

OTTIMIZZAZIONI MICROARCHITETTURALI PER L'HIGH PERFORMANCE COMPUTING

Relatore
prof. Renato Campanini

Presentata da
Luca Benini

Co-relatore
dott. Matteo Roffilli

Legge di Moore

Negli ultimi 30 anni i microprocessori hanno registrato una crescita senza precedenti in termini di:

- Capacità di memorizzazione
- Potenza di calcolo
- Grandi possibilità in tutti quei campi che non hanno limiti precostituiti alla potenza di calcolo richiesta
(Simulazioni, previsioni meteo, algoritmi genetici e machine learning)

Compilatori ed istruzioni vettoriali

- Le generazioni di processori attuali offrono anche a processori desktop la possibilità di operare in modalità vettoriale
- MMX, SSE, SSE2, SSE3, 3DNow!, AltiVec
- Tuttavia i progettisti di compilatori faticano ad integrare queste nuove istruzioni nei loro schemi di ottimizzazione ...
- ... di fatto queste istruzioni vengono spesso ignorate dai compilatori

Obiettivi

In questa tesi ci proponiamo di incrementare le prestazioni computazionali del progetto CAD (**C**omputer **A**ided **D**etection)

- Identificare i punti critici hardware e software
- Realizzare un'implementazione con performance crescenti della funzione scelta come target per le nostre ottimizzazioni
- Valutare e validare i risultati ottenuti

