

# Computer Architecture

## 02. Performance Analysis

Jianhua Li

College of Computer and Information  
Hefei University of Technology

slides are adapted from CA course of wisc, princeton, mit, berkeley, etc.

The uses of the slides of this course are for educational purposes only and should be used only in conjunction with the textbook. Derivatives of the slides must acknowledge the copyright notices of this and the originals.



# 性能分析

- Which of the following airplanes has the best performance?

<u>Airplane</u>	<u>Passengers</u>	<u>Range (mi)</u>	<u>Speed (mph)</u>
Boeing 737-100	101	630	598
Boeing 747	470	4150	610
BAC/Sud Concorde	132	4000	1350
Douglas DC-8-50	146	8720	544

# 性能分析

- Which of the following airplanes has the best performance?

Airplane	Passengers	Range (mi)	Speed (mph)
Boeing 737-100	101	630	598
Boeing 747	470	4150	610
BAC/Sud Concorde	132	4000	1350
Douglas DC-8-50	146	8720	544

- How much faster is the Concorde vs. the 747
- How much bigger is the 747 vs. DC-8?

# 性能分析

- Which computer is the fastest?
  - 给你2台计算机，你觉得那一台好？
  - 你如何去衡量计算机的性能？

# 性能分析

- Which computer is the fastest?
- The answer is not simple:
  - Scientific simulation – FP performance
  - Program development – Integer performance
  - Database workload – Memory, I/O

# 计算机的性能

- Want to buy the fastest computer for what you want to do?
  - **Workload** is very important
  - Correct measurement and analysis method

# 计算机的性能

- Want to buy the fastest computer for what you want to do?
  - Workload is very important
  - Correct measurement and analysis method
- Want to design the fastest computer for what the customer wants to pay?
  - Cost is an important criterion

# 本节内容

- Time and performance
- Iron Law → Time
- MIPS and MFLOPS
- Which programs and how to average
- Amdahl's law

# 如何定义性能？

- What is important to whom?

# 如何定义性能？

- What is important to whom?
- Computer system user
  - Minimize elapsed time for program:  
$$t_{\text{resp}} = t_{\text{end}} - t_{\text{start}}$$
  - Called **response time (响应时间)**

# 如何定义性能？

- What is important to whom?
- Computer system user
  - Minimize elapsed time for program:  
$$t_{\text{resp}} = t_{\text{end}} - t_{\text{start}}$$
  
– Called response time (响应时间)
- Datacenter manager
  - Maximize completion rate = #jobs/second
  - Called **throughput** (吞吐率)

# Response Time vs. Throughput

- Is throughput =  $1/\text{avg. response time}$ ?

# Response Time vs. Throughput

- Is throughput =  $1/\text{avg. response time}$ ?
  - Only if **NO** overlap
  - Otherwise, throughput  $> 1/\text{avg. response time}$

# Response Time vs. Throughput

- Is throughput =  $1/\text{avg. response time}$ ?
  - Only if NO overlap
  - Otherwise, throughput >  $1/\text{avg. response time}$
  - 举例：A lunch buffet (自助餐)
    - ✓ Assume 5 entrees (主菜)
    - ✓ Each person takes 2 minutes/entree
    - ✓ Throughput is 1 person every 2 minutes
    - ✓ BUT time to fill up tray (盘子) is 10 minutes

Why and what would the throughput be otherwise?

- ✓ 5 people simultaneously filling tray (overlap)
- ✓ Without overlap, throughput =  $1/10$

# 对于计算机，什么是性能？

- For computer architects
  - CPU time = time spent running a program
- Intuitively, **bigger should be faster**, so:
  - Performance =  $1/X$  time, where X is response, CPU execution, etc.
- Elapsed time = CPU time + I/O wait
- **We will concentrate on CPU time**

# 如何提升性能？

- Improve (a) response time or (b) throughput?
  - Faster CPU
  - Add more CPUs

# 如何提升性能？

- Improve (a) response time or (b) throughput?
  - Faster CPU
    - Helps both (a) and (b)
  - Add more CPUs

# 如何提升性能？

- Improve (a) response time or (b) throughput?
  - Faster CPU
    - Helps both (a) and (b)
  - Add more CPUs
    - Helps (b) and perhaps (a) due to less queueing

