CMPSCI 590S Systems for Data Science **Fall 2017**

Lecture 4

Lecturer: Emery Berger *Scribe: Jayanth Hegde, Nidhi Mundra*

4.1 Overview

The class started with discussion on how multiprocessing is difficult to implement and Dennard scaling is becoming obsolete. The solutions to this problem is: Parallel processing.

4.2 Approaches to Performing Parallel Processing

Parallel processing can be performed using two approaches: Shared Memory and Distributed Memory.

The shared memory approach makes use of threads while distributed memory makes use of processes on multiple machines. In case of the latter, explicit communication mechanisms such as message passing are required.

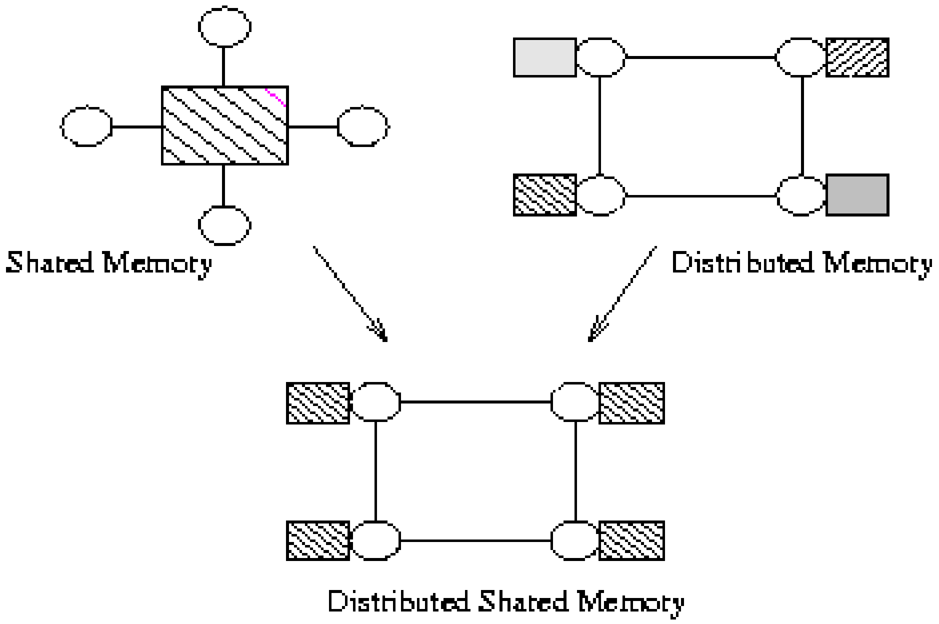


Figure 4.1: Shared memory and Distributed memories

Message passing and threads are duals in the sense that thread actions become communication at the hardware level.

4.3 Threads vs Processes

1) Heap is shared amongst various threads while processes do not share the heap.
2) Global variables are shared amongst threads while processes do not share them.
3) Code is shared in case of both threads and processes.
4) Stack variables are not shared in case of both threads and processes.

4.4 Race condition

Non-determinism in the access of shared resources between threads.
To prevent racing, the ordering of events must be controlled. This can be done in the following ways:

1) *Wait for threads*
2) *Synchronization operation*: Prevents concurrency and orders events. A lock/mutex is used for this purpose. A condition variable can also be used to do this. A condition variable incorporates a signaling along with locking mechanism. A thread can wait over a condition variable for a resource under use. As soon as the resource becomes available, the waiting thread is signaled to resume execution by acquiring the resource. In Java, the wait and notify mechanism provides the same functionality. This method can be used to enforce happens-before relation among events.

4.5 Challenges of Multithreading

1) Correctness: If proper care is not taken, heisenbugs can arise due to inherent non-determinism in multi-threaded programs.
2) Performance: Ironically, multithreaded programs can perform badly if synchronization is done inappropriately.
3) Load imbalance: It is possible to have situations where one thread does more work than the other.

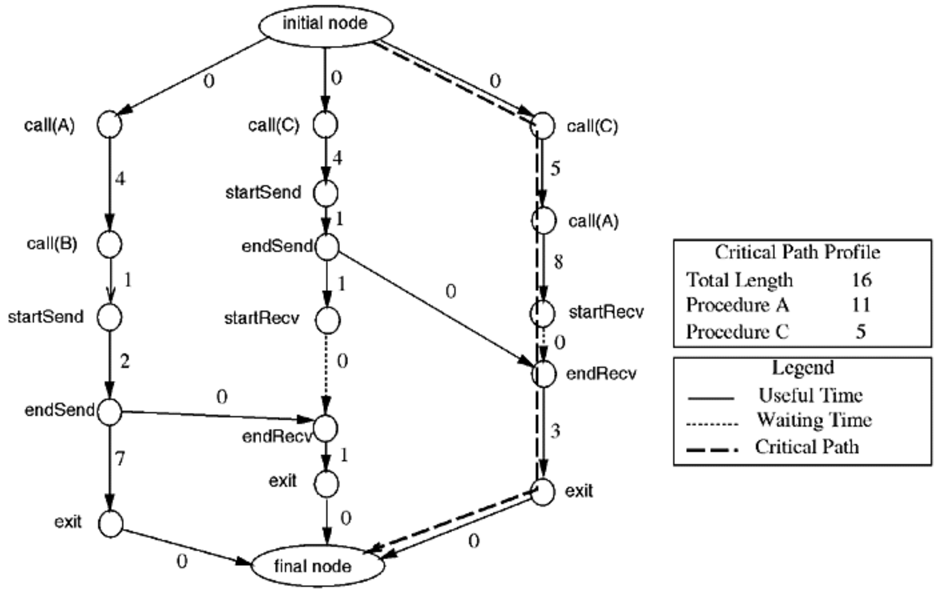
4.6 Some Definitions

1) *Critical Path or Span of computation*: The maximum sequential path of execution in a multithreaded program. Look at figure 4.2.

2) *Embarrasingly parallel*: A problem that lends itself naturally to parallel processing.

3) *Heisenbug*: A bug that surfaces rarely under certain conditions.

4) *Gustafson's law*: The computation can be scaled in accordance to the scale of the task provided the growth of the problem is comparable to growth in the power of computation. Eg: Creating inverted index of webpages.



Critical Path Profile	
Total Length	16
Procedure A	11
Procedure C	5

Figure 4.2: Critical Path