Parallel Programming Models and Paradigms

An Introduction

Prof. Rajkumar Buyya

Cloud Computing and **D**istributed **S**ystems (CLOUDS) Lab**.**

The University of Melbourne, Australia

www.cloudbus.org

Presentation Outline

- Introduction
- Parallel Application Development Strategies
- Code Granularity and Levels of Parallelism
- Parallel Models
- Methodical Design Steps
- Parallelization Paradigms
- Summary

Parallel Programming is a Complex Task

- The development of parallel applications largely dependent on the availability of adequate software tools and environments.
- Parallel software developers handle issues/challenges such as:
  - Non-determinism, communication, synchronization, data partitioning and distribution, load-balancing, fault-tolerance, heterogeneity, shared or distributed memory, deadlocks, and race conditions.

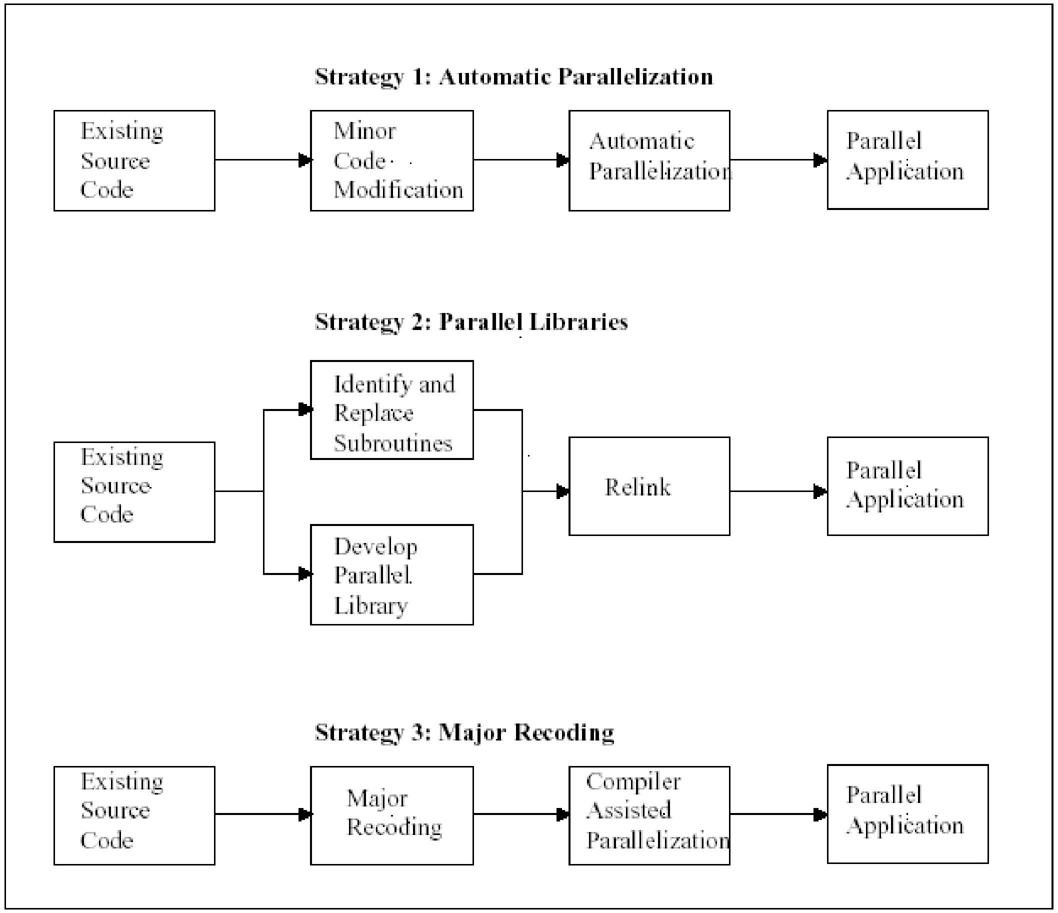
Users' Expectations from Parallel Programming Environments

- Currently, only few expert developers have the knowledge of programming parallel and distributed systems.
- Parallel computing can only be widely successful if parallel software is able to meet expectations of the users, such as:
  - provide architecture/processor type transparency;
  - provide network/communication transparency;
  - be easy-to-use and reliable;
  - provide support for fault-tolerance;
  - accommodate heterogeneity;
  - assure portability;
  - provide support for traditional high-level languages;
  - be capable of delivering increased performance; and finally,
  - to provide parallelism transparency.

Approaches for Parallel Programs Development

- Implicit Parallelism
  - Supported by parallel languages and parallelizing compilers that take care of identifying parallelism, the scheduling of calculations and the placement of data.
- Explicit Parallelism
  - In this approach, the programmer is responsible for most of the parallelization effort such as task decomposition, mapping task to processors, the communication structure.
  - This approach is based on the assumption that the user is often the best judge of how parallelism can be exploited for a particular application.

