Concepts Introduced

- parallel programming
- classifying parallel processors
- multithreading processors
- multicore processors
- GPU processors
- large-scale multiprocessing

Some Definitions

- A multiprocessor is a computer system with two or more conventional processors.
- Task-level parallelism utilizes multiple processors by simultaneously executing largely independent programs.
- A parallel processing application is a single program that simultaneously runs on multiple processors to achieve improved performance.
- A cluster is a set of computers connected over a local area network (LAN) that functions as a single large multiprocessor.
- A multicore processor is a microprocessor containing multiple processors (cores) on a single chip.

Trend toward Multiprocessing

- Slowdown in uniprocessor performance improvements.
  - There are diminishing returns from attempting to exploit more ILP.
  - Clock rates can no longer be increased due to thermal constraints.
- Growing use of servers and data-intensive applications, which have more thread level parallelism.
- Having multiple processors on a single chip allows for much faster communication between processors.

Golden Vision of Multiprocessing

- Multiprocessing is now being more strongly considered as a viable option due to hitting the power wall on increasing clock rates and the difficulty of improving CPI through conventional ILP.
- The golden vision for multiprocessors is to connect multiple existing processors and achieve:
  - Linear speedup based on the number of processors.
  - Improved throughput.
  - Improved reliability.
- The latter two goals are somewhat easier to achieve.

Difficulty of Creating Effective Parallel Programs

- Why are parallel processing programs so much more difficult to develop than sequential programs?
  - Course-grain parallelism is difficult to automatically find and may require new algorithms to be written for the application.
  - The task must be divided into roughly equal-sized portions and these portions must be properly scheduled.
  - Communication overhead must be minimized.
  - Some coordination is required to sychronize the tasks when dealing with shared data.

Amdahl's Law

- Suppose one wishes to achieve a speedup by a factor of 90 with 100 processors. What portion of the original computation can be sequential?

$$speedup_{overall} = \frac{1}{(1 - fraction_{enhanced}) + \frac{fraction_{enhanced}}{speedup_{enhanced}}}$$

$$90 = \frac{1}{(1 - fraction_{enhanced}) + \frac{fraction_{enhanced}}{100}}$$

$$90 * (1 - 0.99 * fraction_{enhanced}) = 1$$

$$90 - 89.1 * fraction_{enhanced} = 1$$

$$89 = 89.1 * fraction_{enhanced}$$

$$0.999 = fraction_{enhanced}$$

- So the sequential portion of the computation can only be about 0.001.

Levels of Parallelism

- levels of parallelism
  - ILP - instruction-level parallelism
  - DLP - data-level parallelism
  - TLP - thread (task) level parallelism
- software models for exploiting TLP
  - parallel processing - tightly coupled set of threads collaborating on a single application
  - request-level parallelism - multiple independent processes that may originate from multiple users

Categorizing Hardware for Parallelism

- traditional categorization
  - SISD - single instruction stream, single data stream (conventional uniprocessor, often exploits ILP)
  - SIMD - single instruction stream, multiple data streams (exploits DLP)
  - MISD - multiple instruction streams, single data stream (not used)
  - MIMD - multiple instruction stream, multiple data streams (exploits TLP)

		Data Streams	
		Single	**Multiple**
Instruction Streams	Single	SISD: Intel Pentium 4	SIMD: SSE instructions of x86
	Multiple	MISD: No examples today	MIMD: Intel Core i7

- The conventional MIMD programming model is sometimes referred to as SPMD (single program, multiple data streams).

SIMD Processors

- SIMD processors operate on vectors of data.
- general types of SIMD processors
  - deeply pipelined vector processors
  - multimedia extensions to conventional processors
  - GPUs

SIMD Advantages

- SIMD architectures can significantly improve performance by exploiting DLP when available in applications.
- SIMD processors are more energy efficient than MIMD as they only need to fetch a single instruction to perform the same operation on multiple data items.
- SIMD allows programmers to continue to think sequentially and sometimes SIMD parallelism can be automatically exploited.

Vector Architectures

- A vector architecture can only be effective on applications that have significant data-level parallelism (DLP).
- A vector architecture includes instruction set extensions to an ISA to support vector operations, which are deeply pipelined.
  - Vector operations are on vector registers, which are fixed-length bank of registers.
  - Data is transferred between a vector register and the memory system.
  - Each vector ALU operation takes vector registers or a vector register and a scalar value as input.
- A vector length register is used to indicate the number of operations that are to be performed on the vector register.
- Vector processors have multiple parallel pipelines called vector lanes, where each lane can produce a result each cycle.

