Parallel Computing/Programming

Distributed memory programming with Message Passing

João Luís Ferreira Sobral
Departamento do Informática
Universidade do Minho

Nov 2022

Computação/Programação Paralela

Níveis de paralelismo (Software/Hardware) (revisão)

Instrução (ILP)

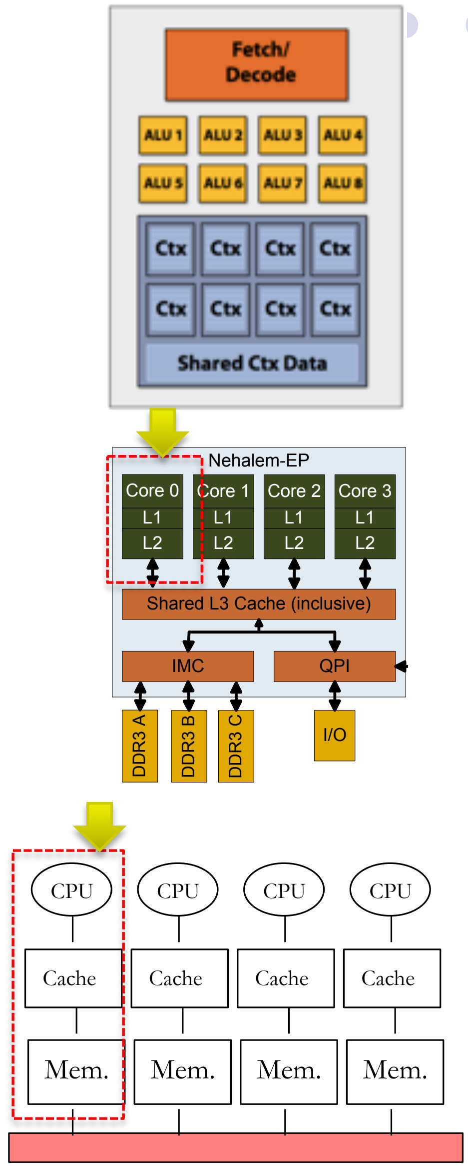
- **Execução de múltiplas instruções de um programa em paralelo**
- **Processamento vetorial (SIMD)**
- **Explorado pelo hardware atual**
- **Limitado pelas dependências de dados/controlo do programa**

Tarefas / fios de execução

- **múltiplos fluxos de instruções de um mesmo programa executam em paralelo**
- **Limitado pelas dependências e características do algoritmo**