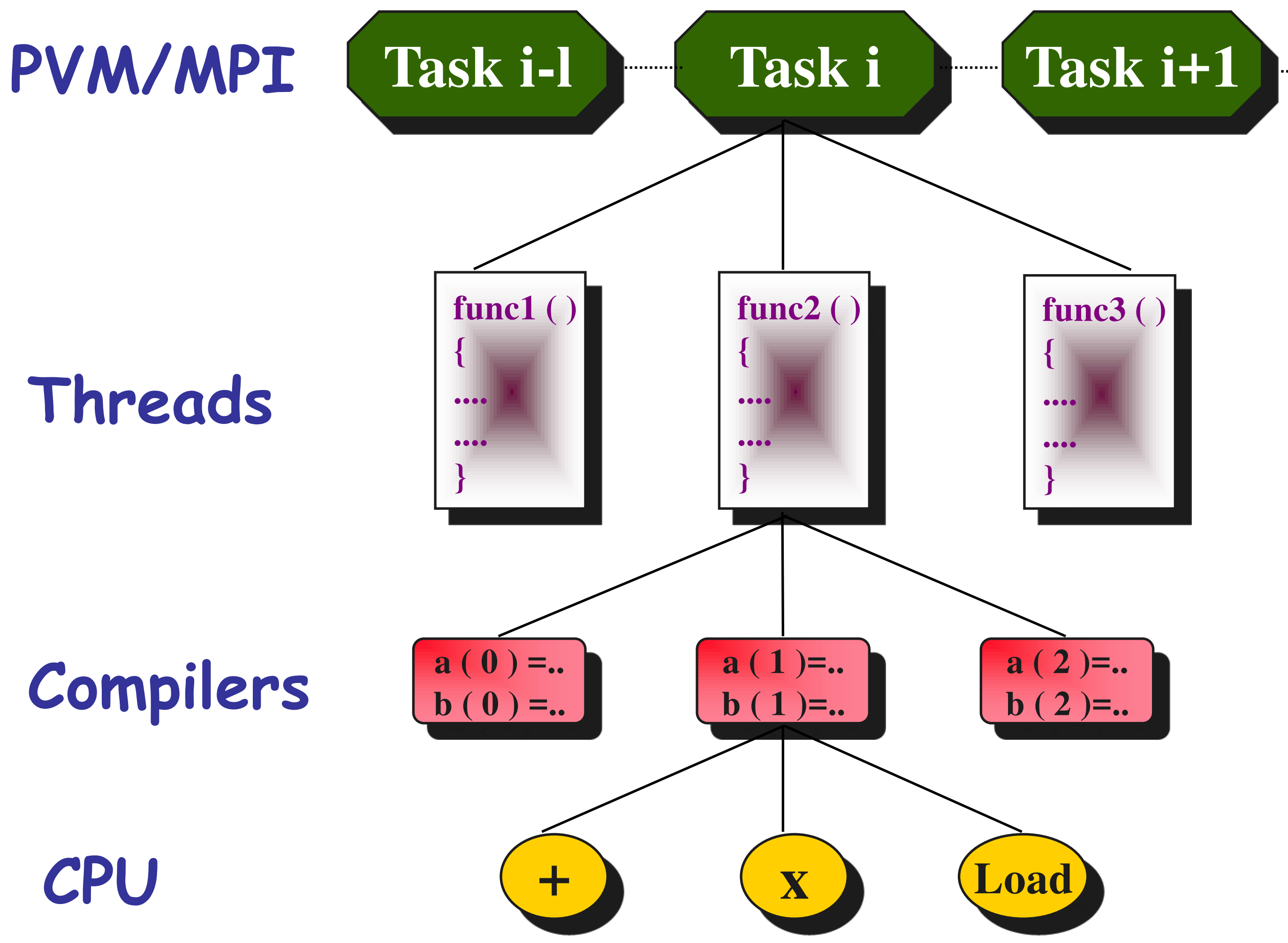
Strategies for Development



Parallel Programming Models

- Shared Memory Model
  - DSM (Distributed Shared Memory)
  - Threads/OpenMP (enabled for clusters)
  - Java threads (HKU JESSICA, IBM cJVM)
- Message Passing Model
  - PVM (Parallel Virtual Machine)
  - MPI (Message Passing Interface)
- Hybrid Model
  - Mixing shared and distributed memory model
  - Using OpenMP and MPI together
- Object and Service Oriented Models
  - Wide area distributed computing technologies
    - OO: CORBA, DCOM, etc.
    - Services: Web Services-based service composition

Levels of Parallelism



Code-Granularity
Code Item

Large grain
(task level)
Program

Medium grain
(control level)
Function (thread)

