Il progetto COKA

programmazione parallela su MIC

F. Di Renzo

Università di Parma e I.N.F.N. Parma

Genova, 29 Maggio 2013

Workshop CCR 2013

Il progetto Coka si propone di indagare le capacità di calcolo di architetture innovative, in particolare la architettura many-core Intel MIC. Ci chiederemo quali attenzioni siano da prestare per una programmazione ragionevolmente efficiente di simili dispositivi.

Mentre sono possibili approcci che, mirando alla ricerca della massima efficienza, richiedono una completa ristrutturazione di codici eventualmente già esistenti e consolidati nel tempo, evidenzieremo come questa non sia l'unica soluzione possibile.

Sono praticabili metodologie di programmazione che cerchino un accettabile compromesso fra efficienza di calcolo e portabilità di codice esistente.

Anche per un simile approccio è comunque necessario avere presenti alcune linee guida ineludibili, che strutturano uno stile di programmazione un poco diverso da quello cui siamo abitutati su architetture non parallele.

Segnatamente parleremo di ...

- PILLOLE di CALCOLO PARALLELO

- La filosofia della architettura MIC

- PILLOLE di STRATEGIA per il CALCOLO PARALLELO

  Non entreremo nei dettagli di approcci “aggressivi”:
  guarderemo a volo d’uccello alcune linee per uno stile di programmazione

- STRUMENTI per PROGRAMMARE in modo sostanzialmente semplice

  Un benchmark elementare (operazioni matriciali)

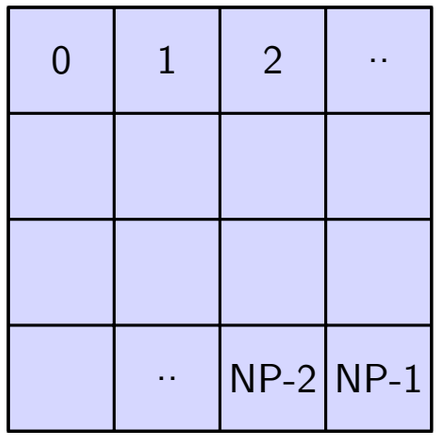
PILLOLE di CALCOLO PARALLELO

Un GRANDE problema può essere calcolato in PARALLELO...

0	1	2	..
	..	NP-2	NP-1

Ho ripartito un conto su NP nodi di calcolo; può essere che questo sia banale e non abbia altro di cui preoccuparmi ...

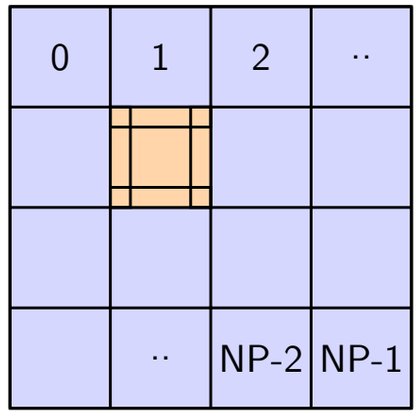
Un GRANDE problema può essere calcolato in PARALLELO...

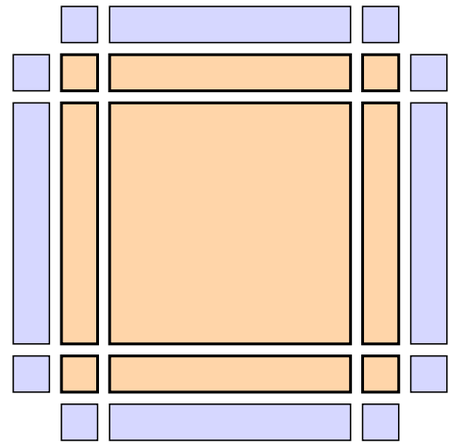


Ho ripartito un conto su NP nodi di calcolo; può essere che questo sia banale e non abbia altro di cui preoccuparmi ...

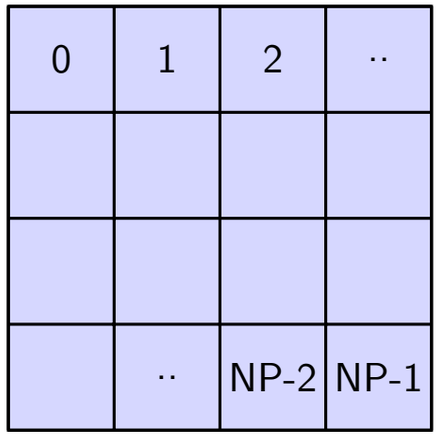
... o ci sono dati dipendenti/condivisi fra i nodi;

se chiedete a me, vi dirò che sto dividendo un reticolo in *sottoreticoli*



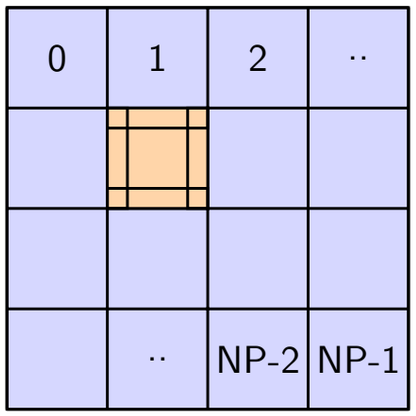


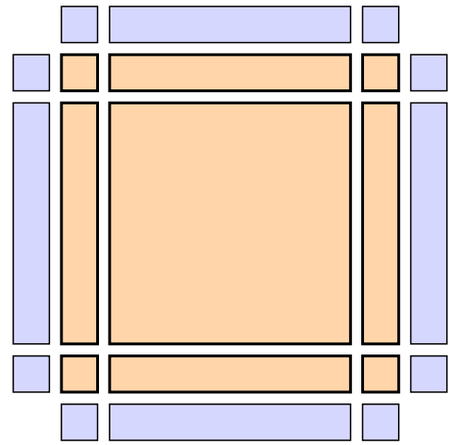
Un GRANDE problema può essere calcolato in PARALLELO...

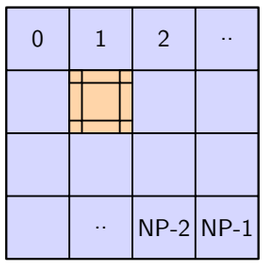


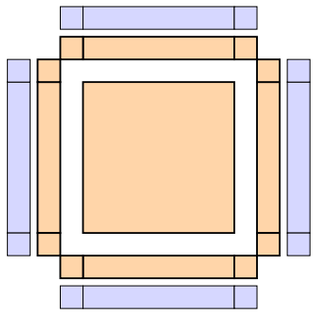
Ho ripartito un conto su NP nodi di calcolo; può essere che questo sia banale e non abbia altro di cui preoccuparmi ...

... o ci sono dati dipendenti/condivisi fra i nodi;
se chiedete a me, vi dirò che sto dividendo un reticolo in *sottoreticoli*



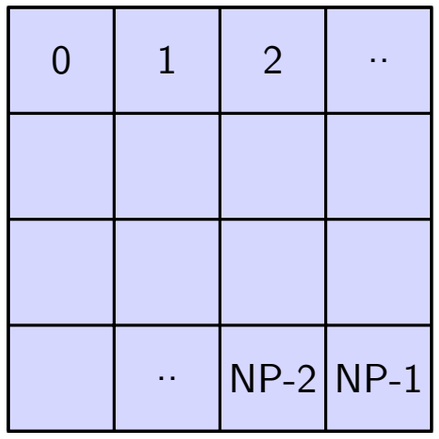






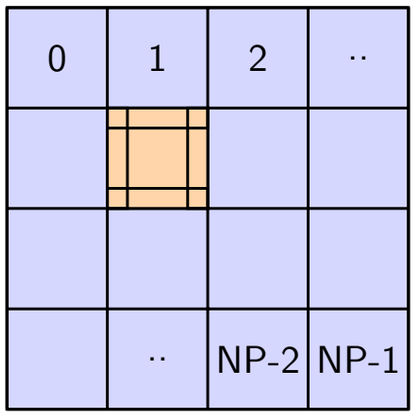
Ho segnato dei bordi (problema semplice perchè locale): un nodo deve accedere a dati su bordi altrui e rendere disponibili i dati sui bordi propri. I nodi condividono la MEMORIA?

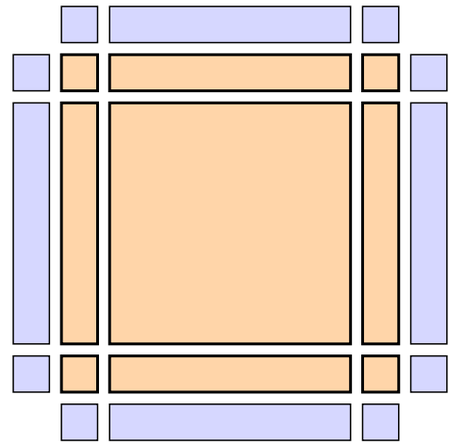
Un GRANDE problema può essere calcolato in PARALLELO...

