Introduction to Parallel Programming

Sequential Architecture

- Sequential computers are based on the model presented by John von Neumann
- Performance of the model is limited by:
  - Speed of information exchange between memory and CPU
  - Execution rate of instructions
- Performance can be improved by improving these aspects

Memory Banks & Cache

- The *speed* of **information exchange** between memory and CPU can be improved
  - by using **memory interleaving**
    - (= simultaneous memory access by having **several** memory banks)
  - by using very fast memory (**cache**)

Pipeline

- Execution rate of the instructions can be improved by **overlapping** the execution of several instructions
  - ***instruction pipelining*** means that instructions are executed and fetched from the memory at the same time
  - ***execution pipelining*** means that different instructions are in different functional units (multipliers, adders, ...)

Instruction Pipelines

- Execution of instructions has 4 phases:
  - Fetch
  - Decode
  - Execute
  - Write-back
- Usually these phases are executed as in industrial assembly line

  => overlapping of execution

Note

- It takes some time to fill the pipeline but as soon as the pipeline is filled the results come out in every cycle
- Branches affect the execution as new instruction need to be fetched

  Slows down the execution

  => Branch prediction

About Sequential Architectures

- **Pipelining**, memory interleaving and **caching** are commonly used in sequential computers
- Some restrictions
  - Cache speed is limited by technology
  - Memory interlaving and pipelining are useful only in **some cases**
- Alternate way is to use real ***parallel*** architectures

Parallel Architectures

- Parallel architectures can be **classified** in several ways:
  - **Flynn's taxonomy**
  - control mechanism
  - address-space organization
  - interconnection network
  - **granularity** of processors
- Note! ovelapping categories

Flynn's taxonomy

- The most famous classification according to the data and instruction streams:
  - **SISD** (Single Instruction Single Data)
    - Sequential architecture
  - **MISD** (Multiple Instruction Single Data)
    - Several processors execute different instructions to the same data => not too practical
  - **SIMD** (Single Instruction Multiple Data)
    - Several processors execute same instruction to several different data
  - **MIMD** (Multiple Instruction Multiple Data)
    - Several processors execute instructions independently to their own data

Control Mechanism

- Parallel architectures consist of several processing elements PE
- Processing elements may operate
  - under centralized control (=SISD)
  - independently (=MIMD)

SIMD & MIMD

- **SIMD:** Control unit dispatches instructions to processing elements
  - synchronous execution
  - PE is on/off
  - CM-2, Illiac IV, MP-1, MP-2
- **MIMD:** Each processing element may execute its own program
  - iPSC, Symmetry, nCube 2, CM-5

Comparison

- SIMD requires less hardware because of *single control unit*
- MIMD processors are more complex because of *separate control units*
- SIMD requires less memory because only one copy of the program is used
- SIMD suits for *data-parallel* programs

Comparison

- SIMD may execute only **one** instruction in each **clock cycle**
- MIMD is suitable for programs limited by condition statements
- SIMD offers *automatic synchronization*
- MIMD processors are **common** whereas SIMD are special design

Address-space Organization

- Solving a problem in parallel architectures requires **communication** between processors
- Communication can be achieved by using
  - **shared memory**
  - **message passing**

Message Passing

- Multicomputer
- Each processor has its own memory
- Processors are connected through a ***message passing*** interconnection network
- Processors communicate by sending messages to each other
- Example: CM-5, iPSC

Shared Memory

- Multiprocessor
- Shared memory architecture uses shared memory for the communication by changing variables
- Different variations
  - **UMA**, uniform memory access
  - **NUMA**, non-uniform memory access
  - **COMA**, cache-only memory access

Uniform Memory Access

- First approach is to offer equal access to shared memory
- Interconnection network capacity problem
  - all processors may need to have something from the memory at the same time
  - memory access through the network may be slow (because of the structure)

Non-uniform Memory Access

- Avoid unnecessary memory references by using local memory
  - Program
  - Non-shared data structures
- Local memory concept can be extended by removing global (shared) memory
- Memory references are mapped by the hardware

Cache-only Memory Access

- NUMA architecture is similar to message passing architecture
  - NUMA provides hardware support
  - Message passing architecture requires explicit message passing
- In some cases the processor contains only cache memory and no local memory

