Vector Microprocessors: A Case Study in VLSI Processor Design

Krste Asanovic

MIT Laboratory for Computer Science

krste@mit.edu
http://www.cag.lcs.mit.edu/~krste

Seminar Outline

- **Day 1** Torrent-0: Design, rationale, and retrospective
- **Day 2** VLSI microprocessor design flow
- **Day 3** Advanced vector microprocessor architectures

Day 1

Torrent-0: Design, Rationale, and Retrospective

Session A: Background and motivation

Break

Session B: Torrent ISA and T0 microarchitecture overview

Lunch

Session C: Microarchitecture details

Break

Session D: Results and retrospective

The T0 Vector Microprocessor

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

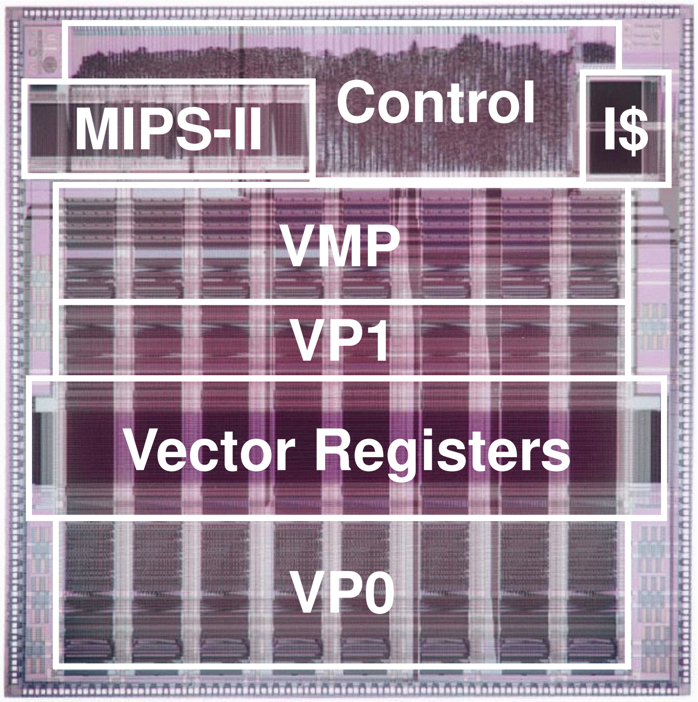
University of California at Berkeley
and the
International Computer Science Institute

`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.

T0 Die



Die Statistics:
HP CMOS 26G process
1.0 μm MOSIS SCMOS
2 metal, 1 poly
16.75 x 16.75 mm^2
730,701 transistors
4W typ. @ 5V, 40MHz
12W max.

Peak Performance:
640 MOP/s
320 MMAC/s
640 MB/s

T0 Project Background

GOAL:
Fast ***systems*** to train artificial neural networks (ANNs) for speech recognition

Team combined applications + VLSI experience:

Speech recognition group at ICSI (International Computer Science Institute), Berkeley
(Prof. Nelson Morgan)

VLSI group in the CS Division, UC Berkeley
(Prof. John Wawrzynek)

ICSI Speech Recognition System

Hybrid System, ANNs plus Hidden Markov Models (HMMs)

Research is compute-limited by ANN training
ICSI speech researchers routinely run GFLOP-day jobs

First ICSI system, Ring Array Processor (RAP) (1989)
up to 40 TMS320C30 DSPs
plus Xilinx-based ring interconnect
~100 MCUPS (Million Connection Updates/Second)
(contemporary Sparcstation-1 achieved ~1 MCUPS)

RAP successful, but large and expensive (~$100,000)

Exploiting Application Characteristics

Simulation experiments showed that 8-bit x 16-bit fixed-point multiplies and 32-bit adds sufficient for ANN training.

ANN training is embarrasingly data parallel

=> Special purpose architecture could do significantly better than commercial workstations.

UCB/ICSI VLSI Group History

1990 HiPNeT-1 (HIghly Pipelined NEural Trainer)

Full-custom application-specific circuit for binary neural network training

2.0μm CMOS, 2 metal layers, 16mm^2 (16Mλ^2)

Test chips fully functional at 25MHz

1991 Fast Datapath

Experiment in very high speed processor design

Full-custom 64-bit RISC integer datapath

1.2μm CMOS, 2 metal layers, 36mm^2 (100Mλ^2)

Two revisions, second version fully functional at 180-220MHz

1992 SQUIRT

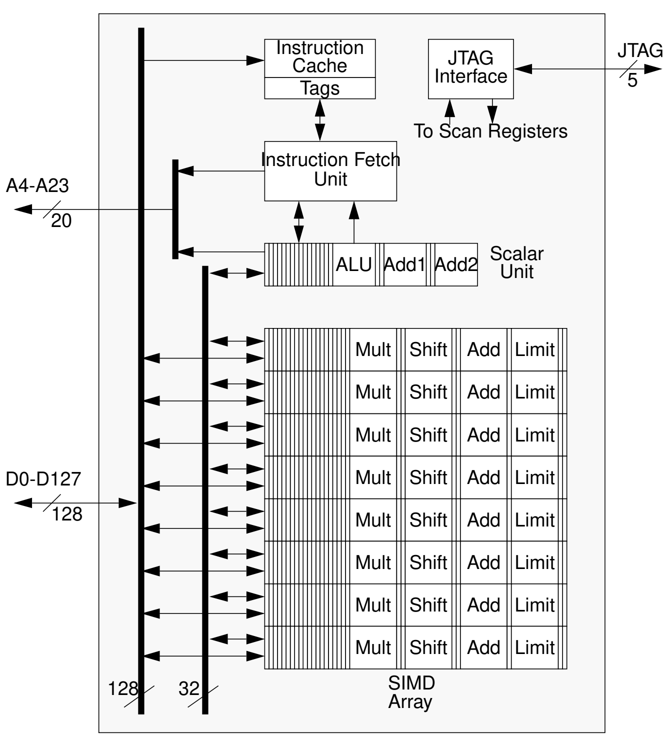
Test chip for old-SPERT VLIW/SIMD design (one slice of SIMD unit)

