Thread-Level Parallelism — Simultaneous MultiThreading (SMT) & Chip Multi-Processors (CMP)

Hung-Wei
Recap: What about “linked list”

Static instructions:
- LOOP: `ld X10, 8(X10)`
- `addi X7, X7, 1`
- `bne X10, X0, LOOP`

Dynamic instructions:
- `ld X10, 8(X10)`
- `addi X7, X7, 1`
- `bne X10, X0, LOOP`

ILP is low because of data dependencies.
Demo: ILP within a program

- perf is a tool that captures performance counters of your processors and can generate results like branch mis-prediction rate, cache miss rates and ILP.
Simultaneous multithreading
Simultaneous multithreading

• The processor can schedule instructions from different threads/processes/programs
• Fetch instructions from different threads/processes to fill the not utilized part of pipeline
  • Exploit “thread level parallelism” (TLP) to solve the problem of insufficient ILP in a single thread
  • You need to create an illusion of multiple processors for OSs
Simultaneous multithreading

1. `ld X10, 8(X10)`
2. `addi X7, X7, 1`
3. `bne X10, X0, LOOP`
4. `ld X10, 8(X10)`
5. `addi X7, X7, 1`
6. `bne X10, X0, LOOP`
7. `ld X10, 8(X10)`
8. `addi X7, X7, 1`
9. `bne X10, X0, LOOP`
10. `ld X1, 0(X10)`
11. `addi X10, X10, 8`
12. `add X20, X20, X1`
13. `bne X10, X2, LOOP`
14. `ld X1, 0(X10)`
15. `addi X10, X10, 8`
16. `add X20, X20, X1`
17. `bne X10, X2, LOOP`
18. `ld X1, 0(X10)`
19. `addi X10, X10, 8`
20. `add X20, X20, X1`
21. `bne X10, X2, LOOP`
SuperScalar Processor w/ ROB

Fetch/decode instruction

Renaming logic

Instruction Queue

Unresolved Branch

Register mapping table

Physical Registers

Address Resolution

Integer ALU

Floating-Point Adder

Floating-Point Mul/Div

Branch

Address Queue

Store Queue

Memory

Load Queue

Store Queue
SMT

- Improve the throughput of execution
  - May increase the latency of a single thread
- Less branch penalty per thread
- Increase hardware utilization
- Simple hardware design: Only need to duplicate PC/Register Files
- Real Case:
  - Intel HyperThreading (supports up to two threads per core)
    - Intel Pentium 4, Intel Atom, Intel Core i7
  - AMD RyZen
SMT SuperScalar Processor w/ ROB

Instruction Queue

Fetch/decode instruction

Renaming logic

Address Resolution

Integer ALU

Floating-Point Adder

Floating-Point Mul/Div

Branch

Load Queue

Store Queue

Memory

PC #1

Fetch/decode instruction

PC #2

Register mapping table #1

Physical Registers

Register mapping table #2

O(IW^4)
Power consumption
Power & Energy

• Regarding power and energy, how many of the following statements are correct?
  ① Lowering the power consumption helps extending the battery life
  ② Lowering the power consumption helps reducing the heat generation
  ③ Lowering the energy consumption helps reducing the electricity bill
  ④ A CPU with 10% utilization can still consume 33% of the peak power

A. 0  
B. 1  
C. 2  
D. 3  
E. 4
Dynamic/Active Power

• The power consumption due to the switching of transistor states
• Dynamic power per transistor
  \[ P_{\text{dynamic}} \sim \alpha \times C \times V^2 \times f \times N \]
  • \( \alpha \): average switches per cycle
  • \( C \): capacitance
  • \( V \): voltage
  • \( f \): frequency, usually linear with \( V \)
  • \( N \): the number of transistors
Dynamic/Active Power

• The power consumption due to the switching of transistor states
• Dynamic power per transistor
  \[ P_{\text{dynamic}} \sim \alpha \times C \times V^2 \times f \times N \]
  • \( \alpha \): average switches per cycle
  • \( C \): capacitance
  • \( V \): voltage
  • \( f \): frequency, usually linear with \( V \)
  • \( N \): the number of transistors
Chip-multiprocessor: a response to dynamic power issues
More cores per chip, slower per core

