Vector Processors, Etc.

Some material from Appendix B of Hennessy and Patterson.

Outline

- Memory Latency Hiding v. Reduction
- Program Characteristics
- Vector Processors
- Data Prefetch
- Processor /DRAM Integration

Memory Latency Hiding v. Reduction

Memory Latency Hiding
Finding something else to do while waiting for memory.

Memory Latency Reduction
Reducing time operations must wait for memory.

Latency Hiding in Covered Architectures

- Superscalar (Hiding cache miss.)
  While waiting execute other instructions.

  Can only cover part of miss latency.
- Multithreaded
  While waiting execute other threads.

  Can cover full latency.

  Requires effort to parallelize code.

  Parallel code possibly less efficient.
- Multiprocessor
  Latency problem worse due to coherence hardware and distributed memory.

Memory Latency Reduction

Approaches

- Vector Processors
  ISA and hardware efficiently access regularly arranged data.

  Commercially available for many years.
- Software Prefetch
  Compiler or programmer fetch data in advance.

  Available in some ISAs.
- Hardware Prefetch
  Hardware fetches data in advance by guessing access patterns.
- Integrated Processor /DRAM
  Reduce latency by placing processor and memory on same chip.
- Active Messages
  Send operations to where data is located.

Code Characteristics

Some latency reduction uses advance knowledge of memory access.

Advance knowledge may predicted or part of program.

Feasibility depends on program characteristics.

Integer Programs

Examples: compiler. compression.

Few FP operations.

Short loops.

Difficult to predict branches.

Relatively low memory bandwidth.

Arbitrary access patterns. (Pointer chasing.)

Floating-Point Programs

Many FP operations.

Loops have many iterations.

Memory accessed in sequential and stride patterns.

High memory bandwidth.

Easy to predict branches.

Vector Processors

Vector Processor
A machine designed to efficiently execute code that manipulates vectors and similarly structured computations.

Commercially available for decades.

Example, Cray T90.

Approach

Code specifies operation on vectors . . .
. . . allowing hardware to execute at high rate . . .
. . . without cycle-stretching dynamic execution mechanisms.

High bandwidth memory supporting common access patterns.

Typical Characteristics

Includes registers that hold vectors (rather than single values).

Instructions to operate on vectors.

Instructions to load and store vectors without using a cache.

High-performance floating-point units.

High-bandwidth memory system.

High clock frequency.

A vector version of DLX, DLXV, will be described.

Vector Registers

Vector Register
A register designed to hold a vector.

Maximum length of vector is limited. (*E.g.*, 128 elements.)

Like integer and FP registers, vector ISA has multiple vector registers.

Number of elements in vector registers specified in special register.

DLXV has eight 64-element vector registers, V0-V7.

Vector Instructions

Vector-Vector Arithmetic Instructions

Three vector operands, perform $\vec{A} = \vec{B} \circ \vec{C}$, where $\circ$ is an arithmetic operation $(+, -, \div,)$ etc.

E.g.:

`addv v1, v2, v3 !` $\vec{v1} = \vec{v2} + \vec{v3}$.

`multv v1, v2, v3 !` $\vec{v1} = \vec{v2} \cdot \vec{v3}$.

Scalar-Vector Arithmetic Instructions

Two vector operands, perform $\vec{A} = b \circ \vec{C}$.

E.g.:

`addsv v1, f2, v3 !` $\vec{v1} = f2 + \vec{v3}$.

`addvs v1, v2, f3 !` $\vec{v1} = \vec{v2} + f3$.

Vector Instructions, Continued

Vector Load/Store Instructions

Load vector register using consecutive addresses:

Used when vector stored in consecutive locations.

`lv v1, r1 !` Load v1 with memory starting at r1.

Load vector register using stride:

Used when vector stored at regular stride.

E.g., 1000, 1040, 1080, …

`lvws v1, (r1,r2) !` Load v1 with values at r1, r1+r2, r1+2× r2, …

Load vector register using index:

Used when vector stored arbitrarily. Indices stored in vector.

