Implicit vs. Explicit Parallelism

- Implicit
  - Hardware
    - Superscalar Processors
  - Compiler
    - Explicitly Parallel Architectures

- Explicit
Outline

- Implicit Parallelism: Superscalar Processors
- Explicit Parallelism
- Shared Instruction Processors
- Shared Sequencer Processors
- Shared Network Processors
- Shared Memory Processors
- Multicore Processors
Implicit Parallelism: Superscalar Processors

- Issue varying numbers of instructions per clock
  - statically scheduled
    - using compiler techniques
    - in-order execution
  - dynamically scheduled
    - Extracting ILP by examining 100’s of instructions
    - Scheduling them in parallel as operands become available
    - Rename registers to eliminate anti dependences
    - out-of-order execution
    - Speculative execution
Pipelining Execution

IF: Instruction fetch
ID: Instruction decode
EX: Execution
WB: Write back

Cycles

<table>
<thead>
<tr>
<th>Instruction #</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instruction i</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>WB</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>WB</td>
</tr>
<tr>
<td>Instruction i+1</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>WB</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>WB</td>
</tr>
<tr>
<td>Instruction i+2</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>WB</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>WB</td>
</tr>
<tr>
<td>Instruction i+3</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>WB</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>WB</td>
</tr>
<tr>
<td>Instruction i+4</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>WB</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>WB</td>
</tr>
</tbody>
</table>
# Super-Scalar Execution

## 2-issue super-scalar machine

<table>
<thead>
<tr>
<th>Instruction type</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>WB</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
</tr>
<tr>
<td>Floating point</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>WB</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
</tr>
<tr>
<td>Integer</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>WB</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
</tr>
<tr>
<td>Floating point</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>WB</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
</tr>
<tr>
<td>Integer</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>WB</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
</tr>
<tr>
<td>Floating point</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>WB</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
</tr>
</tbody>
</table>

Prof. Saman Amarasinghe, MIT.
Data Dependence and Hazards

- Instr\textsubscript{J} is data dependent (aka true dependence) on Instr\textsubscript{I}.

\[
\begin{align*}
I: & \quad \text{add } r1, r2, r3 \\
J: & \quad \text{sub } r4, r1, r3
\end{align*}
\]

- If two instructions are data dependent, they cannot execute simultaneously, be completely overlapped or execute in out-of-order.

- If data dependence caused a hazard in pipeline, called a Read After Write (RAW) hazard.
ILP and Data Dependencies, Hazards

- HW/SW must preserve program order: order instructions would execute in if executed sequentially as determined by original source program
  - Dependences are a property of programs
- Importance of the data dependencies
  - 1) indicates the possibility of a hazard
  - 2) determines order in which results must be calculated
  - 3) sets an upper bound on how much parallelism can possibly be exploited
- Goal: exploit parallelism by preserving program order only where it affects the outcome of the program
Name Dependence #1: Anti-dependence

- **Name dependence:** when 2 instructions use same register or memory location, called a name, but no flow of data between the instructions associated with that name; 2 versions of name dependence
- **Instr\_j writes operand** *before* Instr\_i reads it

\[
\begin{align*}
I: & \text{ sub } r4, r1, r3 \\
J: & \text{ add } r1, r2, r3 \\
K: & \text{ mul } r6, r1, r7
\end{align*}
\]

Calculated an “anti-dependence” by compiler writers. This results from reuse of the name “r1”

- If anti-dependence caused a hazard in the pipeline, called a Write After Read (WAR) hazard
Name Dependence #2: Output dependence

- Instr\(_J\) writes operand *before* Instr\(_I\) writes it.

\[
\begin{align*}
I &: \text{sub } r1, r4, r3 \\
J &: \text{add } r1, r2, r3 \\
K &: \text{mul } r6, r1, r7
\end{align*}
\]

- Called an “output dependence” by compiler writers. This also results from the reuse of name “r1”

- If anti-dependence caused a hazard in the pipeline, called a Write After Write (WAW) hazard

- Instructions involved in a name dependence can execute simultaneously if name used in instructions is changed so instructions do not conflict
  - Register renaming resolves name dependence for registers
  - Renaming can be done either by compiler or by HW
Control Dependencies

- Every instruction is control dependent on some set of branches, and, in general, these control dependencies must be preserved to preserve program order
  
  ```
  if p1 {
    S1;
  }
  if p2 {
    S2;
  }
  ```

- \( S_1 \) is control dependent on \( p_1 \), and \( S_2 \) is control dependent on \( p_2 \) but not on \( p_1 \).

