

计算机组成与系统结构

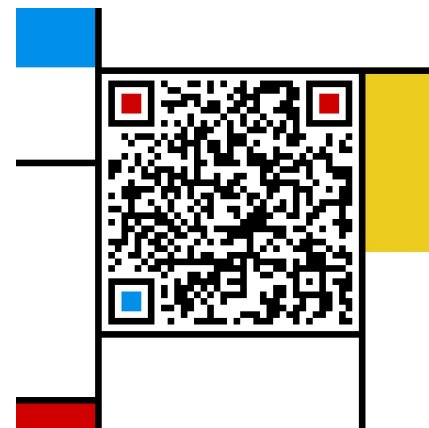
Computer Organization & System Architecture

Huang Kejie (黄科杰) 百人计划研究员

Office: 玉泉校区老生仪楼 304

Email address: huangkejie@zju.edu.cn

HP: 17706443800



Sub-Matrix Multiplication or: Beating Amdahl's Law

- Blocking:
 - Rearrange code to use values loaded in cache many times
 - Only “few” accesses to slow main memory (DRAM) per floating point operation
 - → throughput limited by FP hardware and cache, not slow DRAM
 - P&H, RISC-V edition p. 465

Memory Access Blocking

```
// Cache blocking; P&H p. 556
const int BLOCKSIZE = 32;

void do_block(int n, int si, int sj, int sk, double *A, double *B, double *C) {
    for (int i=si; i<si+BLOCKSIZE; i+=UNROLL*4)
        for (int j=sj; j<sj+BLOCKSIZE; j++) {
            __m256d c[4];
            for (int x=0; x<UNROLL; x++)
                c[x] = _mm256_load_pd(C+i+x*4+j*n);
            for (int k=sk; k<sk+BLOCKSIZE; k++) {
                __m256d b = _mm256_broadcast_sd(B+k+j*n);
                for (int x=0; x<UNROLL; x++)
                    c[x] = _mm256_add_pd(c[x],
                                         _mm256_mul_pd(_mm256_load_pd(A+n*k+x*4+i), b));
            }
            for (int x=0; x<UNROLL; x++)
                _mm256_store_pd(C+i+x*4+j*n, c[x]);
        }
}

void dgemm_block(int n, double* A, double* B, double* C) {
    for(int sj=0; sj<n; sj+=BLOCKSIZE)
        for(int si=0; si<n; si+=BLOCKSIZE)
            for (int sk=0; sk<n; sk += BLOCKSIZE)
                do_block(n, si, sj, sk, A, B, C);
}
```

Performance

N	Gflops			
	scalar	avx	unroll	blocking
32	1.30	4.56	12.95	13.80
160	1.30	5.47	19.70	21.79
480	1.32	5.27	14.50	20.17
960	0.91	3.64	6.91	15.82

Improving Performance

- Increase clock rate fs
 - Reached practical maximum for today's technology
 - < 5GHz for general purpose computers
- Lower CPI (cycles per instruction)
 - SIMD, “instruction level parallelism”
- Perform multiple tasks simultaneously
 - Multiple CPUs, each executing different program
 - Tasks may be related
 - E.g. each CPU performs part of a big matrix multiplication
 - or unrelated
 - E.g. distribute different web http requests over different computers
 - E.g. run pptx (view lecture slides) and browser (youtube) simultaneously
- Do all of the above:
 - High fs, SIMD, multiple parallel tasks

Today's
lecture

New-School Machine Structures (It's a bit more complicated!)

- Parallel Requests
Assigned to computer
e.g., Search “Katz”

- Parallel Threads
Assigned to core
e.g., Lookup, Ads

- Parallel Instructions
>1 instruction @ one time
e.g., 5 pipelined instructions

- Parallel Data
>1 data item @ one time
e.g., Add of 4 pairs of words

- Hardware descriptions
All gates @ one time

- Programming Languages

Software

Hardware

Warehouse
Scale
Computer

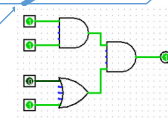
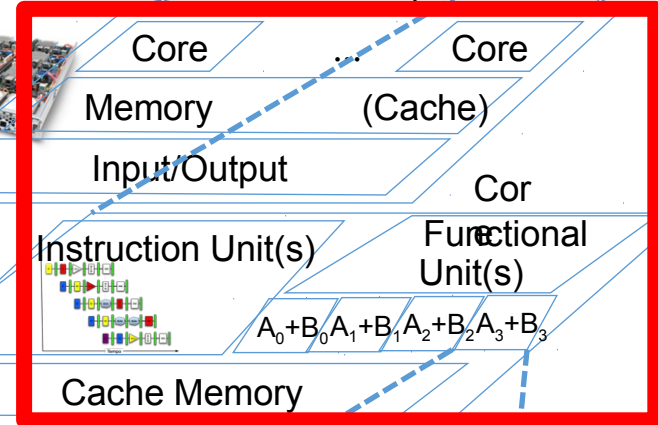
Smart
Phone

*Harness
Parallelism &
Achieve High
Performance*



Computer

Today's
Lecture



Logic Gates

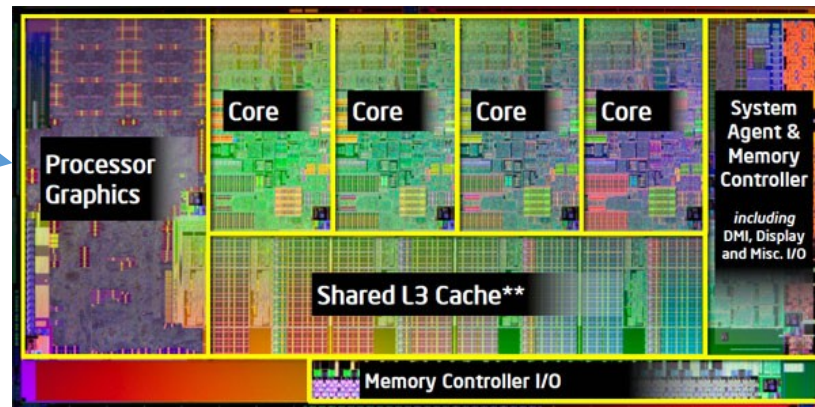
Parallel Computer Architectures



Several separate computers,
some means for communication
(e.g., Ethernet)

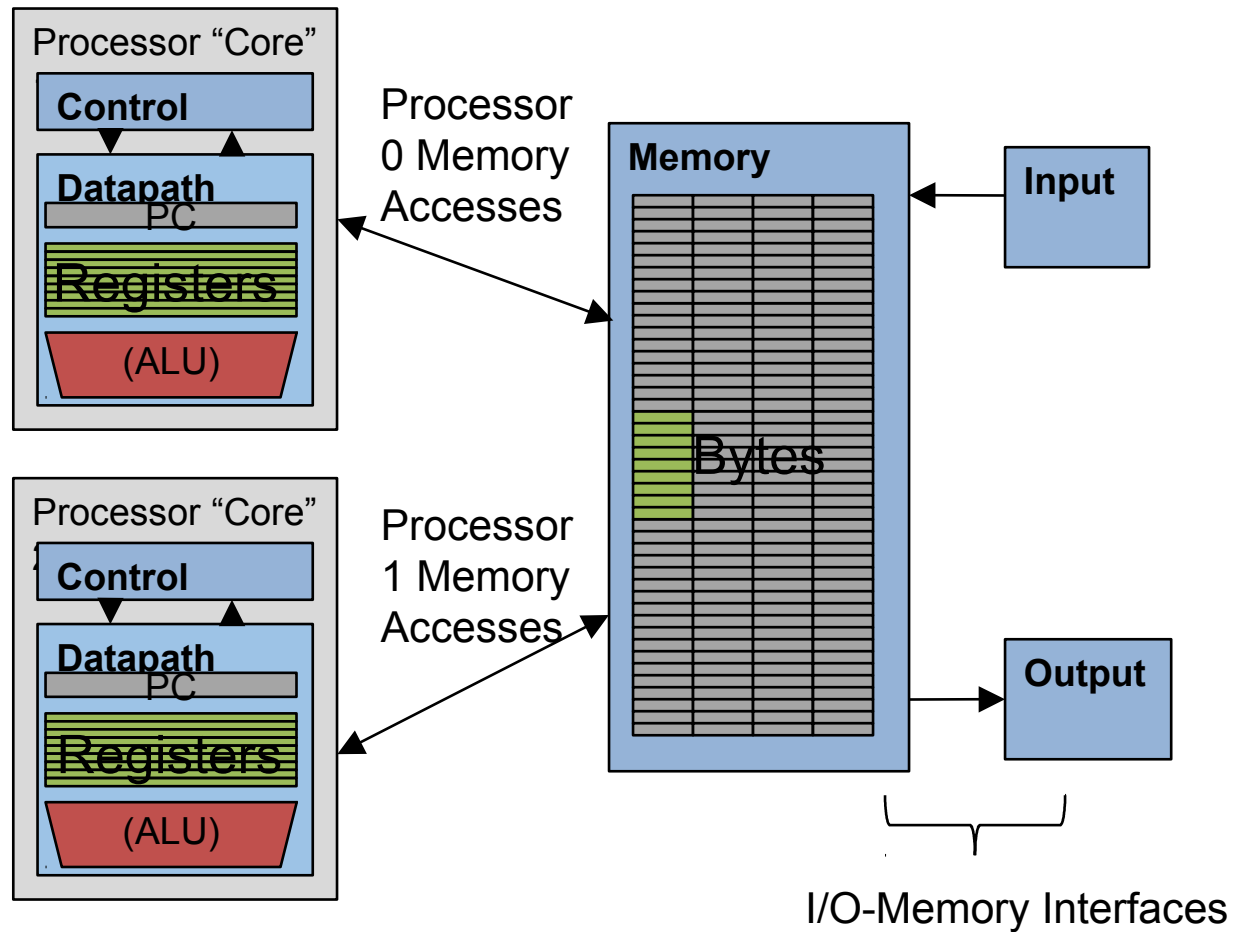
GPU “graphics processing unit” →

Multi-core CPU:
1 datapath in single chip
share L3 cache, memory, peripherals
Example: Hive machines



