



Computación de Altas Prestaciones

Optimización en microprocesadores

José Luis Risco Martín

Dpto. Arquitectura de Computadores y Automática
Universidad Complutense de Madrid

This work is derivative of “Optimización en Microprocesadores”
by [Ignacio Martín Llorente](#), licensed under [CC BY-SA 4.0](#)



Índice

1. Introducción
2. Objetivos de la optimización monoprocesador
3. Preguntas previas a la optimización de un código
4. Recomendaciones generales
5. Técnicas directas de optimización
6. Optimización de la jerarquía de memoria
7. Optimización de la arquitectura del procesador
8. Optimización de la entrada/salida
9. Ejemplos de herramientas de ayuda a la optimización
10. Pasos en la optimización secuencial





Introducción

- Algunas de las recomendaciones presentadas en el tema son generales, ¿es necesario ajustarlas a las características de la plataforma?
 - Falta de portabilidad
 - Incremento de rendimiento escaso
 - El cuello de botella es el mismo: acceso a memoria
- Algunas de las optimizaciones entran en conflicto
- Algunas de las optimizaciones presentadas son realizadas por el compilador
 - **El compilador es un programa:** Cuanto más le ayudemos mejores resultados se obtendrán

“Debemos dar de comer al procesador”

“En los microprocesadores la principal fuente de ineficiencia es el tiempo de acceso a memoria”

“En los multiprocesadores actuales la principal fuente de ineficiencia ha pasado de ser la red de interconexión al acceso a memoria”

Objetivos de la Optimización Monoprocesador



Conseguir que el procesador esté continuamente trabajando
Reducir el tiempo total de ejecución



Reducir el tamaño del programa

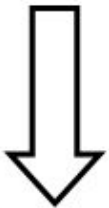
Preguntas Previas a la Optimización de Código



¿**Por qué** el código no se ejecuta eficientemente?

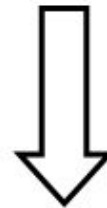


- Procesador
- Entrada/salida
- Memoria



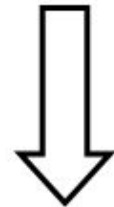
time

¿**Donde** no se ejecuta eficientemente?



profiler

¿**Cómo** se puede resolver?



•Técnicas de optimización



Recomendaciones Generales (1/2)

- Lo primero es saber en **qué parte del programa se consume la mayoría del tiempo**
 - Si una rutina A consume el 10% y otra B el 80% debemos centrarnos en B
 - Es muy importante recordar la ley de Amdahl:
 - “Si s es la fracción de código que no puede ser optimizada, incluso con un infinito nivel de optimización la ganancia siempre será menor que $1/s$ ”
- **Invertir el tiempo en** las zonas de programa más costosas computacionalmente (**puntos calientes**)
 - Estos puntos se detectan por medio de:
 - Uso de las herramientas de profiling del sistema: prof, gprof, tconv, ...
 - Rutinas de medida de tiempos: time, etime, ...
 - Cuando queremos disminuir el tiempo de ejecución tenemos que conocer perfectamente cuál es la causa de la ineficiencia
- **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 paquetes de rutinas existentes (IMSL, LAPACK, BLAS, ...)
 - Muchas son de dominio público !!!



Recomendaciones Generales (2/2)

■ Ejemplo de bibliotecas matemáticas

EJEMPLO DE BIBLIOTECAS MATEMÁTICAS (compilador Irix SGI)

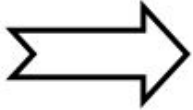
<u>librería</u>	<u>manual</u>	<u>comentario</u>
fastmath	<i>libfastm(3M)</i>	Versiones muy rápidas para sin, cos, tan, ...
complib	<i>complib(3F)</i>	FFT, convoluciones, BLAS, LAPACK, resolutores...
SCSL	<i>intro_libscsl(3S)</i>	FFT, BLAS, LAPACK
vector intrinsics	<i>f77(1), cc(1)</i>	vsin, vcos, ...; permiten vectorización automática con -O3

Técnicas Directas de Optimización (1/2)



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

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



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

- Mejor usar constantes que variables
 - El compilador realiza una mejor optimización
- La exponenciación debe evitarse cuando sea posible
- Evitar las conversiones de tipo
- No usar IF aritmético en Fortran
- Si el programa consume mucho tiempo en una expresión
 - Igual se puede sustituir por una tabla
- Eliminar los saltos, llamadas a subrutinas y operaciones de entrada/salida dentro de los bucles

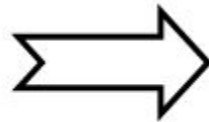
La aplicación de estas optimizaciones dentro de un bucle puede dar lugar a grandes ganancias

Técnicas Directas de Optimización (2/2)



- Hacer el menor número posible de llamadas a subrutinas
- **Problemas:**
 - Producen fallos en la memoria cache de instrucciones
 - Las llamadas a subrutinas tienen un alto overhead (salto y almacenamiento de contexto)
 - Disminuye la posibilidad de que el compilador optimice (y paralelice)
- **Soluciones:**
 - Copiar el código
 - Introducir el bucle dentro de la función
 - Sacar la función fuera del bucle

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

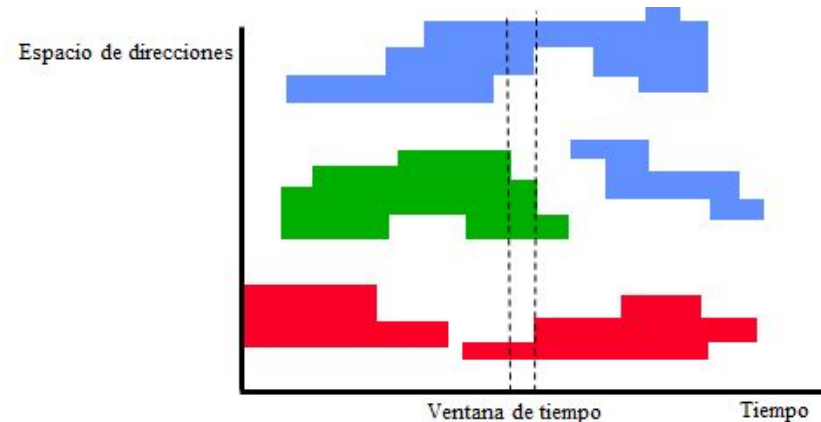
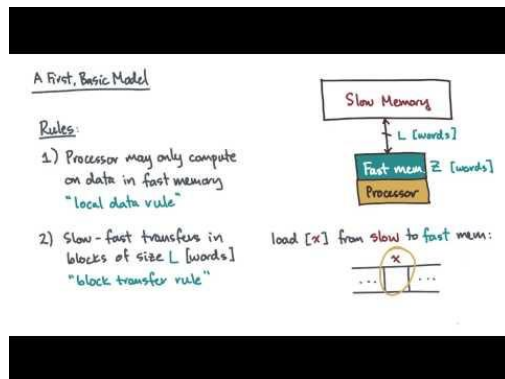


```
E = log(3)  
FOR I=1,N  
  A(I) = B(I)+E  
ENDDO
```

Repaso de Jerarquía de Memoria (1/2)

■ Propiedad de localidad de los programas

- Referencias concentradas en regiones de tiempo y espacio
- Resultados experimentales:
 - El 90% del tiempo se suele consumir en el 10% del código (bucles) [HePa96]
 - Referencias a memoria por programas de prueba [Hwang93]:



■ Tipos de localidad

- **Localidad espacial** (secuencial) : Direcciones próximas
 - Acceso a instrucciones en programas; y a datos en arrays o tablas
- **Localidad temporal**: Misma dirección
 - Bucles y subrutinas en los programas; y variables temporales

Repaso de Jerarquía de Memoria (2/2)



[Hennessy 2011]

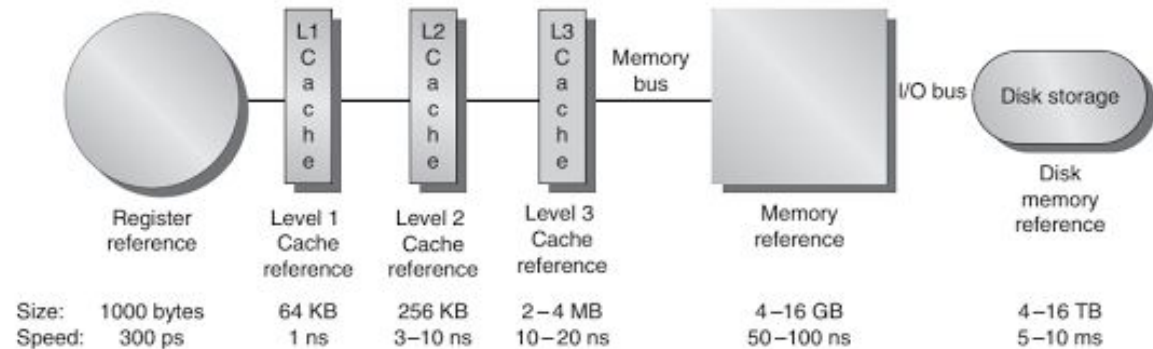
Según nos alejamos del procesador, la memoria del nivel inferior es más lenta y de mayor capacidad.

Las unidades de tiempo varían en un factor de 10^9 (de picosegundos a milisegundos)

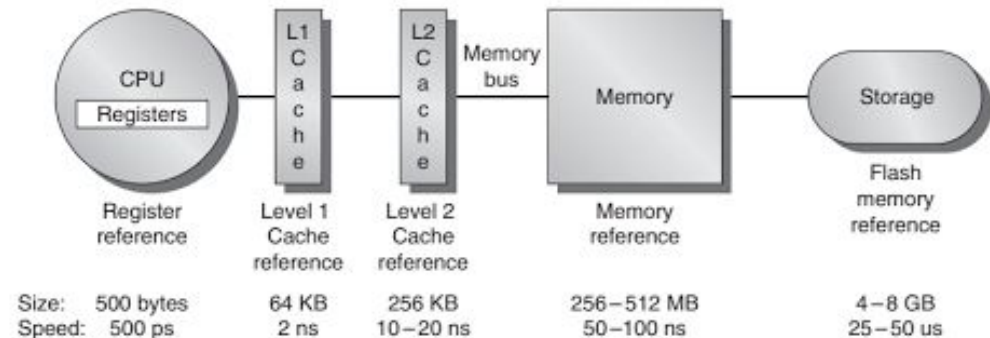
Las unidades de tamaño varían en un factor de 10^{12} (de bytes a terabytes)

El dispositivo móvil tiene un reloj más lento y memorias de menos tamaño.

El servidor usa almacenamiento en disco, mientras que el móvil usa memoria flash (basada en tecnología EEPROM).



