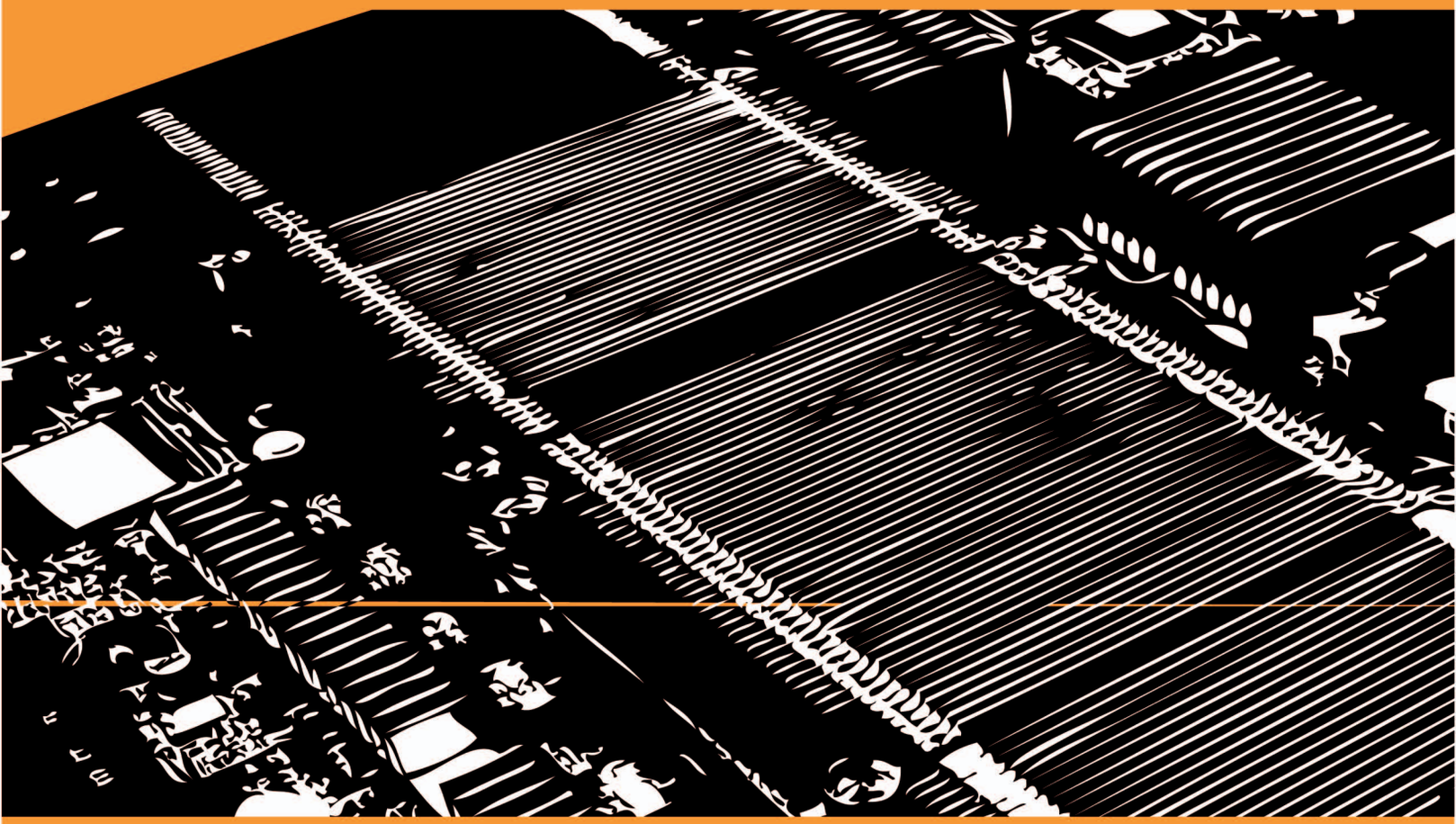
del 30 de junio al 4 de julio de 2008

CURSO

MATEMÁTICA COMPUTACIONAL

Compilación, ejecución y optimización de programas



Organizado por:

i·ingenio i·math mathematica

CESGA

Colaboran:

Grupo de Arquitectura de Computadores - UDC

Red Mathematica Consulting & Computing de Galicia

Red Gallega de Bioinformática

Cofinanciado por:

Fondo Europeo de Desenvolvemento Rexional Iniciativa Comunitaria

GOBIERNO DE ESPAÑA

MINISTERIO DE CIENCIA E INNOVACIÓN

CESGA

Matemática Computacional: Compilación, ejecución y optimización de programas

Patricia González Gómez
pglez@udc.es
Grupo de Arquitectura de Computadores
Dpt. Electrónica y Sistemas
Universidade da Coruña

Introducción

- Nivel del curso: básico
- Dirigido a: Investigadores de Ingenio MATHEMATICA que desarrollen sus propios códigos usando lenguaje C y/o Fortran, y quieran mejorar sus tiempos de respuesta.

Contenidos del curso

- Tema 0: Visión general y organización del curso
- Tema 1: Compilación y linkado de programas
  - Descripción del proceso de compilación/linkado
  - Compiladores C y Fortran
  - Descripción y uso de las opciones de compilación
  - Descripción y uso de las opciones de optimización
  - Llamadas a subrutinas Fortran desde C y viceversa

Contenidos del curso

- Tema 2: Aritmética del computador y sus implicaciones en computación matemática.
  - Formatos de números enteros
  - Formato de números en punto flotante IEEE 754
  - Rango, precisión y redondeo
  - Excepciones en punto flotante: overflow, underflow, NaN
  - Big endian - little endian
  - Ejemplos aplicados a computación matemática

Contenidos del curso

- Tema 3: Utilización de librerías matemáticas
  - Definición y uso de librerías
  - Librerías estáticas y dinámicas
  - Librerías matemáticas (BLAS,LAPACK,…)
  - Creación y manipulación de librerías propias

Contenidos del curso

- Tema 4: Optimización del rendimiento de la jerarquía de memoria
  - La jerarquía de memoria: principio de localidad
  - Fundamentos del funcionamiento de la memoria caché: carga, ubicación, reemplazo de líneas
  - Técnicas de optimización software: intercambio de bucles, fusión de bucles, partición en bloques, …

Contenidos del curso

- Tema 5: Depuración y evaluación del rendimiento
  - Herramientas de depuración
  - Evaluación del rendimiento durante el proceso de desarrollo
  - Herramientas para la evaluación del rendimiento

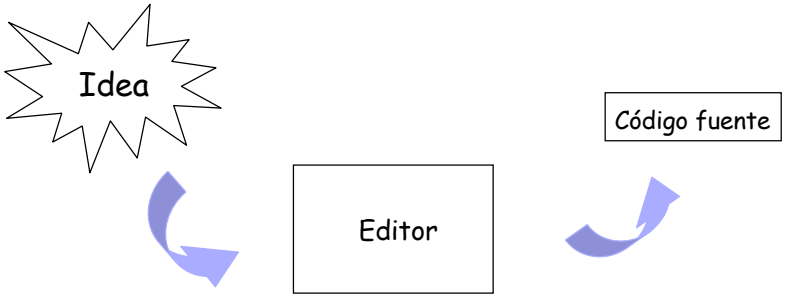
Contenidos del curso

- Tema 6: Introducción a la computación paralela
  - Clasificación de las arquitecturas paralelas
  - Paradigmas de programación paralela
  - Elección del paradigma de programación paralela
  - Ejemplos de aplicaciones de introducción a la computación, paralelización y optimización.

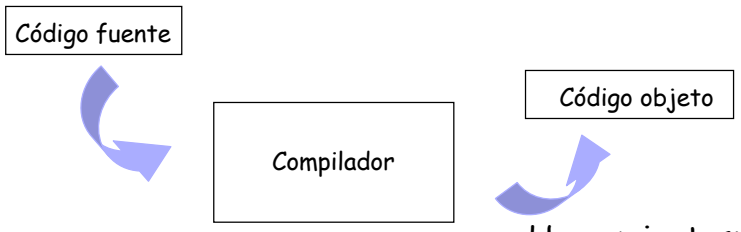
Ciclo de vida en el desarrollo de programas

- Creación
- Compilación
- Linkado
- Ejecución y análisis
- Depuración

Fase de creación



Fase de compilación

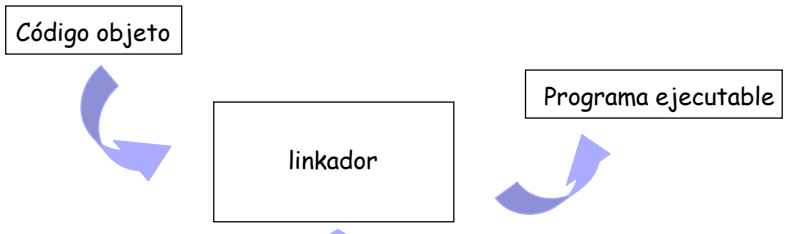


Compiladores:
- gnu gcc
- gnu g77
- hp cc
- hp f90

} SuperDome

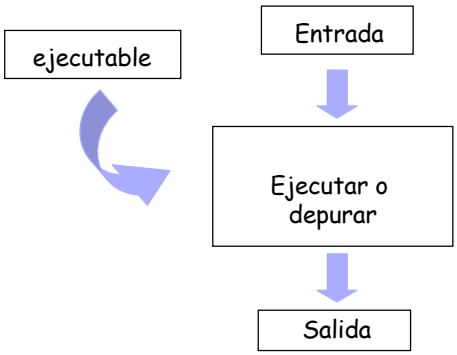
Herramientas:
- make

Fase de linkado



Linkador:
- ld

Fase de ejecución y depuración



Depuradores:
•GDB

•WDB (SuperDome)

Manuales de programación

- Manuales/Tutoriales de C:
  - Aprenda informática como si estuviera en Primero. http://www.tecnun.es/asignaturas/Informat1/AyudaInf/
    - Apuntes de ANSI C, C++, Java, etc.
  - Programming in C. http://www.cs.cf.ac.uk/Dave/C/
- Manuales/Tutoriales de Fortran:
  - The Fortran Market. http://www.fortran.com/fortran/F77_std/rjcnf0001.html
  - User Notes on Fortran Programming. http://sunsite.informatik.rwth-aachen.de/fortran/
- Manuales de los compiladores libres de GNU:
  - http://gcc.gnu.org/onlinedocs/

Manuales de programación

- Manuales en el HP SuperDome:
  - En http://docs.hp.com/hpux/os/11.0 encontrareis todas las guías de usuario y otros manuales necesarios relacionados con HP-UX 11.0
    - HP-UX Reference Manual (sección 1)
    - Programming on HP-UX
    - HP-UX Software Developer´s Guide
    - ...

Compilación y ejecución de programas

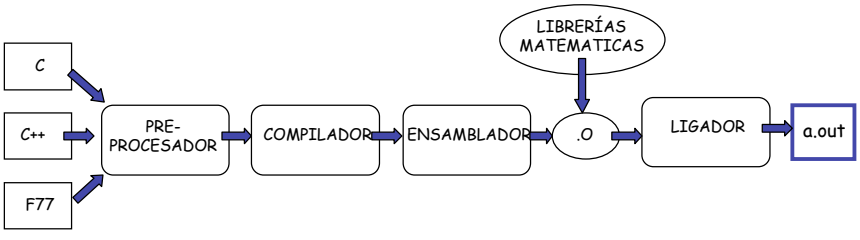
Grupo de Arquitectura de Computadores
Dpt. Electrónica y Sistemas
Universidade da Coruña

Compilación y enlace

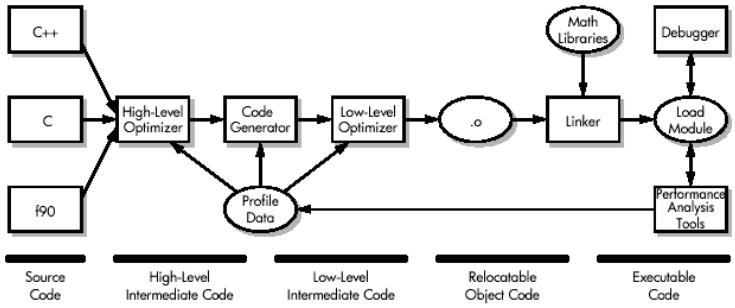
- Objetivos:
  - Dar una visión general de la importancia del proceso de compilación
  - Describir el proceso de compilación/enlace
  - Describir y usar los compiladores para C y Fortran
  - Identificar los ficheros involucrados en el proceso
  - Definir y usar opciones de los compiladores
  - Invocar opciones de optimización
  - Describir los pasos necesarios para crear un programa ejecutable desde códigos fuente en diferentes lenguajes.

El modelo de compilación

Proceso de compilación y enlazado



Compiladores HP para Itanium



Estructura de los compiladores HP para sistemas basados en Itanium®

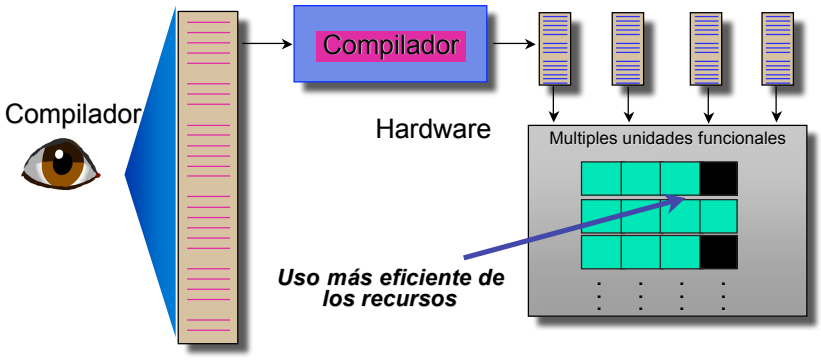
El modelo de compilación

- El preprocesador
  - Acepta código fuente como entrada
  - Es responsable de:
    - Quitar comentarios
    - Interpretar directivas del preprocesador
      - #include <math.h>
      - #define TAM_MAX_MATRIZ 100
- El compilador
  - Traduce código fuente en código ensamblador
  - El código fuente es recibido del preprocesador

El modelo de compilación

- El ensamblador
  - Traduce el código ensamblador a código máquina
  - Crea los ficheros objeto (.o)
- El enlazador o ligador
  - Combina las funciones propias del programa con funciones de librerías a las que hace referencia este, que están definidas en otros ficheros. Resuelve referencias externas.
  - Crea un ejecutable.

