
# Compilazione e linking

Introduction to modern Fortran

**Tommaso Gorni**
t.gorni@cineca.it

# Dal sorgente all'eseguibile

Codice sorgente (file di testo)

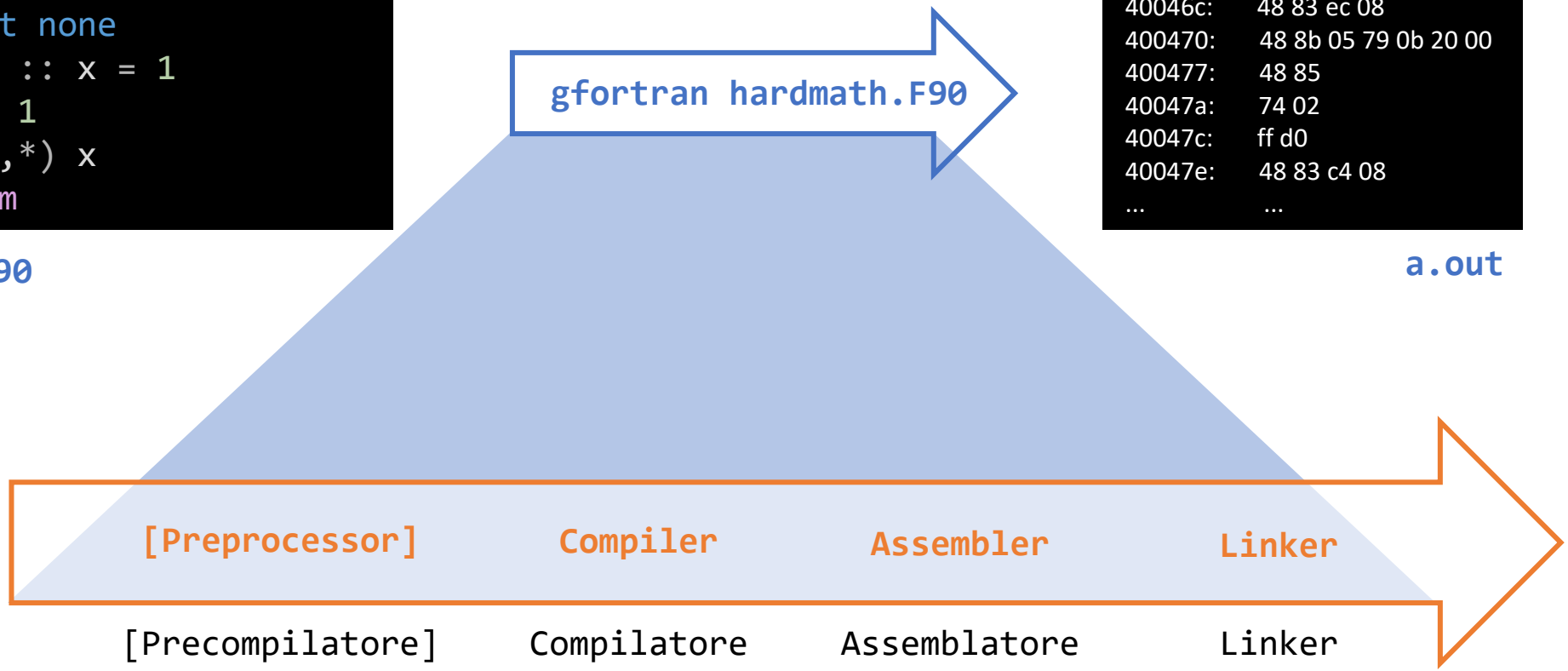
```
program hardmath
   implicit none
   integer :: x = 1
   x = x + 1
   write(*,*) x
end program
```

hardmath.F90

Eseguibile (linguaggio macchina)

```
400468:    f3 0f 1e fa
40046c:    48 83 ec 08
400470:    48 8b 05 79 0b 20 00
400477:    48 85
40047a:    74 02
40047c:    ff d0
40047e:    48 83 c4 08
...        ...
```

a.out



# Fase di precompilazione

Il **precompilatore (*preprocessor*)** analizza il codice sorgente e restituisce in output lo stesso sorgente dove tutte le **direttive di precompilazione** sono state sostituite. Le direttive più comuni per il preprocessore C (`cpp`) sono:

- l'**inclusione** di *header*

```
#include <stdio.h>
#include "my_awesome_header.h"
```

- compilatione **condizionale**

```
#ifdef MY_AWESOME_FEATURE
   do_this()
#else
   do_that()
#endif
```

- espansione di **macro**

```
#define PI 3.1415926535897932
```

⇨ Percorsi per la ricerca per gli *header* possono essere aggiunti con l'opzione `-I`:

```
cpp -I/another/place/where/to/look ...
```

⇨ Attivate con la l'opzione `-D`:

```
cpp -DMY_AWESOME_FEATURE ...
```

(equivalente a `MY_AWESOME_FEATURE=1`).

The C Preprocessor (gnu.org)

# Fase di precompilazione

Anche se **lo standard Fortran non prevede nessuna fase di precompilazione**, il precompilatore C è correntemente usato su sorgenti Fortran, in particolare per la compilazione condizionale.

Tutti i principali compilatori Fortran eseguono automaticamente il precompilatore C su un sorgente Fortran se il relativo file termina per *.F o *.F90.

In tal caso, l'opzione –E può essere passata al compilatore affinché questo si arresti subito dopo la fase di precompilazione, così da esaminare il codice precompilato.

