Multi-core: The Ultimate Dose of Moore's Law

Mainak Chaudhuri
Dept. of Computer Science and Engineering,
IIT Kanpur

[A tale of 50+ years of glory and success]

Trends in Chip Industry

- Long history since 1971
  - Introduction of Intel 4004
  - http://www.intel4004.com/
- Today we talk about billions of transistors on a die
  - Intel's 24-core Core i9 processors (Core Ultra family) have 20+ billion transistors
  - AMD's Ryzen 9 family of processors have close to 20 billion transistors
  - IBM's POWER11 processor has 30 billion transistors
    - Last month IBM announced the first sub-1 nm (0.7 nm) chip that can pack 100 billion transistors

Trends in Chip Industry

- Today we talk about billions of transistors on a die
  - Nvidia's GeForce RTX 5090 (Blackwell architecture) has 90+ billion transistors
  - AMD's Radeon RX 9070 (RDNA 4 architecture) has 50+ billion transistors
  - Minimum feature size has shrunk from 10 micron in 1971 to 0.0007 micron today

Agenda

- Unpipelined microprocessors
- Pipelining: simplest form of ILP
- Out-of-order execution: more ILP
- Multiple issue: drink more ILP
- Scaling issues and Moore's Law
- Why multi-core
  - TLP and de-centralized design
- Tiled CMP and shared cache
- Connection with GPUs
- Implications on software

Unpipelined Microprocessors

- Typically an instruction enjoys five phases in its life
  - Instruction fetch from memory
  - Instruction decode and operand register read
  - Execute
  - Data memory access
  - Register write
- Unpipelined execution would take a long single cycle or multiple short cycles
  - Only one instruction inside processor at any point in time

Pipelining

- One simple observation
  - Exactly one piece of hardware is active at any point in time
- Why not fetch a new instruction every cycle?
  - Five instructions in five different phases
  - Throughput increases five times (ideally)
- Bottom-line is
  - If consecutive instructions are independent, they can be processed in parallel
  - The first form of instruction-level parallelism (ILP)

Pipelining Hazards

- Instruction dependence limits achievable parallelism
  - Control and data dependence (aka hazards)
- Finite amount of hardware limits achievable parallelism
  - Structural hazards
- Control dependence
  - On average, every fifth instruction is a branch (coming from if-else, for, do-while,…)
  - Branches execute in the third phase
    - Introduces bubbles unless you are smart

Control Dependence

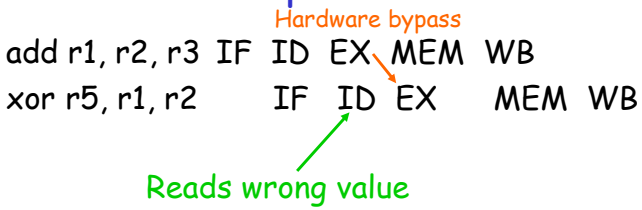
Branch	IF	ID	EX	MEM	WB	
Instr. X		IF	ID	EX	MEM	WB
Instr. Y			IF	ID	EX	…
Target				IF	ID	…

```
if (cond) { … }
else { … }
```

What do you fetch in X and y slots?

Options: nothing, fall-through, learn past history and predict (today best predictors achieve very high accuracy, often more than 95%)

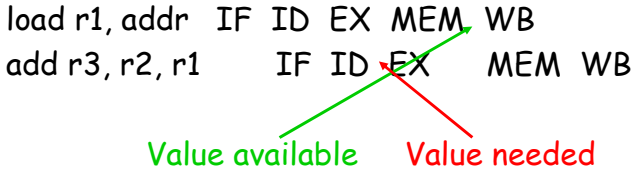
Data Dependence



Take three bubbles?

