Informe Técnico CESGA-2005-004

Opciones del compilador Fortran 90 de HP

J.Carlos Mouriño, Andrés Gómez
CESGA. Centro de Supercomputación de Galicia
e-mail: *jmourino@cesga.es*

Resumen: En este documento se explican algunas de las opciones de compilación que presenta el compilador de Fortran de HP, presente en el sistema Superdome con Itanum2 existente en el CESGA.

Palabras Clave: Compilación, Optimización, Mejora de Rendimiento, Precisión Numérica.

Este trabajo ha sido cofinanciado por el FSE (Fondo Social Europeo)

Identificador documento:	OT_APL_OpcionesHPf90_V1
Fecha:	21/12/05
Autor	José Carlos Mouriño Gallego
Responsable:	Andrés Gómez
Status:	Revisado

Historia de Revisiones

Versión	Autor	Descripción
0	Carlos Mouriño	Desarrollo Informe Técnico

Tabla de contenidos

1 Introducción 4
2 Opciones de Compilación 5
3 Opciones de linkado 13
4 Conclusiones 14

1 Introducción

La compilación de aplicaciones en entornos de Altas Prestaciones (HPC) ha de realizarse adecuadamente para obtener un buen aprovechamiento del sistema. El CESGA realiza la actividad de soporte a sus usuarios para la compilación y el desarrollo de pequeñas aplicaciones de cálculo científico. Dentro de este programa de apoyo, ha realizado un estudio de las opciones de optimización que presenta el compilador de Fortran de HP, presente en el sistema Superdome con Itanum2 existente en el CESGA, para asesorar mejor a los investigadores.

En muchos casos, las opciones de compilación apenas se utilizan, suponiendo que las que el compilador toma por defecto son las adecuadas. Pero nada más lejos de la realidad.
Como ejemplo se ha diseñado un simple programa que realiza la multiplicación de dos matrices, previa aplicación de una función matemática a cada una de ellas.

$$A = \exp(B) \times \sin(C)$$

Para ello se han construido tres bucles y la llamada a una función.
Este programa que ha compilado sin incluir ninguna optimización (+O0) y su tiempo de ejecución ha sido de más de 400 minutos. Con las opciones por defecto se ha reducido su tiempo un 25 % rondando los 300 minutos
Aplicándole algunas opciones de compilación (se han probado varias y hemos seleccionado las que mejores resultados obtenían) se ha reducido este tiempo a unos 20 minutos. Es decir, se ha reducido por un factor de 20 el tiempo de ejecución sin optimización y por un factor de 15 el de las opciones por defecto, todo ello sin variar ni una sola de las líneas del código.

Estas opciones deben utilizarse, ya que se consiguen importantes mejoras en el tiempo de ejecución de un programa, aunque a veces pueden producir errores de precisión numérica.
Por ello, no solo se han de utilizar las opciones de compilación y librerías matemáticas más adecuadas, sino también comprobar el funcionamiento correcto del programa ejecutando los tests disponibles.

2 Opciones de Compilación

Cada opción de compilación se presenta en negrita y justificada a la izquierda. Justo debajo de ella se da una explicación formal de la opción en cuestión, en letra normal.
A continuación, y en letra cursiva, se ofrece alguna explicación adicional de que es realmente lo que hace la opción y en que casos es deseable utilizarla.

+check=all

Chequeo límite arrays en tiempo de ejecución.

Esta opción es deseable en tiempo de desarrollo, pero no debe ser incluida cuando el programa se compila para producción.

+Dsitanium2

Optimiza para el procesador Itanium2.

Opción recomendada, pues optimiza el código generado para la arquitectura.

-dynamic

Linka dinámicamente.

Reduce el tamaña del ejecutable y por tanto su consumo de memoria, por tanto es una opción deseable.
Pero es necesario tener las librerías que utilicemos instaladas en el sistema que pretendemos ejecutar.
Además si se produce la actualización de alguna librería en el sistema puede que pierda compatibilidad.