Sistema di Riferimento



Dual AMD Opteron 140, 2 GB ram

Gnu GCC 3.4.0



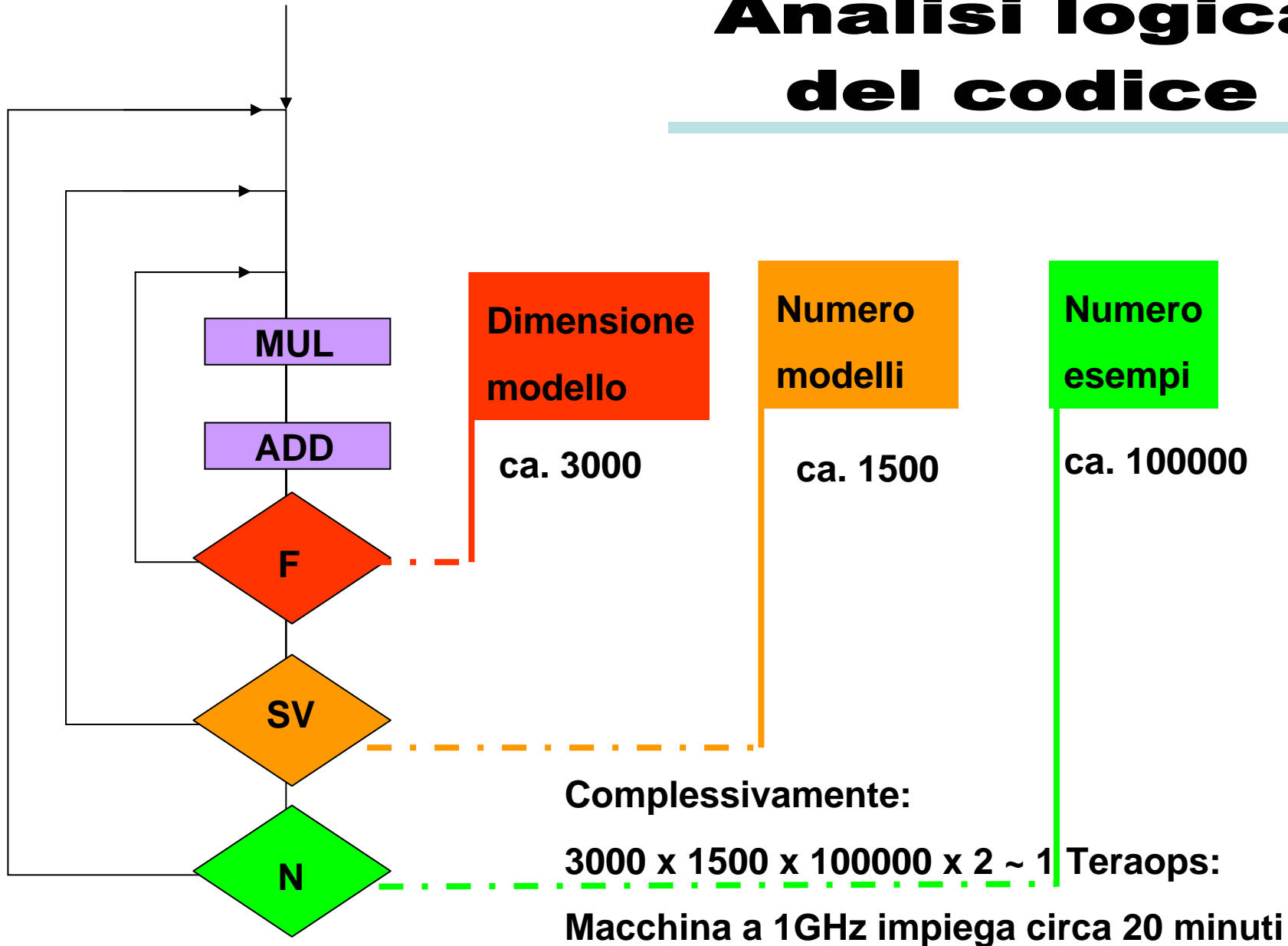
GNU/Linux Gentoo 2004.1

Passi preliminari

Per affrontare al meglio il processo di ottimizzazione si sono resi necessari alcuni passi preliminari:

- Analisi del codice sorgente con Intel VTune / Valgrind
