Parallel Programming Overview

Pham Quang Dung

Hanoi, 2012

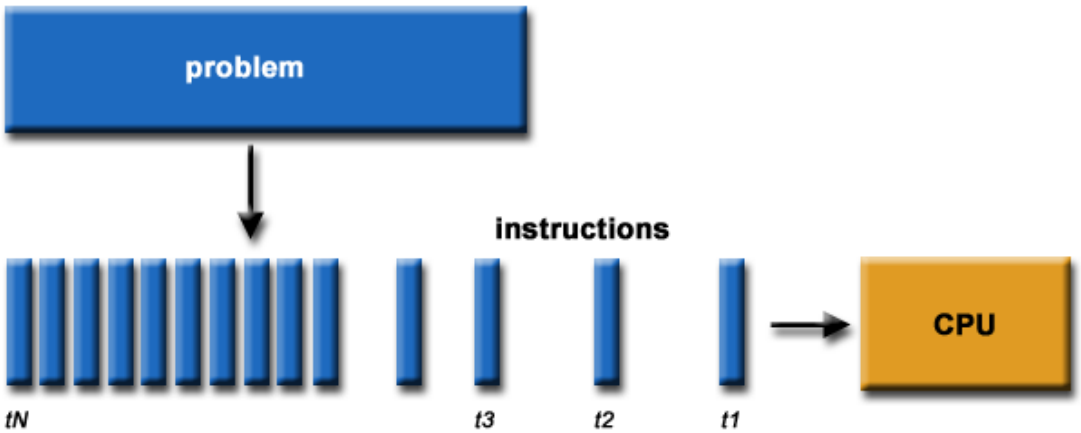
Objective and Plan

- Objective of the course
  - Learn mechanisms of for implementing parallel programs on a given plateform (Linux/Ubuntu)
  - Students will be able to design and implement parallel programs for solving some computational problems
- Plan of the course
  - Lectures
    - Overview
    - POSIX threads
    - OpenMP
    - Message Passing (MPI)
  - Projects
    - Groups of 2-3 students, each of which solve a problem
    - Presentation and demo
- Environment : Ubuntu
- Online reference : https ://computing.llnl.gov/tutorials/parallel_comp/

Outline

Overview

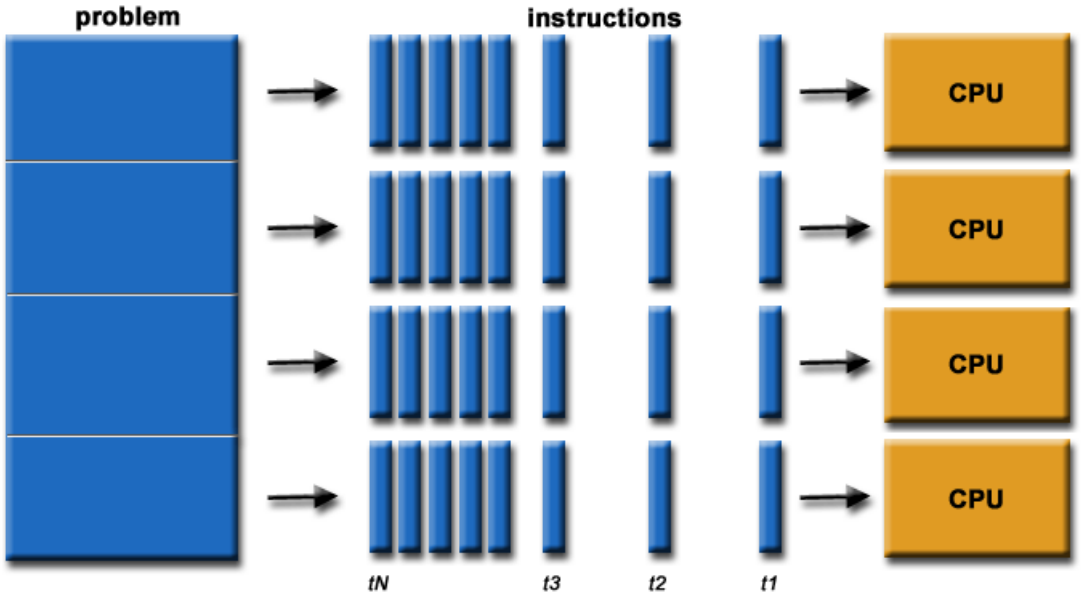
- Traditionally, programs have been written for serial computation : program = sequence of instructions
- Run on a single computer with a single processor
- Only one instruction may execute at any moment of time



