CS294-4 IRAM, March 6, 1996

Vector Microprocessors

Krste Asanovic

krste@cs.berkeley.edu

http://www.cs.berkeley.edu/~krste

University of California at Berkeley
and the
International Computer Science Institute

Talk Outline

- Vector micro myths dispelled
- Inside T0
- Why vector micros are different from vector mainframes
- Vector IRAM possibilities

Vector Microprocessor Myths

- Not enough pin bandwidth
- Memory system too complex and expensive
- Not enough vectorizable applications

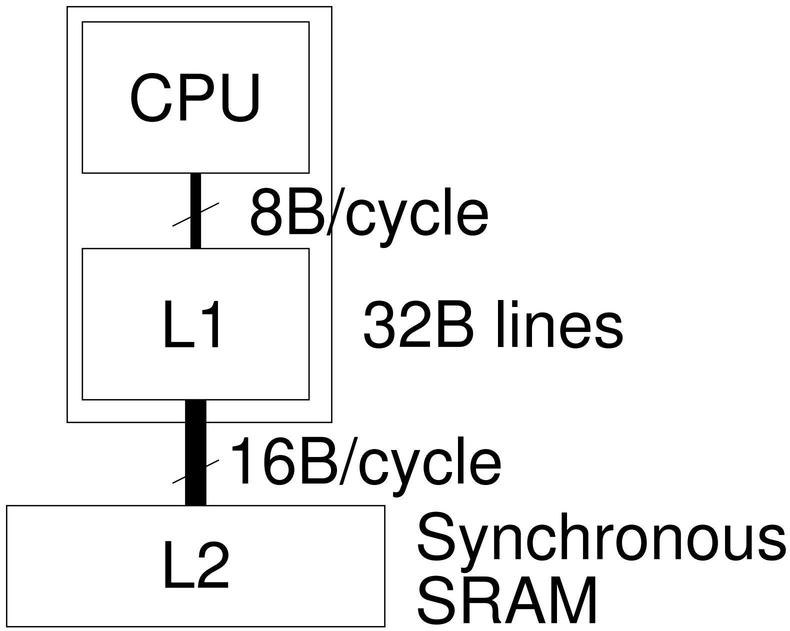
Cached CPUs waste pin and memory bandwidth

McCalpin's STREAM benchmark results:

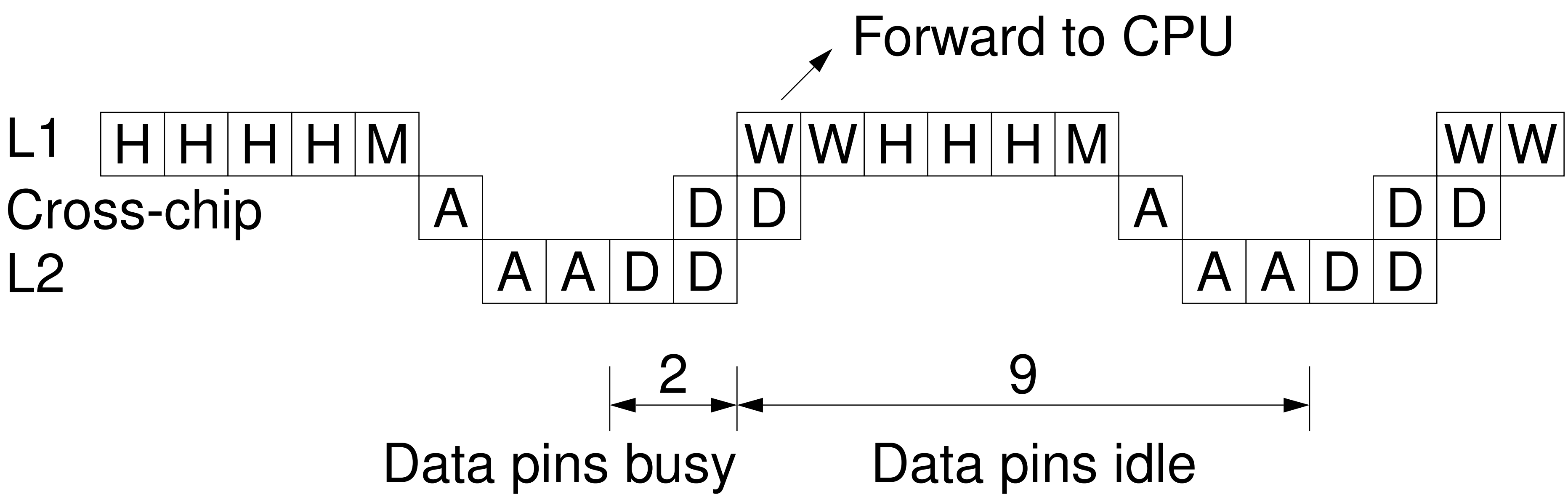
System	CPUs	Single CPU Data Pin BW	Maximum Aggregate Sustained Memory BW Measured	Maximum Single CPU Memory BW Measured
DEC 8400 "TurboLaser"	8 * 300 MHz Alpha 21164	1200	978.8	198.3
SGI Power Challenge	8 * 75 MHz R8000	1200	749.3	134.9
Cray CS6400 Superserver	32 * 60 MHz SuperSPARC	480	882.6	51.1

All bandwidths in MB/s.

Where does all the BW go?




```
DO 10 I = 1,N
  SUM = SUM + A(I)
10 CONTINUE
```



Why does a Cray T932 have 1024 memory banks?

32 processors
3 (128 bit) load/stores per processor per cycle
2.167ns processor cycle versus 15ns SRAM cycle

So, 32*3*15/2.167 = 665 banks minimum

Total CPU-Memory BW, 32*3*16B / 2.167ns > 700 GB/s

+ concurrent I/O!

Where did faster DRAMs come from?

Not from high-end desktop designers - we've seen that it wouldn't help much with 2-3 levels of cache in the way.

- PCs see ~5% performance boost with EDO DRAM (40 MHz column cycle) versus fast page mode DRAM (25 MHz column cycle). Any workstations with EDO or SDRAM memory yet?

Not from Cray's pleas - too small a market.

Faster DRAMs arose due to graphics card demands.

This is the key enabling technology for vector micros!

New Intel Triton VX chipset to support SDRAM for UMA.

Vector Micro Memory Subsystem

Some form of synchronous DRAM, e.g. SDRAM, MDRAM.

Clock rates equal to mid-range CPU, half of high-end CPU.

- Now: Samsung 16 Mb @153 MHz vs. DEC Alpha 21164 @ 333 MHz.
- ISSCC'96: NEC 1 Gb @ 250 MHz vs. DEC Alpha 21164A @ 433 MHz

CAS latency 3-4 clocks, fully pipelined.
Precharge + RAS latency 3-6 clocks, bank busy.

One bank, no interleaving, **NO PROBLEM!**

128-bit data bus @ 150 MHz yields 2.4 GB/s peak.
256-bit data bus @ 250 MHz yields 8.0 GB/s peak.

