Tema 1. Vectorización

Multiprocesadores

Héctor Lacueva Sacristán

Índice

Objetivos

1. Reducir el tiempo de ejecución de una aplicación.
2. Aumentar la productividad múltiples usuarios.
3. Aumentar la productividad de aplicaciones multihilo.
    1. Servidores Web, bases de datos, ...
4. Tolerancia a fallos: p. ej. sistemas de navegación de un avión.
5. Simplificar componentes y especializar función.
    1. Por ejemplo: sistemas empotrados en-chip (teléfono móvil).

Consecuencias

Hardware

Combinación de:

- Ejecutar **varias instrucciones por ciclo** → **ILP**.
- La **misma instrucción opera sobre varios datos** → **SIMD**.
- Un procesador **ejecuta varios hilos** (threads) → **MT**.
- Muchos **procesadores interconectados** → **MIMD**.

En cuanto al almacenamiento:

- Mucha memoria accesible desde los procesadores con un gran ancho de banda.
- Mucho disco accesible desde la memoria con gran ancho de banda (entrada/salida).

Compilación

Extracción automática de paralelismo a partir de códigos secuenciales.

- Vectorización → SIMD.
- Paralelización → MIMD.

Tenemos el ejemplo de godbolt

Software

Modelos de programación paralela.

- Paralelismo vectorial: FORTRAN 90.
- Paralelismo de datos: High Performance FORTRAN
- Single-program, multiple-data (SPMD).
- Memoria compartida:
    - Pthreads
    - Java
    - ...
- Paso de mensajes
    - MPI
    - ...

1. Reducir el tiempo de ejecución de una aplicación.

Es necesario para mejorar las aplicaciones numéricas (Simuladores de aerodinámica, predictores meteorológicos, ...). Para manejar estructuras de datos como grandes matrices densas o dispersas. Diseñados para trabajar generalmente con números reales, para mejorar rendimiento "pocos" bucles con muchas iteraciones.

Por lo general, el tiempo de ejecución se ve limitado por los accesos a memoria o el cálculo.

Métricas rendimiento en Bucles

Ciclos por iteración

$$\frac{Ciclos}{Iteración}$$

En esta tabla se puede observar las diferencias en rendimiento de los diferentes tipos. Mostrando el TPF para cada uno y entre paréntesis la velocidad R en MFLOPS

Figure 1: En esta tabla se puede observar las diferencias en rendimiento de los diferentes tipos. Mostrando el TPF para cada uno y entre paréntesis la velocidad R en MFLOPS

Número de ciclos que pasan desde que empieza una iteración de un bucle hasta el comienzo de la siguiente. **Teniendo en cuenta todas las posibles dependencias**.

Ciclos por FLOP (CPF)

$$CPF = \frac{Ciclos}{N_{FLOP}}$$

Número de **ciclos por cada operación en punto flotante**. Generalmente se calculan los ciclos de una iteración y se dividen por el número de operaciones en punto flotante.

Tiempo por FLOP (TPF)

$$TPF = \frac{ns}{N_{FLOP}} = CPF * T_C$$

Tiempo por operación en punto flotante.

R (MFLOPS)

$$R(MFLOPS) = \frac{N_{FLOP}}{T_{\mu s}}$$

Rendimiento del procesador, medido en operaciones de punto flotante por segundo.

Sobrecarga de un bucle

Se refiere a los ciclos usados para la lectura y escritura de operandos así como los ciclos usados para el control de bucle y punteros.

Procesadores

Procesador segmentado

Procesador que puede lanzar una instrucción por ciclo y se ejecutan en varios ciclos. Se detiene por dependencias estructurales.

Procesador supersegmentado

Como un segmentado pero con medidas para evitar dependencias estructurales. Se segmentan las unidades funcionales y se crean etapas más pequeñas, permitiendo disminuir el T_C. De esta forma se pueden lanzar hasta el doble de instrucciones en el mismo periodo de tiempo.

Procesador superescalar

Se busca explotar el paralelismo, buscando ejecutar varias instrucciones por ciclo. Dos instrucciones se ejecutan en paralelo si utilizan rutas de datos distintas. Por ejemplo, podría se una instrucción entera y una coma flotante. Necesita de planificación.

Extensión vectorial

Arquitecturas de los supercomputadores

CISC

Permiten instrucciones con operandos en memoria.

Una instrucción como por ejemplo `ADDV @md, @mf1, @mf2`:

Imagen que muestra la historia que ha seguido SIMD (Single Instruction Multiple Data)

Figure 2: Imagen que muestra la historia que ha seguido SIMD (Single Instruction Multiple Data)

- Lee dos flujos desde memoria
- Calcula
- Escribe el flujo resultado hacia memoria.

La latencia de obtención de operandos es muy grande, sobre todo para no secuencial.

RISC

Desacoplar instrucciones de cálculo y de acceso a memoria.