Full-custom 32-bit datapath including fast multiplier

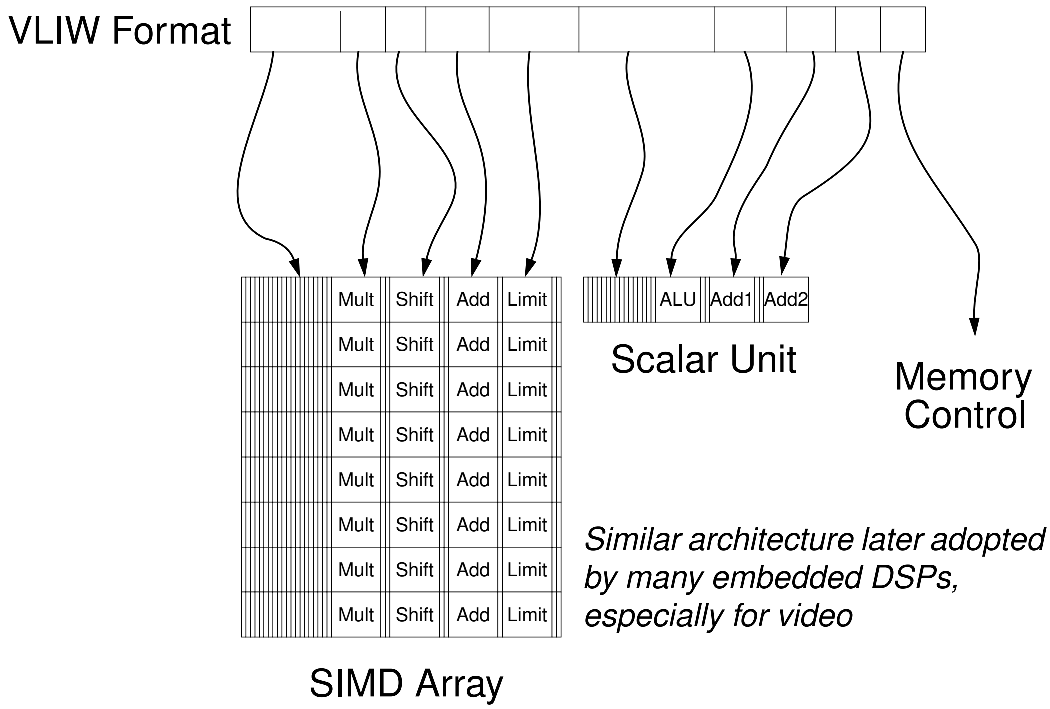
1.2μm CMOS, 2 metal layers, 62K transistors, 32mm^2 (89Mλ^2)