`lvi v1, (r1+v2) !`

Load v1 with values at r1 + v2[0], r1+v2[1], r1+v2[2], …

Vector Instruction Use

Sample Code: DAXPY Loop in C

```
for(i=0; i<len; i++) y[i] = a * x[i] + y[i];
```

DAXPY Loop in DLXV (assuming `len` ≤ 64).

```
! Initially:
!  r4:  Vector length. (len in C code).
!  r20: Address of scalar a.
!  r10: Address of first element of vector x.
!  r11: Address of first element of vector y.
!
movi2s vlr,r4         ! Set vector length to r4.
ld     f0, 0(r20)     ! Load a from 0(r20).
lv     v1, r10        ! Load vector x. Starting address in r10.
multsv v2, f0, v1     ! v2 = a * x.
lv     v3, r11        ! Load vector y. Starting address in r11.
addv   v4, v2, v3     ! v4 = (a * x) + y
sv     r11, v4        ! Store completed vector.
```

Vector Processor Implementation

Typical Vector Processor Hardware

Conventional processor with vector hardware added:

- Fully pipelined vector floating-point functional units.
  FP units may generate more than one result per cycle …
  … but much less than a complete 64-element vector per cycle.
- Vector Registers.
- Vector Load/Store Unit.

Vector Processor Instruction Execution

Instruction Execution Common to Conventional Processor

Fetch, decode, and dispatch similar to conventional machine.

Execution of non-vector instructions similar to conventional machine . . .
. . . but to keep cycle times short scheduling is less aggressive.

Execution of Vector Instructions

Executed by vector functional units.

May read and write vector and ordinary registers.

Single instruction *executes for many cycles* and spans many pipeline segments.

DLXV Latencies

Load/Store Unit, 12 cycles.

Multiply Unit, 7 cycles.

Add Unit, 6 cycles.

Vector Processor Instruction Execution, Continued

Possible Execution Restrictions

Number of instructions issued (started) per cycle.

For simplicity possibly just one.

Data Dependencies

Dependencies more complex because of multiple elements in vector register.

Structural Hazards

Number of vector instructions that can simultaneously execute.

Number of register ports.

Vector Dependencies and Chaining

In conservative (and outdated) implementation . . .
. . . vector instruction may wait for preceding dependent vector instruction to complete . . .
. . . even though data for first element available much sooner.

```
addv   v1, v2, v3
multv  v4, v1, v5  ! Can start when first element of v1 available.
```

In modern vector machines execution of dependent instructions can overlap . . .
. . . by allowing vector registers to be read before they are completely written.

Such dependent execution is called *chaining*.

Vector Processor Memory Systems

Organization: Two paths from memory.

(1) Memory-Cache-Processor (Instructions accessing ordinary registers.)

(2) Memory-Processor (Instructions accessing vector registers)

Design goal: Sustained single-cycle access to vectors which aren't cached . . .
. . . and wouldn't fit in a cache if they were.

Memory Access Patterns

Conventional: Cache block fill. (Sequential access.)

Vector: read elements of vector. (Sometimes sequential.)

Memory System Organization

Single Bank

Memory can handle single access at a time.

Works like a single memory device.

Access time is number of bus widths needed times cycle time.

Interleaved Memory (Interleaved Banks)

Bank: One of a set of memories.

Memory divided into multiple *banks*.

Interleave Factor: Number of banks.

Consecutive (based on address) words in different banks.

Interleaved banks work together.

Banked Memory (Independent Banks)

Memory divided into multiple banks.

Consecutive words (based on address) in different banks.

Each bank can work on different part of address space (non-consecutive access).

Single Bank Timing

Single Bank Example.

Consider `lv v1, r1 !  Load 64-element vector.`

Find time that first, second, and last word loaded to vector register.

Timing in Example System

Load instruction start to access start: 4 cycles. (Can overlap.)

Memory access time: 24 cycles.

Memory cycle time: 30 cycles.

Data transfer time (memory to register): 4 cycles.

Bus width: one word (64 bits in vector machine).

Solution:

First word arrives: $4 + 24 + 4 = 32$.

Second word arrives: $32 + 30 = 62$.

ith word arrives: $32 + 30i$.

Last, 15th, word arrives: 482.

Interleaved Memory

Memory divided into banks.

Number of banks usually (but not always) power of 2.

Address space divided so consecutive words in different banks.

Let B denote number of banks.

Let W denote bytes per word.

Let A be a word-aligned address.

Storage for A is in bank $\lfloor A/W \rfloor \bmod B$.

Address within bank is $\lfloor A/(WB) \rfloor$.

Example

$B = 16$. (Sixteen banks.), $W = 8$. (Eight byte words.)

Address 0x1234 in bank 6, address 0x24.

Address 0x123c in bank 7, address 0x24.

Address 0x12bc in bank 7, address 0x25.

Interleaved Memory

Interleaved Memory Access Timing

Same address presented to all memories.

All memory banks retrieve data. (Bank 0, first word; bank 1, second; etc.)

Needed data placed on bus in order.

Access Timing

Access to first word no faster.

Access to second word (if needed) right after first.