- Back-to-back dependence is too frequent
- Solution: hardware bypass paths
  - Allow the ALU to bypass the produced value in time: not always possible

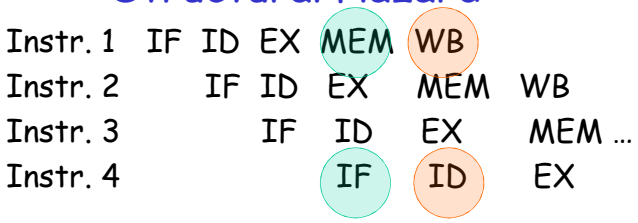
Data dependence



Need a live bypass! (requires some negative time travel: not yet feasible in real world)

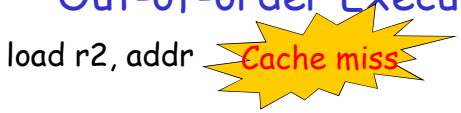
- No option but to take one bubble
- Bigger problems: load latency is often high; you may not find the data in cache

Structural Hazard



Usual solution is to put more resources

Out-of-order Execution




```
add r3, r2, r1
xor r10, r5, r3
sub  r9, r10, r1
addi r29, r29, 0xffff
sll  r29, r29, 2
mul r20, r20
```

Results must become visible in-order

Multiple Issue

load r2, addr


```
add r3, r2, r1
xor r10, r5, r3
sub  r9, r10, r1
addi r29, r29, 0xffff
sll  r29, r29, 2
mul r20, r20
```

Results must become visible in-order

Out-of-order Multiple Issue

- Some hardware nightmares
  - Complex issue logic to discover independent instructions
  - Increased pressure on cache
    - Impact of a cache miss is much bigger now in terms of lost opportunity
    - Various speculative techniques are in place to "ignore" the slow and stupid memory
  - Increased impact of control dependence
    - Must feed the processor with multiple correct instructions every cycle
    - One cycle of bubble means lost opportunity of multiple instructions
  - Complex logic to verify

Moore's Law

- Number of transistors on-chip doubles every 12 months (1965); revised in 1975 to every 24 months
  - Phenomenal 58% performance growth every year till late-nineties made possible by the increasing number of on-chip transistors
- Doesn't power consumption increase with increasing number of transistors?
  - Dynamic power consumption is proportional to the number of switching transistors
  - For a transistor, dynamic power is CV^2f
    - C=capacitance, V=voltage, f=frequency

Dennard scaling

- How power density scales with technology generation (due to Robert Dennard, 1974)
  - In each generation, transistor length shrinks by 30% i.e., new size is 0.7x of old size
    - 180 nm → 130 nm → 90 nm → 65 nm → 45 nm → 32 nm → 22 nm → 14 nm → 10 nm → 7 nm → 5 nm
  - New area per transistor is 0.5x of old area
    - Allows doubling of transistor count in same area
  - New capacitance per transistor is 0.7x of old
    - C is proportional to area/length
  - To maintain same electric field, voltage must be 0.7x because V = E.d

Dennard scaling

- How power density scales with technology generation (due to Robert Dennard, 1974)
  - RC delay through transistor becomes 0.7x as it is proportional to capacitance
    - Clock cycle time can be reduced to 0.7x
    - Frequency can be increased to (1/0.7)x i.e., 1.4x
  - Power consumption per transistor becomes (0.7*0.7*0.7*1.4)x or roughly 0.5x
  - Since area per transistor becomes 0.5x, power per unit area i.e., power density remains unchanged (known as Dennard scaling)

Dennard scaling

- Since 2x transistors can be packed in the same area Moore's law does not pose any problem with power consumption
  - But Dennard scaling ignores static power consumption
    - Power consumed even when a transistor is not switching i.e., off
    - Arises because a transistor does not switch off completely even if a low voltage is applied
    - Static power increases exponentially with decreasing transistor size
    - Today with very small transistors, Dennard scaling does not hold any more
    - Moore's law meets the power wall