https ://computing.llnl.gov/tutorials/parallel_comp/

Overview

- Parallel Computing : use multiple computer resources simultaneously
- A problem is broken into a number of parts that can be executed in parallel
- Run on mulitple processors concurently



https ://computing.llnl.gov/tutorials/parallel_comp/

Overview

- Computer resources :
  - A single computer with multiple processors
  - Mulitple computers connected to each other by a network
  - Combination of both

Concept and Terminology

- Node
  - Comprised multiple CPUs/processors/cores
  - Networked together to comprise a supercomputer
- CPU (processor)/core : CPU (or processor) is divided into multiple cores (unique execution unit)
- Task
  - A program or a set of instructions that is executed by a processor
  - A parallel program consists of multiple taks running on multiple processors
- Shared memory : all processors have direct (bus based) access to common physical memory
- Symmetric multi-processor (SMP) : hardware architecture where multiple processors share a single address space and access to all resources

Concept and Terminology

- Distributed memory :
  - Hardware : Network based memory access for physical memory
  - Programming model : tasks must use communication to access memory on other machines where other taks are executing
- Communication : Paralle tasks need to exchange data (shared memory bus or over a network)
- Synchronization :
  - A point where one task may not proceed further until other tasks reach the same point
  - Invole waiting, therefore the paralle application's wall clock may increase
- Observered speedup : $\frac{\text{wall-clock time of serial execution}}{\text{wall-clock time of parallel execution}}$

Concept and Terminology

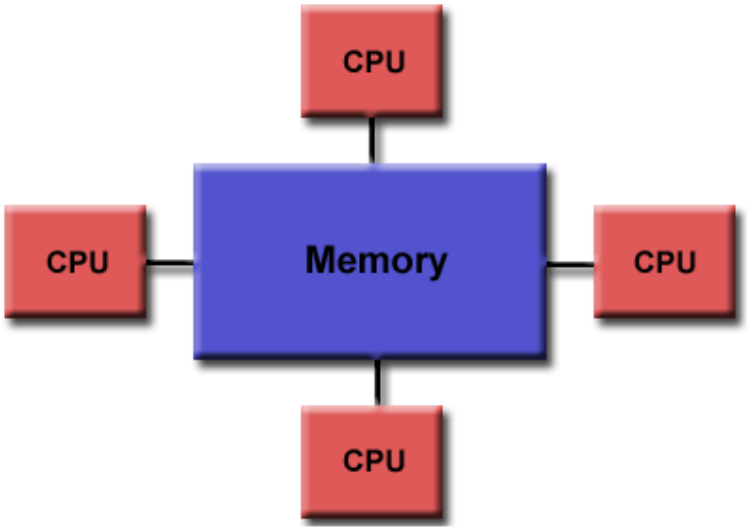
- Parallel overhead :
  - Task start-up time
  - Synchronizations
  - Data communications
  - Software overhead imposed by parallel compilers, libraries, tools, operating system, etc.
  - Task termination time
- Scalability
  - Refers to a parallel system's (hardware and/or software) ability to increas speedup with addition of more processors