Massive array of computers,
fast communication between processors

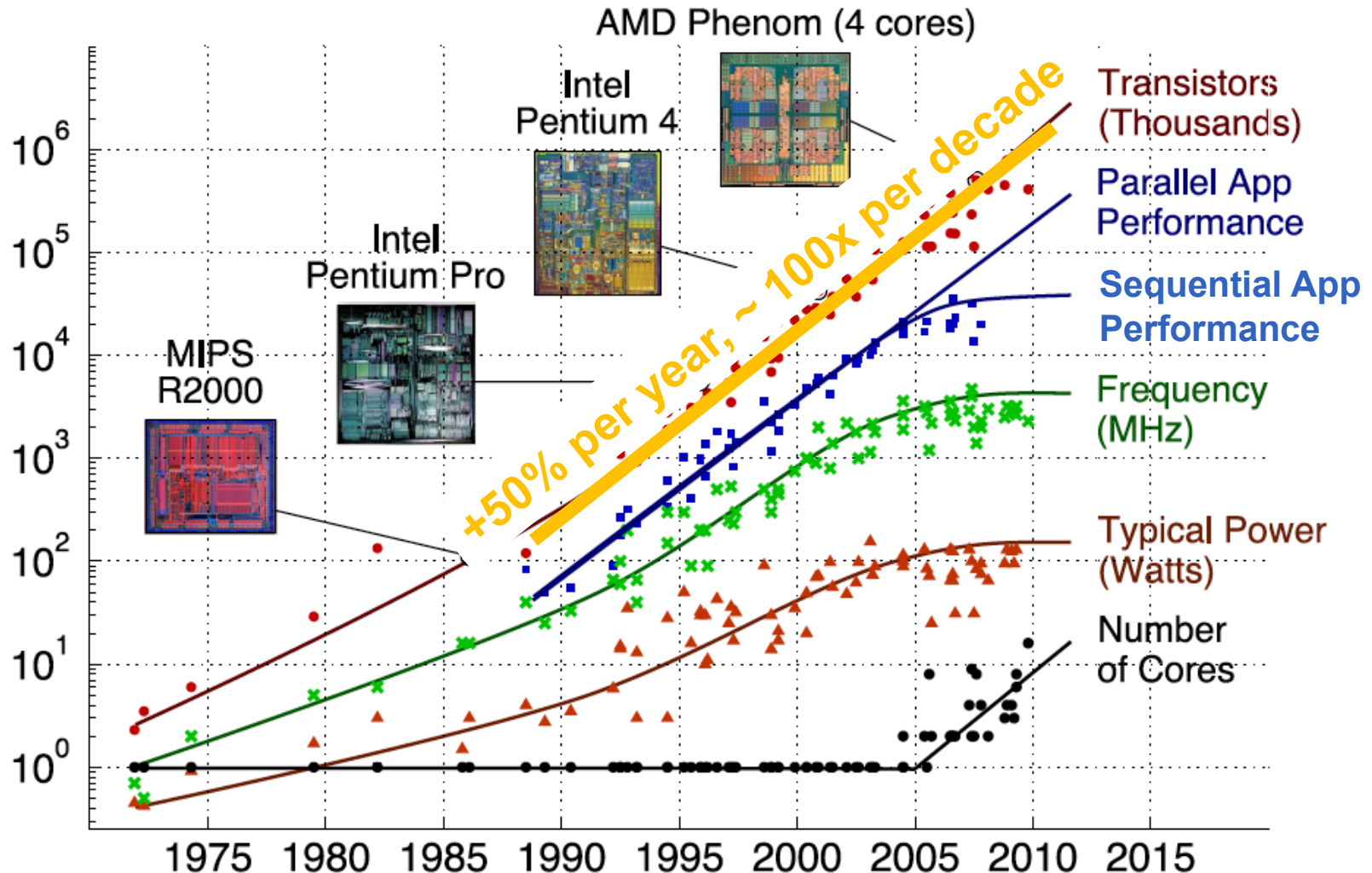
Example: CPU with Two Cores



Multiprocessor Execution Model



- Each processor (core) executes its own instructions
- Separate resources (not shared)
 - Datapath (PC, registers, ALU)
 - Highest level caches (e.g., 1st and 2nd)
- Shared resources
 - Memory (DRAM)
 - Often 3rd level cache
 - Often on same silicon chip
 - But not a requirement
- Nomenclature
 - “Multiprocessor Microprocessor”
 - Multicore processor
 - E.g., four core CPU (central processing unit)
 - Executes four different instruction streams simultaneously

Transition to Multicore



Data partially collected by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond

Pixel 2 vs. iPhone 8

		
	Pixel 2	iPhone 8
Size	5.74 x 2.74 x 0.31 inches	5.45 x 2.65 x 0.29 inches
Weight	143 grams (5.04 ounces)	148 grams (5.22 ounces)
Screen	5-inch AMOLED display	4.7-inch Retina HD LCD-backlit widescreen
Resolution	1,920 x 1,080 pixels (441 ppi)	1,334 x 750 pixels (326 pixels-per-inch)
OS	Android 8.0	iOS 11
Storage	64GB, 128GB	64GB, 256GB
MicroSD card slot	No	No
NFC support	Yes	Yes, Apple Pay only
Processor	Snapdragon 835, with Adreno 540	A11 Bionic with 64-bit architecture, M11 motion coprocessor
RAM	4GB	2GB
Connectivity	GSM, CDMA, HSPA, EVDO, LTE, 802.11a/b/g/n/ac Wi-Fi	4G LTE, GSM, CDMA, HSPA+, 802.11a/b/g/n/ac Wi-Fi
Camera	12.2 MP rear, 8 MP HD front	12.2 MP rear, 8 MP HD front
Video	Up to 4K at 30fps, 1080p at 120fps, 720p at 240fps	Up to 4K at 60fps, 1080p at 240fps

Pixel 2 vs. iPhone 8

Qualcomm Technologies leading with 10nm

Kryo 280 [\[edit\]](#)

A new generation of this microarchitecture, named Kryo 280, was announced along with the Snapdragon 835 chipset in November 2016.^[3] The Kryo 280 CPU core is not a derivative of the original Kryo, but rather is a customized derivative of the ARM's Cortex-A73.^[4] The new core improves integer instructions per clock (+17%), while having much lower (~32%) performance at floating point math relative to the original Kryo.^[4]

Overview [\[edit\]](#)

- 32 KiB + 32 KiB L1 cache
- 2 MiB L2 cache (performance cluster only)
- Core performance: ~6.35 DMIPS/MHz

2.35Ghz + 1.9Ghz, 64Bit Octa-Core

More feature integration											
Smaller die size & package											
Adreno 540	256 ALUs	10 nm	710 MHz	?	?	?	567 ^[21] GFlops				Snapdragon 835 (MSM8998)
Larger battery											
Lower power, better battery life											
Snapdragon 820						Snapdragon 835					
*Compared to Snapdragon 820											










Pixel 2 vs. iPhone 8










Design [\[edit \]](#)

The A11 features an Apple-designed 64-bit ARMv8-A **six-core CPU**, with two high-performance cores, called **Monsoon**, and four energy-efficient cores, called **Mistral**.^{[1][5][3]} The A11 uses a new second-generation performance controller, which permits the A11 to use all six cores simultaneously,^[7] unlike its predecessor the A10. The A11 also integrates an Apple-designed three-core **graphics processing unit** (GPU) with 30% faster graphics performance than the A10.^[5] Embedded in the A11 is the M11 **motion coprocessor**.^[8] The A11 includes a new **image processor** which supports **computational photography** functions such as lighting estimation, wide color capture, and advanced pixel processing.^[5] The A11 also includes **dedicated neural network hardware** that Apple calls a **"Neural Engine"**, which can perform up to 600 billion operations per second, used for **Face ID**, **Animoji** and other **machine learning** tasks.^[7]

The A11 is manufactured by **TSMC** using a **10 nm FinFET** process^[1] and contains 4.3 billion transistors^[6] on a die 87.66 mm² in size, 41% smaller than the A10.^[9] It is manufactured in a **package on package** (PoP) together with 2 GB of **LPDDR4X** memory in the **iPhone 8**^[2] and 3 GB of LPDDR4X memory in the **iPhone 8 Plus** and **iPhone X**.^{[9][10]}

Pixel 2 vs. iPhone 8

Geekbench Browser		
Geekbench 4 ▾ Geekbench 3 ▾		
Single Core Multi-Core RenderScript		
Device	Score	
 Samsung Galaxy Note 8 Samsung Exynos 8895 Octa @ 1.7 GHz	6479	
 Samsung Galaxy S8 Samsung Exynos 8895 Octa @ 1.7 GHz	6437	
 Samsung Galaxy S8+ Samsung Exynos 8895 Octa @ 1.7 GHz	6406	
 Samsung Galaxy Note 8 Qualcomm MSM8998 Snapdragon 835 @ 1.9 GHz	6306	
 Xiaomi MI 6 Qualcomm MSM8998 Snapdragon 835 @ 1.9 GHz	6053	
 Samsung Galaxy S8+ Qualcomm MSM8998 Snapdragon 835 @ 1.9 GHz	6045	
 Huawei Honor V9 Hisilicon Kirin 960 @ 1.8 GHz	6024	
 Huawei P10 Hisilicon Kirin 960 @ 1.8 GHz	6000	
 Samsung Galaxy S8 Qualcomm MSM8998 Snapdragon 835 @ 1.9 GHz	5983	