Visión general de la compilación



Incremento del paralelismo

Compiladores disponibles

- Cada sistema tiene sus propios compiladores, puesto que cada arquitectura necesita compiladores específicos para ella.
- GCC (GNU Compiler Collection) conjunto de compiladores de los lenguajes más importantes (C, C++, Fortran, Java, Ada, Pascal y Cobol)
  - Todos ellos son compiladores nativos, existen versiones para los procesadores y máquinas más importantes: Intel, DEC, HP, Motorola, SPARC, …
- En este curso manejaremos códigos escritos en lenguaje C y Fortran
  - Compilador para C: hp cc (por defecto en el SuperDome) ó gnu gcc
  - Compilador para Fortran: hp f90 (en el SuperDome) o gnu g77

Ficheros involucrados

- Sufijos:
  - .c fichero con código fuente en C
  - .f fichero con código fuente en Fortran
  - .f90 fichero con código fuente en Fortran 90
  - .i / .i90 ficheros con código fuente preprocesado
  - .s fichero con código ensamblador
  - .o fichero con código objeto
  - .a fichero de librería (*archive*)
  - .sl fichero de librería (*sharable*)
  - .out fichero ejecutable creado por defecto

Ejemplos

- >gcc main.c sub1.o sub2.i mylib.a
  - genera a.out
- >g77 main.f sub1.o sub2.f mylib.a
  - genera a.out
- >cc main.c sub1.o sub2.i mylib.a
  - genera a.out
- >f90 main.f sub1.o sub2.f mylib.a
  - genera a.out

Opciones de compilación

- Genéricas y comunes a los compiladores de C y Fortran
  - -c suprime la fase de linkado y crea ficheros .o como salida
  - -g genera información adicional necesaria para el depurador
  - -I dir añade este directorio al camino de búsqueda de los ficheros especificados en las líneas de include.
  - -lx busca la librería denominada libx
  - -L dir añade este directorio al camino de búsqueda de las librerías
  - -o outfile nombra el fichero de salida ejecutable como outfile
  - -O invoca al optimizador
  - -p prepara ficheros objeto para usar con el perfilador prof(1)
  - -P ejecuta sólo el preprocesador, creando un fichero .i
  - -S ofrece la salida en lenguaje ensamblador en los ficheros .s
  - -v produce una descripción paso a paso del proceso del compilación

Ejemplos

- >gcc prog.c –c –s
- >cc prog.c –g –lm –o prog
- >cc prog.c –p –o prog
- >g77 prog.f –L /usr/lib/X11R6 –lX11 –o prog
- >f90 prog.f -P

Niveles de optimización

Opción	descripción
-O0	No realiza optimizaciones. El tiempo de compilación es pequeño.
-O1	Nivel de optimización por defecto. Planificación de instrucciones y optimizaciones que se pueden realizar en secciones pequeñas de código. Eliminación de código muerto.
-O2	O1 + optimizaciones realizadas sobre funciones enteras en un solo fichero. Optimiza lazos para reducir las paradas del pipeline. Realiza análisis de flujo de datos, uso de memoria, lazos y expresiones
-O3	O2 + optimización a través de todas las funciones de un fichero. Incluye inlining dentro de un fichero. (O3 y superiores necesitan datos PBO)
-O4	O3 + optimizaciones a través de todas las funciones de la aplicación. Incluye optimización de variables globales y estáticas e inlining en todo el programa. (Disponible con la opción +P)

Ejemplos

- >gcc -O0 prog.c –o prog optimización nivel 0
- >cc -O1 prog.c –o prog optimización nivel 1
- >cc -O prog.c –o prog optimización nivel 2
- >cc -O2 prog.c –o prog optimización nivel 2
- >f90 -O2 prog.f –o prog optimización nivel 2

Optimizaciones basadas en perfil

- Funcionalidad de los compiladores HP para Itanium:
  - Gran impacto en el rendimiento del procesador Itanium
    - +Oprofile=collect marca los programas para recoger datos PBO
    - +Oprofile=use*[:file]* optimiza el código usando datos PBO
  - Estudio del comportamiento de los saltos en una aplicación
  - La información recogida en el perfil se tiene en cuenta para guiar el proceso de optimización
    - Predicación
    - Especulación
    - inlining/cloning de procedimientos
    - Transformaciones a nivel de lazo

El enlazador ld

- **ld** toma uno o más ficheros objeto o librerías como entrada y los combina para producir un único fichero (generalmente un ejecutable).
  - Resuelve referencia externas
  - Asigna direcciones finales
  - Actualiza información simbólica para el depurador
  - Produce ficheros ejecutables
- Reconoce:
  - Ficheros .o
  - Ficheros .a
  - Ficheros .so

Ejemplos

- >cc –c prog.c
- >ld /usr/ccs/lib/crt0.o prog.o –lc –o prog
- >ld –r file1.o file2.o –o prog.o
- >ld /usr/ccs/lib/crt0.o prog.o –L . –lfunc –a archive –lc

Enlazado incremental

```
>cc -c *.c
>ls *.o
   a1.o
   a2.o
   b1.o
>ld -r a*.o -o A.o
>ld -r b*.o -o B.o
>cc A.o B.o -o prog
```

Más tarde....

```
>cc -c a2.c
>ld -r a*.o -o A.o
>cc A.o B.o -o prog
```

Programación con varios lenguajes

C: main.c

```
main()
{
  _____
  _____
  _____
  _____
  sub1();
  func2();
  _____
  _____
}
```

Fortran: sub1.f

```
subroutine sub1
  _____
  _____
  _____
  _____

END
```

C: func2.c

```
func2()
{
  _____
  _____
  _____
  _____

}
```

Programación con varios lenguajes

- La mezcla de Fortran y C no está estandarizada
- Las reglas pueden variar entre plataformas distintas
- Llamadas a funciones C desde códigos Fortran:
  - no hay que hacer cambios, pero el nombre de la función en el código C ha de terminar en guión bajo (_)
- Llamadas a funciones Fortran desde códigos C:
  - añadir un guión bajo al nombre de las funciones Fortran en la llamada que se hace en el código C

Programación con varios lenguajes

- Cuestiones a tener en cuenta al mezclar Fortran y C:
  - Paso de parámetros
    - En Fortran por referencia, en C por referencia o por valor
  - Índice de los arrays
    - En Fortran comienzan en 1, en C comienzan en 0
  - Punteros
    - Se pasan como puntero a puntero
  - Entrada/salida
    - Fortran usa canales de E/S identificados con números, que se traducen en descriptores de ficheros, C usa subrutinas de E/S
    - No se recomienda mezclar E/S entre Fortran y C, es decir, solo uno de ellos debería de ser el responsable de la E/S
    - Problema: entrada/salida en Fortran cuando las funciones se llaman desde C: en el superdome la solución es compilar usando la librería /usr/lib/pa20_64/libcl.sl

Programación con varios lenguajes

- Ejemplos:
  - >cc –c main.c
  - >f90 –c sub1.f
  - >cc –c sub2.c
  - >cc main.o sub1.o sub2.o –o main

  - >f90 –c main.c
  - >f90 –c sub1.f
  - >cc –c sub2.c
  - >f90 main.o sub1.o sub2.o –o main

Sesión de laboratorio

Lab 1

Documentación

- GCC: The GNU Compiler Collection
  http://gcc.gnu.org/
- From Source to Binary: The Inner Workings of GCC
  http://www.redhat.com/magazine/002dec04/features/gcc/

Conclusiones

- El proceso de compilación tiene una importancia especial
- El impacto de la compilación en el rendimiento de una aplicación puede ser grande
- Elegir un buen compilador y conocer las opciones de compilación y, en especial, las de optimización automática, que nos ofrece es importante

Aritmética del computador

Grupo de Arquitectura de Computadores
Dpt. Electrónica y Sistemas
Universidade da Coruña

Aritmética del computador y sus implicaciones en computación matemática

- Objetivos:
  - Conocer la representación de los datos numéricos en el computador: punto fijo, punto flotante
  - Conocer las implicaciones de estos formatos en rango y precisión
  - Conocer la problemática del redondeo
  - Conocer las implicaciones de la representación big-endian, little-endian

Introducción

- El mundo de los computadores es un mundo binario
- Flujos de información en el computador:
  - Flujo de datos
  - Flujo de instrucciones máquina
- La forma de representar datos e instrucciones es uno de los atributos más importantes de la arquitectura de un computador.
- Su definición es uno de los aspectos más importantes en el diseño de un computador.

Introducción

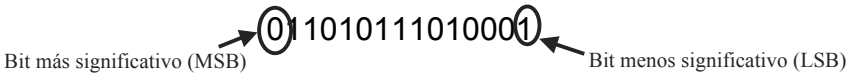
- En la selección de una representación para los datos en un computador se deben tener en cuenta varios factores:
  - Representación numérica:
    - Tipos de números a representar (enteros, reales, etc.)
    - Rango de los valores posibles
    - Precisión
    - Costo hardware requerido para almacenar y procesar los números
  - Representación alfanumérica

Introducción

- Formatos principales de la representación numérica:
  - Punto fijo:
    - Rango limitado de valores
    - Requerimientos hardware simples
  - Punto flotante:
    - Mayor rango de valores
    - Procesamiento más costoso

Sistemas de numeración

- Sistema de numeración binario:
  - Base: 2
  - Dígitos: 0 y 1 (denominados bits)
  - Una cantidad N se representa mediante una secuencia de bits



- Otros sistemas de numeración útiles:
  - Hexadecimal:
    - Base: 16
    - Dígitos: 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F

Representación en punto fijo

- Todos los números a representar tienen exactamente la misma cantidad de dígitos y la coma fraccionaria está siempre en el mismo lugar:

  $11{,}10\ (3{,}5)_{10}$ $01{,}10\ (1{,}5)_{10}$
- Rango: diferencia entre el número mayor y el menor
- Resolución: diferencia entre dos números consecutivos
  - En el ejemplo anterior en sistema decimal:
    - Rango: [0…9,99]
    - Resolución: 0,01
- Notar que hay un compromiso entre rango y resolución
  - Si mantenemos 3 dígitos y desplazamos la coma dos lugares a la derecha, el rango pasa a ser [0…999] y la resolución pasa a ser 1.

Representación en punto fijo

- Para poder representar enteros se necesita una representación que distinga números positivos y negativos. En punto fijo es posible representar un rango de enteros positivos y negativos.
  - Signo magnitud
    - MSB representa el signo del número, el resto de los bits representan el valor absoluto en binario natural

      $(0110)_2=(+6)_{10}$ $(1110)_2=(-6)_{10}$
  - Complemento a uno
    - Los positivos se representan por su valor absoluto en binario natural, los negativos se representan por el complemento a 1

      $(0110)_2=(+6)_{10}$ $(1110)_2=(-1)_{10}$

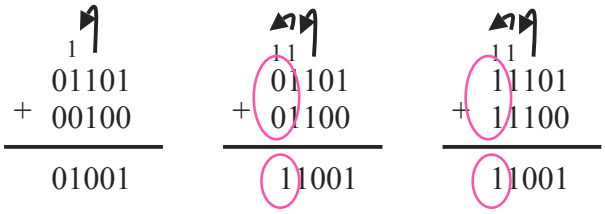
Problema de ambos: la doble representación del cero

Representación en punto fijo

- Complemento a dos:
  - Los positivos se representan por su valor absoluto en binario natural y los negativos por su complemento a dos. $(0110)_2=(+6)_{10}$ $(1110)_2=(-2)_{10}$
  - Ventajas:
    - las operaciones de suma y resta no necesitan realizar ajustes para obtener el resultado de la operación, por lo tanto este es el convenio más usado cuando se trata de operar con números enteros
    - el cero tiene una única representación

Aritmética en punto fijo

- Se produce ***desbordamiento*** cuando el resultado es mayor que el número de bits que se pueden almacenar



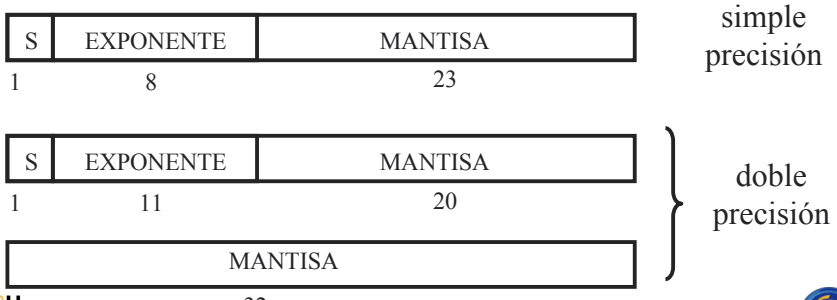
Representación en punto flotante

- Limitaciones de los números en punto fijo:
  - Parte fraccionaria: 3.1415
  - Números muy pequeños: 0.0000001
  - Números muy grandes: $3.16x10^9$
- Notación científica: $M\text{x}10^E$
- Aplicada a números binarios:
  - Signo (S)
  - Mantisa (M)
  - Exponente (E)

$$(-1)^S \text{xMx}2^E$$

Formato IEEE 754

- La representación IEEE 754 se utiliza en prácticamente todos los procesadores y coprocesadores aritméticos actuales.
- Formatos:



Formato IEEE 754

- Representación normalizada:

$$\pm 1.M \times 2^{e}$$

- El formato hace implícito el 1 de la parte entera.
- El exponente se codifica en representación por exceso, siendo el exceso 127 para precisión simple y 1023 para precisión doble.

$$(-1)^{S} \text{x} 1.\text{Mx} 2^{E-\text{exceso}}$$

Formato IEEE 754

- Símbolos especiales:

Exponente	Mantisa	Objeto
00000000	00...00	CERO
00000000	10...11	número no normalizado
11111111	00...00	Infinito
11111111	11...11	NaN

Aritmética en punto flotante

- Se produce desbordamiento superior (overflow), cuando el resultado es superior al mayor número normalizado que se puede representar en el estándar.
- Se produce desbordamiento inferior (underflow), cuando el resultado está estrictamente entre cero y el menor número normalizado que se puede representar en el estándar.
- Es decir, el exponente no se puede representar con los bits que tiene asignados en el formato.

Redondeo

- En un computador los números se representan con un número finito de cifras
- Ciertos números no se pueden representar exactamente y al aproximarlos se comete un error
- En muchos casos, los resultados de las operaciones aritméticas tienen más cifras de las que se puede almacenar y hay que eliminar las cifras menos significativas. A este proceso se llama **redondeo:**
  - **Redondeo simétrico:** se aproxima al número más cercano
  - **Redondeo truncado**: se eliminan las cifras que no se pueden almacenar

Redondeo

- Los errores de redondeo son inevitables, pero con frecuencia controlables:
  - En muchas ocasiones son poco significativos y no tienen ninguna importancia
  - En otros problemas pueden llegar a destruir por completo el significado de un resultado. Conviene detectar estos casos y tomar las medidas adecuadas.
  - Unos errores de redondeo catastróficos pueden ser consecuencia de un problema difícil, un mal algoritmo, o ambas cosas a la vez.

Perdida de precisión

- Existen formas de realizar los cálculos que implican pérdidas de precisión y que pueden ser sustituidas por otras equivalentes más precisas
  - Sumas de series de datos
  - Aumento innecesario del rango dinámico de los datos
  - Utilización de constantes inexactas

Perdida de precisión

- Sumas de series de datos.
  - Ejemplo: Suma de Taylor. Es más preciso sumar de menor valor absoluto a mayor

```
float s1, SS[20];
SS[0]=1;
for(i=1; i<20; ++i)
      SS[i]=SS[i-1]/2.5;
s1=0.0;
for(i=0; i<20; ++i)
      s1 += SS[i];
printf("Sumando de mayor a menor: %.12f\n", s1);
s1=0.0;
for(i=19; i>=0; --i)
      s1 += SS[i];
printf("Sumando de menor a mayor: %.12f\n", s1);
```

Perdida de precisión

- Aumento del rango dinámico:
  - La ecuación $x^2 – y^2$ se puede implementar como (x+y)(x-y)
    - Con menor coste computacional
    - Y mayor precisión

```
float s1, s2, s3, s4;
double d1, d2, d3, d4;
s1 = 1.00000023841858;          s2 = 1.00000011920929;
s3 = s1*s1;                s4 = s2*s2;
printf("  (x2 - y2) = %.15f\n", s3-s4);
s3 = s1 + s2; s4 = s1 - s2;
printf("  (x + y) (x - y) = %.15f\n", s3*s4);
d1 = 1.00000023841858;          d2 = 1.00000011920929;
d3 = d1*d1;                d4 = d2*d2;
printf("  (x2 - y2) = %.15lf\n", d3-d4);
d3 = d1 + d2; d4 = d1 - d2;
printf("  (x + y) (x - y) = %.15lf\n", d3*d4);
```

Perdida de precisión

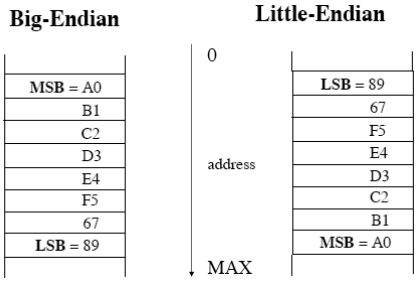
- Constantes inexactas
  - La división es una operación mucho más costosa que la multiplicación.
  - Así pues: x * 0.5 es preferible a x/2
  - Sin embargo, x * 0.1 no es igual a x/10
  - El motivo es que 0.1 no tiene una representación exacta en binario
  - Si la precisión fuese muy importante no está de más tener esto en cuenta.

Ejemplos de problemas mal planteados

- Diferencia de números parecidos:
  - La resta entre números de una magnitud parecida puede hacer perder varias cifras significativas
  - La solución es tratar de realizar las operaciones de otra forma
- Orden de las operaciones de acumulación
  - Si se suman primero los términos más pequeños se pierden menos cifras significativas que si se empieza sumando los términos de mayor valor
- Comparaciones:
  - Nunca se deben comparar con el operador == directamente. Hay que ver si el valor absoluto de su diferencia dividido por el valor absoluto del número es menor que un determinado número pequeño (1e-12 por ejemplo).

Big endian - little endian

- Big endian - el byte de mayor peso se encuentra en la dirección menor. Forma “natural”.
- Little endian - el byte de menor peso se encuentra en la dirección menor.