Outline

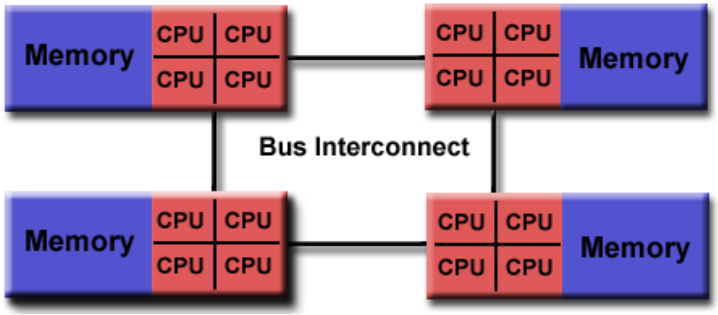
Parallel computer memory architectures

- Shared memory
  - All processors can access to all memory as global address space
  - Multi processors operate independently
  - Changes in memory effected by one processor are visible to all other processors
  - Two classes : UMA and NUMA (detailed later)
- Distributed memory
  - Base on communication network
  - Processors have their own local memory, no global address space across all processors
  - Processors operate independently, changes of a local memory of a processor do not effect on the memory of other processors
  - Communication is based on data transfer
  - Network fabric used for data transfer is usually Ethernet

Shared memory classification



a. Uniform Memory Access (UMA)



b. Non-Uniform Memory Access (NUMA)

source : https ://computing.llnl.gov/tutorials/parallel_comp/

Shared memory classification : UMA

- Common today : SMP (Symetric MultiProcessor) machines
- Identical processors
- Equal access and access time to memory
- Sometimes called CC-UMA (Caches Coherent UMA)
  - All processors know about an update to a location in shared memory made by one processor
  - Accomplish at the hardware level

Shared memory classification : NUMA

- Physically link two or more SMPs
- One SMP can access directly to memory of another SMP
- Processors do not have idetical access and access time to all memory
- Memory access across link is slower
- If cache coherency is maintained, we call CC-NUMA

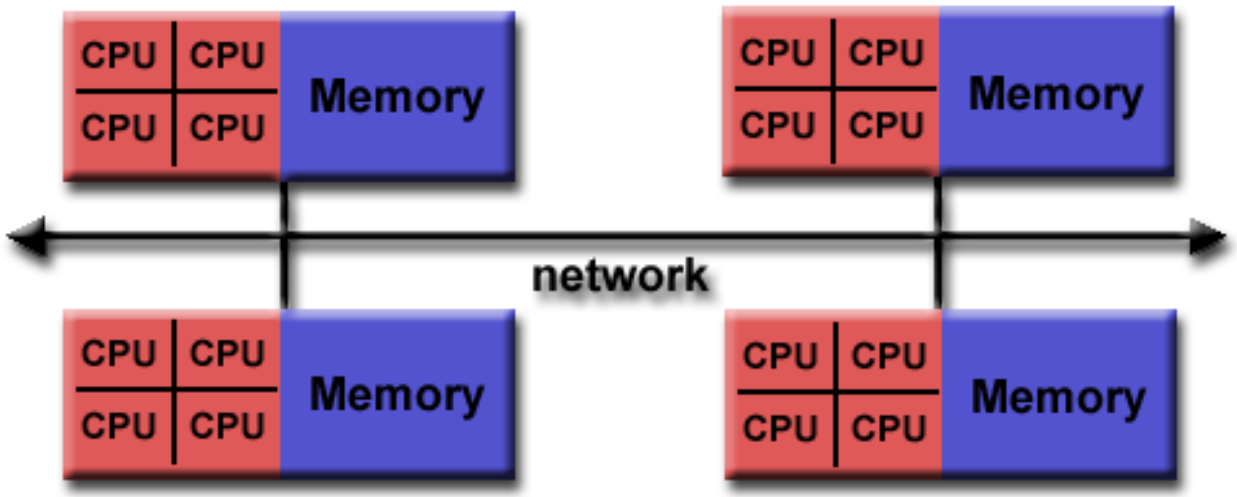
Shared memory : Advantages and Disavantages