Example of Vector Code

```
/* Source Code */
for (i = 0; i < 64; i++)
   Y[i] = a * X[i] + Y[i];

/* Scalar MIPS Code */
   l.d    $f0,a($sp)
   addiu  $t0,$s0,#512
loop:
   l.d    $f2,0($s0)
   mul.d  $f2,$f2,$f0
   l.d    $f4,0($s1)
   add.d  $f4,$f4,$f2
   s.d    $f4,0($s1)
   addiu  $s0,$s0,#8
   addiu  $s1,$s1,#8
   subu   $t1,$t0,$s0
   bne    $t1,$zero,loop
```

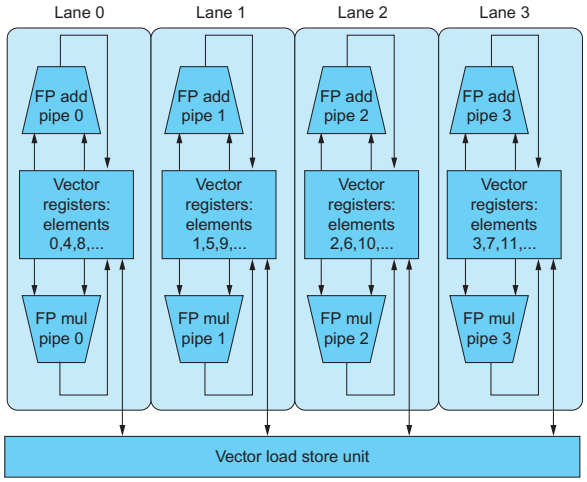
Example of Vector Code (cont.)

- Executes only 6 instructions versus 578 for traditional MIPS.
  - Vector operations work on 64 elements at a time.
  - Loop overhead instructions not present in vector code.
- Only stalls for first element of each vector rather than stalling for each vector element in the traditional MIPS code.

```
/* MIPS Code with Vector Extensions */
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 result in y
```


Vector Unit Containing Four Lanes

- The vector-register storage is divided across the lanes, with each lane holding every four element of each vector register.



Vector Processing vs Scalar Processing

- advantages
  - Greatly reduces the dynamic instruction bandwidth.
  - Hardware only needs to check for data hazards between two vector instructions once and not for every vector element.
  - Vector memory operations work well with interleaved memory banks if the vector elements in memory are adjacent.
  - Generally execution time is reduced due to:
    - eliminating loop overhead
    - stalls only occurring on the first vector element rather than stalling on each vector element
    - performing vector operations in parallel
- disadvantages
  - All iterations of the loop to be vectorized have to be independent.
  - Requires a lot of state to store if a context switch.
  - Vector loads and stores are complex when dealing with a memory hierarchy due to cache misses.

SIMD Extensions to GP Processors

- Many GP processors now have SIMD extensions to support simultaneous operations on applications, including for multimedia.
- SIMD extensions are a subset of vector operations, which operate on a small fixed number of operands (no vector length register).
- SIMD operations work on shorter vectors and all operations are typically performed in parallel, as opposed to being pipelined.
- Examples include the x86 SIMD extensions.
  - MultiMedia eXtensions (MMX) in 1996 - used FP registers
  - Streaming Simd Extensions (SSE) 1999 - separate 128-bit registers
  - Advanced Vector eXtensions (AVX) 2010 - separate 256-bit registers

SIMD Extensions Easier to Implement

- Can be added with little cost. For instance, an option to a conventional integer adder can be to not perform carries across specific partitions (e.g. parallel 8-bit additions).
- Require little state as compared to vector architectures, which means it is easier to implement context switches.
- Need much less memory bandwidth.
- Operands in memory for SIMD extensions on many architectures have to be aligned within a L1 DC line, which means one instruction only needs one access to the memory system. However, due to this SIMD alignment problem, it is much harder for compilers to automatically exploit these SIMD extensions.