Fine grain
(data level)
Loop (Compiler)

Very fine grain
(multiple issue)
With hardware

Responsible for Parallelization

Grain Size	**Code Item**	**Parallelised by**
Very Fine	Instruction	Processor
Fine	Loop/Instruction block	Compiler
Medium	(Standard one page) Function	Programmer
Large	Program/Separate heavy-weight process	Programmer

Methodical Design or Stages of Parallel Programs

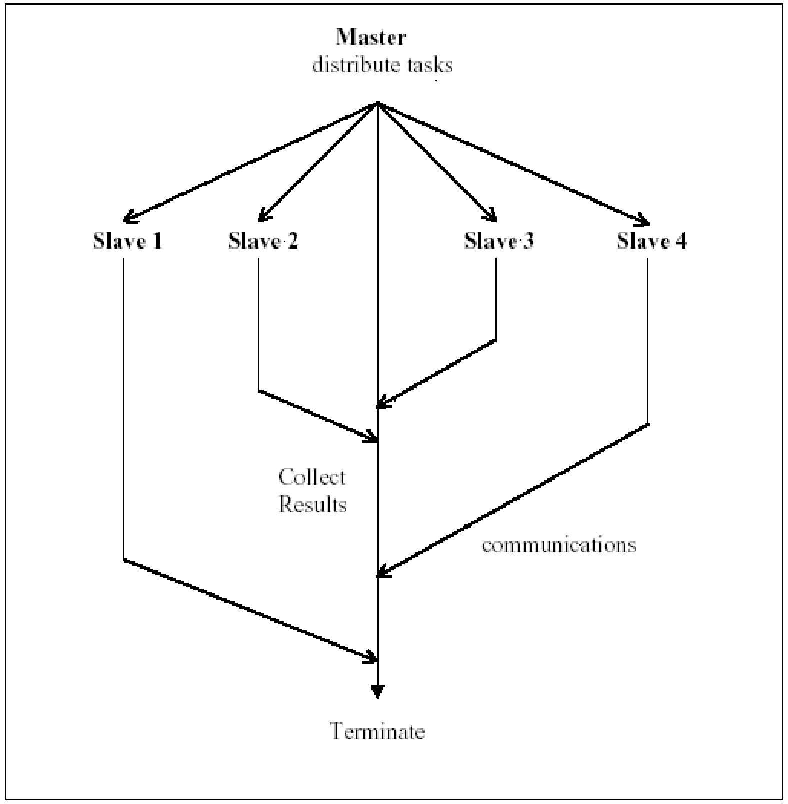
- Partitioning
  - Decomposition of computational activities and the data into small tasks – there exist number of paradigms – e.g. master worker, pipeline, divide and conquer, SPMD, and speculation.
- Communication
  - Flow of information and coordination among tasks that are created in the portioning stage.
- Agglomeration
  - Tasks and communication structure created in the above stages are evaluated for performance and implementation cost. Tasks may be grouped into larger tasks to improve communication. Individual communications can be bundled.
- Mapping / Scheduling
  - Assigning tasks to processors such that job completion time is minimized and resource utilization is maximized. Even the cost of computation can be minimized based on QoS requirements.

Parallelisation Paradigms

- Task-Farming/Master-Worker
- Single-Program Multiple-Data (SPMD)
- Pipelining
- Divide and Conquer
- Speculation
- Parametric Computation
  - Nimrod-G - for computational intensive parameteric applications.
  - Gridbus Broker – for distributed data intensive parameteric applications.

Master Worker/Slave Model

- Master decomposes the problem into small tasks, distributes to workers and gathers partial results to produce the final result.
- Mapping/Load Balancing
  - Static
  - Dynamic
    - When number of tasks are larger than the number of CPUs / they are know at runtime / CPUs are heterogeneous.



Static

Single-Program Multiple-Data

- Most commonly used model.
- Each process executes the same piece of code, but on different parts of the data.—splitting the data among the available processors.
- Different names: geometric/domain decomposition, data parallelism.

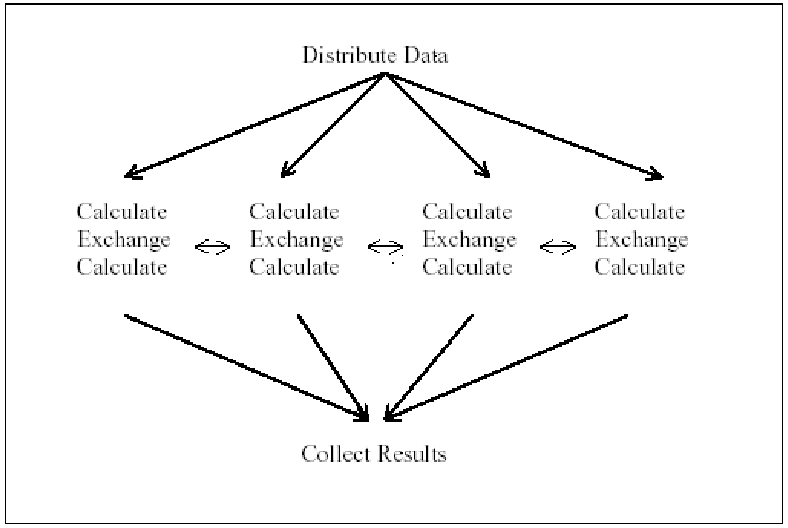


Figure 1.5 Basic structure of a SPMD program.