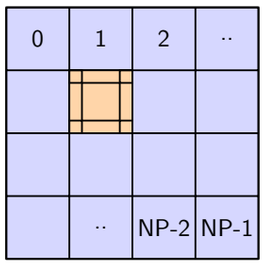


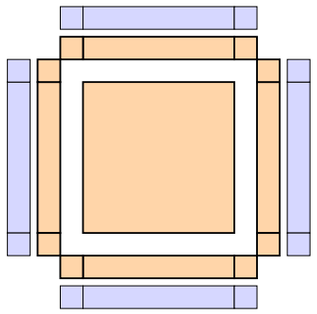
Ho ripartito un conto su NP nodi di calcolo; può essere che questo sia banale e non abbia altro di cui preoccuparmi ...

... o ci sono dati dipendenti/condivisi fra i nodi;
se chiedete a me, vi dirò che sto dividendo un reticolo in *sottoreticoli*





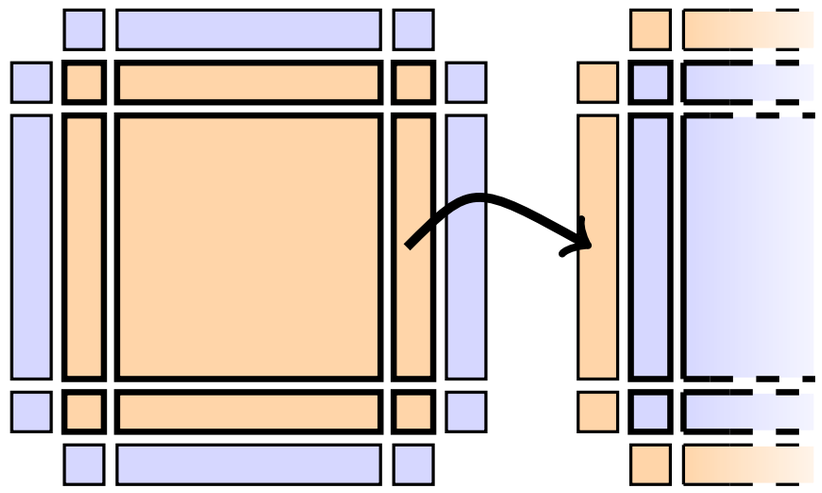




Ho segnato dei bordi (problema semplice perchè locale): un nodo deve accedere a dati su bordi altrui e rendere disponibili i dati sui bordi propri. I nodi condividono la MEMORIA?

In ogni caso ho un problema di sincronizzazione!
Tipicamente, avrò spesso un problema di comunicazione.

Da quest'ultimo punto di vista, il trucco ultimo è cercare di mascherare le comunicazioni con i calcoli...



La formula più importante del calcolo parallelo...

Quanto ci metto a fare il mio calcolo?

$$T = \max\{\frac{C}{NP}, \frac{I}{NB}, \frac{I_R}{B_R}\} = \frac{C}{NP}\max\{1, \frac{IP}{CB}, \frac{I_R NP}{CB_R}\}$$

- C/N computational Cost per Node
- P computational Performance
- I/N local exchange of Information per Node
- B memory Bandwith
- I_R Remote exchange of Information
- B_R corresponding (remote) bandwith.

Nota: ho assunto di avere tanti nodi di calcolo, cioè una risorsa parallela.

Problema: cosa scelgo? Qual è la risorsa di calcolo parallela su cui calcolo?

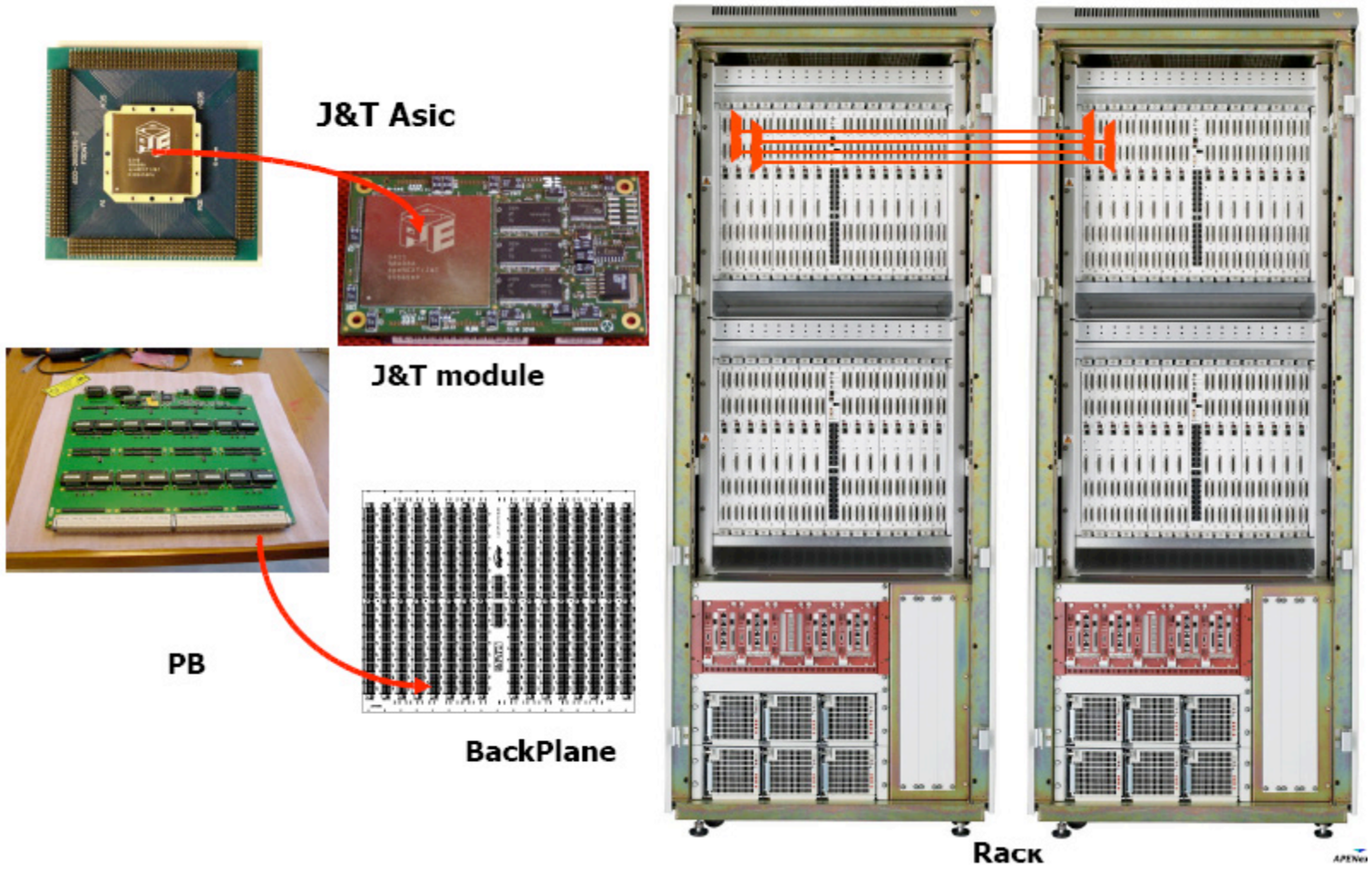
Una buona notizia: ci sono in giro tantissime risorse di calcolo parallele!

Una cattiva (?) notizia: devo darmi uno stile di programmazione parallela...

La filosofia della architettura MIC

(Una gloria di famiglia ... qualche anno fa ...)

apeNEXT



NB Notare la struttura modulare ...

(Abbiamo fatto scuola ... ma le cose sono un po' cambiate)

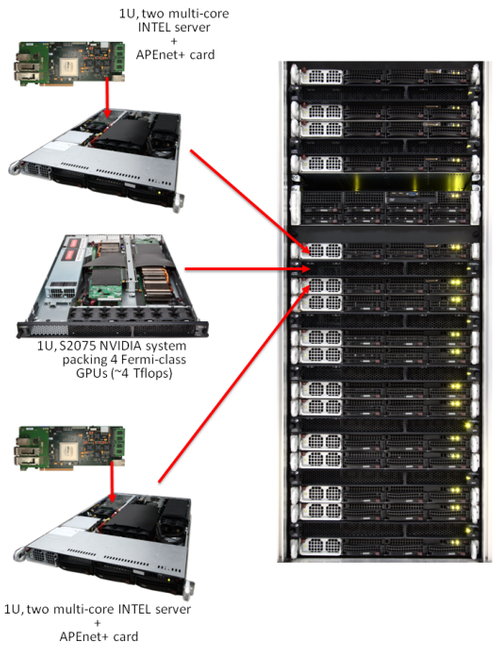
Guardando dentro una BG/Q si troverebbe una struttura modulare, con componenti più standard...

CINECA

NVIDIA TESLA

EUROTECH

NB Notare qui le componenti: GPU



Una occhiata a TOP500 ...

Scorrendo la classifica di TOP500, noteremmo molte architetture composite.

Attualmente le componenti di grandi sistemi paralleli sono di vario tipo:

- Innanzitutto, molti nodi sono ora sostanzialmente vicini ai sistemi che abbiamo sotto mano tutti i giorni (desktop, laptot), i.e. nodi MULTICORE

- Sono tornati di moda degli ACCELERATORI (number-crunching...), tipicamente GPU.

