Day 3
Advanced Vector Architectures

Session A: Vector Instruction Execution Pipelines
Break
Session B: Vector Flag Processing & Vector Register Files
Lunch
Session C: Virtual Processor Caches
Break
Session D: Vector IRAM

Vector Instruction Execution Pipelines

Main issues:
- Hiding/Tolerating Memory Latency
- Handling Exceptions
- Avoiding complexity
Tolerating Memory Latency with Short Chimes

=> Can use same techniques as scalar processors:

Static Scheduling: Move Load Earlier

Hardware or Software Prefetch: Request Data Earlier

Dynamic Scheduling: Execute Add Later
(Decoupled Pipeline or Out-of-Order Execution [Espasa, PhD '97])

Vectors allow simple control logic to buffer 1000s of outstanding operations

(also multithreading with parallel threads)

Tolerating Memory Latency with Vectors

Traditional In-Order Vector Issue Pipeline

Decoupled Vector Pipeline (Espasa, PhD'97)

Also, full out-of-order issue is possible (Espasa, PhD'97)
Memory Latency and Short Vectors

Vector Instruction Sequence

VLD v1
VMUL v2, v1, r1
VLD v3
VMUL v4, v3, r2

Instruction Execution in Time

VLD v1 Address Mem Idle VLD v3 Address
VLD v1 Data VMUL v2, v1, r1 Mult Idle VMUL v4, v3, r2
VLD v3 Data

Cray-style

Decoupled Pipeline

Addr.Gen. Data Bus Multiplier

VLD v1 Address VLD v3 Address
VLD v1 Data VMUL v2, v1, r1 VMUL v4, v3, r2
VLD v3 Data

Decoupled Pipeline Issues

Latencies:
Decoupling hides memory latency in most cases but exposes latency in others.

Scalar Pipe

F D X M W

Instruction Queues

Scatter/Gather Indices
Load/Store Masks

Vector Load Pipe

A W

Vector Arithmetic Pipe

R X X X W

Exceptions:
- IEEE Floating-Point
- Page Faults for Virtual Memory
Vector IEEE Floating-Point Model

Vector FP arithmetic instructions never cause machine traps
- (Except in special debugging modes)
- IEEE default results handled without user-visible traps (unlike Alpha)
- Largest expense is hardware subnormal handling

Vector FP exceptions signaled by writes to vector flag registers
- Reserve 5 vector flag registers to receive exception information:
  Invalid, DivideByZero, Overflow, Underflow, InExact

User trap handlers: inline conditional code or trap barrier
- Use normal vector conditional execution to handle vector FP exceptions
- Explicit trap barrier instruction checks flags and takes precise trap

  Full IEEE support at full speed in deep vector pipeline

Short-Running Vector Instructions Simplify Virtual Memory

- Address translate/check (C) of whole vector takes only 4-8 clocks
- Overlap checks with memory latency - no added latency for VM
- Buffer following instructions until address check complete
- For in-order machine, short vectors limit size of state to save/restore
- For out-of-order machine, short vectors limit reorder buffer size
Instruction Queue Design

Delayed Pipeline

Replace queues with fixed length instruction pipeline:

Simpler than decoupled, no data buffers.
Works best for fixed latency memory with few collision.
Out-of-Order Vector Execution

Simpler than scalar out-of-order because of reduced instruction bandwidth.

Vector register renaming solves exception problem.

But problems in vector register renaming:
- Elements beyond vector length (change ISA to mark undefined)
- Masked elements (change ISA to leave undefined - requires merges)
- Scalar insert into vector register (Make it slow so programmers avoid this)

But maybe OOO not a big win with more vector registers, better vector compiler, decoupled pipeline.
(vector loops should be mostly statically schedulable)

OOO without vector register renaming may give small boost (put OOO after address commit)

Day 3, Session B
Vector Flag Processing Model
& Vector Register Files
Flags are more than Masks

Flags are used for:
- Conditional Execution (Mask Registers)
- Reporting Status (Popcount and Count Leading/Trailing Zeros)
- Exception Reporting (IEEE754 FP)
- Speculative Execution

Flag Priority Instructions

Goal: Avoid latency of scalar read-flags -> write-new-length
Approach: Generate mask vector with correct length

Reads flag register, writes flag register, three forms:

0 7  
0001 0110 Source flags
1000 0110 Flag-before-first (fbf)
1000 0010 Flag-including-first (fif)
1000 0000 Flag-only-first (fof)

Also, operation that compresses flag register

0001 0110 Source flags
1111 1110 Compress-flags (cpf)
Vector Register File Design

Construct high bandwidth VRF from multiple banks of less highly multiported memory.

Design decisions:
• form of bank partitioning
• number of banks versus ports/bank

Bank Partitioning Alternatives

Register Partitioned

Element Partitioned

Register and Element Partitioned
Example Vector Register File: 1 Lane

Multiported Storage Cells

(all designs double-pumped)
All designs provide 256 64-bit elements, and 5R+3W ports.

