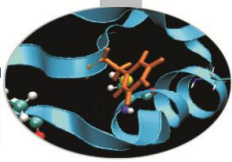
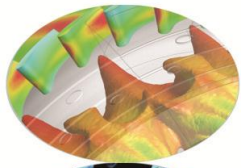
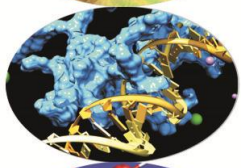
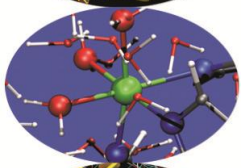
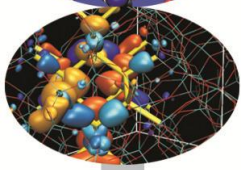
CINECA SCAI
SuperComputing Applications and Innovation



Compilatori e Livelli di Compilazione

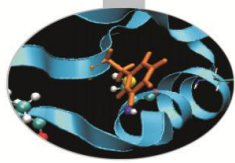





Introduction to Fortran 90

Paolo Ramieri, *CINECA*

Aprile 2014

CINECA



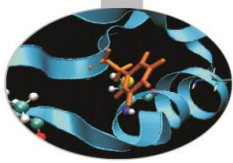
Il compilatore

Il programmatore ha a disposizione un'ampia scelta di compilatori sviluppati per diversi sistemi operativi quali Linux/Unix, Windows, Macintosh.

Oltre ai compilatori gratuiti derivanti dal progetto GNU, esistono diversi compilatori commerciali.

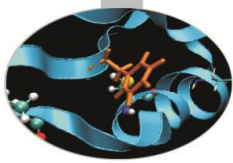
I più conosciuti sono probabilmente quelli prodotti dall'Intel e dalla Portland.

Una certa rilevanza di mercato hanno anche i compilatori commercializzati dalla Absoft, dalla HP e da IBM.



Il compilatore

- Traduce il codice sorgente in codice macchina
- Rifiuta codici sintatticamente errati
- Segnala (alcuni) potenziali problemi semantici
- Può tentare di “ottimizzare” il codice
  - ottimizzazioni indipendenti dal linguaggio
  - ottimizzazioni dipendenti dal linguaggio
  - ottimizzazioni dipendenti dalla CPU
  - ottimizzazioni dell’uso della memoria e della cache
  - suggerimenti al processore su cosa probabilmente farà il codice

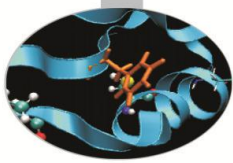


Il compilatore

Esegue trasformazioni del codice come:

- Dead and redundant code removal
- Common subexpression elimination (CSE)
- Strength reduction
- Inlining
- Index reordering
- Loop pipelining/unrolling, merging
- Cache blocking

Lo scopo è massimizzare le prestazioni

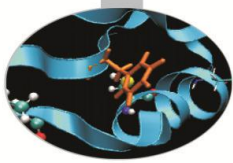


Opzioni di Ottimizzazione

I compilatori prevedono numerose **opzioni di ottimizzazione** che vengono **specificate in fase di compilazione**, unitamente alle altre.

Esistono livelli di ottimizzazione **generali** e **specifici,** legati all'hardware sottostante.

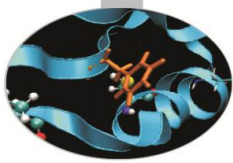
Alcune opzioni possono risultare onerose dal punto di vista del tempo richiesto per la compilazione. Per questo motivo, è consigliabile utilizzare livelli bassi di ottimizzazione quando si è ancora in fase di sviluppo del codice.



Opzioni generiche

Le opzioni generiche non sono legate ad un particolare hardware ma permettono comunque di ottenere eseguibili molto performanti.

