AAAC & TAC
Clase #5
Rendimiento

Facultad de Ingeniería
Universidad de la República

Instituto de Computación
Curso 2014

Rendimiento (Performance)

- Encontrar definiciones consistentes, más allá del marketing
- Tomar opciones inteligentes (al menos, justificadas cuantitativamente)
- Entender el problema/motivación
  - ¿Por qué algunos programas se comportan mejor en determinado hardware y no en otro?
  - ¿Qué factores del rendimiento de un sistema dependen del hardware?
    - (ejemplo: se necesita una nueva máquina, o un nuevo sistema operativo?)
  - ¿Cómo afecta el set de instrucciones al rendimiento? ¿Y la organización?

¿Cuál es mejor?

Plane	DC to Paris	Speed	Passengers	Throughput (pmph)
Boeing 747	6.5 hours	610 mph	470	286,700
Concorde	3 hours	1350 mph	132	178,200

- Tiempo de ejecución de una tarea
  - Tiempo de Ejecución, tiempo de respuesta, latencia
- Tareas por día, hora, semana, seg, ns …
  - Throughput

Medida de rendimiento: TIEMPO

- Tiempo de ejecución o respuesta (latencia)
  - ¿Cuánto tiempo lleva ejecutar un trabajo?
  - ¿Cuánto hay que esperar por el resultado de una consulta a la base de datos?
- Throughput
  - ¿Cuántos trabajos pueden ejecutarse a la vez?
  - ¿Cuál es el tiempo de ejecución total de varias tareas?
  - ¿Cuántos trabajos terminan por unidad de tiempo?
- Preguntas típicas
  - ¿Cuánto se mejora si agregamos un procesador al servidor?
  - ¿Qué mejora se obtiene agregando una nueva máquina?

Otra medida de rendimiento: MIPS

- MIPS (millones de instrucciones por segundo)
- MIPS=Recuento de Instrucciones/(Tiempo_Ejec x 10^6)

- Ejemplo
  - Dos diferentes compiladores se prueban en una máquina de 100 MHz con tres clases de instrucciones A, B, y C, que requieren uno, dos y tres ciclos, respectivamente. Ambos compiladores producen código para un programa que al correrse provoca la ejecución de:
    - Compilador 1: 5 millones de instrucciones A, 1 millón de B y 1 millón de C
    - Compilador 2: 10 millones de instrucciones A, 1 millón de B y 1 millón de C

  - Cual ejecuta más rápido?
  - Cual es “mejor” de acuerdo a los MIPS?

Tiempo de Ejecución

- Tiempo transcurrido
  - Mide “todo”
    - Accesos a memoria y disco, E/S, etc
  - Una medida útil, pero difícil de usar para comparar sistemas

- Tiempo de CPU
  - No cuenta E/S, o el tiempo que se emplea en ejecutar otros programas (en sistemas multitarea)
  - Se puede dividir en tiempo del sistema y tiempo de usuario

Definición de Performance

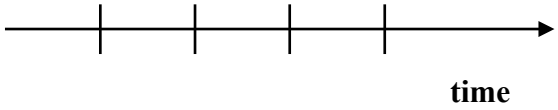
- Performance: unidades de ejecución por unidad de tiempo
  - Más grande es mejor

- Performance(x) = 1/ Execution_time(x)
- "X es n veces más rápido que Y" significa

$$n = \frac{ExTime(Y)}{ExTime(X)} = \frac{Performance(X)}{Performance(Y)}$$

Ciclos de Reloj

- Tiempo discreto en "ticks" de reloj:



- Tiempo de ciclo = tiempo entre ticks = período (seg/ciclo)

- Frecuencia del reloj (ciclos / seg)
  - (1 Hz = 1 ciclo/seg)
  - Un reloj de 2 Ghz tiene un período de:

$$\frac{1}{2 \times 10^9 \ \text{ciclo}/\text{seg}} = \frac{1}{2} \times 10^{-9} \ \text{seg/ciclo} = 0.5 \ \text{nanosegs/ciclo}$$

- Período constante => #ciclos de reloj es una medida alternativa para el tiempo de ejecución

$$\text{Tiempo de CPU} = \#ciclos \times período$$

$$\text{Tiempo de CPU} = \frac{\#ciclos}{frecuencia}$$

Ecuación de performance de CPU

$$\text{Tiempo de CPU} = \#ciclos \times período$$

$$\text{Tiempo de CPU} = \#instrucciones \times \frac{\#ciclos}{\#instrucciones} \times período$$

$$\text{Tiempo CPU} = \frac{\text{Segundos}}{\text{Programa}} = \frac{\text{Instrucciones}}{\text{Programa}} \times \frac{\text{Ciclos}}{\text{Instrucciones}} \times \frac{\text{Segundos}}{\text{Ciclo}}$$

- Identificamos tres factores que afectan la performance
  - **CPI**: # ciclos / # instrucciones
  - Recuento de instrucciones: # de instrucciones del programa
  - Frecuencia (o período): # de ciclos por segundo

- Un error habitual es considerar que solamente uno o dos de estos factores son determinantes de la performance
  - Caso típico: frecuencia de reloj
  - MIPS = Frecuencia del Reloj / (CPI x 10^6)

Aspectos de la Performance de la CPU

	Recuento Inst	CPI	Ciclo de Reloj
Programa	X	(X)	
Compilador	X	(X)	
Set de Inst.	X	X	
Organización		X	X
Tecnología			X

- Tener en cuenta esta matriz para mejorar la performance

Ciclos por Instrucción

En un set de *n* (categorías de) instrucciones

$$\#ciclos = \sum_{j=1}^{n} \#ciclos_j = \sum_{j=1}^{n} \frac{\#ciclos_j}{\#ocurrencias_j} \times \#ocurrencias_j$$

$$\mathrm{CPI} = \frac{\#ciclos}{\#instrucciones} = \sum_{j=1}^{n} \frac{\#ciclos_j}{\#ocurrencias_j} \times \frac{\#ocurrencias_j}{\#instrucciones}$$

"Frecuencia de Instrucciones"

$$\mathrm{CPI} = \sum_{j=1}^{n} \mathrm{CPI}_j \times \mathrm{F}_j, \quad \text{siendo } \mathrm{CPI}_j = \frac{\#ciclos_j}{\#ocurrencias_j}, \mathrm{F}_j = \frac{\#ocurrencias_j}{\#instrucciones}$$

CPI_j debe medirse, ya que la referencia técnica (los manuales) no tiene en cuenta, por ejemplo, misses de cache o retardos en pipeline.

