W • SENSE
INTEGRATED CABLELESS SOLUTIONS

SAPIENZA
UNIVERSITÀ DI ROMA

Programmazione di sistemi multicore MPI

Fabrizio Gattuso

Esercizio FreeRTOS

Esercizio da svolgere entro le ore 16:50.

Scrivere un programma composto da 3 task e un interrupt handler:

1) Due task che comunicano tra di loro (tramite un oggetto studiato in classe) e stampano il risultato
2) Un terzo task che ricevuto lo sblocco dall'interrupt (usate il pulsante come visto in classe) stampa "bottone premuto" e dopo 4 secondi (**utilizzando un timer** e non delay) fa blinkare 4 volte il LED2

Calcolo parallelo

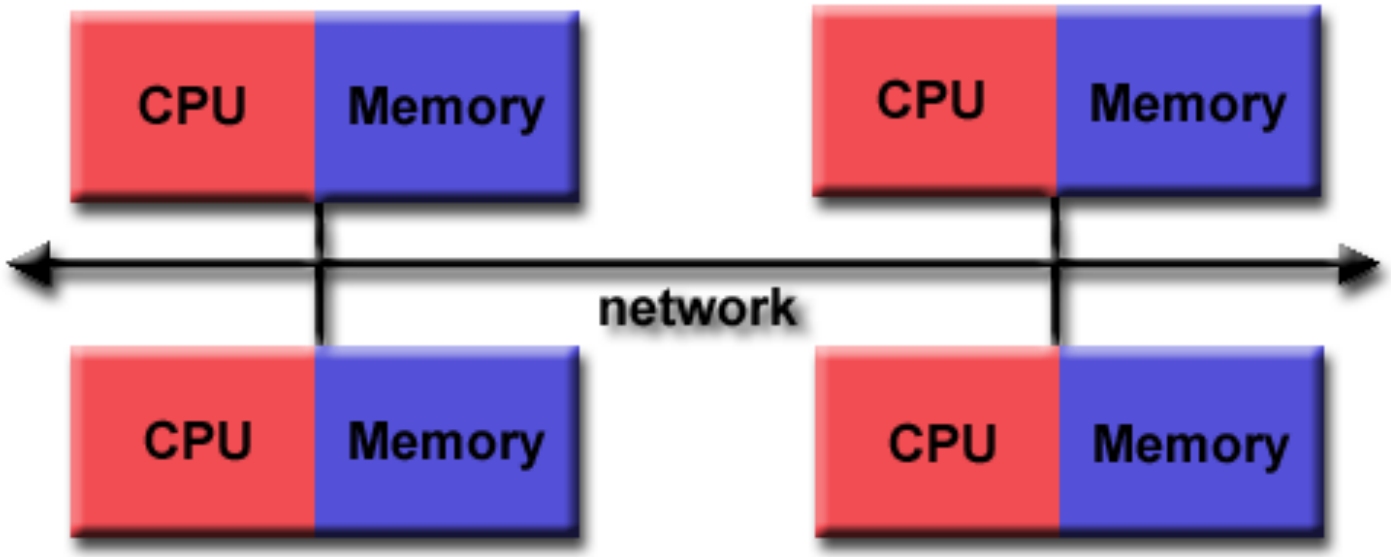
Il calcolo parallelo si basa sull'uso simultaneo di più processi per risolvere un problema in comune.

1. Il **problema** deve essere **diviso** in più **parti indipendenti** che possono essere eseguite in parallelo.
2. Per raggiungere un vero parallelismo ogni parte dovrà essere **eseguita** realmente in **contemporanea** ad altre parti.

Quindi il multitask e il multithreading non sono veri e propri parallelismi.

Message passing

Un paradigma utilizzato nel calcolo parallelo è il **message passing**.



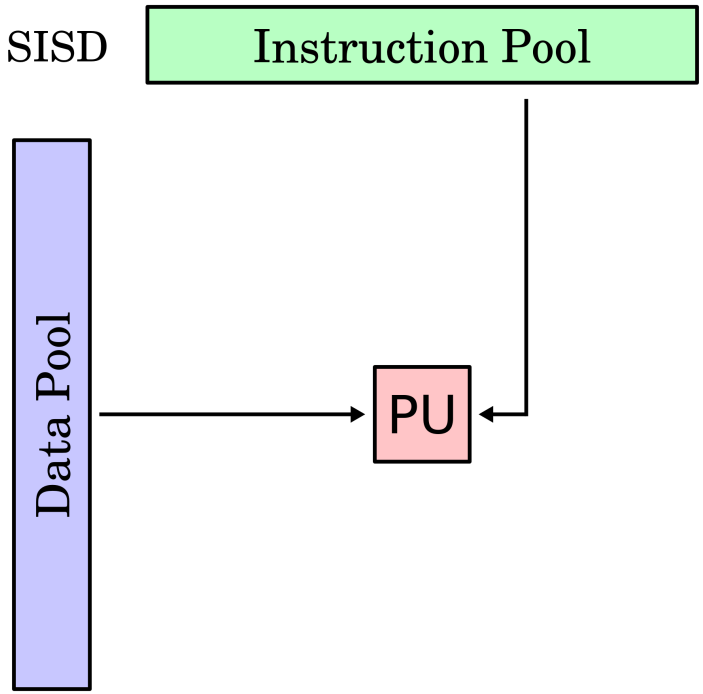
Message passing (2)