- Instrucciones vectoriales de acceso a memoria para carga/descarga de registros vectoriales. `LV` y `SV`.
- Instrucciones vectoriales de cálculo sobre los registros vectoriales `ADDV`.
- **VLR**: Vector Length Register
  - Número de elementos a procesar, para vectores de 64 elementos necesito 6+1 bits.
- **VMR**: Vector Mask Register
  - Impide que algunas operaciones se realicen.

Ventajas de un repertorio vectorial

Compacto

Una instrucción pequeña codifica N operaciones.

Expresivo

La instrucción indica al Hardware:

- Operaciones independientes, eficiente energéticamente.
- El número de operaciones.
- El "patrón" de adcceso a memoria:
  - En secuencia (stride[1] = 1)
  - A saltos (stride > 1), tiene peor rendimiento en procesadores actuales.

Escalable

El mismo binario puede ejecutarse en una o varias "pistas" segmentadas (parallel pipelines or lanes).

Bajo consumo de energía

- Ahorramos energía en el acceso a la memoria de instrucciones y en la decodificación.
- Cuando se ejecutan las operaciones de una instrucción vectorial no hay que gastar energías en comprobar dependencias.

Historia del calculo vectorial

Procesadores Matriciales (Array Processors)

- Historicamente fueron los primeros.
- Contaban con varios Nodos(mem + regs + ALU) y una red de interconexión directa entre nodos.

Procesadores vectoriales segmentados (pipelined)

- Pueden ser mem-to-mem o mem-to-reg.
- Soportan vectores de tamaño grande, hasta 4k elementos.
- Necesitan soporte HW:
  - Banco de Registros Vectorial (BRV) + Memoria Multibanco + ALUs + segmentación.

[1]stride: diferencia de posición entre elementos consecutivos, se mide en elementos o en bytes (dependiendo de procesador).

Arquitectura básica de un procesador con soporte para extensión vectorial.

Figure 3: Arquitectura básica de un procesador con soporte para extensión vectorial.

Extensiones multimedia

- Se trabaja con vectores cortos donde no existe un registro VLR y por tanto, la longitud vectorial va en el código de operación.
- Primero fue Intel MMX (1982) y luego ya Intel SSE.

Extensiones vectoriales

- Necesitan de soporte HW:
  - ALUs replicadas y BRV.
  - Por ejemplo, para AVX-512 el BRV tiene 32 registros de 512 bits
- Destacan AVX, AVX-512, ARM SVE, RVV.

Arquitectura procesador vectorial

Banco de registros vectorial (BRV)

Se pueden construir con diferentes especificaciones dependiendo del número de buses de lectura y escritura y los puertos de los registros vectoriales.

BRV con 2 buses L, 1 bus E y RV de 1 puerto

Se permite concurrentemente o bien una lectura o bien una escritura ($1L \oplus 1E$).

BRV con P buses L, Q buses E y RV de 1 puerto

Solo puede haber un flujo por registro:

- O bien lectura desde un bus L (L1..LP).
- O bien escritura desde un bus E (E1..EQ).

Globalmente máximo de 1 flujo por RV.

2 puertos por *RV*

Permite lectura o escritura, puedes leer y escribir de manera "simultánea" ($1L + 1E$). De tal forma que:

- `addv v1, v5, v1` Se puede ejecutar.
- Se pueden solapar dependencias:

  ```
  addv v1, v2, v3
  subv v4, v5, v1
  ```

  - Se conoce como **encadenamiento general**.
- Se pueden solapar antidependencias:

  ```
  addv v1, v2, v3
  subv v3, v4, v5
  ```

  - Tiene un control similar

Unidades funcionales (UFs)

Latencia (L)

Tiempo desde que entran los operandos en la UF hasta que sale el resultado.

Latencia de iniciación (LI) o latencia de finalización

Número mínimo de ciclos entre entradas consecutivas de operandos.

Sumadores, Multiplicadores

Generalmente son UFs totalmente segmentadas en n etapas. Su L = n y su LI es de 1 ciclo/op.

División, Raices, Exponenciales, Trigonométricas

Generalmente son UFs parcialmente segmentadas y algunas de esas etapas se reutilizan. Su L es mayor al número de etapas y la LI es mayor que 1.

Memoria multibanco

Queremos conseguir que se puedan leer de memoria datos con este comportamiento.

$$4ciclos \rightarrow 1ciclo \rightarrow 1ciclo \cdots$$

Para ello necesitamos:

Multibanco

$$N_{BANCOS} = L_{MEM} \wedge (tam_{BANCO} == \frac{tam_{MEM}}{N_{BANCOS}})$$

Entrelazado "por palabras"

Es una estrategia de reparto de memoria que consiste en colocar palabras consecutivas en bancos consecutivos.

Método de acceso

Acceso síncrono Si el primer elemento cae en el primer bloque de memoria, funciona sin pérdidas. Pero si no cae en el primer bloque de memoria se puede perder hasta un total de N-1 ciclos, siendo N el número de bancos.

- **Ventajas**: simplicidad
- **Inconvenientes**:
  - lectura: latencia variable o irregular
  - escritura:
    - Un registro MDRin para cada banco
    - Es lenta: (1 1 1 1 4) (1 1 1 1 4)...