Big endian - little endian

- Big endian
  - facilita determinar el signo de un número
  - facilita las comparaciones entre dos números
  - facilita la división de dos números
  - facilita la impresión de los números
- Little endian
  - Facilita la suma y la multiplicación de números de múltiple precisión

Big endian - little endian

- El problema surge cuando se imprimen números a fichero, en formato binario, y se leen en otra máquina diferente de aquella en la que se han generado.
- Podríamos estar operando con números diferentes a los originales y sin embargo el código ser capaz de ejecutarse sin problemas.
- Subrutina para reconocer si una máquina es big endian o little endian:

```
#include <stdio.h>
int main(void){
int i = 1;
char *p = (char *) &i;
if ( p[0] == 1 )
      printf("Little Endian¥n");
else
      printf("Big Endian¥n");   return 0;
}
```

Documentación y bibliografía

- “What Every Computer Scientist Should Know About Floating-Point Arithmetic”. David Goldberg. ACM Computing Surveys. Marzo 1991.
  http://docs-pdf.sun.com/800-7895/800-7895.pdf
- Numerical Recipes: http://www.nr.com/

Sesión de laboratorio

Lab 2

Conclusiones

- Es importante conocer como se almacenan los datos numéricos en el computador
- Los errores debido al número limitado de cifras que almacena el ordenador se producen en cada operación aritmética y se pueden ir propagando a lo largo de los cálculos
- Ciertas técnicas de optimización pueden influir en la aparición de errores de este tipo o en su propagación
- Es importante evaluar la influencia de estos errores en el resultado final

Utilización de librerías matemáticas

Grupo de Arquitectura de Computadores
Dpt. Electrónica y Sistemas
Universidade da Coruña

Utilización de librerías matemáticas

- Objetivos:
  - Describir qué es una librería y las diferencias entre librerías estáticas y librerías dinámicas
  - Especificar librerías en la compilación
  - Ejemplos de librerías matemáticas
  - Uso de **ar** para crear y manipular librerías estáticas
  - Crear y actualizar librerías dinámicas
  - Los ficheros make en la construcción de software

Definición y uso de librerías

- *Librería:* fichero(s) que contiene código objeto con subrutinas y datos que pueden ser usados por otros programas.
- Ventajas:
  - Ahorran trabajo al programador
  - Mayor fiabilidad
- Tipos de librerías
  - Librerías de archivo
  - Librerías compartidas

Definición y uso de librerías

- Librerías estáticas (o de archivo):
  - Sufijo .a
  - Se crean usando el comando ar
- Librerías dinámicas (o compartidas):
  - Sufijos .sl .so
  - Se crean usando **ld** a partir de ficheros objeto que continen PIC (Position-Independent Code)

Definición y uso de librerías

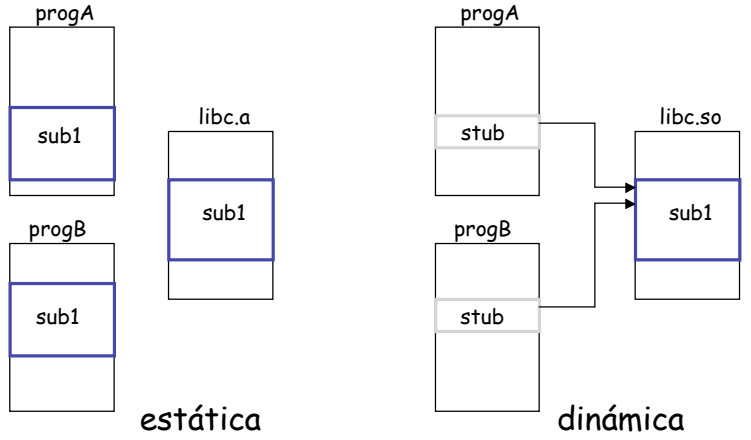
- Librerías estáticas:
  - Contienen uno o más ficheros objeto
  - Cuando un fichero objeto de una aplicación se enlaza con una librería estática, **ld** busca en la librería las definiciones globales que coinciden con las referencias externas del fichero objeto de la aplicación
  - Si se produce una coincidencia, **ld** copia el fichero objeto que contiene esa definición desde la librería al ejecutable a.out.
  - Conclusión: cada ejecutable contiene una copia de la subrutina empleada, tanto en disco como en memoria cuando se esta ejecutando.

Definición y uso de librerías

- Librerías dinámicas:
  - Contienen uno o más ficheros objeto
  - Cuando se enlaza un fichero objeto de la aplicación con la librería dinámica, **ld** no copia el código de la librería en el fichero final ejecutable
  - **ld** añade una lengueta (*stub*) por cada código objeto que se necesite, que simplemente notifica al ejecutable donde comienza en la librería el código que se necesita.
  - Un ejecutable que contiene una lengueta al código de la librería se denomina *ejecutable incompleto*
  - Conclusiones:
    - Los ficheros ejecutables generados ocupan menos espacio
    - La memoria necesaria para la ejecución también se reduce

Definición y uso de librerías

- Estáticas vs dinámicas:



Definición y uso de librerías

- Estátivas vs dinámicas:
  - Un programa compilado con librerías estáticas es más grande, ya que todo lo que se necesita se copia
  - Un programa compilado con librerías estáticas se puede llevar a otro ordenador sin necesidad de llevar las librerías
  - Un programa compilado con librerías estáticas es, en principio, más rápido en ejecución
  - Un programa compilado con librerías dinámicas ocupa, en general, menos memoria en ejecución
  - Las librerías dinámicas (o compartidas) permiten cambios en el código de las mismas sin tener que recompilar todos los ficheros de los programas que las usan.
  - Si cambiamos la librería estática para usar una nueva versión diferente de la antigua, a los ejecutables no les afecta

Definición y uso de librerías

- Uso de librerías en la compilación:
  - Si nuestra librería se llama lib***mia***.a hemos de enlazar usando la opción **-l**mia (lib y .a lo añade el compilador)
  - **-l** busca la librería en /usr/lib
  - Las librerías personales no suelen estar en esa localización y necesitan que se especifique
  - **-L** indica al *linker* donde se encuentran las librerías que se indican
  - Ejemplos:

    >cc long.c /home/lib/liba.sl /home/lib/libb.a ....

    o

    >cc short.c -L /home/lib -la -lb

    >cc prog.c -L . -lmia

Ejemplos de librerías matemáticas

- BLAS: *Basic Linear Algebra Subprograms*
- LAPACK: *Linear Algebra PACKage*
- LINPACK: Colección de subrutinas para la resolución de sistemas de ecuaciones lineales
- EISPACK: Colección de subrutinas para la computación de autovalores y autovectores de diferentes clases de matrices
- NAG: Colección desarrollada por el *Numerical Algebra Group*

Ejemplos de librerías matemáticas

- BLAS
  - Conjunto de rutinas para realizar operaciones básicas sobre vectores y matrices
  - Según su coste computacional, las rutinas de BLAS se dividen en 3 niveles:
    - Operaciones entre vectores: nivel 1
    - Operaciones entre matrices y vectores: nivel 2
    - Operaciones entre matrices: nivel 3
  - Existen versiones optimizadas para diversas arquitecturas
  - Sirven de base para la construcción de otros paquetes, como LAPACK.

Ejemplos de librerías matemáticas

- BLAS (2 y 3). Nomenclatura: XYYZZZ
  - X: indica el tipo de dato
    - S: real
    - D: double
    - C: complex
    - Z: double complex
  - YY: indica el tipo de matriz.
    - GE (general)
    - GB (banda)
    - HE (hermítica)
    - SY (simétrica)
    - TR (triangular
    - Etc.
  - ZZZ: indica la operación a realizar.
    - MV: producto matriz vector
    - R: actualización de rango 1
    - R2: actualización de rango 2
    - SV: resolución de sistema de ecuaciones

Ejemplos de librerías matemáticas

- BLAS nivel 1
  - Coste O(n)
  - Operaciones vector-vector
  - Ejemplos:
    - DSCAL:

$$x = \alpha x$$

call DSCAL (n, a, x, incx)

    - DDOT:

$$x = x \cdot y$$

call DDOT (n, x, incx, y, incy)

Ejemplos de librerías matemáticas

- BLAS nivel 2
  - Coste O(n2)
  - Operaciones matriz-vector
  - Ejemplo:
    - DGEMV:

$$y = \alpha Ax + \beta y$$

call DGEMV (trans, m, n, alpha, a, lda, x, incx, beta, y, incy)

Ejemplos de librerías matemáticas

- BLAS nivel 3
  - Coste O(n3)
  - Operaciones matriz-matriz
  - Ejemplo:
    - DGEMM:

$$C = \alpha AB + \beta y$$

call DGEMM (transa, transb, m, n, k, alpha, a, lda, b, ldb, beta, c, ldc)

Ejemplos de librerías matemáticas

- BLAS está originalmente en F77
- CBLAS es un interfaz C a las funciones BLAS
- La biblioteca BLAS en C es una conversión automática de la original fortran, por lo que necesita las bibliotecas de f2c: libl77.a y libf77.a
- CBLAS llama a BLAS, BLAS a libl77 y libl77 a libf77

Ejemplos de librerías matemáticas

- LAPACK
  - Librería de subrutinas F77 para la resolución de los problemas más frecuentes en álgebra lineal
  - Diseñado para ser eficiente: todas las operaciones están orientadas a bloques.
  - Las rutinas LAPACK están escritas de forma que la computación se haga mediante llamadas a BLAS
  - Permite conseguir un alto rendimiento y un código portable

Ejemplos de librerías matemáticas

- Problemas que LAPACK resuelve
  - Sistemas de ecuaciones lineales
  - Problemas de mínimos cuadrados
  - Problemas de valores propios
  - Problemas de valores singulares
  - Otros: factorización de matrices, estimación del número de condición, etc.

Ejemplos de librerías matemáticas

- LAPACK. Nomenclatura XYYZZZ
  - X: indica el tipo de dato
    - S: real
    - D: double
    - C: complex
    - Z: double complex
  - YY: indica el tipo de matriz.
    - GE (general)
    - GB (banda)
    - HE (hermítica)
    - SY (simétrica)
    - TR (triangular
    - Etc.
  - ZZZ: indica la operación a realizar.
    - SV: resolución de ecuaciones lineales
    - EV: problemas de valores propios
    - LS: problema de mínimos cuadrados
    - Etc.

Ejemplos de librerías matemáticas

- Ejemplo de subrutina LAPACK:
  - DGESV: resolución del sistema lineal de ecuaciones

$$Ax = b$$

call DGESV (n, nrhs, a, lda, ipvt, bx, ldb, info)

Documentación

- Documentación:
  - En http://www.netlib.org/lapack se encuentra la última edición de *LAPACK User´s Guide*
  - *En* http://www.netlib.org/blas/ se encuentra una lista FAQ sobre BLAS, que contiene una guía rápida de uso
  - La página de NAG: http://www.nag.co.uk/
  - Existen librerías matemáticas para máquinas paralelas. Las más conocidas:
    - ScalaPACK: versión paralela de LAPACK y más. http://www.netlib.org/scalapack/
    - *SuperLU* es una librería para resolución de sistemas de ecuaciones en máquinas paralelas. http://crd.lbl.gov/~xiaoye/SuperLU/

La librería MLIB en el HP Superdome

- HP MLIB 8.4. es un producto que funciona con sistemas operativos HP-UX 11i o superior y está optimizado para plataformas con procesadores Itanium 2, como el Superdome.
- MLIB incluye los siguientes paquetes de librerías:
  - BLAS niveles 1,2 y 3
  - sparse BLAS
  - LAPACK
  - ScaLAPACK
- además de diferentes transformadas, como FFTs, y convoluciones.
- HP MLIB 8.4 incluye además:
  - METIS
  - SuperLU_DIST
  - Paralelismo SMP

La librería MLIB en el HP Superdome

- HP MLIB 8.4 esta disponible en forma de librerías estáticas y dinámicas, que se relacionan en la siguiente tabla

32-bit	/opt/mlib/lib/hpux32	libveclib.a libveclib.so liblapack.a liblapack.so libscalapack.a libscalapack.so libsuperlu_dist.a libsuperlu_dist.so
64-bit	/opt/mlib/lib/hpux64	libveclib.a libveclib.so libveclib8.a libveclib8.so liblapack.a liblapack.so liblapack8.a liblapack8.so libscalapack.a libscalapack.so libscalapack8.a libscalapack8.so libsuperlu_dist.a libsuperlu_dist.so libsuperlu_dist8.a libsuperlu_dist8.so

La librería MLIB en el HP Superdome

- Compilar y enlazar
  - Se pueden elegir indistintamente las librerías estáticas o dinámicas
  - El rendimiento, en general, será superior cuando se usen librerías estáticas
  - Sin embargo, si los ejecutables tienen que ser pequeños, se pueden usar las librerías dinámicas
  - Usar: -aarchive_shared en la línea de comandos del compilador para asegurar que el compilador intenta enlazar con las librerías estáticas en primer lugar. Si no encuentra las librerías estáticas enlazará con las dinámicas
  - Usar: -ashared_archive para enlazar en primer lugar con las librerías dinámicas. Si ninguno de estas opciones se especifica el compilador, por defecto, tratará de enlazar con las librerías dinámicas.

La librería MLIB en el HP Superdome

- Ejemplos:

```
>f90 -o prog prog.f   -Wl -aarchive_shared -lveclib

>cc -o prog prog.c   -Wl -aarchive_shared -llapack -lcl -lm

>cc -o prog prog.c  /opt/mlib/lib/hpux32/libveclib.a -lcl -lm

>f90 -o prog prog.f -Wl -aarchive_shared -L /opt/mlib/lib/hpux64   \
    -llapack -lveclib

>mpicc  -o  mpiprog  mpiprog.c  -Wl -aarchive_shared -lscalapack  -lcl  -lm
```

La librería MLIB en el HP Superdome

- Uso de la variable de entorno LDOPTS:

```
>setenv LDOPTS -aarchive_shared -L/opt/mlib/lib/hpux32

>f90  -o prog prog.f  -lveclib -llapack

>cc -o prog prog.c  -lveclib -lcl -lm
```

- Uso de las versiones para 64 bits VECLIB8, LAPACK8, ScaLAPACK8, SuperLU_DIST8:

```
>f90 +DD64 +i8 -o prog prog.f -lveclib8

>cc +DD64 -o prog prog.c  -llapack8 -lcl -lm
```

La librería MLIB en el HP Superdome

- Documentación:
  - Paginas de manuales: /install_dir/share/man
  - En http://www.hp.com/go/mlib se pueden encontrar guías de usuario, por ejemplo: *HP MLIB User´s Guide VECLIB, LAPACK, ScaLAPACK and SuperLU_DIST*
  - En http://www.netlib.org/lapack/lug se encuentra la última edición de *LAPACK User´s Guide*
  - En http://www.netlib.org/lapack/slug se encuentra la última edición de *ScaLAPACK User´s Guide*
  - *SuperLU_DIST User´s Guide* se puede encontrar en http://www.nersc.gov/ ~xiaoye/SuperLU

Creación de librerías propias

- El comando ar
  - **ar** *options [posname] archivefile [objectfile ...]*
  - *-r* reemplaza o añade un fichero objeto en la librería
  - *-u* actualiza la librería
  - *-d* borra el fichero objeto de la librería
  - *-t* imprime una tabla con los contenidos de la librería
  - *-x* extrae un fichero objeto de la librería
  - *-v* muestra las acciones que tienen lugar en el proceso

  Nota: las opciones de ar no tienen que ir precedidas del signo -

Creación de librerías propias

Crear y usar librerías estáticas:

- Compilar el codigo de la librería

  >cc –c func1.c func2.c
- Crear la librería

  >ar –rv mylib.a func1.o func2.o

  a - func1.o

  a - func2.o

  ar: creating mylib.a
- Linkar con la librería

  >cc prog.c mylib.a –o prog

Creación de librerías propias

- Desplegar una tabla de contenidos de una librería estática

```
>ar –t mylib.a
func1.o
func2.o

>ar –tv mylib.a
rw-r--r--  201/   20    260 Aug 27 15:30 2003  func1.o
rw-r--r--  201/   20    260 Aug 27 15:30 2003  func1.o
```

Creación de librerías propias

- Borrar un fichero de una librería estática:

```
>ar –d mylib.a  func1.o

>ar –dv mylib.a  func1.o
  d – func1.o
```

- Extraer un fichero de una librería estática:

```
>ar  -x mylib.a  func2.o

>ar  -xv mylib.a  func2.o
  x -  func2.o
```

Creación de librerías propias

Crear y usar librerías dinámicas:

- Compilar con +z

  >cc –c +z func1.o func2.o

  func1:

  func2:

- Combinar con ld –b

  >ld –b –o mylib.sl func1.o func2.o

- Linkar con la librería

  >cc prog.c mylib.sl –o prog

Creación de librerías propias

- Modificar una librería dinámica:
  - Recompilar el código modificado

    >cc –c +z func2.c

  - Reconstruir la librería:

    >ld –b –o mylib.sl func1.o func2.o

Sesión de laboratorio

Lab 3.2

Ficheros make

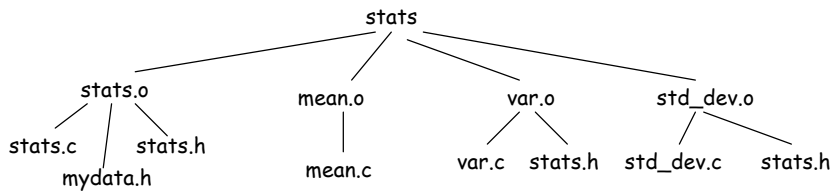
- Servicios que proporciona make
  - Automatiza el proceso de compilar y linkar ficheros
  - Documenta las relaciones entre los ficheros y los paquetes de software
  - Identifica los ficheros modificados y vuelve a procesar sólo los ficheros que lo necesitan
  - Automatiza el testeo, instalación y generación de documentación
  - Busca un fichero makefile que defina reglas de actuación, usando las reglas por defecto si no existe este fichero

Ficheros make

- Makefile
  - Lista de dependencias y relaciones entre ficheros
  - Contiene los comandos adecuados de compilación y linkado
- Make
  - Lee el fichero makefile
  - Examina las dependencias
  - Ejecuta los comandos de compilación y linkado

Ficheros make

- Ejemplo simple:
  - Proyecto: stats
  - Ficheros fuente: stats.c, mean.c var.c std_dev.c
  - Ficheros cabecera: stats.h mydata.h
  - Arbol de dependencias:



Ficheros make

- Ejemplo simple:
  - El comando:
    >make stats
  - La descripción del fichero makefile:

```
stats:      stats.o mean.o var.o std_dev.o
            cc -Aa stats.o mean.o var.o std_dev.o -lm -o stats
stats.o:    stats.c mydata.h stats.h
            cc -Aa -c stats.c
mean.o:     mean.c
            cc -Aa -c mean.c
var.o:      var.c stats.h
            cc -Aa -c var.c
std_dev.o:  std_dev.c stats.h
            cc -Aa -c std_dev.c
```

Ficheros make

- Líneas de dependencias
  - Más de una etiqueta:

```
app new_app: main.o button.o menu.o
        cc main.o button.o menu.o -lm -o app
        cc main.o button.o menu.o -L /usr/lib/X11R4 -lX11 -o new_app
```

  - Ocurrencia múltiple de componentes:

```
stats: stats.o mean.o var.o std_dev.o
        cc stats.o mean.o var.o std_dev.o -lm -o stats
mini_stats: mini_stats.o mean.o
        cc mini_stats.o mean.o -lm -o mini_stats
```

  - Sin componentes:

```
update:
   cc prog.c -o prog
```

Ficheros make

- Líneas de comandos:

```
update:
    rm *.o

    cd ./bin; cc prog.c -o prog
    if grep prog.c print.list; \
    then \
      lp prog.c; \
    fi

    echo programa actualizado....
```

Ficheros make

- Generación de ficheros:

```
prog: *.o
    cc *.o -o prog
    lp *.c

file1.o: file1.c
...
file2.o: file2.c
```

Ficheros make

- Uso de macros:

```
OBJECTS = file1.o file2.o file3.o file4.o
LIBS = proj.a
CFLAGS = -Aa -O
...
prog: $(OBJECTS)
    cc  $(CFLAGS)  $(OBJECTS)  $(LIBS) -o prog
file1.o: file1.c
    cc  $(CFLAGS)  file1.c
```

Ficheros make

- Uso de variables de entorno:

```
>env
  HOME = /users/patricia
  CFLAGS = -Aa -O

>cat makefile
  prog: prog.o
      cd $(HOME)/bin; cc $(CFLAGS) prog.o -o prog

      for i in *.c; \
      do \
          lp $$i; \
      done
```

Ficheros make

- Prioridad de asignaciones:
  - Definiciones en línea de comandos:
    >make stats CFLAGS="-Aa -O"
  - Definiciones dentro del fichero makefile
  - Variables de entorno
  - Definiciones internas del comando make

Ficheros make

- Macros internas:
  - $? Lista de componentes que han sido modificados más recientemente que la etiqueta en curso
  - $@ el nombre de la etiqueta en curso (para ser usado en línea de comandos)
  - $$@ el nombre de la etiqueta en curso (para ser usado en línea de dependencias)

Ejemplo, posible makefile en el directorio /usr/bin:

```
CMDS = cat chown date ls tar who
$(CMDS): $$@.c
        cc -Aa -O $? -o $@
```

Ficheros make

- Reglas con sufijos:
  - Un regla con sufijo indica cómo construir un fichero de un sufijo determinado partiendo de otro fichero (componente) con un sufijo distinto.
  - $< fichero componente necesario para contruir el fichero resultado
  - $* nombre de la etiqueta actual sin el sufijo
  - Regla para construir ficheros .o desde ficheros .c:

```
.SUFFIXES: .o .c
.c.o:
        $(CC)  $(CFLAGS)  -c  $<
```

Ficheros make

- Construcción de nuevas reglas con sufijos:
  - Añadir los sufijos .doc y .t a la lista actual de sufijos:

```
.SUFFIXES:    .doc   .t
```

  - Borrar todas los sufijos actuales:

```
.SUFFIXES:
```

  - Contruir una regla para crear .doc desde .c:

```
.SUFFIXES:    .doc
.c.doc:
        cat  header $<    >    $*.doc
```

  - Sufijo nulo:

```
.c:
    $(CC) $(CFLAGS) $< $(LDFLAGS) -o  $@
```

Ficheros make

- Pseudo-etiquetas:
  - **.SILENT**: ejecuta todos los comandos pero no los detalla en la pantalla
    - make -s etiqueta
    - @ al principio de la línea que no queremos desplegar por pantalla
  - **.IGNORE**: ignora los errores que devuelven los comandos
    - Make -i etiqueta
    - - al principio de la línea concreta
  - **.DEFAULT**: reglas usadas por defecto

Documentación y bibliografía

- The GNU Make manual

http://www.gnu.org/software/make/manual/html_node/index.html

- "Managing Projects with GNU Make". R. Mecklenburg. O'Reilly, 2004

Sesión de laboratorio

Lab 3.3

Bibliografía

- "Matrix Computations". G. Golub, C.F. Van Load. The Johns Hopkins University Press, 1996.
- Numerical Recipes
  http://www.nr.com/
- "Templates for the solution of linear systems". R. Barret et al. SIAM.
  http://www.netlib.org/linalg/html_templates/Templates.html

Conclusiones

- Librerías:
  - El uso de librerías resulta muy ventajoso, tanto si estas son librerías existentes en la plataforma como si son librerías propias
  - Ambos tipos de librerías:
    - Ahorran trabajo al programador
    - Proporcionan mayor fiabilidad pues ya están depuradas y comprobadas
  - Las librerías propias del sistema además:
    - Suelen estar optimizadas para el procesador usado, obteniendo mayor rendimiento
- Utilidad make:
  - Automatiza el proceso de construcción del ejecutable
  - Simplifica el proceso de creación y mantenimiento de la aplicación

Optimización del rendimiento de la jerarquía de memoria

Grupo de Arquitectura de Computadores
Dpt. Electrónica y Sistemas
Universidade da Coruña

Optimización del rendimiento de la jerarquía de memoria

- Objetivos:
  - Mostrar recomendaciones generales y técnicas directas de optimización
  - Estudiar la optimización de los códigos desde el punto de vista de la jerarquía de memoria
    - Describir la jerarquía de memoria
    - Describir el funcionamiento de la memoria cache
    - Mostrar reglas generales para optimizar el uso de la memoria cache
    - Definir memoria virtual
    - Mostrar reglas para optimizar teniendo en cuenta la memoria virtual

Recomendaciones generales

- Averiguar qué parte del programa consume la mayoría del tiempo
  - Si la rutina A consume el 20% del tiempo, y otra B el 80%, centrarse en B
- Invertir el tiempo en las zonas del programa computacionalmente más costosas (puntos calientes)
  - Se detectan por medio de:
    - Uso de herramientas de profiling del sistema: prof, gprof, tconv, ...
    - Rutinas de medida de tiempo: time, etime, ...
  - Cuando se quiere disminuir el tiempo de ejecución se tiene que conocer perfectamente cuál es la causa de la ineficiencia

Recomendaciones generales

- Compilar el programa con la máxima optimización
- Seleccionar el algoritmo óptimo
  - Generalmente hay muchos métodos para resolver un mismo problema
- Siempre que sea posible, usar los paquetes de rutinas matemáticas existentes
- Utilizar técnicas de programación estructurada

Técnicas directas de optimización

- Reordenar las expresiones para minimizar las operaciones
  - Ejemplo: Algoritmo de Horner
- No todas las operaciones aritméticas consumen el mismo tiempo
  - Generalmente, las divisiones son más costosas que las multiplicaciones

```
DO I=1, N
   A(I)=B(I)*C(I)/D
ENDDO
```

```
E=1/D
DO I=1, N
   A(I)=B(I)*C(I)*E
ENDDO
```

Técnicas directas de optimización

- Mejor usar constantes que variables
  - El compilador realiza una mejor optimización
- La exponenciación debe evitarse si es posible
- Evitar las conversiones de tipo
- Si el programa consume mucho tiempo en una expresión
  - Tal vez se pueda sustituir por una tabla

Técnicas directas de optimización

- Dentro de los bucles, eliminar
  - Saltos
  - Llamadas a subrutinas
  - Operaciones de entrada/salida

Técnicas directas de optimización

- Hacer el menor número posible de llamadas a rutinas
  - Problemas
    - Producen fallos en la cache de instrucciones
    - Las llamadas a rutinas tienen una alta sobrecarga (overhead)
    - Disminuye la probabilidad de que el compilador optimice
  - Soluciones
    - Copiar el código
    - Introducir el bucle dentro de la función
    - Sacar la función fuera del bucle

```
DO I=1, N
   A(I)=B(I)+log(3)
ENDDO
```

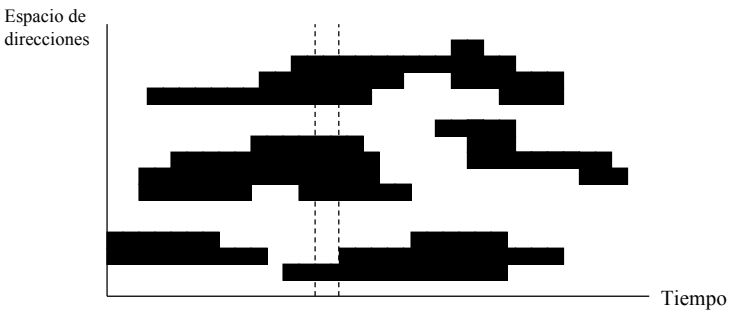
```
E=log(3)
DO I=1, N
   A(I)=B(I)+E
ENDDO
```

Jerarquía de memoria

- Propiedad de localidad de los programas
  - Referencias concentradas en regiones de tiempo y espacio
  - Resultados experimentales muestran:
    - El 90% del tiempo se suele consumir en el 10% del código (bucles)
- Tipos de localidad:
  - Localidad espacial (secuencial): direcciones próximas
    - Acceso a instrucciones en programas y a datos en arrays
  - Localidad temporal: misma dirección
    - Bucles y subrutinas en los programas
    - Variables temporales

Jerarquía de memoria

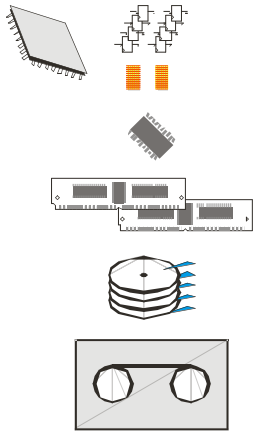
- Ejemplo de la localidad de un programa



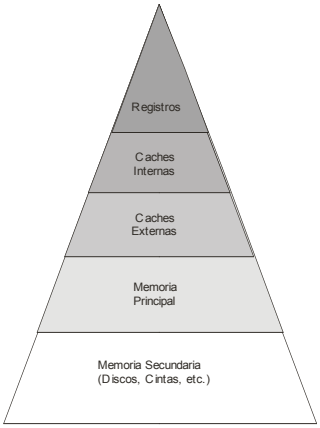
Jerarquía de memoria

- Niveles de una jerarquía de memoria
  - Registros
  - Memoria cache
  - Memoria principal
  - Memoria secundaria
- A medida que se desciende en la jerarquía
  - Aumenta el tiempo de acceso, la capacidad
  - Disminuye el coste

Jerarquía de memoria



- Registros del procesador
  - Hasta 2KB
  - Hasta 57GB/seg
- Caches internas
  - 32KB - 1MB
  - 45 – 25GB/seg
- Caches externas
  - Hasta 2MB
  - Hasta 25GB/seg
- Memoria Principal
  - > 4GB
  - > 5GB/seg
- Disco duro
  - Cientos de GB
  - Hasta 150 MB/seg
- Cintas de back-up
  - 200 GB por cinta
  - 30 MB/seg



Jerarquía de memoria

Precios por GB en $

Tipo	Precio	Referencia
SRAM	8800	8MB Cisco
DRAM	90	1 GB DDR 400
Disco duro	0,46	250 GB
DVD	0,10	100 DVD -R SL
Cinta	0,24	200 GB LTO

Referencia: Precio más bajo mostrado en www.cnet.com Abril 2005

Memoria cache

- La diferencia entre las velocidades del procesador y la memoria crece constantemente
- Crecimiento del rendimiento del procesador y de la memoria con respecto a un computador de 1980:
  - Memoria
    - 7% anual
    - Cada 3 años, aumenta por 4 la capacidad
  - Procesador
    - 35% anual (CISC)
    - 55% anual (RISC)

Memoria cache

- El diseño y el uso de la memoria cache es crítico en el computador
  - Llena el vacío entre procesador y memoria principal
- Memoria pequeña y rápida situada entre el procesador y la memoria principal
  - Almacena la información de la memoria principal actualmente en uso
- Objetivo: disminuir el tiempo de acceso a memoria
- Primer computador: IBM 360/85 (1969)

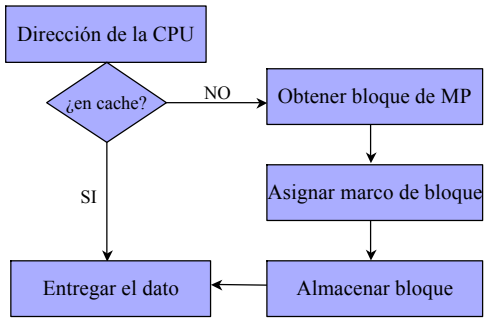
Memoria cache

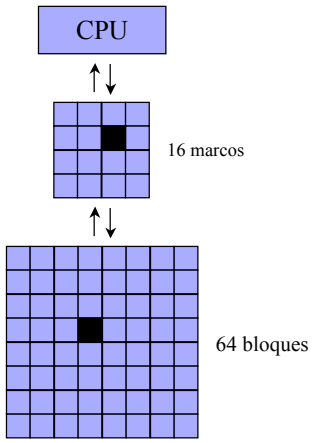
- Conceptos sobre la la cache
  - Bloque: unidad de transferencia
  - Marco de bloque: porción en la que se ubica el bloque
  - Acierto: el dato pedido se encuentra en la cache
    - Tiempo de acierto: t_{cache}
  - Fallo: el dato pedido no se encuentra en la cache
    - Tasa de fallos: T_{fallo}
  - Penalización: coste de un fallo (P)

$$T = t_{cache} + T_{fallo} * P$$

Memoria cache

- Funcionamiento de la cache





Optimización de la memoria cache

- Optimizar la localidad espacial
  - Cuanto más pequeño es el stride de acceso a un array, mejor
    - Stride óptimo = 1
- Optimizar la localidad temporal
  - Utilizar los datos todo lo posible una vez traídos de la memoria principal

Optimización de la memoria cache

- Técnicas por parte del compilador o del programador
- Objetivo: mejorar la localidad espacial y temporal de los programas
  - “Usar los datos que se encuentran en la cache antes de descartarlos”
- A continuación veremos ejemplos.
  - DEC Alpha 21064: Mc=8 Kb, E=1, M=256, Bloque=4x64 bits
  - Cada 1024 palabras se repite la asignación de marcos de bloques