- In ogni caso, la gerarchia del parallelismo si sta facendo molto composita: il nodo è a sua volta una struttura parallela.

Home / Lists / November 2012

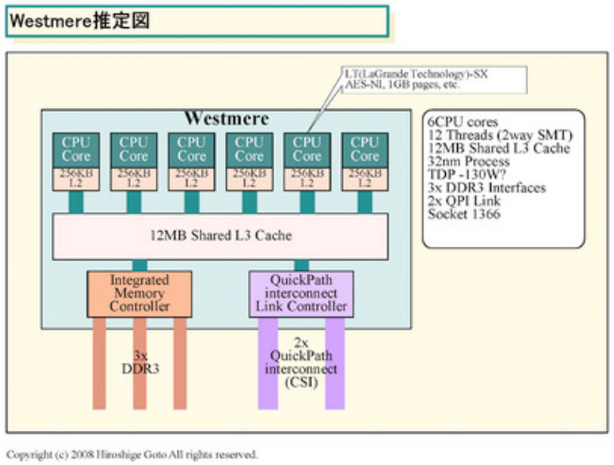
Top500 List - November 2012

R_{max} and R_{peak} values are in TFlops. For more details about other fields, check the TOP500 description.

previous | 1 | 2 | 3 | 4 | 5 | next

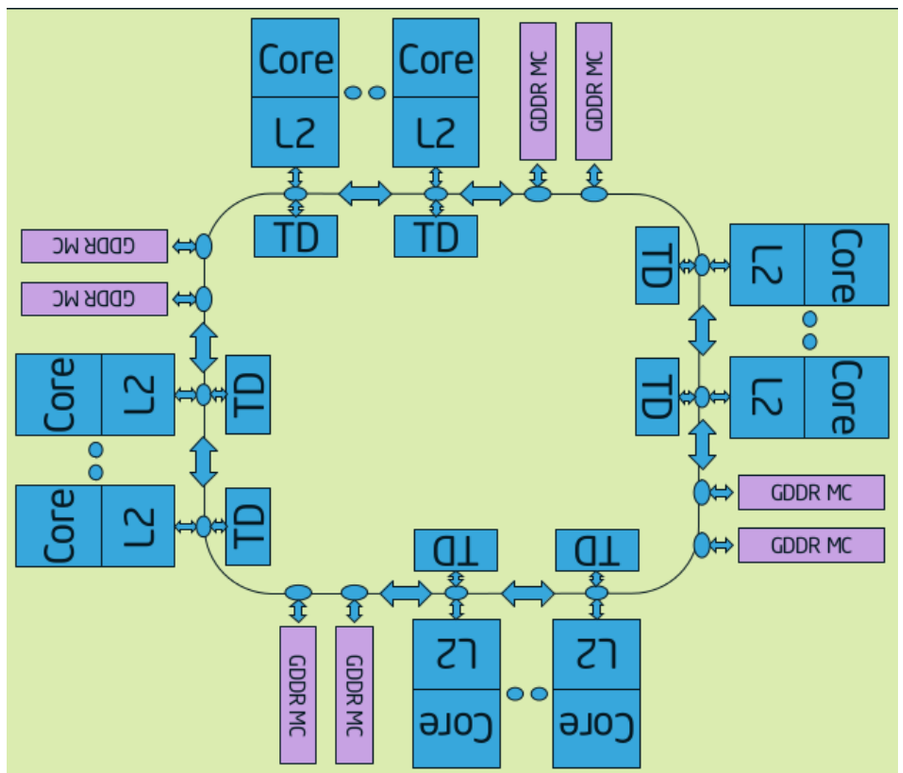
Rank	Site	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	DOE/SC/Oak Ridge National Laboratory United States	**Titan** - Cray XK7 , Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x Cray Inc.	560640	17590.0	27112.5	8209
2	DOE/NNSA/LLNL United States	**Sequoia** - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom IBM	1572864	16324.8	20132.7	7890
3	RIKEN Advanced Institute for Computational Science (AICS) Japan	K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect Fujitsu	705024	10510.0	11280.4	12660
4	DOE/SC/Argonne National Laboratory United States	**Mira** - BlueGene/Q, Power BQC 16C 1.60GHz, Custom IBM	786432	8162.4	10066.3	3945
5	Forschungszentrum Juelich (FZJ) Germany	**JUQUEEN** - BlueGene/Q, Power BQC 16C 1.600GHz, Custom Interconnect IBM	393216	4141.2	5033.2	1970
6	Leibniz Rechenzentrum Germany	**SuperMUC** - iDataPlex DX360M4, Xeon E5-2680 8C 2.70GHz, Infiniband FDR IBM	147456	2897.0	3185.1	3423
7	Texas Advanced Computing Center/Univ. of Texas United States	**Stampede** - PowerEdge C8220, Xeon E5-2680 8C 2.700GHz, Infiniband FDR, Intel Xeon Phi Dell	204900	2660.3	3959.0	
8	National Supercomputing Center in Tianjin China	**Tianhe-1A** - NUDT YH MPP, Xeon X5670 6C 2.93 GHz, NVIDIA 2050 NUDT	186368	2566.0	4701.0	4040
9	CINECA Italy	**Fermi** - BlueGene/Q, Power BQC 16C 1.60GHz, Custom IBM	163840	1725.5	2097.2	822
10	IBM Development Engineering United States	**DARPA Trial Subset** - Power 775, POWER7 8C 3.836GHz, Custom Interconnect IBM	63360	1515.0	1944.4	3576

MULTICORE <-- MIC (MANYCORE) --> GPU



Tipico processore multicore (Westmere) O(10) cores

Ciò che Nvidia diceva delle GPU per il calcolo: poca logica, molta potenza FPU!



Intel MIC

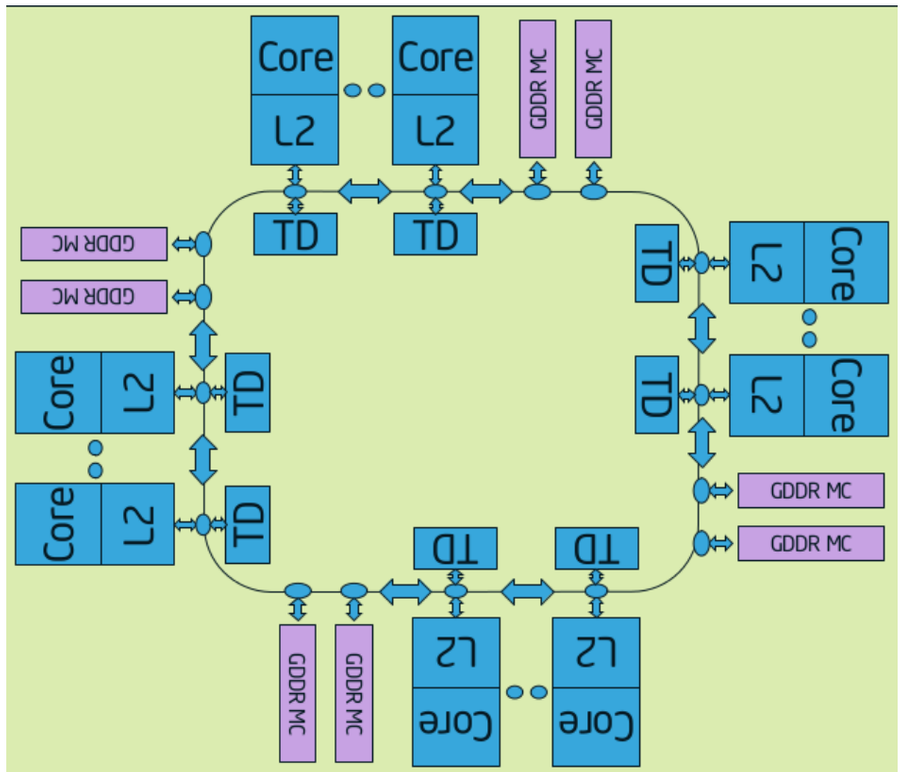
o (nome commerciale) Phi

Un ANELLO di O(100) cores.

Il MANYCORE interpola fra multicore e GPU.

NB: il MIC potrà essere pensato come NODO o come RISORSA PARALLELA esso stesso...

Intel Phi (MIC) (segue...)



Intel MIC

o (nome commerciale) Phi

Un ANELLO di O(100) cores.

Il MANYCORE interpola fra multicore e GPU.

NB: il MIC potrà essere pensato come NODO o come RISORSA PARALLELA esso stesso...