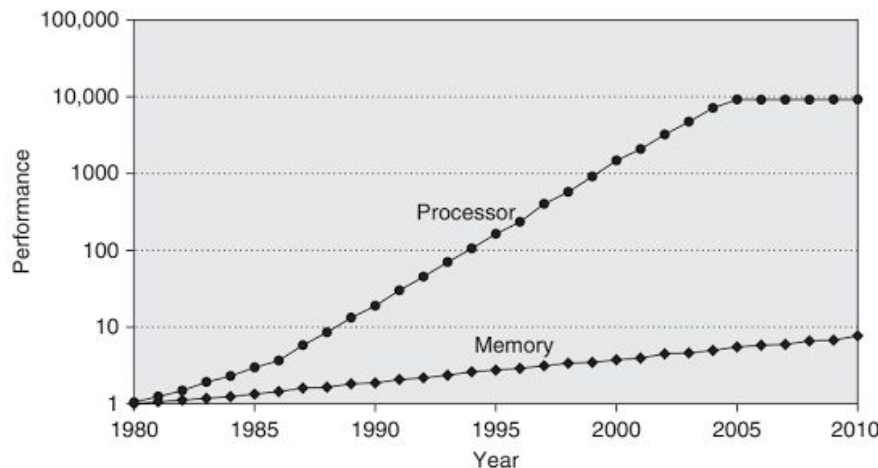
(a) Memory hierarchy for server



(b) Memory hierarchy for a personal mobile device

Repaso de la memoria cache (1/2)

- La diferencia entre la velocidad del procesador y de memoria crece constantemente
- **Crecimiento de las velocidades de procesador y memoria**



Crecimiento de rendimientos del procesador y latencia de memoria respecto a un computador de 1980 [HePa11]:

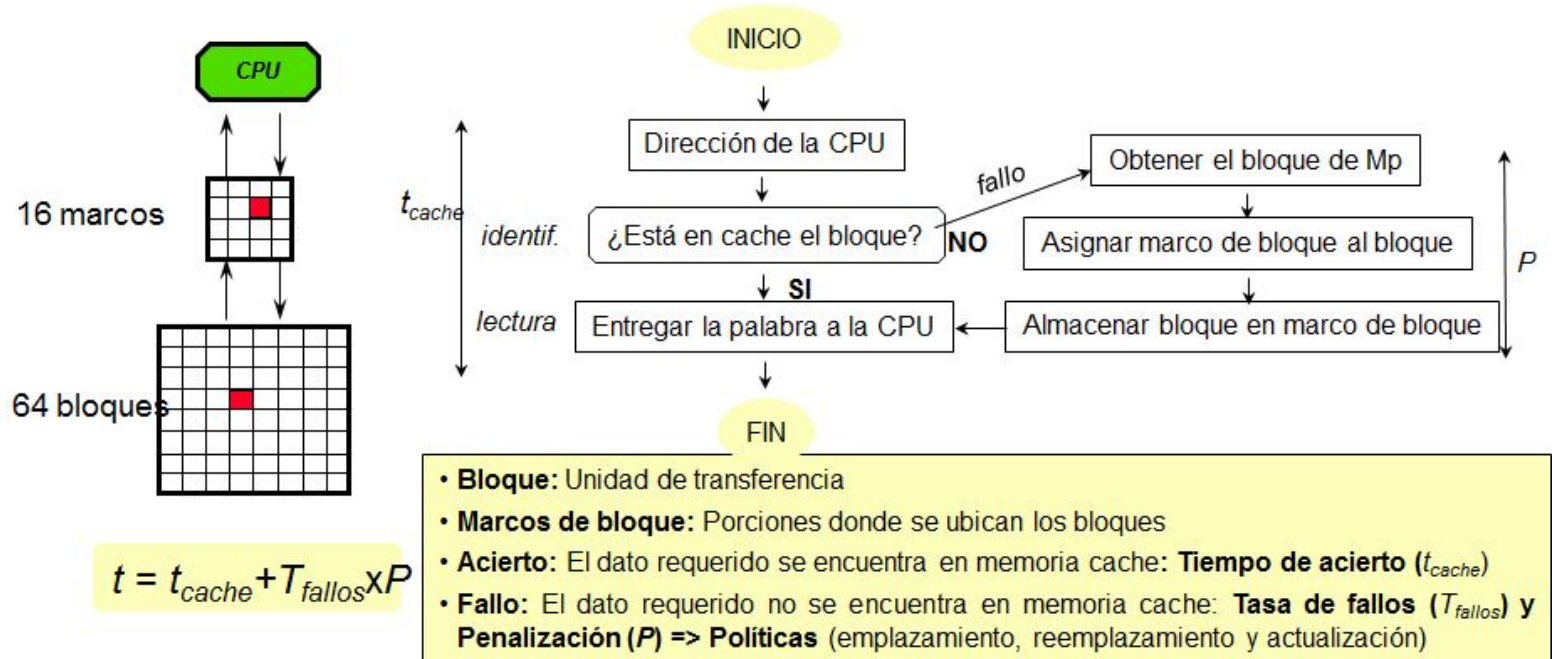
- Memoria: +1.07/año
- Procesador:
 - Hasta 1986: +1.25/año
 - Hasta 2000: +1.52/año
 - Hasta 2005: +1.20/año
 - Hasta 2010: +0.00/año

- El diseño y uso de la memoria cache es crítico en el computador
 - vacío procesador- memoria principal

Repaso de la memoria cache (2/2)

- Memoria pequeña y rápida situada entre el procesador y la memoria principal
 - Almacena la información actualmente en uso de la memoria
 - **Objetivo:** Disminuir el tiempo de acceso a memoria
 - Primer computador: IBM 360/85 (1969)

CARACTERÍSTICAS DE UNA MEMORIA CACHE



Reglas Fundamentales para Optimizar el Uso de la Memoria Cache



- **Optimizar la localidad espacial**
 - Cuanto más pequeño el stride de acceso a un array mejor: óptimo stride igual a 1
- **Optimizar la localidad temporal**
 - Utilizar los datos todo lo que se pueda una vez que los hemos traído
- **Si traemos un dato de memoria tenemos que usarlo todo lo posible**

Optimización de la Memoria Cache (1/5)



■ Técnicas por parte del compilador o del programador

- El objetivo es mejorar la localidad espacial y temporal de los programas:
- “Usar los datos que se encuentra en cache antes de desecharlos”

$$t = t_{\text{cache}} + T_{\text{fallos}} \times P$$

■ Ejemplo 1

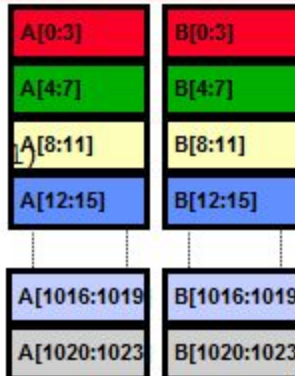
- DEC Alpha 21064: Mc de 8 KB, directa, con 256 bloques de 4 pal. de 64 bits
- Cada 1024 palabras se repite la asignación de marcos de bloques.

■ Fusión de arrays

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

```
double A[1024];  
double B[1024];  
  
for (i = 0; i < 1024; i = i + 1)  
    C = C + (A[i] + B[i]);
```

2x1024 fallos
2x256 de inicio
1536 de conflicto

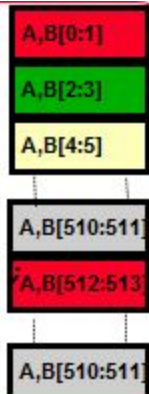


```
struct fusion{  
    double A;  
    double B;  
} array[1024];
```

```
for (i = 0; i < 1024; i = i + 1)  
    C = C + (array[i].A + array[i].B)
```

1024/2 fallos
2x256 de inicio

Ganancia: 4

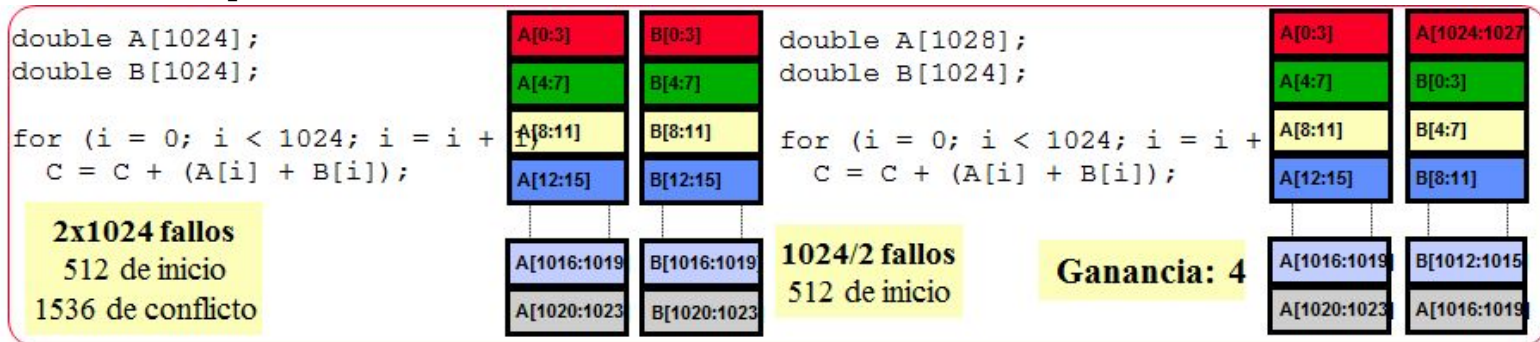


Optimización de la Memoria Cache (2/5)



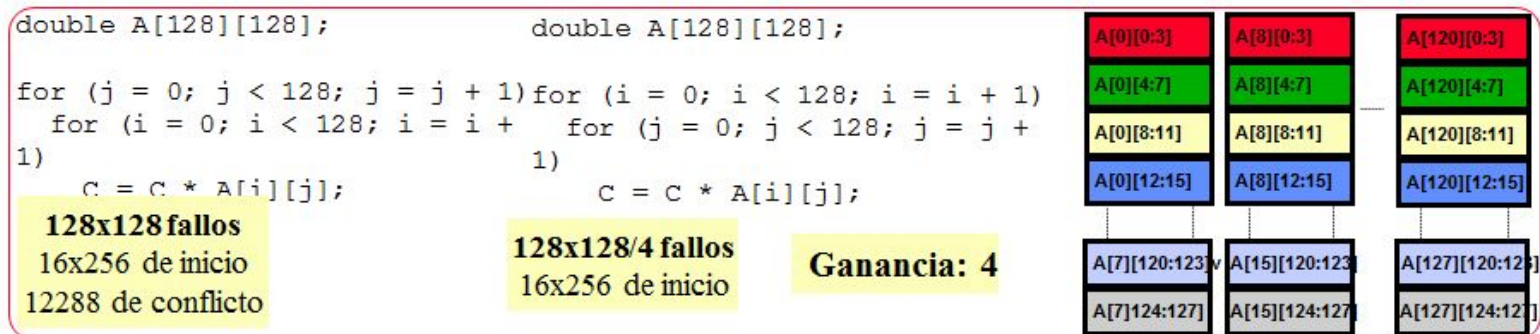
■ Alargamiento de arrays

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



■ Intercambio de bucles

- Mejora la **localidad espacial** para disminuir los **fallos de conflicto**
 - En lenguaje C las matrices se almacenan por filas, luego se debe variar en el bucle interno la columna



Optimización de la Memoria Cache (3/5)



