University of Southern California

Viterbi School of Engineering

EE352
Computer Organization and Architecture

Exploiting Thread Level Parallelism and Data Level Parallelism

References:

1) Textbook
2) Mark Redekopp's slide series

Shahin Nazarian

Spring 2010

Exploiting Parallelism

- With increasing transistor budgets of modern processors (i.e., can do more things at the same time) the question becomes how do we find enough *useful* tasks to increase performance, or, put another way, what are the most effective ways of exploiting parallelism!
- Many types of parallelism are available
  - Instruction Level Parallelism (ILP): Overlapping instructions within a single process/thread of execution
  - Thread Level Parallelism (TLP): Overlap execution of multiple processes / threads
  - Data Level Parallelism (DLP): Overlap an operation (instruction) that is to be applied to multiple data values (usually in an array)
    - For (i=0; i < MAX; i++) { A[i] = A[i] + 5; }

Data Level Parallelism

SIMD/Vector Units

Introductory Example for SIMD

- An image is just a 2-D array of pixel values
- Pixel color represented as a number (between 0 = black, to 255 = white)

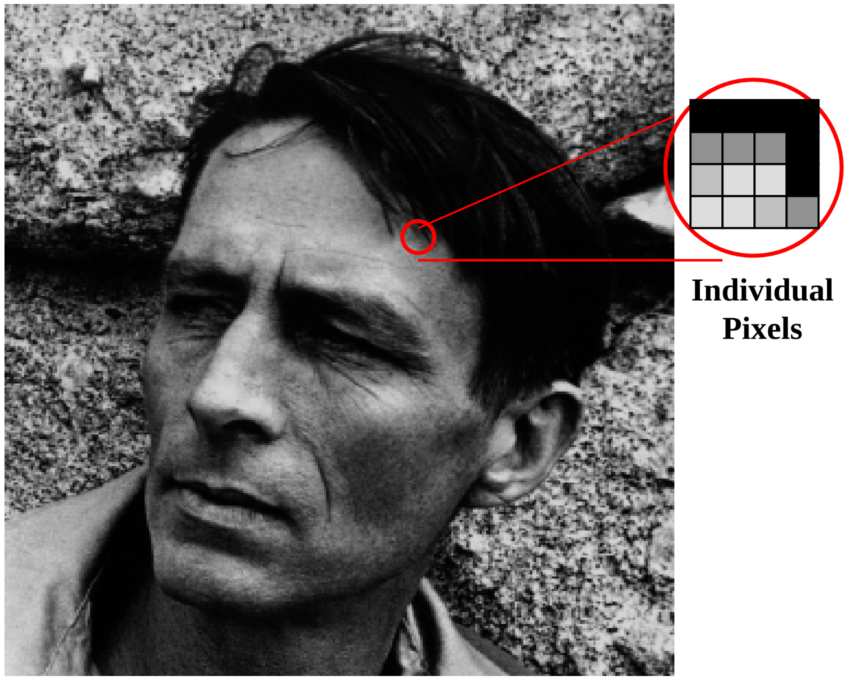
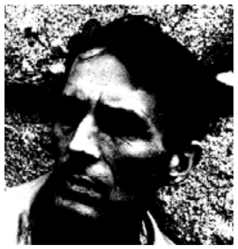


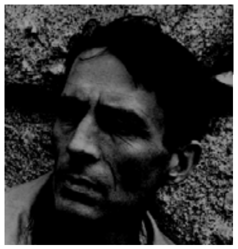
Image taken from the photo "Robin Jeffers at Ton House" (1927) by Edward Weston

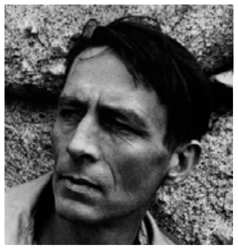
0	0	0	0
64	64	64	0
128	192	192	0
192	192	128	64

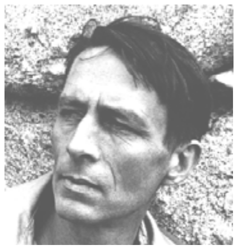
Graphics Operations

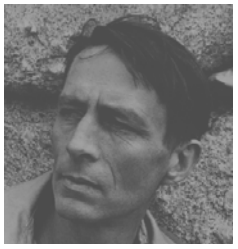
- Brightness
  - Each pixel value is increased/decreased by a constant amount
  - $P_{new} = P_{old} + B$
    - $B > 0$ = brighter
    - $B < 0$ = less bright
- Contrast
  - Each pixel value is multiplied by a constant amount
  - $P_{new} = C^*P_{old}$
    - $C > 1$ = more contrast
    - $0 < C < 1$ = less contrast
- Same operations performed on all pixels


