizable

If done entirely sequential
10 scalars = 10t 10x10 matrix = 100t total = 110t

Using 10 processors (only for the matrix)
10 scalars = 10t 10x10 matrix = 100t/10 = 10t total = 20t speed-up = 110t/20t = 5.5

Using 40 processors (only for the matrix)
10 scalars = 10t 10x10 matrix = 100t/40 = 2.5t total = 12.5t speed-up = 110t/12.5t = 8.8

- We quadrupled the number of processors and got less than 2x speed-up

Parallel Processing

Challenges

- It is difficult to create parallel processing programs

  - Example – calculate 2 sums using 10 and 40 processors
    - a) 10 scalars - must be sequential
    - b) 20 x 20 matrix - parallelizable

If done entirely sequential
10 scalars = 10t   20x20 matrix = 400t   total = 410t

Using 10 processors (only for the matrix)
10 scalars = 10t   20x20 matrix = 400t/10 = 40t   total = 50t   speed-up = 410t/50t = 8.2

Using 40 processors (only for the matrix)
10 scalars = 10t   20x20 matrix = 400t/40 = 10t   total = 20t   speed-up = 410t/20t = 20.5

- We quadrupled the number of processors and got a little more than 2x speed-up

Parallel Processing

Challenges

- 2 ways to measure the speedup associated with a parallel processing solution

  - Strong Scaling
    - Keep the problem size fixed while increasing parallelism
    - Fixed customer base using a server farm
      - Faster streaming to customer base

  - Weak Scaling
    - Increase the problem size with increasing parallelism
    - ATM processing central office
      - More customers, not more ATM transactions per customer

Parallel Processing

Challenges

- Scaling Example – strong

  - calculate 2 sums using 10 and 100 processors

    a) 10 scalars - must be sequential

    b) 10 x 10 matrix - parallelizable

If done entirely sequential

10 scalars = 10t 10x10 matrix = 100t total = 110t

Using 10 processors (only for the matrix)

10 scalars = 10t 10x10 matrix = 100t/10 = 10t total = 20t speed-up = 110t/20t = 5.5

Using 100 processors (only for the matrix)

10 scalars = 10t 10x10 matrix = 100t/100 = 1t total = 11t speed-up = 110t/11t = 10

- We achieved 55% of the potential for 10 processors
- We achieved 10% of the potential for 100 processors

Parallel Processing

Challenges

- Scaling Example – weak (100x100 matrix)
  - calculate 2 sums using 10 and 100 processors
    a) 10 scalars - must be sequential
    b) 100 x 100 matrix - parallelizable

If done entirely sequential
10 scalars = 10t 100x100 matrix = 10,000t total = 10,010t

Using 10 processors (only for the matrix)
10 scalars = 10t 100x100 matrix = 10,000t/10 = 1000t total = 1010t speed-up = 10010t/1010t = 9.9

Using 100 processors (only for the matrix)
10 scalars = 10t 100x100 matrix = 10,000t/100 = 100t total = 110t speed-up = 10,010t/110t = 91

- We achieved 99% of the potential for 10 processors
- We achieved 91% of the potential for 100 processors

Parallel Processing

Challenges

- Balance Example
  - Example – calculate 2 sums using 40 processors

    a) 10 scalars - must be sequential

    b) 20 x 20 matrix - parallelizable

  AND – force 1 parallel processor to carry 2x and 5x the normal load

Using 40 processors (only for the matrix) and a balanced load

10 scalars = 10t 20x20 matrix = 400t/40 = 10t total = 20t speed-up = 410t/20t = 20.5

With unbalanced load of 2x, remaining processor sit idle so just look at this case

10 scalars = 10t 20x20 matrix = max[20t/1, 380t/39] = 20t total = 30t speed-up = 410t/30t = 14

With unbalanced load of 5x, remaining processor sit idle so just look at this case

10 scalars = 10t 20x20 matrix = max[50t/1, 350t/39] = 50t total = 60t speed-up = 410t/60t = 7

- The unbalanced load significantly limits the performance improvement

Parallel Processing

Instruction/Data Architectures

- 4 Basic Instruction / Data Configurations

		Data Streams	
		Single	Multiple
Instruction Streams	Single	**SISD**: Intel Pentium 4	**SIMD**: SSE instructions of x86
	Multiple	**MISD**: No examples today	**MIMD**: Intel Xeon e5345