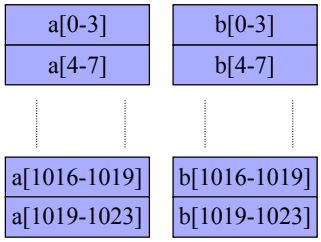
Optimización de la memoria cache

- Fusión de arrays
  - Mejora la localidad espacial para disminuir los fallos de conflicto
  - Coloca las mismas posiciones de diferentes arrays en posiciones contiguas de memoria

Optimización de la memoria cache

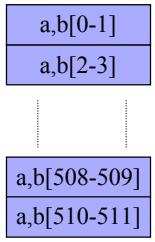
- Fusión de arrays: ejemplo

```
double a[1024], b[1024], c;
for(i=0;i<1024;i++)
   c+=a[i]+b[i];
```



2x1024 fallos =2x256 de inicio + 1536 de conflicto

```
struct { double a, b; } array;
double c;
for(i=0;i<1024;i++)
   c+=array.a[i]+array.b[i];
```



1024/2 fallos =2x256 de inicio

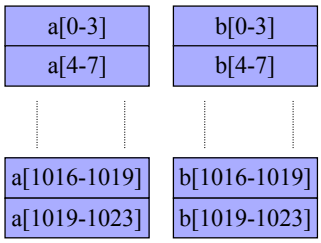
Optimización de la memoria cache

- Alargamiento de arrays
  - Mejora la localidad espacial para disminuir los fallos de conflicto
  - Impide que en cada iteración del bucle se compita por el mismo marco de bloque

Optimización de la memoria cache

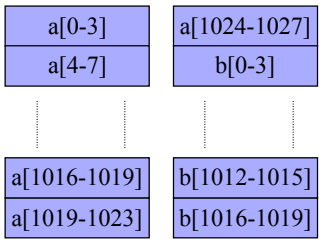
- Alargamiento de arrays: ejemplo

```
double a[1024], b[1024], c;
for(i=0;i<1024;i++)
   c+=a[i]+b[i];
```



2x1024 fallos =2x256 de inicio + 1536 de conflicto

```
double a[1028], b[1024], c;
for(i=0;i<1024;i++)
   c+=a[i]+b[i];
```



1024/2 fallos =2x256 de inicio

Optimización de la memoria cache

- Intercambio de bucles
  - Mejora la localidad espacial para disminuir los fallos de conflicto
  - En C las matrices se almacenan por filas, por ello en el bucle interno debe variar la fila

Optimización de la memoria cache

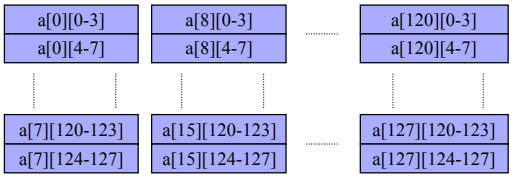
- Intercambio de bucles: ejemplo

```
double a[128][128];
for(j=0;j<128;j++)
   for(i=0;i<128;i++)
      c*=a[i][j];
```

128x128 fallos =16x256 de inicio + 12218 de conflicto

```
double a[128][128];
for(i=0;i<128;i++)
   for(j=0;j<128;j++)
      c*=a[i][j];
```

128x128/4 fallos =16x256 de inicio



Optimización de la memoria cache

- Fusión de bucles
  - Mejora la localidad espacial para disminuir los fallos de capacidad
  - Fusionar los bucles que utilizan los mismos arrays para reutilizar los datos de la cache
  - En C las matrices se almacenan por filas, por ello en el bucle interno debe variar la fila

Optimización de la memoria cache

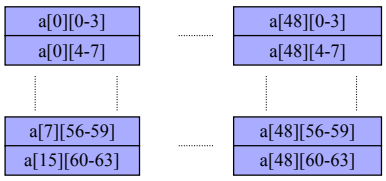
- Fusión de bucles: ejemplo

```
double a[64][64];
for(i=0;j<64;i++)
   for(j=0;j<64;j++)
      c*=a[i][j];
for(i=0;i<64;i++)
   for(j=0;j<64;j++)
      d+=a[i][j];
```

(64x64/4)x2 fallos=4x256 de inicio + 4x256 de capacidad

```
double a[64][64];
for(i=0;i<64;i++)
   for(j=0;j<64;j++) {
      c*=a[i][j];
      d+=a[i][j];
   {
```

64x64/4 fallos =4x256 de inicio



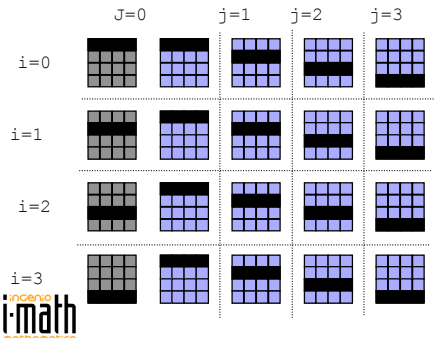
Optimización de la memoria cache

- Bloqueo de matrices
  - Mejora la localidad temporal para disminuir los fallos de capacidad
  - Operar con submatrices para reutilizar los datos de la cache
  - Ejemplo
    - Mc de 128 bytes
    - Totalmente asociativa (reemplazo LRU)
    - 4 marcos
    - Bloque de 4 palabras de 64 bits

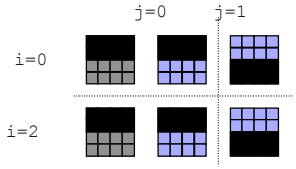
Optimización de la memoria cache

- Bloqueo de matrices: ejemplo

```
double a[4][4], b[4][4];
for(i=0;i<4;i++)
   for(j=0;j<4;j++)
      a[i][j] += b[j][i];
```




```
double a[4][4], b[4][4];
for(i=0;i<4;i+=2)
   for(j=0;j<4;j+=2){
      a[i][j] += b[j][i];
      a[i][j+1] += b[j+1][i];
      a[i+1][j] += b[j][i+1];
      a[i+1][j+1] += b[j+1][i+1];
   }
```

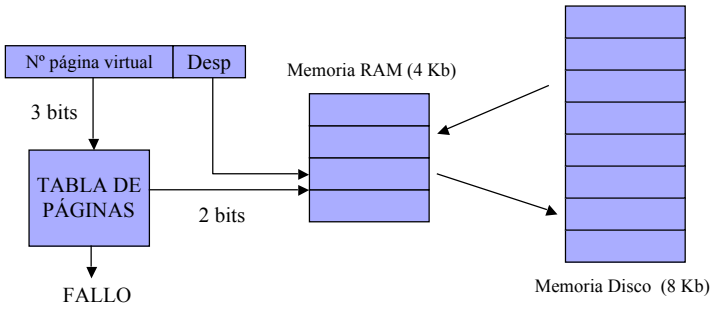


Ganancia=20/12=1.66

Memoria virtual

- La memoria virtual
  - Crea la ilusión de un espacio de memoria propio e ilimitado
  - Facilita la compartición de memoria por varios procesos
- Funcionamiento
  - Las direcciones que genera un procesador son virtuales
  - La traducción a direcciones físicas se gestiona dividiendo el espacio en páginas

Memoria virtual



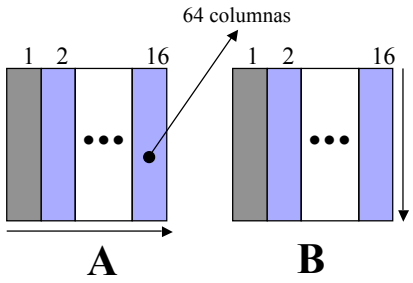
Optimización de la memoria virtual

- Se aplican las mismas técnicas que en la memoria cache
- La penalización por fallo es mucho mayor
- Ejemplo: calcular C=C+A*B, siendo A,B,C matrices de 1024x1024
- Características del sistema de memoria virtual:
  - 1 página = 2^{16} palabras = 64 columnas de A,B o C
  - 1 fallo de página = 0.5 segundos
  - Almacenamiento de A,B,C = 3x1024x1024 palabras
  - Capacidad de la memoria principal = 16 páginas = 1024x1024 palabras = 1 matriz

Optimización de la memoria virtual

- Variante **ijk**
  - Acceso a las columnas de A: 1024*1024 fallos
  - Acceso a las columnas de B, C: 1024*16 fallos
  - Tiempo = 1024*1024*0.5 = 93 días.

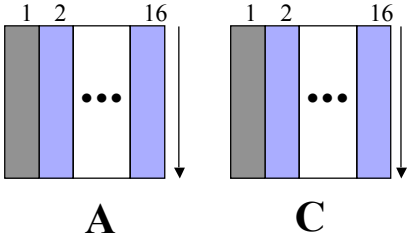
```
DO I=1,N
   DO J=1,N
      DO K=1,N
         C(I,J)=C(I,J)+A(I,K)*B(K,J)
      ENDDO
   ENDDO
ENDDO
```



Optimización de la memoria virtual

- Variante **jki**
  - Acceso a las columnas de A: 1024*16 fallos
  - Acceso a las columnas de B, C: 16 fallos
  - Tiempo = 1024*16*0.5 = 8192 seg.

```
DO J=1,N
   DO K=1,N
      DO I=1,N
         C(I,J)=C(I,J)+A(I,K)*B(K,J)
      ENDDO
   ENDDO
ENDDO
```



Optimización de la memoria virtual

- Variante **jki** bloqueado
  - Acceso a las columnas de A: 16*16 fallos
  - Acceso a las columnas de B, C: 16 fallos
  - Tiempo = 16*16*0.5 = 128 seg.

```
*
* NB=64
*
DO J=1,N,NB
   JB=MIN(N-J+1,NB)
   DO K=1,N,NB
   KB=MIN(N-K+1,NB)
*
* Multiplicación submatriz-submatriz
*
      DO JJ=J,J+JB-1
         DO KK=K,K+KB-1
            DO I=1,N
               C(I,JJ)=C(I,JJ)+A(I,KK)*B(KK,JJ)
            ENDDO
         ENDDO
      ENDDO
   ENDDO
ENDDO
```

Sesión de laboratorio

Lab 4

Documentación y bibliografía

- Programming Optimization: techniques, examples and discussion
  http://www.azillionmonkeys.com/qed/optimize.html
- "Code Optimization: Effective Memory Usage". K. Kaspersky. A-List Publishing, 2003

Conclusiones

- Muchas de las optimizaciones a nivel de jerarquía de memoria las realizan automáticamente los compiladores, pero no siempre…
- Pasos en la optimización de un código secuencial:
  - Compilar sin optimización
    - Medir tiempo
    - Guardar resultados para usar como referencia
  - Compilar con la optimización máxima que permita el compilador
    - Medir tiempo y comparar resultados
    - Compilar usando diferentes optimizaciones (de más a menos) hasta encontrar que los resultados son correctos
  - Reestructurar el código
    - Recomendaciones generales, codificación a nivel de bucle
    - Medir tiempos y comparar resultados
- Escribir código es difícil, pero optimizar código ya escrito es más difícil todavía.

Depuración y evaluación del rendimiento

Grupo de Arquitectura de Computadores
Dpt. Electrónica y Sistemas
Universidade da Coruña

Depuración y evaluación del rendimiento

- Objetivos:
  - Aprender a manejar el depurador GDB:
    - Arrancar y parar una sesión de depuración
    - Invocar y ejecutar un programa bajo el control del depurador
    - Usar puntos de ruptura y trazas
    - Ver y manipular los datos del programa
  - Evaluar el rendimiento de un programa
    - Usar las rutinas de medidas de tiempos
    - Usar la herramienta genérica **gprof** para obtener el perfil de la aplicación

El depurador GDB

- No dispone de un interfaz gráfico
- Ayuda online extensa
- Capacidades de depuración que incluye:
  - Vista del código fuente
  - Vista/modificación de los datos del programa
  - Vista de la pila durante la ejecución
  - Puntos de ruptura y pasos a través del código
  - Muestra de trazas en la ejecución del programa

El depurador GDB

- Guía rápida:
  - Compilar el programa con la opción -g

```
>>cc -g prueba.c -o prueba
```

  - Ejecutar el programa desde el debugger. Si la ejecución falla se visualizará donde ocurrió el fallo. Es posible cortar el programa en cualquier momento con CTRL+C y regresar al debugger, lo que permite verificar lazos infinitos, etc.

```
>>gdb prueba
(gdb) run [argumentos]
…
CTRL+C
(gdb)
```

El depurador GDB

- Guía rápida:
  - Una vez detectado funciones donde puede haber problemas.

```
(gdb) list funcion
…
(gdb) break linea
(gdb) run [argumentos]
…
break
(gdb) print expr
(gdb) next
…
(gdb) c
…
```

El depurador GDB

- Guía rápida:
  - Determinar donde termina un programa con core:

```
>>gdb programa core
#0 main () at prueba.c:100
100*(char *)0 = 10;
(gdb) bt
…
```

El depurador GDB

- Guía rápida:
  - Es posible depurar un programa en ejecución:

```
>>gdb programa pid
(gdb)
…
```

El depurador GDB

- Breakpoints:
  - Son puntos donde el programa se detiene al pasar por ellos
    - Para colocar un breakpoint al comienzo de una función o la línea indicada:

      (gdb) break [archivo:]funcion

      (gdb) break [archivo:]linea
    - El breakpoint es valido por una sola vez. Para crear breakpoints temporales:

      (gdb) tbreak [archivo:]funcion

      (gdb) tbreak [archivo:]linea

El depurador GDB

- Watchpoints:
  - Son expresiones que detienen el programa cuando un determinado valor cambia
    - Para habilitar un watchpoint cuando una expresión cambia:

      (gdb) watch expr
- Catchpoints:
  - Son breakpoints sobre señales (signals)
    - Para colocar breakpoints en todos los handlers de excepciones del contexto actual:

      (gdb) catch

El depurador GDB

- Para obtener información sobre los breakpoints o watchpoints:
  (gdb) info break
  (gdb) info watch
- Para eliminar un breakpoint de una línea o función:
  (gdb) clear linea
  (gdb) clear funcion
- Para eliminar un breakpoint por número:
  (gdb) delete numero

El depurador GDB

- Para habilitar o deshabilitar temporalmente un breakpoint
  (gdb) disable breakpoint
  (gdb) enable breakpoint
- Para hacer que un breakpoint sea condicional, es decir, que sólo se habilite si la expresión es cierta:
  (gdb) condition numero_breakpoint [expr]
- Para ignorar el breakpoint durante varias pasadas:
  (gdb) ignore numero_breakpoint [count]

El depurador GDB

- Otros comandos útiles:
  - Listar el fuente a partir de una función o una línea:
    (gdb) list [archivo:]funcion
    (gdb) list [archivo:]linea
    (gdb) list
    (gdb) list -
  - Mostrar la pila, indicando las funciones invocadas y en que lugares fueron llamadas (backtrace)
    (gdb) bt
  - Continuar la ejecución del programa después de que haya sido detenido:
    (gdb) c
  - Mostrar el valor de una expresión:
    (gdb) print expr

El depurador GDB

- Ejecutar la próxima línea del programa sin entrar dentro de las funciones
  (gdb) next
- Ejecutar la próxima línea del programa entrando en las funciones
  (gdb) step
- Saltar los comandos siguientes y comienzar a partir de “linea”. Útil para continuar un programa saltando las líneas defectuosas
  (gdb) jump linea
- Ayuda en línea:
  (gdb) help [item]
- Salir de GDB:
  (gdb) quit

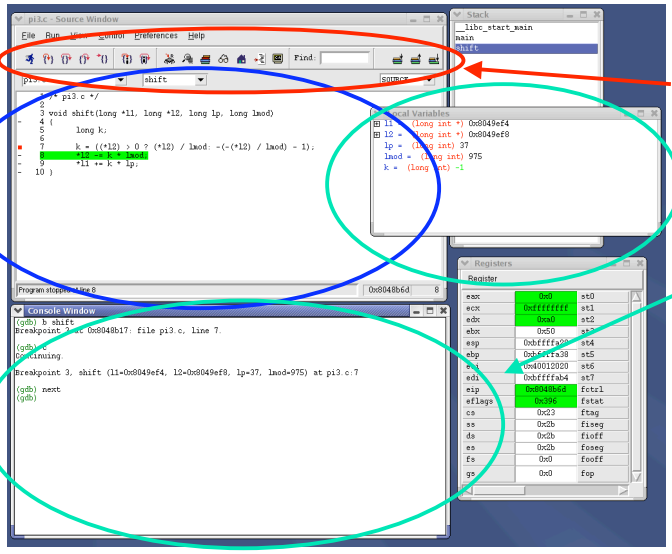
El depurador GDB

- Examinando y cambiando los datos:
  - Para ver de que tipo es una variable:
    (gdb) whatis variable
  - Para imprimir una expresión:
    (gdb) print [formato] expr
  - Para imprimir una expresión de forma continua durante el depurado (cada vez que el programa se detiene hace un print de la expresión):
    (gdb) display [formato] expr
    (gdb)
    (gdb) undisplay numero
  - Para modificar el valor de una variable:
    (gdb) variable=expr