# 如何比较性能？

- Machine A is n times faster than machine B **iff** :  
 $\text{perf}(A)/\text{perf}(B) = \text{time}(B)/\text{time}(A) = n$
- Machine A is x% faster than machine B iff  
 $\text{perf}(A)/\text{perf}(B) = \text{time}(B)/\text{time}(A) = 1 + x/100$

# 如何比较性能？

- Machine A is n times faster than machine B iff  
 $\text{perf}(A)/\text{perf}(B) = \text{time}(B)/\text{time}(A) = n$
- Machine A is x% faster than machine B iff  
 $\text{perf}(A)/\text{perf}(B) = \text{time}(B)/\text{time}(A) = 1 + x/100$
- 举例 :  $\text{time}(A) = 10\text{s}$ ,  $\text{time}(B) = 15\text{s}$ 
  - $15/10 = 1.5 \Rightarrow A$  is 1.5 times faster than B
  - $15/10 = 1.5 \Rightarrow A$  is 50% faster than B

# 如何比较性能？

- Machine A is n times faster than machine B iff  
 $\text{perf}(A)/\text{perf}(B) = \text{time}(B)/\text{time}(A) = n$
- Machine A is x% faster than machine B iff  
 $\text{perf}(A)/\text{perf}(B) = \text{time}(B)/\text{time}(A) = 1 + x/100$
- 举例 :  $\text{time}(A) = 10\text{s}$ ,  $\text{time}(B) = 15\text{s}$ 
  - $15/10 = 1.5 \Rightarrow A$  is 1.5 times faster than B
  - $15/10 = 1.5 \Rightarrow A$  is 50% faster than B

How to obtain the time?

# 方法：通过分解指令

- Breaking program into instructions
  - HW is aware of instructions, not programs

# 方法：通过分解指令

- Breaking program into instructions
  - HW is aware of instructions, not programs
- At lower level, HW breaks instructions into cycles
  - Lower level state machines change state every cycle

# 方法：通过分解指令

- Breaking program into instructions
  - H/W is aware of instructions, not programs
- At lower level, H/W breaks instructions into cycles
  - Lower level state machines change state every cycle
- For example:
  - 1GHz Snapdragon runs 1000M cycles/sec, 1 cycle = 1ns
  - 2.5GHz Core i7 runs 2.5G cycles/sec, 1 cycle = 0.25ns

# Iron Law in Performance

$$\text{Processor Performance} = \frac{\text{Time}}{\text{Program}}$$
$$= \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Time}}{\text{Cycle}}$$

**(code size)**                    **(CPI)**                    **(cycle time)**

Architecture → Implementation → Realization

Compiler Designer

Processor Designer

Chip Designer

# Iron Law in Performance

- Instructions/Program
  - Instructions executed, not static code size
  - Determined by algorithm, compiler, ISA

# Iron Law in Performance

- Instructions/Program
  - Instructions executed, not static code size ← why?
  - Determined by algorithm, compiler, ISA

# Iron Law in Performance

- Instructions/Program
  - Instructions executed, not static code size
  - Determined by algorithm, compiler, ISA
- Cycles/Instruction
  - Determined by ISA and CPU organization
  - Overlap among instructions reduces this term

# Iron Law in Performance

- Instructions/Program
  - Instructions executed, not static code size
  - Determined by algorithm, compiler, ISA
- Cycles/Instruction
  - Determined by ISA and CPU organization
  - Overlap among instructions reduces this term
- Time/cycle
  - Determined by technology, organization, clever circuit design

# Iron Law

- Instructions/Program
  - Instructions executed, not static code size
  - Determined by algorithm, compiler, ISA
- Cycles/Instruction
  - Determined by ISA and CPU organization
  - Overlap among instructions reduces this term
- Time/cycle
  - Determined by technology, organization, clever circuit design

 可以从计算机组成的角度考虑  
一下这个问题

# 设计的目标

- Minimize time which is the product, **NOT** the isolated terms
- Common error to miss terms while devising optimizations
  - E.g. ISA change to decrease instruction count
  - BUT leads to CPU organization which makes clock slower

# 设计的目标

- Minimize time which is the product, NOT isolated terms
- Common error to miss terms while devising optimizations
  - E.g. ISA change to decrease instruction count
  - BUT leads to CPU organization which makes clock slower
- Be mind that: **terms are inter-related**