■ Fusión de bucles

- Mejora la localidad temporal para disminuir los fallos de capacidad
 - Fusionar los bucles que usen los mismos arrays para usar los datos que se encuentran en cache antes de desecharlos

```
double A[64][64];
```

```
for (i = 0; i < 64; i = i + 1)
  for (j = 0; j < 64; j = j + 1)
    C = C * A[i][j];
```

```
for (i = 0; i < 64; i = i + 1)
  for (j = 0; j < 64; j = j + 1)
    D = D + A[i][j];
```

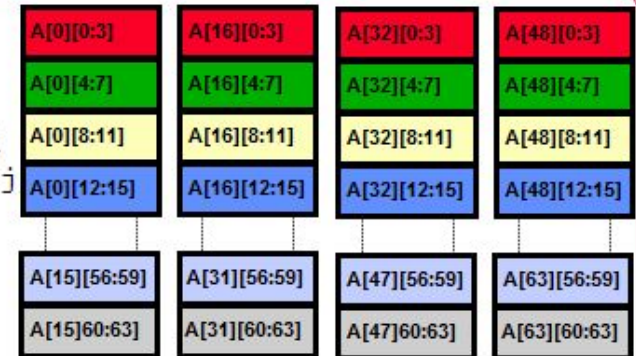
(64x64/4)x2 fallos
4x256 de inicio
4x256 de capacidad

```
double A[64][64];
```

```
for (i = 0; i < 64; i = i + 1)
  for (j = 0; j < 64; j = j + 1)
  {
    C = C * A[i][j];
    D = D + A[i][j];
  }
```

64x64/4 fallos
4x256 de inicio

Ganancia: 2



Optimización de la Memoria Cache (4/5)



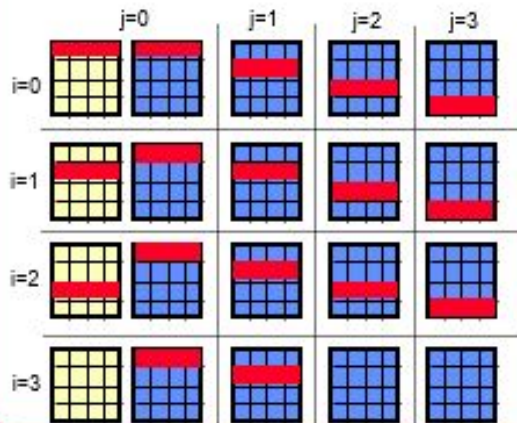
■ Ejemplo 2

- Procesador con memoria cache de 128 bytes totalmente asociativa (reempl. LRU), 4 marcos y tamaño de bloque de 4 palabras de 64 bits

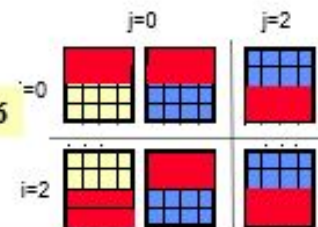
■ Bloqueo de matrices

- Mejora la **localidad temporal** para disminuir los **fallos de capacidad**
 - Operar con submatrices para usar los datos que se encuentran en cache antes de desecharlos

```
double A[4][4];  
double B[4][4];  
  
for (i = 0; i < 4; i = i + 1)  
  for (j = 0; j < 4; j = j + 1)  
    A[i][j] = A[i][j] + B[j][i];
```



```
double A[4][4];  
double B[4][4];  
  
for (i = 0; i < 4; i = i + 2)  
  for (j = 0; j < 4; j = j + 2)  
  {  
    A[i][j] = A[i][j] + B[j][i];  
    A[i][j+1] = A[i][j+1] + B[j+1][i];  
    A[i+1][j] = A[i+1][j] + B[j][i+1];  
    A[i+1][j+1] = A[i+1][j+1] + B[j+1][i+1];  
  }
```



Ganancia: $20/12=1,66$

Optimización de la Memoria Cache (5/5)



BIT REVERSAL

```
double A[4][4];
double B[4][4];

for (i=0; i<4; i=i+1)
  for (j=0; j<4; j=j+1)
    A[i][j] = A[i][j] + B[j][i]
```

Ganancia = 20/8

Únicamente supone un cambio de significado
(inicializar B como B^T)

Esta técnica puede usarse siempre que el número de dimensiones de los arrays sea igual al número de bucles anidados.

```
double A[4][4];
double B[4][4];

for (i=0; i<4; i=i+1)
  for (j=0; j<4; j=j+1)
    A[i][j] = A[i][j] + B[i][j]
```

Repaso de la Memoria Virtual (1/2)

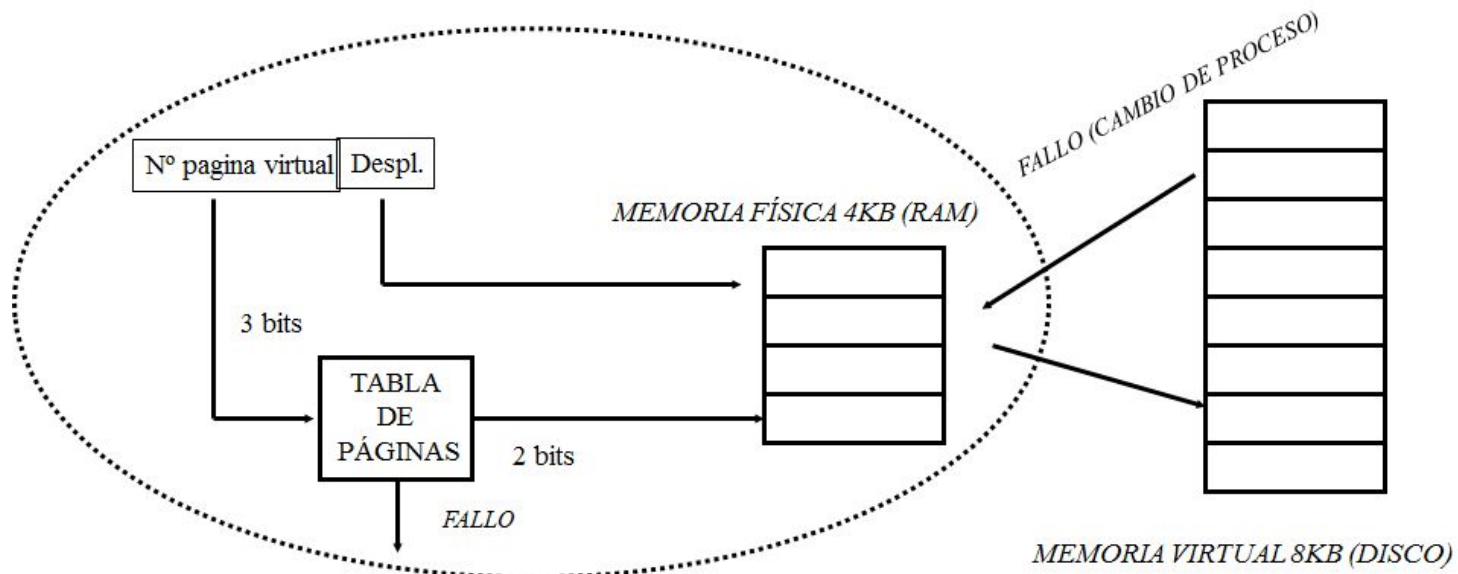


■ La memoria virtual:

- Crea al programador la ilusión de un espacio de memoria propio e ilimitado (espacio virtual)
- Facilita la compartición de la memoria por varios procesos

■ Funcionamiento:

- Direcciones que genera un procesador son virtuales, se deben traducir a direcciones físicas
- La traducción se gestiona dividiendo el espacio en páginas



Repaso de la Memoria Virtual (2/2)



- Mismas técnicas que las aplicadas en el caso de memoria cache
- En este caso la penalización por fallo es muy superior
 - **Efecto del paginamiento de memoria** (nseg-useg \Rightarrow mseg-seg)
- **Ejemplo 3:**
 - Problema:
 - $C \leftarrow C + A * B$, donde A, B, C son matrices 1024×1024
 - Sistema con memoria virtual:
 - 1 página: 2^{16} palabras = 64 columnas de A, B o C
 - ✓ Si se accede secuencialmente en columnas se produce un fallo de página cada 64 columnas
 - 1 fallo de página: 0.5 segundos
 - Almacenamiento de A, B y C: $3 \times 1024 \times 1024$ palabras
 - Capacidad de la memoria principal: 16 páginas = 1024×1024 palabras = una matriz

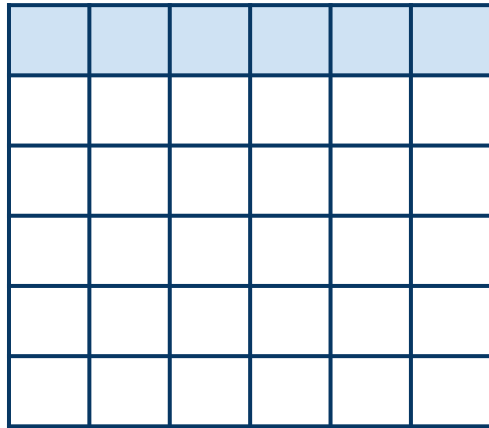
Optimización de la Memoria Virtual

Bucle IJK

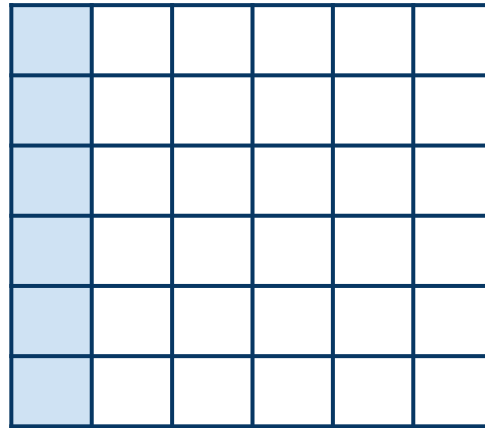


```
DO I=1,N
  DO J=1,N
    DO K=1,N
      C(I,J) = C(I,J)+A(I,K)*B(K,J)
    ENDDO
  ENDDO
ENDDO
```

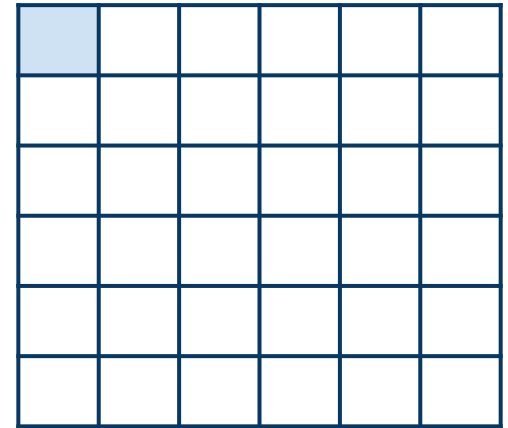
*Nota: Para calcular cada posición de C utiliza todos los elementos de una fila de A y todos los elementos de una columna de B.
(Multiplicación de matrices)*



A



B



C

Optimización de la Memoria Virtual

Bucle IJK



Fallos iniciales

3 fallos (Carga A,B,C)

×					

A

×					

B

×					

C

Nota: Para facilitar la comprensión del ejemplo probamos que ocurre con una matriz de 6x6, donde 1 página equivale a 2 columnas de la matriz.