+save

Guarda todas las variables locales en todos los subprogramas en memoria.
- las no inicializadas se inicializan a cero
- baja el rendimiento

Esta opción reduce el siempre limitado tamaño de stack necesario para ejecución, pero reduce el rendimiento del programa.
Utilizarla solo en caso de que el tamaño máximo de stack resulte insuficiente.

+nosrcpos

No genera información sobre el código fuente para herramientas de análisis de rendimientos (caliper).

Por defecto el compilador genera esta información, por tanto debemos incluir esta directiva si no tenemos pensado utilizar estas herramientas para evitar pérdidas en el rendimiento.

+O0

Apenas realiza optimizaciones. Replaza constantes por valores, evaluación de expresiones en cortocircuito.

Debe utilizarse sólo para depurar el programa, o bien cuando cualquier otra optimización superior da errores de precisión no aceptables.

+O1

Optimizaciones que afectan a pequeñas secciones de código. Optimización de saltos, eliminación de dead code, scheduling de instrucciones, uso de los registros más eficiente

Son pequeñas optimizaciones. Debe utilizarse cuando un nivel de optimización mayor no da resultados aceptables.

+O2, -O

Optimizaciones dentro de un subprograma. Optimización global de registros, reducción de variables de inducción, sustituye algunos productos por sumas, eliminación de expresiones comunes, replaza constantes expresiones por valores, extrae invariantes de bucle, sustituye registros por direcciones de memoria, elimina definiciones no utilizadas, ordena las instrucciones de un bucle, reasociación de registros, desenrollamiento de bucles (los más internos).

Optimización por defecto. Produce resultados de rendimiento bastante aceptables sin apenas perder precisión. Ideal para la fase de depuración pues tanto el tiempo de compilación como el de ejecución son aceptables..

+O3

Transformación de bucles (distribución, intercambio, fusión, vectorización, unroll, paralelización automática), intercambio IF-DO, inlining dentro de fichero, etc.

Opción más agresiva que la anterior, y que produce una mejora bastante considerable en el tiempo de ejecución

+O4

Optimización en tiempo de linkado. Inlining entre ficheros, etc.

Opción recomendada, siempre y cuando el resultado del programa sea el esperado, o por lo menos sea aceptado. El tiempo de compilación se incrementa considerablemente, por tanto no debe utilizarse en la fase de depuración.

+Ofast

Elige varias opciones de compilación intentando minimizar el tiempo de ejecución. En concreto las siguientes:

```
+O2 +Olibcalls +Onolimit +Ofltacc=relaxed +FPD
+Dsitanium2
-Wl,+pi4M,+pd4M,+mergeseg
```

Es una opción muy recomendada, que da unos muy buenos resultados en tiempo de ejecución, sobre todo por las opciones de linkado (las que siguen a -wl)
En caso de que no convenga alguna de las opciones que introduce siempre se puede solapar con su opuesta (siempre después, pues la opción que prevalece es la que se ha declarado en último lugar)
Por ejemplo es muy recomendable utilizar –Ofast +O4, o bien aumentar la precisión numérica con la opción Ofltacc.

+Oall

Aplica la máxima optimización intentando obtener el mayor rendimiento en tiempo de ejecución. Equivale a las siguientes opciones:

```
+Oaggresive +Onolimit
```

Realiza una optimización muy agresiva, que se nota sobre todo a niveles altos de optimización (O3 y O4, sobre todo este último)
Combinando esta opción con +Ofast y +O4 se obtienen unos tiempos de ejecución sorprendentes, aunque también se incrementa el tiempo de compilación.
Como siempre hay que tener cuidado con la precisión del resultado.

+Olibcalls

En caso de existir varias versiones de una rutina en una librería, escoge la que menor sobrecarga produce. Tiene efecto solamente para un nivel de optimización O2 o superior.

En la práctica no se han notado mejoras considerables de tiempo utilizando esta opción, aunque todo depende de que librerías estemos utilizando.