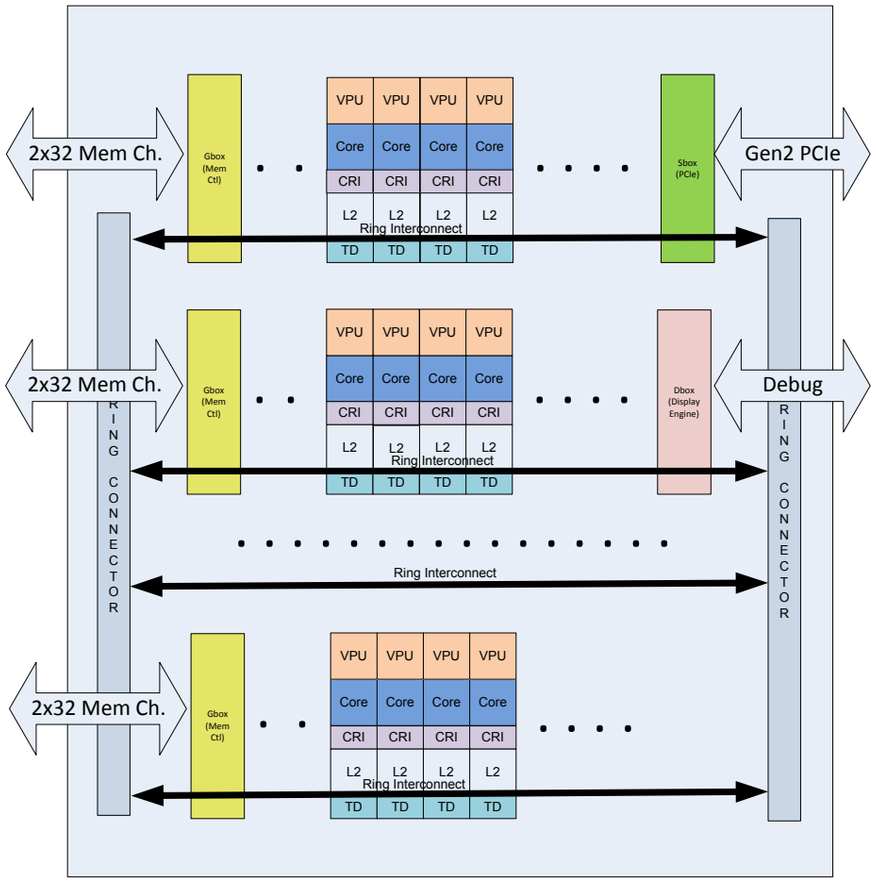
Qualche dettaglio in più sulla architettura

The Intel® Xeon PhiTM coprocessor comprises of up to sixty-one (61) processor cores connected by a high performance on-die bidirectional interconnect.

Each core is a fully functional, in-order core, which supports fetch and decode instructions from four hardware thread execution contexts. 512 bit wide vector processor unit (VPU)

The Core-Ring Interface hosts the 512KB, 8-way, L2 cache and connects each core to an Intel® Xeon PhiTM coprocessor Ring Stop. Primarily, it comprises the core-private L2 cache itself plus all of the off-core transaction tracking queues and transaction / data routing logic.

Each memory controller is based on the GDDR5 specification, and supports two channels per memory controller. At up to 5.5 GT/s transfer speed, this provides a theoretical aggregate bandwidth of 352 GB/s

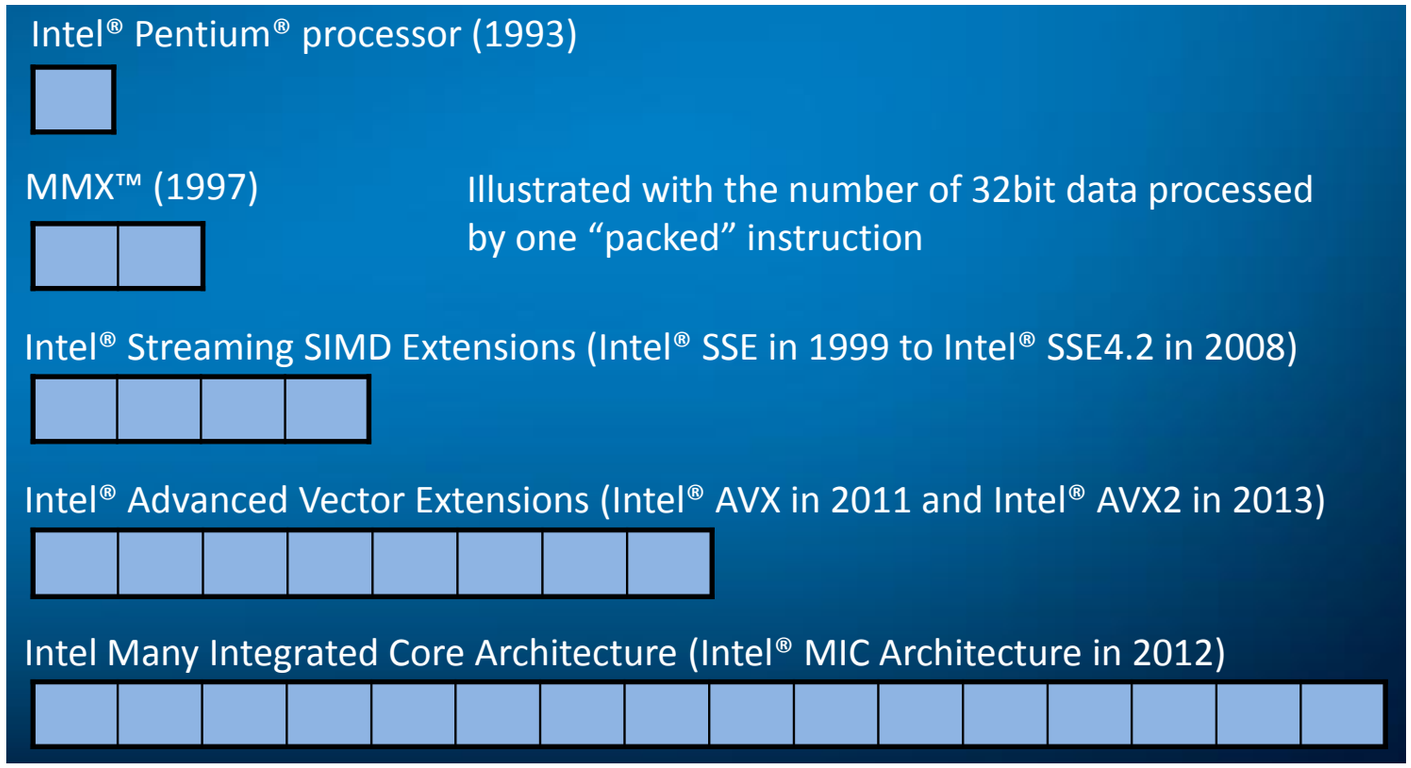


PILLOLE di STRATEGIA per il CALCOLO PARALLELO

quello che abbiamo imparato sui multicore
e che vorremmo mettere a frutto in generale (anche sul MIC)

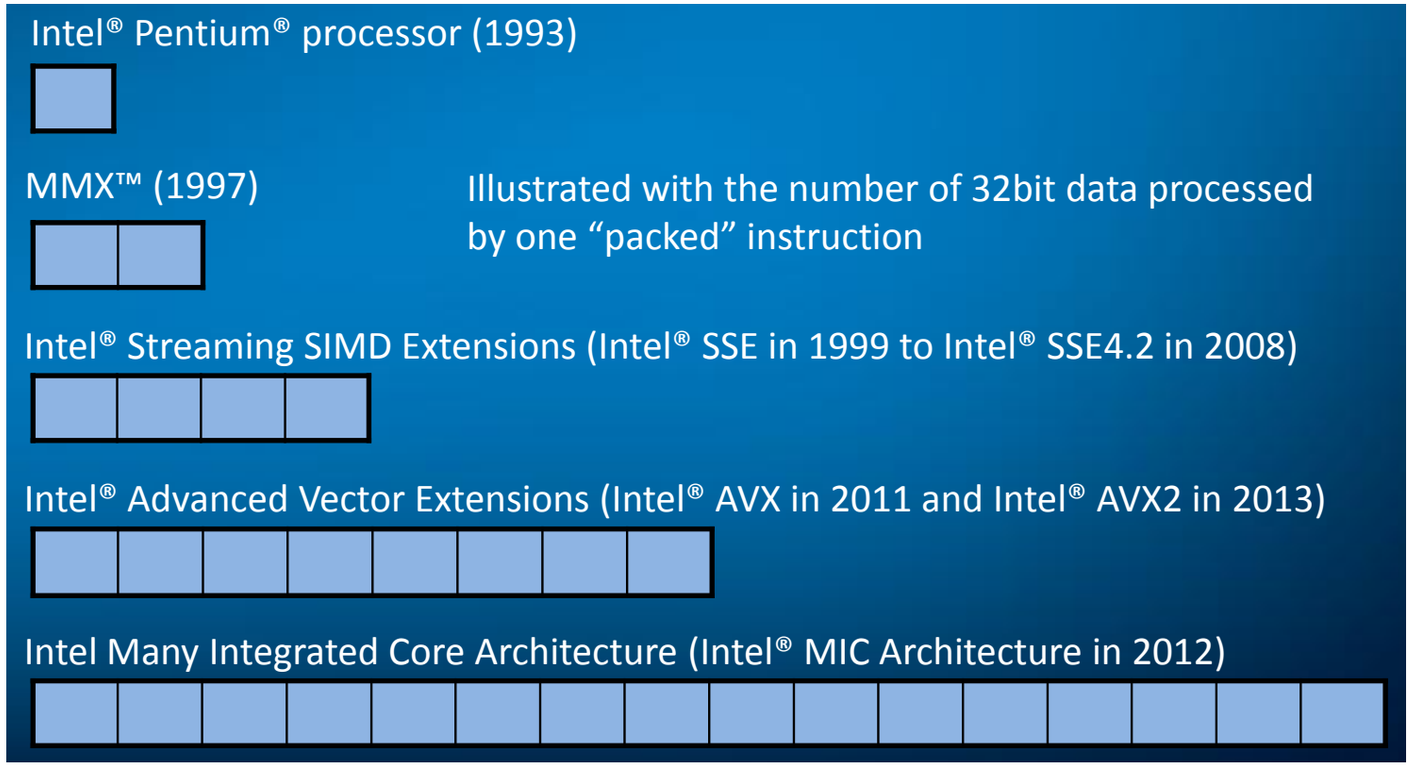
Quello che abbiamo imparato su MULTICORE deve valere ancora...(1)

La gerarchia del parallelismo arriva fino al nodo: i core hanno POTENZA FPU VETTORIALE!

