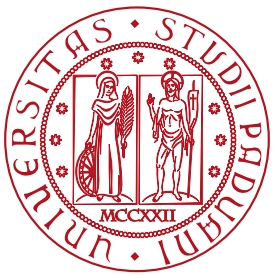
Calcolo Parallelo – Progetto

Lezione 1 – Aspetti pratici, Makefile . . .

Michele Schimd

schimdmi@dei.unipd.it

2 maggio, 2019



Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Reminder

- Il progetto è facoltativo, ma non farlo implica **2 punti di penalità** sul voto finale.
- Il progetto va fatto in gruppo 2–4 persone.
- Il gruppo va comunicato via email insieme all'argomento sul quale si vuole svolgere il progetto.
- Riceverete le credenziali per accedere alla macchine parallela Power 7 dopo aver comunicato il gruppo
- Il progetto va conseganto 7 giorni prima della registrazione.
- Il Power 7 rimarrà disposizione fino a settembre dopodiché verrà spento.

Una volta ricevute le credenziali di accesso connettersi **con SSH**.

- Da una **shell** (Linux, Mac OX, Windows powershell, CygWin, . . .)
  `$ ssh utente@power7a.dei.unipd.it`
- utilizzando programmi appositi: Putty, Filezilla (no shell, solo file transfer), . . .
- Al primo accesso cambiate subito la password con il comando
  `$ passwd`

Spero di no, ma. . .

Se non avete mai usato una shell è giunto il momento di *provare l'ebrezza*. A cosa serve il tasto `TAB`? (Usatelo e inizierete ad odiare mouse e finestre. . .)

I seguenti comandi dovrebbero essere noti a tutti...

- `cd` **c**hange **d**irectory
- `ls` **ls**t directory content
- `mkdir` **m**a**k**e **dir**ectory
- `rm` re**m**ove (`-r` per rimuovere directories)
- `mv` **mv**e element(s)
- `man` **man**ual pages
- `passwd` change **passw**or**d**

Sono disponibili alcuni programmi utili (editor, compilatori, archiviazione)

- `nano` Editor di base
- `vim` Editor avanzato
- `emacs` Editor definitivo
- `xlc` Compilatore IBM
- `tar` Creare archivi

La maggior parte delle shell offre il comando `scp` (`man scp`) per copiare file utilizzando una connessione sicura (*secure copy*).

- Da locale a remoto (file)
  ```
  scp hello.c user@power7a.dei.unipd.it:/remotedir
  ```
- Da remoto a locale (`./`)
  ```
  scp user@power7a.dei.unipd.it:/remotedir/hello.c ./
  ```
- per copiare directory usate `-r`
  ```
  scp -r src/ user@power7a.dei.unipd.it:
  ```

Notate **sempre** il carattere `:` (due punti) tra host e path.

Shame on you!

Se usate Filezilla o simili non dovete preoccuparvi di questo (ma che gusto c'é?)

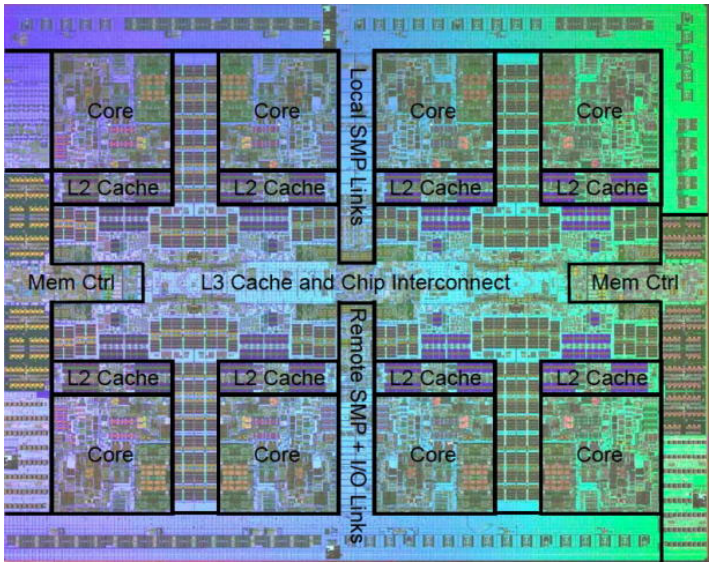
System

- Three drawers, each with 2 IBM Power7 Processors
- 6 Power7 Processors $\Rightarrow$ 48 cores
- 640 GB RAM
- 16 TB hard disk space (external SAN storage)
- Suse Linux Enterprise 11, AIX 6.1