Application Examples

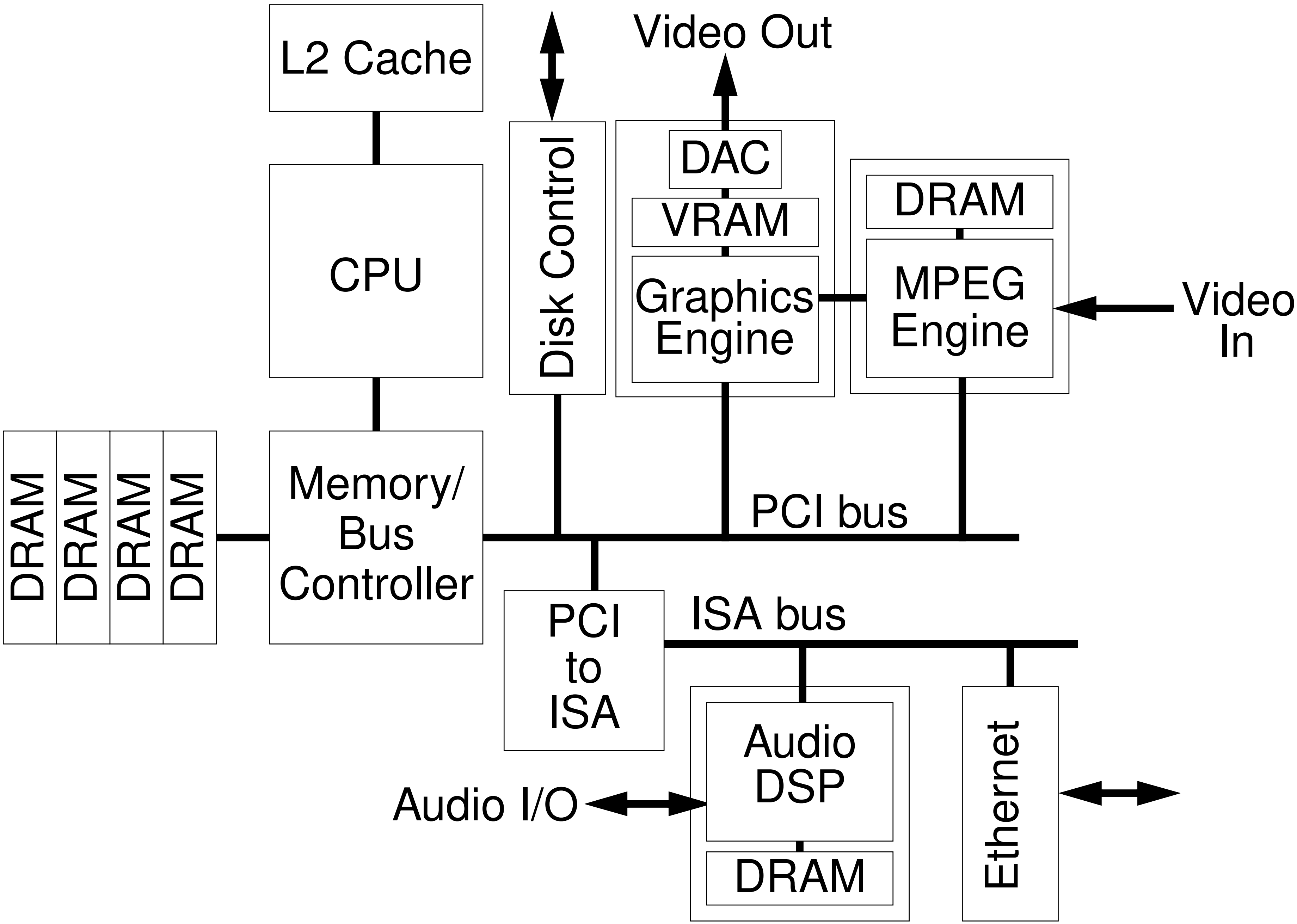
Neural nets for speech and handwriting recognition.
Adaptive speech filter (RASTA) for robust speech recognition.
DTW/HMM decoding for speech and handwriting recognition.
Image convolution for lip reading.
OCR for document conversion.

JPEG/MPEG encoding/decoding (motion estimation, DCTs, dithering).
Adaptive long echo cancellation for video conferencing.
Error control coding (PET, also RAID parity?).

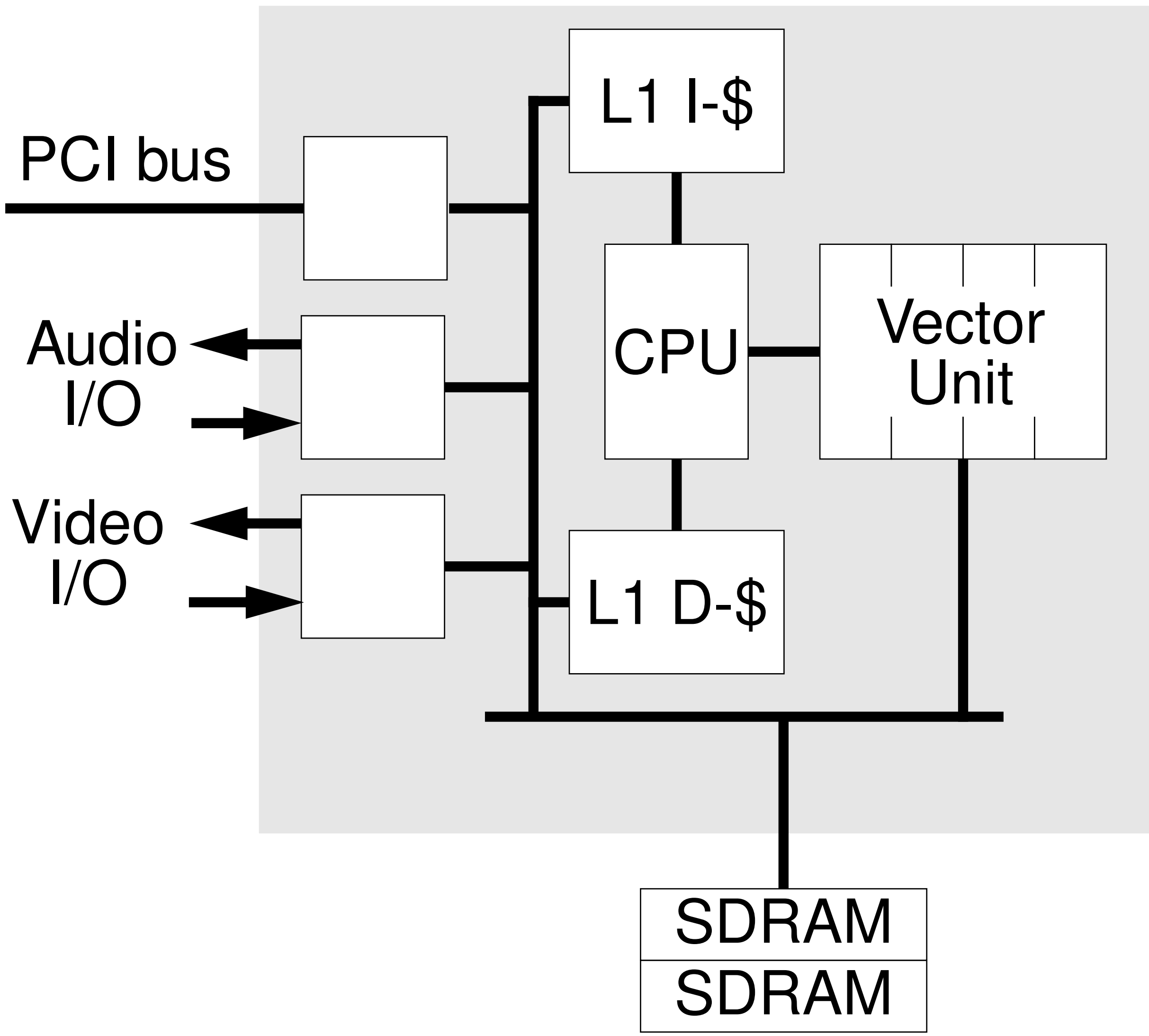
Audio synthesis.
3D sound.
Image histogram/filtering for Adobe Photoshop.
2D/3D graphics generation.
Video effects processing.