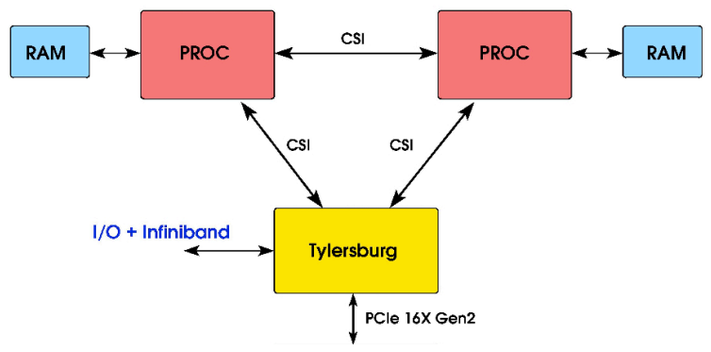


Quello che abbiamo imparato su MULTICORE deve valere ancora...(1)

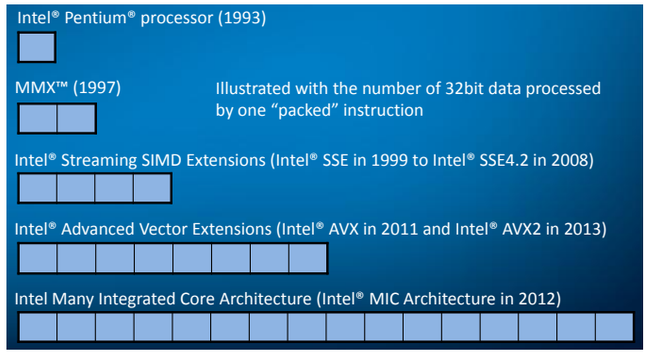
La gerarchia del parallelismo arriva fino al nodo: i core hanno POTENZA FPU VETTORIALE!



Notare che i MULTICORE sono (quasi sempre) montati in sistemi SMP: (bi)(quadri)processori...



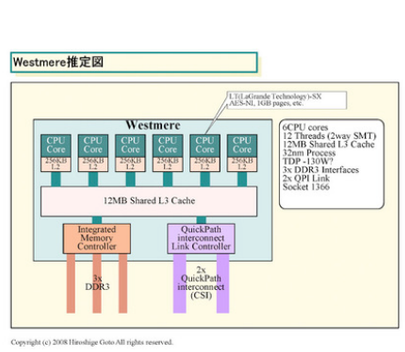
Quello che abbiamo imparato su MULTICORE deve valere ancora...(2)

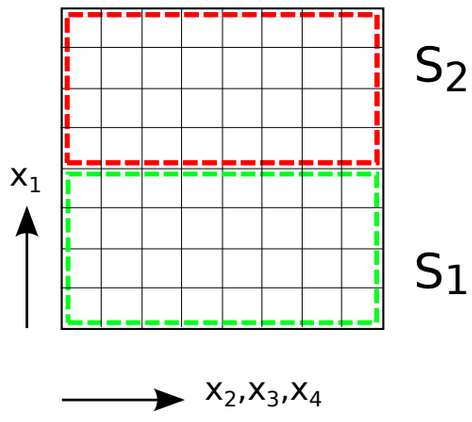


Il carattere vettoriale della potenza FPU è il dato cruciale

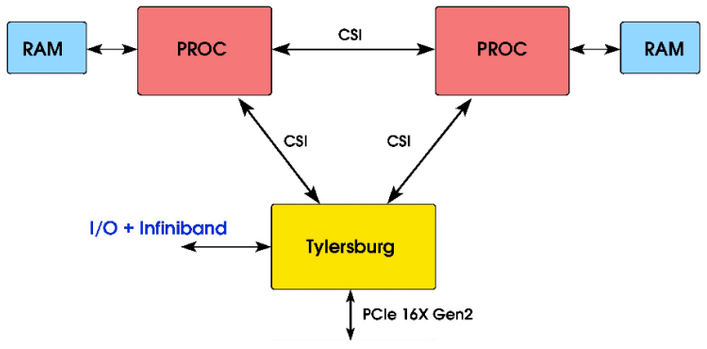
- SSE intrinsics
- MPI/multithread:
  1 rank per node + (HT) *Pthread*
- optimization of L1/L2/L3 usage
- core affinity and memory binding

```
__m128d x2 = _mm_mul_pd(R2,(v+i)->whr[0].m);
v2 = _mm_addsub_pd(_mm_shuffle_pd(x2,x2,1), ...
```





Il layout dei dati deve conformarsi il più possibile alla architettura di memoria



STRUMENTI per PROGRAMMARE
in modo sostanzialmente semplice

Un benchmark elementare (operazioni matriciali)

La domanda ultima...

Abbiamo dato (in modo stringatissimo...) qualche lineamento di strategie per il calcolo parallelo. Ripensandoci, abbiamo detto (forse solo alluso a ...) essenzialmente due cose importanti:

- A livello del singolo core vogliamo VETTORIZZARE quanto più conformemente alla architettura che abbiamo a disposizione.

- A livello multicore, vogliamo rendere efficiente la esecuzione multithreading.

La domanda ultima...

Abbiamo dato (in modo stringatissimo...) qualche lineamento di strategie per il calcolo parallelo. Ripensandoci, abbiamo detto (forse solo alluso a ...) essenzialmente due cose importanti:

- A livello del singolo core vogliamo VETTORIZZARE quanto più conformemente alla architettura che abbiamo a disposizione.

- A livello multicore, vogliamo rendere efficiente la esecuzione multithreading.

Su che strumenti possiamo contare per avere PORTABILITA‘ e per

(a) ottimizzare il mio codice (i.e. VETTORIZZARE) in fase di compilazione?

(b) realizzare (run-time) il multithreading in modo semplice?

La domanda ultima...

Abbiamo dato (in modo stringatissimo...) qualche lineamento di strategie per il calcolo parallelo. Ripensandoci, abbiamo detto (forse solo alluso a ...) essenzialmente due cose importanti:

- A livello del singolo core vogliamo VETTORIZZARE quanto più conformemente alla architettura che abbiamo a disposizione.

- A livello multicore, vogliamo rendere efficiente la esecuzione multithreading.

Su che strumenti possiamo contare per avere PORTABILITA‘ e per

(a) ottimizzare il mio codice (i.e. VETTORIZZARE) in fase di compilazione?

(b) realizzare (run-time) il multithreading in modo semplice?

Vediamo all’opera su un semplicissimo benchmark (operazioni matriciali)

(a) ARRAY NOTATION, disponibile in ambiente di compilazione Intel (icc)

(b) Direttive OpenMP (ormai uno standard) e Cilk (disponibile in ambiente icc)

Risorse compilation-time e run-time

Abbiamo parlato di (sono strumenti che assicurano PORTABILITA!)

(a) ARRAY NOTATION, disponibile in ambiente di compilazione Intel (icc)

(b) OpenMP (ormai uno standard) e Cilk (disponibile in ambiente icc)

(a) ARRAY NOTATION:

Il compilatore è avvertito che qualcosa è SIMD

- **Array Notation**
  - A[0:N] = B[0:N] + C[0:N];

- **Open Mp**
  - #pragma omp parallel for
    for(i=0; i<N; i++)
    { A[i] = B[i] + C[i]; }
- **Cilk**
  - Cilk_for(i=0; i<N; i++)
    { A[i] = B[i] + C[i]; }

(b) OpenMP e Cilk, l'esempio più semplice:

In esecuzione il carico è distribuito fra più cores
(e se non dico niente sono davvero quanti piu' ne ha)

Qualcosa di più su ARRAY NOTATION (facciamolo dire ad Intel)

Operations on Array Sections

- **C/C++ operators**

```
d[:] = a[:] + (b[:] * c[:])
```

- **Function calls**

```
b[:] = foo(a[:]); // Call foo() on each element of a[]
```

- **Reductions** combine array elements to get a single result

```
// Add all elements of a[]
sum = __sec_reduce_add(a[:]);
// More reductions exist...
```

- **If-then-else and conditional operators** allow masked operations

```
if (mask[:]) {
  a[:] = b[:];  // If mask[i] is true, a[i]=b[i]
}
```

```
a[0:5] = b[0:6]; // No. Size mismatch.
a[0:5][0:4] = b[0:5]; // No. Rank mismatch.
a[0:5] = b[0:5][0:5]; // No. No 2D->1D reduction.
a[0:4] = 5; // OK. 4 elements of A filled w/ 5.
a[0:4] = b[i]; // OK. Fill with scalar b[i].
a[10][0:4] = b[1:4]; // OK. Both are 1D sections.
b[i] = a[0:4]; // No. Use reduction intrinsic.
```

```
float dot_product(unsigned int size, float A[size], float B[size])
{
    return __sec_reduce_add(A[:] * B[:]);
}
```

Ovviamente, posso combinare ARRAY NOTATION e OpenMP o Cilk.