- Identificati i moduli del codice che impiegano più dell'80% del tempo computazionale
- Identificato il singolo modulo su cui applicare le ottimizzazioni

Analisi logica del codice



Software Hot Spot

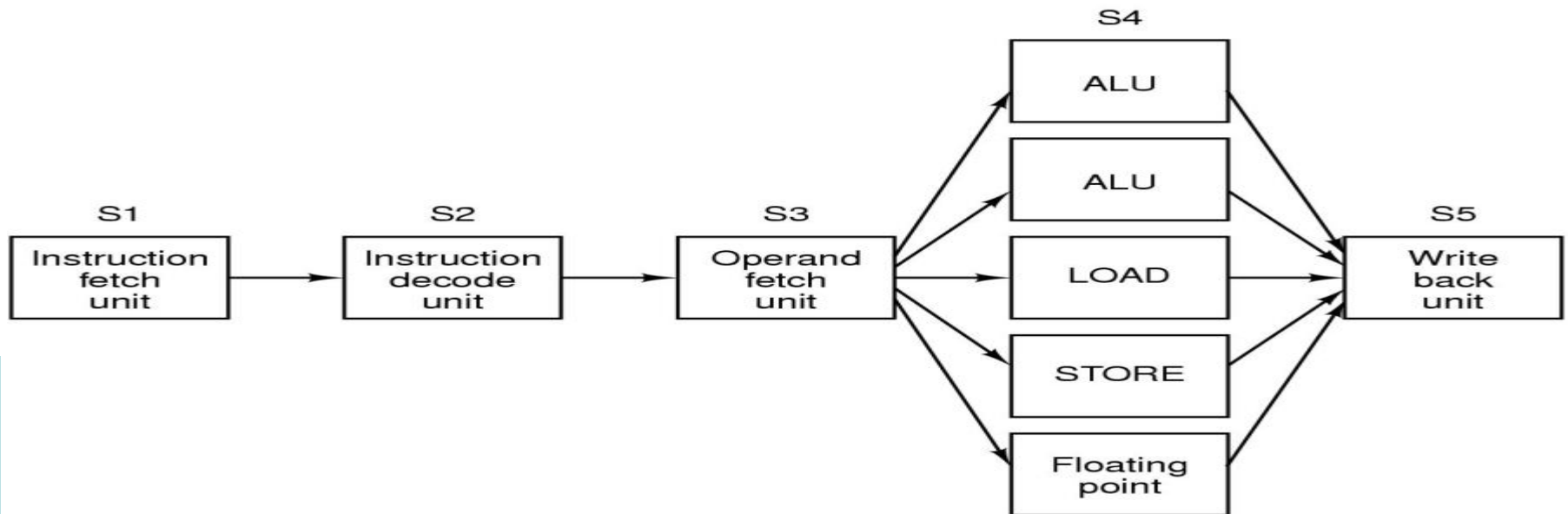
I più importanti target del processo ottimizzativo:

- Il prodotto scalare è il fulcro computazione dell'intero modulo
- L'uso seriale dell'unità di calcolo, che non sfrutta né il bus HyperTransport, né i due microprocessori

Hardware Hot Spot

Problematiche hardware note:

- Le latenze della memoria e della cache di secondo livello
- Le diramazioni interrompono il flusso della pipeline
- Le dipendenze dei dati impediscono l'esecuzione fuori ordine



Ottimizzazioni a livello sorgente

Dopo numerosi test abbiamo scelto di applicare le seguenti ottimizzazioni:

- **Multithreading (SMP) ;**
 - **S**ymmetric **M**ulti**P**rocessing
 - Le applicazioni devono utilizzare il multitasking
- **Loop Pipelining;**
 - Ridurre al minimo la necessità di verificare la variabile condizionale
 - Sfruttare al meglio le capacità superscalari dei microprocessori moderni

Alcune osservazioni

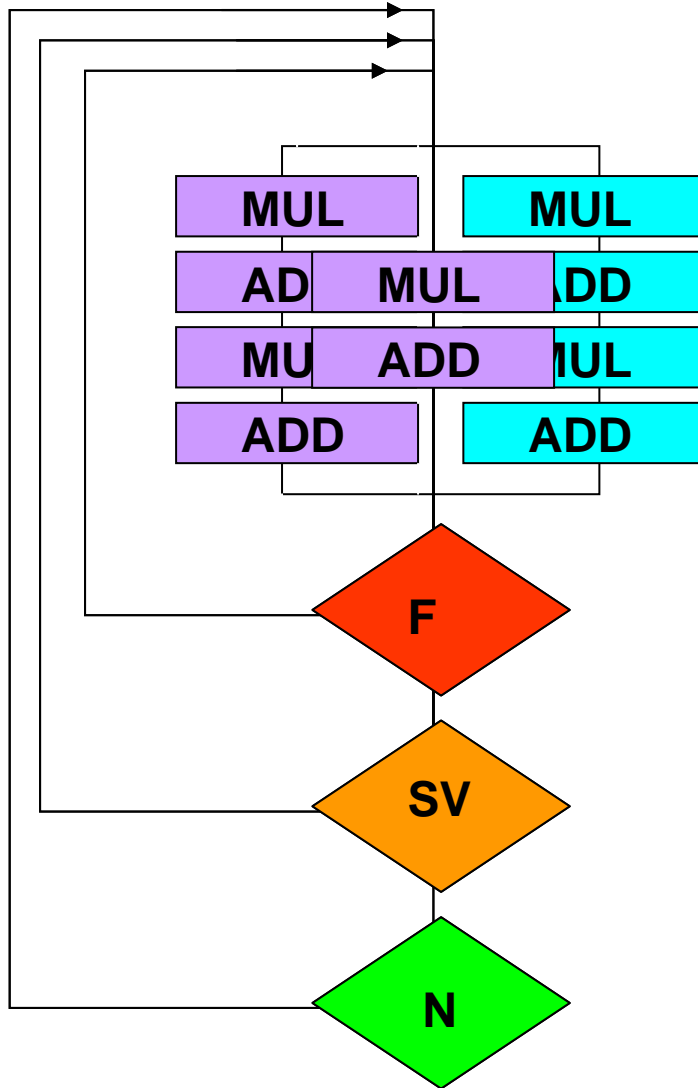
Perché il multithreading?