Encryption/decryption.
Speaker identification.
Fingerprint recognition.

Modern Desktop System



Vector Micro System Design



The T0 Vector Microprocessor

Krste Asanovic
James Beck
Bertrand Irissou
Brian E. D. Kingsbury
Nelson Morgan
John Wawrzynek

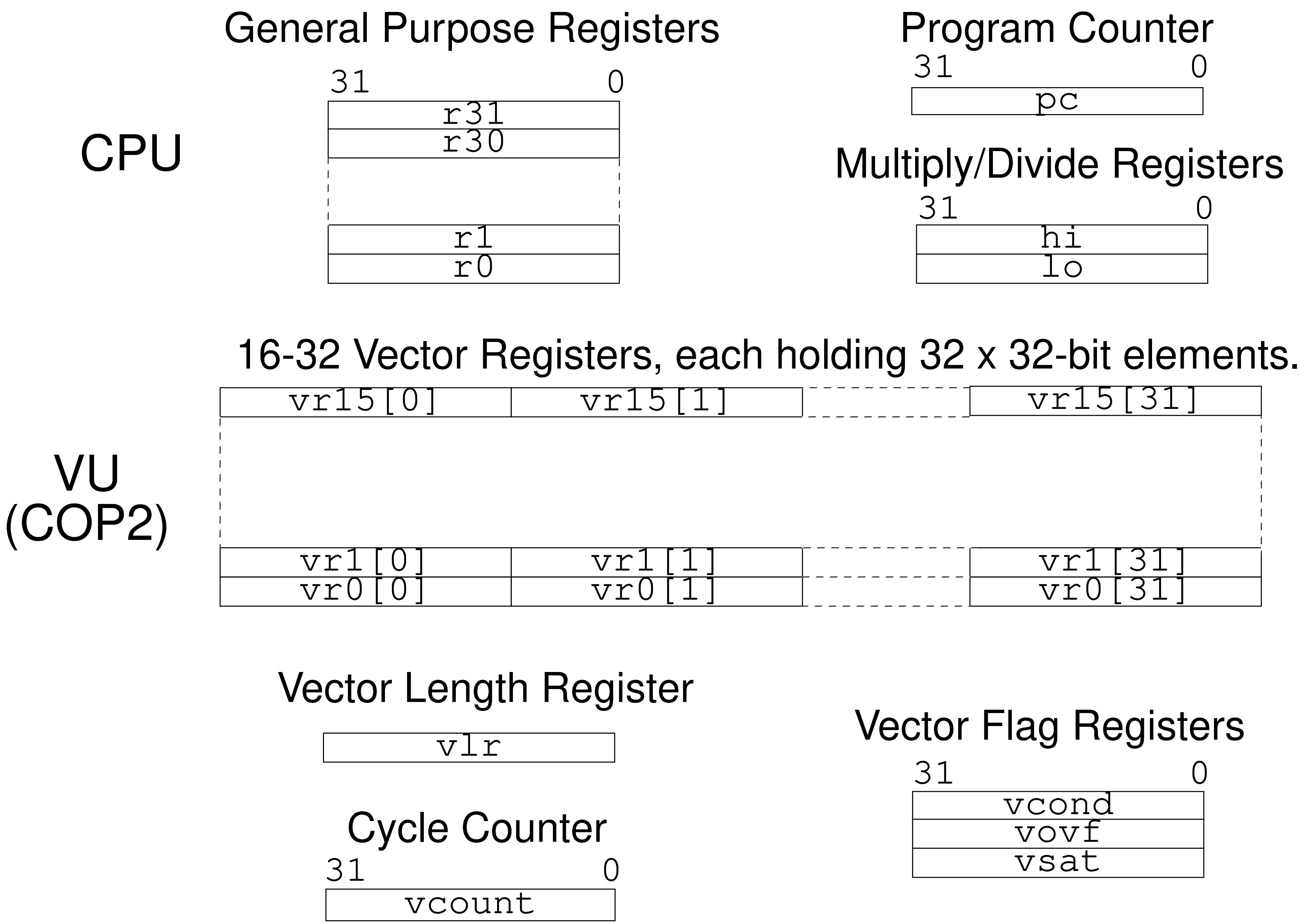
University of California at Berkeley
and the
International Computer Science Institute

http://www.icsi.berkeley.edu/real/spert/t0-intro.html

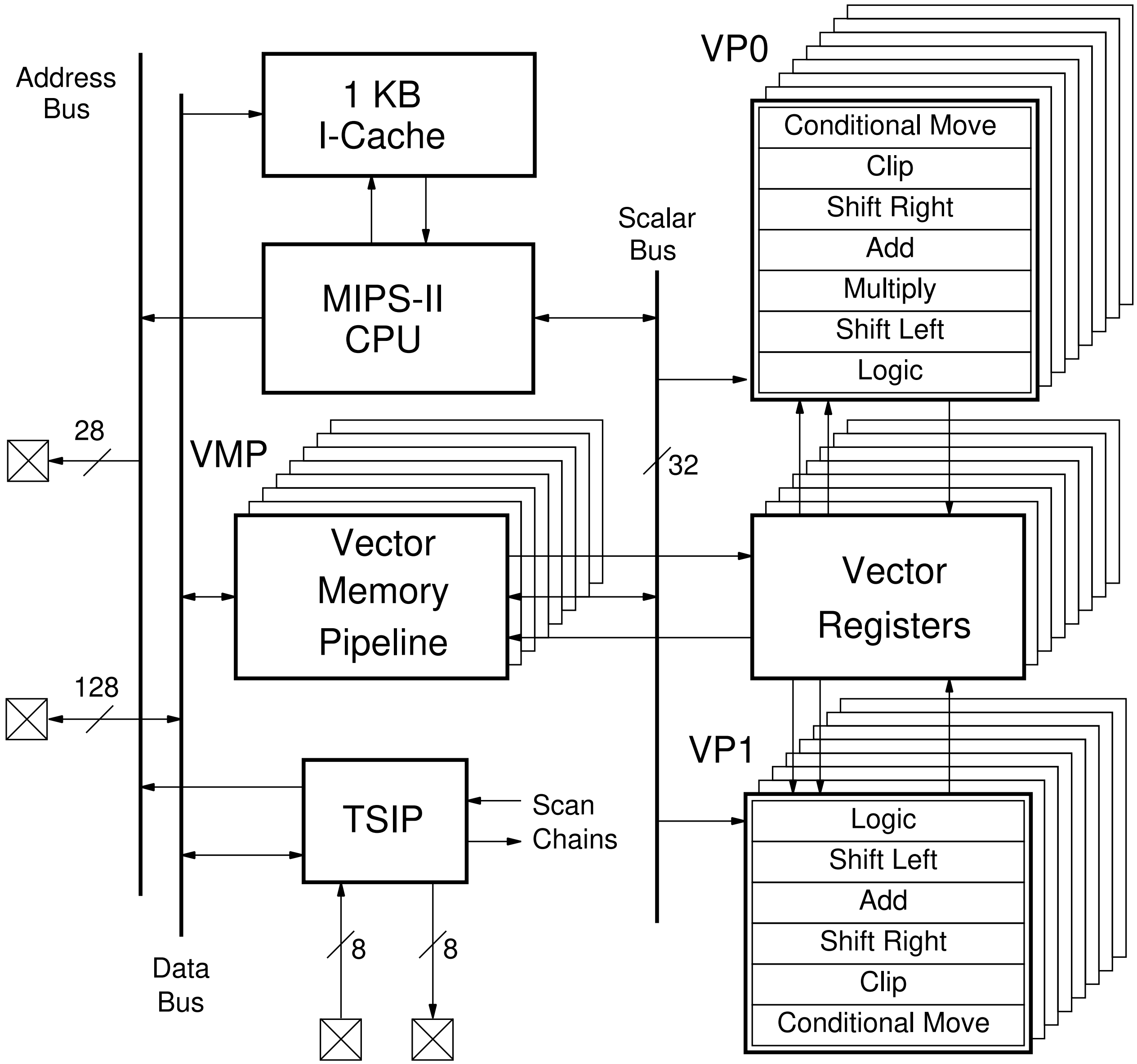
Primary support for this work was from the ONR, URI Grant N00014-92-J-1617, the NSF, grants MIP-8922354/MIP-9311980, and ARPA, contract number N0001493-C0249.