SIMD Example

- X and Y have to be aligned on a 32 byte boundary.

```
/* sparse array loop */
for (i = 0; i < 64; i++)
   Y[i] = a * X[i] + Y[i];
```

```
/* MIPS with SIMD extensions */
   l.d     $f0,a($sp)
   mov     $f1,$f0
   mov     $f2,$f0
   mov     $f3,$f0
   addiu   $t0,$s0,#512
Loop:
   l.4d    $f4,0($s0)
   mul.4d  $f4,$f4,$f0
   l.4d    $f8,0($s1)
   add.4d  $f8,$f8,$f4
   s.4d    $f8,0($s0)
   addiu   $s0,$s0,#32
   addiu   $s1,$s1,#32
   bne     $s0,$t0,Loop
```

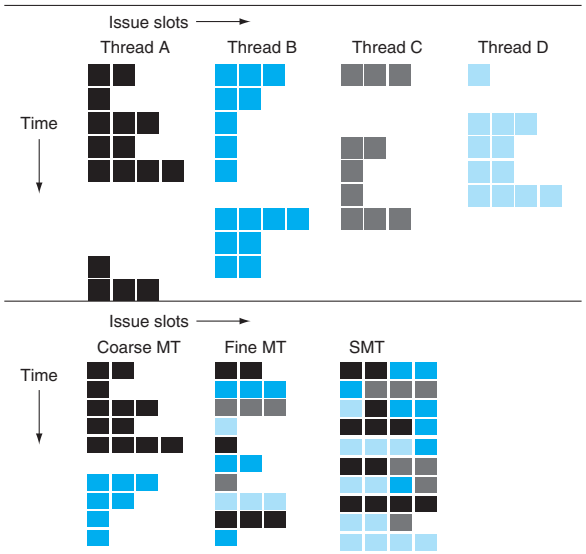

Multithreading

- Multithreading allows multiple threads to share the functional units of a single processor in an overlapping fashion.
- A processor must save the state of each process (register file, PC, condition codes) on a context switch, which can require hundreds to thousands of cycles.
- Thread switches must be more efficient than process switches, which is accomplished by duplicating the state (register file, PC, condition codes) and sharing the virtual address space of the process.
- Multithreading may result in more cache and TLB conflicts.

Types of Multithreading

- *coarse-grained multithreading*
  - Switches threads on costly stalls (e.g. last-level cache misses).
  - Pipeline must be drained on each thread switch.
- *fine-grained multithreading*
  - Switches between threads on each clock cycle in a round-robin fashion.
  - Advantage is that it can hide thoughput losses on short stalls by performing useful work in other threads.
  - Disadvantage is that it can slow down the execution of an individual thread.
- *simultaneous multithreading*
  - Allows multiple threads to simultaneously run by exploiting the resources of a multiple-issue, dynamically scheduled processor.
  - Potentially provides best performance of the three approaches by more effectively utilizing the issue slots.
  - Instruction issue is more complex.

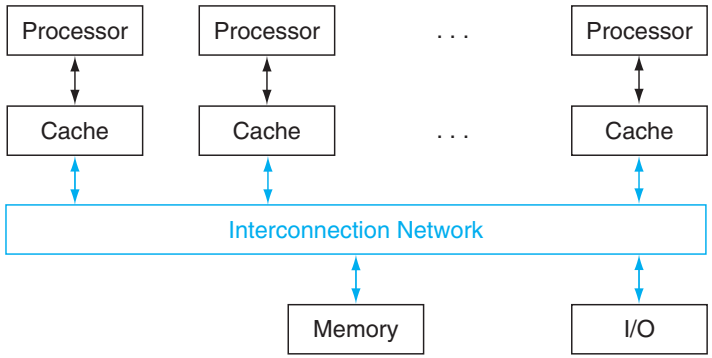
Multithreading Approaches with Four Threads



Shared Memory Multiprocessors

- Shared memory multiprocessors (SMPs) is where the application has a single physical address space across all the processors and communicate by updating variables through a shared address space.
- A *symmetric (or centralized) shared memory* multiprocessor (SMP) means that they share a single centralized memory.
  - Used for multiprocessors with a small number of processors.
  - They are sometimes called *uniform memory access* (UMA) multiprocessors.
- A *distributed shared memory* (DSM) multiprocessor means that the memory is distributed among the processors.
  - Easier to scale to more processors and improves latency/bandwidth for accesses to local memory.
  - Requires more effort to appropriately allocate data in the distributed memories.
  - They are sometimes called *nonuniform memory access* (NUMA) multiprocessors.