- Avantages : Data sharing is fast and uniform (all processors have the same time to all memory)
- Disavantages
  - Lack of scalability of memory and processors : adding more memory and processors increases traffic on the shared memory-processors path, and for cache coherent system
  - Programmer responsibility for synchronization
  - Expensive to design and produce shared memory machines with ever increasing number of processors

Distributed memory : Advantages and Disavantages

- Avantages
  - Each processor can access rapidly to its own local memory without interference and without overhead incurred when trying to maintain cache coherency
  - Cheap for extension : can use commodity machine and networking
- Disavantages
  - Difficult to map existing data structures (based on global memory) to this memory organization
  - Programmer responsibility for exchanging data
  - Non-uniform memory access

Hybrid Distributed-Shared Memory



- The fastest and largest computers in the world today employ both shared and distributed memory architectures

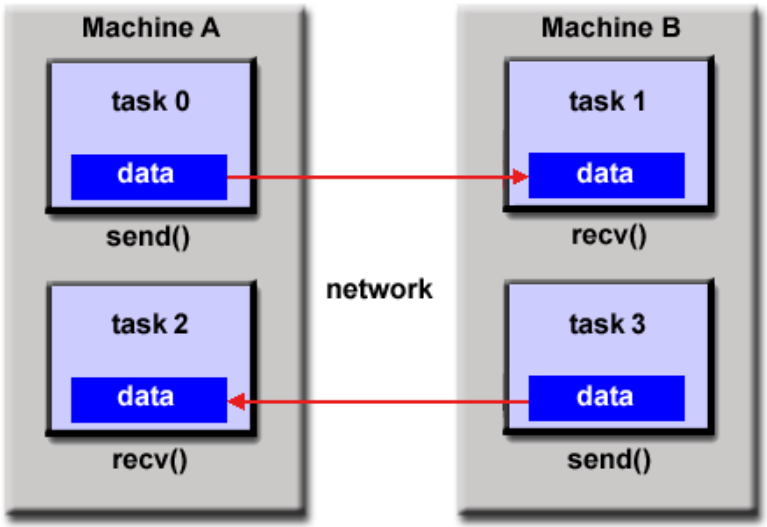
Outline

Parallel Programming Models - Thread model

- A single process can have multiple, concurrent execution paths
- A program (main routine of the application) performs some serial work, and then creates a number of tasks (threads) that can be scheduled and run by the operating system concurrently
- Each thread has local data, but also shares the entire resources of the program
- A program has subroutines and any thread can execute any subroutine
- A thread can communicate with other threads through global memory. This requires synchronization constructs to ensure that more than one thread cannot update the same global address at any time
- Threads can some and go but the main routine remains present to provide the neccessary shared resources until the application has completed
- Implementation : POSIX Threads and OpenMP

Parallel Programming Models - Distributed memory/Message Passing Model

- A set of tasks use their own local memory. Multiple tasks can reside on the same physical machine and/or across different machines
- Tasks exchange data through communications by sending and receiving messages
- Implementation : Most common is Message Passing Interface (MPI)

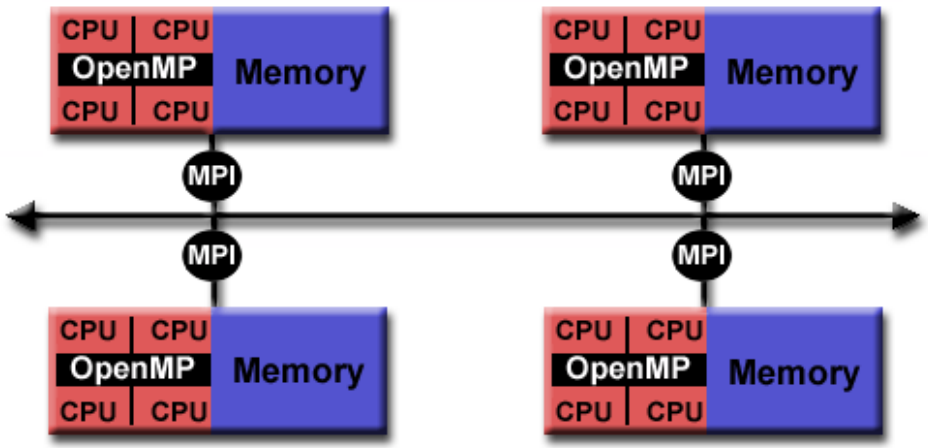


source :