Parallel Processing

Instruction/Data Architectures

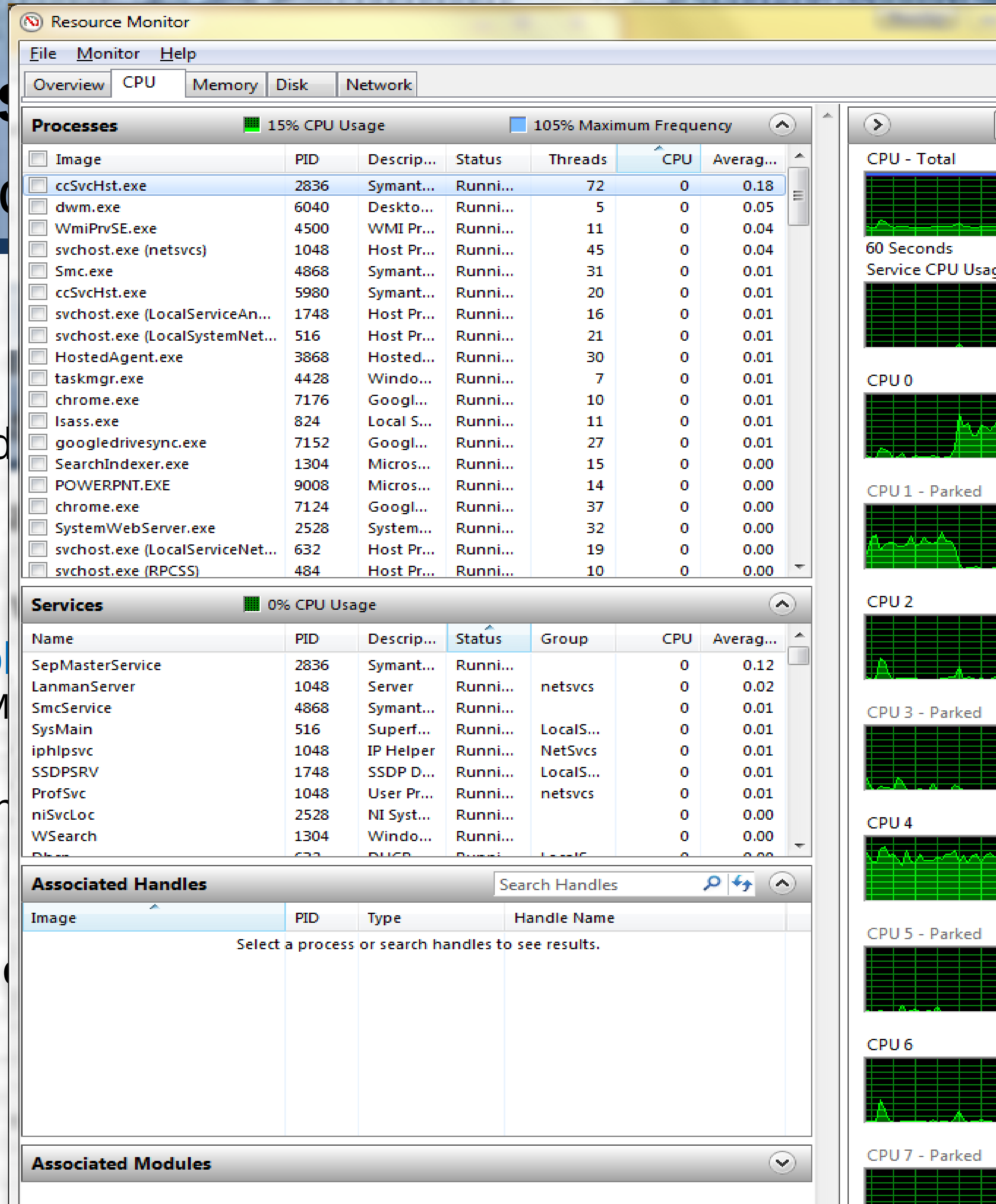
- MIMD
  - While we could spread multiple programs across multiple processors in a MIMD system– the usual case is to spread a single program's instructions across multiple processors

    Single Program Multiple Data (SPMD)
    - Typical application for MIMD
  - Use conditional statements to spread portions of the program to various processors
  - May need a copy of the code for each processor

Parallel Process

Instruction/Data Ar

- MIMD
  - While we could spread processors in a MIMD program's instructions
  - Single Program Multipl
    - Typical application for M
  - Use conditional statem various processors
  - May need a copy of th



Parallel Processing

Instruction/Data Architectures

- SIMD
  - Very much like SISD
    - Execution of a single instruction across multiple processors using vector data
    - One PC, n register sets
    - Program looks just like a sequential program
    - One copy of the code