Ejemplo #1: Cálculo de CPI

Op	Freq	CPI	F_j*CPI_j	(% Tiempo)
ALU	50%	1	.5	(33%)
Load	20%	2	.4	(27%)
Store	10%	2	.2	(13%)
Branch	20%	2	.4	(27%)
			1.5	

Mezcla Típica

Ejemplo #2: Cálculo de CPI

Op	Freq	CPI	F_j*CPI_j	(% Tiempo)
Multiply	30%	5	1.5	(56%)
Other ALU	20%	1	.2	(7%)
Load	20%	2	.4	(15%)
Store	10%	2	.2	(7%)
Branch	20%	2	.4	(15%)
			2.7	

Mezcla Típica

- Donde se invierte el tiempo?

Ejemplo #3: Impacto del "Branch Stall"

- Se asume CPI = 1.0 ignorando bifurcaciones (ideal)
- En realidad se produce un "stall" de 3 ciclos por bifurcación
  - Si 30% bifurcaciones, "stall" 3 ciclos en 30%

Op	Freq	Cycles	F_j x CPI_j	(% Tiempo)
Otras	**70%**	**1**	**.7**	**(37%)**
Bifurc.	**30%**	**4**	**1.2**	**(63%)**

  - CPI real= 1.9
- La máquina real es 1/1.9 = 0.52 veces "más rápida" (o sea el doble de lenta!)

Tiempo de CPU, MIPS, MFLOP/s

$$\text{CPU time} = \frac{\text{CPI} \times \text{Instruction count}}{\text{Clockrate}}$$

$$\frac{\text{Clockrate}}{\text{CPI}} = \frac{\text{Instruction count}}{\text{CPU time}}$$

$$\frac{\text{Clockrate}}{\text{CPI} \times 10^6} = \frac{\text{Instruction count}}{\text{CPU time} \times 10^6} = \text{MIPS}$$

- Cómo comparar máquinas con diferentes ISA?
- Cómo comparar programas con diferentes mezclas de instrucciones?
- No tiene correlación con el rendimiento
  - Métrica de "marketing"

$$\text{MFLOP/s} = \frac{\text{Number of FP operations}}{\text{CPU time} \times 10^6}$$

- Popular en comunidad de supercomputadoras
- Normalmente no refleja donde se gasta el tiempo
- No todas las ops de PF son iguales
- Puede magnificar diferencias de rendimiento
  - Un algoritmo mejor puede correr más rápido, aún con mayor recuento de FLOP

Ley de Amdahl

- Artículo de Gene Amdahl [AMDA67]
- Estudia la aceleración potencial de un programa utilizando múltiples procesadores
- Concluye que:
  - El código debe ser paralelizable
  - La aceleración está acotada, y no mejora sensiblemente a partir de determinada cantidad de procesadores ("diminishing returns")

AMDA67 *Amdahl, G. "Validity of the Single-Processor Approach to Achieving Large-Scale Computing Capability", Proceedings of the AFIPS Conference, 1967.*

Ley de Amdahl: Fórmula

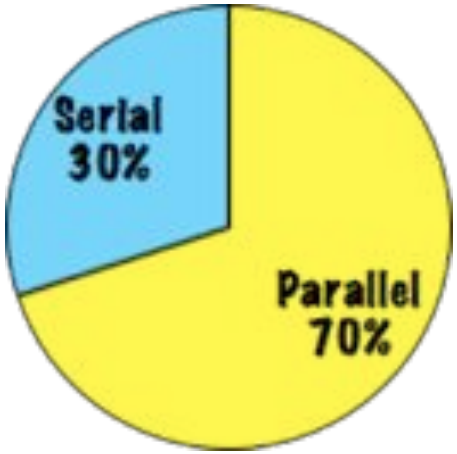
- Para un programa ejecutando en un único procesador
  - Fracción *f* del código infinitamente paralelizable sin "scheduling overhead"
  - Fracción (1-*f*) del código inherentemente serial
  - T es el tiempo total de ejecución en un procesador
  - N es la cantidad de procesadores que explotan completamente las porciones paralelizables del código

$$Speedup = \frac{\text{time to execute program on a single processor}}{\text{time to execute program on } N \text{ parallel processors}} = \frac{T(1-f)+Tf}{T(1-f)+\frac{Tf}{N}} = \frac{1}{(1-f)+\frac{f}{N}}$$

- Conclusiones
  - *f* pequeño, múltiples procesadores tienen poco efecto
  - *N* ->∞, aceleración acotada por 1/(1 – *f*)
    - "Diminishing returns" si se usan más procesadores

Ley de Amdahl: ejemplo (1/2)