Additional support was provided by ICSI.

Torrent User Programming Model



T0 Block Diagram



T0 I-Cache and Scalar Unit

Instruction Cache

1 KB, direct-mapped, 16 byte lines.

Cache line prefetch if memory otherwise idle:

- 2 cycle miss penalty with prefetch,
- 3 cycle miss penalty without prefetch.

Service misses in parallel with interlocks.

System Coprocessor 0

Exception handling registers.

Host communication registers.

32-bit counter/timer.

MIPS-II 32-bit Integer RISC CPU

One instruction/cycle in 6 stage pipeline.

Single architected branch delay slot.

Annulling branch likelies.

Interlocked load delay slots.

3 cycle load latency (no data cache).

18 cycle 32-bit integer multiply.

33 cycle 32-bit integer divide.

T0 Vector Memory Operations

Unit-stride with address post-increment

```
lbai.v vv1, t0, t1  # t1 holds post-increment.
```

Eight 8-bit elements per cycle.

Eight 16-bit elements per cycle.

Four 32-bit elements per cycle.

+1 cycle if first element not aligned to 16 byte boundary.

Strided operations

```
lwst.v vv3, t0, t1  # t1 holds byte stride.
```

One 8-bit, 16-bit, or 32-bit element per cycle.

Indexed operations (scatter/gather)

```
shx.v vv1, t0, vv3  # vv3 holds byte offsets.
```

One 8-bit, 16-bit, or 32-bit element per cycle.

+ 3 cycle startup for first index.

Indexed stores need 1 extra cycle every 8 elements.

T0 Vector Arithmetic Pipelines

Full set of 32-bit integer vector instructions: add, shift, logical.

Vector fixed-point instructions perform a complete scaled, rounded, and clipped fixed-point arithmetic operation in one pass through pipeline.

Multiplier in VP0 provides 16-bit x 16-bit -> 32-bit pipelined multiplies.

Scale results by any shift amount.

Provides 4 rounding modes including round-to-nearest-even.

Clip results to 8-bit, 16-bit, or 32-bit range.

VP0 and VP1 each produce up to 8 results per cycle.

Vector arithmetic operations have 3 cycle latency.

T0 Vector Conditional Operations

Vectorize loops containing conditional statements.

Executed in either arithmetic pipeline.

Vector compare:

```
# vv2[i] = (vv5[i] < vv6[i])
slt.vv vv2, vv5, vv6
```

Vector conditional move:

```
# if (vv2[i] > 0) then vv1[i] = vv3[i]
cmvgtz.vv vv1, vv2, vv3
```

Vector condition flag register:

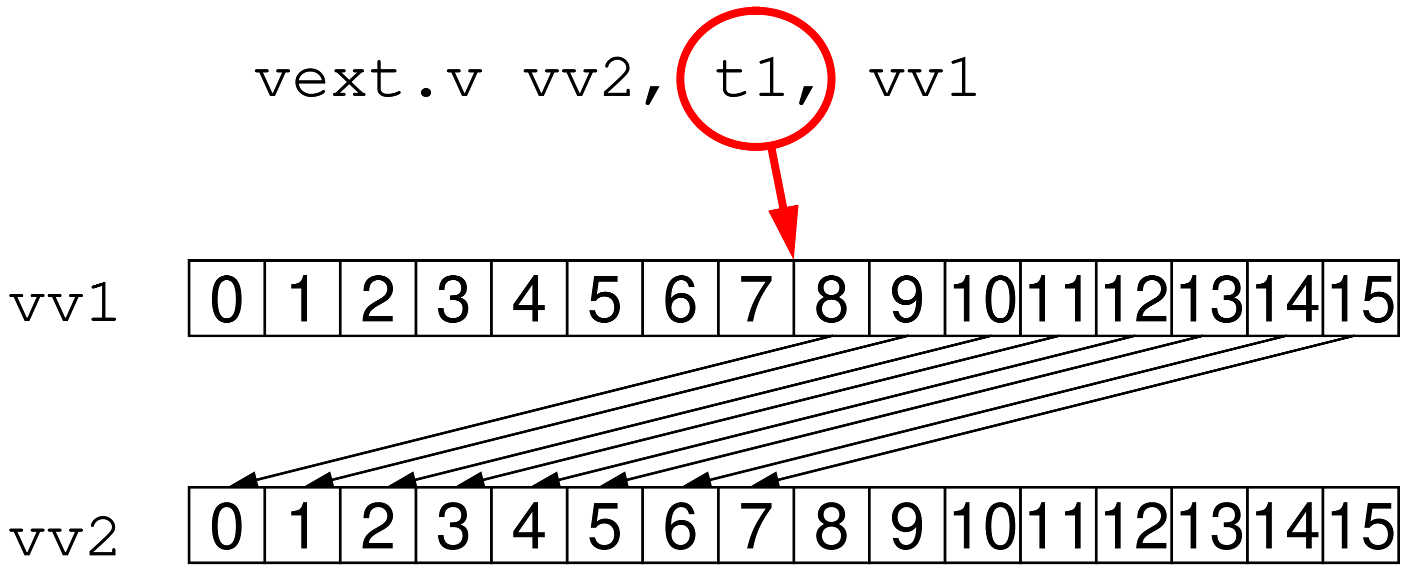
```
# vcond[i] = (vv1[i] < vv2[i])
flt.vv vv1, vv2     # Set flag bits.
cfc2 r1, vcond      # Read into scalar reg.
```

T0 Vector Editing Instructions

Executed in vector memory unit.

Scalar insert/extract to/from vector register element.

Vector extract supports reduction operations:

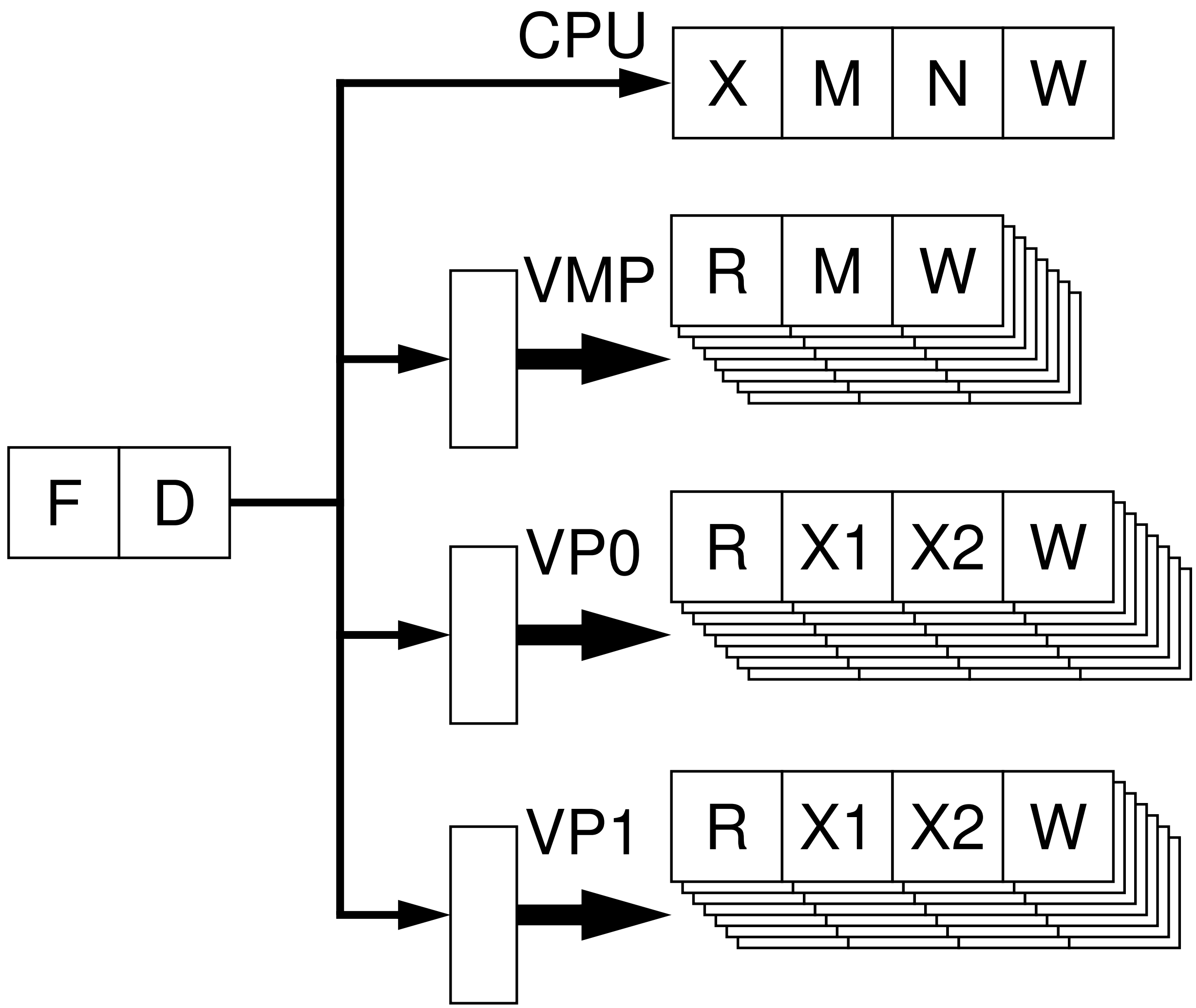


Avoids multiple memory accesses.

Separates data movement from arithmetic operations.

Software can schedule component instructions within reduction.