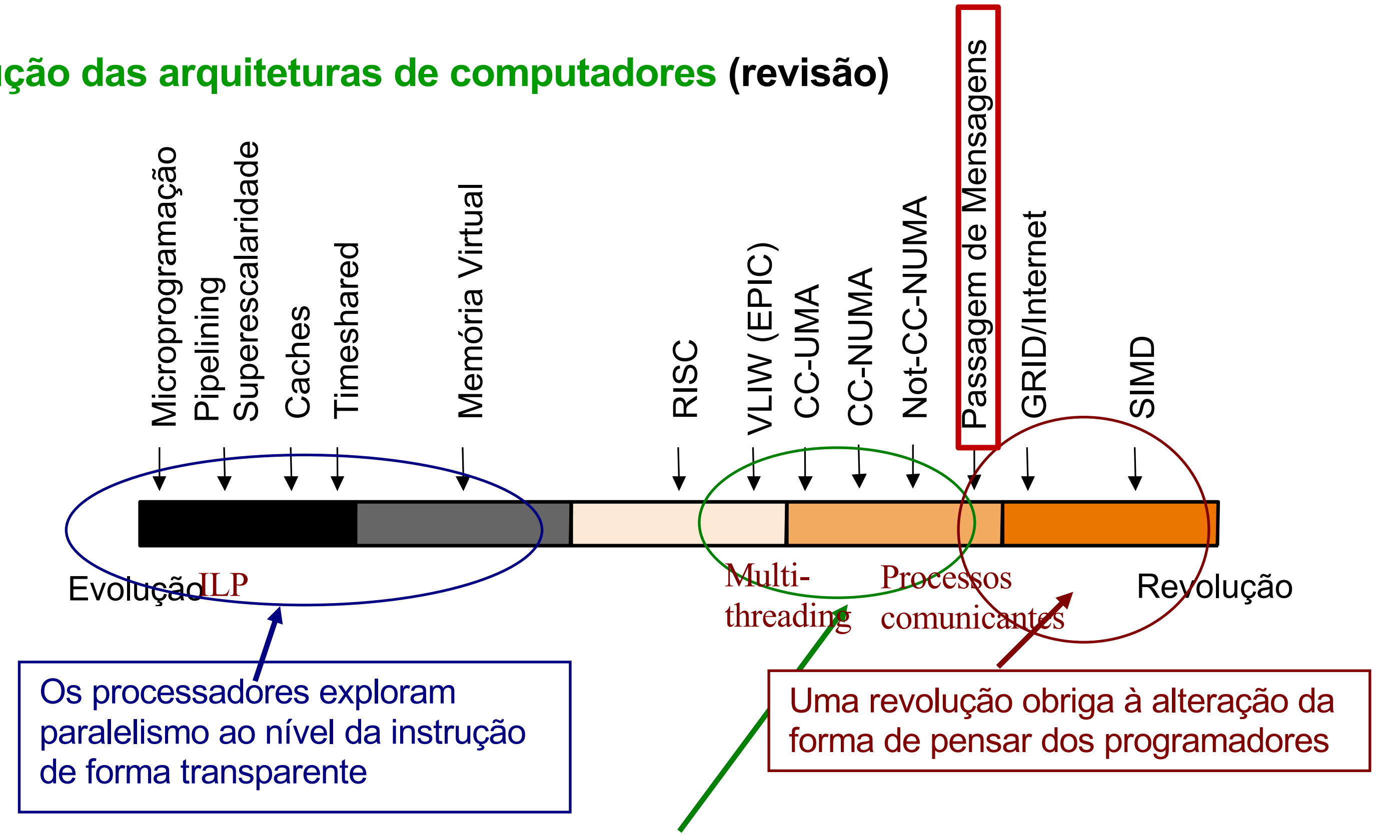
Processos

- **Múltiplos processos de um mesmo programa / ou de vários programas**



Computação/Programação Paralela

Evolução das arquiteturas de computadores (revisão)



A ênfase de computação paralela é na programação deste tipo de arquitecturas

Specification of concurrency/parallelism

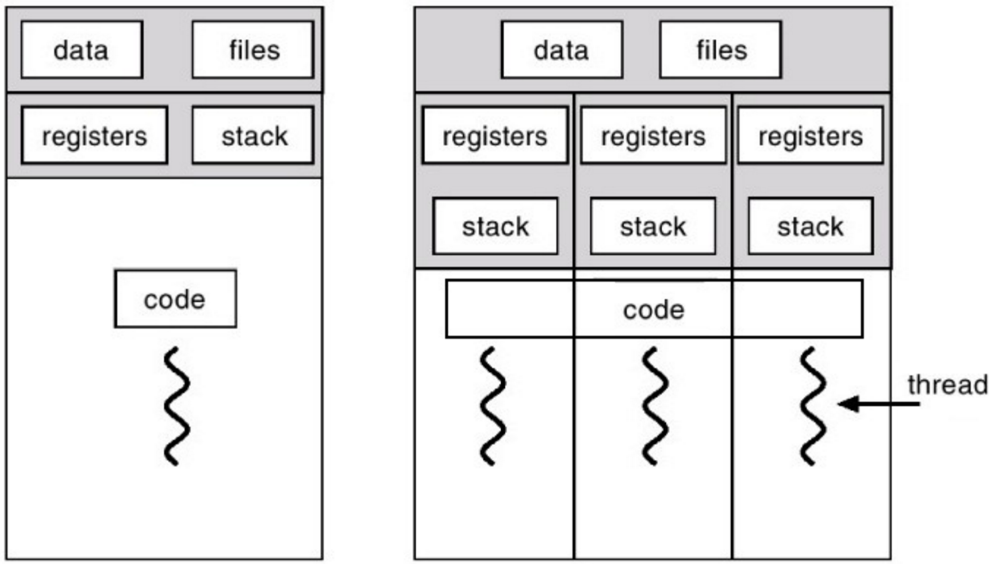
Processes

- **Used for unrelated tasks**
  - (e.g., a program)
- **Own address space**
  - Address space is proteded from other process
- **Swithching at the kernel level**