Queremos ejecutar un programa en N CPUs

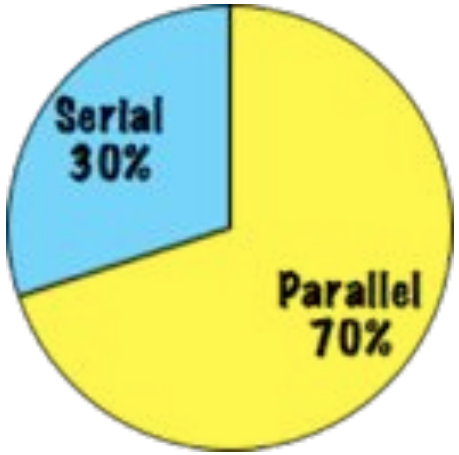


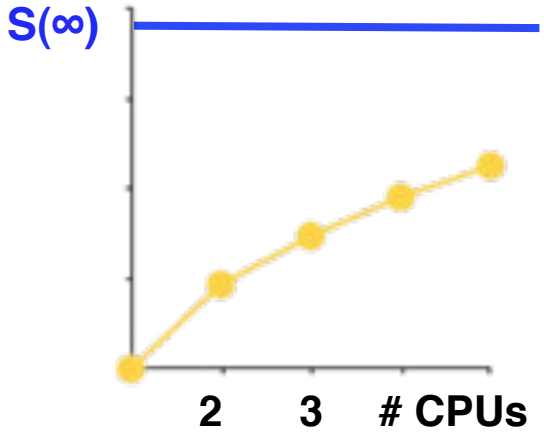
Se emplea el 30% del tiempo ejecutando código que no se puede paralelizar

Calcularemos aceleración para N = 2, 3, 4, 5, ∞

CPUs	2	3	4	5	∞
Aceleración					

Ley de Amdahl: ejemplo (2/2)





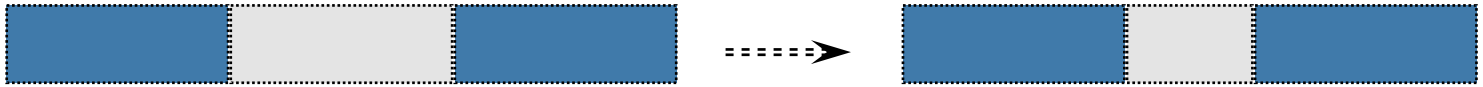
$$S = \frac{1}{(30\ \% + (70\% / N)\ ) / 100\ \%}$$

CPUs	2	3	4	5	∞
Aceleración	1.54	1.85	2.1	2.3	3.3

Ley de Amdahl: generalización

- Aceleración (Speedup) debida a mejora E:

$$\text{Speedup}(E) = \frac{\text{ExTime sin E}}{\text{ExTime con E}} = \frac{\text{Performance con E}}{\text{Performance sin E}}$$



La aceleración global depende no sólo del factor de aceleración de la mejora, sino también del tiempo que se aprovecha esa mejora

- Ejemplo:
  - Consideramos un programa que se ejecuta en 100 segs; la multiplicación es responsable de 80 segs. del total. ¿Cuánto debemos mejorar la multiplicación para que el programa se ejecute 4 veces más rápido?

- Principio: mejorar el caso más común

Ley de Amdahl: generalización

$$T_{old} = T_{no\ afectado} + T_{afectado\ old}$$

$$T_{new} = T_{no\ afectado} + T_{afectado\ new}$$

$$T_{new} = T_{no\ afectado} + \frac{T_{afectado\ old}}{speedup_{afectado}}, \text{ donde } speedup_{afectado} = \frac{T_{afectado\ old}}{T_{afectado\ new}}$$

$$T_{new} = \left(1 - Fracción_{afectada}\right) T_{old} + \frac{Fracción_{afectada}\, T_{old}}{speedup_{afectado}}$$

$$speedup_{total} = \frac{T_{old}}{T_{new}} = \frac{1}{\left(1 - Fracción_{afectada}\right) + \frac{Fracción_{afectada}}{speedup_{afectado}}}$$

Observación: $Fracción_{afectada}$ es con respecto a T_{old}

Ley de Amdahl: generalización

$$\text{speedup}_{\text{total}} = \frac{T_{\text{old}}}{T_{\text{new}}} = \frac{1}{\left(1 - \text{Fracción}_{\text{afectada}}\right) + \frac{\text{Fracción}_{\text{afectada}}}{\text{speedup}_{\text{afectado}}}}$$

- Ejemplo:
  - Instrucciones de Punto Flotante mejorada para correr a 2X; pero únicamente 10% del tiempo se consume en instrucciones de Punto Flotante

$$\textbf{Speedup}_{\textbf{total}} = \frac{1}{0.9 + 0.1 / 2} = 1.053$$

Ley de Amdahl: generalización

Qué cambio es más efectivo en una determinada máquina: Acelerar 10x la operación raíz cuadrada (sqrt), que ocupa el 20% del tiempo de ejecución, o acelerar 2x todas las operaciones de punto flotante, que ocupan el 50% del tiempo de ejecución? (Asumiendo que el costo de implementar cualquiera de las mejoras es el mismo, y que son mutuamente exclusivos)

Alternativa #1: mejorar solo sqrt

$$\text{Speedup}_{\text{sqrt}} = \frac{1}{0.8 + 0.2 / 10} = 1.22$$

Alternativa #1: mejorar todas las ops de PF

$$\text{Speedup}_{\text{total}} = \frac{1}{0.5 + 0.5 / 2} = 1.33$$

Mejorar todas las ops de PF es más efectivo

Métricas de Performance

- Kernels
  - Extractos “clave” de programas reales
- Synthetic benchmarks
  - Filosofía similar a kernels, intentan promediar frecuencia de operaciones de un gran conjunto de programas típicos
    - Whetstone (numérico), Dhrystone (E/S datos)
- Fácil de estandarizar, pero…
    - Pueden ser “violados”

- Programas reales
  - Se usan programas que representan una carga de trabajo típica, para determinada clase de aplicaciones
    - Compiladores de C, procesadores de texto, herramientas CAD

Benchmarks

- Benchmark Suites
  - Combinaciones de benchmarks, enfocados en algún aspecto relevante: CPU, transferencia de archivos, transacciones
- SPEC (System Performance Evaluation Cooperative)
  - Las compañías se ponen de acuerdo en un conjunto de programas y entradas reales
    - Aún pueden ser "violados"…
  - Buen indicador de performance (y tecnología de compiladores)
- SPECCPU: popular en sistemas de escritorio
  - Mide solamente CPU, programas enteros y de punto flotante
    - SPECint2000: 12 pgms. enteros, SPECfp2000: 14 pgms. punto flotante
  - Actualmente está vigente el SPECCPU2006
  - SPEC89, 92, 95, 2000, 2006
  - Los programas varían entre generaciones de SPEC
- Más información en http://www.spec.org/