Parallel Processing

Instruction/Data Architectures

- Vector Architecture
  - Old days – array of processors
  - Now – large register set feeding a pipelined execution unit
    - e.g. 32 vector registers, each with 64, 64bit words
    - Reduces overhead code
      - loops reduced
    - Reduces potential hazards
    - Reduces fetch bandwidth
    - Reduces data bandwidth
    - Data with-in a vector must be independent

Parallel Processing

Instruction/Data Architectures

- Vector Architecture
  - Y = (a x X) + Y
  - Conventional MIPS code

```
      l.d   $f0,a($sp)       ;load scalar a
      addiu r4,$s0,#512      ;upper bound of what to load
loop: l.d   $f2,0($s0)       ;load x(i)
      mul.d $f2,$f2,$f0      ;a × x(i)
      l.d   $f4,0($s1)       ;load y(i)
      add.d $f4,$f4,$f2      ;a × x(i) + y(i)
      s.d   $f4,0($s1)       ;store into y(i)
      addiu $s0,$s0,#8       ;increment index to x
      addiu $s1,$s1,#8       ;increment index to y
      subu  $t0,r4,$s0       ;compute bound
      bne   $t0,$zero,loop ;check if done
```

Parallel Processing

Instruction/Data Architectures

- Vector Architecture
  - Y = (a x X) + Y
  - Vector MIPS code

```
l.d     $f0,a($sp)    ;load scalar a
lv      $v1,0($s0)    ;load vector x
mulvs.d $v2,$v1,$f0   ;vector-scalar multiply
lv      $v3,0($s1)    ;load vector y
addv.d  $v4,$v2,$v3   ;add y to product
sv      $v4,0($s1)    ;store the result
```

vectors: 64 double precision FP #s

Parallel Processing
Instruction/Data Architectures

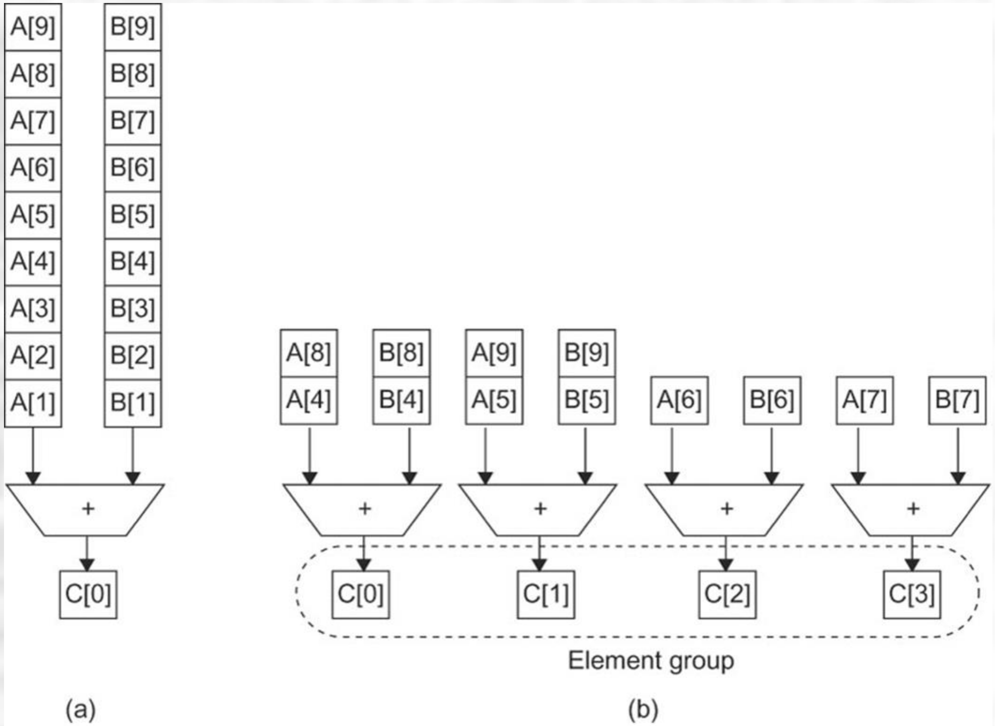
- Vector Architecture

  - Strided Access
    - Read every nth element from memory to place in the vector register
    - Replaces a n iteration loop

  - Gather-scatter
    - Read the vector values from around the memory (gather)
    - Store the results around the memory (scatter)

Parallel Processing
Instruction/Data Architectures

- Vector Architecture
  - Lanes
    - Parallel combinations of vector pipelines



Parallel Processing

Instruction/Data Architectures

- Vector Architecture
  - Lanes
    - Parallel combinations of vector pipelines