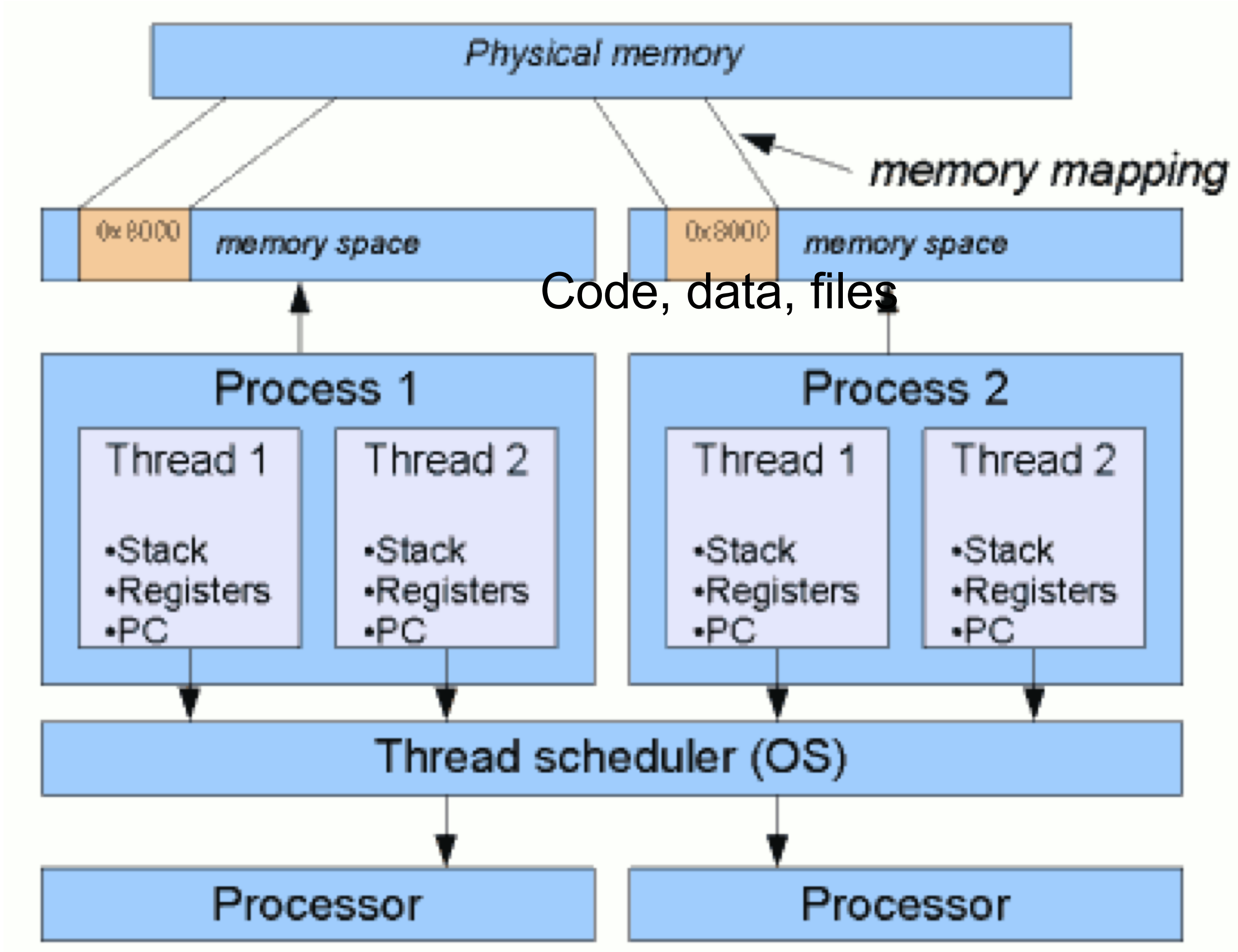
Threads

- **Are part from the same job**
- **Share address space, code, data and files**
- **Swithching at the user or kernel level**