Geekbench Browser		
Geekbench 4 ▾ Geekbench 3 ▾		
Single Core Multi-Core Metal		
Device	Score	
 iPhone 8 Plus Apple A11 Bionic @ 2.4 GHz	10122	
 iPhone 8 Apple A11 Bionic @ 2.4 GHz	10030	
 iPhone X Apple A11 Bionic @ 2.4 GHz	9939	
 iPad Pro (10.5-inch) Apple A10X Fusion @ 2.3 GHz	9249	
 iPad Pro (12.9-inch 2nd Generation) Apple A10X Fusion @ 2.3 GHz	9236	
 iPhone 7 Apple A10 Fusion @ 2.3 GHz	5753	
 iPhone 7 Plus Apple A10 Fusion @ 2.3 GHz	5747	
 iPad Pro (12.9-inch) Apple A9X @ 2.3 GHz	5017	
 iPad Pro (9.7-inch) Apple A9X @ 2.3 GHz	4905	

Multiprocessor Execution Model

- Shared memory
 - Each “core” has access to the entire memory in the processor
 - Special hardware keeps caches consistent
 - Advantages:
 - Simplifies communication in program via shared variables
 - Drawbacks:
 - Does not scale well:
 - “Slow” memory shared by many “customers” (cores)
 - May become bottleneck (Amdahl’s Law)
- Two ways to use a multiprocessor:
 - Job-level parallelism
 - Processors work on unrelated problems
 - No communication between programs
 - Partition work of single task between several cores
 - E.g., each performs part of large matrix multiplication

Parallel Processing

- It's difficult!
- It's inevitable
 - Only path to increase performance
 - Only path to lower energy consumption (improve battery life)
- In mobile systems (e.g., smart phones, tablets)
 - Multiple cores
 - Dedicated processors, e.g.,
 - Motion processor, image processor, neural processor in iPhone 8 + X
 - GPU (graphics processing unit)
- Warehouse-scale computers
 - Multiple “nodes”
 - “Boxes” with several CPUs, disks per box
 - MIMD (multi-core) and SIMD (e.g. AVX) in each node

Potential Parallel Performance (assuming software can use it)

Year	Cores		SIMD bits /Core		Core * SIMD bits	Total, e.g. FLOPs/Cycle
2003	MIMD	2	SIMD	128	256	MIMD 4
2005	+2/	4	2X/	128	512	& SIMD 8
2007	2yrs	6	4yrs	128	768	12
2009		8		128	1024	16
2011		10		256	2560	40
2013	2.5X	12	8X	256	3072	48
2015		14		512	7168	112
2017		16		512	8192	128
2019		18		1024	18432	288
2021		20		1024	20480	320

**12
years**

20 x in 12 years
 $20^{1/12} = 1.28 \times \rightarrow$ **28% per year or 2x every 3 years!**
 IF (!) we can use it

Programs Running on my Computer

```
PID TTY      TIME CMD
220 ??      0:04.34 /usr/libexec/UserEventAgent (Aqua)
222 ??      0:10.60 /usr/sbin/distnoted agent
224 ??      0:09.11 /usr/sbin/cfprefsd agent
229 ??      0:04.71 /usr/sbin/usernoted
230 ??      0:02.35 /usr/libexec/nsurlsessiond
232 ??      0:28.68 /System/Library/PrivateFrameworks/CalendarAgent.framework/Executables/CalendarAgent
234 ??      0:04.36 /System/Library/PrivateFrameworks/GameCenterFoundation.framework/Versions/A/gamed
235 ??      0:01.90 /System/Library/CoreServices/cloudphotosd.app/Contents/MacOS/cloudphotosd
236 ??      0:49.72 /usr/libexec/secinitd
239 ??      0:01.66 /System/Library/PrivateFrameworks/TCC.framework/Resources/tccd
240 ??      0:12.68 /System/Library/Frameworks/Accounts.framework/Versions/A/Support/accountsd
241 ??      0:09.56 /usr/libexec/SafariCloudHistoryPushAgent
242 ??      0:00.27 /System/Library/PrivateFrameworks/CallHistory.framework/Support/CallHistorySyncHelper
243 ??      0:00.74 /System/Library/CoreServices/mapspushd
244 ??      0:00.79 /usr/libexec/fmfd
246 ??      0:00.09 /System/Library/PrivateFrameworks/AskPermission.framework/Versions/A/Resources/askpermissiond
248 ??      0:01.03 /System/Library/PrivateFrameworks/CloudDocsDaemon.framework/Versions/A/Support/bird
249 ??      0:02.50 /System/Library/PrivateFrameworks/IDS.framework/identityservicesd.app/Contents/MacOS/identityservicesd
250 ??      0:04.81 /usr/libexec/secd
254 ??      0:24.01 /System/Library/PrivateFrameworks/CloudKitDaemon.framework/Support/cloudkd
258 ??      0:04.73 /System/Library/PrivateFrameworks/TelephonyUtilities.framework/callservicesd
267 ??      0:02.15 /System/Library/CoreServices/AirPlayUIAgent.app/Contents/MacOS/AirPlayUIAgent --launchd
271 ??      0:03.91 /usr/libexec/nsurlstoraged
274 ??      0:00.90 /System/Library/PrivateFrameworks/CommerceKit.framework/Versions/A/Resources/storeaccountd
282 ??      0:00.09 /usr/sbin/pboard
283 ??      0:00.90 /System/Library/PrivateFrameworks/InternetAccounts.framework/Versions/A/XPCServices/com.apple.internetaccounts.xpc/Contents/MacOS/com.apple.internetaccounts
285 ??      0:04.72 /System/Library/Frameworks/ApplicationServices.framework/Frameworks/ATS.framework/Support/Fontd
291 ??      0:00.25 /System/Library/Frameworks/Security.framework/Versions/A/Resources/CloudKeychainProxy.bundle/Contents/MacOS/CloudKeychainProxy
292 ??      0:09.54 /System/Library/CoreServices/CoreServicesUIAgent.app/Contents/MacOS/CoreServicesUIAgent
293 ??      0:00.29 /System/Library/PrivateFrameworks/CloudPhotoServices.framework/Versions/A/Frameworks/CloudPhotoServicesConfiguration.framework/Versions/A/XPCServices/com.apple.CloudPhotosConfiguration.xpc/Contents/MacOS/
com.apple.CloudPhotosConfiguration
297 ??      0:00.84 /System/Library/PrivateFrameworks/CloudServices.framework/Resources/com.apple.sbd
302 ??      0:26.11 /System/Library/CoreServices/Dock.app/Contents/MacOS/Dock
303 ??      0:09.55 /System/Library/CoreServices/SystemUIServer.app/Contents/MacOS/SystemUIServer
```

... 156 total at this moment

How does my laptop do this?

Imagine doing 156 assignments all at the same time!

Threads

- Sequential flow of instructions that performs some task
 - Up to now we just called this a “program”
- Each thread has:
 - Dedicated PC (program counter)
 - Separate registers
 - Accesses the shared memory
- Each physical core provides one (or more)
 - Hardware threads that actively execute instructions
 - Each executes one “hardware thread”
- Operating system multiplexes multiple
 - Software threads onto the available hardware threads
 - All threads except those mapped to hardware threads are waiting

Operating System Threads

Give illusion of many “simultaneously” active threads

- Multiplex software threads onto hardware threads:
 - Switch out blocked threads (e.g., cache miss, user input, network access)
 - Timer (e.g., switch active thread every 1 ms)
- Remove a software thread from a hardware thread by
 - Interrupting its execution
 - Saving its registers and PC to memory
- Start executing a different software thread by
 - Loading its previously saved registers into a hardware thread's registers
 - Jumping to its saved PC

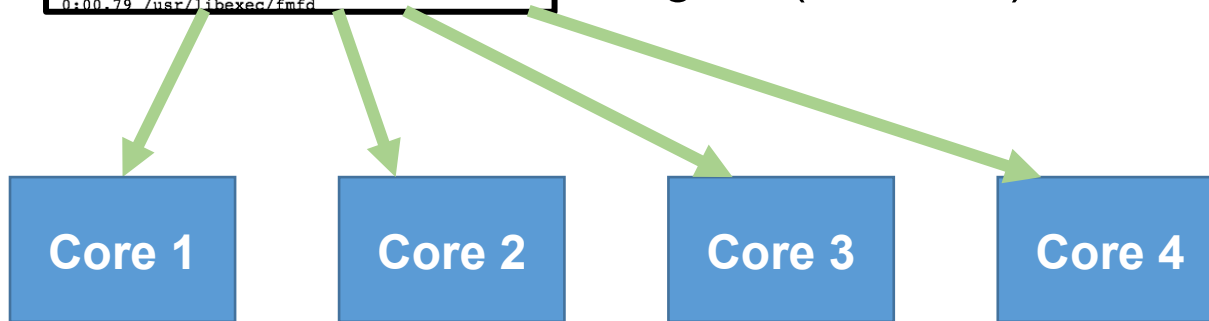
Example: Four Cores

```
0:04.34 /usr/libexec/UserEventAgent (Aqua)
0:10.60 /usr/sbin/distnoted agent
0:09.11 /usr/sbin/cfprefsd agent
0:04.71 /usr/sbin/usernoted
0:02.35 /usr/libexec/nsurlsessiond
0:28.68 /System/Library/PrivateFrameworks/Calend
0:04.36 /System/Library/PrivateFrameworks/GameCe
0:01.90 /System/Library/CoreServices/cloudphotos
0:49.72 /usr/libexec/secinitd
0:01.66 /System/Library/PrivateFrameworks/TCC.fr
0:12.68 /System/Library/Frameworks/Accounts.fram
0:09.56 /usr/libexec/SafariCloudHistoryPushAgent
0:00.27 /System/Library/PrivateFrameworks/CallHi
0:00.74 /System/Library/CoreServices/mapspushd
0:00.79 /usr/libexec/fmfd
```

Thread pool:

List of threads competing for processor

OS maps threads to cores and schedules logical (software) threads



Each “Core” actively runs one instruction stream at a time

Multithreading

- Typical scenario:
 - Active thread encounters cache miss
 - Active thread waits ~ 1000 cycles for data from DRAM
 - \rightarrow switch out and run different thread until data available
- Problem
 - Must save current thread state and load new thread state
 - PC, all registers (could be many, e.g. AVX)
 - \rightarrow must perform switch in $\ll 1000$ cycles
- Can hardware help?
 - Moore's Law: transistors are plenty