Optimización de la Memoria Virtual

Bucle IJK



Por cada fila de A aparecen 3 errores
(las 3 cargas que se necesitan para
tener la matriz completa)

×	✓	×	✓	×	✓

A

×					
✓					
✓					
✓					
✓					
✓					

B

×					

C

2 fallos (Carga B,C)
3 fallos (Cargas A)

5 fallos

Nota: Para facilitar la comprensión del ejemplo probamos que ocurre con una matriz de 6x6, donde 1 página equivale a 2 columnas de la matriz.

Optimización de la Memoria Virtual

Bucle IJK



×	✓	×	✓	×	✓

A

	✓				
	✓				
	✓				
	✓				
	✓				
	✓				

B

×	✓				

C

3 fallos (Cargas A)

Nota: Para facilitar la comprensión del ejemplo probamos que ocurre con una matriz de 6x6, donde 1 página equivale a 2 columnas de la matriz.

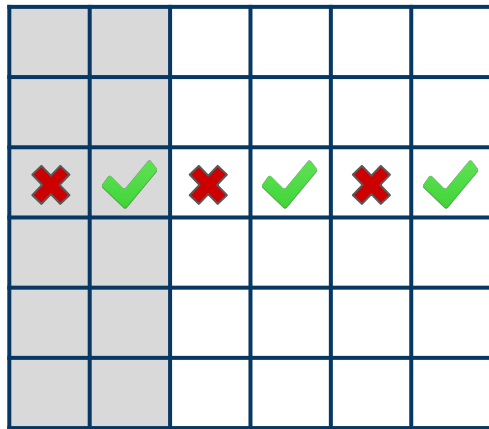
Optimización de la Memoria Virtual

Bucle IJK

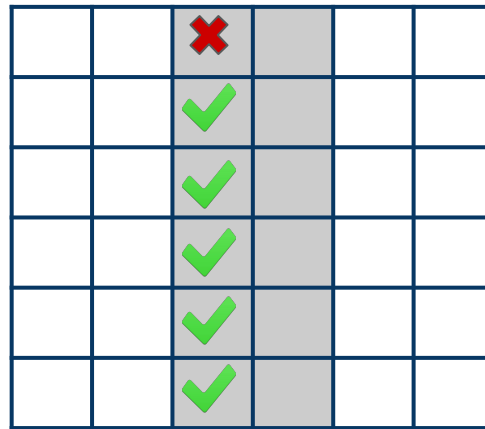


2 fallos (Carga B,C)
3 fallos (Cargas A)

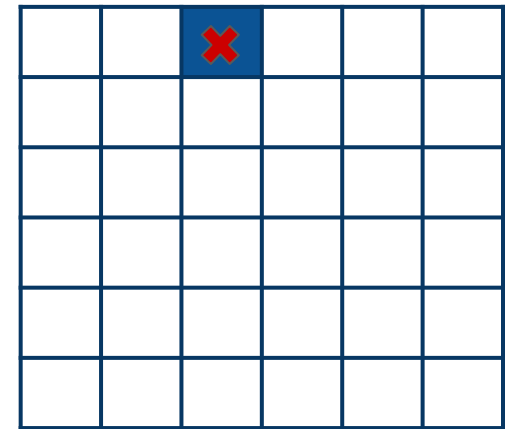
5 fallos



A



B



C

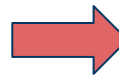
Nota: Para facilitar la comprensión del ejemplo probamos que ocurre con una matriz de 6x6, donde 1 página equivale a 2 columnas de la matriz.

Optimización de la Memoria Virtual

Bucle IJK



Para cada k
 $C[i][j], j=1,3,5 \Rightarrow 5$ fallos
 $C[i][j], j=2,4,6 \Rightarrow 3$ fallos



$$6_k * [(6/2)_{j=1,3,5} * 5 + (6/2)_{j=2,4,6} * 3] =$$

$$= 144 \text{ fallos}$$

×	✓	×	✓	×	✓
×	✓	×	✓	×	✓
×	✓	×	✓	×	✓
×	✓	×	✓	×	✓
×	✓	×	✓	×	✓
×	✓	×	✓	×	✓

A

×	✓	×	✓	×	✓
✓	✓	✓	✓	✓	✓
✓	✓	✓	✓	✓	✓
✓	✓	✓	✓	✓	✓
✓	✓	✓	✓	✓	✓
✓	✓	✓	✓	✓	✓

B

×	✓	×	✓	×	✓
×	✓	×	✓	×	✓
×	✓	×	✓	×	✓
×	✓	×	✓	×	✓
×	✓	×	✓	×	✓
×	✓	×	✓	×	✓

C

Nota: Para facilitar la comprensión del ejemplo probamos que ocurre con una matriz de 6x6, donde 1 página equivale a 2 columnas de la matriz.

Optimización de la Memoria Virtual

Bucle IJK



1 2 3 ... 65 66 ... 130 ...

1								
2								
3								
...								
65								
66								
...								
130								
...								

1 página = 64 columnas
1 matriz = 16 páginas

$$6_k * [(6/2)_{j=1,3,5} * 5 + (6/2)_{j=2,4,6} * 3] = 144 \text{ fallos}$$



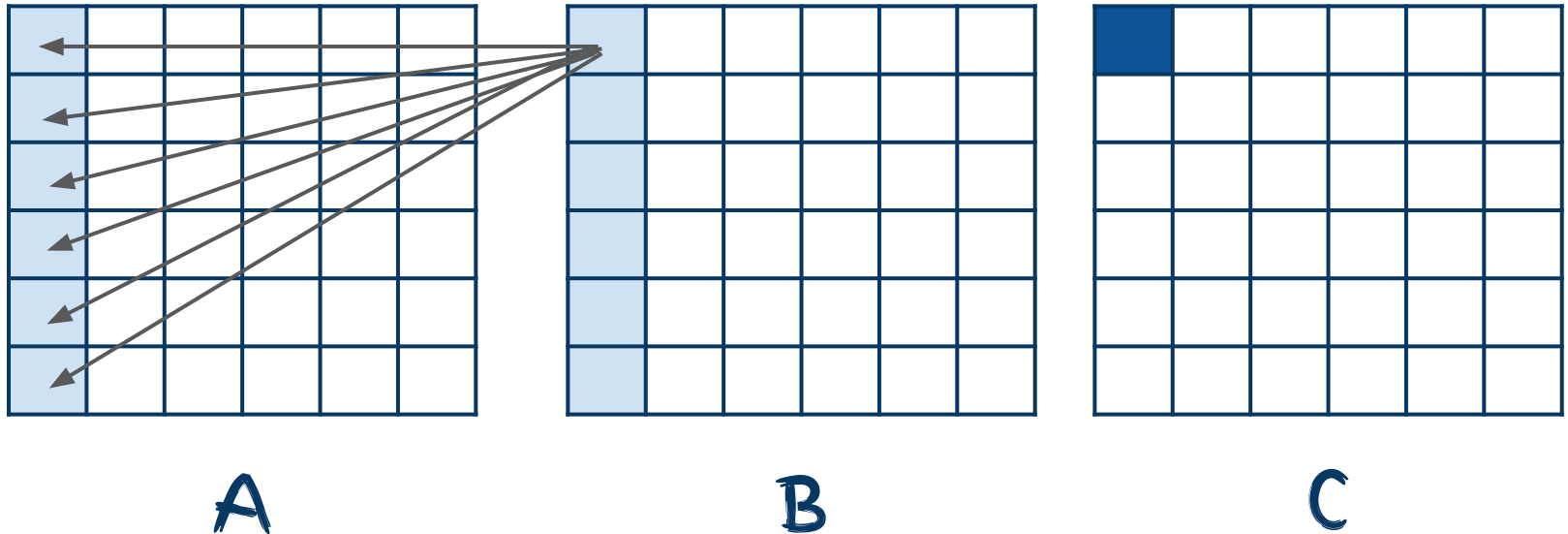
$$1024 * [(1024/64) * (2+16) + (1024/64) * (1024-16)] = 16809984 \text{ fallos} = 16.81 * 10^6$$

Optimización de la Memoria Virtual

Bucle JKI



```
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

Bucle JKI



×					
✓					
✓					
✓					
✓					
✓					

A

×					

B

×					

C

3 fallos (Cargas A, B, C)

Nota: Para facilitar la comprensión del ejemplo probamos que ocurre con una matriz de 6x6, donde 1 página equivale a 2 columnas de la matriz.

Optimización de la Memoria Virtual

Bucle JKI



0 fallos

	✓				
	✓				
	✓				
	✓				
	✓				
	✓				

A

✓					

B

3					
✓					

C

Nota: Para facilitar la comprensión del ejemplo probamos que ocurre con una matriz de 6x6, donde 1 página equivale a 2 columnas de la matriz.

Optimización de la Memoria Virtual

Bucle JKI



		×			
		✓			
		✓			
		✓			
		✓			
		✓			

A

✓					

B

3					
0					
✓					

1 fallo (Carga A)

C

Nota: Para facilitar la comprensión del ejemplo probamos que ocurre con una matriz de 6x6, donde 1 página equivale a 2 columnas de la matriz.

Optimización de la Memoria Virtual

Bucle JKI



0 fallos

			✓		
			✓		
			✓		
			✓		
			✓		
			✓		

A

✓					

B

3					
0					
1					
✓					

C

Nota: Para facilitar la comprensión del ejemplo probamos que ocurre con una matriz de 6x6, donde 1 página equivale a 2 columnas de la matriz.

Optimización de la Memoria Virtual

Bucle JKI



×					
✓					
✓					
✓					
✓					
✓					

A

	✓				

B

3	✓				
0					
1					
0					
1					
0					

C

1 fallo (Carga A)

Nota: Para facilitar la comprensión del ejemplo probamos que ocurre con una matriz de 6x6, donde 1 página equivale a 2 columnas de la matriz.

Optimización de la Memoria Virtual

Bucle JKI



3 páginas/matrizC
 3 fallos cada página de C
 1 fallo cada $(\text{filasC}/(\text{colsA}/\text{pag}) * \text{colsC}/\text{pág} - 1)$

$$3_{\text{págC}} * [3_{\text{ABC}} + 1_A * (3_{\text{filC}/(\text{colsA}/\text{pag})} * 2_{\text{colC}/\text{pg}} - 1)] = 24 \text{ fallos}$$

✗	✓	✗	✓	✗	✓
✓	✓	✓	✓	✓	✓
✓	✓	✓	✓	✓	✓
✓	✓	✓	✓	✓	✓
✓	✓	✓	✓	✓	✓
✓	✓	✓	✓	✓	✓

A

✗	✓	✗	✓	✗	✓
✓	✓	✓	✓	✓	✓
✓	✓	✓	✓	✓	✓
✓	✓	✓	✓	✓	✓
✓	✓	✓	✓	✓	✓
✓	✓	✓	✓	✓	✓

B

3	1	3	1	3	1
0	0	0	0	0	0
1	1	1	1	1	1
0	0	0	0	0	0
1	1	1	1	1	1
0	0	0	0	0	0

C

Nota: Para facilitar la comprensión del ejemplo probamos que ocurre con una matriz de 6x6, donde 1 página equivale a 2 columnas de la matriz.

Optimización de la Memoria Virtual

Bucle JKI



1 2 3 ... 65 66 ... 131 ...

1 página = 64 columnas
1 matriz = 16 páginas

1	3	1	1		3	1	1	3	
2	0	0	0		0	0	0	0	
3	0	0	0		0	0	0	0	
...									
65	1	1	1		1	1	1	1	
66	0	0	0		0	0	0	0	
...									
131	1	1	1		1	1	1	1	
...	0	0	0		0	0	0	0	

$$3_{\text{págC}} * [3_{\text{ABC}} + 1_A * (3_{\text{filC}/(\text{colsA}/\text{pg})} * 2_{\text{colsC}/\text{pg}} - 1)]$$



$$16 * [3 + 1 * (16 * 64 - 1)] = 16416 \text{ fallos} \\ = 16.42 * 10^3 \text{ fallos}$$

Optimización de la Memoria Virtual

Bucle JKI bloqueado