https ://computing.llnl.gov/tutorials/parallel_comp/

Parallel Programming Models - hybrid model

- A common hybrid model is the combination between message passing model and threads model
  - Threads is responsible for intensive computation locally, on-node
  - Communication between processes on different nodes via networks using MPI



source :

https ://computing.llnl.gov/tutorials/parallel_comp/

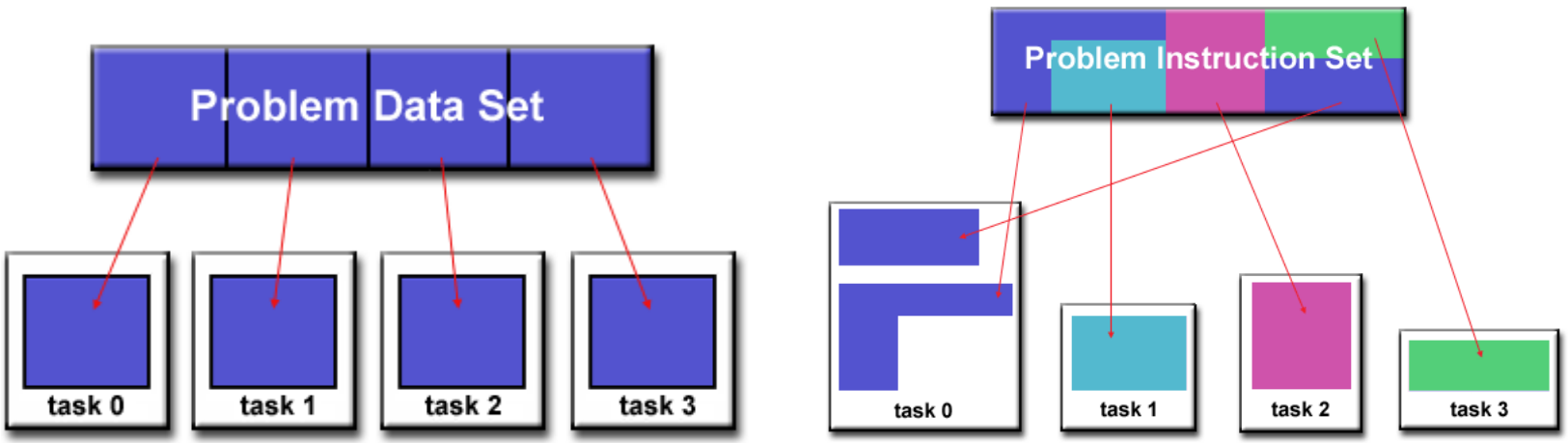
Outline

Designing Parallel Programs

- Study the problem to see weather or not it can be solved by parallel programs
  - Calculate the sum of a large array : YES
  - Calculate the Fibonacci series : NO
- Study existing serial programs to see if it (or some sections) can be parallelized
- Investigate other algorithms if possible. For example when solving Minimum Spanning Tree :
  - KRUSKAL and PRIM algorithms cannot be parallized
  - BORUVKA algorithm can be
- Employ optimized parallel softwares and/or math libraries from leading vendors (IBM's ESSL, Intel's MKL, AMD's AMCL, etc.)