Failures

Recently the system had a failure and we have no longer access to the whole resources thus you now have available

- 4 Power7 Processors $\Rightarrow$ 32 cores
- 348 GB RAM

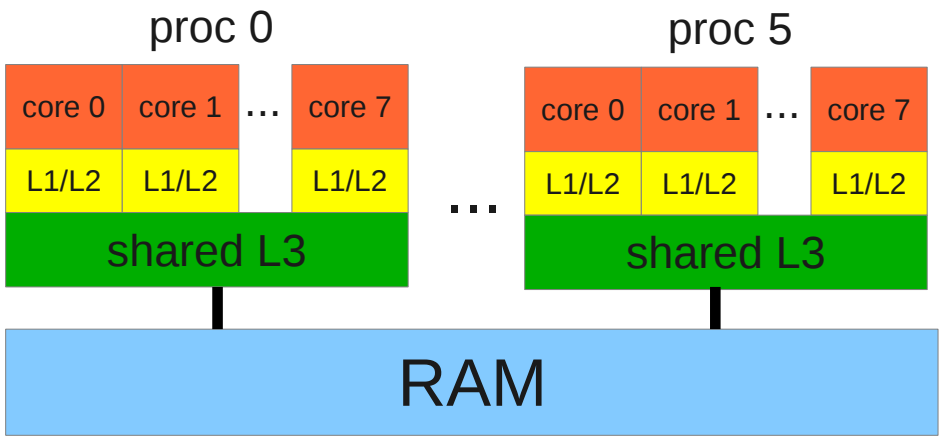


Processore IBM Power7:

- RISC architecture
- 45 nm technology, $1.2 \cdot 10^9$ transistors
- 8 Cores per processor
- 4 way SMT (4 threads/core)
- 32KB + 32 KB L1 cache/core
- 256 KB L2 cache/core
- 32 MB L3 cache/processor, on chip
- Memory controller on chip

Peak performance

6 cpus · 8 cores · 4 threads · 3.1 GHz · 2 vect = 1190.4 GFlops
4 cpus · 8 cores · 4 threads · 3.1 GHz · 2 vect = 793.6 GFlops



Memoria

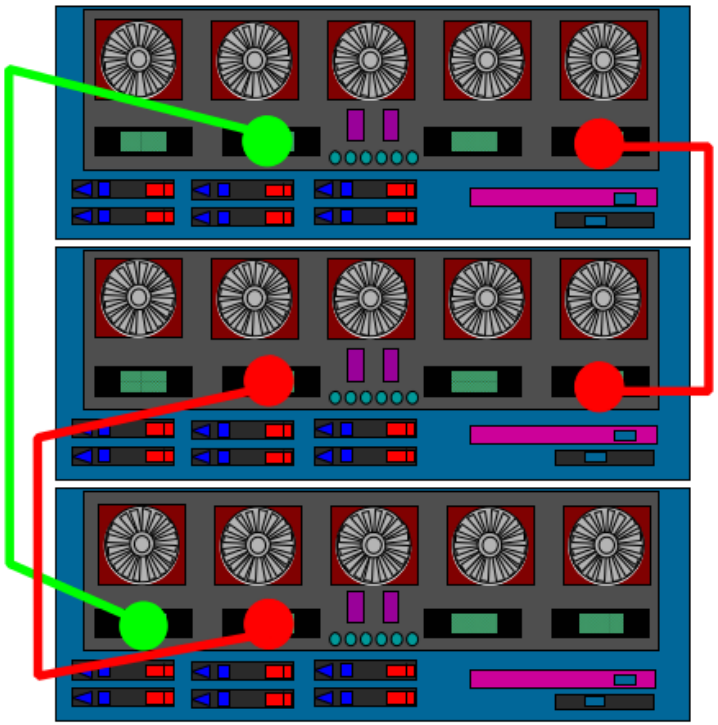
- Tutti i core hanno accesso a tutta la memoria.
- Un unico sistema operativo.
- I core sullo stesso chip condividono L3.