Alternativamente, il precompilatore C può essere invocato manualmente con le seguenti opzioni (che assicurano che venga generato codice Fortran valido):

```
$ cpp -C -P -traditional my_src.f90 > my_src.i
```

# Un esempio più realistico

```
$ cd intro-fortran-bologna/day2/compilazione_e_linking/esempio_libmath
$ ls
hardmath.F90  math.F90
```

```
program hardmath
   use math
   implicit none
   integer :: x = 1
   call increment(x)
   print '("1 + 1 = ",i0)', x
end program
```

hardmath.F90

```
module math
   implicit none
contains
   subroutine increment(x)
      implicit none
      integer :: x
      x = x + 1
   end subroutine increment
   subroutine decrement(x)
      implicit none
      integer :: x
      x = x - 1
   end subroutine decrement
end module math
```

math.F90

# Fase di compilazione

Nella fase di compilazione (compilazione + assemblaggio, in realtà), ogni file sorgente `*.f90` (*source file*) viene convertito in file oggetto `*.o` (*object file*).

```
$ ls
hardmath.f90          math.f90
$
$ gfortran -c math.f90
$ gfortran -c hardmath.f90
$
$ ls
hardmath.f90  hardmath.o  math.f90  math.mod  math.o
```

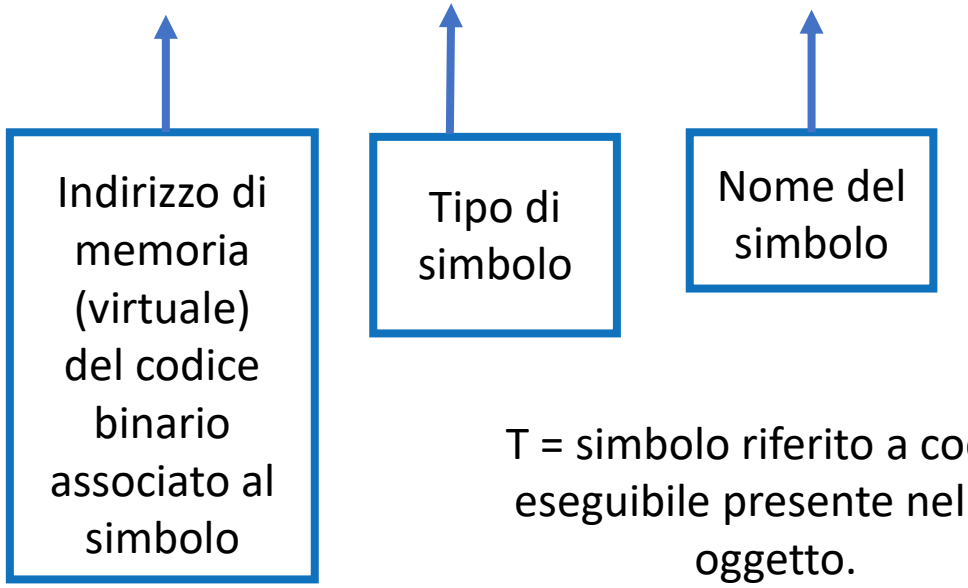
Ogni **file oggetto *.o contiene** il codice macchina corrispondente alle procedure del rispettivo file sorgente, più una **tabella di simboli** che indicano gli **indirizzi** di memoria associati ai **dati** e ai **punti di entrata (*entry point*) delle procedure** ivi definite.

# Fase di compilazione

In ambiente Unix, la tabella dei simboli si può visualizzare attraverso il comando **nm**:

```
$ nm math.o

0000000000000000 T __math_MOD_decrement
000000000000001a T __math_MOD_increment
```




T = simbolo riferito a codice eseguibile presente nel file oggetto.

```
module math
   implicit none
contains
   subroutine increment(x)
      implicit none
      integer :: x
      x = x + 1
   end subroutine increment
   subroutine decrement(x)
      implicit none
      integer :: x
      x = x - 1
   end subroutine decrement
end module math
```

math.F90

# Fase di compilazione

In ambiente Unix, la tabella dei simboli si può visualizzare attraverso il comando **nm**:

```
$ nm hardmath.o

0000000000000000 t MAIN__
                 U __math_MOD_increment
                 U _gfortran_set_args
                 U _gfortran_set_options
                 U _gfortran_st_write
                 U _gfortran_st_write_done
                 U _gfortran_transfer_integer_write
000000000000009e T main
0000000000000020 r options.1.0
0000000000000000 d x.1
```

```
program hardmath
   use math
   implicit none
   integer :: x = 1
   call increment(x)
   print '("1 + 1 = ",i0)', x
end program
```

**hardmath.F90**

**T/t** : codice eseguibile presente nel file oggetto.
**U** : simbolo non definito nel file oggetto, deve essere trovato altrove in fase di *linking*.
**D/d** : dati inizializzati.
**R/r** : *read-only* data (Rodata).

Maiuscolo: simbolo globale (può essere esportato).
Minuscolo: simbolo locale.

# Fase di compilazione

- Codice Fortran ***fixed-form*** e ***free-form*** **possono coesistere**. Il compilatore può essere forzato ad interpretare un determinato sorgente secondo un determinato formato, ad esempio `gfortran` ammette le opzioni `-ffixed-form` e `-ffree-form`.

- Se sono utilizzati **moduli**, **l'ordine di compilazione conta!**

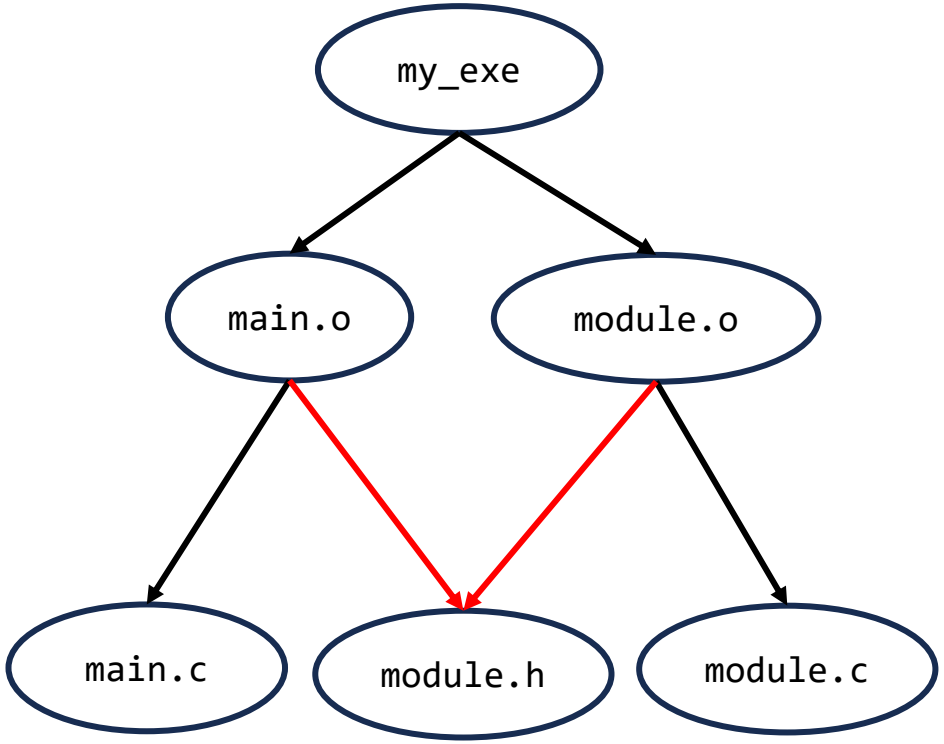
```
$ gfortran -c hardmath.F90 math.F90
hardmath.F90:2:7:

    use math
       1
Fatal Error: Can't open module file ‘math.mod’ for reading at (1): No such file or directory
compilation terminated.
$
$ gfortran -c math.F90 hardmath.F90
```