<table>
<thead>
<tr>
<th>Product</th>
<th>Intel® Xeon® Processor E7-8890 v4</th>
<th>Intel® Xeon® Processor E7-8889 v4</th>
<th>Intel® Xeon® Processor E7-8890 v4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>Launched</td>
<td>Launched</td>
<td>Launched</td>
</tr>
<tr>
<td>Launch Date</td>
<td>Q2’16</td>
<td>Q2’16</td>
<td>Q2’16</td>
</tr>
<tr>
<td>Lithography</td>
<td>14 nm</td>
<td>14 nm</td>
<td>14 nm</td>
</tr>
<tr>
<td>Performance</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td># of Cores</td>
<td>24</td>
<td>4</td>
<td>22</td>
</tr>
<tr>
<td># of Threads</td>
<td>48</td>
<td>8</td>
<td>44</td>
</tr>
<tr>
<td>Processor Base Frequency</td>
<td>2.20 GHz</td>
<td>3.20 GHz</td>
<td>2.20 GHz</td>
</tr>
<tr>
<td>Max Turbo Frequency</td>
<td>3.40 GHz</td>
<td>3.50 GHz</td>
<td>3.30 GHz</td>
</tr>
<tr>
<td>Cache</td>
<td>50 MB</td>
<td>60 MB</td>
<td>55 MB</td>
</tr>
<tr>
<td>Bus Speed</td>
<td>9.6 GT/s</td>
<td>5.6 GT/s</td>
<td>9.6 GT/s</td>
</tr>
<tr>
<td># of QPI Links</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>TDP</td>
<td>165 W</td>
<td>140 W</td>
<td>150 W</td>
</tr>
</tbody>
</table>
ARM’s big.LITTLE architecture

big.LITTLE system

Interrupt Controller

big

LITTLE

Coherent Interconnect

Rest of system
(GPU, Video, Display, etc.)

Memory Controller

DDR
Concept of CMP

Processor

Core
- Registers
  - L1-$
  - L2-$

Core
- Registers
  - L1-$
  - L2-$

Core
- Registers
  - L1-$
  - L2-$

Core
- Registers
  - L1-$
  - L2-$

Last-level $ (LLC)
Architectural Support for Parallel Programming
Parallel programming

• To exploit parallelism you need to break your computation into multiple "processes" or multiple "threads"

• Processes (in OS/software systems)
  • Separate programs actually running (not sitting idle) on your computer at the same time.
  • Each process will have its own virtual memory space and you need explicitly exchange data using inter-process communication APIs

• Threads (in OS/software systems)
  • Independent portions of your program that can run in parallel
  • All threads share the same virtual memory space

• We will refer to these collectively as "threads"
  • A typical user system might have 1-8 actively running threads.
  • Servers can have more if needed (the sysadmins will hopefully configure it that way)
What software thinks about “multiprogramming” hardware
What software thinks about “multiprogramming” hardware

Others do not see the updated value in the cache and keep working — incorrect result!
Coherency & Consistency

- Coherency — Guarantees all processors see the same value for a variable/memory address in the system when the processors need the value at the same time
  - What value should be seen
- Consistency — All threads see the change of data in the same order
  - When the memory operation should be done
Simple cache coherency protocol

- Snooping protocol
  - Each processor broadcasts / listens to cache misses
- State associate with each block (cacheline)
  - Invalid
    - The data in the current block is invalid
  - Shared
    - The processor can read the data
    - The data may also exist on other processors
  - Exclusive
    - The processor has full permission on the data
    - The processor is the only one that has up-to-date data
Coherent way-associative cache

Memory address: 0x0

Memory address: 0b0000100000100100

States

<table>
<thead>
<tr>
<th>D</th>
<th>tag</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>1</td>
<td>0x29 IIJJKKLLMMNOOPP</td>
</tr>
<tr>
<td>01</td>
<td>1</td>
<td>0xDE QQRRSSTTUUVWwXX</td>
</tr>
<tr>
<td>01</td>
<td>0</td>
<td>0x10 YYZZAABBCCDEEFF</td>
</tr>
<tr>
<td>00</td>
<td>1</td>
<td>0x8A AABBCDDEEEGFHH</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>0x60 IIJJKKLLMMNOOPP</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>0x70 QQRRSSTTUUVWwXX</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>0x10 QQRRSSTTUUVWwXX</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>0x11 YYZZAABBCCDEEFF</td>
</tr>
</tbody>
</table>