```
DO J=1,N,NB (NB=64)
  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
```

En esta versión se recorre la matriz por bloques de forma que sus posiciones de memoria se correspondan con las páginas de memoria virtual (Una matriz de 1024x1024 se mapea en 16 páginas de (1024x64)).

En el caso de **C** y **A** el recorrido se hace en bloques de forma vertical, por lo tanto se obtienen **16 fallos por cada una**.

Sin embargo, en el caso de **B** se debe hacer un recorrido horizontal de la matriz por ser una multiplicación de matrices, para reducir los fallos que implica este recorrido se usa un recorrido por bloques de (64x64) que va calculando parte del resultado de la multiplicación en ese bloque (para una celda dada de C no se obtendrá un valor correcto de la operación hasta que se complete una iteración del bucle superior que itera los bloques). Por cada iteración de bloque de C (16 bloques) se obtienen 16 fallos: **16x16 fallos**.

Tiempo = (16 + 16x16) x 0,5 = 256 segundos

Optimización de la Memoria Virtual



■ Bloqueo de Matrices:

- El bloqueo se aplica cuando el número de bucles anidados es mayor que el número de dimensiones de las matrices involucradas

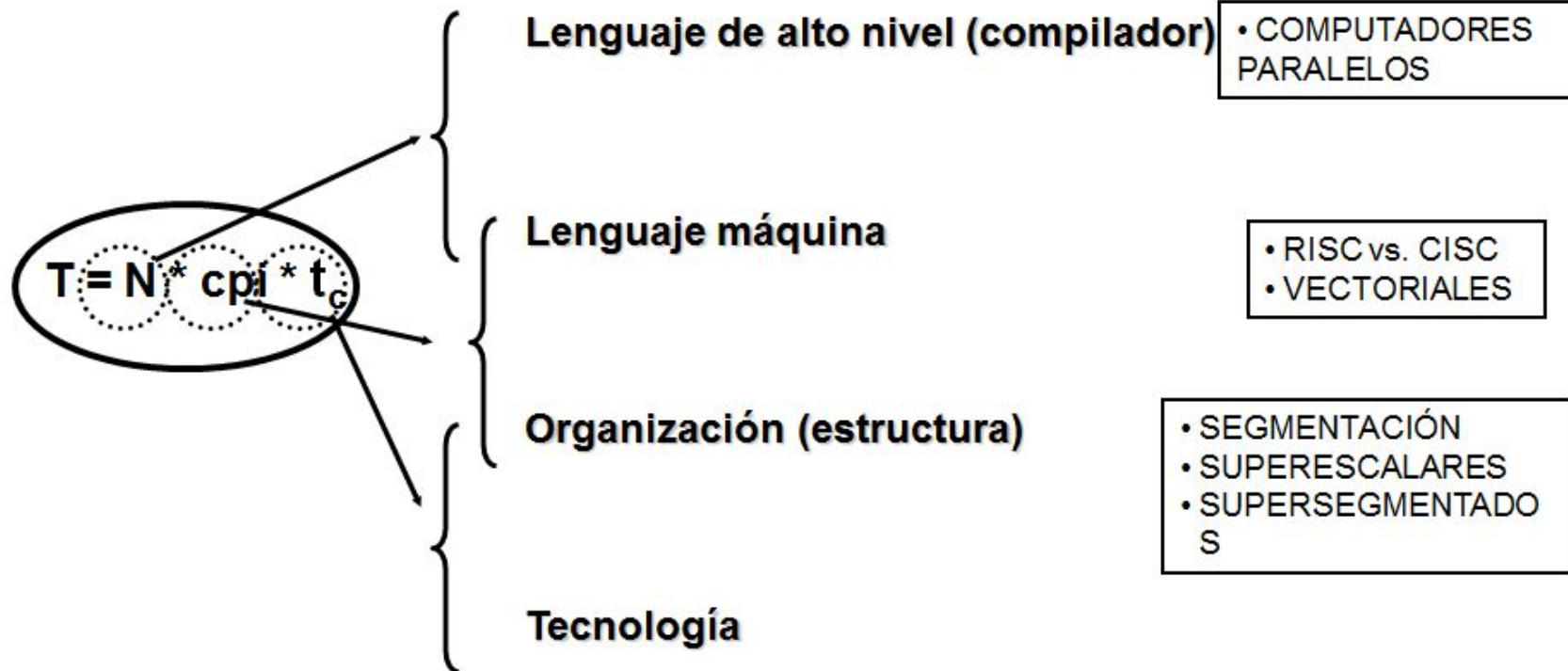
■ Bit Reversal:

- Bit Reversal se aplica cuando el número de bucles anidados es igual que el número de dimensiones de las matrices involucradas

■ Además:

- Uso de funciones `madvise` para indicar al sistema el uso de las páginas de fichero en memoria

Repaso de Procesador Segmentado y Superescalar (1/5)



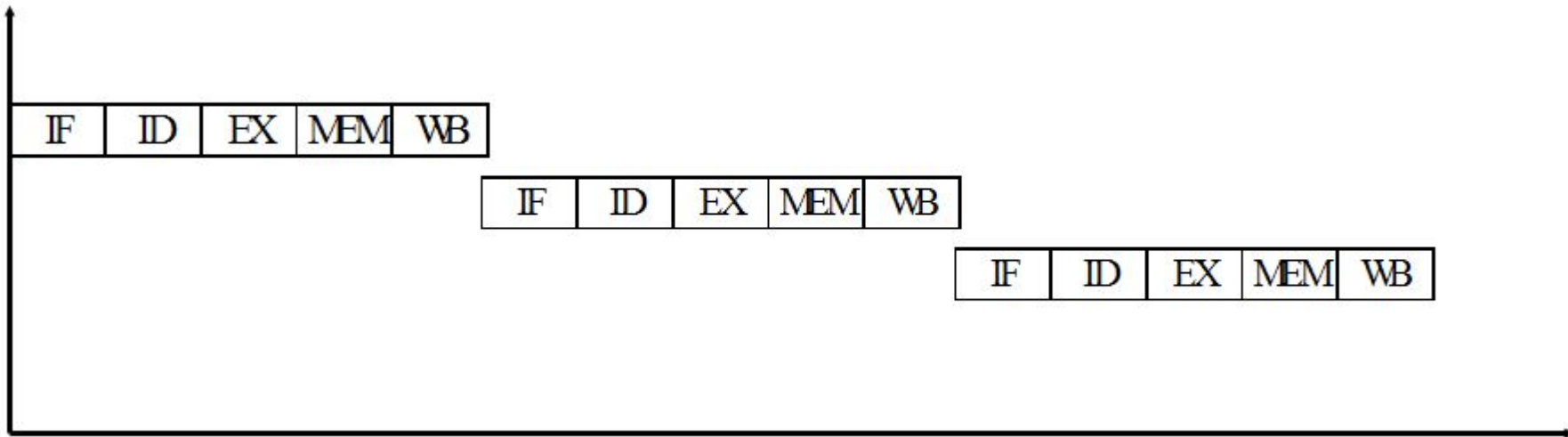
“El compromiso entre estos factores ha guiado el desarrollo de nuevas arquitecturas en los últimos años”

	Nm	CPI	1/t	1/T
i386	1.0	4.5	25	5.5
R/3000	1.2	1.25	25	16.6

Repaso de Procesador Segmentado y Superescalar (2/5)



■ Ejecución secuencial

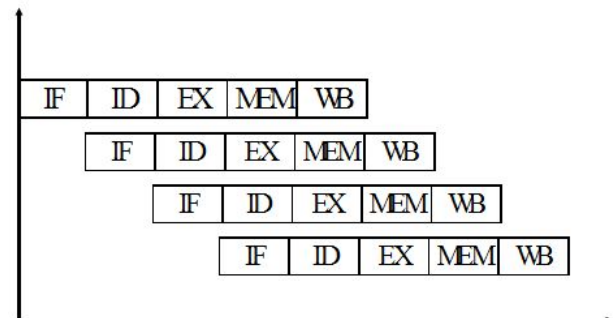


Repaso de Procesador Segmentado y Superescalar (3/5)



■ Ejecución escalar o segmentada

- **Objetivo:** Ejecutar una instrucción por ciclo
- **Problemas con planificación estática:**
 - **Dependencias estructurales:** Replicación de unidades (memoria)
 - **Dependencias de datos LDE:**
 - ✓ Parada del pipe (penalización de 2 ciclos)
 - ✓ Registros de Forwarding (penalización de 1 ciclo solo en caso de instrucción LD)
 - **Dependencias de control:**
 - ✓ Parada del pipe (penalización de 3 ciclos)
 - ✓ Anticipación de la decisión y del cálculo de la dirección (penalización 1 ciclo)
- **Problemas con planificación dinámica:** Emisión en orden, pero lectura de operandos y ejecución en desorden (Scoreboard y Tomasulo)
 - Dependencias de datos EDE: Forwarding
 - Dependencias de datos EDL: Forwarding
 - Predicción dinámica de saltos

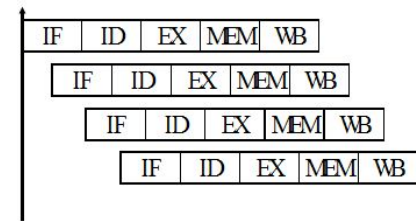
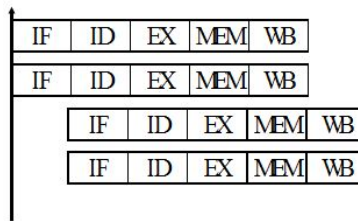


Repaso de Procesador Segmentado y Superescalar (4/5)



■ Ejecución de múltiples instrucciones por ciclo

- **Objetivo:** Ejecutar varias instrucciones por ciclo
 - Aumenta los problemas del caso segmentado
- **Tipos:**
 - **Superescalar:** Emisión de varias instrucciones por ciclo (HP PA 8.000, MIPS 10.000, R/S 6.000, ...)
 - ✓ Se caracterizan por el número y tipo de instrucciones que pueden emitir simultáneamente
 - ✓ Es el procesador en el encargado de realizar la planificación de las instrucciones
 - **Supersegmentado:** Pipes más profundos dividiendo una etapa en varias (DEC 21064)
 - **VLIW:** Una instrucción del procesador contiene varias que se pueden ejecutar simultáneamente (i860)
 - ✓ El compilador es el encargado de realizar la planificación de las instrucciones
 - ✓ Semejante a un superescalar en Sw



Repaso de Procesador Segmentado y Superescalar (5/5)



Unidades funcionales segmentadas

- Una unidad funcional segmentada se caracteriza por:
 - Tiempo de iniciación: Número de ciclos/etapas en realizar una operación
 - Latencia: Número de ciclos entre dos iniciaciones consecutivas



Acceso a memoria segmentado (memoria entrelazada)

- Los accesos a memoria consumen normalmente más de un ciclo
 - El tiempo de espera se suele emplear para ejecutar instrucciones que no impliquen acceso a memoria
- Actualmente, se pueden gestionar varios accesos simultáneamente

Optimización de Procesadores Segmentados y Superescalares (1/7)



Ayuda a las unidades funcionales segmentadas

- Aumentar el grado de paralelismo en el código a nivel de instrucción