T0 Pipeline Structure

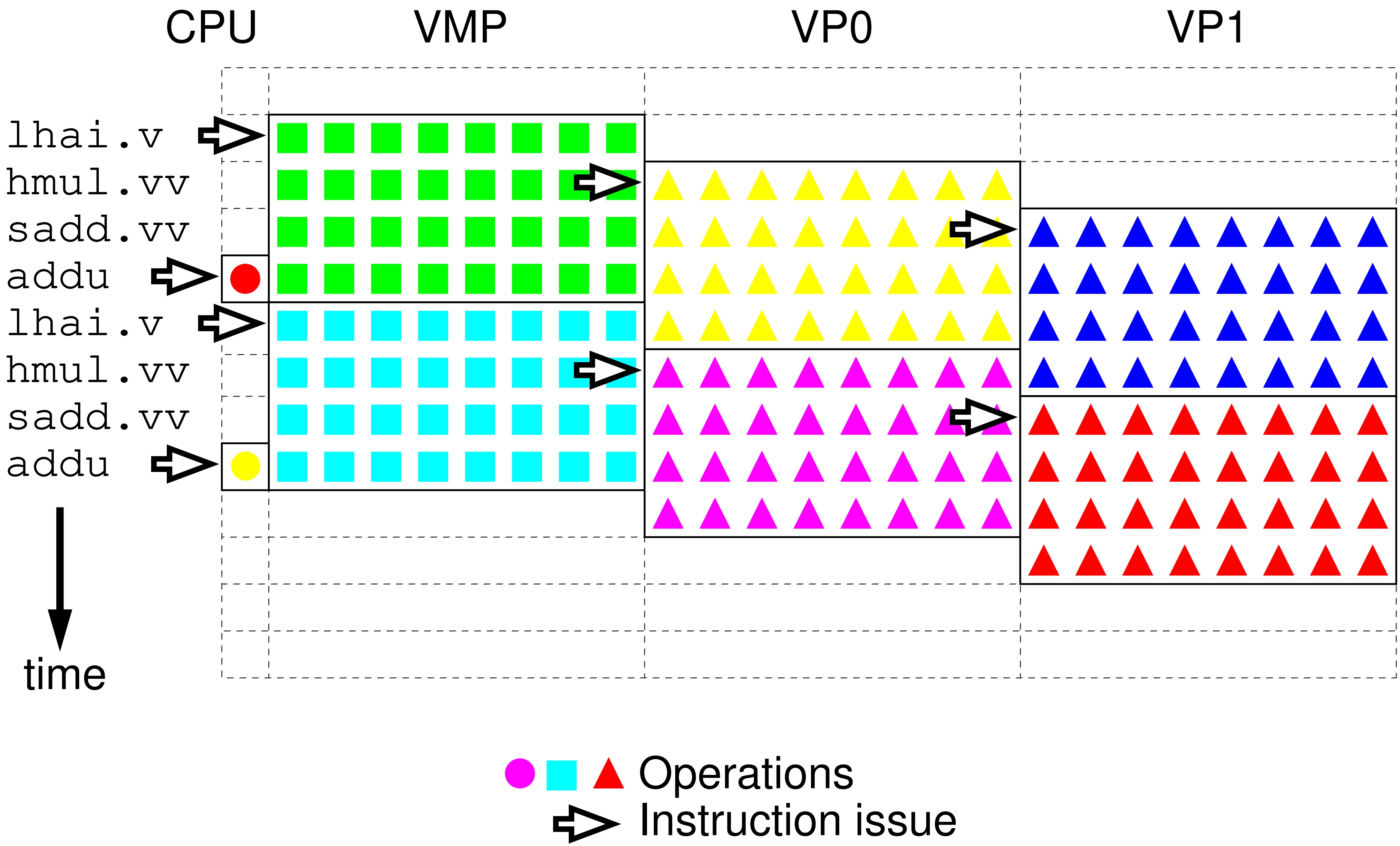


Code Example

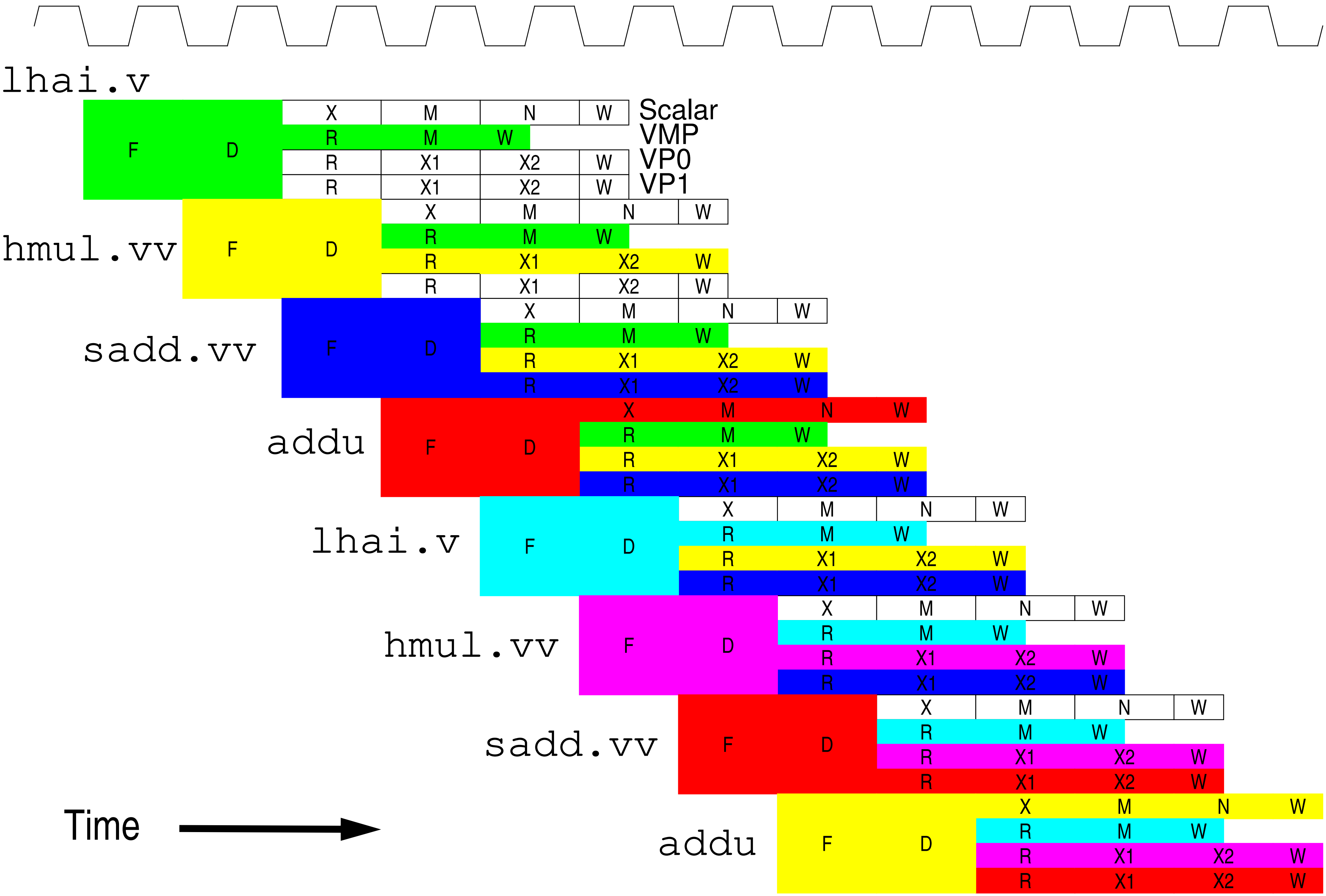
```
lhai.v vv1, t0, t1      # Vector load.
hmul.vv vv4, vv2, vv3   # Vector mul.
sadd.vv vv7, vv5, vv7   # Vector add.
addu t2, -1             # Scalar add.
lhai.v vv2, t0, t1      # Vector load.
hmul.vv vv5, vv1, vv3   # Vector mul.
sadd.vv vv8, vv4, vv8   # Vector add.
addu t7, t4             # Scalar add.
```

(taken from matrix-vector multiply routine)

Execution of Code Example



Full pipeline view of execution



T0 Vector Unit Hazards

All vector unit hazards fully interlocked in hardware.

Vector instruction startup fully pipelined to eliminate strip-mining overhead.

Each functional unit has independent ports into vector register file so no chain slot time and no vector register access conflicts.

All RAW, WAR, and WAW hazards on vector registers fully chained to reduce latency and decrease vector register pressure.

Philosophy: Trade small amount of extra control logic for increased utilization of multiple, expensive datapaths.

T0 External Interfaces

External Memory Interface

Supports up to 4 GB of SRAM with 720 MB/s bandwidth.

SRAM access wave-pipelined over 1.5 cycles.

Industry standard 17ns asynchronous SRAM for 45 MHz.

Serial Interface Port

Based on JTAG, but with 8 bit datapaths.

Provides chip testing and processor single-step.

Supports 30 MB/s host-T0 DMA I/O bandwidth.

Hardware Performance Monitoring

Eight pins give cycle by cycle CPU and VU status.

Fast External Interrupts

Two prioritized fast interrupt pins with dedicated interrupt vectors.

T0 Die Statistics

Technology:

1.0 μm MOSIS scalable CMOS rules, 2 metal, 1 poly.

Contacted M1 pitch 3.25 μm

Contacted M2 pitch 3.75 μm

Fabbed in HP's CMOS 26G process.

Die Size: 16.75 mm x 16.75 mm

Transistor Count: 730,701

Clock Frequency: 45 MHz

Power Supply: 5 V

Power Dissipation: 12 W max.

Why Vector Micros are different from Vector Mainframes

- **Applications:** Multimedia and human-machine interface
- **Implementation technology:** Single-chip CMOS

How do applications and implementation technology affect vector architecture design?

- Memory subsystem
- Vector register file
- Vector arithmetic units
- Scalar unit

Problems with SDRAM

Vector loops have several simultaneous access streams, but one rank of JEDEC SDRAM chips has a single open page:

->**Frequent page misses**

More simultaneously open pages would help, either with more ranks of chips, more banks per chip (MoSys), or SRAM cache on chip (Mitsubishi). Hiding precharge reduces miss penalty.

No major problems with unit-stride or short-stride accesses (only one page miss for each vector load/store) or if working set size is small enough (all in page/page cache).

But no inexpensive solution for combination of large working sets and long-stride or scatter/gather accesses.

Address Bandwidth

i.e., how many separate addresses per cycle.

Fixed pin budget. Must trade address bandwidth for data bandwidth.

Also, more addresses per cycle requires more address adders, more ports into TLB, more complex memory crossbar, and more collision detection hardware.

Unit-stride 80-95% of vector accesses over vector workload. DMA access for I/O, video, audio, etc., is also unit-stride.

=> **Cheap vector micro generates one address per cycle.**

(Cached system, only one address every block.)

Non-Unit-Stride

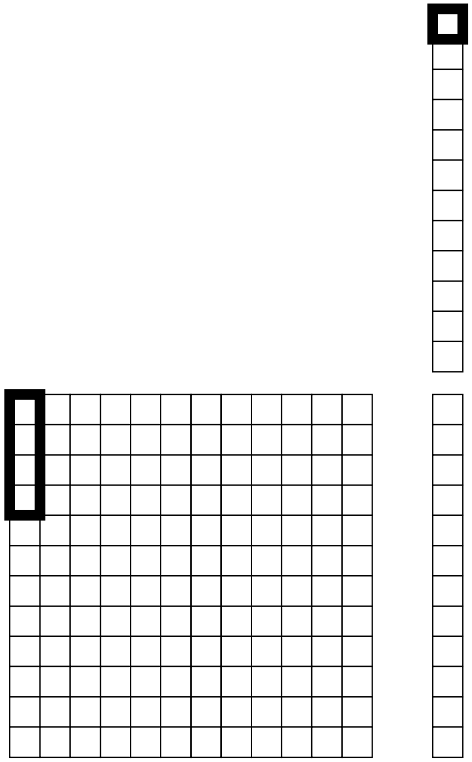
With only 1 address per cycle, and long RAS latency, vector micro performance reduced on non-unit-stride.