Fully functional at over 50MHz

"Old-SPERT" Architecture



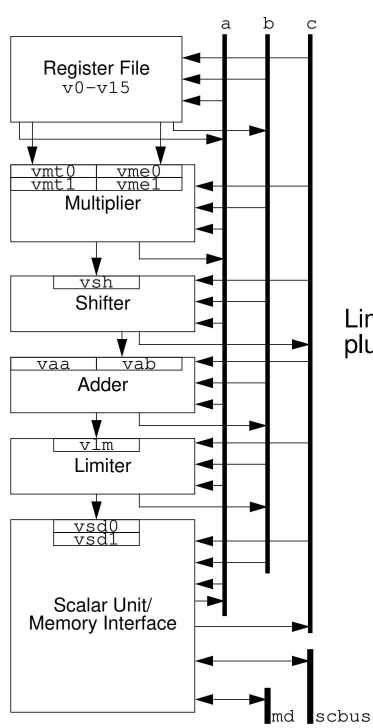
“Old-SPERT” 128-bit VLIW Instruction



Similar architecture later adopted by many embedded DSPs, especially for video

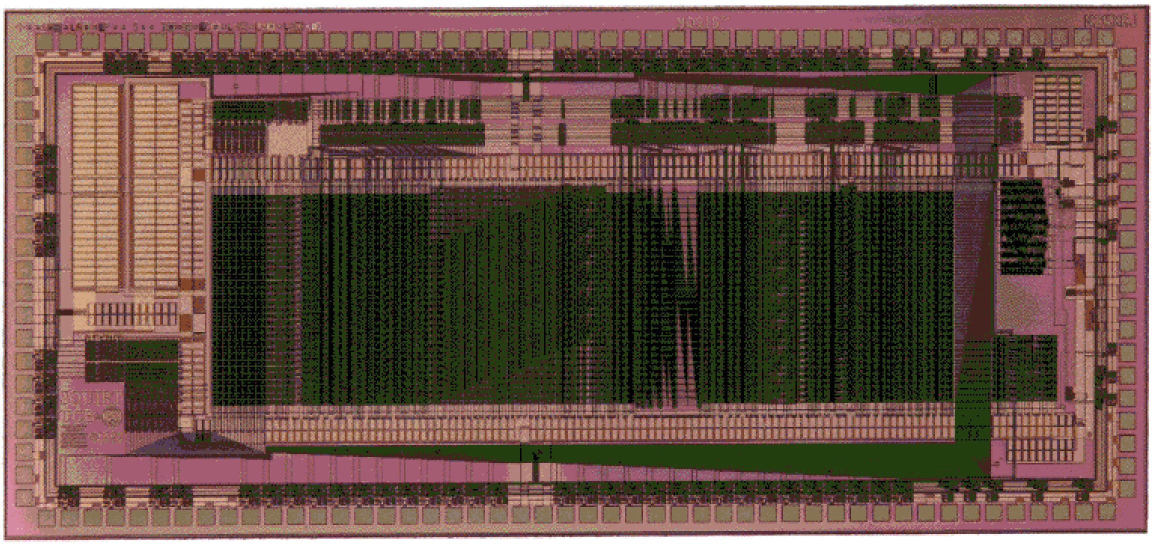
“Old-SPERT” SIMD Datapath

Few-ported central register file plus distributed register files



Limited global bypassing plus local “sneak” paths

SQUIRT: Testchip for “Old-SPERT”



HP CMOS34 1.2µm, 2 metal
61,521 transistors, 8x4 mm^2
0.4W @ 5V, 50MHz

72-bit VLIW instruction word
16x32b register file + local regfiles
24bx8b->32b multiplier
32b ALU, shifter, limiter

Why We Abandoned “Old-SPERT”

Software Reasons:

VLIW means no upward binary-compatibility
- Followup processor (for CNS-1) would have required all new software

VLIW scalar C compiler difficult to write
- VLIW+custom compiler more work than RISC+standard compiler

VLIW/SIMD very difficult to program in assembler
- Even writing initial test code was a chore!

Architectural Reasons:

Difficult to fit some operations into single cycle VLIW format
- Particularly non-unit stride, and misaligned unit-stride memory accesses

VLIW + loop unrolling causes code size explosion
- Instruction cache size/miss rate problems

The “Obvious” Solution: *Vectors!*

Vector architectures old and proven idea

- Vector supercomputers have best performance on many tasks
- Fitted our application domain

Can add vector extensions to standard ISA

- Use existing scalar compiler and other software

Can remain object-code compatible while increasing parallelism

- Second processor implementation planned

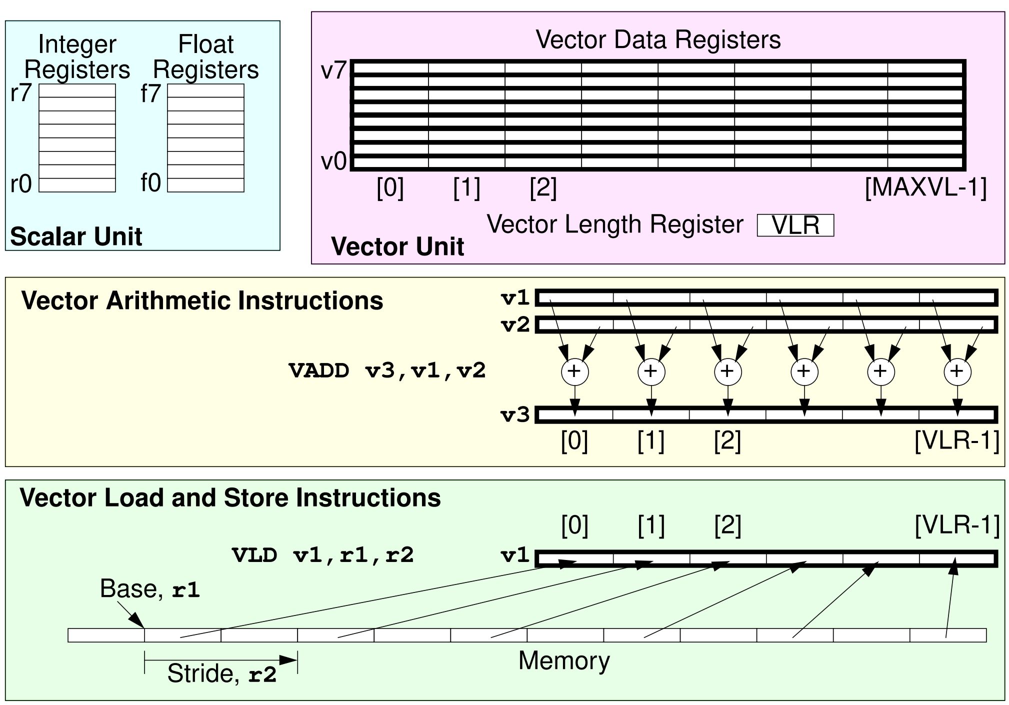
Vector instruction stream more compact

- Single 32-bit instruction fetch per cycle
- Smaller code from reduced loop unrolling and software pipelining
- Easier to write assembly library routines

More general purpose than VLIW/SIMD

- Vector length control
- Fast scatter/gather, strided, misaligned unit-stride

Vector Programming Model



System Design Choices

Which standard ISA?
=> Easy decision. MIPS is simplest RISC and well-supported.