+ Contrast


- Brightness


Original


+ Brightness


- Contrast

Vector Operations

- Need to perform single operation (instruction) such as +B or *C on multiple data values
- General-purpose computers use instructions that perform an operation on a single ("scalar") data value (i.e. lw \$t0,P_{old} / add \$t0,\$t0,5 / sw \$t0, P_{new})
- Modern processors include HW capability & instructions that perform the same operation on multiple data items (a.k.a. "vectors") using a single instruction
  - HW adds "vector" registers which can hold 128- or 256-bit data
  - 128- or 256-bit data can be interpreted as a set of 8, 16, or 32-data items
  - Example instruction: 'vaddw \$v0,\$v0,5' where vaddw stands for Vector Add Word and \$v0 is a 128-bit chunk

Vector Operations (Cont.)

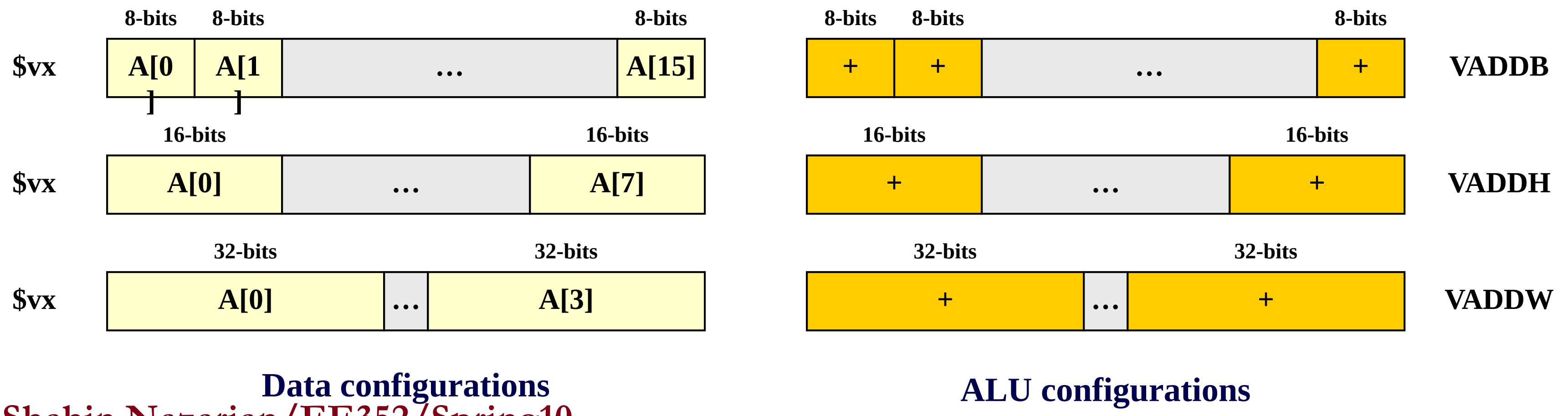
- Computers that perform the same operation/instruction on several data values are referred to as SIMD (Single Instruction, Multiple Data) machines or Vector/Stream processing machines

Vector Processing

- New instructions and WIDE registers that can be selected to operate on different size data
    - Treat data as 8-bit, 16-bit, 32-bit chunks