Every process has at lest one thread



Programming abstraction: Thread vs Process

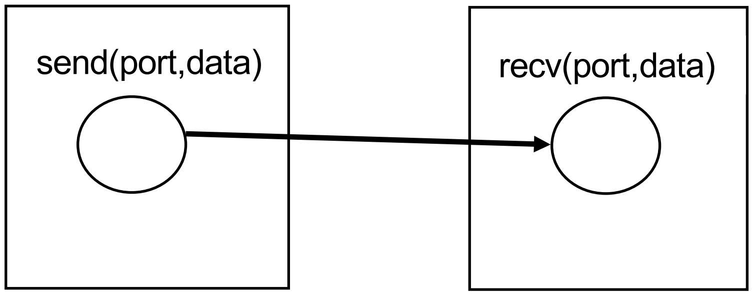


Message Passing

Basic concepts

- Specification of parallel activities through **processes** with **disjoint address spaces**
  - No shared memory among processes => message passing parallelism
  - Processes can be identical (Single Process Multiple data, SPMD, e.g., MPI) or not (Multiple Instructions Multiple Data, MIMD, e.g., PVM)

- Parallel activities **communicate** through ports or channels
  - Message **send** and **reception is explicit** (from/to a port or channel)



- **Data** must be **explicitly marshalled** into messages
- There are more sophisticated communication primitives (broadcast, reduction, barrier)

Message Passing

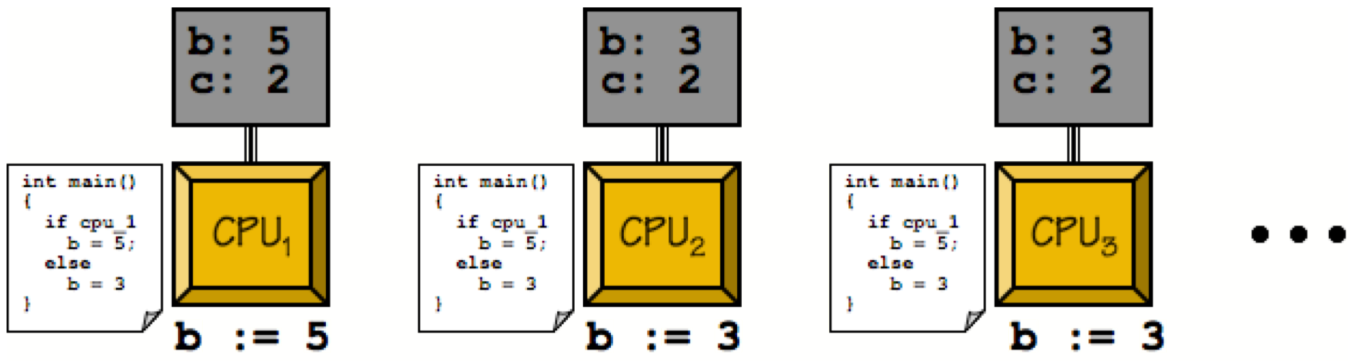
MPI (Message Passing Interface) http://www.mpi-forum.org

- **Standard** for message passing, outcome of an effort to provide a way to develop **portable** parallel applications (based on distributed memory)
- Based on the **SPMD** model (the same code is executed on all processes)
- Message passing with **in order message delivery** on point-to-point communication
- Implemented as a **library** of functions
- Common Libraries (Open Source): OpenMPI, MPICH and LamMPI
- **Main features**:
  - Several modes of message passing: synchronous / asynchronous
  - Communication groups / topologies
  - Large set of collective operations: Broadcast, Scatter/gather, reduce, all-to-all, barrier
  - MPI-2: Dynamic processes, parallel I/O, Remote memory access (RMA - put/get)
    - Limited shared memory programming
  - MPI-3: explicit shared memory programming

Message Passing

Single Program Multiple Data model (SPMD)

- **The same executable is launched on a set of processes (e.g. several machines)**
  - Asynchronous execution of the same program
  - Each process has an unique identifier
- The rank of each process is used to define each process-specific behaviour
  - Process-specific control flow
    - Data processing and inter-process communication
  - Example with 3 processes



- **Easy to write a program that works with an arbitrary number of process (machines)**

Message Passing

Structure of a MPI program

- **Initialize the library**
  - **MPI_Init - **Initializes the library
- **Get information for process**
  - **MPI_Comm_size**
    - Gets total number of process
  - **MPI_Comm_rank**
    - Get the id of current process
- **Execute the body of the program**
  - **MPI_Send / MPI_Recv**
    - Do processing and send/recv data
- **And cleanup**
  - **MPI_Finalize**

```
#include <mpi.h>
#include <stdio.h>
int main( int argc, char *argv[]) {
    int rank, msg;
    MPI_Status status;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank( MPI_COMM_WORLD, &rank );

    /* Process 0 sends and Process 1 receives */
    if (rank == 0) {
        msg = 123456;
        MPI_Send( &msg, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
    }
    else if (rank == 1) {
        MPI_Recv( &msg, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, &status );
        printf( “Received %d\n”, msg);
    }

    MPI_Finalize();
    return 0;
}
```

Compile and execute the program

- **compile:** mpicc (or mpicxx for C++)
- **execute:** mpirun –np <*number of processes*> a.out

Message Passing

MPI (Functionalities – cont.)