Add vector coprocessor to commercial R3000 chipset?
- Scalar caches would have complicated vector unit memory interface
- Vector CoP. must connect to I-cache as well as memory system, more pins
- Large board design required, many high pin-count chips plus memory
- Increased latency and reduced bandwidth between scalar and vector units
- Standard coprocessor interface awkward for vector unit

=>Design our own MIPS and integrate everything on one die

State of Vector Architecture

Revelation: Existing vector designs obviously bad, especially for highly parallel vector micro.

Examples:

Huge (128KB) vector register files (VRFs) would have filled chip!
What length for VRs? How many VRs?

Dead time between vector instructions, why?

Limited chaining on commercial machines, why?

Vector ISAs with built-in scalability problems, e.g., instructions that read vector registers not starting at element zero, using scalar unit to handle flags, etc.

Accepted Research Approach

First:

- Build simulation infrastructure
- Write compilers
- Collect benchmarks
- Propose alternatives

Then:

- Compile benchmarks, get simulator results, compare alternatives

Great way of generating papers! Can also get real insight in some cases, but results only valid:

- if simulation valid (i.e., machine is buildable, parameters realistic, no bugs)
- if benchmarks realistic and complete
- if equal compiler effort for all alternatives

Generally, this approach is most applicable to small tweaks for established designs.

Designing Torrent-0

Started with conventional RISC ISA
plus conventional vector ISA designed for future scalability.

RISC microarchitecture fairly standard.
Vector microarchitecture designed from scratch.

Aimed for “general-purpose” performance.

- Very little microarchitecture tuning based on application kernels

Detailed T0 design mostly driven by low-level VLSI constraints.

- look for “sweet-spots” (e.g., reconfigurable pipelines)
- avoid trouble (e.g., multiple addresses/cycle, superscalar issue)

Whole system designed together.

- T0 VLSI, SBus board, host interface, software environment

Research by Building

Constructing artifacts:

- exposes otherwise hidden flaws in new ideas (it all has to really work)
- provides realistic parameters for further simulation studies
- reveals subtle interactions among design parameters
- (and achieving great results) is how to have impact on industry

But, requires huge engineering effort!

Summary

Initial project goal was to provide a high-performance ***application-specific*** workstation accelerator for ANN training

Chose ***general-purpose*** vector architecture

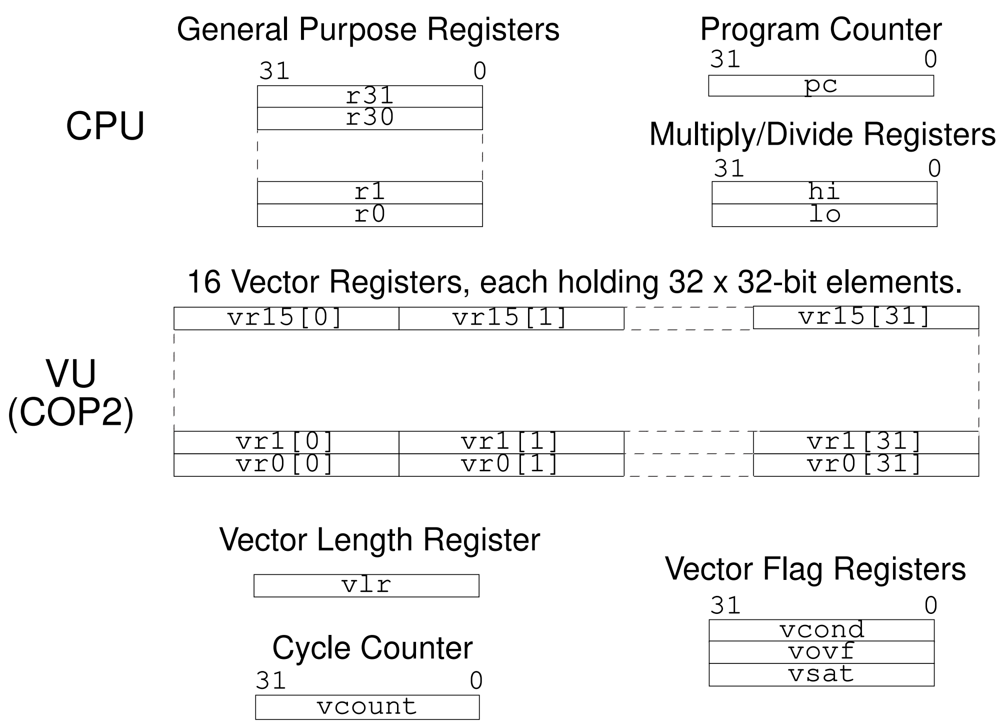
Not much literature, so design vector micro from scratch