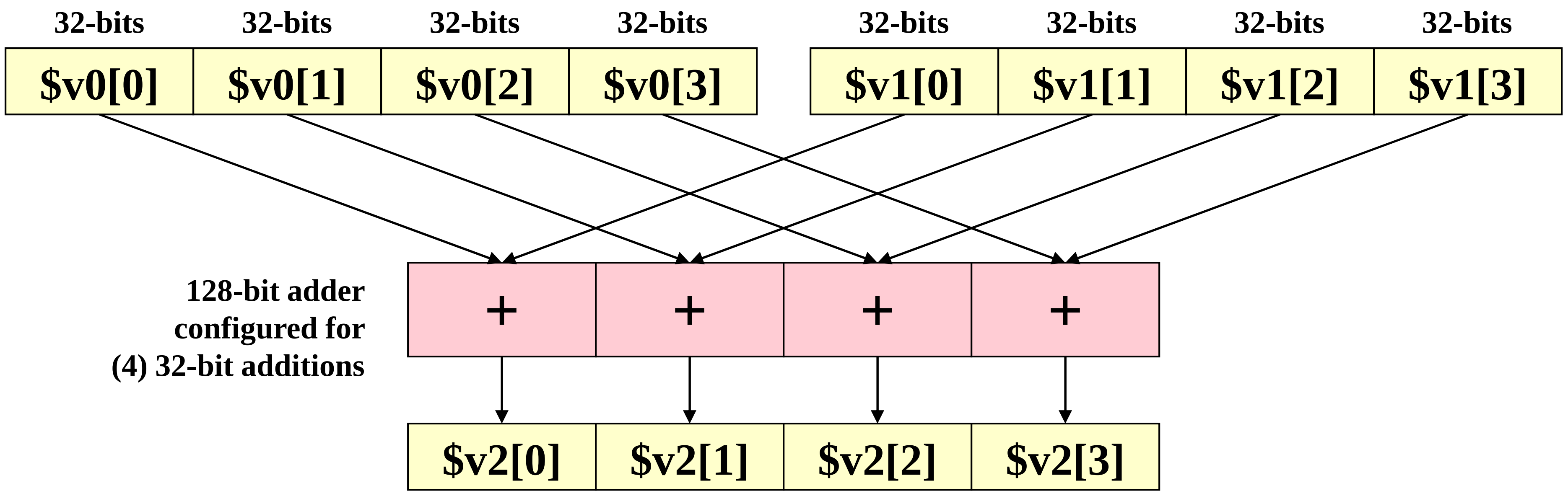
Vector Instructions

- Vector load / store
  - "LV" reads a 128-bit chunk from memory to 128-bit vector register
  - "SV" writes a 128-bit chunk to memory from 128-bit vector register
- Vector operations
  - "VADDB" = (16) 8-bit values
  - "VADDH" = (8) 16-bit values
  - "VADDW" = (4) 32-bit values



Vector Operations

- vaddw $v0,$v1,$v2



More Vector Instructions

```
for(i=MAX; i != 0; i--)
  A[i] = A[i] + 5;
```

Original "scalar" code

```
// Loop unrolled 4 times
for(i=MAX; i != 0; i=i-4){
  A[i] = A[i] + 5;
  A[i+1] = A[i+1] + 5;
  A[i+2] = A[i+2] + 5;
  A[i+3] = A[i+3] + 5;
}
```

Unrolled Scalar Code

```
$s0 = pointer to A[i]
$s1 = i = # of iterations

L1: lw   $t1,0($s0)
    addi $t1,$t1,5
    sw   $t1,0($s0)
    lw   $t1,4($s0)
    addi $t1,$t1,5
    sw   $t1,4($s0)
    lw   $t1,8($s0)
    addi $t1,$t1,5
    sw   $t1,8($s0)
    lw   $t1,12($s0)
    addi $t1,$t1,5
    sw   $t1,12($s0)
    addi $s0,$s0,16
    addi $s1,$s1,-4
    bne  $s1,$zero,L1
```

Vectorized / SIMD Code
(Could unroll this if desired)

```
$s0 = pointer to A[i]
$s1 = i = # of iterations

    lvi   $v2,5
L1: lv    $v1,0($s0)
    vaddw $v1,$v1,$v2
    sv    $v1,0($s0)
    addi  $s0,$s0,16
    addi  $s1,$s1,-4
    bne   $s1,$zero,L1
```

Vector Processing Examples

- Remember: vector processing machines are SIMD type
- Pentium
  - MMX (MultiMedia eXtension) – 64-bit registers with only integer vector operations
  - SSE (Streaming SIMD Extensions,) SSE2,SSE3 – 128-bit vectors with support for single- and double-precision FP ops
- PowerPC
  - Altivec instructions extensions
- Cell Processor
  - Each of the 8 SPEs (Synergistic Processing Elements) use 128-bit vector operations
  - No scalar (single) data units…instead just don't use n-1 slots of vector operations

Optional to Review

- MMX

  http://en.wikipedia.org/wiki/MMX_(instruction_set)