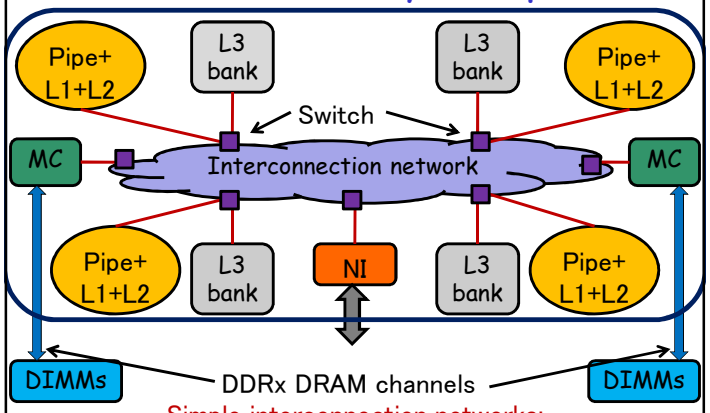
Scaling Issues

- Wires connect transistors
- Wire delay does not scale like transistors
- Hardware for extracting ILP has reached the point of diminishing return
  - Need a large number of in-flight instructions
  - Supporting such a large population inside the chip requires power-hungry delay-sensitive logic and storage
  - Verification complexity is large
- How to exploit so many transistors?
  - Must be a de-centralized design which avoids long wires

Multi-core

- Put a few reasonably complex processors or many simple processors on the chip
  - Each processor has its own primary cache and pipeline
  - Often a processor is called a core
  - Often called a chip-multiprocessor (CMP)
  - What about power consumption?
    - Trade-off between clock frequency, core count, core complexity
- Did we use the transistors properly?
  - Depends on if you can keep the cores busy
    - Introduces the concept of thread-level parallelism (TLP)

Floor of today's chips



Simple interconnection networks:
Linear bus, ring, 2D mesh (usually bidirectional)

Thread-level Parallelism

- Look for concurrency at a granularity coarser than instructions
  - Put a chunk of consecutive instructions together and call it a thread (largely wrong!)
  - Each thread can be seen as a "dynamic" subgraph of the sequential control-flow graph with data dependence edges added: take a loop and unroll its graph

```
for (i=0;i<10;i++) {
  a[i]++;
}
```

→

```
a[0]++;
a[1]++;
a[2]++;
...
a[9]++;
```

→

```
a[0]++;
...
a[4]++;
```

```
a[5]++;
...
a[9]++;
```


Thread-level Parallelism

- One more example

```
for (i=10;i<20;i++) {
  a[i] = a[i-2] + 1;
}
```

→

```
a[10] = a[8] + 1;
a[11] = a[9] + 1;
a[12] = a[10] + 1;
...
a[19] = a[17] + 1;
```

```
a[10] = a[8] + 1;
...
a[14] = a[12] + 1;
```

Data dependence edge

```
a[15] = a[13] + 1;
...
a[19] = a[17] + 1;
```


Thread-level Parallelism

- One more example

```
for (i=10;i<20;i++) {
  a[i] = a[i-2] + 1;
}
```

→

```
a[10] = a[8] + 1;
a[11] = a[9] + 1;
a[12] = a[10] + 1;
...
a[19] = a[17] + 1;
```

```
a[10] = a[8] + 1;
a[12] = a[10] + 1;
...
a[18] = a[16] + 1;
```

```
a[11] = a[9] + 1;
a[13] = a[11] + 1;
...
a[19] = a[17] + 1;
```

Thread-level Parallelism

- Look for concurrency at a granularity coarser than instructions
  - Put a chunk of consecutive instructions together and call it a thread (largely wrong!)
  - Each thread can be seen as a "dynamic" subgraph of the sequential control-flow graph with data dependence edges added: take a loop and unroll its graph
  - The edges spanning the subgraphs represent data/control dependence across threads
    - The goal of parallelization is to minimize such edges
    - Threads should mostly compute independently on different cores; but need to talk once in a while to get things done!

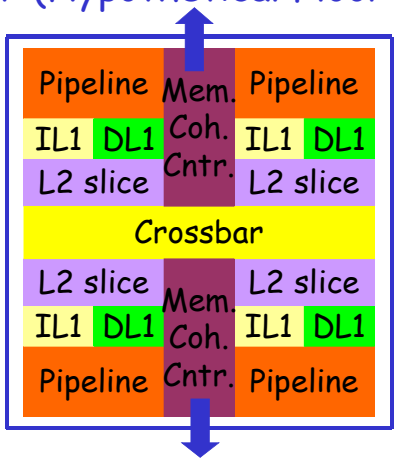
25

Thread-level Parallelism

- Parallelizing sequential programs is fun, but often tedious for non-experts
  - So look for parallelism at even coarser grain
  - Run multiple independent programs simultaneously
    - Known as multi-programming
    - Can play games while running heavy-weight simulations and downloading movies
    - Can browse internet while listening to music and running an anti-virus