VLSI-centric design process

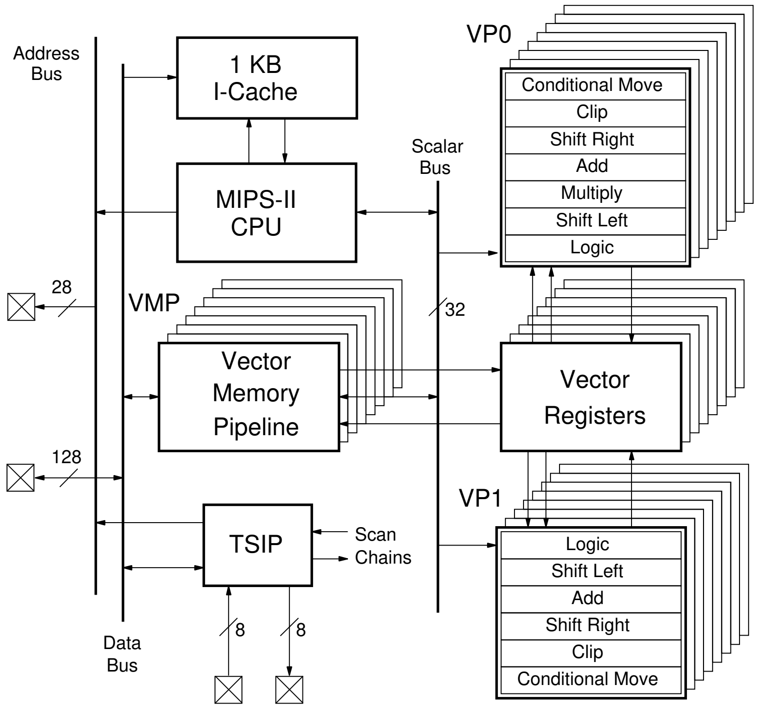
Emphasis on complete usable system => everything must work!

Day 1, Session B:
Torrent ISA, T0 Microarchitecture,
Spert-II System

Torrent User Programming Model



T0 Block Diagram



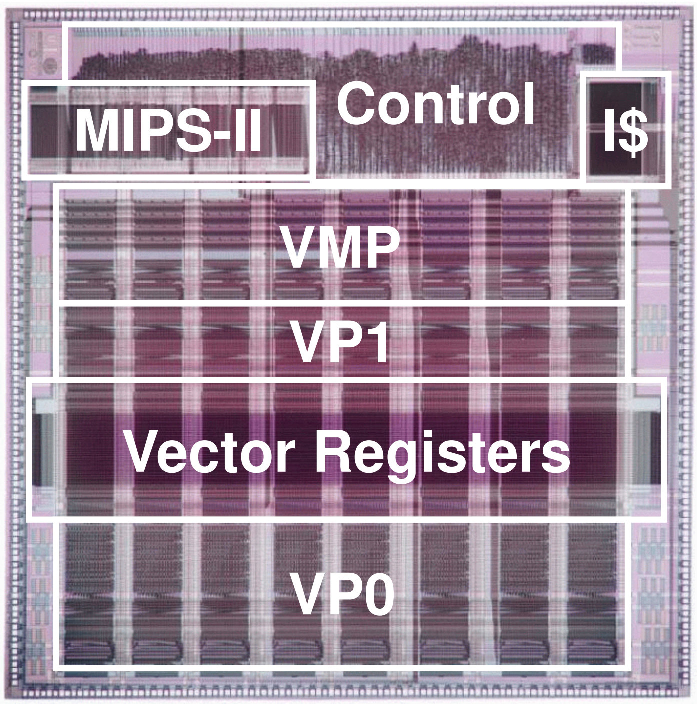
T0 I-Cache and Scalar Unit

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.

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.

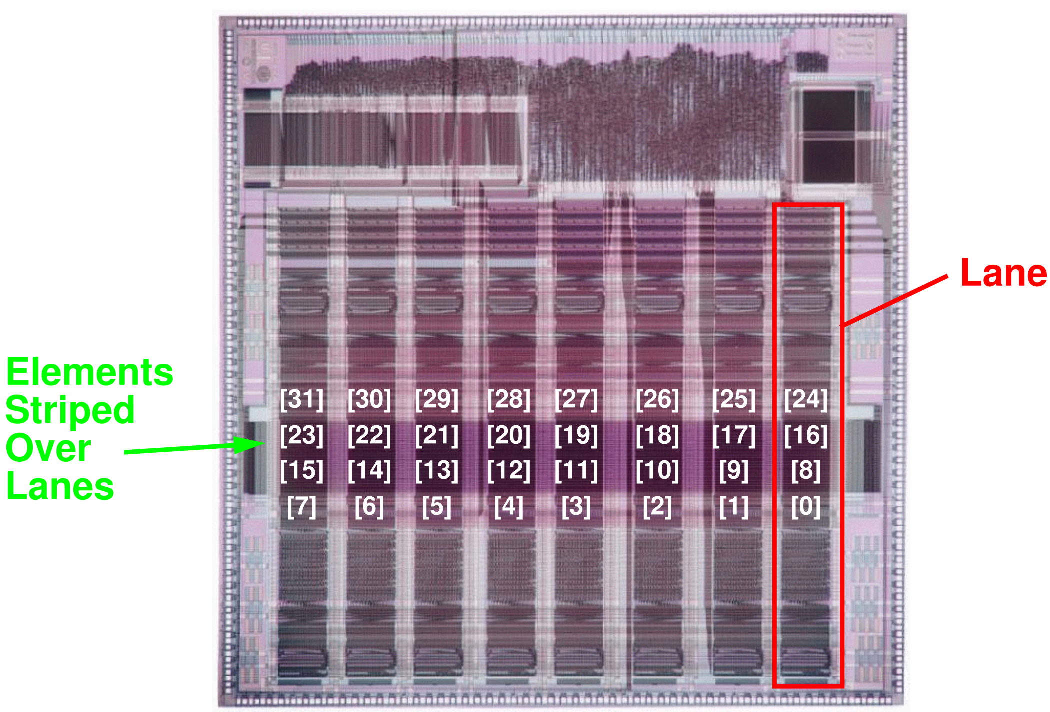
T0 Die



Die Statistics:
HP CMOS 26G process
1.0 µm MOSIS SCMOS
2 metal, 1 poly
16.75 x 16.75 mm^2
730,701 transistors
4W typ. @ 5V, 40MHz
12W max.