Acceso desfasado En este método los bancos son más independientes.

- Se añade un registro MAR para cada banco.
- El control es más complejo.
- **Ventajas**:
  - lectura: (4+1 1 1 1 1 1 1 1 1 1 1 ...)
  - escritura: (... 1 1 1 1 1 1 1 1 1 1 4+1)

Síncrono vs. Desfasado

Acceso síncrono vs. Acceso desfasado
Diferencia en la ejecución de una misma lectura usando Acceso síncrono frente a Acceso desfasado

Algunos ejemplos de procesadores

Procesador 1

- BRV con 2L y 1E (y RV mínimos (1 puerto))
  - 2 buses L: $BRV \rightarrow \{M \oplus ALU\}$
  - 1 bus E: $\{M \oplus ALU\} \rightarrow BRV$
- Memoria de 1 puerto $\{L \oplus E\}$. Y 1 UF.

Segmentación

B	D	Reg	Latencia	Producción
	Riesgos: escalar o vector	Lec BRV Lec VLR	UFs/MEM lectura	escr v3.0 | escr v3.1 | ...

Procesador 2

- BRV con 3L y 2E (y RV mínimos (1 puerto))
  - 3 buses L: $BRV \rightarrow \{M, ALU\}$
  - 2 bus E: $\{M, ALU\} \rightarrow BRV$
- Memoria de 1 puerto $\{L \oplus E\}$. Y 1 UF.

Instrucciones independientes y sin riesgos estructurales Se ejecutan secuencialmente y sin mayores problemas.

Instrucciones dependientes o con riesgos estructurales Se debe de esperar a que estén disponibles tanto los datos como las UFs.

Procesador 3

Este procesador cuenta con 2 UFs:

- UF1: es una ALU para sumar
- UF2: es una ALU para sumar/multiplicar

Cuenta a su vez con 3 buses de escritura y 5 de lectura.

Procesador 4

- Tiene dos ALUs dedicadas (una para sumas y otra para multiplicaciones)
- Dos puertos lectura y uno de escritura por cada RV. La unidad de control permite dos flujos de lectura y uno de escritura en cada registro vectorial.
  - Esto permite solapar dependencias de datos por **Encadenamiento General**.

En este caso se permite realizar la **primera lectura de registros bajo la primera producción**.

Procesador 5

En este procesador se replican los caminos de proceso y memoria:

- El BRV y la memoria se particionan: pares (v1.0, v1.2, ...) en una partición e impares (v1.1, v1.3, ...) en la otra.
- Cada una de las dos ALUs se duplican.
- En el primer ciclo de ejecución entran en paralelo a las dos pistas los elementos 0 y 1 de los RV fuente. En el segundo el 2 y 3, y así sucesivamente.
- El número de ciclos por elemento (C_e), se divide por el número de pistas.

Estructura del procesador 5

Estructura del procesador 5

Rendimiento procesador vectorial

Rendimiento sin seccionar

Rendimiento $n \leq MVL$

El rendimiento depende de n, el tamaño del vector. Matemáticamente se puede calcular de la siguiente manera:

$$R_n = \frac{n \times k}{T_n} = \frac{n \times k \times F}{C_n} \text{ donde k es el número de FLOPs}$$

Supongamos $n \leq MVL$, donde MVL es el tamaño máximo del vector (Maximum Vector Length, número de elementos de un RV). Los costes de ejecutar n elementos es el siguiente:

Costes ejecución

$$C_{n(n \leq MVL)} = C_{fijos} + n \times C_e \text{ ciclos}$$

Donde los ciclos fijos son:

$$C_{fijos} = \text{latencia UFs + penalizaciones(riesgos estructurales, ...)}$$

Y C_e:

$$C_e = \text{ciclos por elemento}$$

El tiempo será el siguiente:

$$T_{n(n \leq MVL)} = C_n \times T_C$$

Rendimiento pico R_∞

El rendimiento si tuviesemos un vector infinito y MVL fuese infinito. El rendimiento sería

$$R_{\infty(MVL\infty)} = \frac{k \times f}{C_e} \text{ donde k es el número de FLOPs}$$

n para MITAD de rendimiento máximo $N_{\frac{1}{2}}$

$$N_{\frac{1}{2}} = \frac{C_{fijos}}{C_e} = \frac{T_{fijos}}{T_e}$$

- Da una idea sobre la sobrecarga vectorial (latencias y penalizaciones).
  - Si $N_{\frac{1}{2}}$ es muy grande, tiene menos sobrecarga y viceversa.
- Es independiente del tiempo de ciclo, solo depende de la arquitectura y del algoritmo.

N_V

Longitud mínima de vector que consigue igual velocidad en las versiones escalar y vectorial. Puede servir para decidir la compilación del código. También mide la sobrecarga vectorial y la velocidad relativa entre los procesadores vectorial y escalar. Se calcula de la siugiente manera:

$$R_{N_V} = R_{esc} \rightarrow N_V = \frac{C_{fijos}}{C_{esc} - C_e} = \frac{T_{fijos}}{T_{esc} - T_e}$$