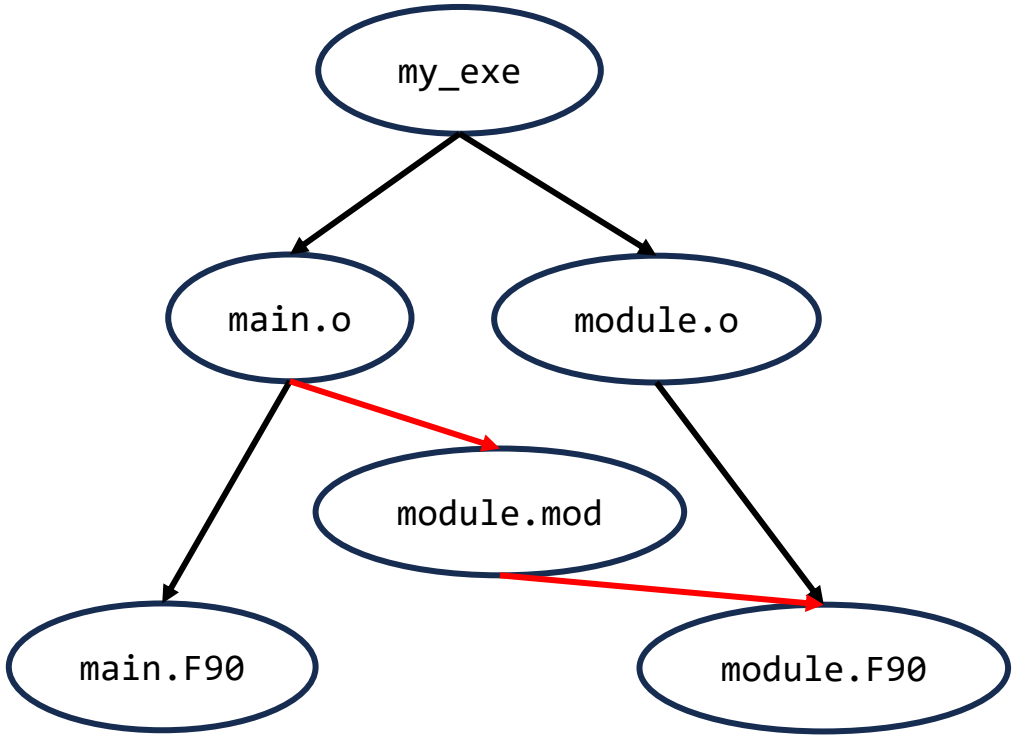
Fortran Dialect Options (The GNU Fortran Compiler)