Types of Parallelism in Applications

- Instruction-level parallelism (ILP)
 - Multiple instructions in execution at the same time, e.g., *instruction pipelining*
 - Generated and managed by hardware (superscalar) or by compiler (VLIW)
 - Limited in practice by data and control dependences, i.e., pipeline hazards
- Thread-level or task-level parallelism (TLP)
 - Multiple threads or instruction sequences from the same application can be executed concurrently
 - **Thread**: sequence of instructions, with own program counter and processor state (e.g., register file)
 - **Multicore**:
 - **Physical CPU**: One thread (at a time) per CPU, in software OS switches threads typically in response to I/O events like disk read/write
 - **Logical CPU**: Fine-grain thread switching, in hardware, when thread blocks due to cache miss/memory access
 - **Hyperthreading** (aka **Simultaneous Multithreading**--SMT): Exploit superscalar architecture to launch instructions from different threads at the same time!
 - Limited in practice by communication/synchronization overheads and by algorithm characteristics

Types of Parallelism in Applications

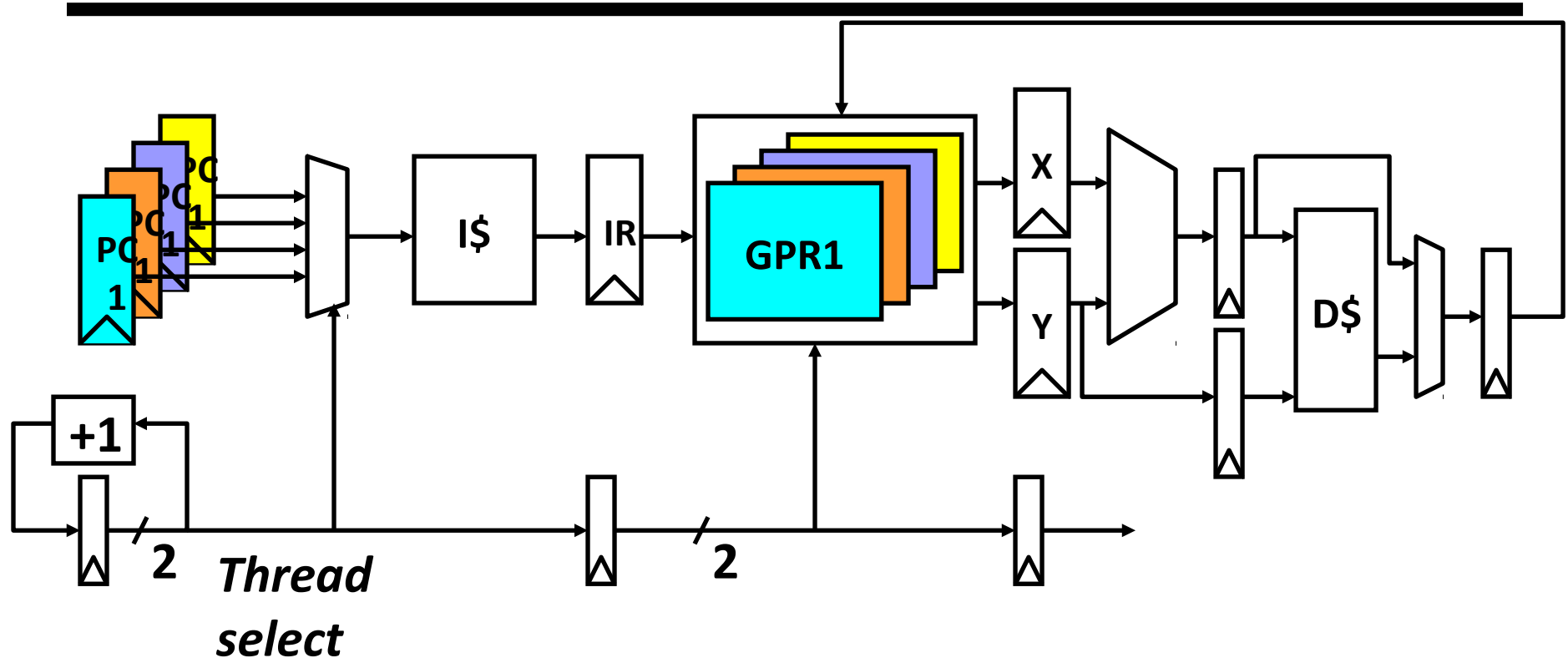
- Data-level parallelism (DLP)
 - Instructions from a single stream operate concurrently on several data
 - Limited by non-regular data manipulation patterns and by memory bandwidth
- Transaction-level parallelism
 - Multiple threads/processes from different transactions can be executed concurrently
 - Limited by concurrency overheads

CDC 6600 Peripheral Processors (Cray, 1964)



- First multithreaded hardware
- 10 “virtual” I/O processors
- Fixed interleave on simple pipeline
- Pipeline has 100ns cycle time
- Each virtual processor executes one instruction every 1000ns
- Accumulator-based instruction set to reduce processor state

Simple Multithreaded Pipeline



- Have to carry thread select down pipeline to ensure correct state bits read/written at each pipe stage
- Appears to software (including OS) as multiple, albeit slower, CPUs

Multithreading Costs

- Each thread requires its own user state
 - PC
 - GPRs
- Also, needs its own system state
 - Virtual-memory page-table-base register
 - Exception-handling registers
- Other overheads:
 - Additional cache/TLB conflicts from competing threads
 - (or add larger cache/TLB capacity)
 - More OS overhead to schedule more threads (where do all these threads come from?)

Thread Scheduling Policies

- Fixed interleave (CDC 6600 PPUs, 1964)
 - Each of N threads executes one instruction every N cycles
 - If thread not ready to go in its slot, insert pipeline bubble
- Software-controlled interleave (TI ASC PPUs, 1971)
 - OS allocates S pipeline slots amongst N threads
 - Hardware performs fixed interleave over S slots, executing whichever thread is in that slot



- Hardware-controlled thread scheduling (HEP, 1982)
 - Hardware keeps track of which threads are ready to go
 - Picks next thread to execute based on hardware priority scheme

Denelcor HEP (Burton Smith, 1982)

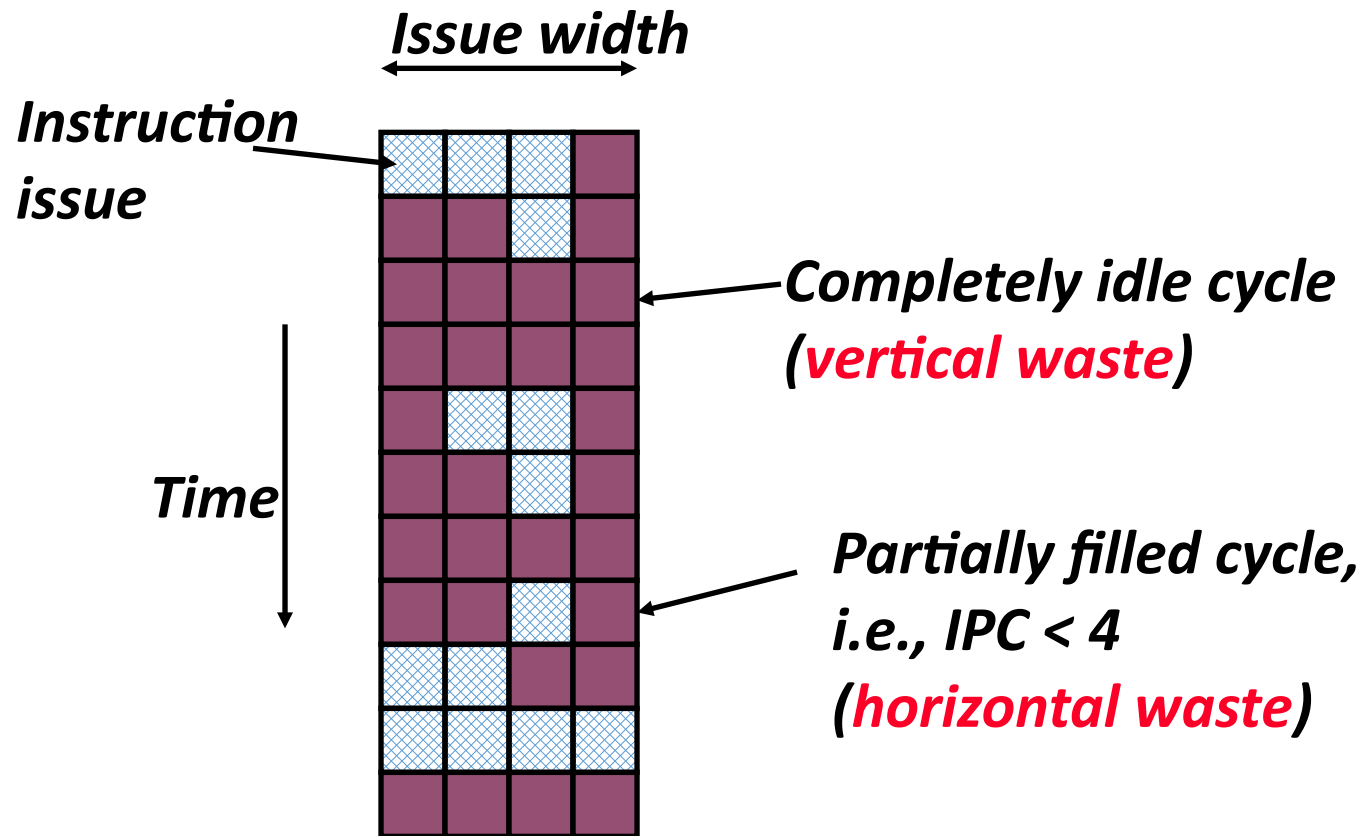


- First commercial machine to use hardware threading in main CPU
 - 120 threads per processor
 - 10 MHz clock rate
 - Up to 8 processors
 - precursor to Tera MTA (Multithreaded Architecture)