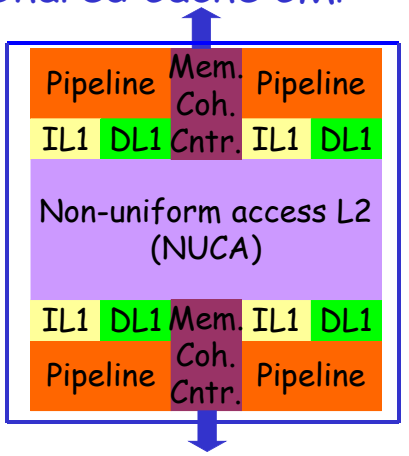
26

Tiled CMP (Hypothetical Floor-plan)



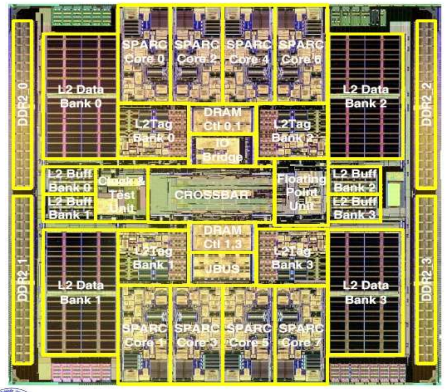
27

Shared Cache CMP



28

Niagara Floor-plan



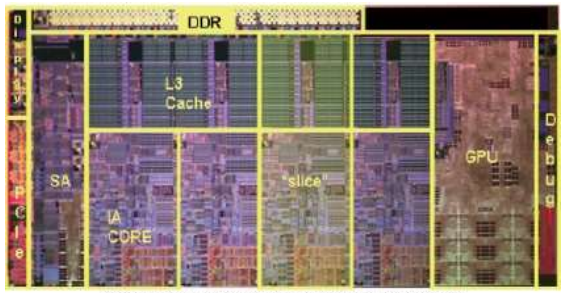
Features:

- Eight 64b Multithreaded SPARC Cores
- Shared 3MB L2 Cache
- 16KB ICache per Core
- 8KB DCache per Core
- Four 144b DDR-2 DRAM Interfaces (400 MTs)
- 3.2GB/s JBUS I/O
- Crypto: Public Key (RSA)
- Extensive RAS

Technology:

- 90nm CMOS Process
- 9LM Copper Interconnect
- Power: 63 Watts @ 1.2GHz
- Die Size: 378mm^2
- 279M Transistors
- Package: Flip-chip ceramic LGA (1933 pins)

Quad-core Sandy Bridge



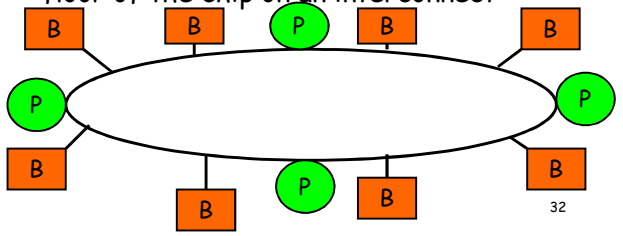
Sandy Bridge die photo (courtesy of ISSCC).

Communication in Multi-core

- Ideal for shared address space
  - Fast on-chip hardwired communication through cache (no OS intervention)
  - Two types of architectures
    - Tiled CMP: each core has its private cache hierarchy (no cache sharing); Intel Pentium D, Dual Core Opteron, Intel Montecito, Sun UltraSPARC IV, IBM Cell (more specialized)
    - Shared cache CMP: Outermost level of cache hierarchy is shared among cores; Intel Nehalem (4 and 8 cores), Intel Dunnington (6 cores), Intel Woodcrest (server-grade Core duo), Intel Conroe (Core2 duo for desktop), Sun Niagara, IBM Power4, IBM Power5

Shared vs. Private in CMPs

- Shared caches are often very large in the CMPs
  - They are banked to avoid worst-case wire delay
  - The banks are usually distributed across the floor of the chip on an interconnect



