Lect. 10: Vector and SIMD Processors

- Many real-world problems, especially in science and engineering, map well to computation on arrays
- RISC approach is inefficient:
  - Based on loops → require dynamic or static unrolling to overlap computations
  - Indexing arrays based on arithmetic updates of induction variables
  - Fetching of array elements from memory based on individual, and unrelated, loads and stores
  - Instruction dependences must be identified for each individual instruction
- Idea:
  - Treat operands as whole vectors, not as individual integer or float-point numbers
  - Single machine instruction now operates on whole vectors (e.g., a vector add)
  - Loads and stores to memory also operate on whole vectors
  - Individual operations on vector elements are independent and only dependences between whole vector operations must be tracked
Execution Model

for (i=0; i<64; i++)
    a[i] = b[i] + s;

- **Straightforward RISC code:**
  - F2 contains the value of s
  - R1 contains the address of the first element of a
  - R2 contains the address of the first element of b
  - R3 contains the address of the last element of a + 8

**loop:**

L.D F0,0(R2) ;F0=array element of b
ADD.D F4,F0,F2 ;main computation
S.D F4,0(R1) ;store result
DADDUI R1,R1,8 ;increment index
DADDUI R2,R2,8 ;increment index
BNE R1,R3,loop ;next iteration
Execution Model

for (i=0; i<64; i++)
    \( a[i] = b[i] + s; \)

- Straightforward vector code:
  - \( F2 \) contains the value of \( s \)
  - \( R1 \) contains the address of the first element of \( a \)
  - \( R2 \) contains the address of the first element of \( b \)
  - Assume vector registers have 64 double precision elements

```assembly
LV   V1,R2 ;   V1=array b
ADDVS.D V2,V1,F2 ; main computation
SV   V2,R1;   store result
```

- Notes:
  - In practice vector registers are not of the exact size of the arrays
  - Only 3 instructions executed compared to 6*64=384 executed in the RISC
Execution Model (Pipelined)

- In practice, the vector units takes several cycles to operate on each element, but is pipelined.
- With multiple vector units, I2 can execute together with I1.
Pros of Vector Processors

- Reduced pressure on instruction fetch
  - Fewer instructions are necessary to specify the same amount of work
- Reduced pressure on instruction issue
  - Reduced number of branches alleviates branch prediction
  - Much simpler hardware for checking dependences
- More streamlined memory accesses
  - Vector loads and stores specify a regular access pattern
  - High latency of initiating memory access is amortized
Cons of Vector Processors

- Still requires a traditional scalar unit (integer and FP) for the non-vector operations
- Difficult to maintain precise interrupts (can’t rollback all the individual operations already completed)
- Compiler or programmer has to vectorize programs
- Not suitable/efficient for many different classes of applications
- Requires a specialized, high-bandwidth, memory system
  - Usually built around heavily banked memory with data interleaving
SIMD Processors: Original Idea

- Network of simple processing elements (PE)
  - PEs operate in lockstep under the control of a master sequencer
  - PEs can exchange results with a small number of neighbours via special data-routing instructions
  - Each PE has its own local memory
  - Very large (up to 64K) number of PEs
  - Usually operated as co-processors with a host computer to perform I/O and to handle external memory

- Intended for use as supercomputers

- Programmed via custom extensions of common HLL
Original SIMD Idea

Instr. Sequencer

[Diagram showing the original SIMD idea with multiple M units connected to a sequencer]

CS4/MSc Parallel Architectures - 2014-2015
Example: Equation Solver Kernel

- The problem:
  - Operate on a \((n+2)\times(n+2)\) matrix
  
  \[
  \]

- SIMD implementation:
  - Assign one node to each PE
  - Step 1: all PE’s send their data to their east neighbours and simultaneously read the data sent by their west neighbours
  - Steps 2 to 4: same as step 1 for west, south, and north (again, appropriate nodes are masked out)
  - Step 5: all PE’s compute the new value using equation above
Multimedia SIMD Extensions

- Key ideas:
  - No network of processing elements, but an array of ALU’s
  - No memories associated with ALU’s, but a pool of relatively wide (64 to 128 bits) registers that store several narrower operands
  - No direct communication between ALU’s, but via registers and with special shuffling/permutation instructions
  - Not co-processors or supercomputers, but tightly integrated into CPU pipeline
  - Still lockstep operation of ALU’s
Graphics Processing Unit (GPU)

- Graphics apps have lot of parallelism
- Take advantage of hardware invested to do graphics well
- GPU is now ubiquitous!
- Several supercomputers in top500 use GPUs
  - Titan uses Nvidia tesla
CPU-style Core

Slide from Beyond programmable shading course ACM Siggraph ‘10
CS4/MSc Parallel Architectures - 2014-2015
Slimming down...

Idea #1: Remove components that help a single instruction stream run fast
Add Cores...

16 cores = 16 simultaneous instruction streams

Slide from Beyond programmable shading course ACM Siggraph ‘10
CS4/MSc Parallel Architectures - 2014-2015
Add ALUs (SIMD)...

Idea #2:
Amortize cost/complexity of managing an instruction stream across many ALUs

SIMD processing
Stalls!

- No caches

- No dynamic scheduling

- Dependencies and memory accesses can cause stalls!
Solution: Multithreading

But we have **LOTS** of independent fragments.

**Idea #3:**
Interleave processing of many fragments on a single core to avoid stalls caused by high latency operations.
16 cores

8 mul-add ALUs per core (128 total)

16 simultaneous instruction streams

64 concurrent (but interleaved) instruction streams

512 concurrent fragments

= 256 GFLOPs (@ 1GHz)
PhD in Pervasive Parallelism

http://pervasiveparallelism.inf.ed.ac.uk
Further Reading

- The first truly successful vector supercomputer:

- Vector processor on a chip:

- Integrating a vector unit with a state-of-the-art superscalar:
Further Reading

- **Seminal SIMD work:**

- **Two commercial SIMD supercomputers:**

- **SIMD co-processor:**