AC 21/22 - Tema 1: Arquitecturas paralelas, clasificación y prestaciones

1. Introducción

Tanto el diseño de nuevos procesadores como el desarrollo de programas eficientes requieren de evaluar las prestaciones de un sistema. En este capítulo vamos a ver formas de obtener esas métricas, que ponen de manifiesto las relaciones existentes entre el tiempo de la CPU y parámetros como el número de instrucciones, capacidad de la microarquitectura o número de instrucciones por ciclo. También estudiaremos los cuellos de botella y el uso de la Ley de Ahmdal para razonar acerca de éstos.

2. Tiempo de CPU

Tiempo de respuesta y de CPU

Tiempo de respuesta: tiempo transcurrido entre el inicio de la ejecución de un programa y la obtención de los resultados

- Tiempo de CPU: contenido en el tiempo de respuesta, es el tiempo que el procesador pasa ejecutando instrucciones del programa.

  El resto del tiempo de respuesta, no incluido en el Tcpu es, por ejemplo, tiempo de E/S o tiempo que el procesador dedica a otros programas distintos durante la ejecución.

  Es decir: el tiempo de respuesta puede ser un indicador de las prestaciones de un ordenador, pero puede verse afectado por operaciones de E/S o por la carga del procesador en el momento de la ejecución. Nos centraremos en el Tcpu, considerando siempre Te/s despreciable.

> Tcpu = Ciclos del programa · Tciclo = Ciclos del programa / frecuencia
>
> > Es decir, Tciclo es el periodo y frecuencia es frecuencia y; como sabemos, f = 1/T

- Ciclos de programa: número de ciclos de reloj del procesador que tarda el programa en ejecutarse.

  > Ciclos de programa = Num instr · Ciclos por instrucción

- Frecuencia: frecuencia del reloj

  > f = 1/Tciclo

Por tanto, otra expresión para el Tcpu sería:

> **Tcpu = CPI · NI · Tciclo = (NI · CPI) / f**

Especificación

En una máquina en la que todas las instrucciones requieren el mismo número de ciclos, su CPI medio puede calcularse de la siguiente manera:

> CPI = (sumatorio from i:0 to W:num instr distintas [NI · CPI]) / NI

Normalmente las instrucciones del programa se codifican en una sóla instrucción máquina. Pero para repertorios que aprovechan el paralelismo de datos esto no es necesariamente cierto, así que podemos calcular su NI de la siguiente manera:

> NI = Num operaciones del programa / Num operaciones que puede codificar cada instr
>
> NI = Noper / OPI

El CPI suele depender de las características de la arquitectura y su disposición física. Pero hay ocasiones (procesador segmentado NO superescalar) en las que el compilador es decisivo para el aprovechamiento de los recursos, en cuyo caso nos interesaría expresar el CPI a partir de las características de la microarquitectura.

Podemos usar el num medio de ciclos que tienen que transcurrir desde que se emiten una o varias instr, hasta que se puede realizar otra emisión (llamado CPE); y el número medio de instr que se emiten (IPE), de forma que:

> CPI = num medio ciclos entre emisión de una instr hasta sig emisión / num medio instr emitidas
>
> CPI = CPE / IPE

- En un procesador no segmentado (las instr se ejecutan de una en una), CPE es igual al número medio de ciclos de relojque tarda en procesarse la instr; y IPE es 1 porque las instr van de una en una.
- En un procesador segmentado, el valor máximo de CPE es 1 porque cada ciclo podrían empezar a emitirse instrucciones. Si el procesador segmentado sólo puede empezar a ejecutar una instrucción por ciclo, se tiene:

  > CPI = 1/IPE = 1/1 = 1

  Lo cual sería un caso ideal, dado que en los procesadores segmentados normalmente no es posible emitir una instrucción por ciclo (por problemas con dependencias, por ejemplo). Así que, normalmente, CPE > 1 y por tanto CPI > 1. Cuanto más cercano a 1 sea el CPI, mejor estamos aprovechando el cauce del procesador.

  En el caso de procesadores superescalares o VLIW, donde se pueden emitir varias instr por ciclo, el CPI puede ser menor que 1.

  Conclusión final tras estas aclaraciones:

  > **Tcpu = NI · CPI · Tciclo = (Noper/OPI) · (CPE/IPE) · Tciclo**

3. Medidas de velocidad de procesamiento y benchmarks

Referidas a la velocidad de ejecución de instrucciones

MIPS: Millones de instrucciones por segundo

> MIPS = NI / (Tcpu * 10^6) = NI / (NI * CPI * Tciclo * 10^6) = 1 / (CPI * Tciclo * 10^6) = F / CPI * 10^6

Con la tecnología actual, no es raro sustituir el 10^6 por 10^9 para obtener GIPS, y así no tener que manejar MIPS tan grandes.

Cuando trabajamos con MIPS hay que tener en cuenta que es un cálculo específico al repertorio: una máquina de repertorio RISC necesita muchas más instr para ejecutar el mismo programa que una CISC así que, incluso si el Tcpu de las dos máquinas fuera el mismo para un mismo programa (ambas máquinas son igualmente eficientes), el MIPS de la máquina RISC sería mayor y caeríamos en error si dijéramos que por ello es mejor. Los MIPS miden la velocidad con la que cada procesador ejecuta las instrucciones de su repertorio - por ello muchas veces se refiere a ellos como la velocidad pico del procesador.