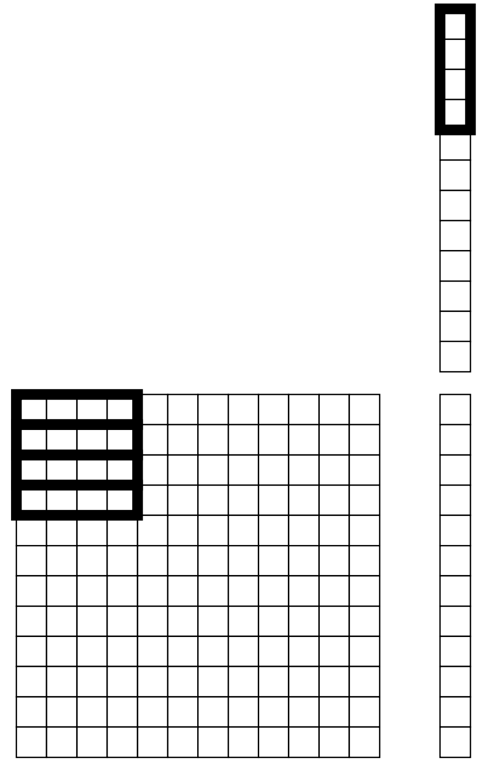
Amdhal’s law says we must do something about vector accesses which are non-unit-stride.

Strategies:

- Re-order loops to use unit-stride, e.g., matrix-vector multiply.
- Add on-chip vector register permute operations to replace some non-unit-stride accesses, e.g., FFT.

Matrix-Vector Multiply



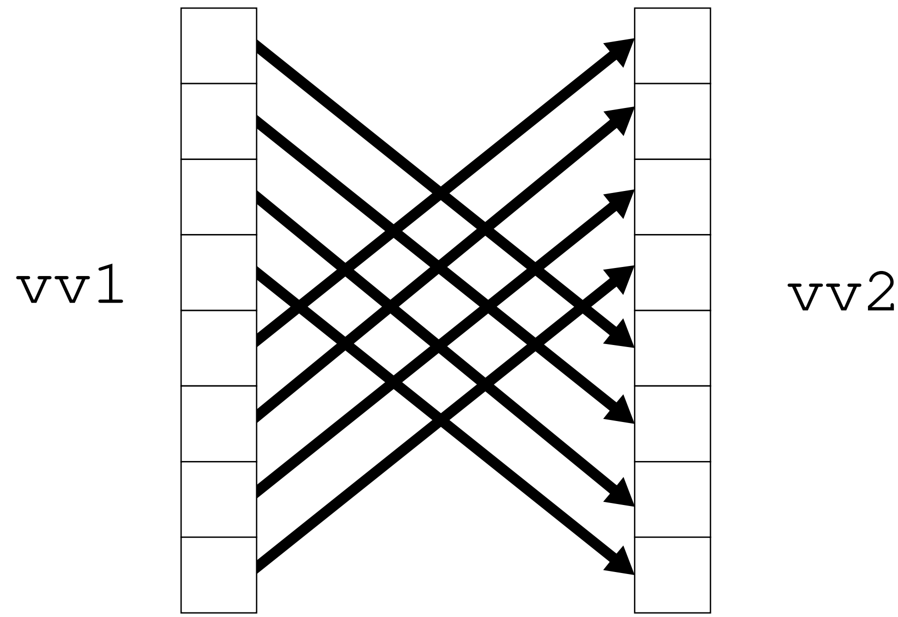


Vector accessed by scalars. Matrix accessed with large stride. Vector-scalar multiplies, vector-vector add into single vector register.

Vector accessed with unit-stride. Matrix accessed in tiles, several rows at a time. Multiple vector-vector multiplies. One vector register holds partial sums for each row. Need reduction at end of row.

FFT

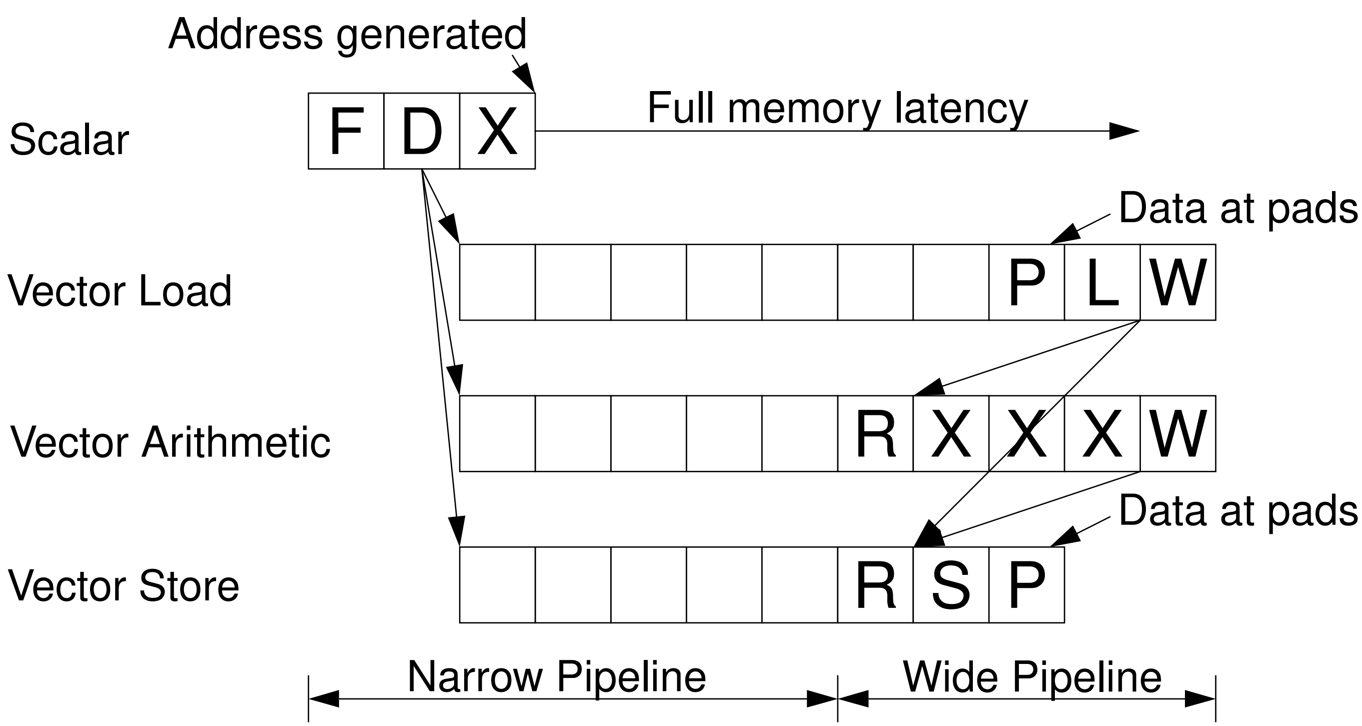
```
# rt distance is power-of-2.
vbfly.vv vv2, vv1, rt
```



- Changes off-chip non-unit-stride to on-chip vector register permutation.
- Same patterns useful for multiple reductions in one vector register.
- Executes in memory pipeline, chained with multiply-adds.
- Low hardware cost (1 extra datapath wire for T0).

Memory Latency

Latency is from address to data.
Pipeline out to CAS hits.



Most vector unit operations only see ~3 cycle latency.
Full latency only seen by:

- Indexed operations (index vector values to address generator).
- Scalar flag reads (flags generated at end of arithmetic pipe).
- Scalar cache refills.