# 其它指标

- MIPS: 每秒百万次整型操作
- MIPS = instruction count/(execution time  $\times 10^6$ )  
= clock rate/(CPI  $\times 10^6$ )
- But MIPS has serious shortcomings

# MIPS的问题(示例)

- E.g. without FP hardware, an FP op may take 50 single-cycle instructions
- With FP hardware, only one 2-cycle instruction
- Thus, adding FP hardware:
  - CPI increases (why?)  $50/50 \Rightarrow 2/1$
  - Instructions/program decreases (why?)  $50 \Rightarrow 1$
  - Total execution time decreases  $50 \Rightarrow 2$
- BUT, MIPS gets worse!  $50 \text{ MIPS} \Rightarrow 2 \text{ MIPS}$

# MIPS的问题(根源)

- Ignores program
- Usually used to quote peak performance
  - Ideal conditions => guaranteed not to exceed!

# MIPS的问题(根源)

- Ignores program
- Usually used to quote peak performance
  - Ideal conditions => guaranteed not to exceed!
- When is MIPS ok?
  - Same compiler, same ISA
  - E.g. same binary running on AMD Jaguar, Intel Core i7
  - Why?

# MIPS的问题(根源)

- Ignores program
- Usually used to quote peak performance
  - Ideal conditions => guaranteed not to exceed!
- When is MIPS ok?
  - Same compiler, same ISA
  - E.g. same binary running on AMD Jaguar, Intel Core i7
  - Why?  
**Instr/program is constant and can be factored out**

# 其它指标

- MFLOPS = FP ops in program/(execution time x 10<sup>6</sup>)
- Assuming FP ops independent of compiler and ISA
  - Often safe for numeric codes: matrix size determines # of FP ops/program
  - However, not always safe:
    - Missing instructions (e.g. FP divide)
    - Optimizing compilers

# 性能分析的主要原则

- Use **ONLY** Time
- Beware when reading, especially if details are omitted
  - Be careful when you buy products
- Beware of Peak
  - “Guaranteed not to exceed”

# Iron Law Example A

- **Machine A:** clock 1ns, CPI 2.0, for program x
- **Machine B:** clock 2ns, CPI 1.2, for program x
- Which machine is faster and how much?

# Iron Law Example A

- Machine A: clock 1ns, CPI 2.0, for program x
- Machine B: clock 2ns, CPI 1.2, for program x
- Which machine is faster and how much?

2分钟, everybody !

Have a try!

# Iron Law Example A

- Machine A: clock 1ns, CPI 2.0, for program x
- Machine B: clock 2ns, CPI 1.2, for program x
- Which machine is faster and how much?

Ans:

Time/Program = instr/program x cycles/instr x sec/cycle

$$\text{Time}(A) = N \times 2.0 \times 1 = 2N$$

$$\text{Time}(B) = N \times 1.2 \times 2 = 2.4N$$

$$\text{Compare: Time}(B)/\text{Time}(A) = 2.4N/2N = 1.2$$

# Iron Law Example A

- Machine A: clock 1ns, CPI 2.0, for program x
- Machine B: clock 2ns, CPI 1.2, for program x
- Which is faster and how much?

Ans:

Time/Program = instr/program x cycles/instr x sec/cycle

$$\text{Time}(A) = N \times 2.0 \times 1 = 2N$$

$$\text{Time}(B) = N \times 1.2 \times 2 = 2.4N$$

$$\text{Compare: Time}(B)/\text{Time}(A) = 2.4N/2N = 1.2$$

- So, Machine A is **20%** faster than Machine B for program x

## Iron Law Example B

Keep clock(A) @ 1ns and clock(B) @2ns

To get equal performance: if CPI(B)=1.2,  
what should CPI(A) be?

# Iron Law Example

Keep clock(A) @ 1ns and clock(B) @2ns

To get equal performance, if CPI(B)=1.2,  
what should CPI(A) be?

$$\begin{aligned}\text{Ans: Time(B) / Time(A)} &= 1 \\ &= (N \times 2 \times 1.2) / (N \times 1 \times \text{CPI}(A)) \\ \Rightarrow \text{CPI}(A) &= 2.4\end{aligned}$$

# Iron Law Example C

- Keep CPI(A)=2.0 and CPI(B)=1.2
- For equal performance: if  $\text{clock}(B)=2\text{ns}$ , what should  $\text{clock}(A)$  be?