Peak Performance:
640 MOP/s
320 MMAC/s
640 MB/s

Vector Unit Organized as Parallel Lanes



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 Operations

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.

Reconfigurable pipelines perform up to six 32-bit integer operations in one instruction (up to 96 ops/cycle).

T0 Vector Conditional Operations

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:

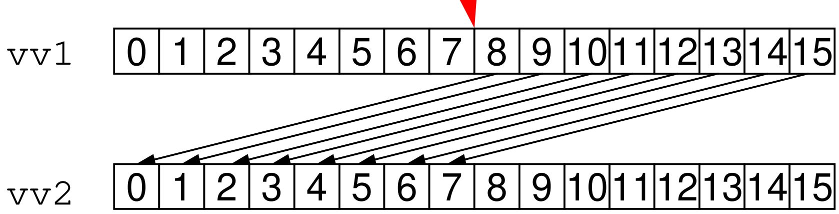
```
# vcond[i] = (vv1[i] < vv2[i])
flt.vv vv1, vv2     # Set flag bits.
cfc2 r1, vcond      # Read into scalar reg.
```

T0 Vector Editing Instructions

Executed in vector memory unit.
Scalar insert/extract to/from vector register element.

Vector extract supports reduction operations:

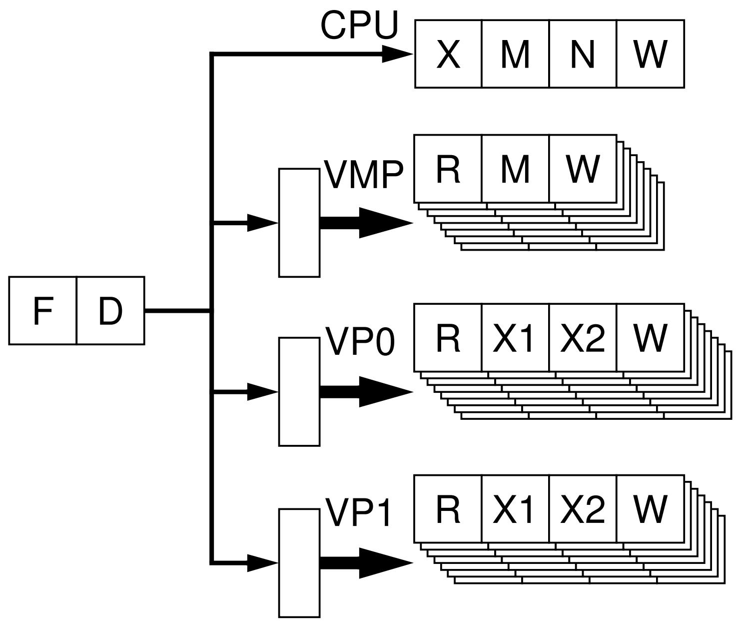
```
vext.v vv2, t1, vv1 # t1==8
```



- Avoids multiple memory accesses.
- Separates data movement from arithmetic operations.
- Software can schedule component instructions within reduction.

(Also added to Cray C90)

T0 Pipeline Structure

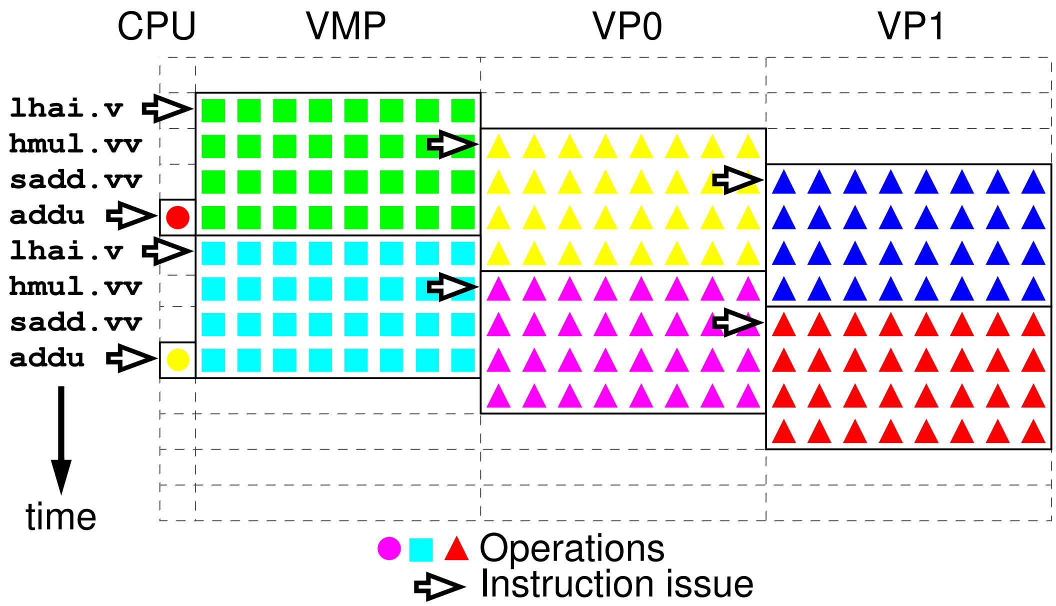


Code Example

(taken from matrix-vector multiply routine)

```
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.
```

Execution of Code Example on T0



Single 32-bit instruction per cycle sustains 24 operations per cycle.