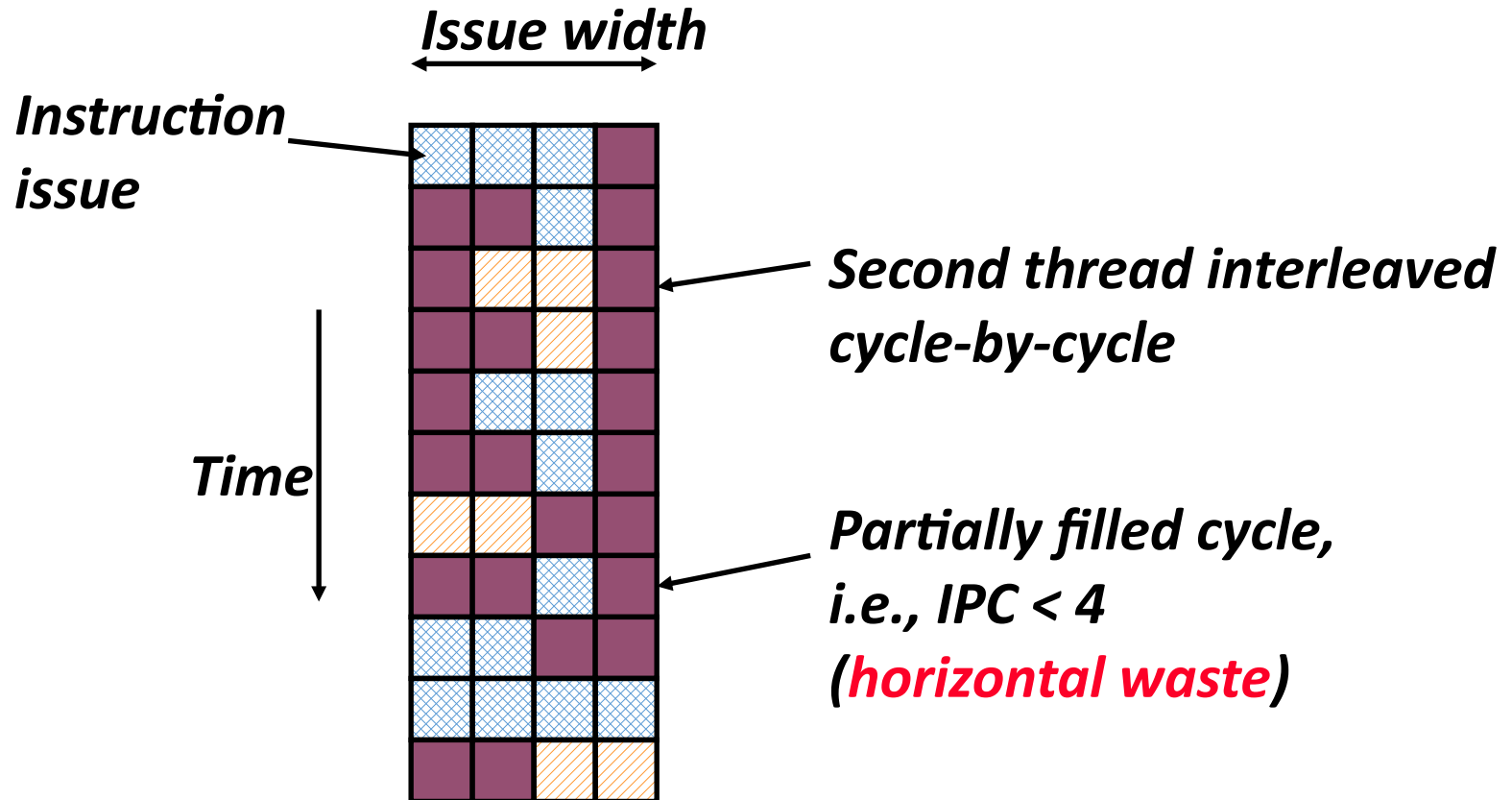
Simultaneous Multithreading (SMT) for OoO Superscalars

- Techniques presented so far have all been “vertical” multithreading where each pipeline stage works on one thread at a time
- SMT uses fine-grain control already present inside an OoO superscalar to allow instructions from multiple threads to enter execution on same clock cycle. Gives better utilization of machine resources.

Superscalar Machine Efficiency

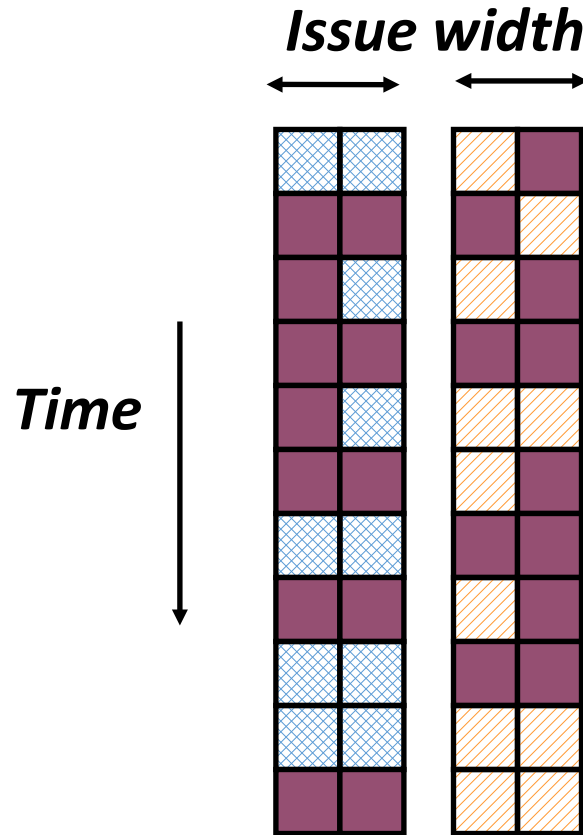


Vertical Multithreading



- Cycle-by-cycle interleaving removes vertical waste, but leaves some horizontal waste

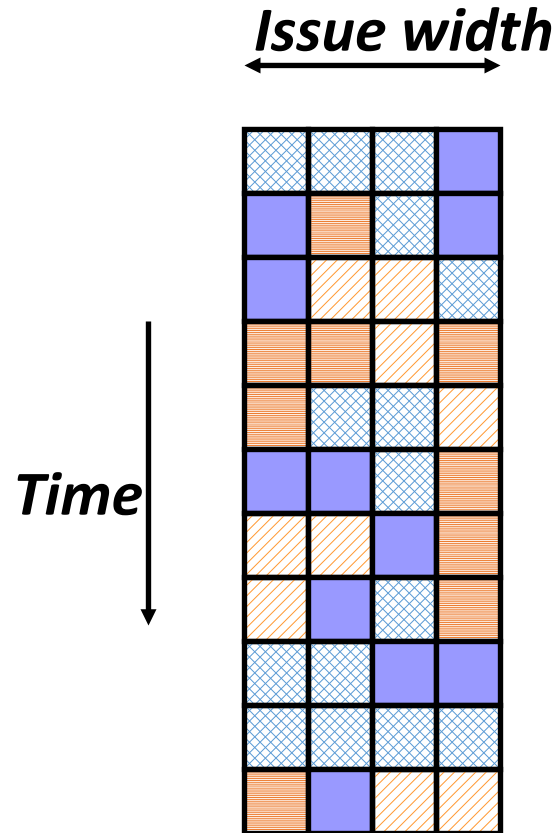
Chip Multiprocessing (CMP)



- What is the effect of splitting into multiple processors?
 - reduces horizontal waste,
 - leaves some vertical waste, and
 - puts upper limit on peak throughput of each thread.

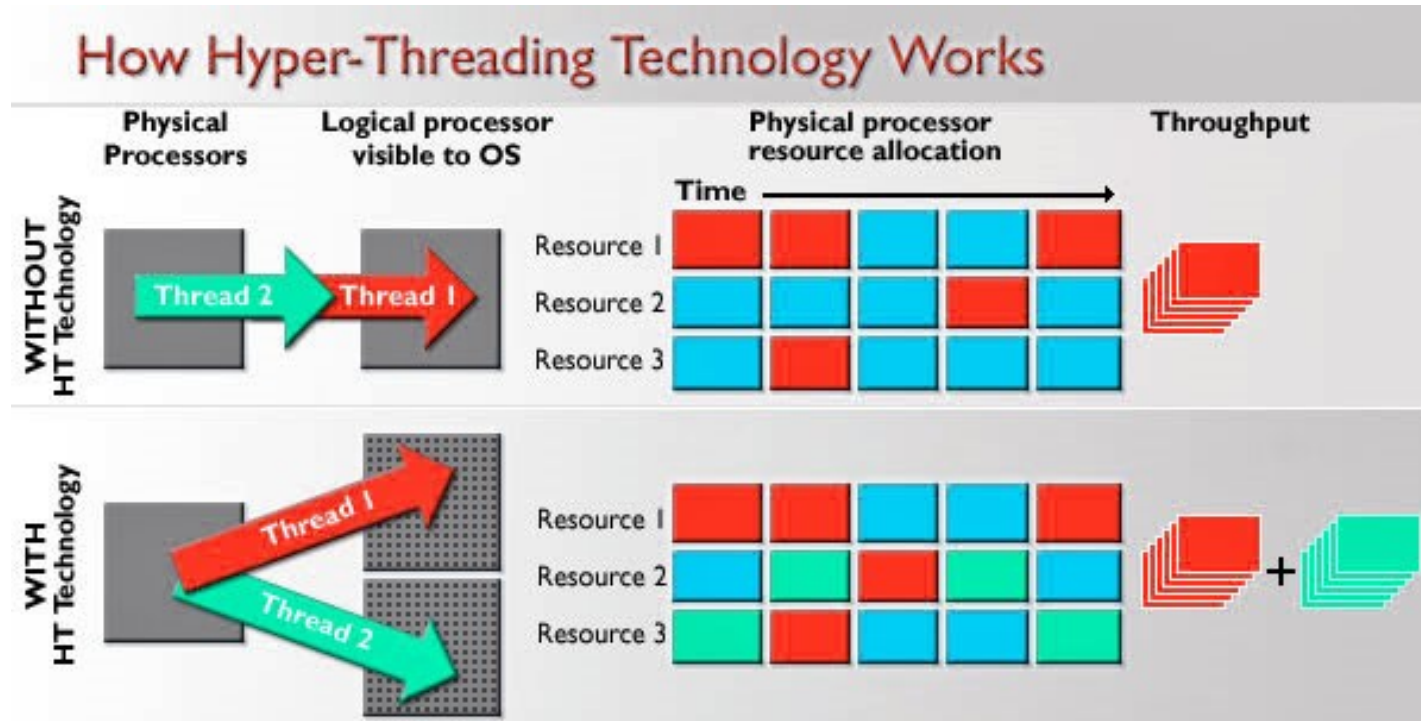
Ideal Superscalar Multithreading

[Tullsen, Eggers, Levy, UW, 1995]