Vector Register Design

Mainframes used many discrete SRAMs to get bandwidth, resulting in large capacity vector register files.

Large vector register file takes significant die area on vector micro (128KB from VPP300 same size as whole T0!).

In full custom VLSI, can customize size/design to fit needs:

- Close to SRAM density by using many small interleaved single- or dual-ported SRAM blocks to provide multiple ports for vector functional units.
- Regular sequential access pattern of vector functional units avoids conflicts in interleaved vector register file.
- Register file to functional unit crossbar lives above the storage cells in metal 3, takes little extra area.

Vector Register Organization

Can effectively concatenate short vector registers through loop unrolling:

```
lwai.v vv1, a0, a1 # Page break.
lwai.v vv2, a0, a1 # Access hits same page.
addu.vs vv1, vv1, t0
addu.vs vv2, vv2, t0
swai.v vv1, a2, a3 # Page break.
swai.v vv2, a2, a3 # Access hits same page.
```

Pipelined instruction startup removes overhead, get 100% utilization during loop.

So more short vector registers better than fewer long vector registers but instruction encoding imposes limit on total number.

Chaining all data hazards (RAW, WAR, WAW) helps ease vector register pressure. Chaining logic area proportional to memory pipeline depth and number of functional units, but not number of operations per cycle.

Short Chimes

Combination of short vector registers and multiple parallel pipelines gives much shorter chimes than vector mainframe.

Crays have natural chime length of ~64 clocks.

T0 has natural chime length of only 4 clocks!

Compiler needs different instruction scheduling algorithm.

Uses greater issue bandwidth, which can be bottleneck.

Vector Functional Unit Latency

CMOS favours spatial parallelism rather than fine pitch pipeline.

Scalar core designed with cycle time ~ integer add + bypass.

So vector micro latencies shorter than vector mainframes':

- Vector mainframes >= 6 clock latencies.
- Current microprocessors have 2-4 clock latencies for 64-bit floating-point (though can save area by stretching latency one or two clocks).

Vector Functional Unit Types

Not just 64-bit floating point.

Low resolution fixed-point (8, 16, and 32 bit) with saturation and rounding required for many applications.

Single precision floating-point performance important for graphics and DSP.

Long integers for cryptography.

Partitionable datapath (1x64b, 2x32b, 4x16b, 8x8b) handled elegantly by vector architecture. Provides 2x to 8x throughput increase for 10-20% area overhead and possibly some latency penalty.

Scalar CPU

Last 15 years of RISC/compiler research greatly improved scalar performance. More scalar registers, cache, greater ILP.

Vector microprocessor scalar CPU same technology as other microprocessors.

Single chip implementation gives low latencies.

Easy to add on-chip scalar data cache to uniprocessor vector micro using write-through with vector write invalidate.

But: *Scalar cache much more difficult in SMP vector system*.

Not FORTRAN

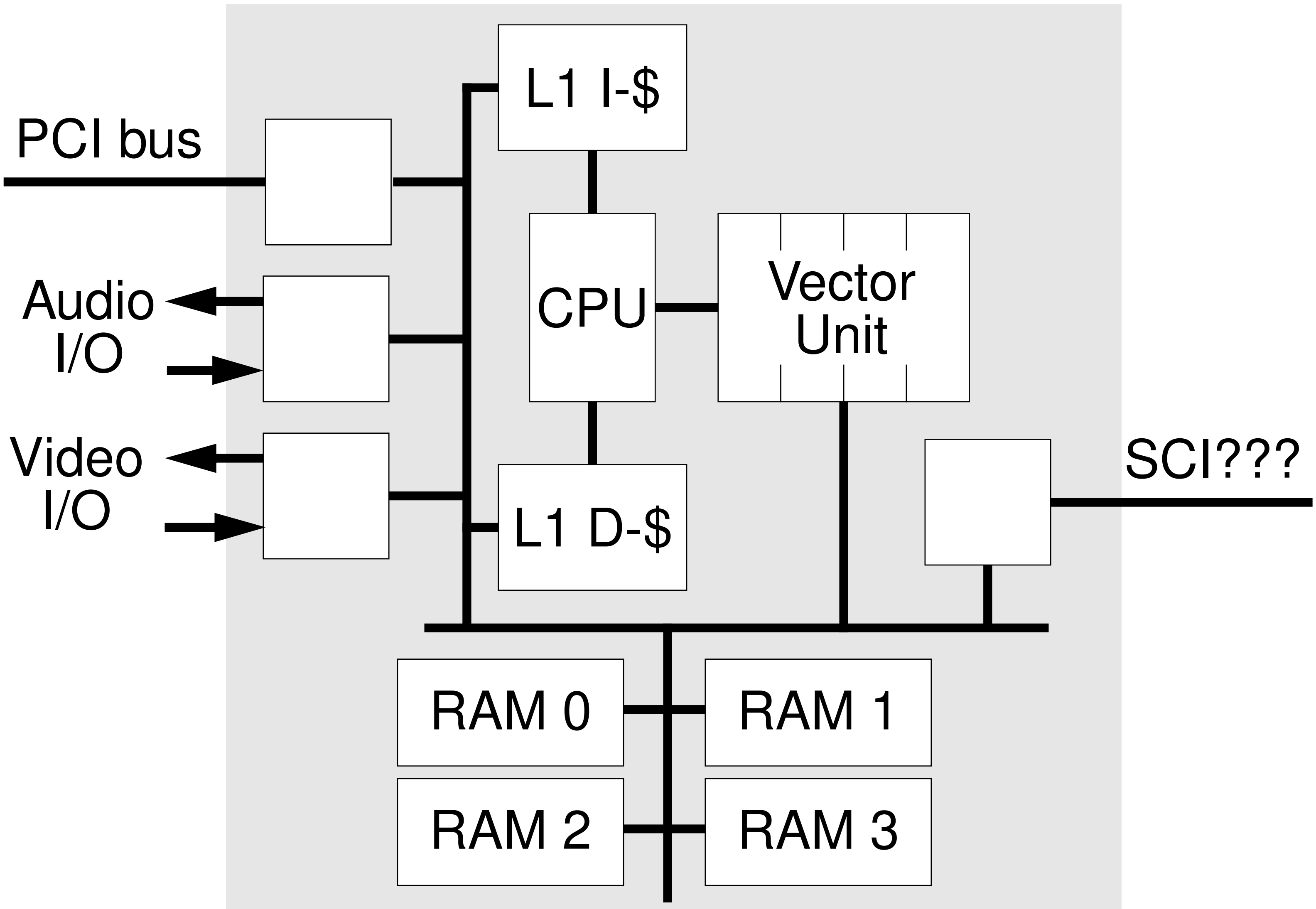
Most commercial DSP/multimedia kernels written in assembler, rest of code in C/C++.

Work progressing on vectorizing C, both commercially at Convex and Cray, and in academia [Lam, Lee].

ANSI Numerical C extensions?

Golden opportunity: High performance vector DSP with vectorizing compiler capable of generating good code.

Vector IRAM



PCI bus
Audio
I/O
Video
I/O
L1 I-$
CPU
Vector
Unit
L1 D-$
SCI???
RAM 0
RAM 1
RAM 2
RAM 3

Vector IRAM

Main advantages over vector micro:

- Lower power - wide off-chip buses requires lots of power.
- Lower cost with enough volume - many fewer pins.
- Lower memory latency - mainly helps scalar code.
- Could solve address bandwidth problem for strided and scatter/gather.

Maybe greater memory bandwidth not a huge advantage?

- Can get many pins on vector micro, and maybe can't put huge vector unit on even a 1Gb DRAM process.