States

<table>
<thead>
<tr>
<th>D</th>
<th>tag</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>1</td>
<td>0x00 AABBCCDDEEEGFHH</td>
</tr>
<tr>
<td>01</td>
<td>1</td>
<td>0x10 IIJJKKLLMMNOOPP</td>
</tr>
<tr>
<td>01</td>
<td>0</td>
<td>0xA1 QQRRSSTTUUVWwXX</td>
</tr>
<tr>
<td>00</td>
<td>1</td>
<td>0x10 YYZZAABBCCDEEFF</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>0x31 AABBCDDEEEGFHH</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>0x45 IIJJKKLLMMNOOPP</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>0x41 QQRRSSTTUUVWwXX</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>0x68 YYZZAABBCCDEEFF</td>
</tr>
</tbody>
</table>

Hit?
Snooping Protocol

- **Invalid**
  - read/write miss (bus)
  - write miss (processor)
  - write miss (bus)
- **Shared**
  - read miss (processor)
  - write miss (bus)
  - write request (processor)
  - read miss (bus)
  - write back data
- **Exclusive**
  - write miss (processor)
  - write miss (bus)
  - write hit
  - write back data

read/hit miss
read request (processor)
What happens when we write in coherent caches?

```plaintext
for(i=0;i<size/4;i++)
    sum += a[i];

for(i=size/4;i<size/2;i++)
    sum += a[i];

for(i=size/2;i<3*size/4;i++)
    sum += a[i];

for(i=3*size/4;i<size;i++)
    sum += a[i];
```

```plaintext
sum = 0
```

```plaintext
write miss/
 invalidate
```
False sharing

Shared Virtual Address Space

Thread

A[0] = 0xDEADBEEF
A[1] = 0
A[2] = 0
A[3] = 0

Thread

A[0] = 0
A[1] = 0
A[2] = 0
A[3] = 0

Thread

A[0] = 0
A[1] = 0
A[2] = 0
A[3] = 0

Thread

A[0] = 0
A[1] = 0
A[2] = 0
A[3] = 0

write miss/invalidate

write back

A[0] = 0
A[1] = 0
A[2] = 0
A[3] = 0

read miss
4Cs of cache misses

- 3Cs:
  - Compulsory, Conflict, Capacity
- Coherency miss:
  - A “block” invalidated because of the sharing among processors.
False sharing

- True sharing
  - Processor A modifies X, processor B also want to access X.

- False sharing
  - Processor A modifies X, processor B also want to access Y. However, Y is invalidated because X and Y are in the same block!
fence instructions

• x86 provides an “mfence” instruction to prevent reordering across the fence instruction
• x86 only supports this kind of “relaxed consistency” model. You still have to be careful enough to make sure that your code behaves as you expected

<table>
<thead>
<tr>
<th>thread 1</th>
<th>thread 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>a=1;</td>
<td>b=1;</td>
</tr>
<tr>
<td>mfence a=1 must occur/update before mfence</td>
<td>mfence b=1 must occur/update before mfence</td>
</tr>
<tr>
<td>x=b;</td>
<td>y=a;</td>
</tr>
</tbody>
</table>
Take-aways of parallel programming

• Processor behaviors are non-deterministic
  • You cannot predict which processor is going faster
  • You cannot predict when OS is going to schedule your thread
• Cache coherency only guarantees that everyone would eventually have a coherent view of data, but not when
• Cache consistency is hard to support
Dark Silicon
Static/Leakage Power

• The power consumption due to leakage — transistors do not turn all the way off during no operation
• Becomes the dominant factor in the most advanced process technologies.

\[ P_{\text{leakage}} \sim N \times V \times e^{-V_t} \]

- \( N \): number of transistors
- \( V \): voltage
- \( V_t \): threshold voltage where transistor conducts (begins to switch)

Figure 1: Leakage power becomes a growing problem as demands for more performance and functionality drive chipmakers to nanometer-scale process nodes (Source: IBS).
**Dennardian Scaling**