- SSE

  http://en.wikipedia.org/wiki/Streaming_SIMD_Extensions

- Altivec

  http://en.wikipedia.org/wiki/AltiVec

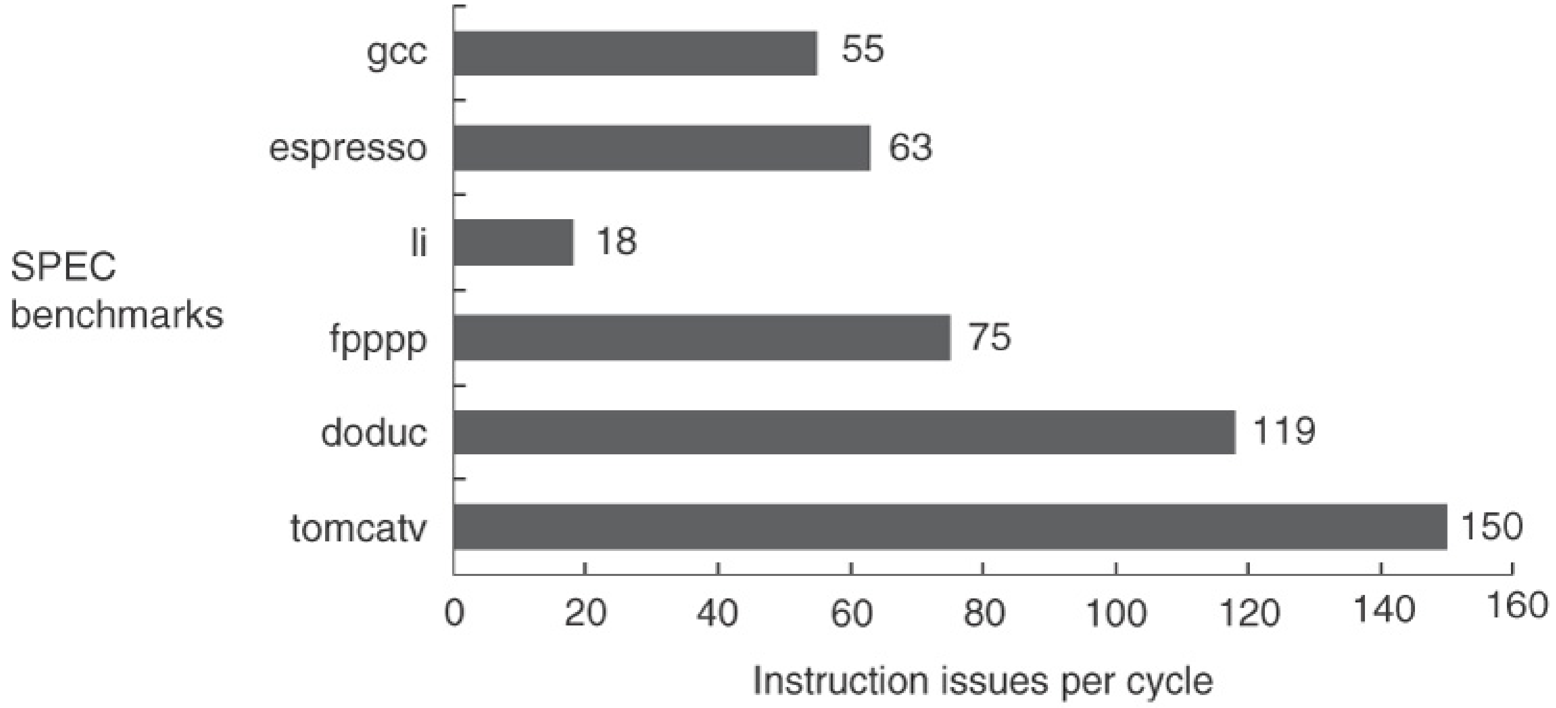
- Cell Processor
  - Check SPE and PPE in the following: http://en.wikipedia.org/wiki/Cell_(microprocessor)#Synergistic_Processing_Elements_.28SPE.29

Thread Level Parallelism

Mulithreaded and Multicore Processors

Limit to ILP

- ILP has limits due to actual data (RAW) dependencies
- Impractical amount of HW needed to get close to this limit
- VLIW (Very Large Instruction Word) processors take advantage of ILP