- Control dependence need not be preserved
  - willing to execute instructions that should not have been executed, thereby violating the control dependences, if can do so without affecting correctness of the program

- Speculative Execution
Speculation

- Greater ILP: Overcome control dependence by hardware speculating on outcome of branches and executing program as if guesses were correct
  - Speculation ⇒ fetch, issue, and execute instructions as if branch predictions were always correct
  - Dynamic scheduling ⇒ only fetches and issues instructions
- Essentially a data flow execution model: Operations execute as soon as their operands are available
Speculation in Rampant in Modern Superscalars

- Different predictors
  - Branch Prediction
  - Value Prediction
  - Prefetching (memory access pattern prediction)

- Inefficient
  - Predictions can go wrong
  - Has to flush out wrongly predicted data
  - While not impacting performance, it consumes power
Today’s CPU Architecture:
Heat becoming an unmanageable problem

Cube relationship between the cycle time and power density.
Pentium-IV

- **Pipelined**
  - minimum of 11 stages for any instruction

- **Instruction-Level Parallelism**
  - Can execute up to 3 x86 instructions per cycle

- **Data Parallel Instructions**
  - MMX (64-bit) and SSE (128-bit) extensions provide short vector support

- **Thread-Level Parallelism at System Level**
  - Bus architecture supports shared memory multiprocessing
Outline

- Implicit Parallelism: Superscalar Processors
- **Explicit Parallelism**
  - Shared Instruction Processors
  - Shared Sequencer Processors
  - Shared Network Processors
  - Shared Memory Processors
- Multicore Processors
Explicit Parallel Processors

- Parallelism is exposed to software
  - Compiler or Programmer

- Many different forms
  - Loosely coupled Multiprocessors to tightly coupled VLIW
Little’s Law

Parallelism = Throughput * Latency

- To maintain throughput T/cycle when each operation has latency L cycles, need T*L independent operations
- For fixed parallelism:
  - decreased latency allows increased throughput
  - decreased throughput allows increased latency tolerance
Types of Parallelism

- **Data-Level Parallelism (DLP)**
- **Thread-Level Parallelism (TLP)**
- **Instruction-Level Parallelism (ILP)**

Prof. Saman Amarasinghe, MIT.
Translating Parallelism Types

Pipelining

Data Parallel

Thread Parallel

Instruction Parallel
Issues in Parallel Machine Design

- **Communication**
  - how do parallel operations communicate data results?

- **Synchronization**
  - how are parallel operations coordinated?

- **Resource Management**
  - how are a large number of parallel tasks scheduled onto finite hardware?

- **Scalability**
  - how large a machine can be built?
Flynn’s Classification (1966)

Broad classification of parallel computing systems based on number of instruction and data streams

- **SISD**: Single Instruction, Single Data
  - conventional uniprocessor

- **SIMD**: Single Instruction, Multiple Data
  - one instruction stream, multiple data paths
  - distributed memory SIMD (MPP, DAP, CM-1&2, Maspar)
  - shared memory SIMD (STARAN, vector computers)

- **MIMD**: Multiple Instruction, Multiple Data
  - message passing machines (Transputers, nCube, CM-5)
  - non-cache-coherent shared memory machines (BBN Butterfly, T3D)
  - cache-coherent shared memory machines (Sequent, Sun Starfire, SGI Origin)

- **MISD**: Multiple Instruction, Single Data
  - no commercial examples
My Classification