# Dipendenze: C vs Fortran

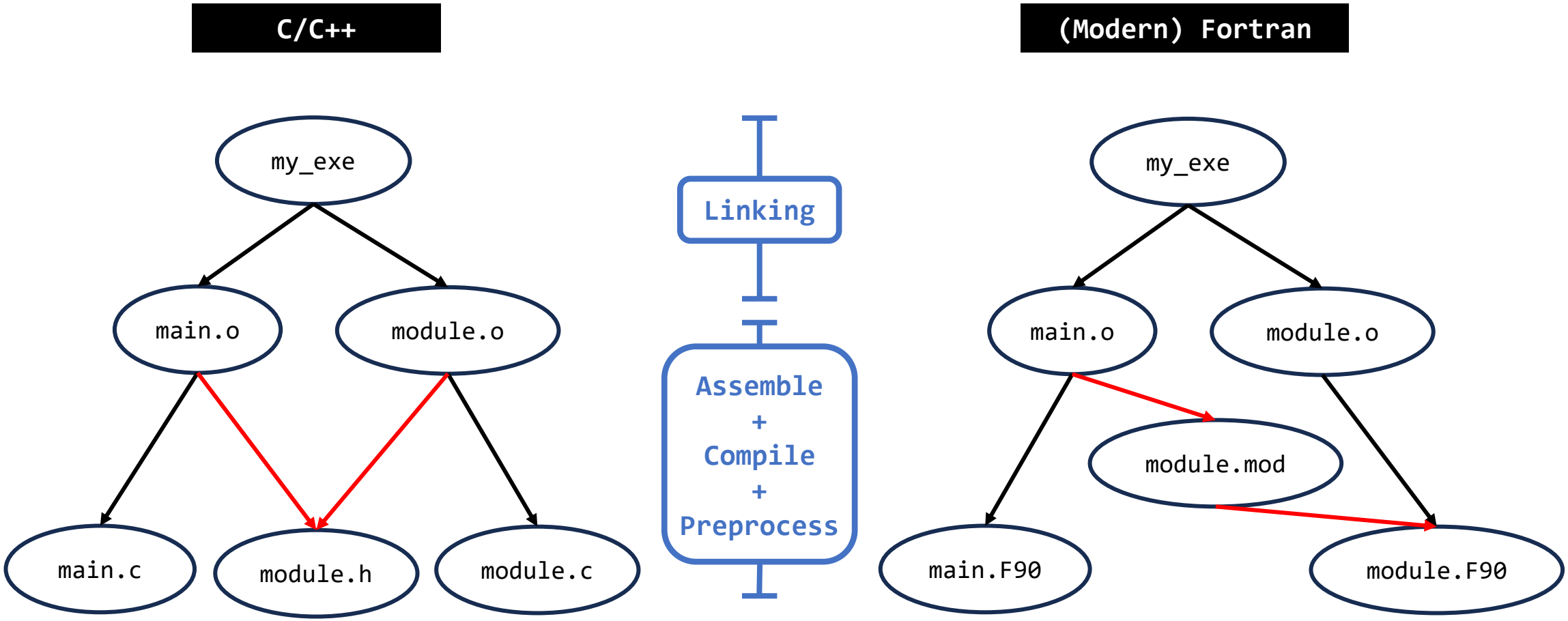
**C/C++**




**(Modern) Fortran**



# Dipendenze: C vs Fortran



# ... ma quale estensione uso per i sorgenti?

| Compiler | Fixed-form | Fixed-form with preprocessor | Free-form | Free form with preprocessor |
|---|---|---|---|---|
| gfortran | .f, .for, .ftn | .fpp, .F, .FOR, .FPP, .FTN | .f90, .f95, .f03, .f08 | .F90, .F95, .F03, .F08 |
| ifort | .f, .for, .ftn, .i | .fpp, .FPP, .F, .FOR, .FTN | .f90, .i90 | .F90 |
| nvfortran | .f .for .ftn | .F .FOR .FTN .fpp .FPP | .f90 .f95 .f03 | .F90 .F95 .F03 |
| nagfor | .f, .for, .ftn | .ff, .F | .f90, .f95 | .ff90, .ff95, .F90, .F95 |
| Cray | .f, .for | .F, .FOR | .f90, .f95, .f03, .f08, .f18, .ftn | .F90, .F95, .F03, .F08, .F18, .FTN |
| IBM | .f, .f77 | .F, .F77 | .f90, .f95, .f03, .f08 | .F90, .F95, .F03, .F08 |
| g95 | .f, .for | .F, .FOR | .f90, .f95, .f03 | .F90, .F95, .F03 |

Is there a "standard" file suffix for modern Fortran code?

# ... ma quale estensione uso per i sorgenti?

| Compiler | Fixed-form | Fixed-form with preprocessor | Free-form | Free form with preprocessor |
|---|---|---|---|---|
| gfortran | .f, .for, .ftn | .fpp, .F, .FOR, .FPP, .FTN | .f90, .f95, .f03, .f08 | .F90, .F95, .F03, .F08 |
| ifort | .f, .for, .ftn, .i | .fpp, .FPP, .F, .FOR, .FTN | .f90, .i90 | .F90 |
| nvfortran | .f .for .ftn | .F .FOR .FTN .fpp .FPP | .f90 .f95 .f03 | .F90 .F95 .F03 |
| nagfor | .f, .for, .ftn | .ff, .F | .f90, .f95 | .ff90, .ff95, .F90, .F95 |
| Cray | .f, .for | .F, .FOR | .f90, .f95, .f03, .f08, .f18, .ftn | .F90, .F95, .F03, .F08, .F18, .FTN |
| IBM | .f, .f77 | .F, .F77 | .f90, .f95, .f03, .f08 | .F90, .F95, .F03, .F08 |
| g95 | .f, .for | .F, .FOR | .f90, .f95, .f03 | .F90, .F95, .F03 |

**Usare `.F90` o `.f90` per sorgenti *free form* e `.F` o `.f` per sorgenti *fixed form*!**

Is there a "standard" file suffix for modern Fortran code?

# Fase di *Linking*

Nella fase di ***linking***, il compilatore (più precisamente il *linker*) accorpa tutti i file oggetto `*.o` assieme per formare l'eseguibile del programma.

```
$ gfortran hardmath.o math.o
$ nm a.out

0000000000011ad t MAIN__
[...]
0000000000001179 T __math_MOD_decrement
0000000000001193 T __math_MOD_increment
[...]
                 U _gfortran_set_args@@GFORTRAN_8
                 U _gfortran_set_options@@GFORTRAN_8
                 U _gfortran_st_write@@GFORTRAN_8
                 U _gfortran_st_write_done@@GFORTRAN_8
                 U _gfortran_transfer_integer_write@@GFORTRAN_8
[...]
```

*Linking*: l'ordine dei *.o non conta.

# Librerie statiche

Una **libreria statica** non è altro che un file **archivio** (tipo un file `.tar`) di uno o più **file oggetto**. In ambiente Unix si può creare l'archivio con `ar`

```
$ ar r libmath.a math.o
ar: creating libmath.a
```

E poi passare la libreria al *linker* come se fosse un normale file oggetto

```
$ gfortran hardmath.o libmath.a
$ ./a.out
1 + 1 = 2
```

oppure con la sintassi

```
$ gfortran hardmath.o -L. -lmath
$ ./a.out
1 + 1 = 2
```

# Librerie statiche

**NB:** nel *linking* di librerie, i simboli vengono risolti da destra a sinistra, **l'ordine conta**!

```
$ gfortran libmath.a hardmath.o

/usr/bin/ld: hardmath.o: in function `MAIN__':
hardmath.f90:(.text+0x13): undefined reference to `__math_MOD_increment'
collect2: error: ld returned 1 exit status

$ gfortran -L. -lmath hardmath.o

/usr/bin/ld: hardmath.o: in function `MAIN__':
hardmath.f90:(.text+0x13): undefined reference to `__math_MOD_increment'
collect2: error: ld returned 1 exit status
```