Example: `lv v1, r1 !  Load 64-element vector.`

$B = 16$. (Sixteen banks.) $W = 8$. (Eight byte words.)

Instruction start to access start: 4 cycles, fully pipelined.

Access time: 24 cycles. Cycle time: 30 cycles.

Data transfer (memory to register) time: 4 cycles, fully pipelined.

Bus width: one word (64 bits).

Answer

First word: 4 + 24 + 4 = 32 cycles.

Second word: 32 + 1 = 33 cycles.

Fifteenth word: 32 + 14 = 46 cycles.

Sixteenth word: 32 + 30 = 62 cycles.

32nd word: 92 cycles.

48th word: 122 cycles.

63rd word: 122+14=136 cycles.

Banked Memory

Multiple memory banks.

Connected to processor using common bus, as with interleaving.

Address space divided as in interleaving.

Each bank can process independent address.

Useful for stride and arbitrary accesses.

Banked Memory Example

Stride Access to Banked Memory Example

The code below runs on a vector processor with banked memory in which:

$B = 16$. (Sixteen banks.), $W = 8$. (Eight byte words.)

```
! r1 = 0x1000 (address of first word).
! r2 = 0x20 (Stride of 32 bytes = 4 words.)
lvws v1, (r1,r2)  ! Load 64-element vector at stride
```

Banks to which accesses directed:

Address:	0x1000	0x1020	0x1040	0x1060	0x1080	0x10a0	0x10c0	⋯
Bank:	0	4	8	12	0	4	8	⋯

Bank Usage in Example

Only four out of 16 memory banks actually used.

Memory cycle time would have to be 4 cycles for full-speed transfer.

Stride Access Timing

As seen in example, stride determines how many banks used.

Determining number of banks used in stride access:

Let B denote number of banks and S denotes stride *in words*.

Number of banks used is

$$B_U = \frac{B}{\gcd(B,S)} = \frac{\operatorname{lcm}(B,S)}{S}$$

where $\gcd(B,S)$ is the greatest common denominator of B and S and $\operatorname{lcm}(B,S)$ is the least common multiple of B and S.

GCD and LCM

To find GCD use intersection of prime factors (repeated factors distinct):

Example: $\gcd(10,5) = \gcd(5 \times 2, 5) = 5$.

Example: $\gcd(16,4) = \gcd(2^4, 2^2) = 2^2 = 4$.

Example: $\gcd(77004, 3138) = \gcd(\underline{2} \times 2 \times \underline{3} \times 3 \times 3 \times 23 \times 31, \underline{2 \times 3} \times 523) = 2 \times 3 = 6$.

Example: $\gcd(7,11) = 1$

To find LCM use union of prime factors (repeated factors distinct):

Example: $\mathrm{lcm}(10,5) = \mathrm{lcm}(2 \times 5, 5) = 5 \times 2 = 10$.

Example: $\mathrm{lcm}(16,4) = \mathrm{lcm}(2^4, 2^2) = 2^4 = 16$.

Example: $\mathrm{lcm}(77004, 3138) = \mathrm{lcm}(\underline{2} \times 2 \times \underline{3} \times 3 \times 3 \times 23 \times 31, \underline{2 \times 3} \times 523) = 2^2 \times 3^3 \times 23 \times 31 \times 523 = 40,273,092$.

Stride Access Timing, Continued

Sustained Access Time

For a system where bus can transfer 1 word per cycle:

$$t_{\text{access}} = \max\left\{1, \frac{t_{\text{cycle}}}{B_U}\right\} = \max\left\{1, t_{\text{cycle}}\frac{S}{\mathrm{lcm}(B,S)}\right\}$$

For low access time want $\mathrm{lcm}(B,S)$ to be large.

If B and S are both powers of same number, such as 2, ...
... $\mathrm{lcm}(B,S)$ will be small.

$\mathrm{lcm}(B,S)$ highest when 1 is only common factor.

Choosing a prime number of banks reduces chance of common factor in B and S.

Prefetch

Prefetch
Bringing a block to a cache in advance of its need.

Prefetch Methods

- Prefetch Instructions.
  Inserted before load instructions that might miss.

  Execute as load, but does not change any registers.

  If inserted in proper place, data arrives just before load executes.

- Stream Buffers.
  Designed to hold consecutive or stride blocks.

  Set up by special instructions specifying start address and stride.

  Items removed from buffer when needed.

  Hardware keeps buffer full.

Prefetch, Continued

- Hardware Prefetch
  Hardware detects consecutive or stride access patterns.

  Block may be prefetched if preceding block accessed.

  If it works, consecutive accesses will only suffer one miss.