  (= cache only memory access)

Cache Coherence

- Cache can be added to processors to speed up the memory references
- Cache improves the performance as it does in sequential architecture
  - but the parallel architecture causes troubles if processors modify variables in their own cache
    - Memory value and cache value may ***differ***

Cache Coherence Problem

- Write-through (WT) and write-back (WB) caches
  - **WT:** memory follows the cache
  - **WB**: memory is not updated until replacement
- Cache coherence problem may appear if
  - Modification of shared data
  - Process migration
  - I/O operations bypassing caches

Cache Coherence

- Cache coherence can be achieved by using several approaches:
  - snoopy coherence protocols (snoopy bus)
    - **write invalidate**
      - Invalidates all other copies
    - **write update**
      - Broadcasts the newly cached copy

Interconnection Network

- Both shared memory and message passing architectures can use
  - **static** interconnection network
  - **dynamic** interconnection network
- **Static networks** usually in message passing computers
- **Dynamic networks** usually in shared memory computers

Granularity of Processors

- Parallel architecture can be implemented by using
  - small number of powerful processors
  - large amount of less powerful processors
- Classification into
  - coarse-grain (only few processors)
  - medium-grain
  - fine-grain (a lot of processors)
- Examples: Cray Y-MP, CM-5, CM-2

Granularity

- Price and availability of processors !
  - powerful processors cost a lot and therefore parallel computers tend to have only limited number of them
- Different applications suit for different architectures
  - limited parallel properties => coarse-grain
  - lot of concurrency => fine-grain

Granularity

- Ratio of the time required for a basic communication to the time required for the basic computation
  - **Small ratio** => suitable for communication intensive algorithms (= fine grain)
  - **Large ratio** => suitable for computation intensive algorithms (= coarse grain)

Another approach

- Similar way is to *classify* parallel computers according to the following terms:
  - **Homogeneity** (SIMD, MIMD, SPMD)
  - **Synchrony** (synchronous, asynhronous, loosely synchornous, BSP)
  - **Interaction** mechanism (shared, messages)
  - **Address** space (UMA, NUMA, NORMA, COMA)
  - **Memory** model (EREW, CREW, ...)

Terms

- **Homogeneity**
  - How alike processors are
- **Synchrony**
  - How tight synchronization is used
- **Interaction mechanism**
  - How processes interact
- **Address space**
  - memory location accessible by the process
- **Memory model**
  - How machine model handles shared memory conflicts

Physical Models

- **Large scale** computer systems can generally be classified into the following practical models:
  - **SIMD**
  - **PVP**, parallel vector processor
  - **SMP**, symmetric multiprocessor
  - **MPP**, massively parallel processor
  - **COW**, cluster of workstations
  - **DSM**, Distributed Shared Memory

PVP

- These systems contain a small number of powerful custom made vector processors
- Each processor is capable of 1 Gflops
- Custom designed crossbar swith network
- Shared memory
- Cray C-90, Cray T-90, NEC SX-4

SMP

- Uses common microprocessors
- Bus / crossbar switch
- Symmetric i.e. each processor has equal access to memory, I/O, ...
- Number of processors is limited by the centralized memory, interconnection (scaling)
- Used for database applications, data warehouses
- IBM R50, SGI Power Challenge, DEC Alpha server 8400

MPP

- **Very large-scale** computer system:
  - common microprocessors
  - physically distributed memory
  - high communication bandwith and low latency interconnection
  - scales well (up to 1000s of processors)
  - asynchronous MIMD machine with message passing
  - program consist of multiple processes (private address space)
- Tightly coupled

COW

- Low-cost variation of MPP
  - each node is a complete workstation (PC/SMP)
  - low-cost network (Ethernet, FDDI, Fiber-channel, ATM, ...)
  - loosely coupled (separate I/O bus)
  - loacl disk
  - complete operating system
- Digital TruCluster, IBM SP2, Berkley NOW

DSM

- Uses cache directory (supports distributed coherent caches) or special hardware and software extensions
- Stanford DASH, Cray T3D
- Difference to SMP is physical distribution of the memory (hardware and software makes on illusion of shared memory)
- Sofware inplementation (TreadMarks)