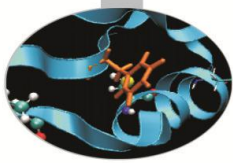
La più comune è indicata dalla sintassi `-On` dove n può assumere i valori interi compresi fra 0 e 4, dove ogni valore è cumulativo, ovvero include le caratteristiche dei livelli precedenti, migliorandone le funzionalità, anche se la compilazione risulterà più lenta.



Livelli di ottimizzazione

Un compilatore presenta livelli incrementali di ottimizzazione, ad esempio:

- -O0: nessuna ottimizzazione, il codice è tradotto letteralmente
- -O1, -O2: ottimizzazioni locali, compromesso tra la velocità di compilazione, ottimizzazione e dimensioni dell'eseguibile (livello di default)
- -O3: ottimizzazioni memory-intensive, può alterare la semantica del programma
- -O4: ottimizzazioni aggressive



Livello O0

Per alcuni compilatori si tratta del livello di ottimizzazione di default.

Opera solo sulle dichiarazioni di variabili riferite ad un unico codice sorgente. Questo livello include le seguenti ottimizzazioni:

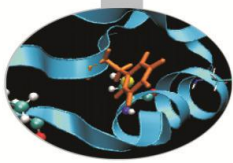
- Ridefinizione dei valori costanti:

  `Y = 5 + 7` diventa `Y = 12`

- Valutazione parziale delle clausole condizionali:

  ```
  IF ((I.EQ.J).OR.(I.EQ.K)) THEN
  ```

- Assegnazioni delle variabili più utilizzate nel registro di memoria e allineamento dei dati.

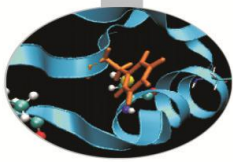


Livello O1

Questo livello eredita le caratteristiche del livello precedente, integrandolo con le seguenti ottimizzazioni:

- Eliminazione delle parti mai eseguite del codice
- Ottimizzazione delle diramazioni trasformando in modo efficiente i costrutti decisionali
- Migliore programmazione dell'ordine di esecuzione delle istruzioni, cercando di diminuire la latenza dovuta alla richiesta di dati non in cache

Livello O2



Oltre a quanto previsto nei livelli precenti vengono eseguite anche le seguenti operazioni:

- Propagazione della ridefinizione dei valori costanti:
  `A = 10; B = A + 5` diventa `A=10; B=15`
- Eliminazione delle sottoespressioni comuni:

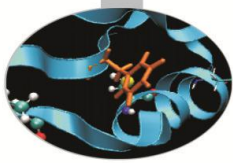
```
A = X + Y + Z
B = X + Y + W
```

diventa

```
T1 = X + Y
A = T1 + Z
B = T1 + W
```

- Eliminazione delle variabili ormai inutilizzate a causa di ottimizzazioni successive.

Livello O2

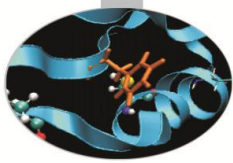


Loop Unrolling: questa tecnica molto importante permette di ridurre l'overhead dovuto alla modifica del loop counter e alla verifica del raggiungimento di termine loop. Inoltre permette di sfruttare l'esecuzione in parallelo nel caso di pipeline superscalari.

```
DO i = 1, 100
 A (i) = B (i) + C (i)
ENDDO
```

diventa

```
DO i = 1, 100, 4
      A (i) = B (i) + C (i)
      A (i + 1) = B (i + 1) + C (i + 1)
      A (i + 2) = B (i + 2) + C (i + 2)
      A (i + 3) = B (i + 3) + C (i + 3)
ENDDO
```



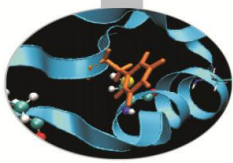
Livello O2

- **Sostituzione di funzioni lineari dell'indice del loop**: vengono rimosse espressioni che sono funzioni lineari dell'indice del loop e sostituite con variabili che contengono il risultato di tali funzioni.

```
DO i = 1, 25
      R (i) = i * k
ENDDO
```

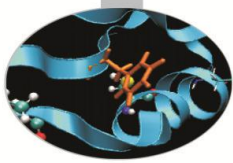
diventa

```
T1 = k
DO i = 1, 25
      R (i) = T1
      T1 = T1 + k
ENDDO
```



Livello O2

- **Inlining**: tecnica attraverso la quale il corpo di una procedura semplice, richiamata molte volte, viene sostituito alla chiamata della procedura stessa, eliminando l'overhead dovuto alla chiamata.