Thread-Level Parallelism

- ILP attempts to find, fetch, and execute independent instructions from within a single thread
  - Requires a lot of HW resources and may not be possible
- TLP fetches instruction from different threads which are independent *by definition*
  - A thread: Separate logical stream of instructions + data
  - Separate instructions = Separate PC
  - Separate data = Separate register sets, stack, and possibly memory address space
- This way we can fill our issue slots w/o having to work so hard to find independent instructions

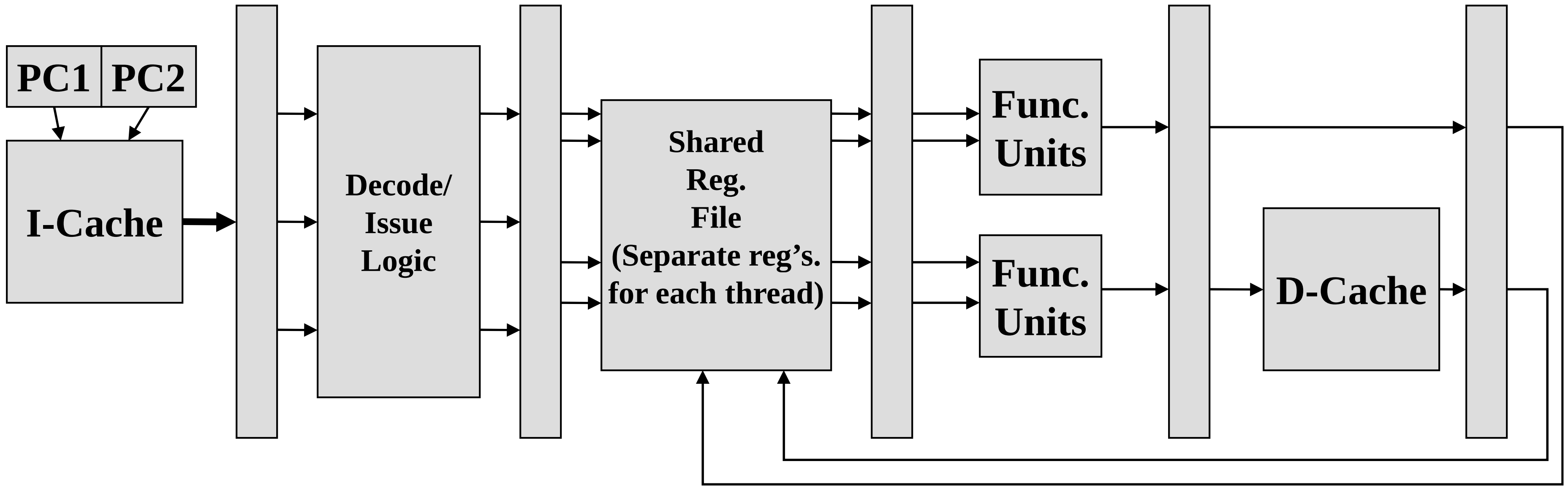
Multi-threaded Processors

- Requires duplicating and/or sharing HW resources to support multiple threads
- Duplicated HW
  - 2 PCs to be able to fetch instructions from different threads
  - Instructions will be "tagged" with thread_id to maintain separability
- Duplicated or shared HW
  - Logically separate registers (can be physically separate or shared)
    - $t0 for thread A should be different from $t0 for thread B
  - Branch Predictors, TLBs* (translation-lookaside buffer), etc.