Diciamo che è la tipica combinazione da cui mi aspetto prestazioni ragionevoli ... a basso costo

Velocemente: benchmark su SOMME e MOLTIPLICAZIONI di matrici

2 sfidanti per il MIC

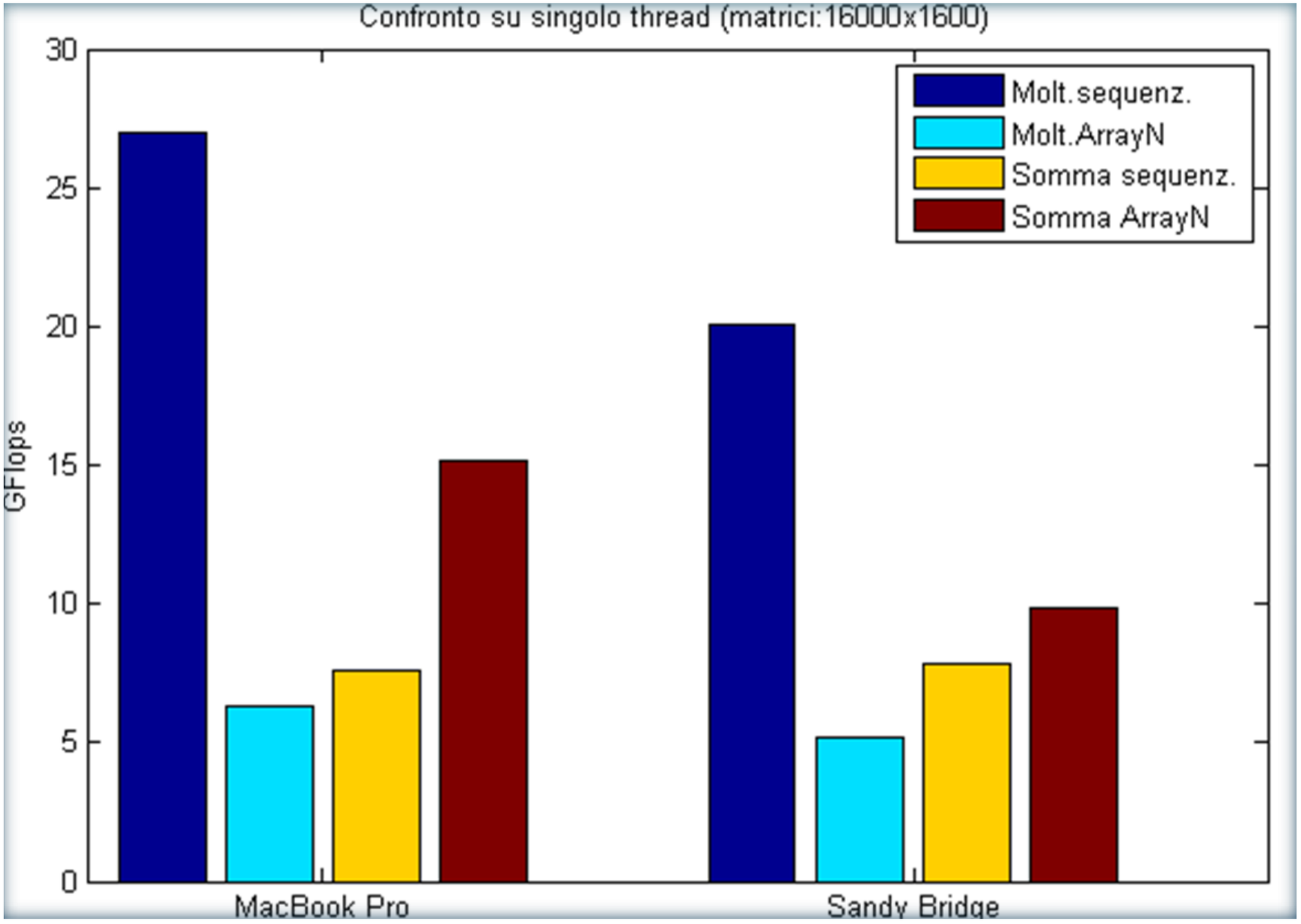
- **Multi Core** : N° core ordine 10
  - Intel I7 (MacBook Pro):
    - 4 core/2.5GHz/8 MB cache L3
    - Istruzioni SIMD a 256 bit (AVX)
  - Intel Sandy Bridge S5:
    - 2 processori/8core/2.6GHz/20 MB cache L3
    - Istruzioni SIMD a 256 bit (AVX)

Un esempio di codice

```
tick_start = cilk_getticks();
cilk_for (int i2 = 0; i2 < iter; ++i2)
for (int i = 0; i < iter; ++i)
for (int q = 0; q < Q; ++q)
        {
        C3[i][i2]+= __sec_reduce_add(( A[i][q*N:N]*B[i2][q*N:N]));
        }
tick_end = cilk_getticks();
```

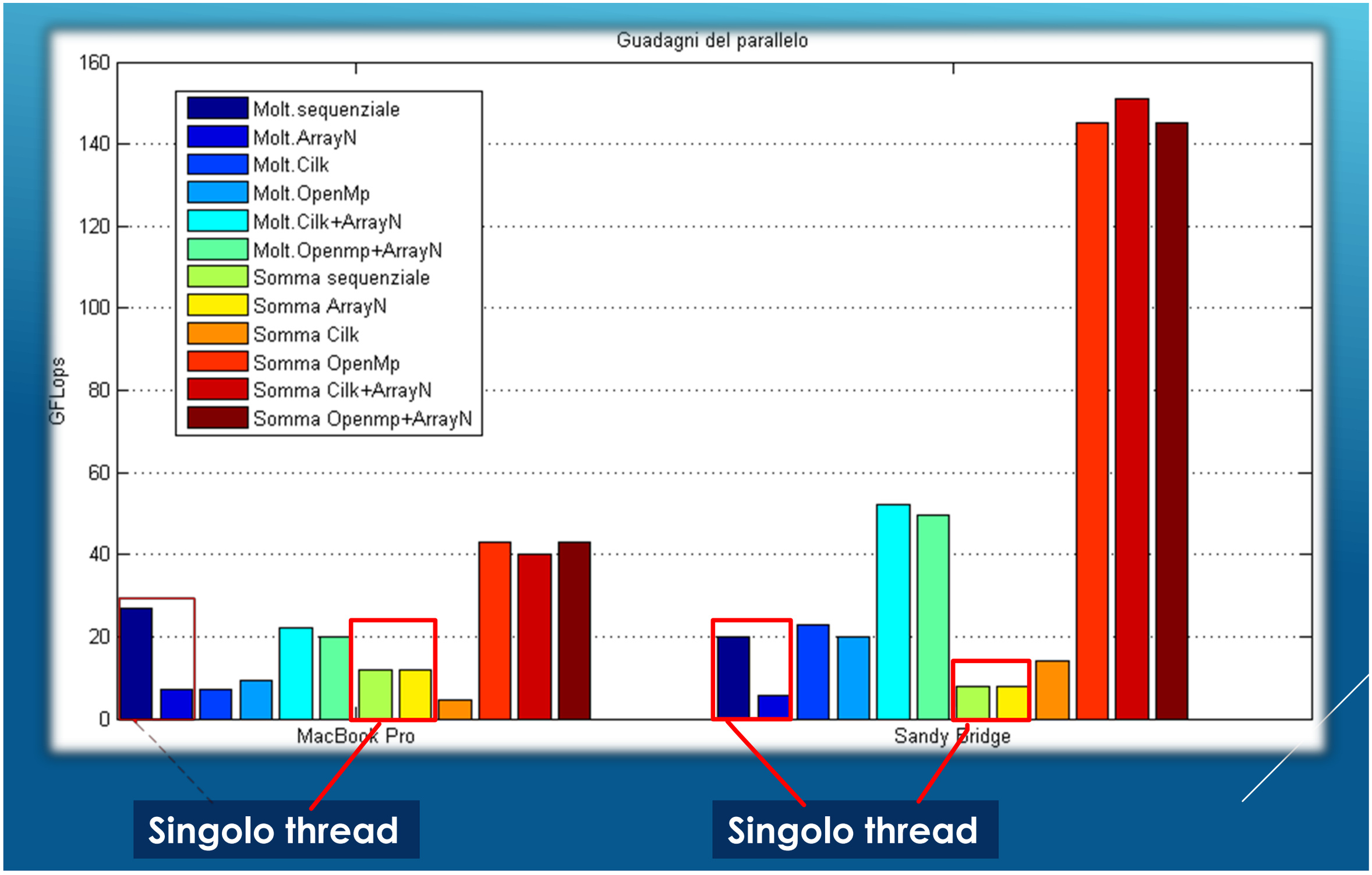
RISULTATI: un portatile non è poi così male...