- By the level of sharing
  - Shared Instruction
  - Shared Sequencer
  - Shared Memory
  - Shared Network
Outline

- Implicit Parallelism: Superscalar Processors
- Explicit Parallelism
- Shared Instruction Processors
- Shared Sequencer Processors
- Shared Network Processors
- Shared Memory Processors
- Multicore Processors
Shared Instruction: SIMD Machines

- **Illiac IV (1972)**
  - 64 64-bit PEs, 16KB/PE, 2D network
- **Goodyear STARAN (1972)**
  - 256 bit-serial associative PEs, 32B/PE, multistage network
- **ICL DAP (Distributed Array Processor) (1980)**
  - 4K bit-serial PEs, 512B/PE, 2D network
- **Goodyear MPP (Massively Parallel Processor) (1982)**
  - 16K bit-serial PEs, 128B/PE, 2D network
- **Thinking Machines Connection Machine CM-1 (1985)**
  - 64K bit-serial PEs, 512B/PE, 2D + hypercube router
  - CM-2: 2048B/PE, plus 2,048 32-bit floating-point units
- **Maspar MP-1 (1989)**
  - 16K 4-bit processors, 16-64KB/PE, 2D + Xnet router
  - MP-2: 16K 32-bit processors, 64KB/PE
Shared Instruction: SIMD Architecture

- Central controller broadcasts instructions to multiple processing elements (PEs)

- Only requires one controller for whole array
- Only requires storage for one copy of program
- All computations fully synchronized
Cray-1 (1976)

- First successful supercomputers
Cray-1 (1976)

- Single Port Memory
- 16 banks of 64-bit words + 8-bit SECDED
- 80MW/sec data load/store
- 320MW/sec instruction buffer refill

64 Element Vector Registers

- V0, V1, V2, V3, V4, V5, V6, V7
- 8-bit SECDED
- 80MW/sec data load/store
- 320MW/sec instruction buffer refill

4 Instruction Buffers

64-bitx16

NIP → CIP

LIP

- memory bank cycle 50 ns
- processor cycle 12.5 ns (80MHz)

Prof. Saman Amarasinghe, MIT.
Vector Instruction Execution

Successive instructions

Cycles

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

IF ID EX WB

EX IF ID EX WB

EX IF ID EX WB
Vector Instruction Execution

VADD C, A, B

Execution using one pipelined functional unit


C[2]
C[1]
C[0]

Execution using four pipelined functional units

A[22] B[22]
A[27] B[27]

C[8]
C[4]
C[0]
C[5]
C[9]

C[10]
C[6]
C[11]
C[7]
C[3]
Vector Unit Structure

Vector Registers

Functional Unit

Memory Subsystem

Lane
Outline

- Implicit Parallelism: Superscalar Processors
- Explicit Parallelism
- Shared Instruction Processors
- **Shared Sequencer Processors**
- Shared Network Processors
- Shared Memory Processors
- Multicore Processors
**Shared Sequencer**

**VLIW: Very Long Instruction Word**

- Compiler schedules parallel execution
- Multiple parallel operations packed into one long instruction word
- Compiler must avoid data hazards (no interlocks)

### Shared Sequencer Diagram

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Two Integer Units, Single Cycle Latency**
- **Two Load/Store Units, Three Cycle Latency**
- **Two Floating-Point Units, Four Cycle Latency**
VLIW Instruction Execution

Cycles

1 2 3 4 5 6

IF ID EX WB
EX
EX

Successive instructions

VLIW execution with degree = 3
ILP Datapath Hardware Scaling

- Replicating functional units and cache/memory banks is straightforward and scales linearly.
- Register file ports and bypass logic for N functional units scale quadratically (N*N).
- Memory interconnection among N functional units and memory banks also scales quadratically.
- (For large N, could try O(N logN) interconnect schemes).
- Technology scaling: Wires are getting even slower relative to gate delays.
- Complex interconnect adds latency as well as area.