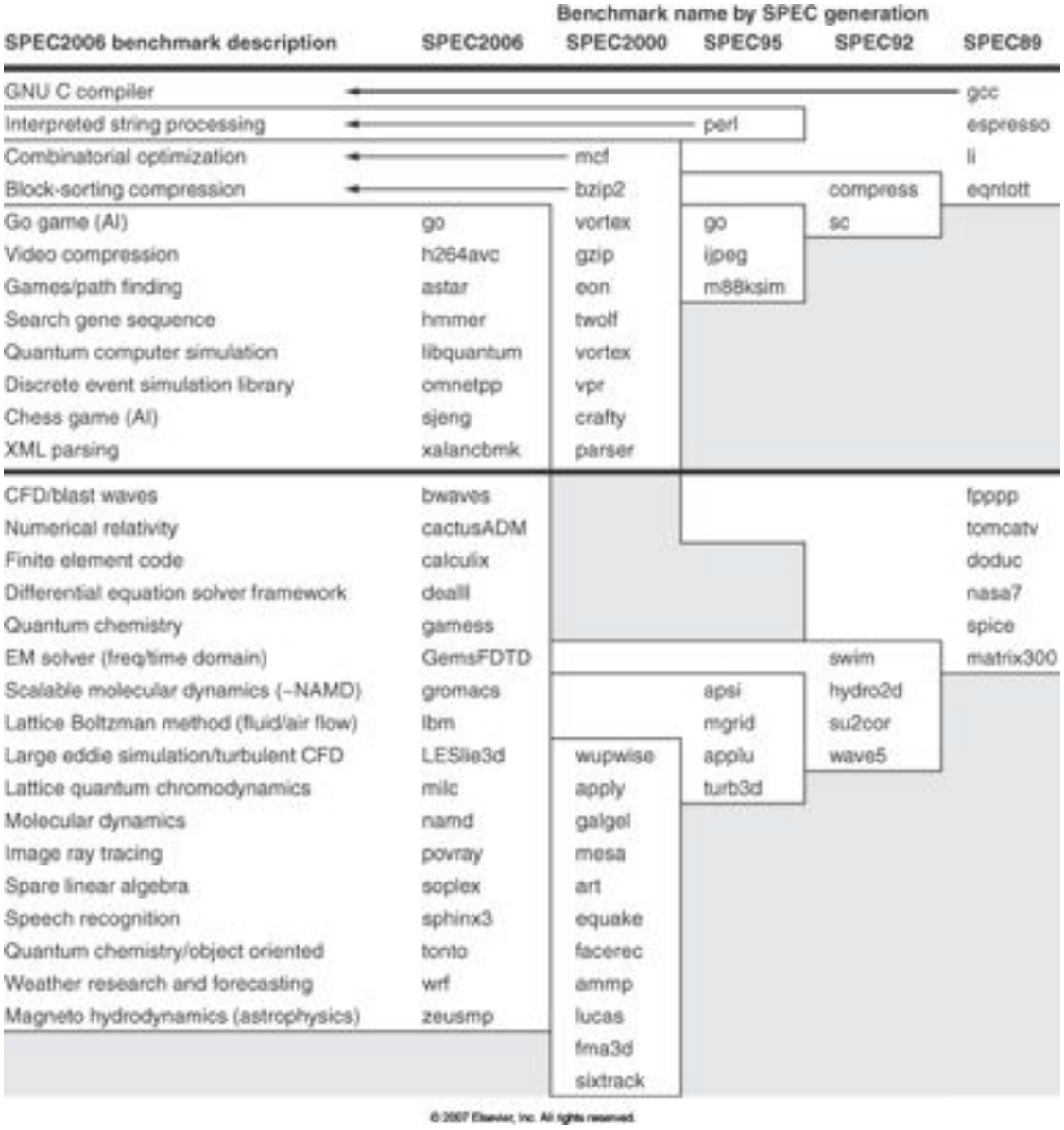
SPEC CPU2000: benchmarks de enteros

164.gzip	C	Compression
175.vpr	C	FPGA Circuit Placement and Routing
176.gcc	C	C Programming Language Compiler
181.mcf	C	Combinatorial Optimization
186.crafty	C	Game Playing: Chess
197.parser	C	Word Processing
252.eon	C++	Computer Visualization
253.perlbmk	C	PERL Programming Language
254.gap	C	Group Theory, Interpreter
255.vortex	C	Object-oriented Database
256.bzip2	C	Compression
300.twolf	C	Place and Route Simulator

SPEC CPU2000: benchmarks de punto flotante

168.wupwise	Fortran 77	Physics / Quantum Chromodynamics
171.swim	Fortran 77	Shallow Water Modeling
172.mgrid	Fortran 77	Multi-grid Solver: 3D Potential Field
173.applu	Fortran 77	Parabolic / Elliptic Partial Differential Equations
177.mesa	C	3-D Graphics Library
178.galgel	Fortran 90	Computational Fluid Dynamics
179.art	C	Image Recognition / Neural Networks
183.equake	C	Seismic Wave Propagation Simulation
187.facerec	Fortran 90	Image Processing: Face Recognition
188.ammp	C	Computational Chemistry
189.lucas	Fortran 90	Number Theory / Primality Testing
191.fma3d	Fortran 90	Finite-element Crash Simulation
200.sixtrack	Fortran 77	High Energy Nuclear Physics Accelerator Design
301.apsi	Fortran 77	Meteorology: Pollutant Distribution