- **Point to point communication between processes**

  int MPI_Send(void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)

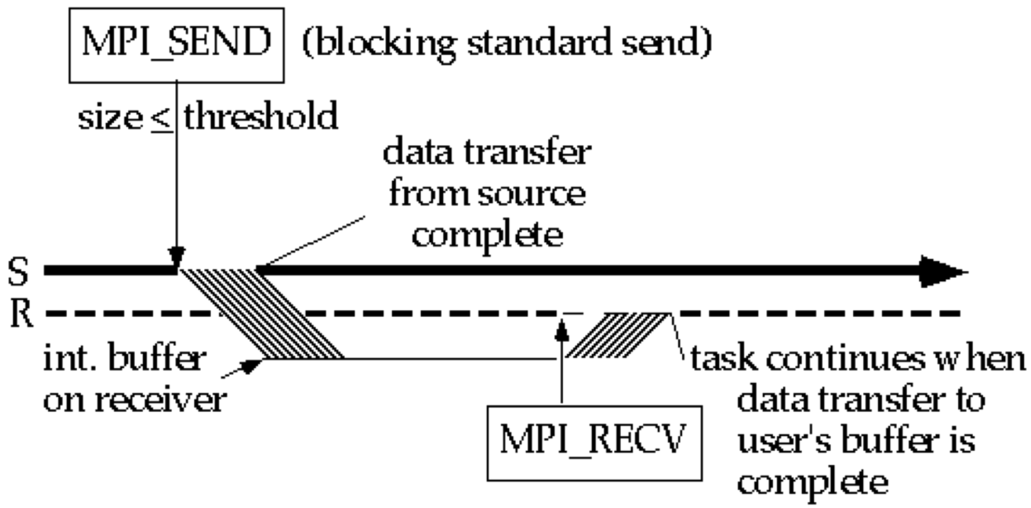
  int MPI_Recv(void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Status *status)

  - **Message data content:** void *buf, int count, MPI_Datatype datatype
    - Requires the specification of the data type (MPI_INT, MPI_DOUBLE, etc)
  - **Each process is identified by its rank in the group**
    - *dest* / *source* provide the destination / source of the message
    - By default there is a group comprising all processes: MPI_COMM_WORLD
  - **The tag can be used to make distinction among messages**
  - **MPI_Recv:** waits for the arrival of a message with the required characteristics
    - MPI_ANY_SOURCE and MPI_ANY_TAG can be used to identify any source / any tag

Message Passing

MPI – Modes of point-to-point communication

- **Message passing overhead**
  - Message transfer time (copy into the network, network transmission, deliver at the receptor buffer)



Message Passing

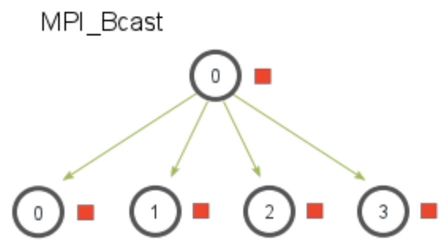
MPI – Modes of point-to-point communication

- “standard” MPI_Send may be implemented on a variety of ways
  - “MPI_Send will not return until you can use the send buffer. It may or may not block (it is allowed to buffer, either on the sender or receiver side, or to wait for the matching receive)” – *MPI standard*
- **Explicit Send implementations (**different options for **buffering** and **synchronization):**
  - **MPI_Ssend (blocking synchronous send)**
    - The sender waits until the message is received (w/ MPI_Recv on the destination process)
  - **MPI_Rsend (Ready send)**
    - Returns as soon as the message has been placed in the network
    - **The receptor side should already posted a MPI_Recv to avoid “deadlocks”**
  - **MPI_Bsend (Buffered send)**
    - Returns as soon as the message has been placed on a buffer on the sender side
    - **Does not suffers from the overhead of receptor synchronization, but may copy to a local buffer**
- **MPI_Ixxx (non-bloking sends) w/ MPI_wait / MPI_Test /MPI_Probe**
  - Returns immediately, being the programmer responsible to verify if the operation has completed (using wait)