... soprattutto se avete una versione (più) recente del compilatore,
i.e. il compilatore può fare spesso bene i(l più de)l lavoro!



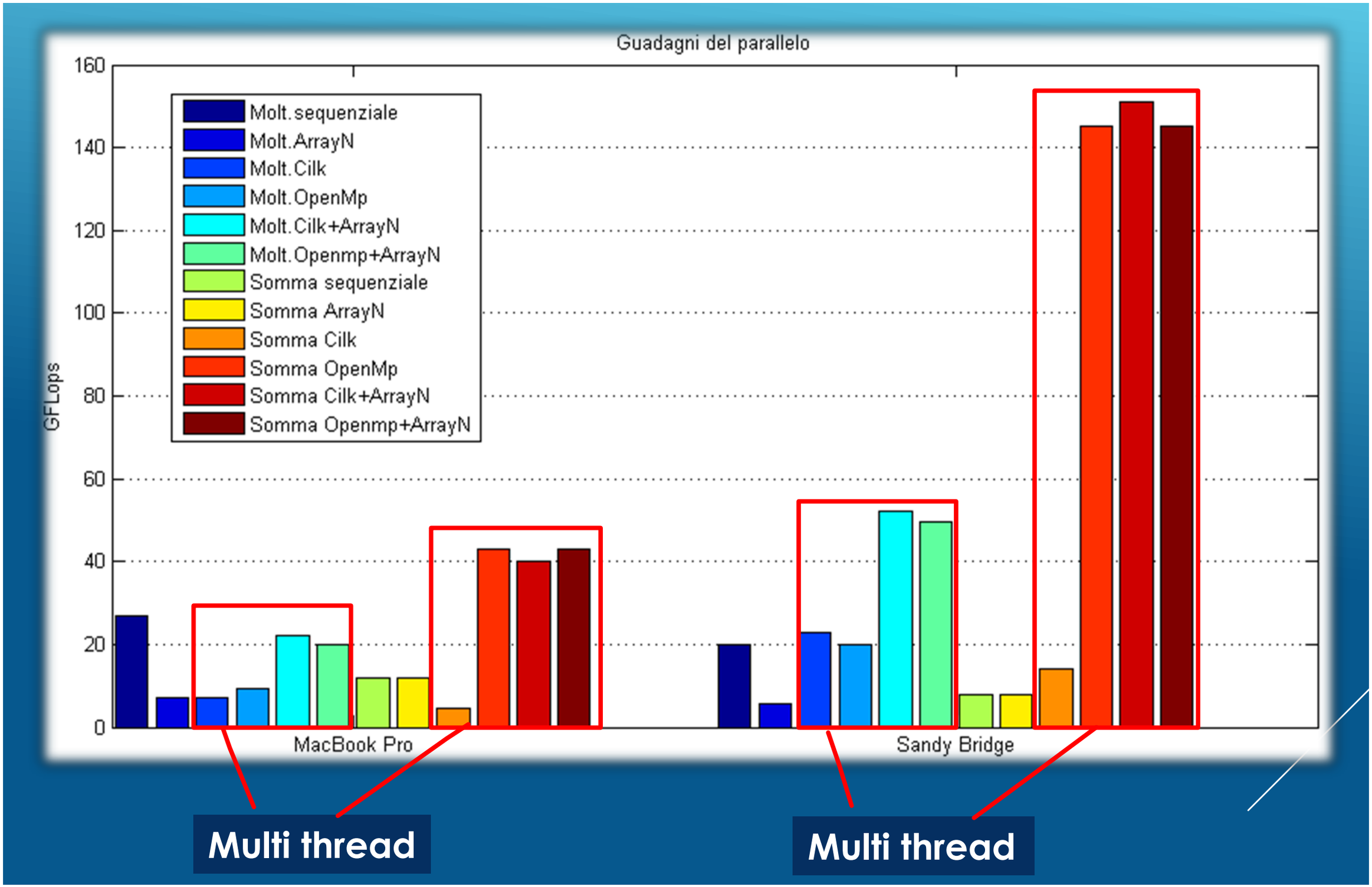
RISULTATI: ARRAY NOTATION e OpenMP/Cilk

caveat per Cilk!



RISULTATI: ARRAY NOTATION e OpenMP/Cilk

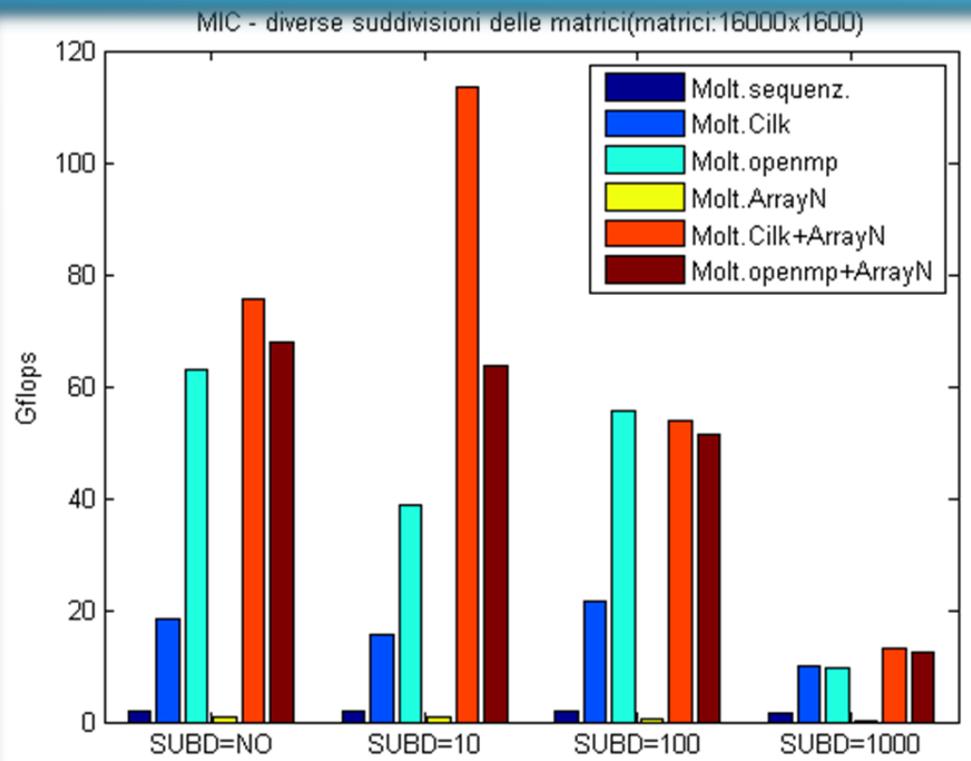
caveat per Cilk!