- Interleave multiple threads to multiple issue slots with no restrictions

Hyperthreading



- SMT (HT): Logical CPUs > Physical CPUs
 - Run multiple threads at the same time per core
 - Each thread has own architectural state (PC, Registers, etc.)
 - Share resources (cache, instruction unit, execution units)
 - Improves Core CPI (clock ticks per instruction)
 - May degrade Thread CPI (Utilization/Bandwidth v. Latency)
 - See <http://dada.cs.washington.edu/smt/>

Multithreaded Categories

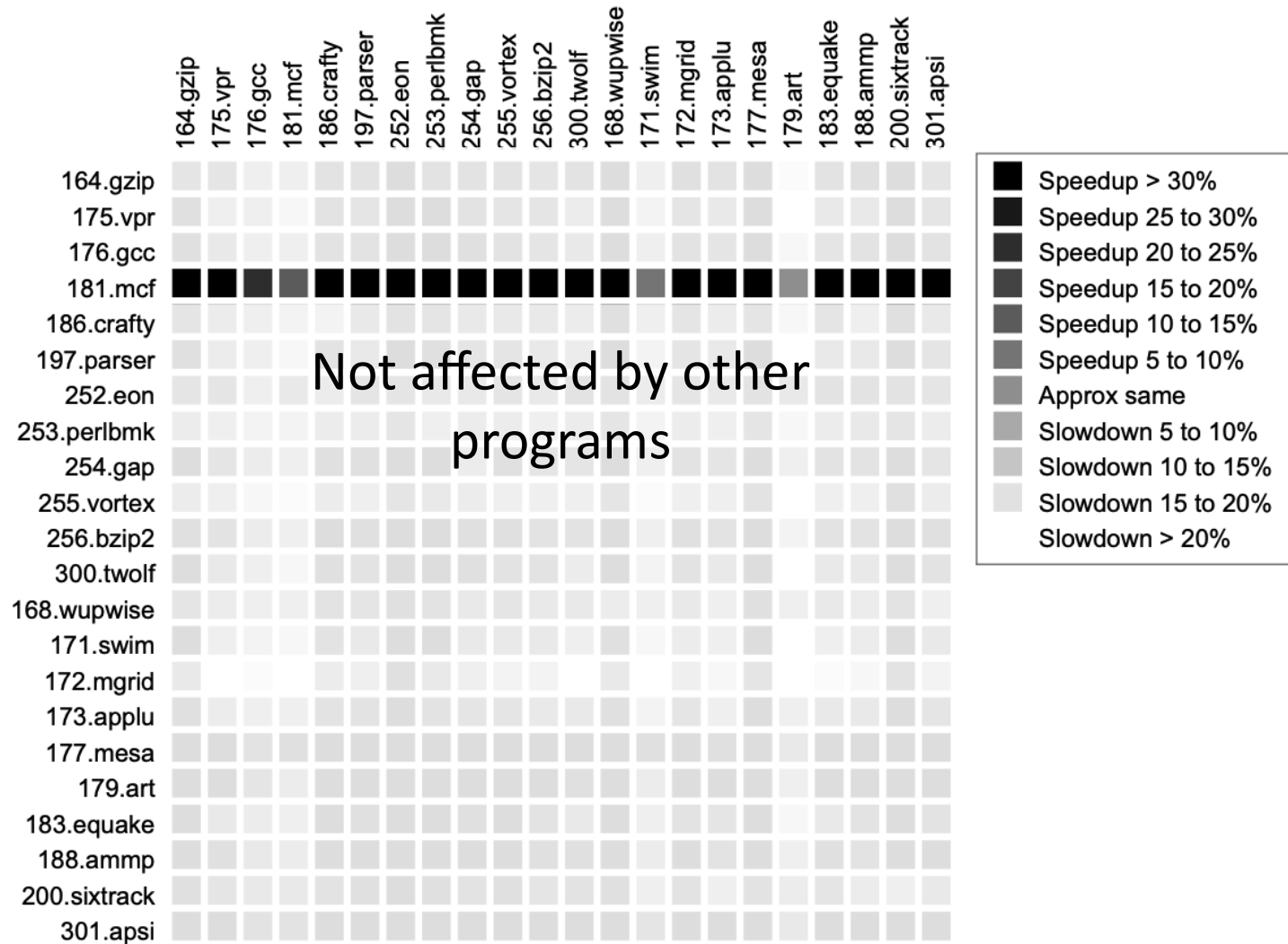
Summary: Multithreaded Categories



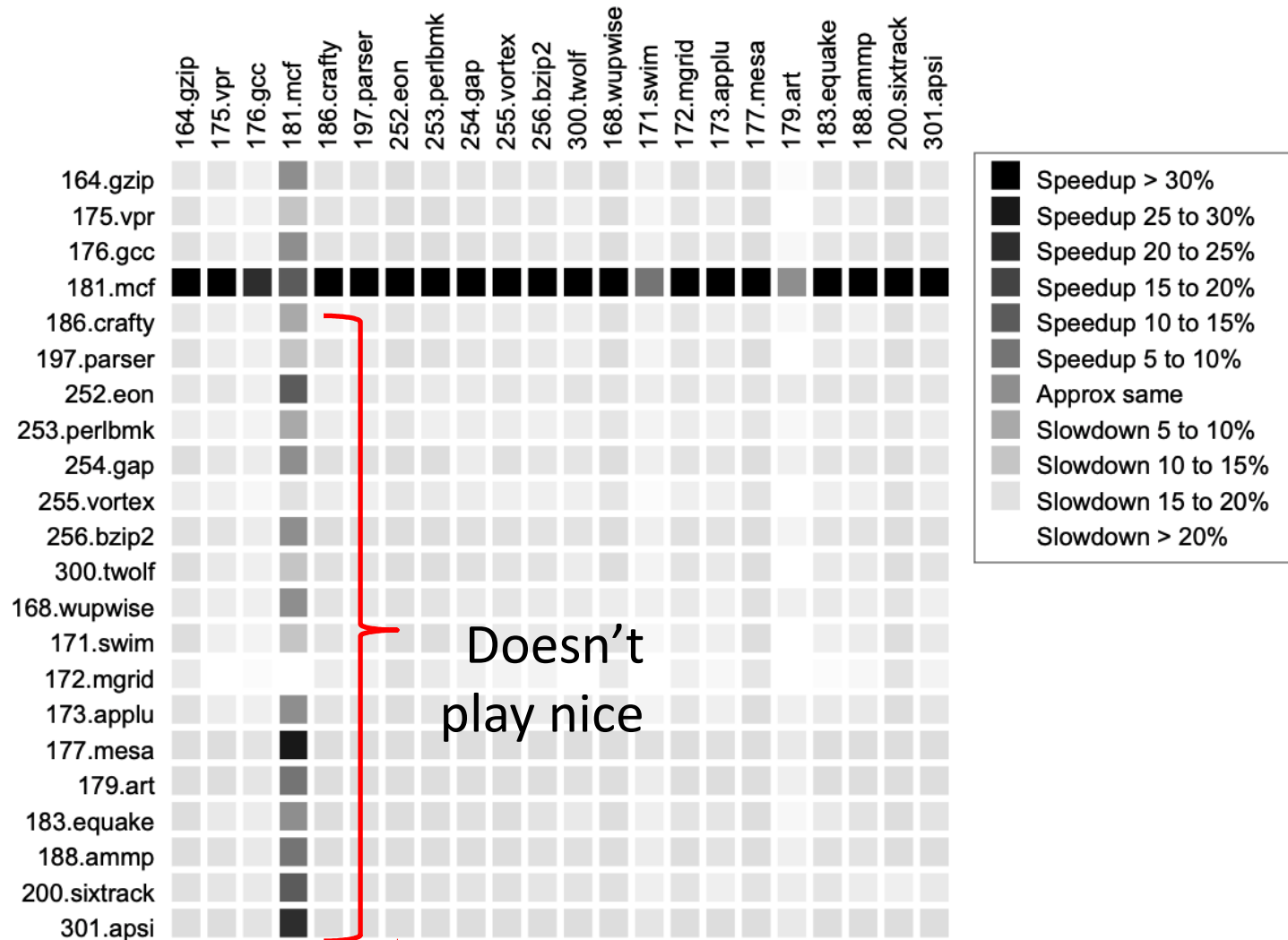
Initial Performance of SMT

- Pentium-4 Extreme SMT yields 1.01 speedup for SPECint_rate benchmark and 1.07 for SPECfp_rate
 - Pentium-4 is dual-threaded SMT
 - SPECRate requires that each SPEC benchmark be run against a vendor-selected number of copies of the same benchmark
- Running on Pentium-4 each of 26 SPEC benchmarks paired with every other (262 runs) speed-ups from 0.90 to 1.58; average was 1.20
- Power 5, 8-processor server 1.23 faster for SPECint_rate with SMT, 1.16 faster for SPECfp_rate
- Power 5 running 2 copies of each app speedup between 0.89 and 1.41
 - Most gained some
 - Fl.Pt. apps had most cache conflicts and least gains