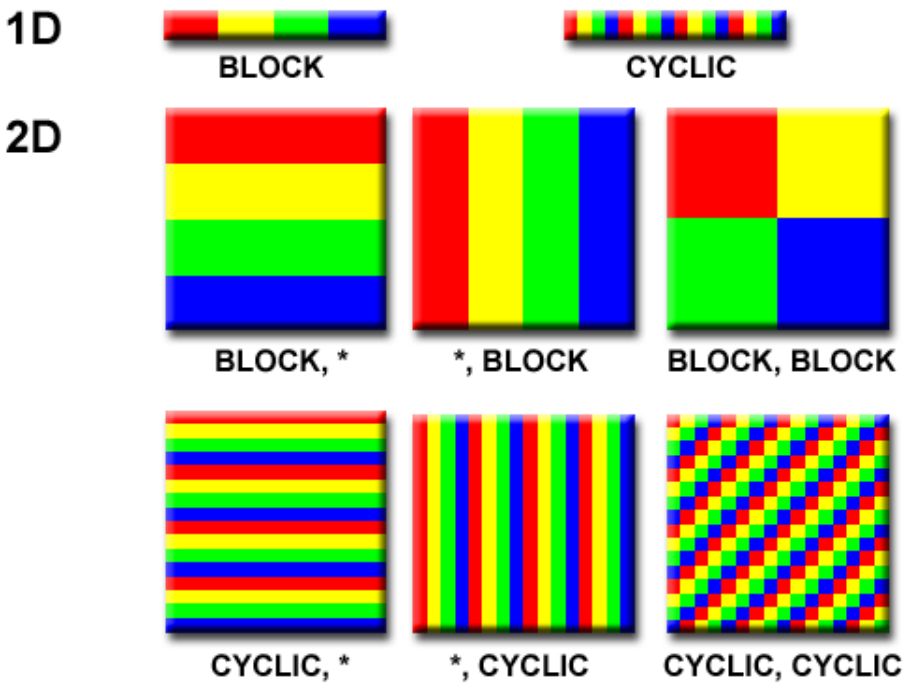
Designing Parallel Programs - partitioning

- Break the problem into chunks of works that can be distributed to multiple tasks
- Two basic ways
  - domain decomposition
  - functional decomposition



https ://computing.llnl.gov/tutorials/parallel_comp/

Designing Parallel Programs - domain decomposition



https ://computing.llnl.gov/tutorials/parallel_comp/

Designing Parallel Programs - domain decomposition

Example

- Compute the sum of N numbers of the array $A[1], \ldots, A[N]$
- Suppose $N = k * M$ where k is the number of tasks
- Task i compute the sum $A[M * (i-1) + 1] + \cdots + A[M * i]$

Designing Parallel Programs

- Communication
  - Some applications do not require communication : reverse a binary image
  - Most of applications need the communication
- Synchronization
  - Barrier
  - Lock/semaphore
  - Synchronization communication
- Load Balancing
  - Distribute works among tasks so that all tasks are kept busy all the time (minimize task idle time)
  - How ?
    - Equally partition the work each task receives
    - Use dynamic work assignment

10 most powerful supercomputers

- Source : www.top500
- Date : June 2013

index	Name	cores	memory (GB)	location	vendor
1	Tianhe-2	3,120,000	1,024,000	China	NUDT
2	Titan-Cray XK7	560,640	710,144	U.S.	Cray
3	Sequoia-BlueGene/Q	1,572,864	1,572,864	U.S.	IBM
4	K computer	705,024	1,410,048	Japan	Fujitsu
5	Mira - BlueGene/Q	786,432		U.S.	IBM
6	Stampede - PowerEdge C8220	462,462	192,192	U.S	Dell
7	JUQUEEN - BlueGene/Q	458,752	458,752	Germany	IBM
8	Vulcan - BlueGene/Q	393,216	393,216	U.S.	IBM
9	SuperMUC	147,456		Germany	IBM
10	Tianhe-1A	186,368	229,376	China	NUDT

Outline

High-performance computing Applications

- Medical imaging
- Financial trading
- Oil and Gas
- Bioscience
- Data warehousing
- Data compression, coder/decoder

Outline

Projects

- Shortest Path Algorithm : Δ-stepping parallel algorithm
- Hadoop system