* will be discussed as part of virtual Memory

Example: 2-way Threaded Machine

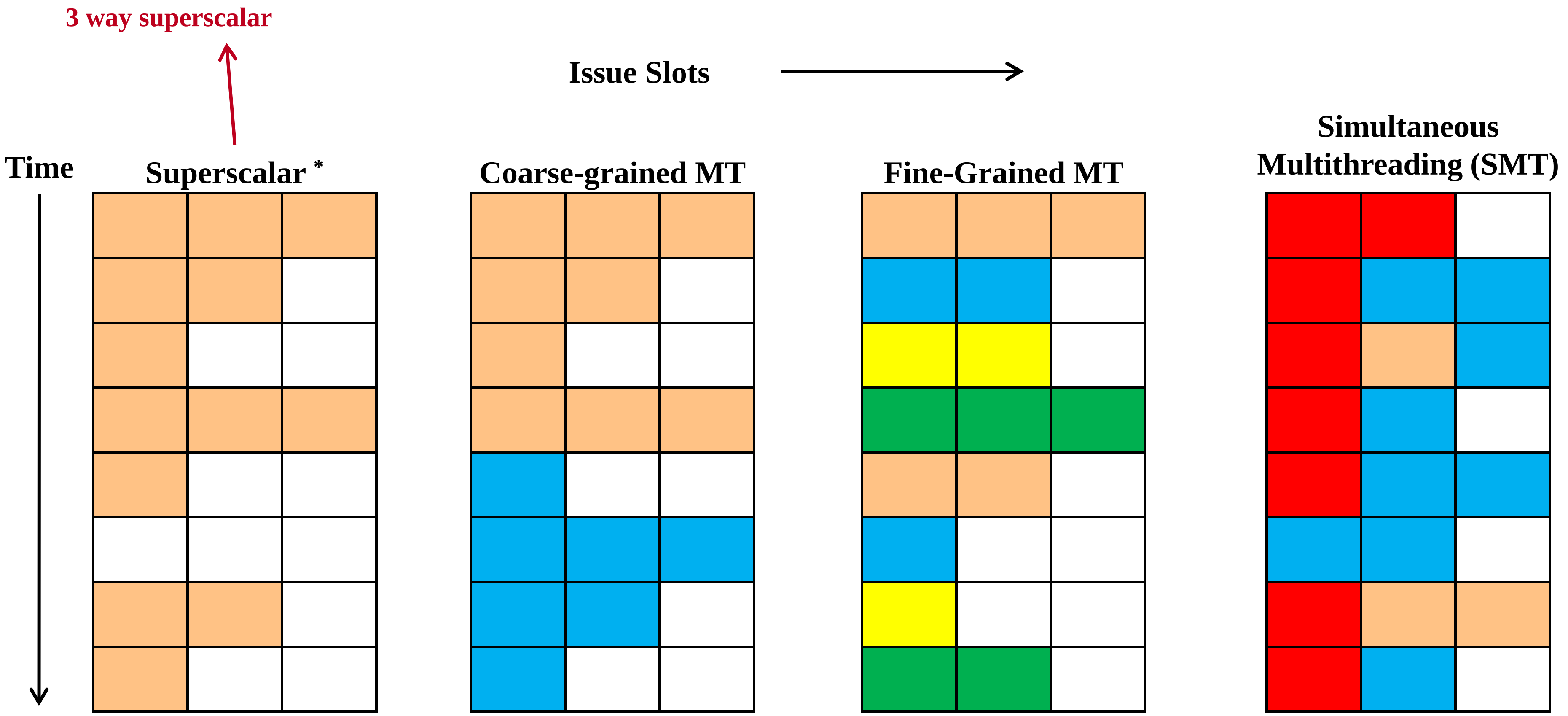
- Instructions are fetched from two different threads
- Different styles of multi-threading determine when they issue (Coarse- or fine-grained, simultaneous)
- There is some level of sharing of resources



Types/Levels of Multithreading

- How should we overlap and share the HW between instructions from different threads
  - Coarse-grained Multithreading: Execute one thread with all HW resources until a long latency event, e.g., cache-miss or misprediction will incur a stall or pipeline flush; in such case switch to another thread
  - Fine-grained Multithreading: Alternate fetching instructions from a different thread each clock
  - Simultaneous Multithreading: Fetch and execute instructions from different threads at the same time

Levels of TLP

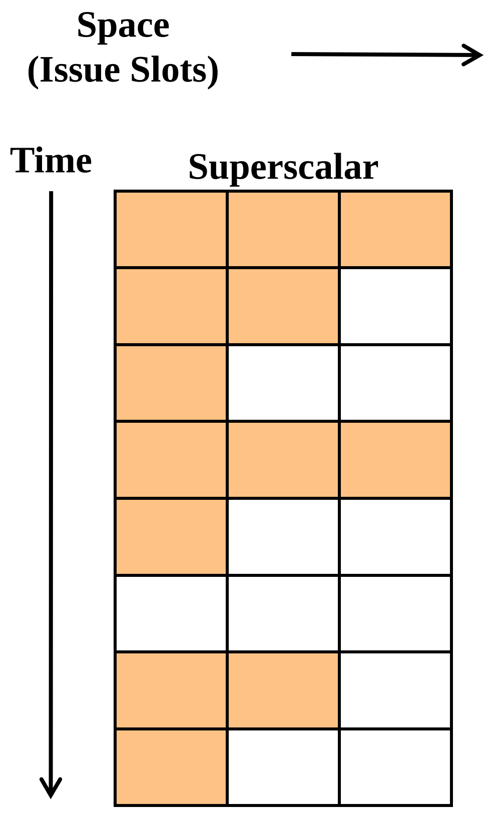


Only instructions from a single thread	Alternate threads when one hits a long-latency event like a stall due to cache-miss, pipeline flush, etc.	Alternate threads every cycle (Sun UltraSparc T2)	Mix instructions from threads during same issue cycle (Intel HyperThreading, IBM Power 5)