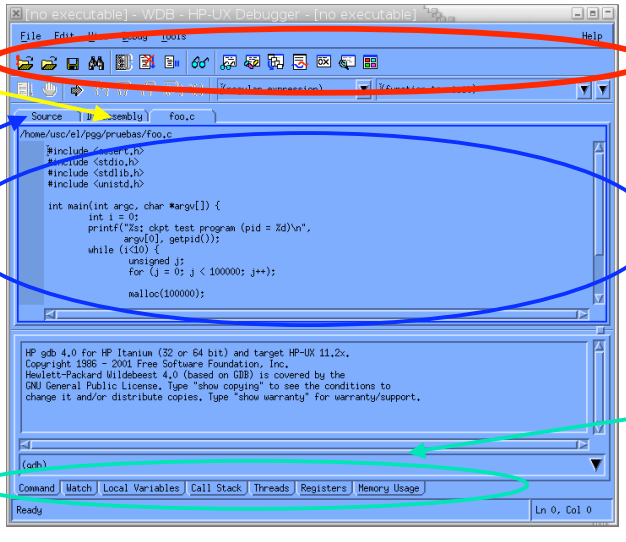
Interfaz gráfico para GDB

- Existen diversos interfaces gráficos para GDB. Los más populares son:
  - DDD: Data Display Debugger, que puede usar diferentes depuradores, incluyendo GDB
    http://www.gnu.org/software/ddd/
  - Insight: The GDB GUI
    http://sources.redhat.com/insight/
  - WDB: depurador para HP basado en GDB
    http://www.hp.com/go/wdb

Insight: the GDB GUI



El depurador HP WDB



Documentación

- GDB: The GNU Project Debugger

http://sources.redhat.com/gdb/documentation/

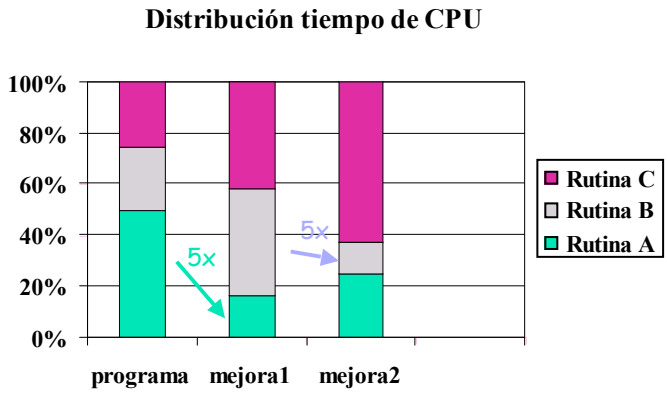
Sesión de laboratorio

Lab 5.1

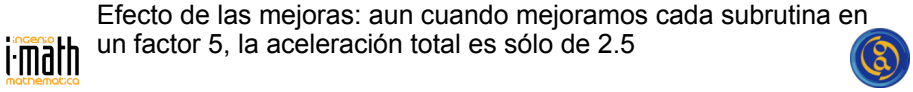
Evaluación del rendimiento

- Evaluando el rendimiento de nuestra aplicación durante el proceso de desarrollo de la misma podremos:
  - Localizar cuellos de botella
  - Decidir la forma de mejorar el rendimiento
  - Eliminar el coste de optimización de módulos poco usados

Evaluación del rendimiento



Efecto de las mejoras: aun cuando mejoramos cada subrutina en un factor 5, la aceleración total es sólo de 2.5



Evaluación del rendimiento

- Tiempo de ejecución: T = Ti + Td

  Ti = tiempo destinado a ejecutar instrucciones

  Td = tiempo de espera por los datos

  - Ti = $\sum$ # instrucciones * (# ciclos/instrucción)
  - Td = $\sum$ # memops * (# ciclos/memop)

  Estos 4 componentes se pueden ver influenciados por distintas técnicas de optimización

Evaluación del rendimiento

- Herramientas para la evaluación del rendimiento:
  - Medidas de tiempo
  - Gprof
  - Herramientas de HP:
    - Caliper
    - Prospect

Evaluación del rendimiento

- Medidas de tiempo:
  - Medida del sistema: comando time
    - Tiempo de CPU de usuario: 00.0u
    - Tiempo de CPU del sistema: 00.0s
    - Tiempo de respuesta del sistema: 0:00
    - Utilización de la CPU: 0%
  - Dentro del programa C: gethrtime()

```
#include <sys/time.h>
void myclock(){
   hrtime_t begin, end;
   int i, count = 1000;
   begin = gethrtime();
   for (i = 0; i < count; i++)
         subrutina();
   end = gethrtime();
   printf("Avg time = %lld nsec", (end - begin)/count );
```

```
#include <sys/time.h>
void myclock(){
    hrtime_t begin, end;
    begin = gethrtime();
    subrutina();
    end = gethrtime();
    printf("Time = %lld nsec", (end - begin));
}
```

Evaluación del rendimiento

- Dentro del programa Fortran: SECOND(), TIMEF()

```
PROGRAM  MAIN
REAL begin, end;
INTEGER i;
   begin = SECOND()
   do i = 0,100
        call subrutina()
   end do
   end = SECOND();
   print*,'Avg time = ', (end - begin)/100,' s'
END
```

```
PROGRAM  MAIN
REAL begin, end;
INTEGER i;
   begin = TIMEF()
   do i = 0,100
        call subrutina()
   end do
   end = TIMEF();
   print*,'Avg time = ', (end - begin)/100,' ms'
END
```

*Nota: si no existen por defecto, compilar con la librería lapack

Evaluación del rendimiento

- Herramientas prof y gprof:
  - Estándares en la mayoría de sistemas UNIX.
  - Estas herramientas muestrean el contador de programa de la CPU en intervalos para determinar en qué funciones y bloques básicos el programa "emplea" el tiempo
  - Permiten conocer las partes del programa que van más lentas de lo que esperamos, y que son candidatas a ser reescritas o intentar optimizarlas
  - También muestran el número de llamadas a las diferentes funciones del programa
  - Esto nos ayuda a identificar las funciones por las que el programa pasa menos o más veces de lo esperado y poder solucionar problemas que de otro modo no se detectarían

Evaluación del rendimiento

- gprof: pasos
  - Compilar y enlazar el programa con la opción que posibilita el perfilado
    >cc -p -o foo foo.c
    >cc -p -c foo.c ← después hay que linkarlo
  - Ejecutar el programa para recoger los datos del perfil
    >foo → gmon.out
  - Ejecutar gprof para analizar los datos recogidos en el paso anterior
    >gprof

Evaluación del rendimiento

- Diferentes análisis
  - Hay varias formas de salida disponibles para analizar los datos
  - El **perfil plano** (flat profile) muestra el tiempo que el programa pasa en cada función y el número de veces que se llama cada función
  - El **grafo de llamadas** (call graph) muestra, para cada función, qué otras funciones la llaman, a qué otras funciones llama ella y cuantas veces sucede esto. Esto puede sugerir lugares donde deberíamos eliminar funciones que se llaman muchas veces
  - La **lista de anotaciones** al fuente (annotated source listing) es una copia del código fuente donde cada línea se etiqueta indicando el número de veces que ha sido ejecutada

Evaluación del rendimiento

gprof options [executable-file [profile-data-files...]] [> outfile]

- Opciones de salida y de análisis:
  - -A (annotated-source) etiqueta el código fuente
  - -b (brief) no imprime la explicación del significado de cada campo de las tablas
  - -C (exec-counts) imprime las funciones y el número de veces que se han llamado
  - -i (info) imprime información resumen sobre el perfil
  - -p (flat-profile) imprime un perfil plano
  - -q (graph) imprime el análisis del grafo de llamadas
  - -w width indica la anchura de las líneas que imprime gprof
  - -s (sum) resumen la información en un fichero gmon.sum que se incrementa si volvemos a usar esta opción de nuevo. Esto nos permite tener un historico de lo que va ocurriendo.

Evaluación del rendimiento

- -a (no_static) suprime la impresión de las funciones estáticas (privadas). Son las funciones que no se declaran como globales, su tiempo se cuenta como perteneciente a la función en la que son llamadas
- -c (static-call-graph) introduce en el grafo de llamadas tambien las funciones que no han sido llamadas (aparecen con contador 0)
- -l (line) habilita el perfil línea a línea, el histograma obtiene un nuevo punto en cada línea de código
- -m num (min-count) suprime la impresión de aquellos símbolos que ocurren menos de num veces
- -z (display-unused-functions) hace que gprof mencione en el perfil plano las funciones que nunca han sido llamadas

Evaluación del rendimiento

- El perfil plano:

Flat profile:
Each sample counts as 0.01 seconds.

% time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
33.34	0.02	0.02	7208	0.00	0.00	open
16.67	0.03	0.01	244	0.04	0.12	offtime
16.67	0.04	0.01	8	1.25	1.25	memccpy
16.67	0.05	0.01	7	1.43	1.43	write
16.67	0.06	0.01				mcount
0.00	0.06	0.00	236	0.00	0.00	tzset
0.00	0.06	0.00	192	0.00	0.00	tolower
0.00	0.06	0.00	47	0.00	0.00	strlen
0.00	0.06	0.00	45	0.00	0.00	strchr
0.00	0.06	0.00	1	0.00	50.00	main
0.00	0.06	0.00	1	0.00	0.00	memcpy
0.00	0.06	0.00	1	0.00	10.11	print
0.00	0.06	0.00	1	0.00	0.00	profil
0.00	0.06	0.00	1	0.00	50.00	report

Evaluación del rendimiento

- Grafo:

```
granularity: each sample hit covers 2 byte(s) for 20.00% of 0.05 seconds
index % time    self  children    called     name
[1]    100.0    0.00    0.05                 start [1]
                0.00    0.05       1/1           main [2]
                0.00    0.00       1/2           on_exit [28]
                0.00    0.00       1/1           exit [59]
-----------------------------------------------
                0.00    0.05       1/1           start [1]
[2]    100.0    0.00    0.05       1           main [2]
                0.00    0.05       1/1           report [3]
-----------------------------------------------
                0.00    0.05       1/1           main [2]
[3]    100.0    0.00    0.05       1           report [3]
                0.00    0.03       8/8           timelocal [6]
                0.00    0.01       1/1           print [9]
                0.00    0.01       9/9           fgets [12]
                0.00    0.00      12/34            strncmp <cycle 1> [40]
                0.00    0.00       8/8           lookup [20]
                0.00    0.00       1/1           fopen [21]
                0.00    0.00       8/8           chewtime [24]
                0.00    0.00       8/16          skipspace [44]
-----------------------------------------------
```

Evaluación del rendimiento

- Etiquetado:

```
1 ulg updcrc(s, n)
2     uch *s;
3     unsigned n;
4 {
5     register ulg c;
6
7     static ulg crc = (ulg)0xffffffffL;
8
9     if (s == NULL) {
10        c = 0xffffffffL;
11    } else {
12        c = crc;
13        if (n) do {
14            c = crc_32_tab[...];
15        } while (--n);
16    }
17    crc = c;
18    return c ^ 0xffffffffL;
19 }
```

```
ulg updcrc(s, n)
                uch *s;
                unsigned n;
          2 ->{
                register ulg c;

                static ulg crc = (ulg)0xffffffffL;

          2 ->    if (s == NULL) {
          1 ->c = 0xffffffffL;
          1 ->    } else {
          1 ->c = crc;
          1 ->        if (n) do {
      26312 ->            c = crc_32_tab[...];
26312,1,26311 ->        } while (--n);
                }
          2 ->    crc = c;
          2 ->    return c ^ 0xffffffffL;
          2 ->}
```

Documentación

- GNU gprof
  http://sources.redhat.com/binutils/docs-2.12/gprof.info/index.html

Evaluación del rendimiento

- Caliper:
  - HP Caliper es una herramienta de análisis de proposito general para aplicaciones en el Itanium
  - HP Caliper permite entender el rendimiento que obtiene el programa y ayuda a identificar formas de mejorarlo
  - HP Caliper trabaja sobre cualquier binario y no requiere que la aplicación se prepare de forma especial para permitir la medida del rendimiento

Evaluación del rendimiento

- Caliper incluye soporte para:
  - Código nativo HP-UX generado con ANSI C, C++ y Fortran
  - Programas compilados con información sobre optimización, depuración o ambos
  - Programas ILP32 (+DD32) y ILP64 (+DD64)
  - Aplicaciones multihilo

Evaluación del rendimiento

- Caliper se usa como:
  - Herramienta para la optimización basada en el perfil (PBO - ver compiladores)
  - Herramienta de análisis del rendimiento

Evaluación del rendimiento

- Caliper como herramienta de optimización basada en el perfil
  - Caliper: pasos en la optimización PBO
    - Paso1: compilar el programa con las opciones normales excepto optimización. El ejecutable usado para generar los datos del perfil no debe estar compilado con un nivel de optimización mayor que +O1. Se puede explícitamente especificar +O1 para asegurar que el nivel es el correcto

>cc +O1 -o prog prog.c

Evaluación del rendimiento

- Caliper: pasos en la optimización PBO
  - Paso2: usar HP Caliper para generar datos de flujo de programa para que el compilador los use para optimizar el programa

>caliper pbo prog

    - Este comando ejecuta el programa y genera el fichero flow.data
    - Se puede repetir este paso tantas veces como se desee, simulando diferentes escenarios para un programa
    - HP agrega los resultados del perfil de datos para cada nueva ejecución en el fichero flow.data
    - Cuantos más escenarios se usen con los modos en que realmente se ejecutará el programa, más podrá optimizar el compilador

Evaluación del rendimiento

- Caliper: pasos en la optimización PBO
  - Paso3: recompilar el programa con la mayor optimización y el fichero con la información sobre el flujo de datos como entrada

    >cc +O3 -o prog2 +Oprofile=use:flow.data prog.c
  - Paso4: medir el rendimiento global de la versión optimizada

    >caliper total_cpu prog2 > resultado

Evaluación del rendimiento

- Caliper como herramienta para análisis del rendimiento
  - Fácil de usar (más fácil que gprof):

    >caliper fprof prog

    >caliper dcache_miss prog
  - Se pueden especificar opciones de medida e informe en los ficheros de configuración, en línea de comandos, que sobreescribe la configuración definida por los ficheros, o a través de la variable de entorno CALIPER_OPTS:
    - Predicción de saltos (branch_prediction)
    - Fallos en la cache de datos (dcache_miss)
    - Fallos en la TLB de datos (dtlb_miss)
    - Fallos en la cache de instrucciones (icache_miss)
    - Fallos en la TLB de instrucciones (itlb_miss)
    - Muestreo de las instrucciones de programa (sample_ip)

Evaluación del rendimiento

- HP Caliper realiza las medidas en todos los procesos en aplicaciones multihilo. Por defecto recoge todos los datos de forma agregada a través de todos los hilos. Pero se puede especificar que la toma de medidas sea separada para cada hilo usando -thread=all
- HP Caliper permite producir un informe del perfil del grafo de llamadas (call graph profile report - CGProf) de cualquier programa, sin necesidad de compilarlo de alguna forma especial para incluir la instrumentación.
- El informe incluye los siguientes resultados:
  - Tiempos de ejecución total para cada función del programa
  - Contadores de llamadas a cada función del programa
  - Información sobre los ciclos, los tiempos y los contadores de llamada

Evaluación del rendimiento

- Perfil del grafo de llamadas:
  - Se puede invocar al pefil del grafo de llamadas desde la línea de comandos, desde un script o desde un fichero Makefile del programa. La sintaxis es:

caliper cgprof [caliper_options] program [program_arguments]

  - -o fichero : especifica el fichero donde se escriben los resultados, Si se omite esta opción la salida va a la salida estandar

>caliper cgprof -o results.save prog "argumento"