# Librerie statiche

Nella fase di *linking* con una libreria statica, il ***linker* accorpa** automaticamente nell'eseguibile **tutti i file oggetto che la compongono e che vengono richiamati nel programma principale.**

```
$ nm libmath.a

math.o:
0000000000000000 T __math_MOD_decrement
000000000000001a T __math_MOD_increment

$ nm a.out
0000000000001179 t MAIN__
[...]
0000000000001235 T __math_MOD_decrement
000000000000124f T __math_MOD_increment
[...]
```

```
module math
   implicit none
contains
   subroutine increment(x)
      implicit none
      integer :: x
      x = x + 1
   end subroutine increment
   subroutine decrement(x)
      implicit none
      integer :: x
      x = x - 1
   end subroutine decrement
end module math
```

math.F90

# Librerie dinamiche

Una libreria dinamica (o condivisa) su Linux può essere generata compilando il sorgenti come *position-independent code* (PIC) e raggruppando i file oggetto in uno *shared object* (`*.so`):

```
$ gfortran -c -fPIC math.f90
$ gfortran -shared -o libmath.so math.o
$
$ nm libmath.so
0000000000003e90 d _DYNAMIC
0000000000004000 d _GLOBAL_OFFSET_TABLE_
[...]
00000000000010f9 T __math_MOD_decrement
0000000000001113 T __math_MOD_increment
```

Il binario così generato contiene le informazioni necessarie per poter essere caricato da un eseguibile durante l'esecuzione (a *runtime*), in questo caso la *global offset table* del formato ELF.

# Librerie dinamiche

Nella fase di *linking* il binario di una libreria dinamica non viene incluso nel programma eseguibile, viene invece inserita una lista dei simboli della libreria che saranno necessari all'eseguibile a *runtime*.

```
$ gfortran hardmath.f90 -L. -lmath
$ nm a.out
0000000000001189 t MAIN__
0000000000003d80 d _DYNAMIC
0000000000003f90 d _GLOBAL_OFFSET_TABLE_
[...]
                 U __math_MOD_increment
```

A *runtime*, il *linker* dinamico cerca librerie che contengano i simboli necessari e ne carica il binario nell'eseguibile.

```
$ ./a.out
./a.out: error while loading shared libraries: libmath.so: cannot open shared object file:
No such file or directory
```

... sempre che sappia dove trovarlo!

# Librerie dinamiche

Su Unix, il programma `ldd` lista lel librerie dinamiche che verranno usate da un determinato eseguibile:

```
$ ldd a.out
  [...]
        libmath.so => not found
        libgfortran.so.5 => /lib/x86_64-linux-gnu/libgfortran.so.5 (0x00007ffba4ec0000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007ffba4cc0000)
```

Per aggiungere percorsi di ricerca si può usare la variabile d'ambiente `LD_LIBRARY_PATH`

```
$ export LD_LIBRARY_PATH=./
$ ./a.out
1 + 1 = 2
```

Oppure salvare un percorso di default direttamente dentro all'eseguibile con `rpath`

```
$ gfortran -Wl,-rpath,./ hardmath.f90 -L. -lmath
$ ./a.out
1 + 1 = 2
```

# Librerie dinamiche

- Una libreria dinamica permette di riutilizzare lo stesso binario in diversi eseguibili senza farne una copia (come invece accadrebbe con una libreria statica).

- Se una libreria dinamica viene aggiornata (ad esempio per eliminare un baco) non bisogna ricompilare l'eseguibile, ma è sufficiente rendere la nuova libreria disponibile a *runtime*. [**NB:** sempre che non cambi l'interfaccia alle procedure, in tal caso la libreria avrebbe subito un *major upgrade* e non sarebbe retrocompatibile, provocando anche l'incremento del numero *major* della versione, ad esempio 5.4.0 => 6.0.0].

- Le librerie dinamiche sono utilizzate dai compilatori stessi per fornire alcune delle funzionalità del linguaggio, come procedure di I/O o funzioni matematiche intrinseche.

# Librerie dinamiche

```
$ gfortran hardmath.o math.o
$ nm a.out

00000000000011ad t MAIN__
[...]
                 U _gfortran_set_args@@GFORTRAN_8
                 U _gfortran_set_options@@GFORTRAN_8
                 U _gfortran_st_write@@GFORTRAN_8
                 U _gfortran_st_write_done@@GFORTRAN_8
                 U _gfortran_transfer_integer_write@@GFORTRAN_8
[...]

$ gfortran -v math.o hardmath.o
Driving: gfortran -v math.o hardmath.o -l gfortran -l m -shared-libgcc
[...]
LIBRARY_PATH=/usr/lib/gcc/x86_64-linux-gnu/9/:/usr/lib/gcc/x86_64-linux-
gnu/9/../../../x86_64-linux-gnu/:/usr/lib/gcc/x86_64-linux-
gnu/9/../../../../lib/:/lib/x86_64-linux-gnu/:/lib/../lib/:/usr/lib/x86_64-linux-
gnu/:/usr/lib/../lib/:/usr/lib/gcc/x86_64-linux-gnu/9/../../../:/lib/:/usr/lib/
```

# ... ma i file .mod?

La compilazione di un modulo Fortran genera automaticamente un corrispondente file `*.mod`, il quale contiene le interfacce alle sue procedure.

Se uno o più moduli sono compilati e raccolti in una libreria, la posizione dei rispettivi file `*.mod` deve essere indicata attraverso l'opzione `-I` già alla fase di **compilazione dell'eseguibile**.

```
$ gfortran -I/path/to/.mod/folder hardmath.f90 -L/path/to/libmath.so/folder -lmath
```

In fase di compilazione è normalmente possibile indicare un percorso dove salvare i file `*.mod`, ad esempio con `gfortran`

```
$ gfortran -c math.f90 -J/path/where/to/save/math.mod
```

I file `*.mod`, dipendendo sia dall'architettura che dal compilatore, sono purtroppo **molto poco portabili**, il che rende poco pratico la distribuzione di librerie Fortran così create.