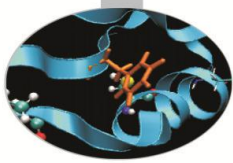


Livello O3

Questo livello applica un'ottimizzazione aggressiva causando anche comportamenti anomali in codici mal programmati.

In particolare agisce profondamente sui loop attraverso tecniche di strip mining, unrolling, interchange, fusion, distribution, blocking, unroll & jam.

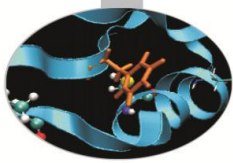
Tutte queste operazioni sono mirate ad ottenere una riduzione dell'overhead e una maggiore località e allineamento dei dati in cache.



Livello O3

- **Prefetching**: tecnica molto importante attraverso la quale i dati vengono resi disponibili prima che siano realmente richiesti riducendo la latenza dovuta all'accesso alla RAM a causa di cache miss.

Livello O3



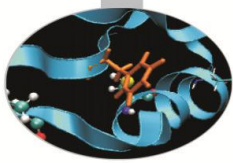
- **Trasformazione istruzioni floating point**: quando possibile le operazioni di divisione vengono modificate in moltiplicazioni.

Le divisioni non possono essere pipelined ed impiegano da 20 a 60 cicli di clock, mentre le moltiplicazioni possono essere pipelined ed impiegano da 5 a 10 cicli di clock.

```
DO i=1,n
      x(i)=a(i)/b
ENDDO
```

diventa:

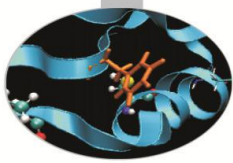
```
rb=1./b
DO i=1,n
      x(i)=a(i)*rb
ENDDO
```



Opzioni legate all'hardware

Istruzioni SIMD: tutti i moderni processori possiedono un instruction set SIMD (Single Instruction Multiple Data), che permette la vettorizzazione delle istruzioni integer e floating point le quali vengono eseguite in parallelo su un set vettoriale di dati.

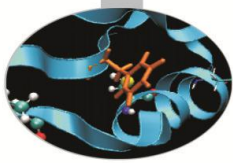
A seconda dei compilatori tali istruzioni vengono richiamate con flag differenti



Opzioni legate all'hardware

Istruzioni SIMD:

- Compilatore Intel:
  - `-ax(flag cpu)` per processori Intel o AMD
  - `-axS` per processori che supportano istruzioni SSE
  - `-mavx2` per processori che supportano istruzioni AVX
- Compilatore PGI:
  - `-fast`
  - `-fastsse` per processori AMD o Intel
- Compilatore GNU:
  - `-msse`
  - `-msse2`
  - `-msse3` per processori Intel o AMD

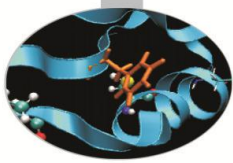


Opzioni raccomandate

- **Compilatori Intel**: `-O3 -ipo -ax[processore]`
  L'opzione `-ipo` abilita l'ottimizzazione interprocedurale fra files, in particolare esegue l'inlining fra procedure definite in files differenti.

- **Compilatori PGI**: `-fastsse -Mcache_align -Minline -Mvect=prefetch -Mipa=fast`

- **Compilatori GNU**: `-fast -msse[n]`



gfortran –o prova -O0 program.f90 sub1.f90 …

ls –l

…. Prova

./prova

Vi program.f90