```
A = V(1)
A = A*V(2)
A = A*V(3)
A = A*V(4)
A = A*V(5)
A = A*V(6)
A = A*V(7)
A = A*V(8)
```



```
A = V(1)
B = V(2)
A = A*V(3)
B = B*V(4)
A = A*V(5)
B = B*V(6)
A = A*V(7)
B = B*V(8)
A = A*B
```

UNIDAD FUNCIONAL
SEGMENTADA

Optimización de Procesadores Segmentados y Superescalares (2/7)



Disminución del recargo de las instrucciones de control

- Las instrucciones de control de un bucle (salto, incremento de contadores y punteros, ...)
 - Paran el pipe de instrucciones
 - No realizan cálculos útiles
- **Posibles optimizaciones:**
 - Evitar los bucles pequeños
 - Desenrollar bucles
 - Fusionar bucles
 - En bucles anidados, los interiores deben ser los que necesitan un mayor número de iteraciones

Ayuda a los procesadores superescalares

- El uso de variables temporales y paréntesis permite duplicar la velocidad de ejecución

$$X(I) = A(I) + B(I) + C(I) + D(I) + E(I) + F(I) + G(I) + H(I)$$



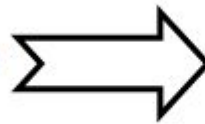
$$X(I) = A(I) + B(I) + C(I) + D(I) + (E(I) + F(I) + G(I) + H(I))$$

Optimización de Procesadores Segmentados y Superescalares (3/7)



Desenrollado de bucles

```
FOR I=1,N  
  A(I) = B(I)*C(I)  
ENDDO
```



```
FOR I=1,N, 2  
  A(I) = B(I)*C(I)  
  A(I+1) = B(I+1)*C(I+1)  
ENDDO
```

■ Ventajas:

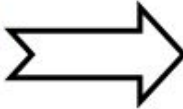
- Se divide por dos el control del bucle
- Disminuye la penalización debida a la ruptura del pipe de instrucciones
- Permite un mayor uso de las unidades aritméticas: uso de varias o uso segmentado de una sola unidad

Optimización de Procesadores Segmentados y Superescalares (4/7)



Fusión de bucles

```
FOR I=1,N
    A(I) = B(I)*C(I)
ENDDO
FOR I=1,N
    D(I) = E(I)+5
ENDDO
```



```
FOR I=1,N
    A(I) = B(I)*C(I)
    D(I) = E(I)+5
ENDDO
```

- **Ventajas:** Afecta positivamente a la arquitectura del procesador
 - Divide por dos el control del bucle, reduciendo la ruptura del pipe de instr.
 - Mayor uso de unidades aritméticas: uso de varias o uso segmentado de una sola unidad
 - Si los bucles comparten variable se realiza un uso óptimo de los registros
- **Inconvenientes:** Afecta negativamente a la jerarquía de memoria si no utilizan los mismos vectores
 - Las instr. del nuevo bucle no entra en el bloque de cache de instrucciones
 - Los arrays implicados pueden llegar a competir por el mismo bloque de cache (más fallos)
 - Al fundir no hay registros suficientes para todas las variables implicadas
 - Puede llegar a ser mejor realizar la operación inversa: **distribución de bucles**


Optimización de Procesadores Segmentados y Superescalares (5/7)



Fusión de bucles

- El compilador fusiona los bucles


```
FOR I=1,N
    A(I) = B(I)*C(I)
ENDDO
FOR I=1,N
    D(I) = E(I)+5
ENDDO
```



```
FOR I=1,N
    A(I) = B(I)*C(I)
    D(I) = E(I)+5
ENDDO
```

- **Solución:** Evitar que tengan el mismo número de iteraciones

```
FOR I=1,N
    A(I) = B(I)*C(I)
ENDDO
FOR I=1,N
    D(I) = E(I)+5
ENDDO
```



```
FOR I=1,N
    A(I) = B(I)*C(I)
ENDDO
FOR I=1,N-1
    D(I) = E(I)+5
ENDDO
D(N) = E(N)+5
```

Optimización de Procesadores Segmentados y Superescalares (6/7)

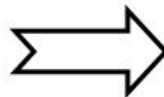


Software pipelining

■ Ejemplo: MIPS R8000,

- Superescalar de 4 vías: 2 LD/ST y/o 2 FP y/o 2 +/-INT y/o 1 shift y/o 1 *//INT
- 75 MHz
- 300 MFLOPS (axpy)

```
FOR I=1,N
    Y(I) = Y(I)+a*X(I)
ENDDO
```



```
LD X
LD Y
MADD Y, X, a
ST Y
X++
Y++
BR
```

Tiempos de ejecución (EX)

LD X	1 ciclo
LD Y	1 ciclo
MADD Y, X, a	4 ciclo
ST Y	1 ciclo
X++	1 ciclo
Y++	1 ciclo
BR	1 ciclo

PLANIFICACIÓN SIMPLE

Ciclo 0:	LD X	LD Y	MADD
Ciclo 1:	X++	Y++	
Ciclo 2:			
Ciclo 3:			
Ciclo 4:			
Ciclo 5:	ST Y	BR	

2/6 op/ciclo => 1/12 del pico

Optimización de Procesadores Segmentados y Superescalares (7/7)



Software pipelining (cont.)

DESENROLLADO X4

```
Ciclo 0: LD X0    LD Y0    MADD0
Ciclo 1: LD X1    LD Y1    MADD1
Ciclo 2: LD X2    LD Y2    MADD2
Ciclo 3: LD X3    LD Y3    MADD3
Ciclo 4: X++      Y++
Ciclo 5: ST Y0
Ciclo 6: ST Y1
Ciclo 7: ST Y2
Ciclo 8: ST Y3    BR
```

8/9 op/ciclo => 2/9 del pico

SOFTWARE PIPELINING

L:

```
Ciclo 0: LD Y0 LD Y1 MADD0 MADD1
Ciclo 1: LD X2 LD X3
Ciclo 2: LD Y2 LD Y3
Ciclo 3: ST Y6 ST Y7
Ciclo 4: LD X4 LD X5 MADD2 MADD3
Ciclo 5: ST Y0 ST Y1 BR D INCR

Ciclo 6: LD Y4 LD Y5 MADD4 MADD5
Ciclo 7: LD X6 LD X7
Ciclo 8: LD Y6 LD Y7
Ciclo 9: ST Y2 ST Y3
Ciclo 10: LD X0 LD X1 MADD6 MADD7
Ciclo 11: ST Y4 ST Y5 BR L INCR
```

D:

16/12 op/ciclo => 1/3 del pico



Optimización de la Entrada/Salida

- La optimización de la entrada/salida es un tema que no se ha abordado lo suficiente
- Cuando un código realiza mucha entrada/salida
 - Reutilizar datos
 - Segmentar entre las diferentes fases de la entrada/salida con la computación
 - Paralelizar la propia entrada/salida por medio de threads accediendo al mismo o diferentes ficheros
 - Utilizar discos (/tmp) o ficheros (mmap) que se encuentren en memoria principal
 - Utilizar formatos binarios y no ASCII para ficheros de datos
 - Utilizar formatos comprimidos (puede ser más rápido comprimir/escribir y leer/descomprimir que leer y escribir descomprimido)
 - Utilizar discos más rápidos
 - Información al sistema sobre uso de ficheros (madvise)

Ejemplos de Herramientas de Ayuda a la Optimización (1/3)



Objetivo: Reducir el tiempo de ejecución de una aplicación

¿Por qué no se ejecuta eficientemente?

¿Dónde se encuentran los puntos críticos?

→ Regla 80/20

CPU

¿Cómo?

optimización

paralelización

Ejemplos de Herramientas de Ayuda a la Optimización (2/3)



¿Por qué el código no se ejecuta eficientemente?



•Procesador
•Entrada/salida
•Memoria



time (Solaris)
timex (Irix)

Tiempos de sistema altos pueden ser debidos a:

- Excepciones de fallo en punto flotante
- Entrada/salida
- Fallos de página en el acceso a memoria virtual
- Llamadas al sistema

Ejemplos de Herramientas de Ayuda a la Optimización (3/3)



Orden time

¿Por qué?

- 135 fallos de página y 0 *swapouts*
- 354 lecturas y 210 escrituras
- 11 Kbytes de memoria compartida + 21Kbytes de memoria privada