Ans:  $\text{Time}(B)/\text{Time}(A) = 1$   
 $= (N \times 2.0 \times \text{clock}(A))/(N \times 1.2 \times 2)$   
 $\Rightarrow \text{clock}(A) = 1.2\text{ns}$

# 总结一

- Time and performance: Machine A n times faster than Machine B
  - Iff  $\text{Time(B)}/\text{Time(A)} = n$
- Iron Law: Performance = Time/program =
$$= \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Time}}{\text{Cycle}}$$

(code size)                            (CPI)                            (cycle time)
- Other Metrics: MIPS and MFLOPS
  - Beware of peak and omitted details

# 怎么衡量不同机器的性能？

- Execution time of what program?
- Best case – you always run the same set of programs
  - Port them and **time the whole workload**

# 怎么衡量不同机器的性能？

- Execution time of what program?
- Best case – you always run the same set of programs
  - Port them and time the whole workload
- In reality, use benchmarks
  - Programs chosen to measure performance
  - Predict performance of actual workload
  - Saves effort and money

Representative? Honest? Benchmarketing...

# 如何表示性能(用单个数值)

	Machine A	Machine B
Program 1	1	10
Program 2	1000	100
Total	1001	110

- One answer: for total execution time, how much faster is B?

$$1001 / 110 = 9.1x$$

# 如何表示性能-总和？

	Machine A	Machine B
Program 1	1	10
Program 2	1000	100
Total	1001	110

- One answer: for total execution time, how much faster is B?

$$1001 / 110 = 9.1x \longleftarrow \text{right?}$$

# How to Average?

- Another: arithmetic mean (same result)
- Arithmetic mean of times:
- $AM(A) = 1001/2 = 500.5$
- $AM(B) = 110/2 = 55$
- Speedup:  $500.5/55 = 9.1x$

$$\left\{ \sum_{i=1}^n time(i) \right\} \times \frac{1}{n}$$

# How to Average?

- Another: arithmetic mean (same result)
- Arithmetic mean of times:
- $AM(A) = 1001/2 = 500.5$
- $AM(B) = 110/2 = 55$
- Speedup:  $500.5/55 = 9.1x$
- Valid only if programs run **equally often**, so use weighted arithmetic mean:

$$\left\{ \sum_{i=1}^n time(i) \right\} \times \frac{1}{n}$$

$$\left\{ \sum_{i=1}^n (weight(i) \times time(i)) \right\} \times \frac{1}{n}$$

# How to Average?

- Another: arithmetic mean (same result)
- Arithmetic mean of times:
- $AM(A) = 1001/2 = 500.5$
- $AM(B) = 110/2 = 55$
- Speedup:  $500.5/55 = 9.1x$
- Valid only if programs run **equally often**, so use weighted arithmetic mean:

$$\left\{ \sum_{i=1}^n time(i) \right\} \times \frac{1}{n}$$

$$\left\{ \sum_{i=1}^n (weight(i) \times time(i)) \right\} \times \frac{1}{n}$$

How to determine the weight?

# Problem of AM

- E.g., 30 mph for first 10 miles, then 90 mph for next 10 miles, what is average speed?
- Average speed =  $(30+90)/2$     ???

# Problem of AM

- E.g., 30 mph for first 10 miles, then 90 mph for next 10 miles, what is average speed?
- Average speed =  $(30+90)/2$  ← **WRONG!**

# Problem of AM

- E.g., 30 mph for first 10 miles, then 90 mph for next 10 miles, what is average speed?
- Average speed =  $(30+90)/2$
- Average speed = total distance / total time  
 $= (20 / (10/30 + 10/90))$   
 $= 45 \text{ mph}$   **Correct!**

# Harmonic Mean

- Harmonic mean of rates =

$$\frac{n}{\left\{ \sum_{i=1}^n \frac{1}{rate(n)} \right\}}$$

- Use HM if forced to start and end with rates (e.g. reporting MIPS or MFLOPS)
- Why?
  - Rate has time in denominator
  - Mean should be proportional to inverse of sums of time (not sum of inverses)
  - See: J.E. Smith, “Characterizing computer performance with a single number,” CACM Volume 31 , Issue 10 (October 1988), pp. 1202-1206.