SPEC2006



SPEC2006 benchmark description	Benchmark name by SPEC generation				
	SPEC2006	SPEC2000	SPEC95	SPEC92	SPEC89
GNU C compiler					gcc
Interpreted string processing			perl		espresso
Combinatorial optimization		mcf			li
Block-sorting compression		bzip2		compress	eqntott
Go game (AI)	go	vortex	go	sc	
Video compression	h264avc	gzip	ijpeg		
Games/path finding	astar	eon	m88ksim		
Search gene sequence	hmmer	twolf			
Quantum computer simulation	libquantum	vortex			
Discrete event simulation library	omnetpp	vpr			
Chess game (AI)	sjeng	crafty			
XML parsing	xalancbmk	parser			
CFD/blast waves	bwaves				fpppp
Numerical relativity	cactusADM				tomcatv
Finite element code	calculix				doduc
Differential equation solver framework	dealII				nasa7
Quantum chemistry	gamess				spice
EM solver (freq/time domain)	GemsFDTD			swim	matrix300
Scalable molecular dynamics (~NAMD)	gromacs		apsi	hydro2d	
Lattice Boltzman method (fluid/air flow)	lbm		mgrid	su2cor	
Large eddie simulation/turbulent CFD	LESlie3d	wupwise	applu	wave5	
Lattice quantum chromodynamics	milc	apply	turb3d		
Molecular dynamics	namd	galgel			
Image ray tracing	povray	mesa			
Spare linear algebra	soplex	art			
Speech recognition	sphinx3	equake			
Quantum chemistry/object oriented	tonto	facerec			
Weather research and forecasting	wrf	ammp			
Magneto hydrodynamics (astrophysics)	zeusmp	lucas			
		fma3d			
		sixtrack			

La CPU no es todo…
…otros benchmarks SPEC

- SPECint y SPECfp miden tareas de computación intensiva, enfatizando los siguientes elementos de la arquitectura:
  - CPU
  - Arquitectura de memoria
  - Compiladores
- NO atacan otros elementos tales como el sistema operativo, la red, los gráficos o el subsistema de E/S. Existen otros benchmarks, por ejemplo para:
  - Graphics and Workstation Performance
  - High Performance Computing, OpenMP, MPI
  - Java Client/Server
  - Mail Servers
  - Network File System
  - Web Servers

Reportes de performance

- Los reportes deben permitir ***reproducir*** las mediciones de performance
  - Se deben detallar exhaustivamente los experimentos
- La máquina de referencia para SPEC CPU 2006 en la "Sun Ultra Enterprise 2", de 1997, que utiliza una CPU UltraSPARC II de 296 MHz.
- "Base" refiere a una configuración estándar de los compiladores, y "Peak" a la configuración con optimizaciones.

Resumen de performance

- Interesa tener UN número que resuma la performance de un sistema

- Lo "natural" sería la Media Aritmética basada en el tiempo de ejecución:
  - $\Sigma(T_i)/n$
  - T_i es el tiempo de ejecución del i-ésimo de los n programas que componen la carga de trabajo
  - Pregunta: cual es la mezcla de programas mas adecuada para medir la performance? "Pesan" todos lo mismo?
- Podríamos usar la Media Aritmética Ponderada
  - $\Sigma(W_i*T_i)$
- Pero, cómo se eligen los pesos?

Resumen de performance

- SPEC benchmarks: SpecRatio = Tiempo de Ejecución normalizado con respecto a una "máquina canónica", (Y corre n veces más rápido que X)
  - En cociente de ratios de dos sistemas NO influye la referencia!

$$\frac{\text{SPECRatio}_A}{\text{SPECRatio}_B} = \frac{\frac{\text{Execution time}_{\text{reference}}}{\text{Execution time}_A}}{\frac{\text{Execution time}_{\text{reference}}}{\text{Execution time}_B}} = \frac{\text{Execution time}_B}{\text{Execution time}_A} = \frac{\text{Performance}_A}{\text{Performance}_B}$$

- Para promediar razones (como SpecRatio) se usa la Media Geométrica:
  - $(\Pi \text{ muestra}_j)^{1/n} = \Pi \ (\ T_j / N_j \ )^{1/n}$
  - La media geométrica de las razones T_j / N_j es igual a la razón entre las medias geométricas de los T_j y los N_j es decir, la media geométrica de los de los SPECratios es igual a la razón de medias geométricas.

$$\frac{\text{Geometric mean}_A}{\text{Geometric mean}_B} = \frac{\sqrt[n]{\prod_{i=1}^{n} \text{SPECRatio A}_i}}{\sqrt[n]{\prod_{i=1}^{n} \text{SPECRatio B}_i}} = \sqrt[n]{\prod_{i=1}^{n} \frac{\text{SPECRatio A}_i}{\text{SPECRatio B}_i}}$$

$$= \sqrt[n]{\prod_{i=1}^{n} \frac{\frac{\text{Execution time}_{\text{reference}_i}}{\text{Execution time}_{A_i}}}{\frac{\text{Execution time}_{\text{reference}_i}}{\text{Execution time}_{B_i}}}} = \sqrt[n]{\prod_{i=1}^{n} \frac{\text{Execution time}_{B_i}}{\text{Execution time}_{A_i}}} = \sqrt[n]{\prod_{i=1}^{n} \frac{\text{Performance}_{A_i}}{\text{Performance}_{B_i}}}$$

Resumen de performance: Ejemplo #1

Benchmarks	Ultra 5 Time (sec)	Opteron Time (sec)	SPECRatio	Itanium 2 Time (sec)	SPECRatio	Opteron/Itanium Times (sec)	Itanium/Opteron SPECRatios
wupwise	1600	51.5	31.06	56.1	28.53	0.92	0.92
swim	3100	125.0	24.73	70.7	43.85	1.77	1.77
mgrid	1800	98.0	18.37	65.8	27.36	1.49	1.49
applu	2100	94.0	22.34	50.9	41.25	1.85	1.85
mesa	1400	64.6	21.69	108.0	12.99	0.60	0.60
galgel	2900	86.4	33.57	40.0	72.47	2.16	2.16
art	2600	92.4	28.13	21.0	123.67	4.40	4.40
equake	1300	72.6	17.92	36.3	35.78	2.00	2.00
facerec	1900	73.6	25.80	86.9	21.86	0.85	0.85
ammp	2200	136.0	16.14	132.0	16.63	1.03	1.03
lucas	2000	88.8	22.52	107.0	18.76	0.83	0.83
fma3d	2100	120.0	17.48	131.0	16.09	0.92	0.92
sixtrack	1100	123.0	8.95	68.8	15.99	1.79	1.79
apsi	2600	150.0	17.36	231.0	11.27	0.65	0.65
Geometric mean			20.86		27.12	1.30	1.30