Classic Organization of a SMP



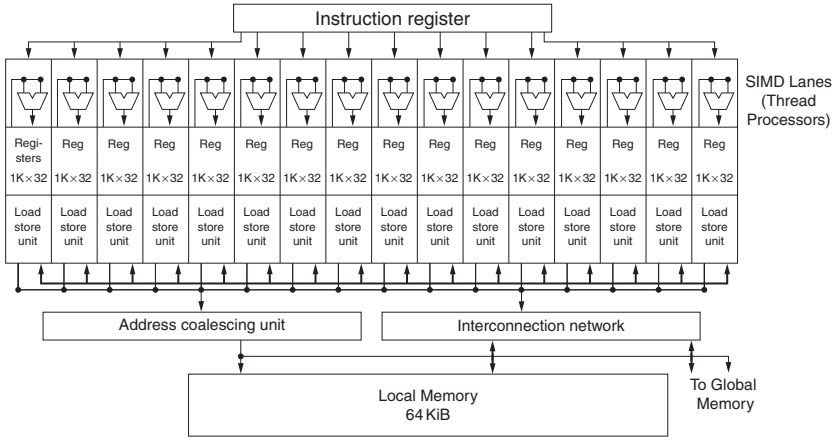
Graphics Processing Units (GPUs)

- GPUs were first developed as graphics accelerators, but now are also starting to be used in mainstream computing (GPGPUs).
- GPUs support many types of parallelism (ILP, SIMD, multithreading, MIMD), but work best with DLP applications.
- Some GPUs have their own programming language.
  - CUDA is offered by NVIDIA.
  - OpenCL is vendor-independent for multiple platforms.
- Relies on hardware multithreading to hide the latency to memory.

NVIDIA GPU Overview

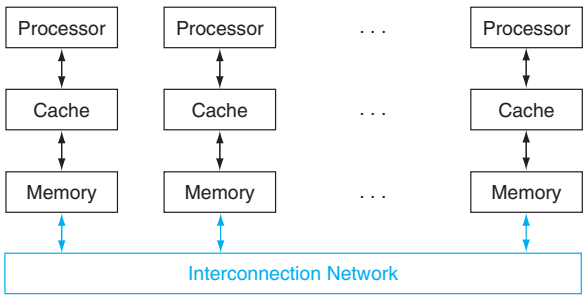
- heterogeneous execution model
  - CPU is the host.
  - GPU is the device.
- CUDA is a C-like programming language to exploit GPU features.
- The programming model is called single instruction, multiple thread (SIMT).
  - Like a traditional thread, but contains exclusively SIMD instructions.
  - SIMD threads have their own PC and run on a multithreaded SIMD processor.

Datapath of a Multithreaded SIMD Processor in a GPU



Message Passing Multiprocessors

- Message passing multiprocessors each have their own private physical address space.
- These multiprocessors communicate by sending and receiving messages.
- Most message-passing multiprocessors use a cluster, where each node has its own OS and they communicate via a local area network (LAN).



Warehouse-Scale Computers

- A *cluster* is a collection of desktop computers or servers connected together by a local area network to act as a single larger computer.
- A *warehouse-scale computer* (WSC) is comprised of tens of thousands of servers.
- The cost may be on the order of $150M for the building, electrical and cooling infrastructure, the servers, and the networking equipment that houses 50,000 to 100,000 servers.
- A WSC can be used to provide internet services.
  - search - Google
  - social networking - Facebook
  - video sharing - YouTube
  - online sales - Amazon
  - cloud computing services - Rackspace
  - and many more applications

Important Design Factors for WSCs

- WSC goals and requirements in common with servers.
  - cost-performance - work done per dollar
  - energy efficiency - work done per joule
  - dependability via redundancy
  - network I/O
  - interactive and batch processing workloads
- WSC aspects that are distinct from servers.
  - Ample parallelism is always available in a WSC.
  - Operational costs represent a greater fraction of the cost of a WSC.
  - Customization is easier for the scale of an WSC.

Fallacies and Pitfalls

- Fallacy: Peak performance tracks observed performance.
- Pitfall: Not developing the software to take advantage of, or optimize for, a multiprocessor architecture.
- Fallacy: You can get good vector performance without providing memory bandwidth.