Day 3, Session C:
Virtual Processor (VP) Caches

Highly parallel primary caches for vector units

Reduce bandwidth demands on main memory

Convert strided and scatter/gather operations to unit-stride

Two forms:
- **Rake Cache (Spatial VP Cache)**
- **Histogram Cache (Temporal VP Cache)**
Many Useful Vector Algorithms use Virtual Processor Paradigm

Developed by Blelloch et al., CMU SCANDAL group:

- Sorting
- Sparse Matrix-Vector Multiply
- Connected Components
- Linear Recurrences
- List Ranking

But frequent scatter/gather and non-unit stride accesses

Address bandwidth expensive:

- Address Crossbars
- TLB ports
- Cache Transactions
- DRAM Page Breaks
Matrix-Vector Multiply

\[ C = A \times B \]

(Row-major matrix storage)

Strided vector accesses **but**
each virtual processor accesses unit-stride stream

Rake Cache

**KEY IDEA:**
Associate one (or more) cache line with each virtual processor

Advantages over shared cache:
- Access local to lane, lower energy and compact layout
- High-bandwidth without multiport or interleaved memories
- No inter-VP conflicts, power-of-2 stride OK!
Rake Cache for Matrix-Vector Multiply

\[ C = A \times B \]

With 4 word cache line, rake cache can reduce address bandwidth by up to 4x

Other Forms of Rake

1D Strided Rake

Indexed Rake (parallel structure access)
Rake Cache Design

Explicitly Selected and Indexed
• Strided and indexed instructions specify use of rake cache (and which line if more than one)

Non-coherent
• weak vector consistency model, flush at vector memory barrier instructions

Virtually Tagged
• reduces TLB accesses
• weak vector consistency model, no problem with synonyms

Per Byte Dirty Bits
• Avoids false sharing problem, only write-back modified bytes

Rake Cache Implementation

[Diagram of Rake Cache implementation with labels for VPN, PPN, V, D, Byte, Index, Base, Stride, Physical Page Number, Rake Cache Index, Page Hit?, Virtual Page Number, Line Hit?, Write Back, Physical Address Bus, Data Bus, etc.]

[Diagram notes: Single Rake Cache Line, Valid bit per line, Virtually Tagged, Per Byte Dirty Bit]
VP Cache Performance

Garbage Collection in SPECint95 li
- Rake cache
  removes 37% of address traffic

OpenGL Rasterization (Aaron Brown, UCB)
- Rake cache
  removes 68% of address traffic

Radix Sorting
- Rake cache + histogram cache
  removes 78% address traffic and 57% data traffic

IDEA encryption
- Rake cache
  removes 74% address traffic

All assuming 1KB rake cache and 32KB histogram cache

Day 3, Session D:
Vector IRAM
UC Berkeley
Profs. Patterson, Yelick, Kubiatowicz

http://iram.cs.berkeley.edu/
Key Observation

DRAM capacity increases 4x per generation (every 3 years)
DRAM cost/bit decreases 2x per generation

=> Fewer DRAMs per system at constant cost
IRAM

Put processor and DRAM main memory on same die
• Memory latency 5x-10x improvement
• Memory bandwidth 50-100x improvement
• Lower power
• Smaller board size

What Type of Processor for IRAM?
• Must be small to leave room for DRAM
• Must convert 100s GB/s memory bandwidth into application speedups
• Desire low power

=> Vector!

Vector IRAM: Pocket Supercomputer

Single die, 256Mb Merged DRAM/Logic Technology
200 MHz, ~2W

Serial I/O Lines
Scalar Unit + I/D Caches
Vector Unit

8 x 1Gb/s
12.8 GB/s
16-32MB DRAM

1.6 GFLOPS (64-bit)
3.2 GFLOPS (32-bit)
6.4 GOPS (16-bit)
**VIRAM-1 Block Diagram**

- Instruction Cache
- Flag Registers
- Scalar Unit
- VFFU0, VFFU1
- Vector Registers
- VAFU0, VAFU1
- VMFU0, VMFU1
- I/O
- DRAM
- Scalar Data Cache
- Write Buffer

**Vector Memory Subsystem**

Total Memory: 32MB
2 wings
X 8 banks
X 8 sub-banks
= 128 sub-banks.

Each sub-bank is
1K rows by 2K bits.
Each column access returns 256 bits.

Total random access memory latency is
12-15 clocks at 200MHz.
VMFU Questions

How many and what type of load/store units?

How many addresses per cycle per unit?

Vector Load and Store Components

<table>
<thead>
<tr>
<th>Load</th>
<th>Store</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generate Address(es)</td>
<td>Send Physical Address to DRAM</td>
</tr>
<tr>
<td>Translate Address(es)</td>
<td>Send Data to DRAM</td>
</tr>
<tr>
<td>Invalidate Scalar Data Cache</td>
<td></td>
</tr>
<tr>
<td>Receive Data from DRAM</td>
<td></td>
</tr>
<tr>
<td>Write Vector Register File</td>
<td></td>
</tr>
</tbody>
</table>
Two Wings Makes Two VMFUs Cheap

Already have two data and address crossbars.

At cost of second address generator, TLB port, two-way address multiplexer and extra vector register write port, can have second load-only VMFU.

VIRAM-1 has one load/store unit plus one load-only unit.

Unit-stride operations alternate between wings and so synchronize after first collision.

Supporting Fast Non-Unit Stride with Many Addresses Per Cycle Expensive

Need:
more address generators
+ more TLB ports
+ more scalar data cache ports (stores only)
+ more address crossbar
+ conflict detect logic
+ more data crossbar control
+ maybe more buffering to smooth out conflicts

=> Not obvious that more addresses/cycle best use of silicon
IRAM Memory Subsystem

Drive DRAM with Fixed Pipeline