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 I/O DMA bandwidth.

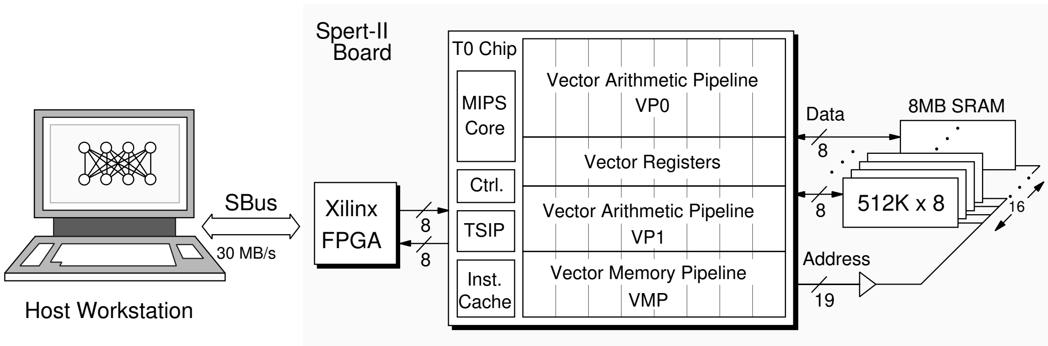
Hardware Performance Monitoring

- Eight pins give cycle by cycle CPU and VU status.

Fast External Interrupts

- Two prioritized fast interrupt pins with dedicated interrupt vectors.

Spert-II System



Spert-II Software Components

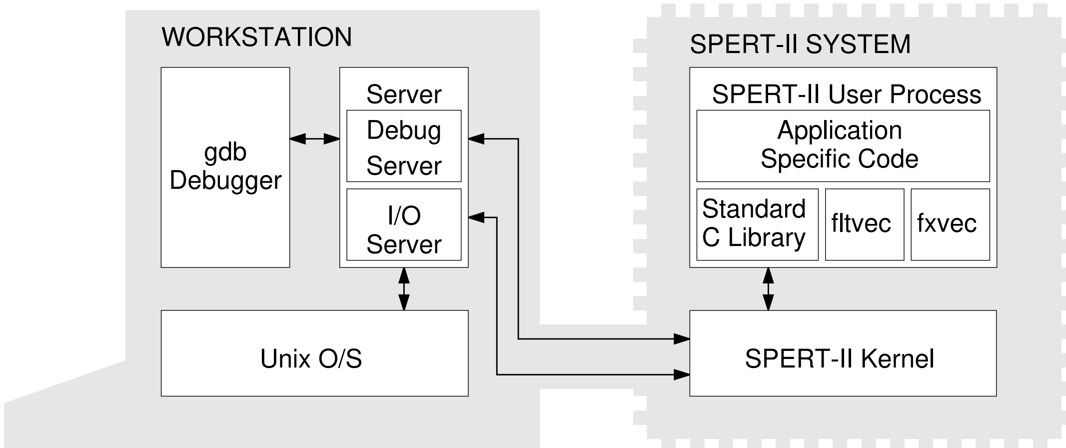
GNU-based tools:

- **`gcc`** scalar C/C++ cross-compiler (unmodified)
- **`gas`** cross-assembler (added vector instructions, instruction scheduling)
- **`gld`** linker, **`objdump`** disassembler (added vector instructions)
- **`gdb`** symbolic remote debugger (added vectors, our debug server)
- C standard library (added vectorized **`str*`** and **`mem*`** routines)

Custom software

- Host I/O server with Irix4 emulation on top of SunOS4
- Spert-II microkernel
- Scalar IEEE floating-point emulation (SoftFloat available on Web)
- Vector software IEEE floating-point libraries (~14 MFLOPS)
- Vector fixed-point libraries
- Applications, primarily QuickNet ANN trainer
- Performance simulators (more tomorrow)

Spert-II Runtime Environment



Summary

T0 is complete single-chip vector microprocessor.

Highly integrated component at core of system.

Software support large part of total effort.

Some simplifications/specializations:
- No floating-point hardware
- No virtual memory hardware
- SRAM main memory

