PMC Lecture 09

Gianmaria Romano

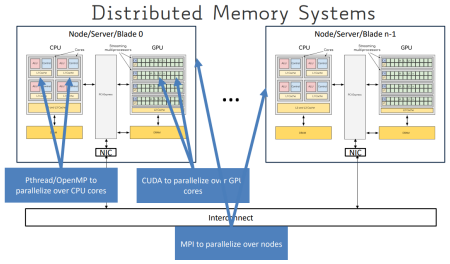
29 October 2025

Chapter 3

Threads

3.1 Defining the main aspects of threads

When dealing with distributed memory systems, it can come in handy to parallelize computations by implementing a message passing interface and, at the same time, each system node handles its CPU/GPU activities using threads that are able to communicate through a shared DRAM.



Generally speaking, threads are chosen over processes because they represent the smallest independent instance that can be executed by a computer.
Nowadays, most operating systems implement threading by referring to the POSIX Threads standard, which, however, is typically implemented using the OpenMP API for better portability.

3.2 Race conditions

A race condition is a problematic situation where the result of a program depends on the non-deterministic scheduling of the threads executing the critical section, resulting in data inconsistencies or unexpected outcomes.
For this reason, it is a good practice to implement a countermeasure, such as

a lock, that makes sure that only one process at a time can access the critical section of a program.

3.2.1 Busy waiting

Busy waiting is a countermeasure where a thread can access the critical section only when a certain condition, typically relative to its rank or to a lock, is achieved.
While this approach is relatively easy to implement, it is considered to be inefficient as the thread always wastes CPU cycles without doing anything.

3.2.2 Mutexes and semaphores

A mutex is a special generalization of a lock that, by using a dedicated inner variable, guarantees mutual exclusion by restricting the access to the critical section to just one thread at a time.

3.3 Deadlock and starvation

Starvation is a problematic situation where, due to unfair scheduling, the execution of a thread is suspended for an indefinite amount of time, although, eventually, the thread will still be able to continue its execution.
On the other hand, a deadlock is a more dangerous situation where no thread can proceed because each entity waits for another entity (or itself) to perform a certain action, resulting in a case of cyclical wait.

3.4 Properly using threads in message passing interfaces

One of the main issues of message passing interface lies in the fact that communication is not thread-safe.
For this reason, when working with threads, it is important to understand which threading level is supported by the message passing interface and whether this level matches with the required threading level.
Generally speaking, message passing interfaces can support one of the following threading levels:

- **Single:** the message passing interface does not support, meaning that the message passing interface will be used just by one main thread.
- **Funneled:** multiple threads can be used, but only the master thread can interact with the message passing interface.
- **Serialized:** multiple threads can interact with the message passing interface, although, at any given time, only one can communicate with the message passing interface.

- **Multiple:** multiple threads can interact with the message passing interface, even simultaneously, although this approach tends to be less efficient compared to funneled or serialized threading.