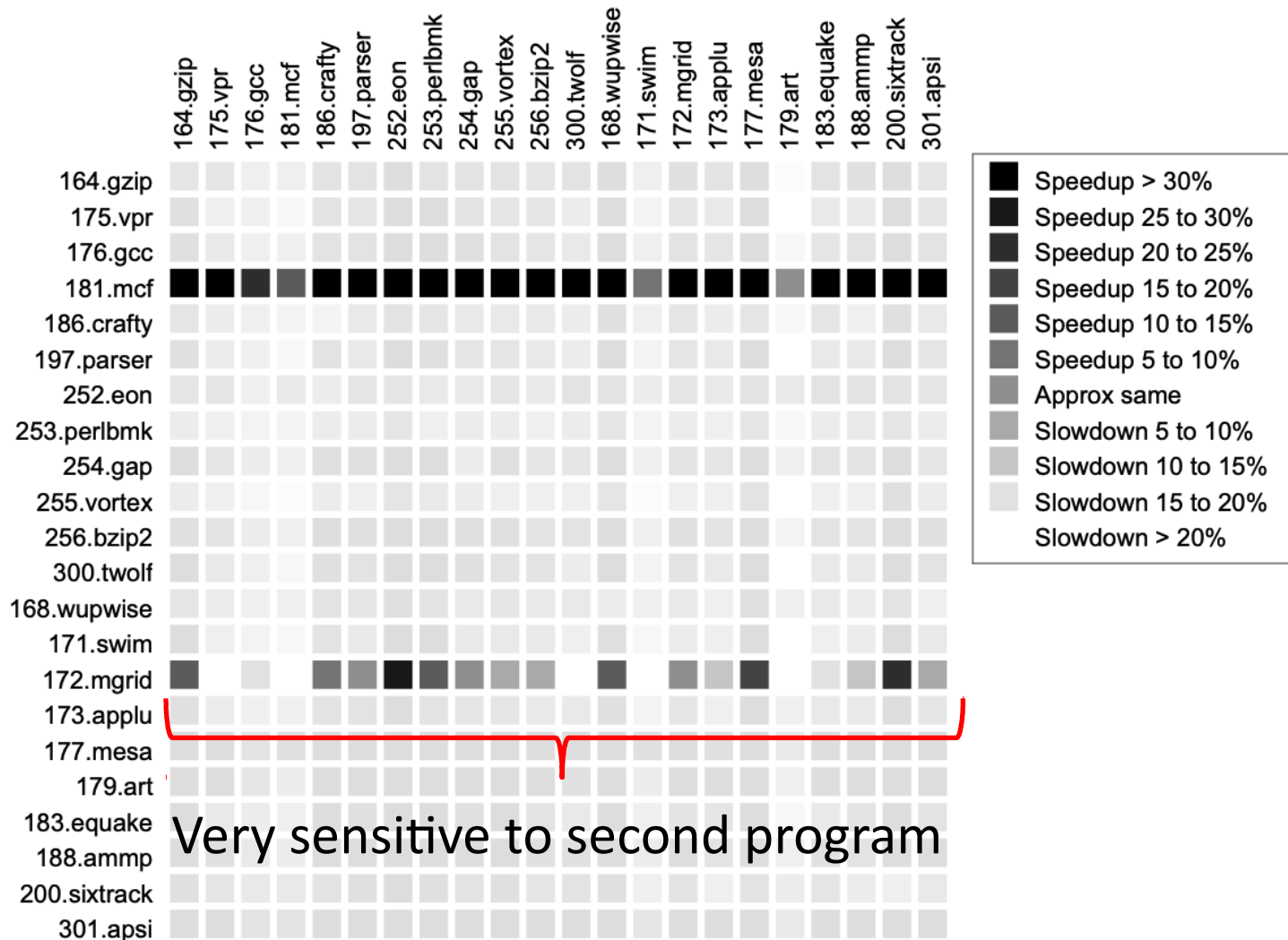
SMT Performance: Application Interaction



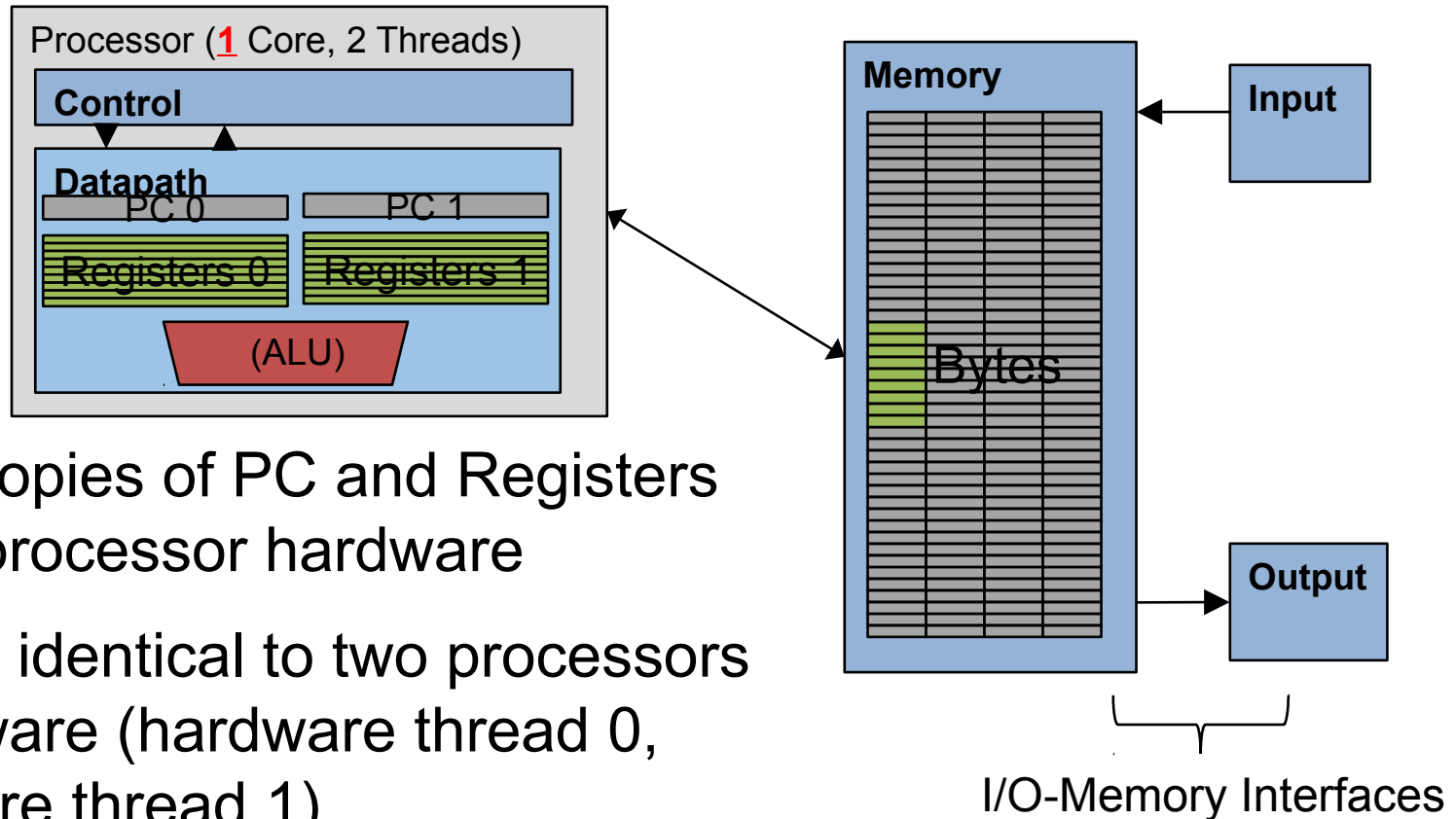
SMT Performance: Application Interaction



SMT Performance: Application Interaction



Hardware Assisted Software Multithreading



- Two copies of PC and Registers inside processor hardware
- Looks identical to two processors to software (hardware thread 0, hardware thread 1)
- Hyperthreading:
 - Both threads can be active simultaneously

Multithreading

- Logical threads
 - $\approx 1\%$ more hardware, $\approx 10\%$ (?) better performance
 - Separate registers
 - Share datapath, ALU(s), caches
- Multicore
 - \Rightarrow Duplicate Processors
 - $\approx 50\%$ more hardware, $\approx 2X$ better performance?
- Modern machines do both
 - Multiple cores with multiple threads per core

Randy's (Rather Old) Mac Air

```
$ sysctl -a | grep hw
```

```
hw.model = Core i7, 4650U
```

```
...
```

```
hw.physicalcpu: 2
```

```
hw.logicalcpu: 4
```

```
...
```

```
hw.cpufrequency =
```

```
1,700,000,000
```

```
hw.physmem =
```

```
8,589,934,592 (8 Gbytes) ...
```

```
hw.cachelinesize = 64
```

```
hw.l1icachesize: 32,768
```

```
hw.l1dcachesize: 32,768
```

```
hw.l2cachesize: 262,144
```

```
hw.l3cachesize: 4,194,304
```

Every machine is multicore,
Even your phone!