- Given a scaling factor $S$

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Relation</th>
<th>Classical Scaling</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power Budget</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Chip Size</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Vdd (Supply Voltage)</td>
<td>$1/S$</td>
<td>1/S</td>
</tr>
<tr>
<td>Vt (Threshold Voltage)</td>
<td>$1/S$</td>
<td>1/S</td>
</tr>
<tr>
<td>tex (oxide thickness)</td>
<td>$1/S$</td>
<td>1/S</td>
</tr>
<tr>
<td>W, L (transistor dimensions)</td>
<td>$1/S$</td>
<td>1/S</td>
</tr>
<tr>
<td>Cgate (gate capacitance)</td>
<td>$WL/tox$</td>
<td>1/S</td>
</tr>
<tr>
<td>Isat (saturation current)</td>
<td>$WVdd/tox$</td>
<td>1/S</td>
</tr>
<tr>
<td>F (device frequency)</td>
<td>$Isat/(CgateVdd)$</td>
<td>$S$</td>
</tr>
<tr>
<td>D (Device/Area)</td>
<td>$1/(WL)$</td>
<td>$S^2$</td>
</tr>
<tr>
<td>p (device power)</td>
<td>$IsatVdd$</td>
<td>$1/S^2$</td>
</tr>
<tr>
<td>P (chip power)</td>
<td>$Dp$</td>
<td>1</td>
</tr>
<tr>
<td>U (utilization)</td>
<td>$1/P$</td>
<td>1</td>
</tr>
</tbody>
</table>
Dennardian Broken

Given a scaling factor $S$

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Relation</th>
<th>Classical Scaling</th>
<th>Leakage Limited</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power Budget</td>
<td></td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Chip Size</td>
<td></td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Vdd (Supply Voltage)</td>
<td></td>
<td>1/S</td>
<td>1</td>
</tr>
<tr>
<td>Vt (Threshold Voltage)</td>
<td></td>
<td>1/S</td>
<td>1</td>
</tr>
<tr>
<td>tex (oxide thickness)</td>
<td></td>
<td>1/S</td>
<td>1/S</td>
</tr>
<tr>
<td>W, L (transistor dimensions)</td>
<td></td>
<td>1/S</td>
<td>1/S</td>
</tr>
<tr>
<td>Cgate (gate capacitance)</td>
<td></td>
<td>WL/tox</td>
<td>1/S</td>
</tr>
<tr>
<td>Isat (saturation current)</td>
<td></td>
<td>WVdd/tox</td>
<td>1/S</td>
</tr>
<tr>
<td>F (device frequency)</td>
<td></td>
<td>Isat/(CgateVdd)</td>
<td>S</td>
</tr>
<tr>
<td>D (Device/Area)</td>
<td></td>
<td>1/(WL)</td>
<td>S^2</td>
</tr>
<tr>
<td>p (device power)</td>
<td></td>
<td>IsatVdd</td>
<td>1/S^2</td>
</tr>
<tr>
<td>P (chip power)</td>
<td></td>
<td>Dp</td>
<td>S^2</td>
</tr>
<tr>
<td>U (utilization)</td>
<td></td>
<td>1/P</td>
<td>1/S^2</td>
</tr>
</tbody>
</table>
Power consumption to light on all transistors

<table>
<thead>
<tr>
<th>Chip</th>
<th>Dennardian Scaling</th>
<th>Dennardian Broken</th>
</tr>
</thead>
<tbody>
<tr>
<td>11111111</td>
<td>0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5</td>
<td>11111111</td>
</tr>
<tr>
<td>11111111</td>
<td>0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5</td>
<td>On ~ 50W</td>
</tr>
<tr>
<td>11111111</td>
<td>0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5</td>
<td>Off ~ 0W</td>
</tr>
<tr>
<td>11111111</td>
<td>0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5</td>
<td>Dark!</td>
</tr>
<tr>
<td>11111111</td>
<td>0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5</td>
<td>11111111</td>
</tr>
<tr>
<td>11111111</td>
<td>0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5</td>
<td>=49W</td>
</tr>
<tr>
<td>11111111</td>
<td>0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5</td>
<td>=50W</td>
</tr>
<tr>
<td>11111111</td>
<td>0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5</td>
<td>=100W!</td>
</tr>
</tbody>
</table>
Dynamic/Active Power

• The power consumption due to the switching of transistor states

• Dynamic power per transistor

\[ P_{\text{dynamic}} \sim \alpha \times C \times V^2 \times f \times N \]

• \( \alpha \): average switches per cycle

• \( C \): capacitance

• \( V \): voltage

• \( f \): frequency, usually linear with \( V \)

• \( N \): the number of transistors
Static/Leakage Power

• The power consumption due to leakage — transistors do not turn all the way off during no operation
• Becomes the dominant factor in the most advanced process technologies.

\[ P_{\text{leakage}} \sim \frac{N}{N_{\text{off}}} \cdot V \cdot e^{-V_t} \]

- \( N \): number of transistors
- \( V \): voltage
- \( V_t \): threshold voltage where transistor conducts (begins to switch)

How about static power?