- Multiprocessore a prezzi ridotti
- HyperThreading (oggi)
- Dual Core (Gennaio 2005)

Perché il Loop Pipelining?

- Processori sempre più veloci delle memorie
- Più unità funzionali su singolo chip

Versione C ottimizzato



CPU 1

CPU 2

Ottimizzazioni a livello assembler

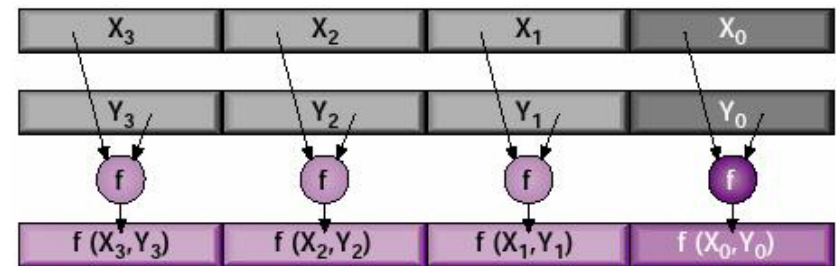
Dopo una fase di sperimentazione abbiamo scelto le seguenti ottimizzazioni:

- Tecnologia SWAR SSE
Simd Within A Register, Single Instruction Multiple Data
- Il prodotto scalare di 4 float contemporaneamente
- L'uso delle apposite istruzioni builtins

Altre osservazioni

Perché le istruzioni SSE?

- Datapath di 128 bit
- Istruzioni Superscalari (nel nostro caso 4 operazioni in contemporanea;
- Conformi IEEE (vs 3DNow!)



Perché le istruzioni builtins?

- Fornite al programmatore dal compilatore
- Prerogativa di tutti i compilatori di alto livello (*GCC*, *VS*, *Intel CC*)
- Possono essere mappate con codice nativo, o con template
- Anche su architettura che non presentano SSE

Assembler builtins

v4sf a,b,c,d,e,f,g,h,m,n;

[...]

a=__builtin_ia32_loadups(va);

b=__builtin_ia32_loadups(vb);

data loading

[...]

a=__builtin_ia32_mulps(a,b);

*Superscalar
multiplication*

[...]

__builtin_ia32_storess (&b2, a);

data store

Assembler hand-made

p_scalar:

```
movaps (%eax), %xmm2
movaps 32(%eax), %xmm3
movaps (%edx), %xmm6
movaps 32(%edx), %xmm7
movaps 16(%eax), %xmm1
movaps 16(%edx), %xmm5
movaps 48(%edx), %xmm0
movaps 48(%eax), %xmm4
```

```
mulps %xmm6, %xmm2
mulps %xmm5, %xmm1
mulps %xmm4, %xmm0
```