Figure 1.14 SPECfp2000 execution times (in seconds) for the Sun Ultra 5—the reference computer of SPEC2000—and execution times and SPECRatios for the AMD Opteron and Intel Itanium 2. (SPEC2000 multiplies the ratio of execution times by 100 to remove the decimal point from the result, so 20.86 is reported as 2086.) The final two columns show the ratios of execution times and SPECratios. This figure demonstrates the irrelevance of the reference computer in relative performance. The ratio of the execution times is identical to the ratio of the SPECRatios, and the ratio of the geometric means (27.12/20.86 = 1.30) is identical to the geometric mean of the ratios (1.30).

Resumen de performance: Ejemplo #2

	Sistema A	Sistema B	Sistema C	W(1)	W(2)	W(3)
Programa P1 (segs)	1	10	20	0,5	0,909	0,999
Programa P2 (segs)	1000	100	20	0,5	0,091	0,001
Tiempo total (segs)	1001	110	40			
Media aritmética: W(1)	500,5	55	20			
Media aritmética: W(2)	91,909	18,19	20			
Media aritmética: W(3)	1,999	10,09	20			

- W(1) pesa igual P1 y P2
- W(2) peso inversamente proporcional a Tiempo de Ejecución en B
- W(3) peso inversamente proporcional a Tiempo de Ejecución en A

- Comparar resultados de diferentes medias:

	Normalizado a A			Normalizado a B			Normalizado a C		
	A	**B**	**C**	**A**	**B**	**C**	**A**	**B**	**C**
Programa P1	1	10	20	0,1	1	2	0,05	0,5	1
Programa P2	1	0,1	0,02	10	1	0,2	50	5	1
Media aritmética	1	5,05	10	5,05	1	1,1	25,025	2,75	1
Media geométrica	1	1	0,63	1	1	0,63	1,5811	1,58	1
Tiempo total	1	0,11	0,04	9,1	1	0,36	25,025	2,75	1

Resumen de performance: comentarios

- Hemos visto que la media geométrica es una métrica consistente cuando el tiempo de ejecución es relativo a una máquina de referencia, como en SPEC.
- Sin embargo, mirando el ejemplo anterior, tomado de [SMITH88], podemos ver que la media geométrica puede no ser adecuada.
- En este artículo se concluye:
  - Si se agrega información de rendimiento con respecto a una referencia, se debe calcular la media antes de normalizar.
  - Una alternativa es usar la media armónica, que está mejor relacionada con la Ley de Amdahl.
  - O usar directamente el tiempo.

SMITH88 *J. E. Smith "Characterizing Computer Performance with a Single Number", Communications of the ACM, Volume 31 (10), Oct 1988, pp 1202 - 1206.*

Resumen de performance: comentarios

- Podemos decidir si la media predice adecuadamente la performance usando la desviación estándar
  - La desviación estándar geométrica es multiplicativa
  - Probando con lognormal: tomar logaritmos de los SPECRatios, calcular media y desviación estándar, y luego exponenciar para tener el resultado correcto:

$$GeometricMean = \exp\left(\frac{1}{n} \times \sum_{i=1}^{n} \ln\left(SPECRatio_i\right)\right)$$

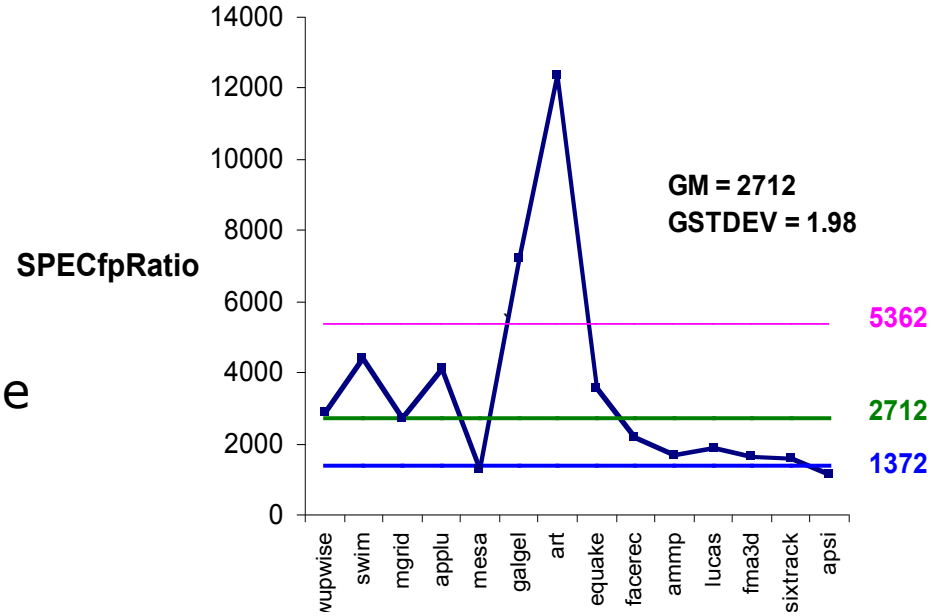
$$GeometricStDev = \exp\left(StDev\left(\ln\left(SPECRatio_i\right)\right)\right)$$