+Onolimit

No evita optimizaciones que incrementen en gran medida el tiempo de compilación o la memoria consumida. Tiene efecto solamente para un nivel de optimización O2 o superior.

Por defecto, el compilador evita una optimización si para ello consume mucha memoria o tiempo. Esto puede ser útil en tiempo de depuración del código. Pero debe incluirse esta directiva para la compilación final del programa, pues toda optimización que el compilador pueda realizar es bienvenida.

+Osize

Evitar optimizaciones que aumenten el tamaño del ejecutable. Tiene efecto solamente para un nivel de optimización O2 o superior.

Es lo mismo que en el caso anterior pero para el tamaño de ejecutable. La diferencia es que la opción por defecto es permitir las optimizaciones. En caso de querer evitarlo debemos incluir esta opción en la compilación.

+FPD

Permite overflow y pone a cero todos los valores desnormalizados.

Es una opción bastante arriesgada. Indica al programa que no detenga su ejecución en caso de overflow. Los valores fuera de rango o desnormalizados son puestos a cero.
Un programa nunca debe dar overflow en sus operaciones, y si así fuera debe corregirse, por tanto esta opción no es deseable, y mucho menos en tiempo de depuración.
Existen otras variantes de esta opción cambiando la última letra. Consultar el manual.

+Ocache_pad_common

Ordena los bloques common para evitar colisiones en la caché. Tiene efecto solamente para un nivel de optimización O3 o superior

Esta ordenación implica introducir datos nulos en los bloques common, ampliando la memoria que ocupan. Si se utiliza de be hacerse en todas las rutinas que utilicen el bloque common en cuestión.

+Ocxlimitedrange

Permite utilizar la unidad de punto flotante en la unidad de compilación.

Esto permite al compilador calcular en tiempo de compilación algunas expresiones con constantes en punto flotante.

+Ofastaccess

Habilita el acceso rápido a datos globales. Tiene efecto solamente para un nivel de optimización O1 o superior.

Este acceso rápido se consigue manteniendo las variables globales en niveles superiores de la jerarquía de memoria. La opción por defecto es que no, y en la práctica no se han conseguido mejoras de tiempo considerables utilizando esta opción.

+Onofltacc, +Ofltacc=relaxed

Habilitar optimizaciones en las operaciones de punto flotante.

Mejora el rendimiento pero a costa de perder precisión. Dependiendo si esta pérdida es aceptable o no se debe incluir esta opción. Por defecto no realiza estas optimizaciones.

+Onoloop_transform

No permitir transformaciones en los bucles (unroll, fusión, etc.).

Hay una serie de opciones, algunas por defecto, que realizan transformaciones en los bucles que mejoran su rendimiento.
Debemos incluir esta opción si queremos que estas transformaciones no se produzcan. De todas formas se pueden controlar estas transformaciones de manera individual con otras opciones.

+Onoinitcheck

No permitir la inicialización implícita de variables. Tiene efecto solamente para un nivel de optimización O2 o superior

Por defecto, las variables no inicializadas las inicializa el compilador (en principio a cero, aunque depende del tipo de variable).
En un programa no se deben utilizar variables no inicializadas, por tanto debemos permitir que el compilador nos avise cuando esto ocurre durante la fase de desarrollo del código.

+Oaggressive

Aplicar optimizaciones agresivas. Tiene efecto solamente para un nivel de optimización O2 o superior. Esta opción está "deprecated". Incluye las siguientes opciones.
`+Oentrysched +Olibcalls +Onofltacc +Onoinitcheck +FPD (+Ovectorize)`

Puede mejorar mucho el rendimiento, pero también puede alterar el comportamiento del programa mediante la pérdida de precisión.

En caso de que esta pérdida sea aceptable es siempre deseable incluir esta opción.

-K

Igual que +save. Usa almacenamiento estático en lugar de la pila.

Tiene exactamente el mismo efecto que la opción +save explicada anteriormente.

+Ofailsafe

Habilita la optimización segura. Tiene efecto solamente para un nivel de optimización O2 o superior.