Comments

- Boundary between MPPs and COW is becoming fuzzy
  - IBM SP2 is MPP but has a cluster architecture with high-performance **switch**
  - **Clustering** is becoming a trend for scalable parallel computers

Clustering

- *Cluster is a collection of complete computers (nodes) that are physically interconnected by a high-performance network or a local area network*
- **Characteristics:**
  - Each computer is an SMP, workstation or PC
  - Nodes work together as a single integrated computing resource
  - Each node can be used separately if necessary

Architecturel concepts

- Cluster nodes
- Internode connection
  - Ethernet, FDDI, ATM, Fiber-channel
  - Standard protocols
- Single system image
  - single resource vs. distributed system
- Enhanced availability
  - cost-effective way to enhance availability (% of time system is available for the user)
- Better performance
  - superserver, minimize time for a job

Architectural comparison

- Clusters, SMP, MPP, distributed systems are overlapping
  - node complexity – hardware and software capability
  - Single system image – relative concept

Benefits and difficulties

- Usability
  - Single nodes are traditional platform

    $\Rightarrow$develop and run application as used
  - More difficult to program than message passing MPP
- Availability
  - Percentage of time system is available for productive use
  - Use of common components (redundancy)
  - Processors, memory, disk array, operating system

Benefits and difficulties

- Scalable performance
  - add performance by increasing nodes
  - processors, memory, disks scale well
- Performance/cost ratio
  - clusters are cost-effective (cmp. PVP, MPP)
  - made of common components (Moore’s law)

Abstract models

- Few abstract models have been developed for the parallel computing
  - PRAM, Parallel random access machine
  - BSP, Bulk synchronous parallel model
  - Phase parallel model
- Design and analysis of algorithms without worrying about the details of physical machines

PRAM

- Ideal architecture
- Characteristics:
  - p processors + unlimited memory
  - MIMD
  - Fine grain
  - Tightly synchronous
  - Zero overhead for synchronization
  - Shared variables (no communication overhead)

PRAM

- Simple and clean approach
  - many theoretical parallel algorithms are specified with PRAM
  - widely used for analysing algorithms
- Zero communication time is unrealistic
- Overheads may affect in real life
- Because of unrealistic assumptions PRAM model has not been used as a machine model for real life parallel computers

Memory models

- Memory references that happen at the same time can be solved as follows:
  - EREW (Exclusive Read Exclusive Write)
    - No memory operations at the same time
    - Weakest model
  - CREW (Concurrent Read Exclusive Write)
    - Several reads allowed but only one write
    - Concurrent writes happen sequentially
  - ERCW (Exclusive Read Concurrent Write)
    - Several writes but only one read
  - CRCW (Concurrent Read Concurrent Write)
    - Several reads and writes at the same time allowed

Concurrent write

- Concurrent write can be solved in CRCW and ERCW models as follows:
  - Common value
    - Concurrent write can be performed if the same value
  - Arbitrary value
    - One arbitrary processor succeeds
  - Priority
    - Processor with the highest priority succeeds
  - Function
    - Use some function of the values e.g. sum

BSP

- Bulk synchronous parallel model
- Overcome shortcomings of PRAM model
  - p processor memory pairs (nodes)
  - Interconnection network
  - p processes (one / node)
  - Time steps & supersteps
    - Computation operations w time steps
    - Communication gh time steps
    - Barrier synchronization l time steps

BSP

- BSP is an MIMD type computer
- Loose synchrony at superstep level (compare tight instruction level synchrony in PRAM)
- Within superstep each process execute instructions asynchronously
- Communication may use shared memory or message passing
- Point-to-point communication

BSP

- BSP is more realistic than PRAM because it consideres overheads
- Exeution time of superstep is determined by:
  - load imbalance: w is max time spent on computation operations
  - synchronization overhead: lower bound for the communication network latency
  - communication overhead: gh time steps, platform dependent part g * h relation
  - time for superstep is w + gh + l

Phase Parallel Model

- Combines both PARM and BSP
  - Parallel program is executed as sequence of phases (superstep)
    - parallelism phase
    - computation phase
    - interaction phase
  - Different computation phases may execute different workloads at different speeds (w, t_f)
  - Different interactions may take different times

Sources

- Lappeenranta Univ of Technology
- Labra