# Dealing with Ratios

	Machine A	Machine B
Program 1	1	10
Program 2	1000	100
Total	1001	110

- If we take ratios with respect to machine A

	Machine A	Machine B
Program 1	1	10
Program 2	1	0.1
Average	1	5.05

# Dealing with Ratios

- Avg. wrt. machine A: A is 1, 5.05
- If we take ratios with respect to machine B

	Machine A	Machine B
Program 1	0.1	1
Program 2	10	1
Average	5.05	1

# Dealing with Ratios

- Avg. wrt. machine A: A is 1, 5.05
- If we take ratios with respect to machine B

	Machine A	Machine B
Program 1	0.1	1
Program 2	10	1
Average	5.05	1

- Can't both be true!!!
- Don't use arithmetic mean on ratios!

# Geometric Mean

- Use geometric mean for ratios
- Geometric mean of ratios =

$$\sqrt[n]{\prod_{i=1}^n ratio(i)}$$

- Independent of reference machine
- In the example, GM for machine a is 1, for machine B is also 1
  - Normalized with respect to either machine

# But...

- GM of ratios is not proportional to total time
- AM in example says machine B is 9.1 times faster
- GM says they are equal
- If we took total execution time, A and B are equal only if
  - Program 1 is run 100 times more often than program 2
- Generally, GM will mispredict for three or more machines

## 总结二

- Use AM for times
- Use HM if forced to use rates
- Use GM if forced to use ratios
- Best of all, use unnormalized numbers to compute time

# 基准测试：SPEC2000

- System Performance Evaluation Cooperative
  - Formed in 80s to combat benchmarketing
  - SPEC89, SPEC92, SPEC95, SPEC2000, SPEC2006
  - Latest one is SPEC2017
- 12 integer and 14 floating-point programs
  - Sun Ultra-5 300MHz reference machine has score of 100
  - Report **GM of ratios** to reference machine

# Benchmarks: SPEC CINT2000

Benchmark	Description
164.gzip	Compression
175.vpr	FPGA place and route
176.gcc	C compiler
181.mcf	Combinatorial optimization
186.crafty	Chess
197.parser	Word processing, grammatical analysis
252.eon	Visualization (ray tracing)
253.perlbench	PERL script execution
254.gap	Group theory interpreter
255.vortex	Object-oriented database
256.bzip2	Compression
300.twolf	Place and route simulator

# Benchmarks: SPEC CFP2000

Benchmark	Description
168.wupwise	Physics/Quantum Chromodynamics
171.swim	Shallow water modeling
172.mgrid	Multi-grid solver: 3D potential field
173.aplu	Parabolic/elliptic PDE
177.mesa	3-D graphics library
178.galgel	Computational Fluid Dynamics
179.art	Image Recognition/Neural Networks
183.equake	Seismic Wave Propagation Simulation
187.facerec	Image processing: face recognition
188.ammp	Computational chemistry
189.lucas	Number theory/primality testing
191.fma3d	Finite-element Crash Simulation
200.sixtrack	High energy nuclear physics accelerator design
301.apsi	Meteorology: Pollutant distribution

# Benchmark Pitfalls

- Benchmark not representative
  - Your workload is I/O bound, SPEC is useless

# Benchmark Pitfalls

- Benchmark not representative
  - Your workload is I/O bound, SPEC is useless
- Benchmark is too old
  - Benchmarks age poorly; benchmarketing pressure causes vendors to optimize compiler, hardware, software to match benchmarks
  - Need to be periodically refreshed

# Amdahl's Law

- Motivation for optimizing common case
- Speedup = old time / new time = new rate / old rate
- Let an optimization speed fraction  $f$  of time by a factor of  $s$

$$\text{New\_time} = (1-f) \times \text{old\_time} + (f/s) \times \text{old\_time}$$

$$\text{Speedup} = \text{old\_time} / \text{new\_time}$$

$$\text{Speedup} = \text{old\_time} / ((1-f) \times \text{old\_time} + (f/s) \times \text{old\_time})$$

$$\begin{aligned}\text{Speedup} &= \frac{[(1-f)+f] \times \text{oldtime}}{[(1-f) \times \text{oldtime}] + \frac{f}{s} \times \text{oldtime}} \\ &= \frac{1}{1-f + \frac{f}{s}}\end{aligned}$$