PackagingDrafts/FortranLibraries - Fedora Project Wiki

Library distribution in Fortran Wiki

# Interfacce a librerie Fortran

Per distribuire librerie Fortran vengono quindi prediletti approcci che non prevedano la distribuzione di moduli pubblici già compilati:

**hardmath.F90**

```
program hardmath
   use math
   implicit none
   integer :: x = 1
   call increment(x)
   print '("1 + 1 = ",i0)', x
end program
```

**math.F90**

```
module math
   implicit none
interface
   subroutine increment(x)
      implicit none
      integer :: x
   end subroutine increment
end interface
interface
   subroutine decrement(x)
      implicit none
      integer :: x
   end subroutine decrement
end interface
end module math
```

**math_add.F90**

```
subroutine increment(x)
   implicit none
   integer :: x
   x = x + 1
end subroutine increment

subroutine decrement(x)
   implicit none
   integer :: x
   x = x - 1
end subroutine decrement
```

`libmath.so`

**Alternativa 1**
Distribuire il sorgente del modulo con interfacce alle funzioni di libreria.

# Interfacce a librerie Fortran

Per distribuire librerie Fortran vengono quindi prediletti approcci che non prevedano la distribuzione di moduli pubblici già compilati:

**hardmath.F90**

```
program hardmath
   implicit none
   include 'math.f90'
   integer :: x = 1
   call increment(x)
   print '("1 + 1 = ",i0)', x
end program
```

**math.F90**

```
interface
   subroutine increment(x)
      implicit none
      integer :: x
   end subroutine increment
end interface
interface
   subroutine decrement(x)
      implicit none
      integer :: x
   end subroutine decrement
end interface
```

**math_add.F90**

```
subroutine increment(x)
   implicit none
   integer :: x
   x = x + 1
end subroutine increment

subroutine decrement(x)
   implicit none
   integer :: x
   x = x - 1
end subroutine decrement
```

`libmath.so`

**Alternativa 2**
Distribuire un *header* con le interfacce alle funzioni di libreria.

# Interfacce a librerie Fortran

Per distribuire librerie Fortran vengono quindi prediletti approcci che non prevedano la distribuzione di moduli pubblici già compilati:

**hardmath.F90**

```
program hardmath
   implicit none
   include 'math.f90'
   integer :: x = 1
   call increment(x)
   print '("1 + 1 = ",i0)', x
end program
```

**math.F90**

```
interface
   subroutine increment(x)
      implicit none
      integer :: x
   end subroutine increment
end interface
interface
   subroutine decrement(x)
      implicit none
      integer :: x
   end subroutine decrement
end interface
```

**math_add.F90**

```
subroutine increment(x)
   implicit none
   integer :: x
   x = x + 1
end subroutine increment

subroutine decrement(x)
   implicit none
   integer :: x
   x = x - 1
end subroutine decrement
```

↓

`libmath.so`

**Vecchia scuola (rischioso)**
Semplicemente lasciare perdere le interfacce.

# Per saperne di più...


Home » HPC » Introduction to build systems and package managers in HPC

Registrations closed

WEDNESDAY, 3 JUNE 2026 09:00 TO FRIDAY, 5 JUNE 2026 18:00

**Provided as:** Ordinary Course
**Where:** CINECA - VIA MAGNANELLI 6/3 40033 CASALECCHIO DI RENO, BO, ITALY
**Registrations closing:** Wednesday, 13 May 2026 at 09:00
Enrollment will open approximately three months before the first day of class.

The course is free of charge.
It will be held EXCLUSIVELY IN PRESENCE (No Streaming Available) and will be held in ENGLISH language.

[Introduction to build systems and package managers in HPC | Cineca Events]

# 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. Ottimizzazioni spinte possono non rispettare la semantica del programma. Ottimizzazioni elevate modificano la sequenza delle istruzioni e possono creare difficoltà nella fase di debugging.

Per questo motivo, è consigliabile utilizzare livelli bassi di ottimizzazione quando si è ancora in fase di sviluppo del codice ed il **livello –O0 quando si deve fare debug**.

# Livelli di ottimizzazione

Un compilatore presenta livelli incrementali di ottimizzazione, ad esempio:

**-O0:** (quasi) nessuna ottimizzazione, il codice è tradotto (pressoché) 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.

**-Ofast/-O4/-fast/…:** ottimizzazioni aggressive, da usare con cautela.

# Livello O0

```
$ gfortran --version
GNU Fortran (Spack GCC) 13.2.0
[...]
$ gfortran –O0 -Q --help=optimizers
```

-faggressive-loop-optimizations -fallocation-dce -fasynchronous-unwind-tables
-fauto-inc-dec -fbit-tests -fdce -fearly-inlining -ffp-contract=fast -ffp-int-builtin-inexact
-ffunction-cse -fgcse-lm -finline-atomics -fipa-stack-alignment
-fipa-strict-aliasing -fira-algorithm=CB -fira-hoist-pressure -fira-region=one -fira-share-save-slots
-fira-share-spill-slots -fivopts -fjump-tables -flifetime-dse -flifetime-dse=2 -fmath-errno -fpeephole
-fplt -fprintf-return-value -freg-struct-return -flive-patching=inline-clone
-fsched-critical-path-heuristic -fsched-dep-count-heuristic -fsched-group-heuristic
-fsched-interblock -fsched-last-insn-heuristic -fsched-rank-heuristic -fsched-spec -fsched-spec-insn-heuristic
-fsched-stalled-insns-dep -fschedule-fusion -fsemantic-interposition
-fshort-enums -fshrink-wrap-separate -fsigned-zeros -fsplit-ivs-in-unroller
-fssa-backprop -fsimd-cost-model=unlimited -fstack-reuse=all -fstdarg-opt -ftrapping-math
-ftree-forwprop -ftree-loop-im -ftree-loop-ivcanon
-ftree-loop-optimize -ftree-parallelize-loops=1 -ftree-phiprop
-ftree-reassoc -ftree-scev-cprop -funreachable-traps -funwind-tables
+
-ffrontend-optimize

# Livello O0

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
```

```
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.

**Si utilizza in caso di debugging:** `$ gfortran -g -O0 [...]`

# Livello O1

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

- Eliminazione delle parti mai eseguite del codice (*dead-code elimination*, *DCE*)
- 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 nel livello –O1, vengono eseguite le seguenti operazioni:

- Propagazione della ridefinizione dei valori costanti:

```
A = 10
B = A + 5
```

```
A = 10
B = 15
```

- Eliminazione delle sottoespressioni comuni (*common subexpression elimination*, *CSE*):

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

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

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

E' il livello ottimale, abbastanza ottimizzato e sicuro.