```
demos% time a.out  
0.04u 0.06s 0:00.51 19.6% 11+21k 354+210io 135pf+0w
```

- Porcentaje de recursos del sistema usados para el programa
- Tiempo de *pared* (tiempo real de espera)
- Tiempo sistema CPU
- Tiempo usuario CPU

En multiproceso, el tiempo de CPU es la suma de tiempos de todos los procesadores, luego no nos sirve para comparar, se debe usar el tiempo de *pared* (real de espera)

- Para hacer medidas fiables es necesario usar la máquina en modo exclusivo

Ejemplos de Herramientas de Ayuda a la Optimización: Solaris (1/10)



¿Por qué?

	top	iostat	vmstat	truss	sar	perfmeter	mpstat	prtdiag
Process	%cpu time	%u,%s,%i time	%u,%s,%i time		%u,%s,%i time	cpu time	cpu time, by cpu	# cpu
Cache								cache size
Bus								# buses, speed
Memory	Memory used		swap & free		swap & free			memory size
IO	IO wait	blocksize, MB/s	summary		summary	disk		controller config
System Calls	Unix Daemon			#calls, sys arguments	#call, forks	context switch	contex, mutex	
Virtual Memory	%Free Space		paging info		paging info	swap, page		physical mem size

Ejemplos de Herramientas de Ayuda a la Optimización: Solaris (2/10)



Orden gprof

- Esta orden proporciona información postmortem del tiempo de ejecución que el programa ha consumido en cada subrutina:
 - Cuantas veces se invocó una subrutina
 - Desde qué subrutinas se invocó la subrutina
 - Qué subrutinas invocó la subrutina
 - Qué tiempo se consumió en la propia subrutina
 - Qué tiempo se consumió en subrutinas llamadas por esta subrutina
- Se compila el código con -pg (disminuye eficiencia) y se ejecuta gprof usando como parámetro el ejecutable
- La información se almacena en gmon.out (y en la salida estándar), se puede volver a visualizar ejecutando gprof

Ejemplos de Herramientas de Ayuda a la Optimización: Solaris (3/10)



Orden gprof

Fortran Programmer's Guide: 8 - Performance Profiling - Netscape

Archivo Edición Ver Ir Comunicador Ayuda

Anterior Siguiente Recargar Inicio Buscar Guía Imprimir Seguridad Parar

Marcadores Dirección: AMA/NETSCAPE/Users/llorente/mail/Inbox?id=353796D4.97F510A0@eucmax.sim.ucm.es&number=13583123&part=1.2

Instant Message Internet Lookup New&Cool

[4]	93.2	0.92	10.99	1000	diff_ [4]
		1.11	4.52	3000/3000	deriv_ [7]
		1.29	2.91	3000/6000	chebl_ [5]
		1.17	0.00	3000/3000	dissip_ [8]

		1.29	2.91	3000/6000	deriv_ [7]
		1.29	2.91	3000/6000	diff_ [4]
[5]	65.7	2.58	5.81	6000	chebl_ [5]
		5.81	0.00	6000/6000	fftb_ [6]
		0.00	0.00	128/321	cos [21]
		0.00	0.00	128/192	__sin [279]

		5.81	0.00	6000/6000	chebl_ [5]
[6]	45.5	5.81	0.00	6000	fftb_ [6]

Documento: Ejecutado

Ejemplos de Herramientas de Ayuda a la Optimización: Solaris (4/10)



Orden gprof

Fortran Programmer's Guide: 8 - Performance Profiling - Netscape

Archivo Edición Ver Ir Comunicador Ayuda

Anterior Siguiente Recargar Inicio Buscar Guía Imprimir Seguridad Parar

Marcadores Dirección: AMA/NETSCAPE/Users/llorente/mail/Inbox?id=353796D4.97F510A0@eucmax.sim.ucm.es&number=13583123&part=1.2

Instant Message Internet Lookup New&Cool

granularity: each sample hit covers 2 byte(s) for 0.08% of 12.84 seconds

	% cumulative	self	self	self	total	
time	seconds	seconds	calls	ms/call	ms/call	name
45.2	5.81	5.81	6000	0.97	0.97	fftb_ [6]
20.1	8.39	2.58	6000	0.43	1.40	chebl_ [5]
9.1	9.56	1.17	3000	0.39	0.39	dissip_ [8]
8.6	10.67	1.11	3000	0.37	1.88	deriv_ [7]
7.1	11.58	0.92	1000	0.92	11.91	diff_ [4]
4.8	12.20	0.62	2001	0.31	0.31	code_ [9]
2.5	12.53	0.33	69000	0.00	0.00	__exp [10]
0.9	12.64	0.11	1000	0.11	0.11	shock_ [11]
...						

Documento: Ejecutado

Ejemplos de Herramientas de Ayuda a la Optimización: Solaris (5/10)



Funciones `dtime` y `etime`

- Miden tiempo de CPU (usuario y sistema) en un código
 - Son válidas en entornos multiprocesador teniendo en cuenta que el tiempo devuelto por `etime` es el tiempo de reloj

Ejemplos de Herramientas de Ayuda a la Optimización: Solaris (6/10)



Orden tcov

- Esta orden proporciona el número de veces que se ejecuta cada sentencia
 - Compilar con -a el fichero nombre.f y se genera el fichero nombre.d
 - Ejecutar el programa y se actualiza el fichero .d
 - Ejecutar tcov sobre el fichero fuente
 - Se crean ficheros de texto nombre.tcov con la información

Ejemplos de Herramientas de Ayuda a la Optimización: Solaris (7/10)



Orden tcov

The screenshot shows a Netscape browser window titled "Fortran Programmer's Guide: 8 - Performance Profiling - Netscape". The address bar shows a URL from the University of Murcia. The main content area displays Fortran code and its execution output. The code includes a program named "one" that calls a function "two". The output shows the execution of "one" and "two", and the generation of coverage files "one.tcov" and "two.tcov". A table titled "Top 10 Blocks" shows the line counts for the executed code.

```
demo% onetwo
... output from program
demo% tcov one.f two.f
demo% cat one.tcov two.tcov

      program one
1 ->      do i=1,10
10 ->          call two(i)
          end do
1 ->      end
Top 10 Blocks
Line    Count
3         10
2          1
5          1
```

Documento: Ejecutado

Ejemplos de Herramientas de Ayuda a la Optimización: Solaris (8/10)



Funciones `start_iostats` y `end_iostats`

- Estas funciones informan de las transferencias de datos que se produjeron entre las dos llamadas, para cada unidad muestra:
 - El número de sentencias de entrada/salida
 - El número de bytes transferidos
 - Otras estadísticas
- El programa se debe compilar con `-pg` y tras su ejecución se genera un fichero `nombre.io_stats` con la información

Ejemplos de Herramientas de Ayuda a la Optimización: Solaris (9/10)



Funciones start_iostats y end_iostats

Fortran Programmer's Guide: 8 - Performance Profiling - Netscape

Archivo Edición Ver Ir Comunicador Ayuda

Anterior Siguiente Recargar Inicio Buscar Guía Imprimir Seguridad Parar

Marcadores Dirección: AMA/NETSCAPE/Users/llorente/mail/Inbox?id=353796D4.97F510A0@eucmax.sim.ucm.es&number=13583123&part=1.2

Instant Message Internet Lookup New&Cool

4 48 12 0

OUTPUT REPORT

1. unit	5. output data	6. blk size	7. fmt	8. direct
	cnt total avg std dev			(rec len)
0	4 40 10 0	-1	Yes	seq
1	40 40 0			
5	0 0 0 0	-1	Yes	seq
0	0 0 0			
6	26 248 9.538 1.63	-1	Yes	seq
6	248 41.33 3.266			
19	8 48 6 4.276	500548	Yes	seq
4	48 12 0			
20	8 48 6 4.276	503116	No	seq

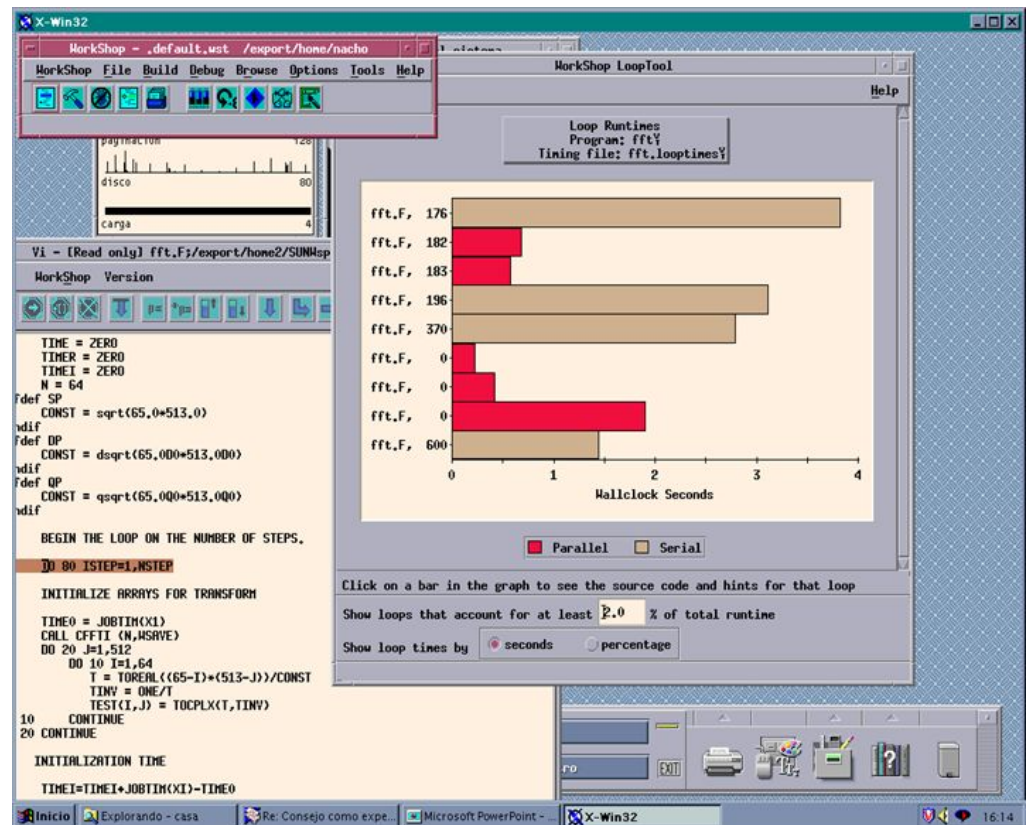
Documento: Ejecutado

Ejemplos de Herramientas de Ayuda a la Optimización: Solaris (10/10)



Workshop

- Compila con -Zlp
- Ejecutar \Rightarrow crea ejecutable.looptimes
- loopreport ejecutable > loopreport.out
- o workshop



Ejemplos de Herramientas de Ayuda a la Optimización: perf (1/7)



Hardware events

- Also referred to as perf_events
- Integrated with Linux kernel (sys_event_perf_open system call)
- Hardware event categories:
 - Cache misses and accesses issued (for L1, L2, and L3 caches) with instruction/data and load/store grouping
 - TLB related (categorized into instruction/data and load/store access types)
 - Branch statistics (branch occurrence, mispredicted branch counts)
 - Cycle (total, stalled, and idle) and instruction (issued, retired) counts
 - Node-level prefetches, loads, and stores
 - “uncore” events, collected by CPU logic shared by all cores (memory controller and NUMA related, last level cache accesses, coherency traffic, power)
- Kernel software events
 - Context switches and migrations, alignment faults, page faults, BPF events

Ejemplos de Herramientas de Ayuda a la Optimización: perf (2/7)



Usage

`perf <command> [<application> <arguments>]`

where supported commands are:

- `list`: outputs events supported on local platform
- `stat`: profiles specified application
- `record`: enables per thread, per process or per CPU profiling
- `report`: performs analysis of recorded data
- `annotate`: correlates profiling data to assembly code
- `top`: displays statistics in real time for a running application
- `bench`: runs tests using predefined benchmark kernels

Ejemplos de Herramientas de Ayuda a la Optimización: perf (3/7)



Example: matrix-vector multiplication