Evaluación del rendimiento

- Caliper: tipos de medidas
  - Genera una versión HTML de los informes para los siguientes tipos de medidas:
    - branch_prediction
    - cgprof
    - dcache_miss
    - dtlb_miss
    - icache_miss
    - itlb_miss
    - sample_ip

Evaluación del rendimiento

- Caliper: ficheros de configuración
  - Los ficheros de configuración permiten especificar eventos que queremos medir
  - Los ficheros de configuración también proporcionan variables que permiten especificar tipos de control de medidas del rendimiento y contenido de los informes
  - Se pueden encontrar ejemplos en el directorio /opt/caliper/config
  - Se pueden usar los ficheros de configuración existentes o crear ficheros propios para medir eventos específicos

Evaluación del rendimiento

- Caliper: algunos ficheros de configuración existentes
  - **arc_count** medida de arcos entre bloques básicos
  - **branch_prediction** medida de predicción de saltos y fallos
  - **cgprof** medida del perfil del grafo de llamadas
  - **dcache_miss** medida de los fallos cache
  - **func_count** medida de los contadores de funciones
  - **sample_ip** medida del muestreo de las instrucciones
  - **total_cpu** medida de los contadoress de eventos de la CPU

Evaluación del rendimiento

- Caliper: generación de informes
  - Informe de texto:

    >caliper config_file [caliper_options] program [program arg]

    Reemplazando config_file por el nombre del fichero de configuración que deseamos usar, como branch_prediction. Para ver la lista de ficheros disponibles usar la opción --help
  - Informe HTML:

    >caliper config_file --html=output_dir [caliper options] program [program arg]

    El informe resultado contienen la misma información que el de texto pero tenemos un mayor control interactivo sobre la información representada. Para ver este informe abrir el fichero ***output_dir/index.html***

Evaluación del rendimiento

- Caliper:
  - http://www.hp.com/go/hpcaliper

Sesión de laboratorio

Lab 5.2

Conclusiones

- Depurador:
  - Disponer de un buen depurador y usarlo nos puede ahorrar mucho tiempo en la búsqueda de errores cuando desarrollamos nuestra aplicación
  - Existen errores difícilmente detectables si no se usa un depurador
- Análisis de rendimiento:
  - Las herramientas que generan diferentes perfiles de nuestra aplicación nos permiten desarrollar código más eficiente de una manera fácil y rápida

Introducción a la computación paralela

Grupo de Arquitectura de Computadores
Dpt. Electrónica y Sistemas
Universidade da Coruña

Introducción a la computación paralela

- Objetivos:
  - Describir en que consiste la computación paralela, y dar una visión general de la misma
  - Mostrar una clasificación de las arquitecturas paralelas
  - Conocer los diferentes paradigmas de computación paralela
  - Describir el proceso de diseño de una aplicación paralela
  - Mostrar ejemplos sencillos de computación paralela

Introducción

- Computación de altas prestaciones: resolver problemas muy costosos computacionalmente, minimizando el tiempo de respuesta.
- Para hacer algo más rápidamente se puede:
  - Trabajar más duro: mejorar la tecnología
  - Trabajar de forma más inteligente: optimizar los algoritmos
  - Buscar ayuda: procesamiento paralelo

Computación paralela

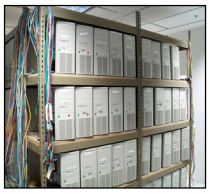
- ¿Qué es?
  - La computación paralela es un método por el que se dividen grandes problemas en componentes más pequeños, llamados tareas o procesos, que pueden resolverse en paralelo.
  - Esta solución surge como una respuesta de la arquitectura de computadores a las crecientes demandas de potencia de cómputo de los usuarios

Computación paralela

- ¿Para qué?
  - Para resolver problemas grandes
    - Muchas aplicaciones necesitan más memoria de la que un PC les puede proporcionar
  - Para resolver problemas más rápidamente
    - A pesar de los avances en la tecnología de computadores, muchas aplicaciones se ejecutan con demasiada lentitud
      - Ej: bases de datos que trabajan con un número creciente de datos
      - Ej: grandes simulaciones que trabajan para conseguir soluciones cada vez más precisas

Computación paralela

- Hasta hace relativamente poco la computación de altas prestaciones …
  - Supercomputadores
  - Códigos que se ejecutaban en un espacio de tiempo relativamente corto
- Ahora la computación de altas prestaciones …
  - Plataformas cada vez más asequibles
    - Sistemas cada vez más grandes
    - Clusters formados por computadores de bajo coste
    - Plataformas Grid
  - Códigos que precisan mayor tiempo de ejecución
    - Problemas de simulación y modelaje
    - Problemas que requieren la manipulación de una gran cantidad de datos

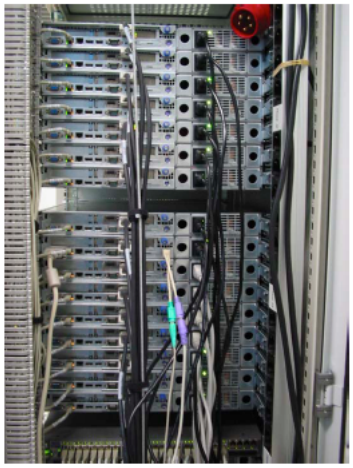


Computación paralela

- Cluster de PCs en 2000:
  - Nodos:
    - 1-4 procesadores
    - 1-4 GB memoria
  - Red de interconexión:
    - Myrinet (~200MB/s,7us)
    - Fast Ethernet (~12MB/s,1m
  - Tamaño:
    - 20 cm cada nodo

Computación paralela

- Cluster de PCs en el 2005
  - Nodos:
    - 1-8 procesadores
    - 1-32GB memoria
  - Red de interconexión:
    - Infiniband (~1GB/s,5us)
    - Gigabit Ethernet (~80MB/s,50us)
  - Tamaño:
    - 5 cm por nodo



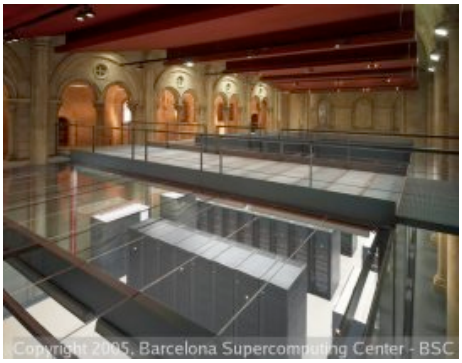
Computación paralela

Lista Top500 (www.top500.org) - Noviembre 2006

Rank	Site	Computer	Processors	Year	R_{max}	R_{peak}
1	DOE/NNSA/LLNL United States	BlueGene/L - eServer Blue Gene Solution IBM	131072	2005	280600	367000
2	NNSA/Sandia National Laboratories United States	Red Storm - Sandia/ Cray Red Storm, Opteron 2.4 GHz dual core Cray Inc.	26544	2006	101400	127411
3	IBM Thomas J. Watson Research Center United States	BGW - eServer Blue Gene Solution IBM	40960	2005	91290	114688
4	DOE/NNSA/LLNL United States	ASC Purple - eServer pSeries p5 575 1.9 GHz IBM	12208	2006	75760	92781
5	Barcelona Supercomputing Center Spain	MareNostrum - BladeCenter JS21 Cluster, PPC 970, 2.3 GHz, Myrinet IBM	10240	2006	62630	94208

Computación paralela

- MareNostrum
  - Puesto 26 del top500
  - Puesto 6 de Europa
  - BSC
  - 63.83 TFlops



Computación paralela

Lista Top500 - junio 2008:

- Tipo de máquina paralela:
  - cluster: 400 (80%)
  - MPP: 98 (19.6%)
  - constelaciones: 2 (0.4%)
- Lugar de residencia:
  - España: 7
  - Europa: 185
  - EEUU: 257

Computación paralela

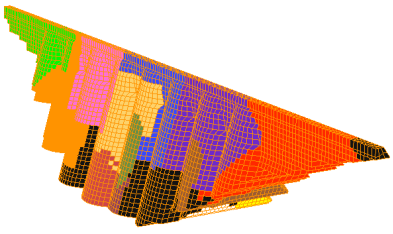
- Finisterrae
  - 2500 procesadores Itanium 2
  - 19000GB memoria
  - 390000GB en disco
  - 1PB en cinta
  - Software libre
  - Puesto 209 en el Top 500

Aplicaciones de la computación paralela

- ¿Por qué usar computación paralela?
  - Las principales razones:
    - Ganar tiempo
    - Resolver problemas más grandes
    - Conseguir concurrencia (hacer varias cosas al mismo tiempo)
  - Otras razones:
    - Sacar partido de recursos remotos
    - Ahorrar dinero, usando varios recursos computacionales "baratos" en lugar de pagar por uno potente
    - Solucionar limitaciones de memoria

Aplicaciones de la computación paralela

- Centros estatales y gubernamentales
  - Energía nuclear, modelado del clima y predicción meteorológica, modelado de los océanos y la corteza terrestre, ...
- Supercomputadores en la industria:
  - petróleo, automóvil, aeroespacial, química, electrónica, energía, farmacéutica, …
- Supercomputadores en centros de investigación:
  - Centros de Supercomputación.
  - Universidades
  - Redes: acceso a un gran número de investigadores.



Una presa de hormigón armado

Aplicaciones de la computación paralela

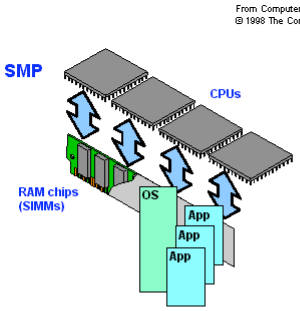
- 12.5 millones de ecuaciones lineales
- Usando un cluster de procesadores “baratos”

El número de procesadores	El tiempo para resolver la sistema de ecuaciones
2	1:21:59
16	0:10:43
128	0:01:33
256	0:00:51
320	0:00:42

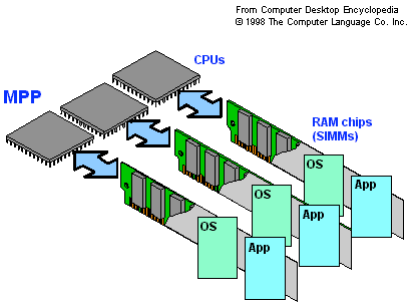
Clasificación de las arquitecturas paralelas

- Según los flujos de datos e instrucciones.
  - *Single Instruction, Single Data* (SISD).
    - Uniprocesadores.
  - *Single Instruction, Multiple Data* (SIMD).
    - Computadores matriciales (única UC, varias UP).
    - Procesadores específicos.
  - *Multiple Instruction, Single Data* (MISD)
    - No disponibles comercialmente.
  - *Multiple Instruction, Multiple Data (MIMD)*
    - Procesadores comerciales de propósito general.

Clasificación de las arquitecturas paralelas



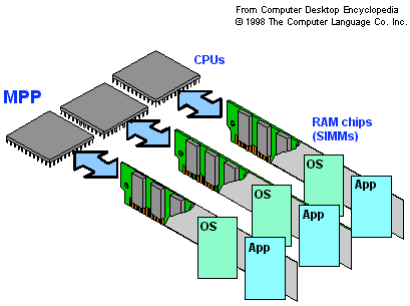
Con memoria compartida



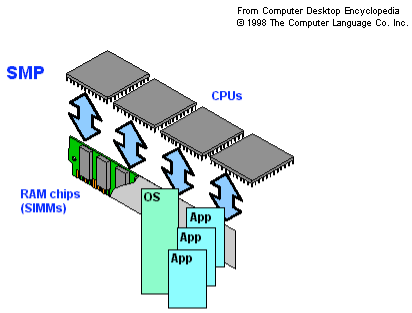
Con memoria distribuida

Clasificación de las arquitecturas paralelas

- Computadores de memoria distribuida:
  - La máquina consiste en muchos procesadores que corren el mismo programa usando datos diferentes.
  - En muchos casos, los procesadores paralelos de memoria distribuida pueden proporcionar tanto la potencia de cálculo como la cantidad de memoria necesaria para resolver este tipo de problemas.
  - En el taxonomía de Flynn, se llaman DM-MIMD: MIMD son las iniciales de múltiple instrucción, múltiples datos



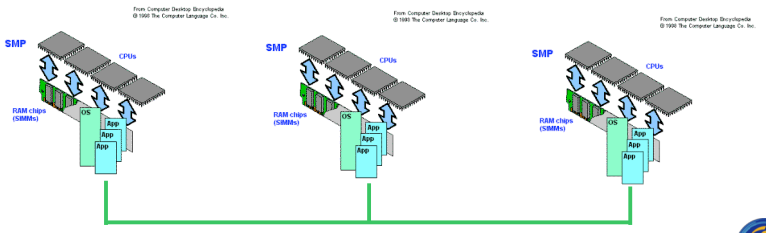
Clasificación de las arquitecturas paralelas



- Computadores de memoria compartida:
  - A diferencia de la arquitectura anterior, hay una única memoria a la cual acceden todos los procesadores.
  - Es decir, no hay memorias privadas, sino una única memoria compartida.

Clasificación de las arquitecturas paralelas

- Computadores *híbridos*:
  - Se pueden combinar estos dos formas básicas de arquitectura tal como Mare Nostrum que consiste en muchos nodos en los cuales hay 32 procesadores que comparten la memoria.
  - Un procesador en un nodo no tiene acceso a la memoria de otra nodo.



Modelos de programación paralela

- La computación en paralelo es una técnica que nos permite distribuir una gran carga computacional entre muchos procesadores.
- Y es bien sabido que una de las mayores dificultades del procesamiento en paralelo es la coordinación de las actividades de los diferentes procesadores y el intercambio de información entre los mismos.
- Dependiendo de la arquitectura de la máquina (memoria distribuida o compartida), existen diferentes técnicas para programar en paralelo e intercambiar información entre procesadores

Modelos de programación paralela

- Existen diferentes alternativas:
  - pase-de-mensajes
  - memoria compartida (hilos paralelos o pthreads)
  - paralelismo de datos (Data Parallel)
  - Híbrida
- Aunque pueda no resultar aparente, estos modelos no son específicos de una arquitectura particular. En teoría al menos, todos ellos se pueden implementar en cualquier arquitectura.

Pase de mensajes

- Multiprocesadores de memoria distribuida y redes de computadores (sistemas heterogéneos).
- Primitivas para:
  - Acceso a la memoria local de otros procesadores.
  - Sincronización.
- Flexible: Control absoluto del programa
- Inconveniente: El programador es responsable la optimización del programa.
  - Distribución de los datos y las computaciones, sincronización, comunicaciones, ...
- Normalmente, mayor rendimiento

Pase de mensajes

- Librerías estándar de pase de mensajes:
  - PARMACS, PVM (Parallel Virtual machine) y MPI (Message Passing Interface)
  - Existen librerías no estándar, desarrolladas para un multiprocesador determinado.
- Lenguaje C o Fortran con rutinas de pase de mensajes

MPI. Producto escalar

```
! initialize MPI
PROGRAM dot_prod
  include "mpif.h"
...
call MPI_INIT(ierr)
call MPI_COMM_RANK(MPI_COMM_WORLD,myid,ierr)
call MPI_COMM_SIZE(MPI_COMM_WORLD,nprocs,ierr)
...
! everyone computes a local dot product
dot_l = 0.d0
do i=1,M
    dot_l = dot_l + a(i) * b(i)
enddo

! now sum up the contributions on processor 0
call MPI_REDUCE(dot_l,dot,1, &
          MPI_DOUBLE_PRECISION, &
              MPI_SUM,0,MPI_COMM_WORLD,ierr)
call MPI_FINALIZE(ierr)
END PROGRAM dot_prod
```

Memoria compartida

- Mecanismo de bajo nivel: pthreads.
- Primitivas:

pthread_create	pthread_cond_init
pthread_exit	pthread_cond_destroy
pthread_join	pthread_cond_wait
pthread_mutex_init	pthread_cond_signal
pthread_mutex_destroy	pthread_cond_broadcast

Memoria compartida: OpenMP

- Standard de programación en memoria compartida.
- Incluye directivas (#pragmas) de compilación junto con funciones de librería.
- Se basa en la semántica de *fork-join*.
- En una plataforma secuencial, las directivas son ignoradas (comentarios).