En caso de que la compilación falle en el actual nivel de optimización, se reinicia automáticamente con un nivel más seguro (+O1 para el Itanium).

+Oloop_unroll(=n)

Desenrolla los bucles por un factor de n. Tiene efecto solamente para un nivel de optimización O2 o superior.

El desenrollamiento de bucles es una técnica que consiste en hacer varias iteraciones de un mismo bucle en una sola iteración, evitando así saltos en el código objeto final. El número de iteraciones que se realizan de una vez es el nivel de desenrollamiento(n).

Por ejemplo, el bucle:

```
Do i=1,n
     A(i)=B(i) + C(i)
End do
```

Quedaría de la siguiente forma si lo desenrollamos con un factor de 3.

```
Do i=1,n,3
     A(i)=B(i)+C(i)
     A(i+1)=B(i+1)+C(i+1)
     A(+ci)=B(i+2)+C(i+2)
End do
```

El compilador desenrolla algunos bucles de manera automática en niveles de optimización superiores a O2. El factor de desenrollamiento que utiliza es típicamente 4. Con esta opción podemos indicar un nuevo valor para este factor o bien indicar al compilador que no desenrolle los bucles (+Onoloop_unroll ó +Onoloop_transform).

Por lo general es la opción por defecto la que suele dar unos mejores resultados en tiempos de ejecución.

+Oinline_budget(=n)

Establece el nivel de agresividad (n) del inlining. Tiene efecto solamente para un nivel de optimización O3 o superior.

El inlining es una técnica que consiste en sustituir la llamada a una función por el cuerpo de dicha función. Esto evita saltos en el código objeto final, aunque incrementa el tamaño del ejecutable al tener que replicar el código varias veces.
El compilador realiza o no el inlining de una función dependiendo de las características de esta (tamaño, si tiene a su vez llamadas, número de veces que se llama, etc.) y teniendo en cuenta un factor de agresividad (n). A mayor valor de este factor se hará inline de un mayor número de funciones.
Con el factor de agresividad más bajo (n=1) se han conseguido en la práctica los mejores resultados.
En vez de un valor de agresividad se le puede indicar explícitamente los nombres de las funciones de las que deseamos hacer el inlining, separados por comas, o bien de las que no queremos hacerlo (anteponiendo la palabra no a inline).
Veamos a continuación un ejemplo de inlining. El código:

```
.....
a=media(b,c)
...

function media (b,c)
integer b,c
real a
a=(b+c)/2
return a
end
```

quedaría de la siguiente forma:

```
...
a=(b+c)/2
...
```

+I

Compila para que genere en tiempo de ejecución información de profile.

Con esta opción el programa genera información de profile durante su ejecución. Esta información puede ser aprovechada por programas que analizan el profile. Para utilizar esta opción no debe indicarse ningún nivel de optimización (+O0).

+P

Optimización basada en información de profile

Compilar teniendo en cuenta la información de profile generada con la opción anterior. Para ello debemos haber compilado el programa con +I y haberlo ejecutado por lo menos una vez. Sabiendo en tiempo de compilación cual va a ser la traza del programa durante su ejecución se pueden intentar ordenar mejor las instrucciones.
Por lo general no se suelen conseguir grandes mejoras con esta opción, aunque en algunos casos puntuales la mejora de tiempos es considerable (programas deterministas con muchas sentencias condicionales y saltos en el código).

-g

Genera información necesaria para programas de debug. No válida para niveles de optimización O3 y O4

Esta opción es muy útil en tiempo de desarrollo para corregir errores en tiempo de ejecución. Genera la información necesaria para los programas que realizan el debug del programa. Es recomendable no utilizar ninguna optimización (+O0).
Esta opción debe eliminarse cuando se ha terminado la fase de debug, pues ralentiza mucho la ejecución.

+Onodataprefetch

No insertar instrucciones en los bucles más internos para precargar explícitamente datos de memoria en caché.