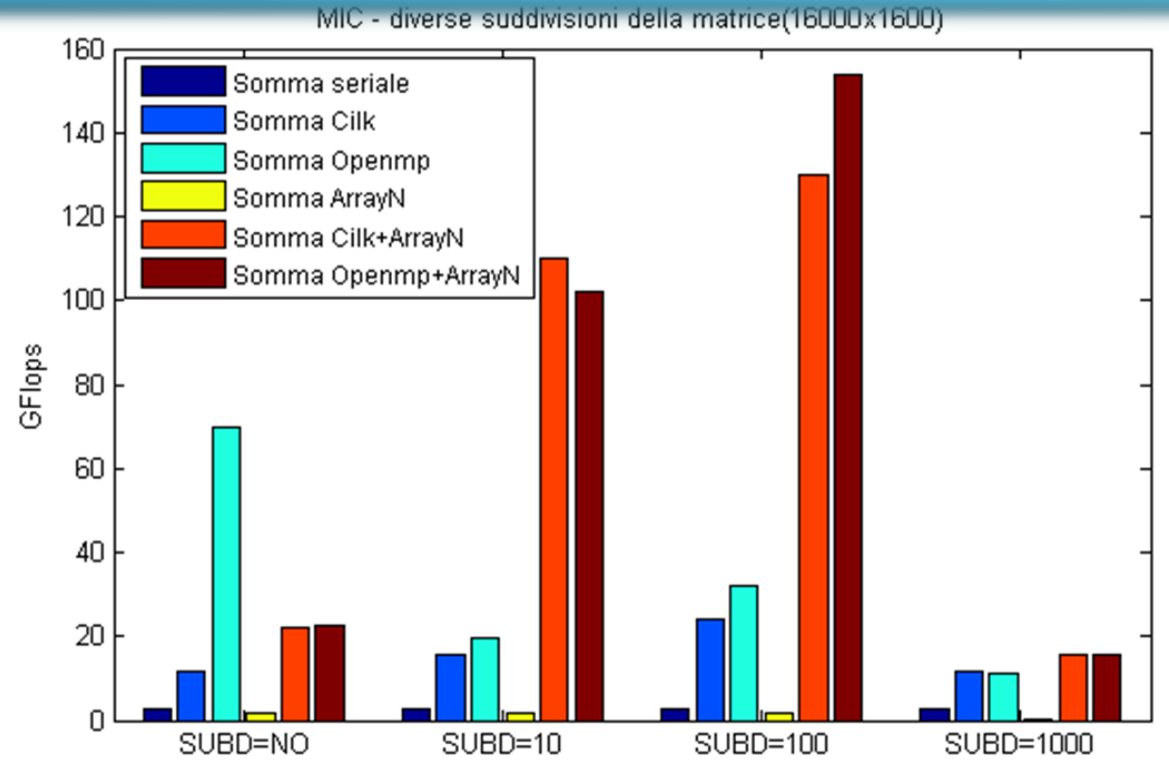
RISULTATI MIC

RISULTATI: MIC

Moltiplicazioni

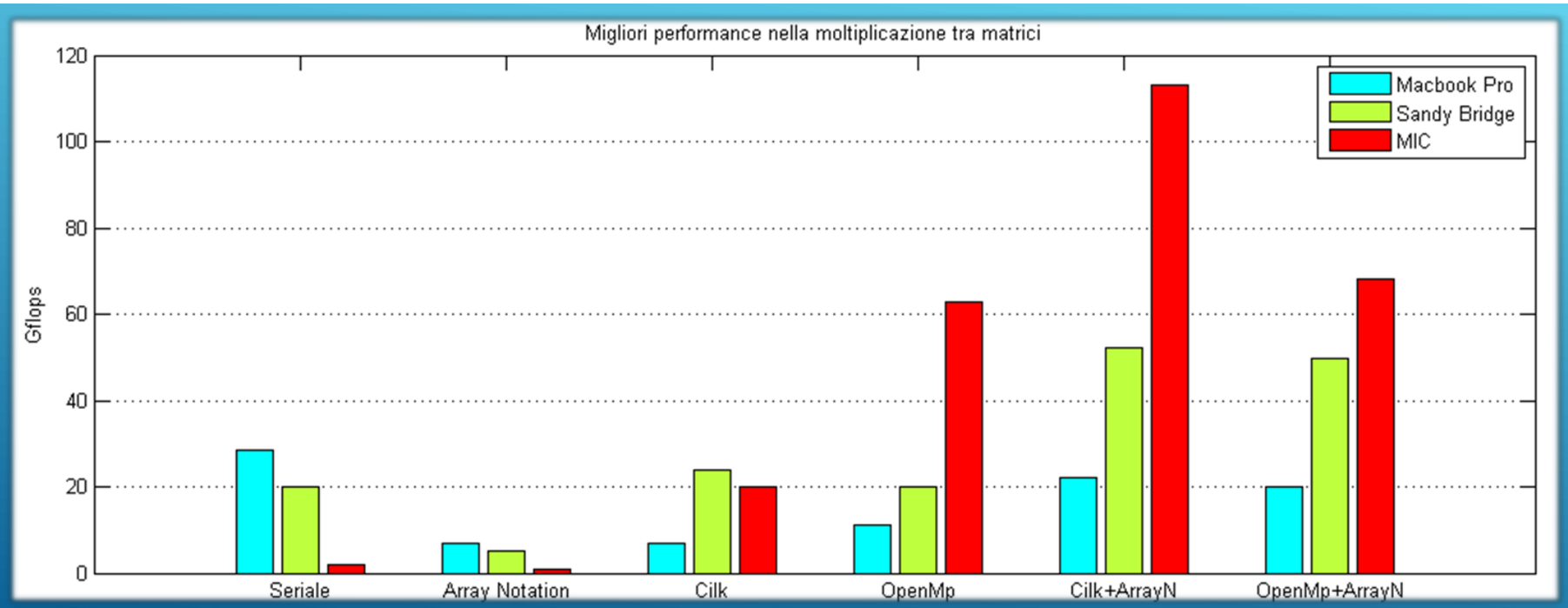


Somme

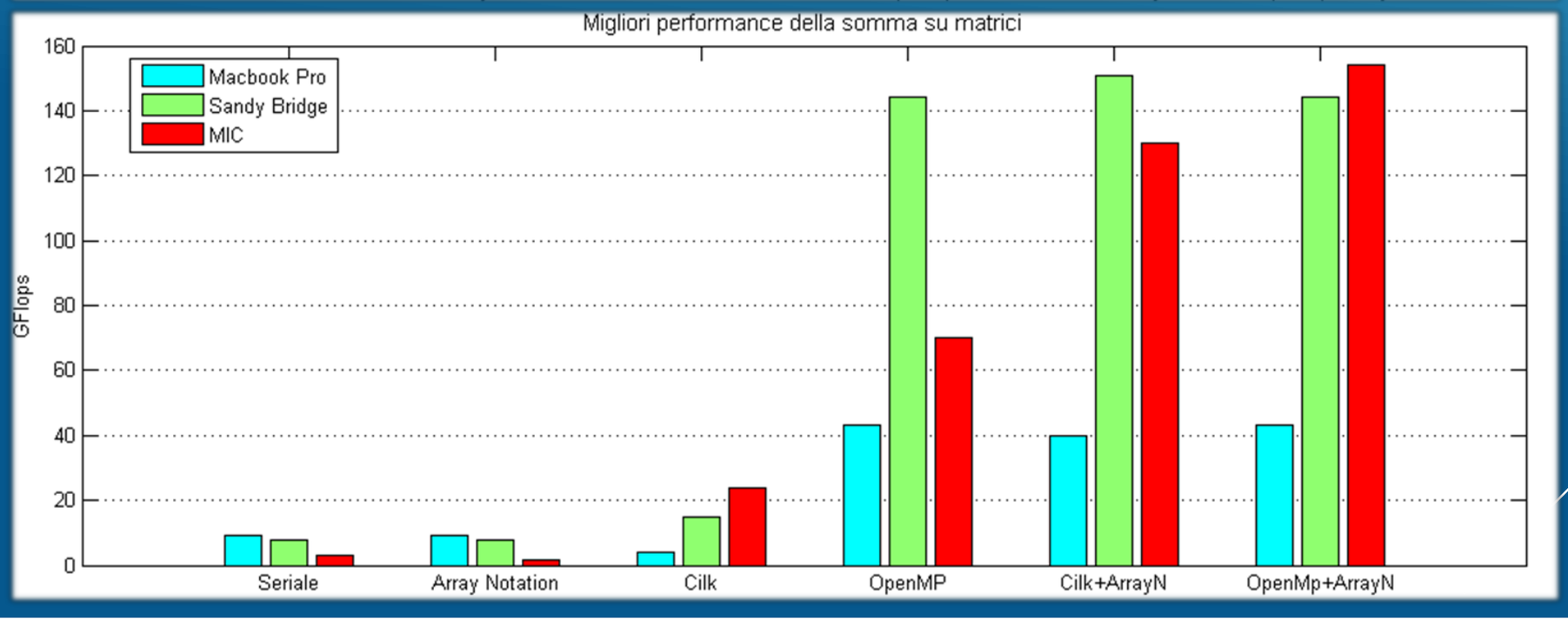


RISULTATI: MIC vs i suoi sfidanti...

Moltiplicazioni



Somme



Qualche conclusione

Le strategie di parallelizzazione possono essere sottili e il lavoro da loro richiesto pesante (riscrivere codice...). E' però vero che in generale possiamo pensare di ottenere risultati ragionevoli se ricordiamo sempre che:

- A livello del singolo core vogliamo VETTORIZZARE quanto più conformemente alla architettura che abbiamo a disposizione.

- A livello multicore, vogliamo rendere efficiente la esecuzione multithreading.

Qualche conclusione

Le strategie di parallelizzazione possono essere sottili e il lavoro da loro richiesto pesante (riscrivere codice...). E' però vero che in generale possiamo pensare di ottenere risultati ragionevoli se ricordiamo sempre che:

- A livello del singolo core vogliamo VETTORIZZARE quanto più conformemente alla architettura che abbiamo a disposizione.

- A livello multicore, vogliamo rendere efficiente la esecuzione multithreading.

Abbiamo visto all'opera su un semplicissimo benchmark (operazioni matriciali)

(a) ARRAY NOTATION, disponibile in ambiente di compilazione Intel (icc)

(b) Direttive OpenMP (ormai uno standard) e Cilk (disponibile in ambiente icc)

Qualche conclusione

Le strategie di parallelizzazione possono essere sottili e il lavoro da loro richiesto pesante (riscrivere codice...). E' però vero che in generale possiamo pensare di ottenere risultati ragionevoli se ricordiamo sempre che:

- A livello del singolo core vogliamo VETTORIZZARE quanto più conformemente alla architettura che abbiamo a disposizione.

- A livello multicore, vogliamo rendere efficiente la esecuzione multithreading.

Abbiamo visto all'opera su un semplicissimo benchmark (operazioni matriciali)

(a) ARRAY NOTATION, disponibile in ambiente di compilazione Intel (icc)

(b) Direttive OpenMP (ormai uno standard) e Cilk (disponibile in ambiente icc)

Un paio di notizie

(una buona) ARRAY NOTATION, OpenMP, Cilk sono strumenti semplici e abbastanza buoni

(una meno buona) Il MIC di suo non ha prestazioni confrontabili al guadagno teorico