Hive Machines

hw.model = Core i7 4770K

hw.physicalcpu: 4

hw.logicalcpu: 8

...

hw.cpufrequency =
3,900,000,000

hw.physmem =
34,359,738,368

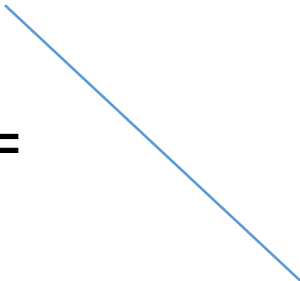
hw.cachelinesize = 64

hw.l1icachesize: 32,768

hw.l1dcachesize: 32,768

hw.l2cachesize: 262,144

hw.l3cachesize: 8,388,608



Therefore, should try up to 8 threads to see if performance gain even though only 4 cores

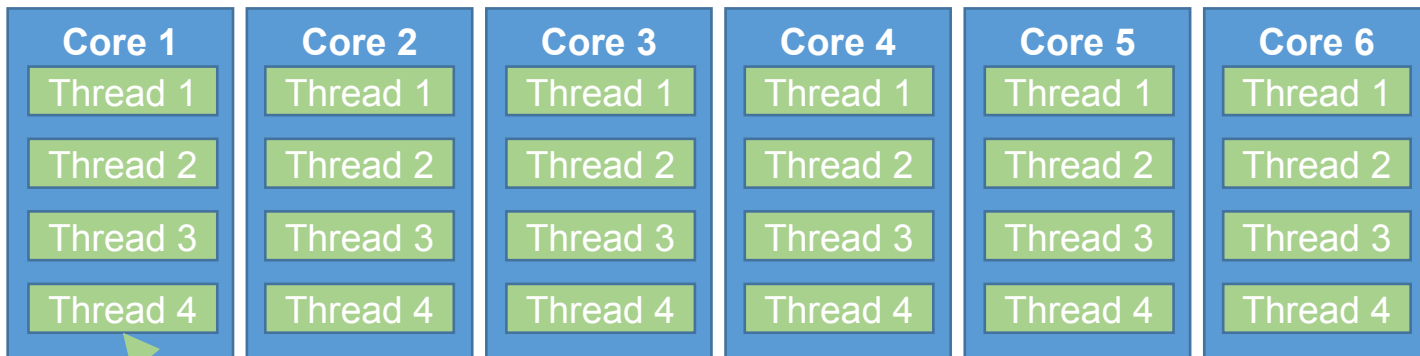
Example: 6 Cores, 24 Logical Threads

```
0:04.34 /usr/libexec/UserEventAgent (Aqua)
0:10.60 /usr/sbin/distnoted agent
0:09.11 /usr/sbin/cfprefsd agent
0:04.71 /usr/sbin/usernoted
0:02.35 /usr/libexec/nsurlsessiond
0:28.68 /System/Library/PrivateFrameworks/Calend
0:04.36 /System/Library/PrivateFrameworks/GameCe
0:01.90 /System/Library/CoreServices/cloudphotos
0:49.72 /usr/libexec/secinitd
0:01.66 /System/Library/PrivateFrameworks/TCC.fr
0:12.68 /System/Library/Frameworks/Accounts.fram
0:09.56 /usr/libexec/SafariCloudHistoryPushAgent
0:00.27 /System/Library/PrivateFrameworks/CallHi
0:00.74 /System/Library/CoreServices/mapspushd
0:00.79 /usr/libexec/fmfd
```

Thread pool:

List of threads competing for processor

OS maps threads to cores and schedules logical (software) threads



4 Logical threads per core (hardware) thread

Languages Supporting Parallel Programming

ActorScript	Concurrent Pascal	JoCaml	Orc
Ada	Concurrent ML	Join	Oz
Afnix	Concurrent Haskell	Java	Pict
Alef	Curry	Joule	Reia
Alice	CUDA	Joyce	SALSA
APL	E	LabVIEW	Scala
Axum	Eiffel	Limbo	SISAL
Chapel	Erlang	Linda	SR
Cilk	Fortran 90	MultiLisp	Stackless Python
Clean	Go	Modula-3	SuperPascal
Clojure	Io	Occam	VHDL
Concurrent C	Janus	occam- π	XC

Which one to pick?

Why So Many Parallel Programming Languages?

- Why “intrinsic”?
 - TO Intel: fix your #()&\$! Compiler!
- It's happening ... but
 - SIMD features are continually added to compilers (Intel, gcc)
 - Intense area of research
 - Research progress:
 - 20+ years to translate C into good (fast!) assembly
 - How long to translate C into good (fast!) parallel code?
 - General problem is very hard to solve
 - Present state: specialized solutions for specific cases
 - Your opportunity to become famous!

Review: Why Parallelism?

- Only path to performance is parallelism
 - Clock rates flat or declining
 - SIMD: 2X width every 3-4 years
 - AVX-512 2015, 1024b in 2018? 2019?
 - MIMD: Add 2 cores every 2 years (2, 4, 6, 8, 10, ...)
Intel Broadwell-Extreme (2Q16): 10 Physical CPUs, 20 Logical CPUs
- Key challenge: craft parallel programs with high performance on multiprocessors as # of processors increase
 - i.e., that scale
 - Scheduling, load balancing, time for synchronization, overhead for communication

Parallel Programming Languages

- Number of choices is indication of
 - No universal solution
 - Needs are very problem specific
 - E.g.,
 - Scientific computing/machine learning (matrix multiply)
 - Webserver: handle many unrelated requests simultaneously
 - Input / output: it's all happening simultaneously!
- Specialized languages for different tasks
 - Some are easier to use (for some problems)
 - None is particularly "easy" to use

Parallel Loops

- Serial execution:


```
for (int i=0; i<100; i++) {  
    ...  
}
```

- Parallel Execution:

```
for (int i=0; i<25; i++) {  
    ...  
}  
  
for (int i=25; i<50; i++) {  
    ...  
}  
  
for (int i=50; i<75; i++) {  
    ...  
}  
  
for (int i=75; i<100; i++) {  
    ...  
}
```

OpenMP Building Block: `for` loop

```
for (i=0; i<max; i++) zero[i] = 0;
```

- Breaks *for loop* into chunks, and allocate each to a separate thread
 - e.g. if **max** = 100 with 2 threads:
assign 0-49 to thread 0, and 50-99 to thread 1
 - Must have relatively simple “shape” for an OpenMP-aware compiler to be able to parallelize it
 - Necessary for the run-time system to be able to determine how many of the loop iterations to assign to each thread
 - No premature exits from the loop allowed 
 - i.e. No **break**, **return**, **exit**, **goto** statements
- In general, don't jump outside of any pragma block

Parallel for in OpenMP

```
#include <omp.h>
```

```
#pragma omp parallel for  
for (int i=0; i<100; i++) {  
    ...  
}
```

OpenMP Example

```
#include <stdio.h>
#include <omp.h>

// gcc-5 -fopenmp for.c

int main(void) {
    omp_set_num_threads(4);
    int a[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
    int N = sizeof(a)/sizeof(int);

    #pragma omp parallel for
    for (int i=0; i<N; i++) {
        printf("thread %d, i = %2d\n",
               omp_get_thread_num(), i);
        a[i] = a[i] + 100*omp_get_thread_num();
    }

    for (int i=0; i<N; i++) printf("%d ", a[i]);
    printf("\n");
}
```

```
$ gcc-5 -fopenmp
for.c; ./a.out
```

```
thread 0, i = 0
```

```
thread 1, i = 3
```

```
thread 2, i = 6
```

```
thread 3, i = 8
```

```
thread 0, i = 1
```

```
thread 1, i = 4
```

```
thread 2, i = 7
```

```
thread 3, i = 9
```

```
thread 0, i = 2
```

```
thread 1, i = 5
```

```
01 02 03 14 15 16 27 28 39 40
```

OpenMP

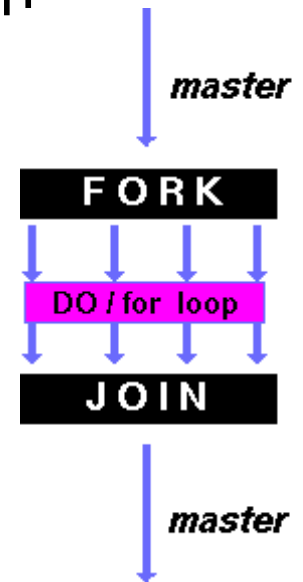
- C extension: no new language to learn
- Multi-threaded, shared-memory parallelism
 - Compiler Directives, `#pragma`
 - Runtime Library Routines, `#include <omp.h>`
- `#pragma`
 - Ignored by compilers unaware of OpenMP
 - Same source for multiple architectures
 - E.g., same program for 1 & 16 cores
- Only works with shared memory

OpenMP Parallel *for* pragma

```
#pragma omp parallel for
```

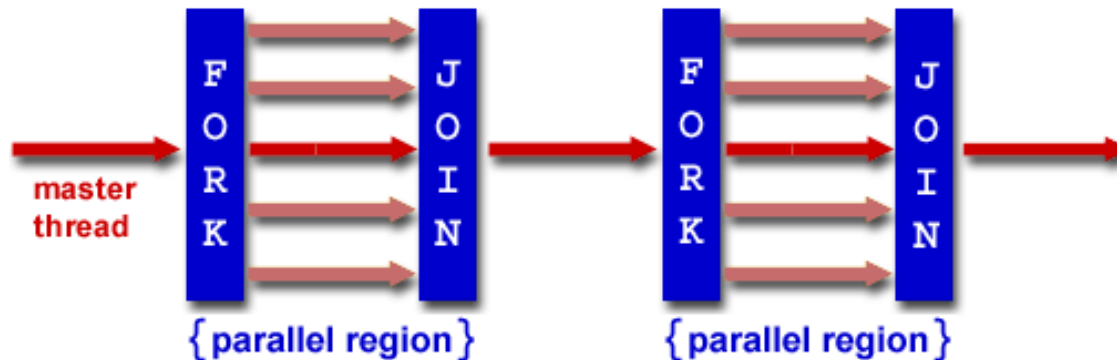
```
  for (i=0; i<max; i++) zero[i] = 0;
```

- Master thread creates additional threads, each with a separate execution context
- All variables declared outside for loop are shared by default, except for loop index which is implicitly *private* per thread
- Implicit “barrier” synchronization at end of for loop
- Divide index regions sequentially per thread
 - Thread 0 gets 0, 1, ..., (max/n)-1;
 - Thread 1 gets max/n, max/n+1, ..., 2*(max/n)-1
 - Why?



OpenMP Programming Model

- Fork - Join Model:



- OpenMP programs begin as single process (master thread)
 - Sequential execution
- When parallel region is encountered
 - Master thread “forks” into team of parallel threads
 - Executed simultaneously
 - At end of parallel region, parallel threads “join”, leaving only master thread
- Process repeats for each parallel region
 - Amdahl’s Law?

What Kind of Threads?

- OpenMP threads are operating system (software) threads
- OS will multiplex requested OpenMP threads onto available hardware threads
- Hopefully each gets a real hardware thread to run on, so no OS-level time-multiplexing
- But other tasks on machine compete for hardware threads!