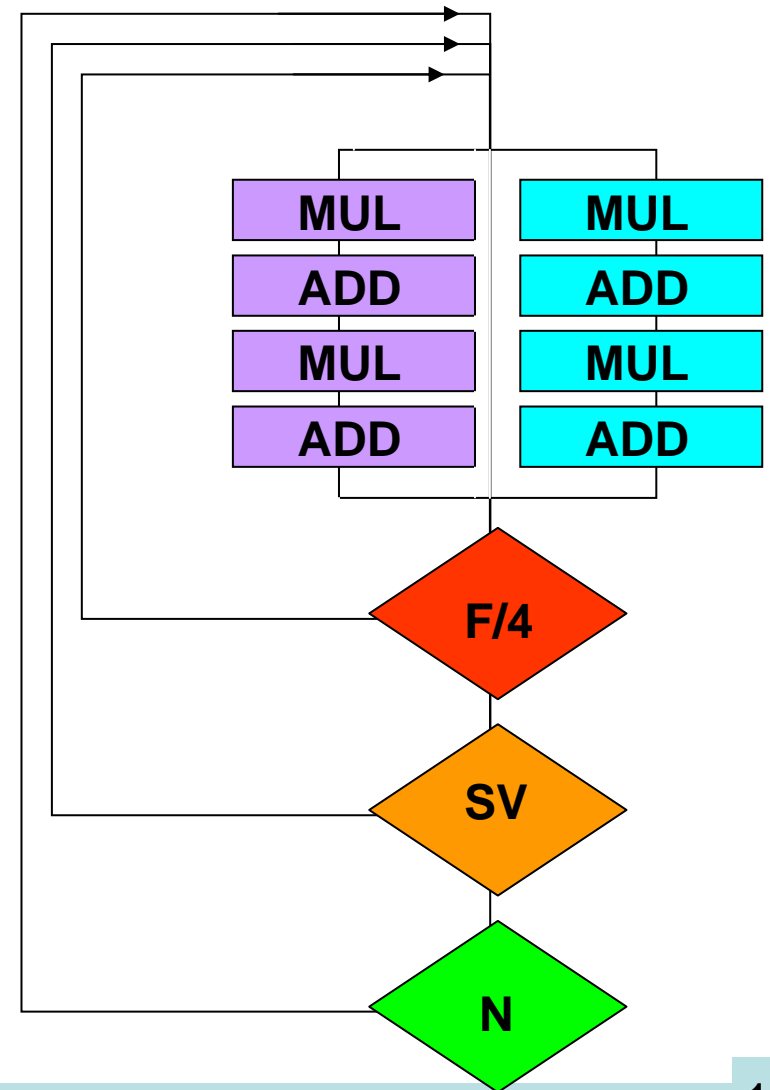
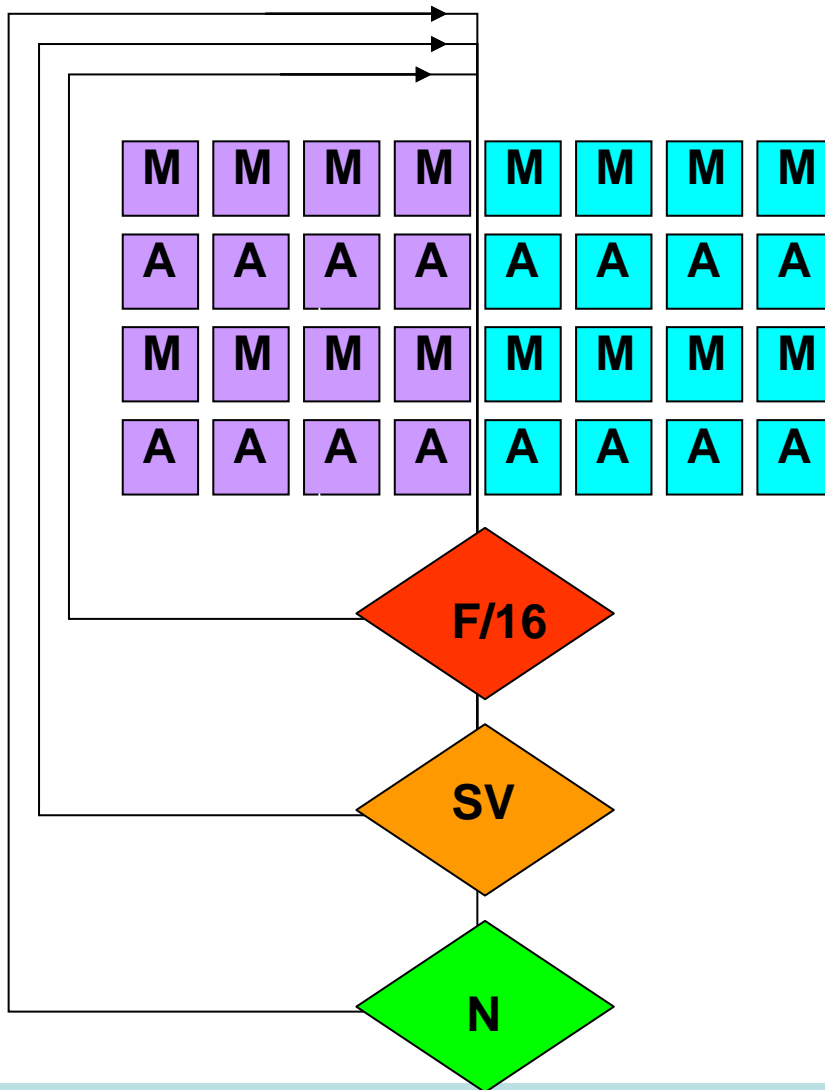
```
mulps %xmm7, %xmm3
addps %xmm1, %xmm2
addps %xmm3, %xmm0
subl $16, %ecx
addps -88(%ebp), %xmm2
addps -104(%ebp), %xmm0
addl $64, %eax
addl $64, %edx
testl %ecx, %ecx
```

```
movaps %xmm2, -88(%ebp)
movaps %xmm0, -104(%ebp)
jg .L12
```