* Discussed later

Performance Insights

- Parallelism can be exploited in time & space
  - Time = Deep pipelining with HIGH clock frequencies
  - Space = Multiple ways / Multi-threading
    - HW needed for an n-way issue processor requires HW to grow as $O(n^2)$
- Improving performance of a single thread
  - Requires wide issue processor with fast clock rate
- All of this is BAD for power consumption!!



Only instructions from a single thread

Multithreading Performance

- Takes better advantage of the multiple issue slots and functional units of a modern processor
- Coarse or fine-grained MT
  - Only issue instructions from one thread per clock (limited by ILP)
  - Single thread performance is lowered because of time division between threads
- Simultaneous MT
  - Good single thread performance while filling 'idle' slots with another thread
  - Requires all HW resources of a multiple-issue processor
- However, n-way issue processor requires HW to grow as $O(n^2)$ which is bad for power consumption

Power

- Power consumption decomposed into:
  - Static: Power constantly being dissipated (grows with # of transistors)
  - Dynamic: Power consumed for switching a bit (1 to 0)
- Most power consumption is dynamic
- $P_{DYN} = I_{DYN} * V_{DD} \approx \frac{1}{2} C_{TOT} V_{DD}^2 f$
  - Recall, I = C dV/dt
  - V_{DD} is the logic '1' voltage, f = clock frequency
- Reducing voltage and frequency reduces power
  - Unfortunately, higher frequencies require higher voltages
  - But, reducing frequency also allows us to reduce V_{DD}
  - Implies power is proportional to f^3 (a cubic savings in power if we can reduce f)

Temperature

- Temperature is related to power consumption
  - Locations on the chip that burn more power will usually run hotter
    - Locations where bits toggle (register file, etc.) often will become quite hot especially if toggling continues for a long period of time
  - Too much heat can destroy a chip
  - Can use sensors to dynamically sense temperature

Temperature (Cont.)

- Techniques for controlling temperature
  - External measures: Remove and spread the heat
    - Heat sinks, fans, even liquid cooled machines
  - Architectural measures
    - Throttle performance (run at slower frequencies / lower voltages)
    - Global clock gating (pause..turn off the clock)
    - None…results can be catastrophic
- Optional to review:

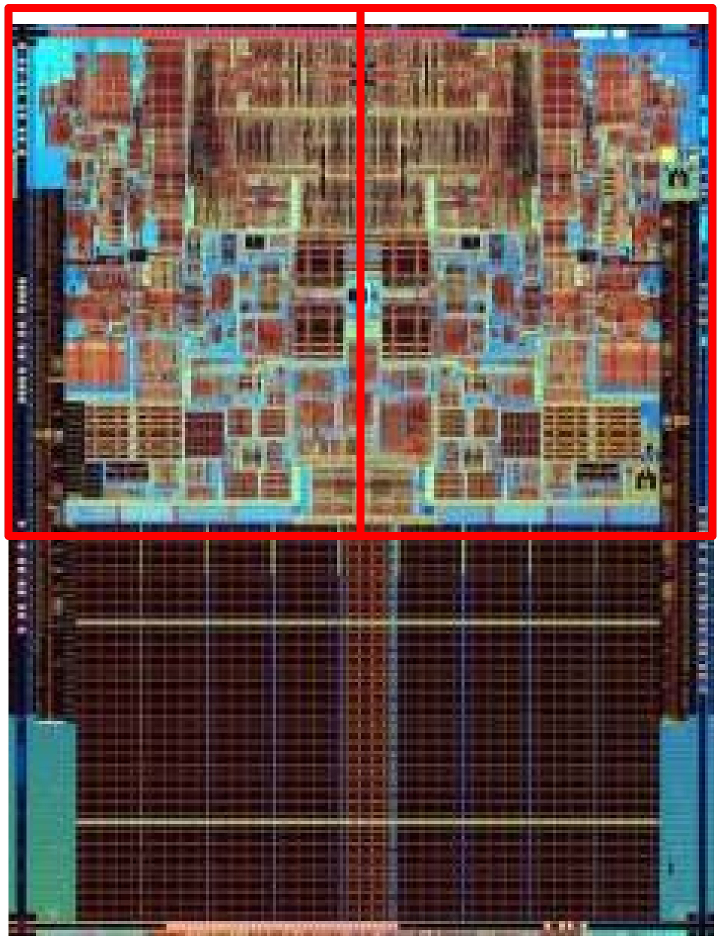
  http://www.tomshardware.com/reviews/hot-spot,365.html