# Livello O2

***Loop unrolling***: permette di ridurre in numero di aggiornamenti del contatore dei loop e delle verifiche 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)
END DO
```

```
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)
END DO
```

# 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
END DO
```

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

# Livello O2

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

```
INTEGER FUNCTION foo(x)
   INTEGER, INTENT(IN) :: x
   foo = x + 1
END FUNCTION
[...]

DO i = 1, 1000000000000000
   j = foo(i)
ENDDO
```

```
DO i = 1, 1000000000000000
   j = i + 1
ENDDO
```

# Livello O3

Questo livello agisce profondamente sui loop attraverso tecniche di *strip mining*, *unrolling*, *interchange*, *fusion*, *distribution*, *blocking*, *unroll & jam*.

***Interchange***:

```
DO i = 1, N
  DO j = 1, N
    DO k = 1, N
      c(i,j) = c(i,j) + a(i,k) * b(k,j)
```

```
DO j = 1, N
  DO k = 1, N
    DO i = 1, N
      c(i,j) = c(i,j) + a(i,k) * b(k,j)
```

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

# Altre opzioni di ottimizzazione

**Trasformazione istruzioni *floating point*:** rimaneggiamento delle operazioni fra float per ottimizzare ulteriormente il codice, possibile perdita di precisione e violazione degli standard IEEE e ISO. Esempio: divisione trasformata in moltiplicazione.

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

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

- Compilatore GNU(13.2):
  `gfortran -Ofast`
  `gfortran -ffast-math`
- Compilatore Intel(v12):
  `ifort -fp-fast=2`
- Compilatore Nvidia(23.9):
  `nvfortran -Mfpapprox`

# Altre opzioni di ottimizzazione

***Prefetching*:** i dati e le istruzioni vengono presi dalla RAM e resi disponibili in cache prima che siano realmente richiesti riducendo la latenza dovuta all'accesso alla memoria.

A seconda dei compilatori tali istruzioni vengono richiamate con *flag* differenti.

- Compilatore GNU(13.2):
  `gfortran -fprefetch-loop-arrays`

- Compilatore Intel(v12):
  `ifort -opt-prefetch=n  n=0, ... 4`

- Compilatore Nvidia(23.9):
  `nvfortran -Mprefetch=plain  (enabled by default)`

# Altre opzioni di ottimizzazione

**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.

- Compilatore GNU(13.2):

```
gfortran -O2 -ftree-vectorize -march=native
gfortran -O3 –march=native
```

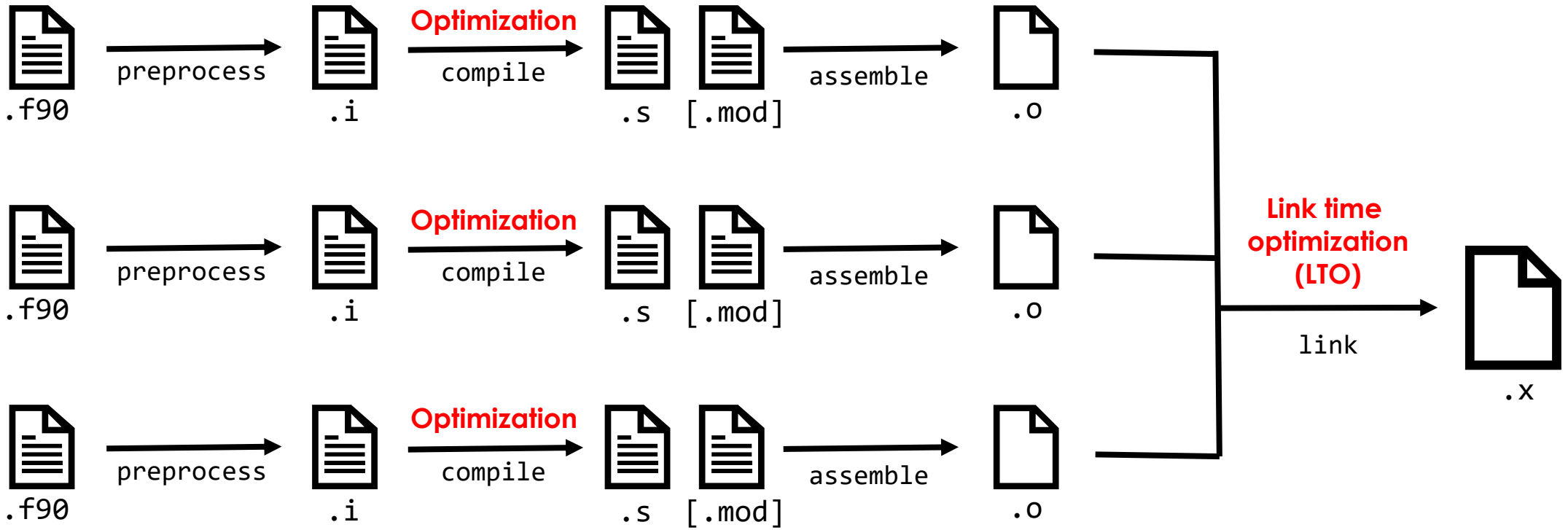
- Compilatore Intel(v12):

```
ifort –xhost
```