CPU 1

CPU 2

Versione assembler



ATLAS

- ATLAS (**A**utomatically **T**uned **L**inear **A**lgebra **S**oftware);
<http://math-atlas.sourceforge.net/>
- Una versione dell' API BLAS (**B**asic **L**inear **A**lgebra **S**ubroutine)
- Ottimizzazione automatica a compile-time
- Utilizzo di *parametri di ottimizzazione*
- GEMM (**G**eneral **M**atrix **M**ultiply)
- Disponibili solo su Linux

Nel progetto:

- Abbiamo effettuato il porting fino ad ottenere una versione statica ottimizzata per Win32 linkabile con VS 6/.NET

Versione Atlas

estratto

```
(void)catlas_sset(ra*rb,1,c,1);
```

Memory set

[...]

```
(void)cblas_sgemm(CblasRowMajor, CblasNoTrans, CblasTrans,  
                [...],coef_lin, coef_const, [...]);
```

Matrix Multiply

[...]

```
(void)cblas_scopy(ra*rb,c,1,tmp,1);
```

Memory copy

```
for(i=0;i<rb;i++)
```

Scalar multiply

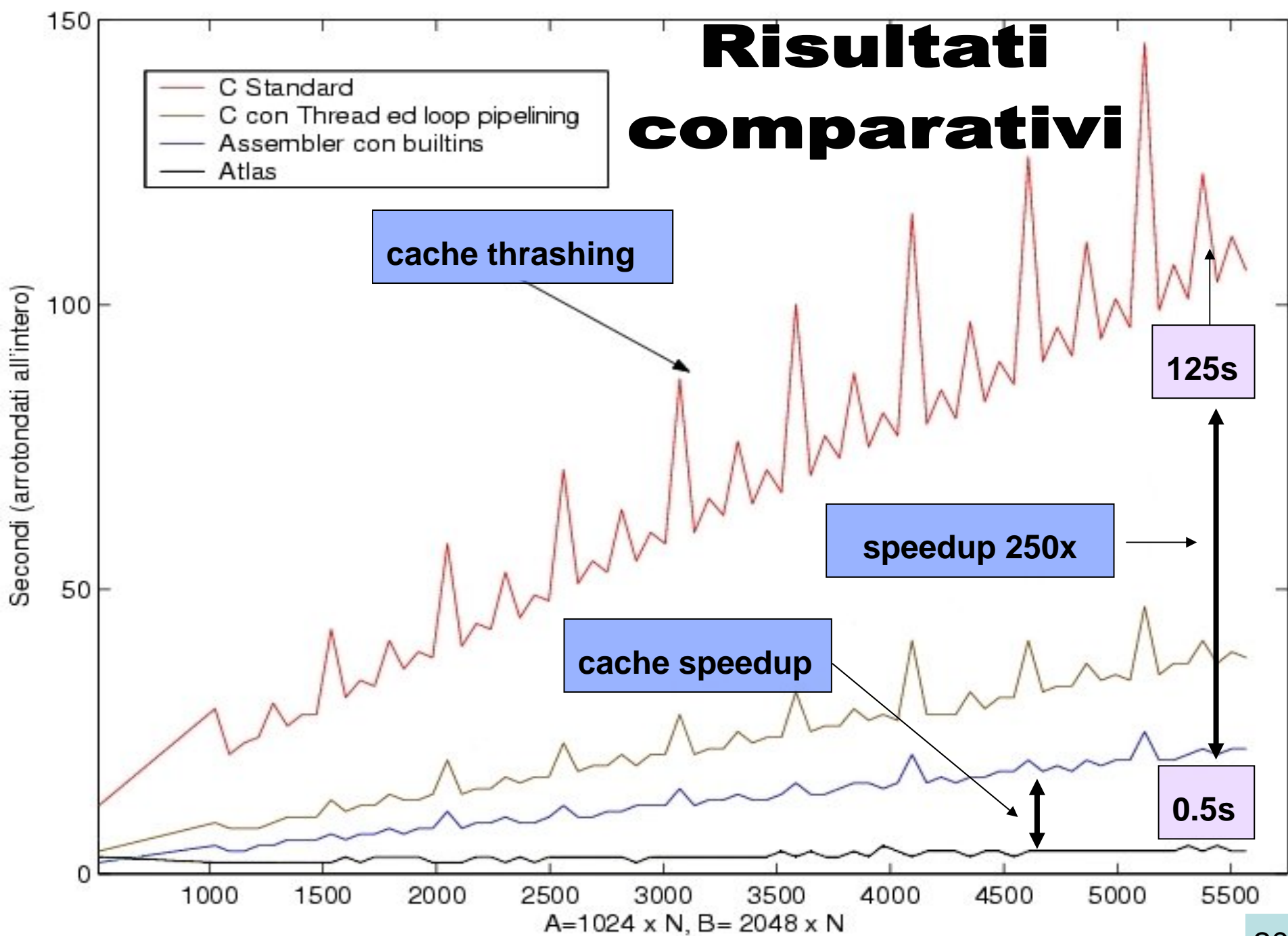
```
(void)cblas_sscal(ra,alfa[i],c+i,rb);
```

```
for(i=0;i<ra;i++)
```

Dot multiply

```
result[i]=cblas_sdot(rb,c+i*rb,1,tmp+i*rb,1)-threshold;
```

Risultati comparativi



Ulteriori risultati

- Tempi da 20 minuti (1000 secondi) a 4 secondi
- Portabilità (*win32* - **nix*)
- Scalabilità (1-16 processori)
- 32-64 bit
- Generalità (*algebra lineare*)
- Handmade assembler approssima Atlas

Conclusioni

- Si è dimostrato che è possibile ottenere uno speed-up del codice di circa 250x rispetto alla versione C ottimizzata
- Le ottimizzazioni dei compilatori nella maggior parte dei casi non sono sufficienti per il core computazionale
- Le macchine moderne hanno ampi margini di miglioramento software
- E' possibile ottenere ottimizzazioni portabili

Prospettive future

Guardando al campo ottimizzativo dei prossimi anni gli elementi più innovativi saranno probabilmente:

- Hand made cache optimization
- PCI-Express, GPU as CPU
 - Multicore, Bus ottici
- Automatic optimizations

OTTIMIZZAZIONI MICROARCHITETTURALI PER L'HIGH PERFORMANCE COMPUTING

Relatore
prof. Renato Campanini

Presentata da
Luca Benini

Co-relatore
dott. Matteo Roffilli