MT Performance Insights

- Change goal from pure performance to performance/HW area or performance/watt
- Multithreading is a step in the right direction but can be taken further by replicating more and more processor cores (which may themselves be multithreaded)
- We can increase total computing throughput by using multiple processor cores on a chip
  - Can keep frequency down while still achieving good overall computing throughput

Chip Multiprocessors (CMP)

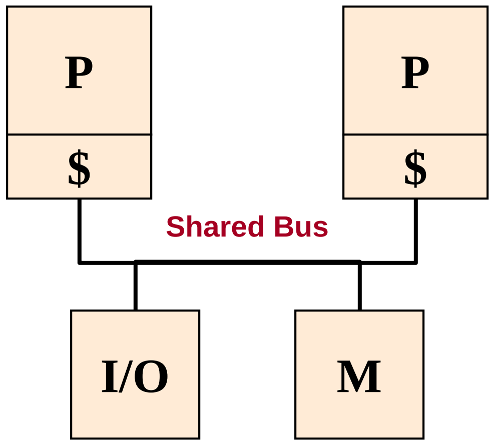
- Separate cores (duplicating all resources) w/ low issue widths
  - Avoids time division of fine- and coarse-grained MT
  - Often requires simpler, smaller HW than MT at similar issue slots
    - A 16-way issue superscalar/SMT would likely take more area and be MUCH harder to design than 16 simple, separate cores
  - Better
    - performance / area
    - performance / watt
- Greater "spatial" parallelism (via the many cores) allows us to reduce clock rates (reduce temporal parallelism)



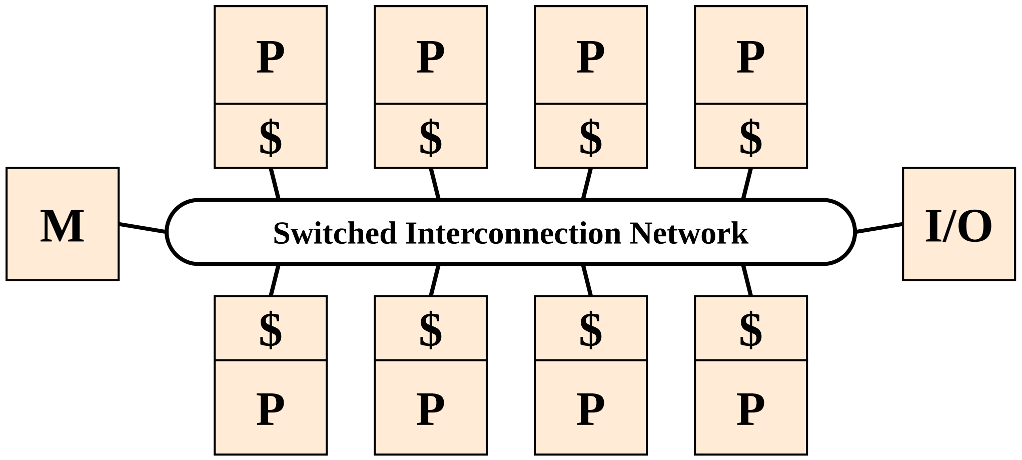
Intel® Core 2 Duo™ Processor Die Photo

Typical CMP Organization

- Processors each have their own L1 caches (always access data from these caches)
- Shared L2 or memory
- Interconnection network for communicating data between caches
  - Shared Bus allows only one transfer at a time
  - Switched interconnection network allows multiple simultaneous transfers



Simple CMP connected via shared bus (Core 2 Duo)



More cores usually requires a more advanced interconnection network (Cell Processor)