Nota

Due processi sullo stesso chip:

- possono comunicare velocemente per mezzo della cache;
- competono per l'uso della cache L3.

È possibile un certo controllo dell'allocazione dei processi ai core, ad esempio con LoadLeveler.



Interconnessione

- I drawers sono collegati punto a punto
- Cavi appositi, “unifica” bus di collegamento processori
- Macchina shared memory

Partizionabilità

- Processori possono essere allocati in unità di 1/10 di CPU
- RAM allocata in blocchi da 256 MB.

Compilazione

Compilatori standard:

- Compilatori GNU (`gcc`, `g++`, ...)
- Compilatori IBM (`xlc`, `xlC`, `xlc++`, ...)
- Script (`gxlc`, `gxlc++`, ...) che convertono parametri per compilatori GNU negli equivalenti per IBM (es `gxlc` chiama `xlc` a partire da parametri per `gcc`)

Compilatori MPI (vedi `man mpcc`):

- C: `mpcc`, `mpcc_r`
- C++: `mpCC`, `mpCC_r`

Note

- i compilatori IBM hanno anche la versione con suffisso `_r` per compilare programmi multi-thread.
- sono presenti compilatori GNU e IBM anche per Fortran

Opzioni comuni

Standard: `-c`, `-o` *nome*, `-I`*dir* `-L`*dir* `-l`*lib*

Ottimizzazione: `-O`*n* `-qarch` `-qtune` `-qcache` `...`

Debug: `-g` (vedi `gdb`)

Progetto

- Esistono molte altre opzioni per ottimizzare i programmi
- Trovare le migliori è parte del lavoro di progetto!
- Ad esempio, non è detto che `-O3` sia meglio che `-O2`!

Documentazione

Ricordatevi che da terminale potete consultare la documentazione di ciascun comando tramite `man comando`.

Modalità di esecuzione

Interattiva: Esecuzione immediata, per il debug; *no esecuzione parallela!*

Batch: Utilizza un sistema di gestione basato su code

Interattiva

- Il classico modo di eseguire comandi da console
- Esempio: `./helloworld`
- ATTENZIONE a non "dimenticarsi" processi!
- Comandi utili:
  - per visualizzare i vostri processi: `ps -u` *`utente`*
  - per terminare eventuali processi: `kill [-9]` *`pid`*
- Da usare *as least as possible*!

Cos'è?

È un sistema per la gestione dei **job**
job = esecuzione di un programma (seriale o MPI) su un sistema IBM

Perché si usa?

- Permette una migliore allocazione delle risorse di sistema
- Ottimizza l'esecuzione
- Suddivide il carico tra i processori
- Permette un utilizzo equo da parte di più utenti
- Permette l'esecuzione batch (non interattiva)

Quando si deve usare?

- Il più possibile, perché aumenta l'efficienza del sistema
- Non è adatto per fare debug

Esempio di job file - myjob.job

```
#! /bin/bash

#@ job_name         =  mm_naive_4_4096
#@ initialdir       =  /home/schimdmi/mm_naive
#@ input            =  /dev/null
#@ output           =  $(job_name).out
#@ error            =  $(job_name).err
#@ class            =  short
#@ job_type         =  parallel
#@ blocking         =  unlimited
#@ total_tasks      =  4
#@ queue

./mm_naive A4096.bin B4096.bin
```

Invio al LoadLeveler

```
llsubmit myjob.job
```

http://www.dei.unipd.it/~addetto/manuali_online/SP/LLUAdmin/lllv2mst85.html

- è un normale script Linux
- le righe che iniziano con `#@` sono direttive per il LoadLeveler
- configura il LoadLeveler per l'esecuzione del programma (processi da usare, file di input/output ecc...)