=> Need greater parallelism to hide latencies.
Clustered VLIW

- Divide machine into clusters of local register files and local functional units
- Lower bandwidth/higher latency interconnect between clusters
- Software responsible for mapping computations to minimize communication overhead
Outline

- Implicit Parallelism: Superscalar Processors
- Explicit Parallelism
- Shared Instruction Processors
- Shared Sequencer Processors
- **Shared Network Processors**
- Shared Memory Processors
- Multicore Processors
Shared Network: Message Passing MPPs
(Massively Parallel Processors)

- Initial Research Projects
  - Caltech Cosmic Cube (early 1980s) using custom Mosaic processors

- Commercial Microprocessors including MPP Support
  - Transputer (1985)

- Standard Microprocessors + Network Interfaces
  - Intel Paragon (i860)
  - TMC CM-5 (SPARC)
  - Meiko CS-2 (SPARC)
  - IBM SP-2 (RS/6000)

- MPP Vector Supers
  - Fujitsu VPP series

Designs scale to 100s or 1000s of nodes
Message Passing MPP Problems

- All data layout must be handled by software
  - cannot retrieve remote data except with message request/reply

- Message passing has high software overhead
  - early machines had to invoke OS on each message (100μs-1ms/message)
  - even user level access to network interface has dozens of cycles overhead (NI might be on I/O bus)
  - sending messages can be cheap (just like stores)
  - receiving messages is expensive, need to poll or interrupt
Outline

- Implicit Parallelism: Superscalar Processors
- Explicit Parallelism
- Shared Instruction Processors
- Shared Sequencer Processors
- Shared Network Processors
- **Shared Memory Processors**
- Multicore Processors
Shared Memory: Shared Memory Multiprocessors

- Will work with any data placement (but might be slow)
  - can choose to optimize only critical portions of code
- Load and store instructions used to communicate data between processes
  - no OS involvement
  - low software overhead
- Usually some special synchronization primitives
  - fetch&op
  - load linked/store conditional
- In large scale systems, the logically shared memory is implemented as physically distributed memory modules
- Two main categories
  - non cache coherent
  - hardware cache coherent
Shared Memory: Shared Memory Multiprocessors

- No hardware cache coherence
  - IBM RP3
  - BBN Butterfly
  - Cray T3D/T3E
  - Parallel vector supercomputers (Cray T90, NEC SX-5)

- Hardware cache coherence
  - many small-scale SMPs (e.g. Quad Pentium Xeon systems)
  - large scale bus/crossbar-based SMPs (Sun Starfire)
  - large scale directory-based SMPs (SGI Origin)
Cray T3E

- Up to 2048 600MHz Alpha 21164 processors connected in 3D torus

- Each node has 256MB-2GB local DRAM memory
- Load and stores access global memory over network
- Only local memory cached by on-chip caches
- Alpha microprocessor surrounded by custom “shell” circuitry to make it into effective MPP node. Shell provides:
  - multiple stream buffers instead of board-level (L3) cache
  - external copy of on-chip cache tags to check against remote writes to local memory, generates on-chip invalidates on match
  - 512 external E registers (asynchronous vector load/store engine)
  - address management to allow all of external physical memory to be addressed
  - atomic memory operations (fetch&op)
  - support for hardware barriers/eureka to synchronize parallel tasks
HW Cache Coherency

- **Bus-based Snooping Solution**
  - Send all requests for data to all processors
  - Processors snoop to see if they have a copy and respond accordingly
  - Requires broadcast, since caching information is at processors
  - Works well with bus (natural broadcast medium)
  - Dominates for small scale machines (most of the market)

- **Directory-Based Schemes**
  - Keep track of what is being shared in 1 centralized place (logically)
  - Distributed memory => distributed directory for scalability (avoids bottlenecks)
  - Send point-to-point requests to processors via network
  - Scales better than Snooping
  - Actually existed BEFORE Snooping-based schemes
Bus-Based Cache-Coherent SMPs

- Small scale (<= 4 processors) bus-based SMPs by far the most common parallel processing platform today
- Bus provides broadcast and serialization point for simple snooping cache coherence protocol
- Modern microprocessors integrate support for this protocol
Sun Starfire (UE10000)