Figure 1: Leakage power becomes a growing problem as demands for more performance and functionality drive chipmakers to nanometer-scale process nodes (Source: IBS).
Disable circuits if not-in-use
NVIDIA’s Turing Architecture

- Cache
- Control
- Registers
  - FP64
  - INT
  - FP32
  - TCU
  - TCU
- Load/Store
- SPU
Programming in Turing Architecture

Use tensor cores
```c
    cublasErrCheck(cublasSetMathMode(cublasHandle, CUBLAS_TENSOR_OP_MATH));
```

Make them 16-bit
```c
    convertFp32ToFp16 <<< (MATRIX_M * MATRIX_K + 255) / 256, 256 >>> (a_fp16, a_fp32, MATRIX_M * MATRIX_K);
    convertFp32ToFp16 <<< (MATRIX_K * MATRIX_N + 255) / 256, 256 >>> (b_fp16, b_fp32, MATRIX_K * MATRIX_N);
```

call Gemm
```c
    cublasErrCheck(cublasGemmEx(cublasHandle, CUBLAS_OP_N, CUBLAS_OP_N, MATRIX_M, MATRIX_N, MATRIX_K, 
                                &alpha, 
                                a_fp16, CUDA_R_16F, MATRIX_M, 
                                b_fp16, CUDA_R_16F, MATRIX_K, 
                                &beta, 
                                c_cublas, CUDA_R_32F, MATRIX_M, 
                                CUDA_R_32F, CUBLAS_GEMM_DFALT_TENSOR_OP));
```
NVIDIA’s Turing Architecture

You can only use either type of these ALUs, but not all of them.
The rise of ASICs
Say, we want to implement \( a[i] += a[i+1] \times 20 \)

- This is what we need in RISC-V in each iteration

\[
\begin{align*}
ld &\quad X1, 0(X0) \\
ld &\quad X2, 8(X0) \\
add &\quad X3, X31, #20 \\
mul &\quad X2, X2, X3 \\
add &\quad X1, X1, X2 \\
sd &\quad X1, 0(X0)
\end{align*}
\]
This is what you need for these instructions
We don’t need instruction fetch given it’s a fixed function.
Specialize the circuit

We don’t need these many registers, complex control, decode

We don’t need instruction fetch given it’s a fixed function
We don’t need ALUs, branches, hazard detections…

We don’t need these many registers, complex control, decode

We don’t need instruction fetch given it’s a fixed function
Specialize the circuit

We don’t need big ALUs, branches, hazard detections…

We don’t need these many registers, complex control, decode

We don’t need instruction fetch given it’s a fixed function
Rearranging the datapath

```
ld   X1, 0(X0)
ld   X2, 8(X0)
add  X3, X31, #20
mul  X2, X2, X3
add  X1, X1, X2
sd   X1, 0(X0)
```
The pipeline for $a[i] += a[i+1]*20$

Each stage can still be as fast as the pipelined processor.

But each stage is now working on what the original 6 instructions would do.
What TPU looks like
TPU Floorplan

Local Unified Buffer for Activations
(96Kx256x8b = 24 MiB)
29% of chip

Matrix Multiply Unit
(256x256x8b = 64K MAC)
24%

Host Interf. 2%
Accumulators (4Kx256x32b = 4 MiB) 6%
Control 2%
Activation Pipeline 6%
PCle Interface 3%
Misc. I/O 1%

DRAM port ddr3 3%
TPU Block diagram
Final words
Conclusion

- Computer architecture is more important than you can ever imagine
- Being a “programmer” is easy. You need to know architecture a lot to be a “performance programmer”
  - Branch prediction
  - Cache
- Multicore era — to get your multithreaded program correct and perform well, you need to take care of coherence and consistency
- We’re now in the “dark silicon era”
  - Single-core isn’t getting any faster
  - Multi-core doesn’t scale anymore
  - We will see more and more ASICs
  - You need to write more “system-level” programs to use these new ASICs.