- Ogni processo **svolge in modo autonomo** la parte di task assegnato
- Ogni processo **accede in lettura e scrittura ai soli dati nella memoria riservata**
- È necessario **comunicare tramite messaggi** per **accedere alla memoria** di un altro processo o per **sincronizzarsi**

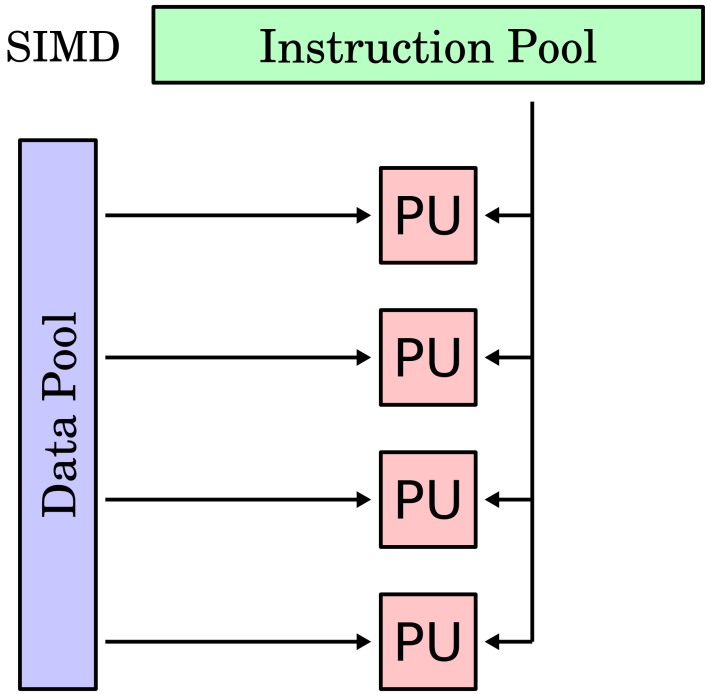
Vi ricorda qualcosa?