- Up to 64-way SMP using bus-based snooping protocol

4 processors + memory module per system board

Uses 4 interleaved address busses to scale snooping protocol

Separate data transfer over high bandwidth crossbar

Prof. Saman Amarasinghe, MIT.
SGI Origin 2000

- Large scale distributed directory SMP
- Scales from 2 processor workstation to 512 processor supercomputer

Node contains:
  - Two MIPS R10000 processors plus caches
  - Memory module including directory
  - Connection to global network
  - Connection to I/O

Scalable hypercube switching network supports up to 64 two-processor nodes (128 processors total)
(Some installations up to 512 processors)
Outline

- Implicit Parallelism: Superscalar Processors
- Explicit Parallelism
- Shared Instruction Processors
- Shared Sequencer Processors
- Shared Network Processors
- Shared Memory Processors
- **Multicore Processors**
Phases in “VLSI” Generation

Bit-level parallelism  Instruction-level  Thread-level

Transistors

1,000 10,000 100,000 1,000,000 10,000,000 100,000,000


multicore

i4004  i8008  i80286  i8086  i80386  i80286  Pentium  R2000  R3000  R10000
Multicores

● Shared Memory
  ■ Intel Yonah, AMD Opteron
  ■ IBM Power 5 & 6
  ■ Sun Niagara

● Shared Network
  ■ MIT Raw
  ■ Cell

● Crippled or Mini cores
  ■ Intel Tflops
  ■ Picochip
Shared Memory Multicores:
Evolution Path for Current Multicore Processors

- **IBM Power5**
  - Shared 1.92 Mbyte L2 cache

- **AMD Opteron**
  - Separate 1 Mbyte L2 caches
  - CPU0 and CPU1 communicate through the SRQ

- **Intel Pentium 4**
  - “Glued” two processors together
CMP: Multiprocessors On One Chip

- By placing multiple processors, their memories and the IN all on one chip, the latencies of chip-to-chip communication are drastically reduced.
  - ARM multi-chip core

Diagram:
- Interrupt Distributor
- Configurable # of hardware intr
- Private IRQ
- Per-CPU aliased peripherals
- Configurable between 1 & 4 symmetric CPUs
- Private peripheral bus
- Primary AXI R/W 64-b bus
- Optional AXI R/W 64-b bus

Prof. Saman Amarasinghe, MIT.
Shared Network Multicores: The MIT Raw Processor

- 16 Flops/ops per cycle
- 208 Operand Routes / cycle
- 2048 KB L1 SRAM
Raw’s three on-chip mesh networks

(225 Gb/s @ 225 Mhz)

8 32-bit buses

Registered at input $\rightarrow$ longest wire = length of tile
Shared Network Multicore: The Cell Processor

- IBM/Toshiba/Sony joint project - 4-5 years, 400 designers
  - 234 million transistors, 4+ Ghz
  - 256 Gflops (billions of floating pointer operations per second)
- One 64-bit PowerPC processor
  - 4+ Ghz, dual issue, two threads
  - 512 kB of second-level cache
- Eight Synergistic Processor Elements
  - Or “Streaming Processor Elements”
  - Co-processors with dedicated 256kB of memory (not cache)
- IO
  - Dual Rambus XDR memory controllers (on chip)
    - 25.6 GB/sec of memory bandwidth
  - 76.8 GB/s chip-to-chip bandwidth (to off-chip GPU)
Mini-core Multicores: PicoChip Processor

- Array of 322 processing elements
- 16-bit RISC
- 3-way LIW
- 4 processor variants:
  - 240 standard (MAC)
  - 64 memory
  - 4 control (+ instr. mem.)
  - 14 function accelerators
Conclusions

● Era of programmers not caring about what is under the hood is over
● A lot of variations/choices in hardware
● Many will have performance implications
● Understanding the hardware will make it easier to make programs get high performance
● A note of caution: If program is too closely tied to the processor → cannot port or migrate
  ■ back to the era of assembly programming