Data Pipelining

- Suitable for fine grained parallelism.
- Also suitable for application involving multiple stages of execution, but need to operate on large number of data sets.

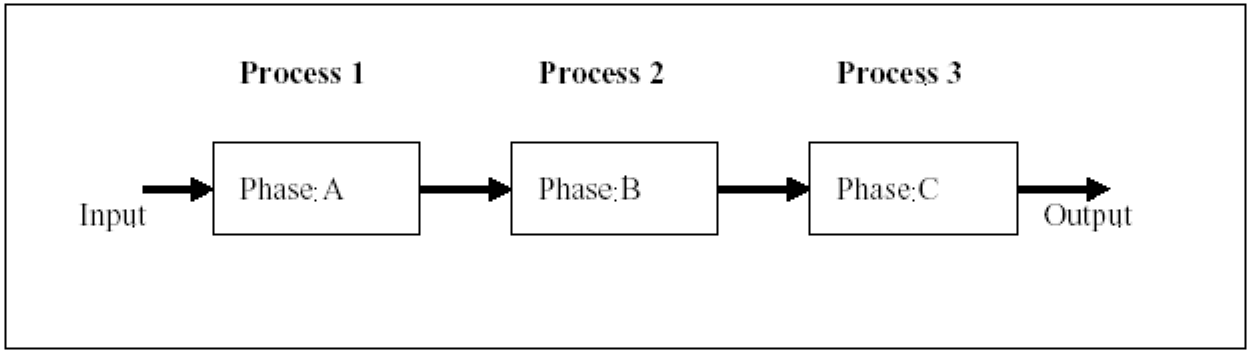


Figure 1.6 Data pipeline structure.

Divide and Conquer

- A problem is divided into two or more sub problems, and each of these sub problems are solved independently, and their results are combined.
- 3 operations: split, compute, and join.
- Master-worker/task-farming is like divide and conquer with master doing both split and join operation.
- (seems like a "hierarchical" master-work technique)

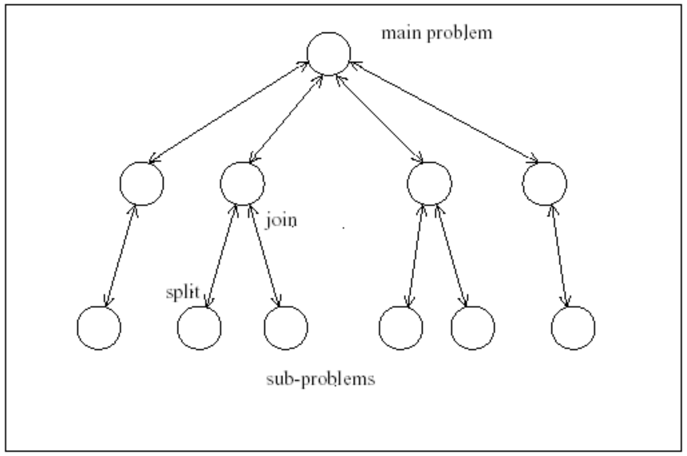


Figure 1.7 Divide and conquer as a virtual tree.

Speculative Parallelism

- It used when it is quite difficult to achieve parallelism through one of the previous paradigms.
  - Studying Grid computing/Java instead of AI/C++ course increases the likelihood/chances of getting a job is an example of speculation!
- Problems with complex dependencies – use "*look ahead*"execution.
- Employing different algorithms for solving the same problem—the first one to give the final solution is the one that is chosen.

Summary

- Parallel programming is complex, but exciting as it appears similar to a real world working model.
- Parallel Application Development:
  - Strategies: automatic, library-based, and explicit parallelisation.
  - Levels: fine grain, loop, function, and task
  - Models: shared, message passing, hybrid, OO/Service Oriented
  - Methodical design: partitioning, communication, agglomeration, and mapping.
  - Parallelization Paradigms: Master worker/task farming, SPMD, pipelining, divide and conquer, and speculations.

Reference

- L. Silva and R. Buyya, *Parallel Programming Models and Paradigms*, High Performance Cluster Computing: Programming and Applications, Rajkumar Buyya (editor), ISBN 0-13-013785-5, Prentice Hall, NJ, USA, 1999.