I modelli di esecuzione

Single Instruction Single Data

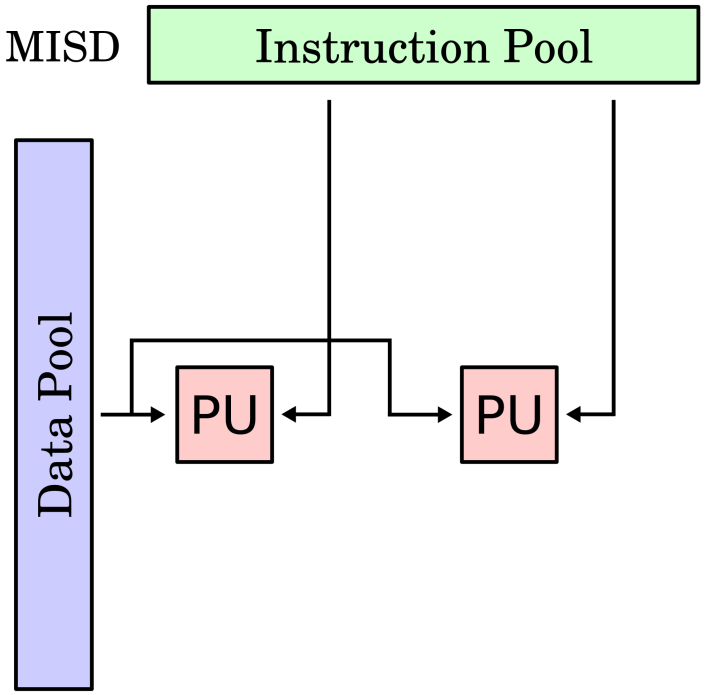


Single Instruction Multiple Data

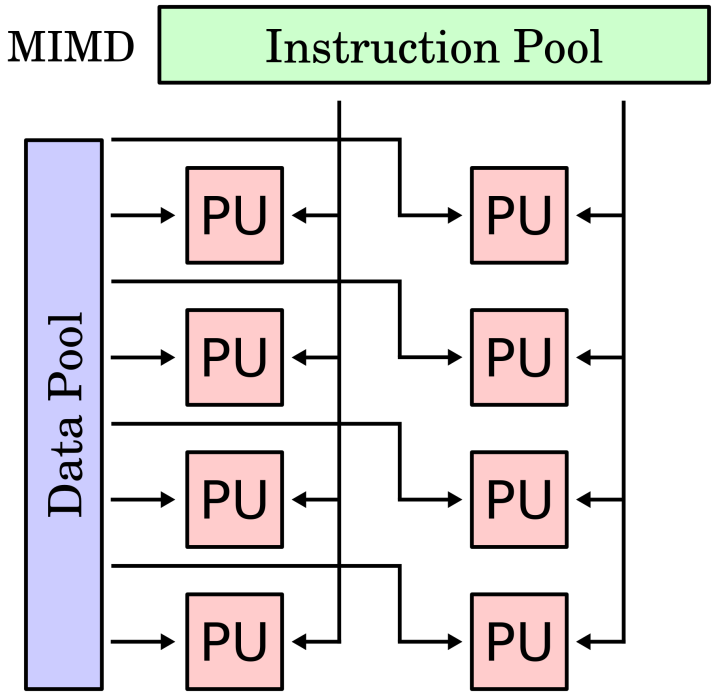


I modelli di esecuzione (2)

Multiple Instructions Single Data



Multiple Instructions Multiple Data



SPMD

Il modello di esecuzione che riflette il message passing è detto **SPMD**:

Single Program Multiple Data

Ogni processo esegue lo stesso programma ma opera su dati diversi o su parti di codice diffrenti ma dello stesso programma

L'implementazione di MPI che useremo è **OPEN MPI**.

MPI

MPI viene definito come specifica e poi viene implementato da diverse librerie:

OPEN MPI, INTEL MPI, MPICH

È una libreria che permette l'implementazione del calcolo parallelo tramite message passing. Permette la:

- comunicazione tra processi/core,
- la gestione dei singoli
- e utilities per semplificare la vita al programmatore.

OPEN MPI è disponibile per C, Fortran e tramite wrapper per Python.

Il primo programma

Open MPI si pone come un'estensione al C standard.

Fase di inizializzazione e terminazione

MPI_Init(int* argc, char*** argv)
MPI_Finalize()

Tutte le funzioni MPI_* vanno chiamate subito dopo l'init e prima del finalize.

Informazioni sull'environment

MPI_Comm_rank(MPI_Comm communicator, int* rank)
MPI_Comm_size(MPI_Comm communicator, int* size)
MPI_Get_processor_name(char* name, int* name_length)

Il primo programma (2)

Per compilare il programma:

mpicc –o mpiprogram mpiprogram.c

Per eseguirlo:

mpirun –n x mpiprogram

X è il numero di core che vogliamo utilizzare.

Il numero massimo dei core utilizzabili è dato dalle capacità hardware.

La comunicazione

Dato il paradigma lo strumento base di MPI è la comunicazione.

La comunicazione può essere intesa come:

- Interna alla macchina (core to core)
- All'interno della rete

Infatti MPI è in grado di creare **cluster** di macchine, in modo da porter **distribuire il calcolo** su tutte le risorse disponibili nella rete.

È usato nei più grandi supercomputer mondiali e in particolare nel calcolo scientifico e nelle simulazioni.

La comunicazione (2)

La comunicazione può essere utilizzata per:

- **scambio di dati**
- **sincronizzazione**

e può essere del tipo:

- **punto a punto**
- **broadcast**
- **sottoinsieme di nodi**

Punto a punto

La comunicazione punto a punto è la base di tutte le altre tipologie e si basa sulle funzioni:

MPI_Send(void* **data**, int **count**, MPI_Datatype datatype, int **destination**, int tag, MPI_Comm **communicator**)

MPI_Recv(void* **data**, int **count**, MPI_Datatype datatype, int **source**, int tag, MPI_Comm **communicator**, MPI_Status* **status**)

*Se la sorgente è impostata a **MPI_ANY_SOURCE** verranno ricevuti tutti i messaggi nel **communicator**.*

Punto a punto (2)

MPI datatype	C equivalent
MPI_SHORT	short int
MPI_INT	int
MPI_LONG	long int
MPI_LONG_LONG	long long int
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHORT	unsigned short int
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_UNSIGNED_LONG_LONG	unsigned long long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double
MPI_BYTE	char

Esercizio MPI

Scrivere un programma che sfrutti i core del proprio laptop per:

1) Creare un **vettore di interi di N dimensioni** (dove N è il vostro comm_size)
2) **Ogni core calcola la somma parziale** della dimensione n-esima corrispondente al numero di core
3) **Un core** si occupare di calcolare la **somma totale** e di stamparla