Por defecto, el compilador introduce instrucciones para precargar datos antes de operar sobre ellos. El prefetch suele mejorar el rendimiento de un programa, por lo que la opción por defecto es la deseable.
(Algún estudio reciente indica que el prefetch no es deseable en aplicaciones científicas de cálculo intensivo)
En caso de utilizar prefecth, se le puede indicar para que tipo de accesos a memoria queremos realizarlo, bien los accesos directos o bien los indirectos (+Odataprefetch=direct,indirec). Por defecto realiza prefetch para referencias indirectas a memoria.

+prefetch_latency(=num)

Se aplica a aquellos bucles para los que el compilador aplica data prefetch. Indica con que anticipación (num en ciclos) deben ser precargados los datos. Tiene efecto solamente para un nivel de optimización O2 o superior.

En caso de querer precargar los datos antes de ejecutar los bucles, con esta opción se le puede indicar con cuanta anticipación se desea realizar la precarga.

Por defecto utiliza 450 ciclos para accesos a punto flotante y 150 para enteros. Se pueden probar otros valores y ver con cual se obtiene un mejor rendimiento. Experimentalmente se ha obtenido una leve mejora de tiempo utilizando 960 ciclos, pero este valor es totalmente dependiente de la aplicación.
Por lo general, en aplicaciones paralelas se obtiene un buen rendimiento con valores menores, del orden de 150 ciclos.

-Wl

Para introducir opciones de linkado.

Con esta opción se le pueden pasar opciones al linkador. Es útil cuando se compila y se linka con la misma instrucción. Las opciones se le pasan, separadas por comas, justo a continuación de esta opción. Algunas opciones de linkado se detallan en la siguiente sección.

3 Opciones de linkado

+pd(size)

Tamaño página de datos.

Establece el tamaño de página que el operativo debe utilizar en memoria.
La opción por defecto es de 4K, pero el rendimiento mejora considerablemente seleccionando un tamaño de página mayor (4M o 16M). La opción +Ofast establece un tamaño de página de datos de 4M.

+pi(size)

Tamaño página de instrucciones.

Lo mismo que en el caso anterior pero para la página de instrucciones. Experimentalmente no se han conseguido mejoras variando este parámetro.

+mergeseg

Permite mezclar ambos segmentos.

Permite que se mezclen en memoria el segmento de datos y el de instrucciones (por defecto están separados). No se han experimentado mejoras utilizando esta opción.

4 Conclusiones

En este documento se han revisado algunas las opciones de optimización del compilador HP Fortran , presente en el sistema Superdome con Itanum2 existente en el CESGA.

Las opciones aquí descritas son las que se han considerado más importantes y/o útiles para el investigador que desee compilar sus propios programas. Existen muchas otras opciones. En caso de querer consultar alguna opción no incluida en este documento, o ampliar la información sobre alguna de las opciones, el lector puede encontrar más información en la web (http://doc.cesga.es/superdome/Programming.html), incluido el manual del compilador. También puede resultar útil la ayuda que facilita el manual del sistema (`man f90`).

Estas opciones deben utilizarse con precaución, pues algunas de ellas hacen variar la ejecución del programa y/o pueden producir errores de precisión numérica. Algunas de ellas también incrementan el tiempo de ejecución.

Es por ello que deben probarse varias y seleccionarse aquellas que ofrecen un mejor resultado. Como recomendación final, y tras mi experiencia en la compilación de programas, recomiendo utilizar las siguientes opciones:

```
+Ofast +Oall +O4 +nosrcpos
```

ya que incluyen la mayoría de las opciones de optimización.
Utilizando simplemente estas opciones en la compilación del programa de ejemplo, el tiempo de ejecución ha sido de poco más de 20 minutos, apenas por encima del mejor tiempo alcanzado.
Recalcar por último que este nivel tan alto de optimización trae consigo, muchas veces, errores de precisión numérica. Estos errores pueden, en muchas ocasiones, inadmisibles, teniendo que reducir el nivel de optimización en la compilación