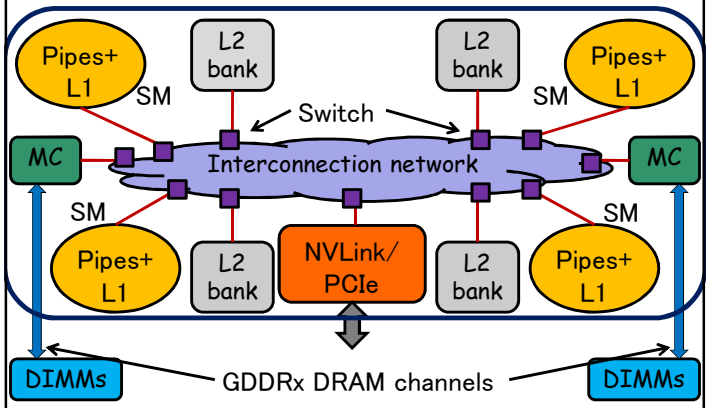
Shared vs. Private in CMPs

- In shared caches, getting a block from a remote bank takes time proportional to the physical distance between the requester and the bank
  - Non-uniform cache access (NUCA)
- This is same for private caches, if the data resides in a remote cache
- Shared cache may have higher average hit latency than the private cache
  - Hopefully most hits in the latter will be local
- Shared caches are most likely to have less misses than private caches
  - Replication of data and partitioning of space

Connection with GPUs

- GPUs are massive multi-core processors
  - Often referred to as many-core processors
- Let us look at GeForce RTX 5090 GPU
  - Architecture family: Blackwell, GB202
  - 170 cores (called streaming multiprocessors)
  - Each streaming multiprocessor has
    - 128 32-bit integer/floating point units (called CUDA cores); can work on four 32-entry vectors
    - Four tensor cores, one ray-tracing core
    - Four texture mapping cores
    - Four L0 instruction caches, one 128 KB L1 cache
    - Four hardwired thread schedulers
    - 65536 registers (4 banks x 16384 per bank)
  - 96 MB shared L2 cache (banked, NUCA)

Floor of a GPU (schematic)



Connection with GPUs

- Each streaming multiprocessor works on four 32-entry vectors
  - In a vector, each element is operated on by a common instruction
    - Single instruction operating on multiple data points in parallel: single instruction multiple data (SIMD) paradigm of vector processing
  - How does an instruction operate on multiple data elements in parallel?
    - Assign a thread to operate on a data element
    - 32 threads per vector each executing the same instruction in parallel: single instruction assigned to multiple threads
    - Single instruction multiple thread (SIMT) paradigm of vector processing

Connection with GPUs

- Each streaming multiprocessor works on four 32-entry vectors
  - 32 threads operating in lock-step form a scheduling entity called warp
    - Think of scheduling a group of 32 threads together
    - Warp scheduler is hardwired (in fact, there is no OS running on the GPU)
    - A warp waiting for a long-latency memory operation must be descheduled and a ready warp can be scheduled in its place
    - A large number of warps would be active at a time inside a GPU of today: context switch is costly
    - GPUs do not switch contexts of warps
      - A large number of registers and a bank of program counters needed to support many in-flight warps

Connection with GPUs

- Each CUDA core has a simple design
  - To keep power consumption in check
  - In-order single-fetch single-issue pipeline
    - Each L0 instruction cache produces the next instruction of a warp
      - The instruction is broadcast or issued to 32 CUDA cores after decoding
    - Four warp instructions are fetched in a streaming multiprocessor from four L0 instruction caches
      - Four pipelines per streaming multiproc. (superscalar)
    - Each pipeline supports all data forwarding paths
    - No branch predictors
      - Predication is used to convert control dependence into data dependence for short-range control dependence
      - Warp scheduling to hide other branch resolution stalls
    - No out-of-order or multi-issue execution: memory stalls are hidden through warp scheduling

Implications on Software

- A tall memory hierarchy
  - Within a streaming multiprocessor, threads communicate through private L1 cache (fastest)
  - Across streaming multiprocessors communication happens through shared L2 cache
  - Multiple GPUs can be connected over a scalable network
    - They can communicate between their memories
    - Adds one more level of memory hierarchy
- A very non-uniform access latency stack