OpenMP. Producto escalar

Programa Secuencial

```
Void main(){
  double dot,a[1000],b[1000]
  for(int i=0; i<1000; i++){
    dot += a[i]*b[i];
  }
}
```

Programa Paralelo

```
Void main(){
  double dot,a[1000],b[1000]
  #pragma omp parallel for \
       reduction(+:dot)
  for(int i=0; i<1000; i++){
    dot += a[i]+b[i]
  }
}
```

Comparativa

- Memoria compartida:
  - Más sencilla en su programación.
  - Más difícil cometer errores.
  - Las estructuras de datos cambian.
  - Se puede paralelizar incrementalmente.
- Pase de mensajes:
  - Más fácil obtener altos rendimientos.
  - Mayor portabilidad.
  - Dificultad en detectar y corregir errores.

Data parallel

- Simplifica la programación con pase de mensajes.
  - Código secuencial con espacio de direcciones global.
  - Lenguaje estándar (C o Fortran)
  - Se genera código paralelo mediante directivas de compilación (p.e distribución de los datos).
- El compilador genera el código paralelo mediante:
  - Distribución y partición de los datos (especificado por el programador)
  - Distribuyendo computaciones.
  - Insertando comunicaciones por pase de mensajes o memoria compartida.

Data parallel

- Fáciles de programar.
- Prueba con varias distribuciones de datos.
- Comunicaciones basadas en primitivas nativas del sistema ==> Buen rendimiento
- Lenguajes:
  - CM Fortran, Vienna Fortran .
  - HPF (*High Performance Fortran*)

HPF

- Standard para paralelismo de datos.
- Extensión del programa secuencial con directivas de distribución de datos entre los procesadores.
- El compilador produce el programa paralelo con las comunicaciones necesarias.
- En una plataforma secuencial, las directivas son ignoradas (comentarios).
- Desventajas:
  - Difícil obtener altos rendimientos.
  - No optimiza las comunicaciones.
  - No sirve para computaciones irregulares.

HPF. Producto escalar

```
PROGRAM dot_prod_hpf

INTEGER, PARAMETER           :: N = 1024
INTEGER                      :: i
REAL(KIND=8)                 :: dot
REAL(KIND=8), DIMENSION(N) :: a, b,c

!HPF$ ALIGN (:) WITH c(:) :: a,b
!HPF$ DISTRIBUTE(BLOCK)    :: c

dot_l = 0.d0

FORALL (i=1:N)
  c = a * b
END FORALL

dot = sum( c(1:N) )

END PROGRAM dot_prod_hpf
```

Paralelización automática

- El compilador genera de forma automática la versión paralela de un código secuencial.
- Herramientas:
  - Parafrase, SuperB, SUIF, Polaris …
- Todavía no hay productos comerciales.
- Campo puntero en la investigación.

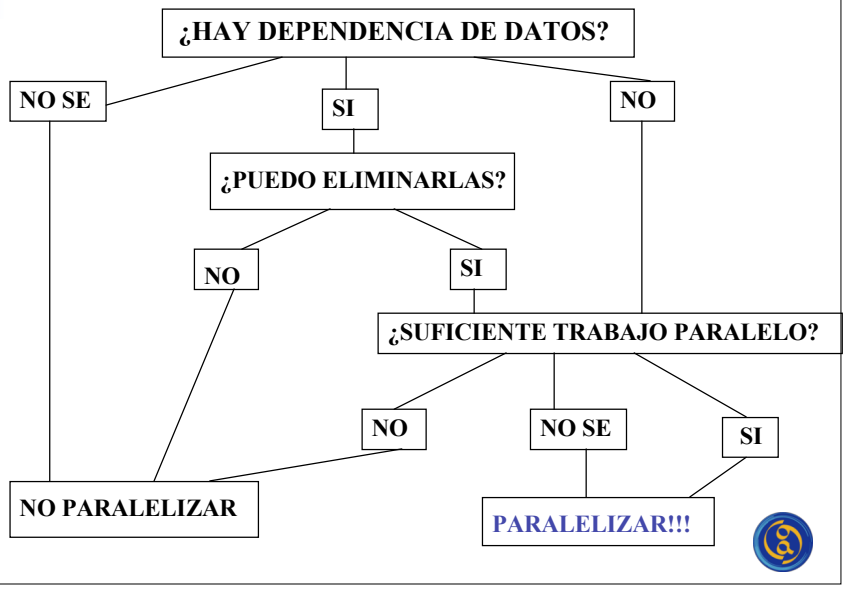
Otros modelos

- Híbrido: combina dos o más de los anteriores
  - Ej: OpenMP para comunicaciones intra-nodo, MPI para comunicaciones inter-nodo
- SPMD: simple programa-múltiples datos
  - Modelo de programación a más alto nivel
  - El mismo programa es ejecutado en todos los procesadores simultáneamente.
  - Suelen tener internamente programadas las condiciones necesarias para permitir que diferentes tareas salten o ejecuten condicionalmente cierta parte del código
- MPMD: múltiples programas-múltiples datos
  - Existen diferentes ejecutables para diferentes tareas. Aunque se ejecutan en paralelo cada tarea puede ejecutar un código diferente o el mismo código que otras tareas.

Diseño de programas paralelos

- Encontrar los cuellos de botella:
  - Aquellos puntos donde se pierde desproporcionadamente tiempo
- Identificar los “inhibidores” de paralelismo:
  - Por ejemplo, las dependencias, como en el ejemplo de la serie de Fibonacci.
  - Investigar otros algoritmos si fuese posible.

Diseño de programas paralelos



Diseño de programas paralelos

- Particionamiento:
  - Uno de los primeros pasos para diseñar programas paralelos es dividir el problema en una serie de piezas de trabajo que puedan ser distribuidas en múltiples tareas.
  - Esto se conoce como descomposición o particionamiento:
  - Descomposición de dominios: se particionan los datos asociados al problema. Cada tarea paralela trabaja con una porción de los datos.
  - Descomposición funcional: el problema se particiona en función del trabajo que se debe hacer. Cada tarea realiza una parte del trabajo total.

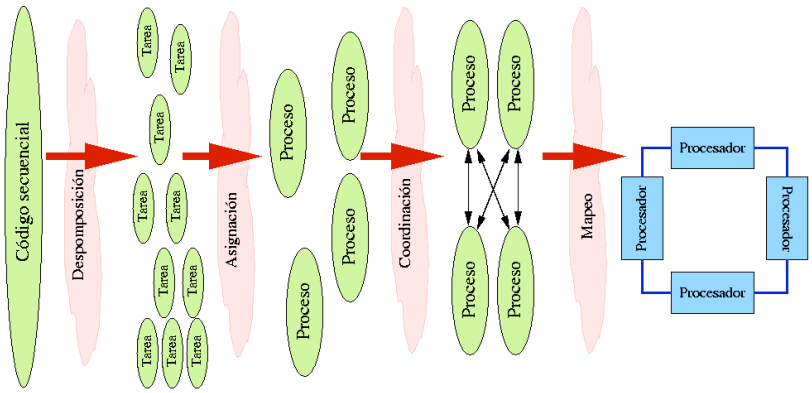
Diseño de programas paralelos

- Comunicaciones:
  - La necesidad de comunicaciones entre las tareas depende del problema
  - NO se necesitan comunicaciones cuando no hay que compartir datos entre las tareas. Problemas embarazosamente paralelos.
  - La mayoría de las aplicaciones paralelas requieren compartir datos entre las tareas, y por lo tanto será necesario algún tipo de comunicación entre ellas.

Diseño de programas paralelos

- Sincronización:
  - Tipos de sincronización:
    - Barrera
    - Lock / semáforo
    - Operaciones de comunicación síncronas

Fases



Ejemplo práctico

```
DO I=1,N
  DO J=1,N
    Y(I)=Y(I)+A(I,J)*X(J)
  END DO
END DO
```

PRODUCTO MATRIZ--VECTOR

A(I,J) es sólo de lectura

Y(I) es de lectura y escritura !!!

↓

Paralelizar lazo EXTERNO

Descomposición

- Dividir las computaciones en tareas para ser distribuidas entre los procesadores.
  - Tarea = trozo arbitrario que forma parte del trabajo total
  - Balanceo de la carga
  - Número de tareas que se pueden evaluar al mismo tiempo => limite superior a la aceleración del programa.

Ejemplo práctico

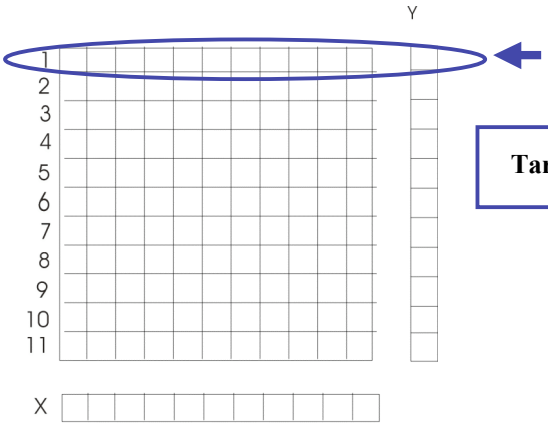
```
DO I=1,N
  DO J=1,N
    Y(I)=Y(I)+A(I,J)*X(J)
  END DO
END DO
```

PRODUCTO MATRIZ--VECTOR

Paralelizar lazo EXTERNO

TAREA ⇄ **lazo INTERNO**

Ejemplo práctico



Tarea → computar fila

Ejemplo: 11 filas → 11 tareas

Asignación

- Mecanismo que especifica como se divide el trabajo entre los procesadores.
- Junto con la descomposición recibe el nombre de *particionamiento.*
- Asignación *estática* vs *dinámica.*
- Normalmente es independiente de la arquitectura.

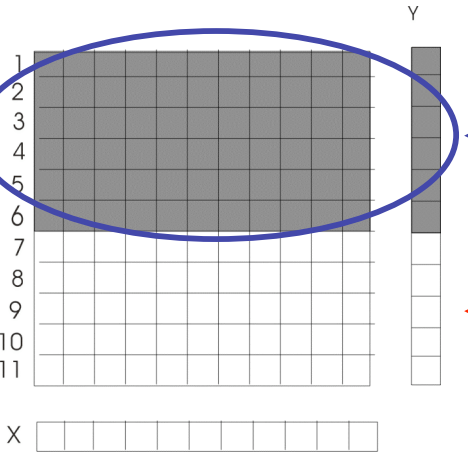
Ejemplo práctico

```
DO I=1,N
   DO J=1,N
      Y(I)=Y(I)+A(I,J)*X(J)
   END DO
END DO
```

PRODUCTO
MATRIZ--VECTOR

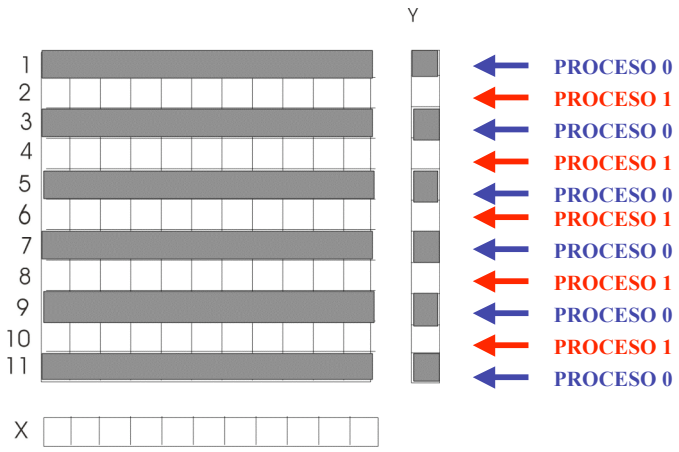
TAREA ⇄ lazo INTERNO

GRUPO DE LAZOS ⇄ PROCESO



BLOQUE

Ejemplo práctico

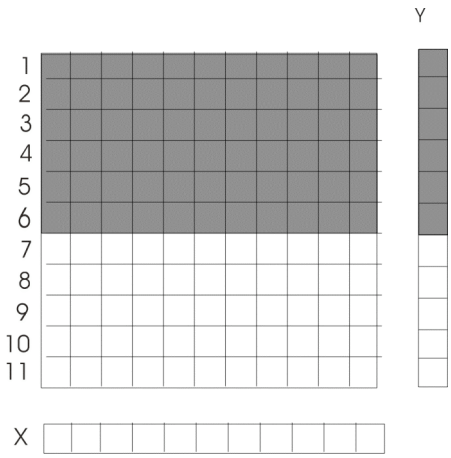


CICLICO

Coordinación

- Objetivo: reducir el coste de comunicaciones y sincronizaciones.
- Aquí las decisiones dependen de la eficiencia de las primitivas.
- Paso muy próximo a la arquitectura.

Ejemplo práctico



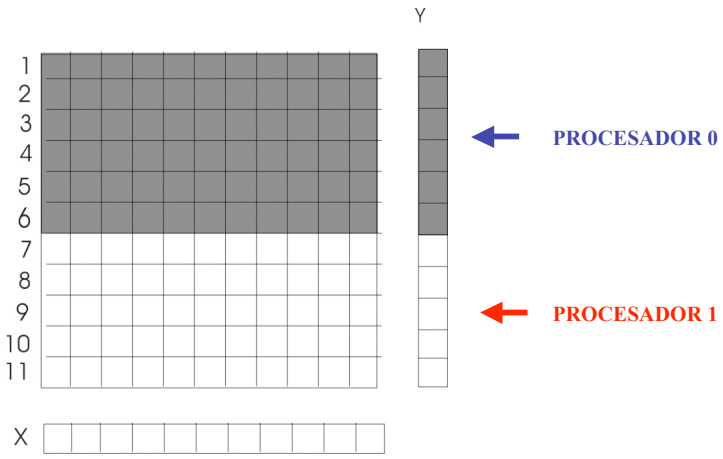
PROCESO 0

PROCESO 1

Mapeo

- Decidir que proceso se ejecutará en cada procesador.
  - *Mapear* a una topología.
- Normalmente:
  proceso <-> procesador

Ejemplo práctico



Ejemplo práctico

- Procesamiento de un array
  - Supongamos cálculos en un array de dos dimensiones, donde el cálculo de cada elemento se realiza de forma independiente al de los demás
  - Trivial de paralelizar (embarrassingly parallel)

```
do i=1,n
   do j=1,n
      A(i,j)=fcn(i,j)
   end do
end do
```

Ejemplo práctico

- Solución
  - Los elementos del array se distribuyen de forma que cada procesador tiene su propia porción de array
  - Se llevan a cabo los cálculos de forma independiente entre los procesadores, sin necesidad de comunicaciones entre ellos
  - El esquema de distribución de los elementos puede ser bloque, cíclico, o cualquier otro
  - Después de la distribución cada procesador ejecuta la porción del lazo que le corresponde

Ejemplo práctico

- Solución I: SPMD

```
Encontrar si soy maestro o esclavo
if soy maestro
        inicializar el array
        enviar a cada esclavo su parte del array
        recibir de cada esclavo los resultados
else
        recibir del maestro la parte del array
        #calcular mi parte
        do j=mi primera columna, mi ultima columna
          do i= 1, n
            A(i,j)=fcn(i,j)
          end do
        endo do
        enviar al maestro os resultados
end if
```

Ejemplo práctico

- Solución II: piscina (pool) de tareas

```
Encontrar si soy maestro o esclavo
if soy maestro
        do hasta que no haya mas trabajos
          enviar al esclavo el siguiente trabajo
          recibir de los esclavos los resultados
        end do
        comunicar a los esclavos que no hay mas trabajos
else
        do hasta que no haya mas trabajos
          recibir del maestro un trabajo
          calcular el elemento del array: A(i,j)=fcn(i,j)
          enviar los resultados al maestro
        endo do
end if
```

Sesión de laboratorio

Lab 6

Bibliografía

- "Parallel programming". B. Wilkinson, M. Allen. Prentice-Hall 1999
- "Introduction to Parallel Computing". A. Grama, A. Gupta, G. Karypis, V. Kumar. Addison-Wesley, 2003
- "Scientific Computing: An Introduction with Parallel Computing". G. Golub y J.M. Ortega. Academic Press 1993

Conclusiones

- La computación de altas prestaciones ha sufrido un cambio en los últimos años: las plataformas son más accesibles y los problemas con los que trata más complejos.
- Existen varios tipos de arquitecturas paralelas y diferentes paradigmas de programación paralela.
- La elección del paradigma de programación depende de varios factores:
  - Arquitectura sobre la que se va a ejecutar
  - Ciertas características del algoritmo/problema en cuestión
  - Experiencia del programador
- La paralelización de un código suele ser un trabajo complejo, que requiere un conocimiento previo del problema y del código secuencial a paralelizar, además de la experiencia con el paradigma de programación elegido.