MFLOPS: Millones de operaciones float por segundo

> MFLOPS = operaciones float / Tcpu * 10^6

De nuevo, es común sustituir el 10^6 por 10^9 y obtener GFLOPS; por 10^{12}, TFLOPS; por 10^{15}, PFLOPS; por 10^{18}, EFLOPS...

Benchmarks

Tanto cuando evaluamos MIPS como FLOPS, si nos referimos a los valores pico de esas magnitudes hay que emplear un programa o cjto de programas sobre los que hacer medidas de tiempo y NI o NIfloat.

Para tener un marco común de programas de pruebas para evaluar prestaciones, tenemos benchmarks. Estos varían según el fin de la evaluación o el tipo de recurso que se va a evaluar. Algunos ejemplos conocidos son SPEC, TPC, Linpacks.

4. Estimación de tiempo de CPU

Hay veces que es necesario estimar el Tcpu que puede necesitar un programa dadas las características del computador en el que se ejecuta, el cjto de operaciones, el patrón de acceso a memoria del programa...

Las expresiones dadas hasta ahora asumen que los datos a los que se accede están en el nivel 1 de la caché - por eso el CPI suele ser pequeño. Pero si los datos están en memoria o más profundos en la caché el tiempo puede aumentar considerablemente. Podemos obtener una expresión del Tcpu mínimo con la expresión del Tcpu: ese es el tiempo mínimo que va a necesitar el programa, sin fallos de caché y con el procesador a pleno rendimiento.

Pero si tenemos información de las características de la jerarquía de memoria de la máquina, podemos tener una mejor estimación del tiempo mínimo de CPU, porque puede existir un solapamiento casi total entre el tiempo de ejecución de instrucciones en el procesador, y el tiempo de acceso a memoria principal (para load o store, para los que se producen los fallos de caché): es decir, el tiempo mínimo será el mayor de estos dos tiempos:

> Tejec_min = max(Tcpu, Tmemoria)
>
> > Tmemoria = Num accesos a memoria · tiempo medio de acceso a memoria

> > Tmemoria = Nacc · tmem

El tiempo de acceso a memoria en una caché look-through (mira toda la caché y si no está ahí lo que quiere, va a memoria) se calcula:

> tmem = a1 · t1 + (1 - a1) · (t1 + tm)

- a1 - tasa de aciertos de la caché, la probabilidad de que un acceso a memoria esté en caché
  - Para estimar la tasa de aciertos se determina el número de accesos a memoria y en cuántos de ellos falla
- t1 - tiempo de acceso a caché L1
- tm - tiempo de acceso a memoria
  - Determinado por la arquitectura

Si además del acceso tenemos en cuenta el tiempo necesario para reemplazar una línea de caché con el contenido que traemos desde memoria, la fórmula queda:

> tmem = a1 · t1 + (1 - a1) · (t1 + tm + preemplazo · tlinea)

- preemplazo - probabilidad de que, cuando se produzca una falta, haya que cambiar una línea de la caché actualizando en memoria la línea reemplazada
  - A partir del patrón de accesos de memoria y las características de la caché se puede determinar también la probabilidad de reemplazo
- tlinea - tiempo necesario para actualizar la línea
  - Determinado por la arquitectura

Para mejorar las prestaciones de la jerarquía de memoria se puede:

- Reducir los tiempos de acceso a los distintos niveles de caché
- Reducir tasa de fallos a niveles de caché
- Reducir la penalización cuando se producen fallos
  - Técnicas basadas en inclusión de recursos (cachés de víctimas o pseudo-asociativas)
  - Procedimientos a nivel de programación como precaptación u optimización de código (mezcla de arrays, fusión de bucles, operaciones con submatrices (blocking))

5. Ganancia de velocidad y ley de Amdahl

Para medir el resultado de una mejora se utiliza la ganancia de velocidad, que compara la velocidad de una máquina antes y después de los cambios. La expresión a ganancia de velocidad es:

> Sp = Vp / V1 = (W/Tp) / (W/T1) = T1 / Tp

- V1 - velocidad previa a la mejora
- Vp - velocidad posterior a la mejora
- W - carga de trabajo que se aplica, es la misma post y pre mejora
- T1 - Tcpu previo a la mejora
- Tp - Tcpu previo a la mejora

La ley de Amdahl establece una cota superior a la ganancia que se puede conseguir mejorando alguno de los recursos en un factor p y según la frecuencia con la que se utiliza ese recurso:

> LEY DE AMDAHL
>
> Sp <= 1 / (1 + f · (p-1))

- Sp - ganancia
- f - fracción del tiempo de ejecución ANTES DE APLICAR LA MEJORA en un recurso QUE NO EMPLEA LA MEJORA
    - O sea, si hay una mejora en el 25% del código, f=0.75
    - Así, si f=1 es que no se aprovecha la mejora en ningún punto del código y Sp será menor o igual a 1, así que no hay mejora de velocidad
    - Si f=0 es que la mejora se aprovecha en todo el código y Sp puede alcanzar un valor igual al factor p