- Volvamos al Ejemplo #1

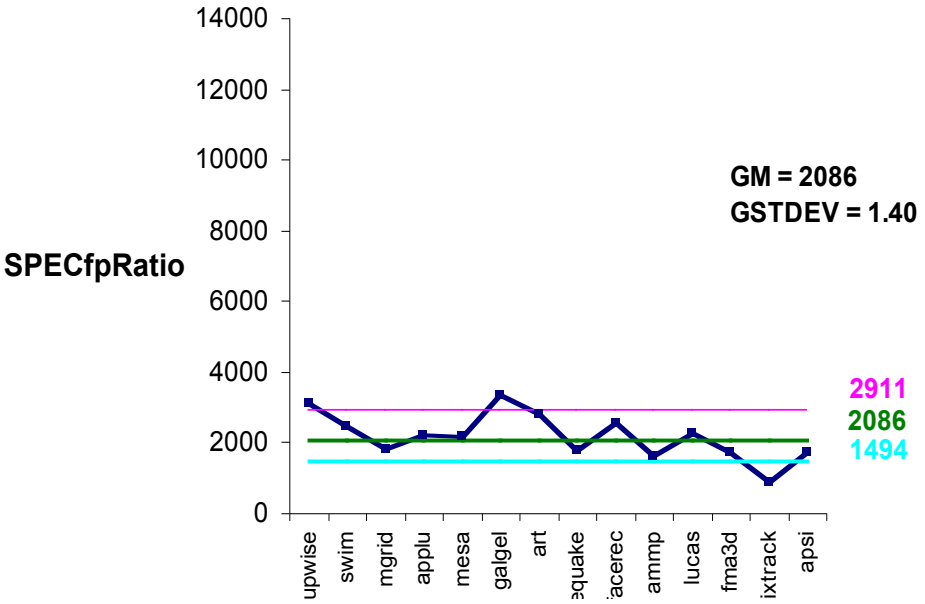
Resumen de performance: comentarios

- La desviación estándar del Itanium 2 (1.98) es mayor que la del Athlon (1.40), es decir que los resultados difieren más de la media, son menos predecibles.
- Dentro de la desviación estándar:
  - 10/ 14 benchmarks (71%) para Itanium 2
  - 11/ 14 benchmarks (78%) para Athlon
- Es decir que los resultados se ajustan razonablemente a una distribución lognormal (valor esperado 68%)

GM y StDev multiplicativa del SPECfp2000 para Itanium 2



GM y StDev multiplicativa del SPECfp2000 para AMD Athlon



Evaluación de Performance

- Los benchmarks son tomados en cuenta por la industria
- El diseño se debe orientar a mejorar la performance de las cargas de trabajo (programas) reales, y no solamente los programas que ayudan a vender!
- Para una arquitectura dada la mejora de la performance viene de:
  - Incremento de la frecuencia de reloj (sin afectar los CPI!)
  - Mejora en la organización del procesador para baja los CPI
  - Mejora en los compiladores para bajar CPI y/o recuento de instrucciones

Performance: impacto del cache

Impacto del cache

- Con la CPU secuencial:
  - Velocidad = f(no. operaciones)
- Mejoras…
  - Ejecución en pipeline, crecimiento de la frecuencia del reloj
  - Ejecución superescalar y fuera de orden
  - Caches
  - ….
- Ahora tenemos:
  - Velocidad = f(accesos a memoria no-cacheados)

Cache performance

- Considerando los Miss del cache:

$$CPUtime = IC \times \left(CPI_{Execution} + \frac{MemAccess}{Inst} \times MissRate \times MissPenalty \right) \times CycleTime$$

$$CPUtime = IC \times \left(CPI_{Execution} + \frac{MemMisses}{Inst} \times MissPenalty \right) \times CycleTime$$

  - $CPI_{Execution}$ includes ALU and Memory instructions

- Separando el componente de memoria:
  - AMAT = Average Memory Access Time
  - CPI_{ALUOps} no incluye instrucciones de memoria

$$CPUtime = IC \times \left(\frac{AluOps}{Inst} \times CPI_{AluOps} + \frac{MemAccess}{Inst} \times AMAT \right) \times CycleTime$$

$$AMAT = HitTime + MissRate \times MissPenalty$$
$$= (HitTime_{Inst} + MissRate_{Inst} \times MissPenalty_{Inst}) +$$
$$(HitTime_{Data} + MissRate_{Data} \times MissPenalty_{Data})$$

Impacto del cache: Ejemplo

- Consideramos un procesador que ejecuta a
  - Clock Rate = 200 MHz (5 ns/ciclo), CPI Ideal (sin misses) = 1.1
  - 50% arit/logic, 30% ld/st, 20% control

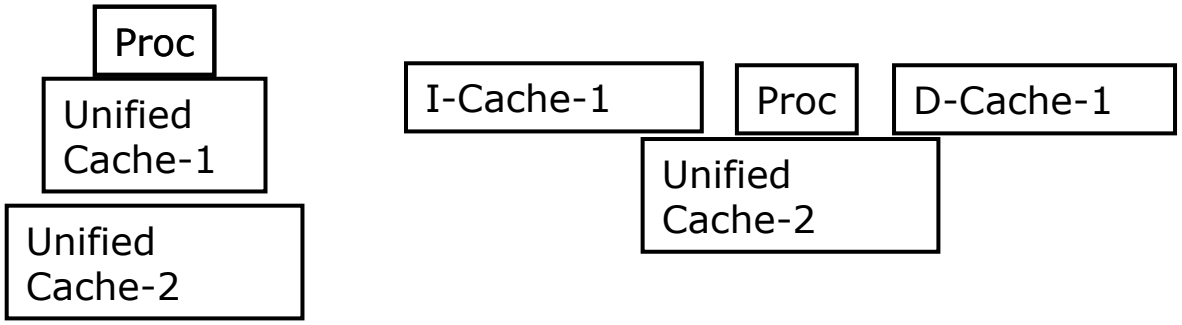
- Se mide:
  - 10% de los accesos a memoria tiene una penalización de 50 ciclos
  - 1% de las instrucciones tiene la misma penalización

- CPI = ideal CPI + average stalls per instruction

  1.1(ciclos/inst) +
  [0.30 (DataMops/inst)
  x 0.10 (miss/DataMop) x 50 (ciclo/miss)] +
  [1 (InstMop/ins)
  x 0.01 (miss/InstMop) x 50 (ciclo/miss)]
  = (1.1 + 1.5 + .5) cycle/ins = 3.1

- 2/3.1=64% del tiempo CPU detenida esperando por la memoria!
- AMAT=(1/1.3)x[1+0.01x50]+(0.3/1.3)x[1+0.1x50]=2.54

Caché unificada de I&D?