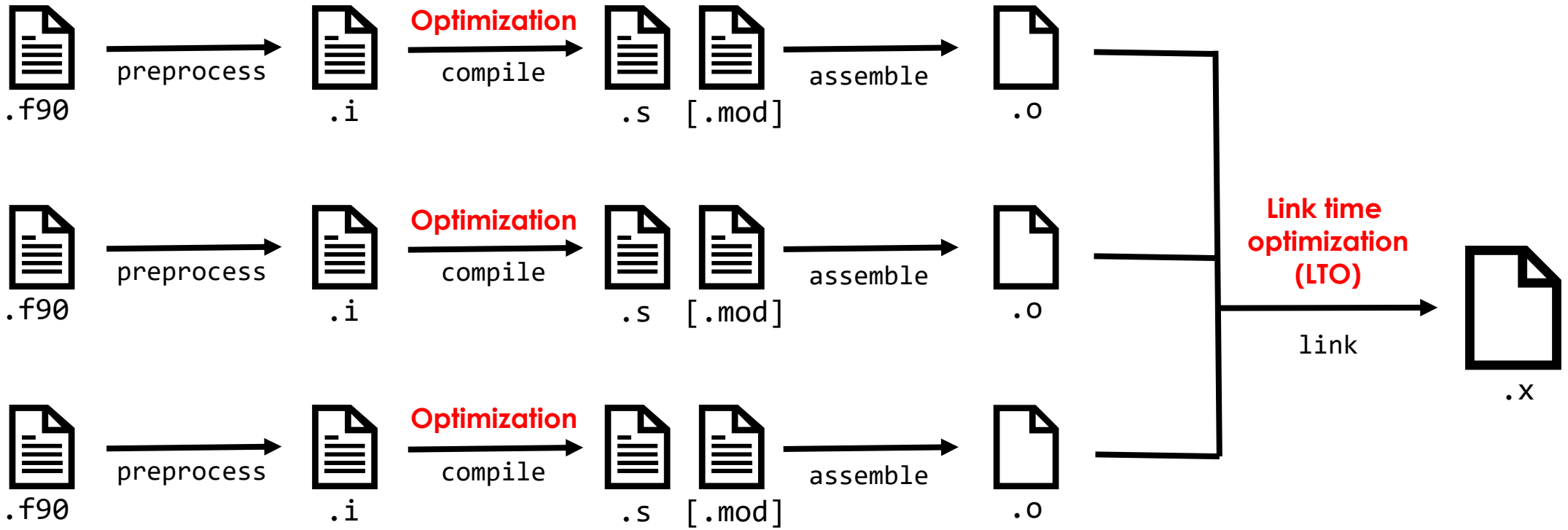
- Compilatore Nvidia(23.9):

```
nvfortran –Mvect=simd
nvfortran -fast
```

# Link time optimization (LTO)

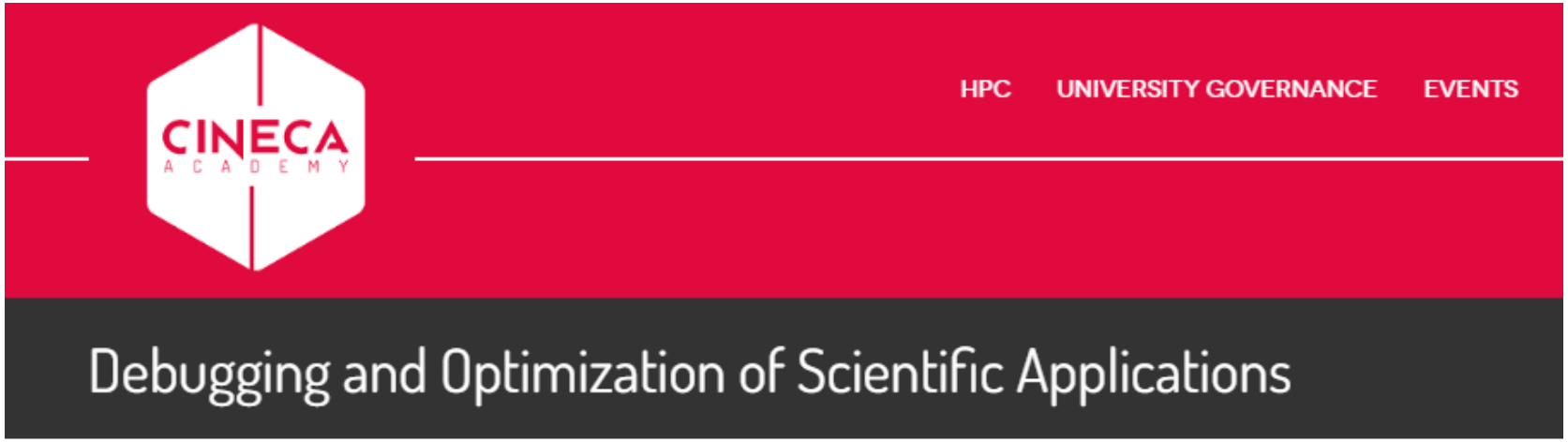


# Link time optimization (LTO)




```
$ gfortran -O2 -flto math.F90 hardmath.F90 -o lto.x
$ gfortran -O2 math.F90 hardmath.F90 -o no-lto.x
$ diff <(nm --format='just-symbols' lto.x) <(nm --format='just-symbols' no-lto.x)
19a20,21
> __math_MOD_decrement
> __math_MOD_increment
```

# Per saperne di più...




Home » HPC » Debugging and Optimization of Scientific Applications

Registrations closed

MONDAY, 16 MARCH 2026 09:00 TO THURSDAY, 19 MARCH 2026 18:00

**Provided as:** Ordinary Course
**Where:** CINECA - VIA MAGNANELLI 6/3 40033 CASALECCHIO DI RENO, BO, ITALY
**Registrations closing:** Sunday, 01 March 2026 at 09:00
**Enrollment will open approximately three months before the first day of class.**

The course is free of charge.
It will be held EXCLUSIVELY IN PRESENCE (No Streaming Available) and will be held in ENGLISH language.

Debugging and Optimization of Scientific Applications | Cineca Events

# Referenze

- Flag di ottimizzazione:

  GNU: Optimize Options (Using the GNU Compiler Collection (GCC))
  Code Gen Options (The GNU Fortran Compiler)

  Intel: Porting Guide for ifort Users to ifx

  Nvidia: HPC Compiler Reference Manual (nvidia.com)

- Creazione di librerie statiche e dinamiche in Fortran (Unix e Windows):
  Managing libraries (static and dynamic libraries) — Fortran Programming Language (fortran-lang.org)

**NB:** i compilatori sono creature cangianti, fare sempre riferimento alla documentazione specifica alla versione del compilatore usata. Per `gfortran` ad esempio:

```
gfortran –O2 -Q --help=optimizers
gfortran –O2 -Q --help=fortran
```

# GRAZIE

**Tommaso Gorni**
t.gorni@cineca.it