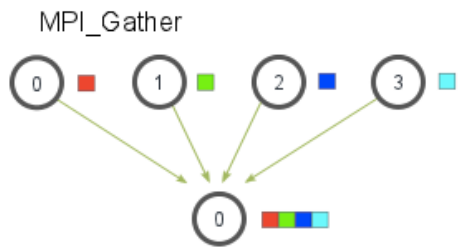
Message Passing

MPI – Collective communications

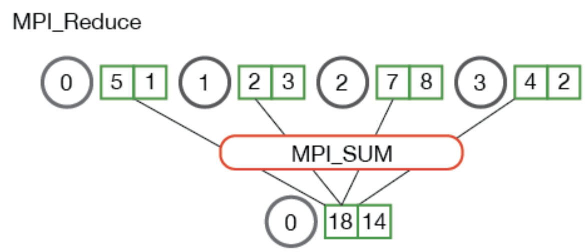
- int MPI_Barrier(MPI_Comm comm)
  - **Wait** until all processes arrive at the barrier



- int MPI_Bcast(void* buffer, int count, MPI_Datatype datatype, int root, MPI_Comm comm)
  - **Broadcast** the data from root to all other processes



- int MPI_Gather & int MPI_Scatter(void* sbuf, int scount, MPI_Datatype stype, void* rbuf, int rcount, MPI_Datatype rtype, int root, MPI_Comm comm)
  - **Gather**: Joints data from all processes into the root
  - **Scather**: scatters data from root into all other processes

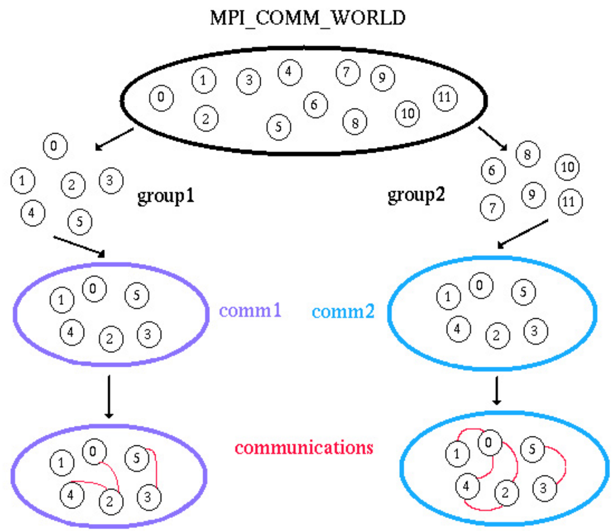


- int MPI_Reduce(void* sbuf, void* rbuf, int count, MPI_Datatype stype, MPI_Op op, int root, MPI_Comm comm)
  - **Combines** the results from all process into the root, using the operator *MPI_Op*
- Compositions: Allgather, Alltoall , Allreduce, Reduce_scatter

Message Passing

MPI – Groups

- Ordered group of process
  - Each process has a rank within the group
- Scope for communication on collective and point to point communications



MPI – Topologies

- Well defined structure of processes
  - Each process has a set of neighbours
    - Easier to identify with a topology
  - Communications though “channels”
  - Example: cartesian 4x4

0 (0,0)	1 (0,1)	2 (0,2)	3 (0,3)
4 (1,0)	5 (1,1)	6 (1,2)	7 (1,3)
8 (2,0)	9 (2,1)	10 (2,2)	11 (2,3)
12 (3,0)	13 (3,1)	14 (3,2)	15 (3,3)

Message Passing

MPI – C++

- **C++ Exemple**

```
#include "mpi.h"
#include <iostream>
int main( int argc, char *argv[]) {
        int rank, buf;
        MPI::Init(argv, argc);
        rank = MPI::COMM_WORLD.Get_rank();
        // Process 0 sends and Process 1 receives
        if (rank == 0) {
                buf = 123456;
                MPI::COMM_WORLD.Send( &buf, 1, MPI::INT, 1, 0 );
        }
        else if (rank == 1) {
                MPI::COMM_WORLD.Recv( &buf, 1, MPI::INT, 0, 0 );
                std::cout << "Received " << buf << "\n";
        }
        MPI::Finalize();
        return 0;
}
```