- Ejemplo:
  - 16KB I&D: Inst miss rate=0.64%, Data miss rate=6.47%
  - 32KB unified: Aggregate miss rate=1.99%
- Cuál es mejor? (ignorando la cache L2)
  - Sabemos que el 33% son ops. de datos ⇒ 75% accesos totales son instrucciones (1.0/1.33)
  - hit time=1, miss time=50
  - Obs: hit de datos tiene un stall para la cache unificada (solo un puerto de acceso a memoria)

$AMAT_{Harvard}$=75%x(1+0.64%x50)+25%x(1+6.47%x50) = 2.05

$AMAT_{Unificada}$=75%x(1+1.99%x50)+25%x(1+1+1.99%x50)= 2.24

Cómo mejorar la performance del Cache?

$$AMAT = HitTime + MissRate \times MissPenalty$$

1. Reducir el miss rate,
2. Reducir la miss penalty, o
3. Reducir el tiempo de hit en el cache.

Recordar: causas de los misses (3C): Compulsivo, Capacidad y Conflicto (y una 4a: Coherencia)

Veremos una técnica de software para reducir misses

Reducción de misses por optimizaciones del compilador

- Instrucciones
  - Reordenar procedimientos en memoria para reducir misses
  - "Profiling" para prevenir conflictos
  - Se han reportado [McFarling89] reducciones del 75% en una cache de 8KB con correspondencia directa y bloques de bytes
- Datos
  - Merging Arrays
    - Mejorar localidad espacial: estructura de datos vs. 2 arrays
  - Loop Interchange
    - Cambiar anidamiento para acceder a los datos en el orden de almacenamiento en memoria
  - Loop Fusion
    - Combinar loops independientes
  - Blocking
    - Mejorar localidad temporal en operaciones matriciales

McFarling89 *S. McFarling, "Program optimization for instruction caches", ASPLOS-III Proceedings of the 3rd international conference on Architectural support for programming languages and operating systems. ACM New York, NY, USA 1989.*

Merging Arrays

```
/* Before */
int val[SIZE];
int key[SIZE];
```

```
/* After */
struct merge {
  int val;
  int key;
};
struct merge merged_array[SIZE];
```

- Reduce conflictos entre **`val`** y **`key`**

Loop Interchange

```
/* Before */
for (k = 0; k < 100; k++)
  for (j = 0; j < 100; j++)
    for (i = 0; i < 5000; i++)
      x[i][j] = 2 * x[i][j];
```

```
/* After */
for (k = 0; k < 100; k++)
  for (i = 0; i < 5000; i++)
    for (j = 0; j < 100; j++)
      x[i][j] = 2 * x[i][j];
```

- Acceso secuencial en lugar de saltar en la memoria cada 100 palabras

Loop Fusion

```
/* Before */
for (i = 0; i < N; i++)
  for (j = 0; j < N; j++)
    a[i][j] = 1/b[i][j] * c[i][j];
for (i = 0; i < N; i++)
  for (j = 0; j < N; j++)
    d[i][j] = a[i][j] + c[i][j];
```

```
/* After */
for (i = 0; i < N; i++)
  for (j = 0; j < N; j++)
    { a[i][j] = 1/b[i][j] * c[i][j];
      d[i][j] = a[i][j] + c[i][j];}
```

- Antes: 2 misses por acceso a **a** y **c**
- Ahora: 1 miss por acceso a **a** y **c**

Otra alternativa: agregar cache L2

- Ecuaciones de la cache L2

  $AMAT = Hit\ Time_{L1} + Miss\ Rate_{L1} \times Miss\ Penalty_{L1}$

  $Miss\ Penalty_{L1} = Hit\ Time_{L2} + Miss\ Rate_{L2} \times Miss\ Penalty_{L2}$

  $AMAT = Hit\ Time_{L1} +$
  $Miss\ Rate_{L1} \times (Hit\ Time_{L2} + Miss\ Rate_{L2} + Miss\ Penalty_{L2})$

- Definiciones:
  - *Local miss rate*— misses en este nivel dividido por el número total de accesos a memoria *a este cache* (Miss $rate_{L2}$)
  - *Global miss rate*—misses en este nivel dividido por la cantidad total de accesos a memoria *generados por la CPU*

  - Importa el Global Miss Rate

Comentarios

- Para considerar alternativas de diseño hay que conocer los fundamentos cuantitativos del rendimiento
- La medida correcta de rendimiento es el tiempo de CPU
- Las mejoras en arquitectura y organización afectan los componentes (CPI, frecuencia, recuento), y en general hay que medirlos para ser precisos
- Algunas técnicas sencillas pueden mejorar el rendimiento de sistemas con cache

Ideas para mejorar el rendimiento

- Aumentar…
  - Incrementar la frecuencia del reloj
    - Pero…los tiempos de acceso a memoria y E/S pueden ser un cuello de botella…
  - Ancho de los registros
  - Ancho del bus de datos
    - Mayor tasa de transferencia
  - Ancho del bus de direcciones
    - NO mejora la velocidad de acceso pero se puede direccionar más memoria directamente…
      - Más memoria física, menos swap de memoria virtual
  - Pipelining
  - Jerarquía de Memoria
    - L1 & L2 cache
    - Memoria virtual