# Amdahl's Law Example

- Your boss asks you to improve performance by:
  - Improve the ALU used 95% of time by 10%
  - Improve memory pipeline used 5% of time by 10x

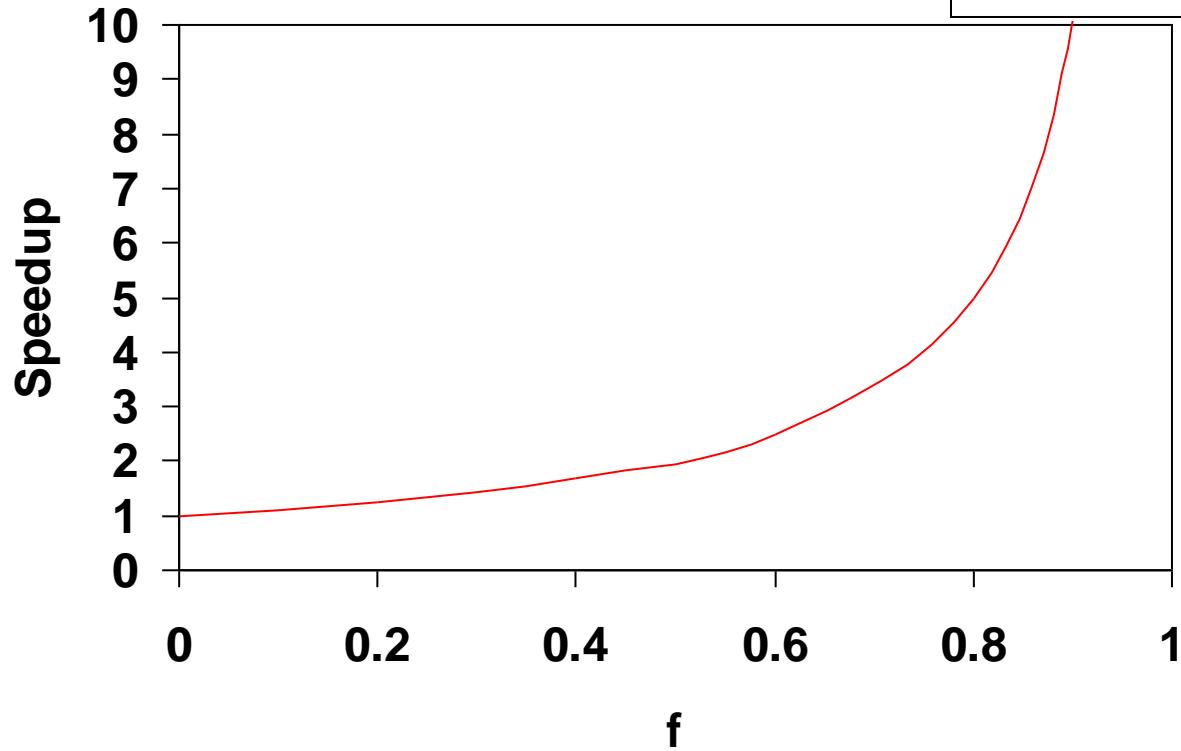
f	s	Speedup
95%	1.10	1.094
5%	10	1.047
5%	$\infty$	1.052

$$\text{Speedup} = \frac{1}{1 - f + \frac{f}{s}}$$

# Amdahl's Law: Limit

- Make common case fast:

$$\lim_{s \rightarrow \infty} \frac{1}{1 - f + \frac{f}{s}} = \frac{1}{1 - f}$$



# Amdahl's Law: Limit

- Consider uncommon case!
- If  $(1-f)$  is nontrivial
  - Speedup is limited!
- Particularly true for exploiting parallelism in the large, where large  $s$  is not cheap
  - GPU with e.g. 1024 processors (shader cores)
  - Parallel portion speeds up by  $s$  (1024x)
  - Serial portion of code  $(1-f)$  limits speedup

$$\lim_{s \rightarrow \infty} \frac{1}{1 - f + \frac{f}{s}} = \frac{1}{1 - f}$$

E.g. 10% serial portion:  $1/0.1 = 10x$  speedup with 1000 cores

# 总结三

- Benchmarks: SPEC2000
- Summarize performance:
  - AM for time
  - HM for rate
  - GM for ratio
- Amdahl's Law:

$$Speedup = \frac{1}{1-f + \frac{f}{S}}$$