Note

initialdir	directory rispetto a cui si indicato i vari file
class	short ($< 2'$), medium ($< 20'$), long ($< 24h$), verylong ($< 15gg$). Vedi `llclass`.
job_type	parallel, serial
total_task	quanti processi sono avviati
input, output, error	reindirizza `stdin`, `stdout` e `stderr`

Semplice esempio di utilizzo:

- Creo uno script per ogni esperimento (`naive_1_1024.job`, ...)
- Li metto in coda al LoadLeveler

launcher.sh

```
#!/bin/bash

for LATO in 1024 2048 4096
do
   for NPROC in 1 2 4 8
   do
      llsubmit "naive_${NPROC}_${LATO}.job"
   done
done
```

- Analizzo i dati ottenuti (bisogna specificare output diversi, altrimenti verranno sovrascritti)

Permettono di interagire con LoadLeveler

Comandi principali

Nome	Descrizione
`llsubmit`	Sottomettere un job file per l'esecuzione di un programma
`llq`	Controllare lo stato di un job
`llcancel`	Cancellare un job precedentemente sottomesso
`llstatus`	Controllare lo stato della macchina
`llclass`	Ottenere la lista delle code di esecuzione

`http://www.dei.unipd.it/~addetto/manuali_online/SP/LLUAdmin/lllv2mst200.html`

Per automatizzare la compilazione dei sorgenti lo strumento standard è `make`. Noi analizzeremo la versione GNU di questo programma.

Il `Makefile`

In ogni directory che contiene sorgenti è necessario creare un file di testo, chiamato `Makefile`, che contiene le istruzioni per la compilazione.

Come avviare la compilazione

Una volta creato il Makefile è sufficiente lanciare il comando `make` (o `make obiettivo`) per avviare la compilazione.

Sintassi delle regole

```
obiettivo: prerequisiti
        comando1
        comando2
        ...
```

Spiegazione

obiettivo	Il file da creare
prerequisiti	I file che servono per crearlo
comando	I comandi (si noti il *tab* obbligatorio) per creare l'*obiettivo*

Cosa fa il make?

Controlla se qualcuno frai *prerequisiti* è stato modificato piú recentemente dell'*obiettivo*. In caso affermativo esegue i *comandi*.

Makefile

```
grosso.txt:  piccolo1.txt piccolo2.txt
        cat piccolo1.txt piccolo2.txt > grosso.txt
```

Il comando `make grosso.txt` controlla che il file *grosso.txt* non esista oppure sia piú vecchio dei file *piccolo1.txt* e *piccolo2.txt*. Quindi esegue il comando *cat* che crea il file *grosso.txt*.
Nel caso l'obiettivo sia già aggiornato make notifica la cosa con la seguente stringa:

```
make: 'grosso.txt' is up to date.
```

Spesso è comodo poter creare dei comandi svincolati da particolari file.
Si usano allora degli obiettivi *phony*.

Makefile

```
.PHONY: clean
clean:
        rm *.o
```

Spiegazione

Eseguire il comando `make clean` è equivalente ad eseguire `rm *.o`

Nei Makefile è possibile utilizzare delle variabili per evitare di riscrivere piú volte gli stessi comandi.

Makefile

```
CFLAGS = -Wall -W -pedantic -ansi

scan.o : scan.c
	gcc $(CFLAGS) -c scan.c

main.o : main.c
	gcc $(CFLAGS) -c main.c
```

Spesso si deve eseguire lo stesso comando su piú file.

Makefile

```
CFLAGS = -Wall -W -pedantic -ansi
%.o : %.c
	gcc $(CFLAGS) -c -o $@ $<
```

Regole implicite

La seconda riga fornisce una regola per trasformare qualunque file `.c` in un file `.o`

Variabili automatiche

$@	Obiettivo
$<	Primo prerequisito
$^	Tutti i prerequisiti

```
CXXFLAGS = -Wall -W -pedantic

%.o : %.cc
        g++ $(CXXFLAGS) -o $@ -c $<

runme :  test.o saluti.o
        g++ $(CXXFLAGS) -o $@ $^
```

Da consegnare

Vi ricordo che

- Parte del progetto è il codice che sviluppate.
- Il codice deve essere corredato di un makefile che mi permetta di compilarlo.
- Riguardate le slide della lezione precedente per i dettagli.