```
#include <stdio.h>
#include <stdlib.h>
#include <blas.h>
#include <time.h>

void init(int n, double **m, double **v, double **p, int trans) {
    *m = calloc(n*n, sizeof(double));
    *v = calloc(n, sizeof(double));
    *p = calloc(n, sizeof(double));
    for (int i = 0; i < n; i++) {
        (*v)[i] = (i & 1)? -1.0: 1.0;
        if (trans) for (int j = 0; j <= i; j++) (*m)[j*n+i] = 1.0;
        else for (int j = 0; j <= i; j++) (*m)[i*n+j] = 1.0;
    }
}

void mult(int size, double *m, double *v, double *p, int trans) {
    int stride = trans? size: 1;
    for (int i = 0; i < size; i++) {
        int mi = trans? i: i*size;
        p[i] = cblas_ddot(size, m+mi, stride, v, 1);
    }
}
```


Ejemplos de Herramientas de Ayuda a la Optimización: perf (4/7)



Example: matrix-vector multiplication

```
double sec(struct timespec *ts) {
    return ts->tv_sec+1e-9*ts->tv_nsec;
}

int main(int argc, char **argv) {
    struct timespec t0, t1, t2, t3, t4;
    clock_gettime(CLOCK_MONOTONIC, &t0);
    int n = 1000, trans = 0;
    if (argc > 1) n = strtol(argv[1], NULL, 10);
    if (argc > 2) trans = (argv[2][0] == 't');

    double *m, *v, *p;
    clock_gettime(CLOCK_MONOTONIC, &t1);
    init(n, &m, &v, &p, trans);
    clock_gettime(CLOCK_MONOTONIC, &t2);
    mult(n, m, v, p, trans);
    clock_gettime(CLOCK_MONOTONIC, &t3);
    double s = cblas_dasum(n, p, 1);
    clock_gettime(CLOCK_MONOTONIC, &t4);
    printf("Size %d; abs. sum: %f (expected: %d)\n", n, s, (n+1)/2);
    printf("Timings:\n  program: %f s\n", sec(&t4)-sec(&t0));
    printf("    init: %f s\n    mult: %f s\n    sum: %f s\n",
           sec(&t2)-sec(&t1), sec(&t3)-sec(&t2), sec(&t4)-sec(&t3));
    return 0;
}
```

Ejemplos de Herramientas de Ayuda a la Optimización: perf (5/7)



Example: matrix-vector multiplication, row-major

```
> perf stat ./mvmult 20000
Size 20000; abs. sum: 10000.000000 (expected: 10000)
```

```
Performance counter stats for './mvmult 20000':
```

1219.404556	task-clock (msec)	#	1.000 CPUs utilized
1	context-switches	#	0.001 K/sec
0	cpu-migrations	#	0.000 K/sec
781,490	page-faults	#	0.641 M/sec
3,898,266,727	cycles	#	3.197 GHz
2,283,166,328	stalled-cycles-frontend	#	58.57% frontend cycles idle
1,372,252,385	stalled-cycles-backend	#	35.20% backend cycles idle
3,764,331,355	instructions	#	0.97 insns per cycle
		#	0.61 stalled cycles per insn
495,220,268	branches	#	406.116 M/sec
815,338	branch-misses	#	0.16% of all branches

```
1.219967824 seconds time elapsed
```

Ejemplos de Herramientas de Ayuda a la Optimización: perf (6/7)



Example: matrix-vector multiplication, column-major

Performance counter stats for './mvmult 20000 t':

12212.530334	task-clock (msec)	#	1.000 CPUs utilized
11	context-switches	#	0.001 K/sec
0	cpu-migrations	#	0.000 K/sec
1,213,417	page-faults	#	0.099 M/sec
42,933,883,759	cycles	#	3.516 GHz
39,567,001,587	stalled-cycles-frontend	#	92.16% frontend cycles idle
37,181,761,140	stalled-cycles-backend	#	86.60% backend cycles idle
6,077,067,370	instructions	#	0.14 insns per cycle
		#	6.51 stalled cycles per insn
918,790,187	branches	#	75.233 M/sec
1,276,503	branch-misses	#	0.14% of all branches
12.213751102 seconds time elapsed			

Ejemplos de Herramientas de Ayuda a la Optimización: perf (7/7)



Example: matrix-vector multiplication, custom events

ROW MAJOR:

```
> perf stat -B -e cache-misses,dTLB-load-misses,iTLB-load-misses ./mvmult 20000
Size 20000; abs. sum: 10000.000000 (expected: 10000)
```

Performance counter stats for './mvmult 20000':

29,307,244	cache-misses
3,121,156	dTLB-load-misses
4,224	iTLB-load-misses

1.227144489 seconds time elapsed

COLUMN MAJOR:

Performance counter stats for './mvmult 20000 t':

79,004,606	cache-misses
405,044,765	dTLB-load-misses
33,124	iTLB-load-misses

12.185000849 seconds time elapsed

Ejemplos de Herramientas de Ayuda a la Optimización



■ **Procesador:**

- Opciones de compilación para optimización secuencial
- Técnicas de opt. secuencial para explotar la arquitectura superescalar
- Técnicas de opt. secuencial para mejorar la explotación de la memoria cache
- Opciones de compilación para paralelización automática
- Inclusión de directivas en el código
- Modificación de código para eliminar dependencias de datos

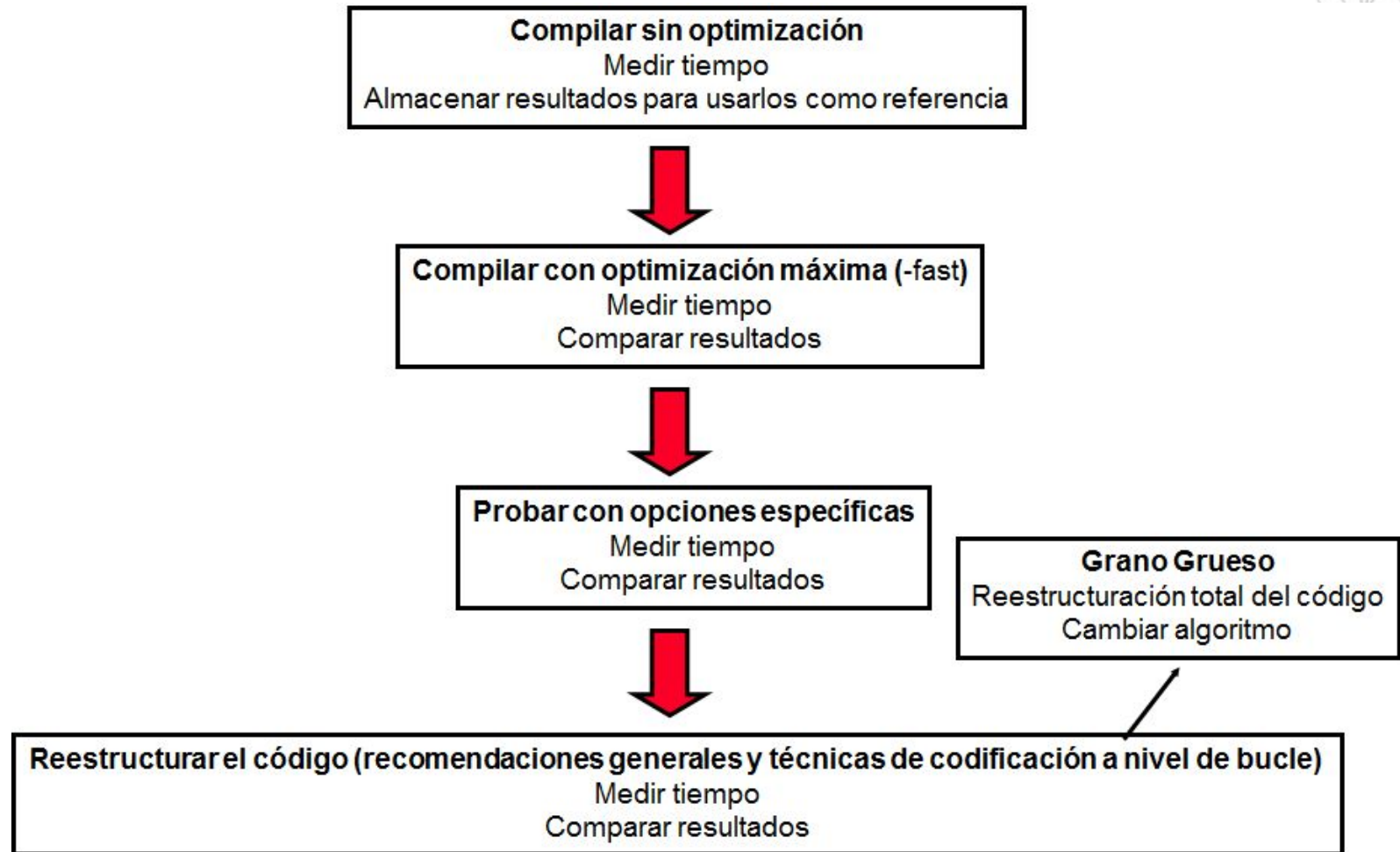
■ **Entrada/salida:**

- Estudio del uso de los ficheros y reorganizar para evitar muchas peticiones cortas (convertir a pocas y largas)
- Uso de funciones mmap para mapear ficheros en memoria
- Uso de funciones madvise para indicar al sistema el uso de las páginas de fichero en memoria

■ **Memoria virtual:**

- Técnicas de optimización secuencial para mejorar la explotación de la memoria virtual
- Uso de funciones madvise para indicar al sistema el uso de las páginas de fichero en memoria

Pasos en la Optimización Secuencial (1/2)

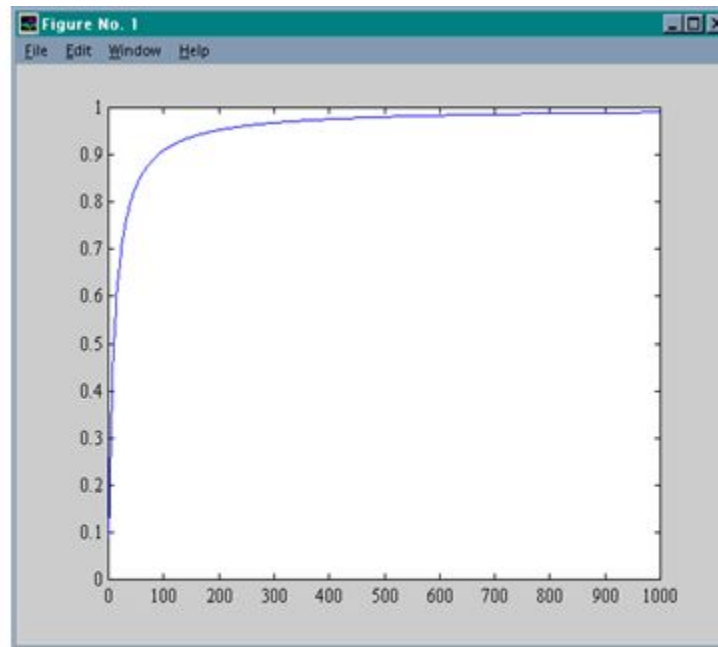


Pasos en la Optimización Secuencial (2/2)



- Siempre centrarse en los bucles más costosos
- Las primeros pasos son los más eficientes

ganancia



esfuerzo