Day 1, Session C:
T0 Microarchitecture in Detail

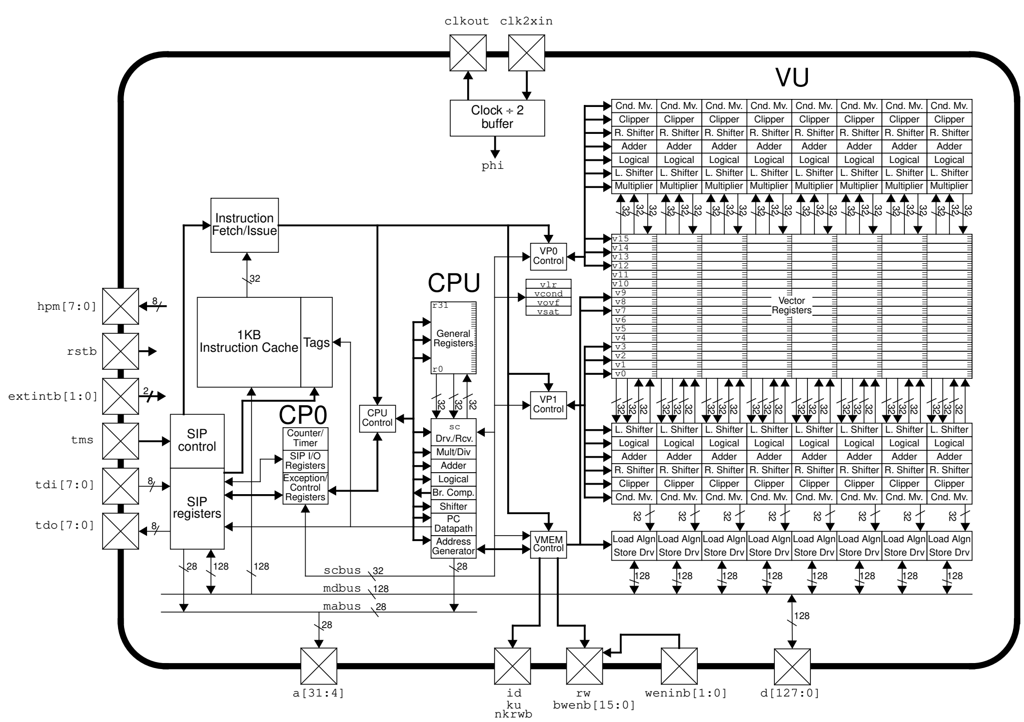
Memory Subsystem

Scalar Unit

Vector Register File

Arithmetic Pipelines

T0 Detailed Structure



Memory Component Choices

Required high-bandwidth, high-capacity commercial part.
In 1992, reliable options were:

Fast Page Mode DRAM, 4Mb available, 16Mb sampling
- •25MHz max cycle rate, would require interleaved banks
- •Extra external multiplexing components required
- •Complicated system design

Asynchronous SRAM, 1Mb available, 4Mb sampling
- •Adequate capacity
- •High performance
- •Simple system design

Industry contemplating high-bandwidth DRAMs (EDO DRAM, SDRAM, Rambus DRAM) but we weren't certain which would survive. (All did!)

=> T0 uses asynchronous SRAM memory

Address Bandwidth vs. Data Bandwidth

(Or, how many non-contiguous addresses per cycle?)

Fixed pin budget
=> must trade address bandwidth for data bandwidth.

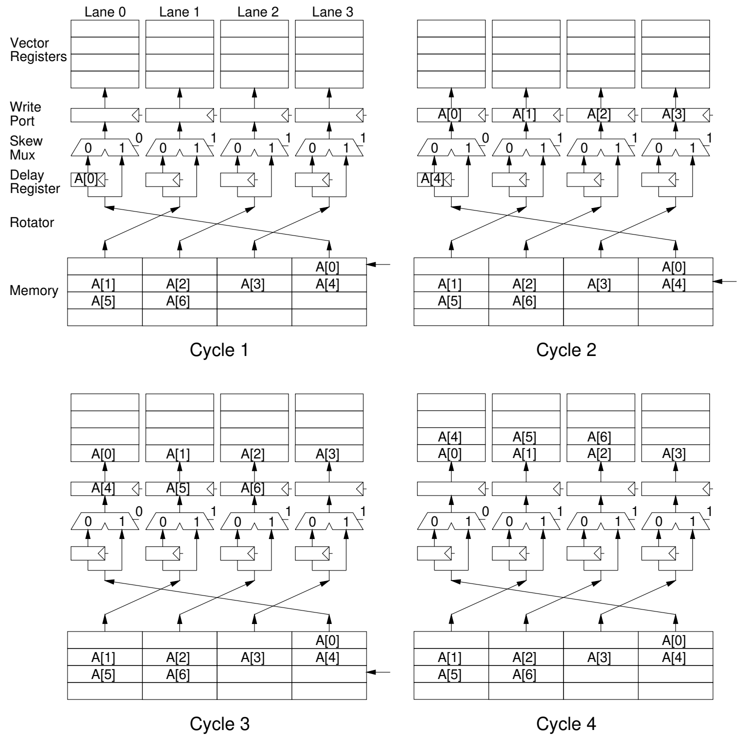
Also, more addresses per cycle requires:
- more address adders
- more ports into TLB (just protection checks on T0)
- more complex memory crossbar
- more address conflict detection hardware

Unit-stride 80-95% of vector memory accesses.
Cache, I/O memory access also unit-stride.

=> T0 generates one address per cycle

(Dedicated address adder in scalar datapath to support concurrent vector memory and scalar ALU operations.)

Misaligned Unit-Stride with One Address per Cycle



Unit-Stride Operations

32b unit-stride, moves 4 elements per cycle
limited by 128b data bus (half of lanes idle)

16b unit-stride moves 8 elements per cycle
saturates both memory bus and 8 lanes' register file ports

8b unit-stride moves 8 elements per cycle
limited by 8 lanes' register file ports (half of memory bus idle)

=>T0 design optimized for 16b unit-stride

(Separate rotate network control for 8b, 16b, and 32b load/stores.)

Strided Operations

Limited by single address port, transfer one element/cycle.

Combination of:

- byte address
- operand size
- active lane

used to control memory crossbar to rotate correct bytes to correct lane.

Indexed Operations

Need to feed indices from vector register file to address generator. (Luckily, only one element per cycle).

Easily the most complex part of T0 design! (source of several early design bugs)

Most of the complexity arose from desire to keep it small in area and reasonably fast.

T0 *dynamically* stretches vector memory pipeline to add extra stages for index read.

T0 time-multiplexes single vector register read port between data and indices for indexed stores.

Vector Extract

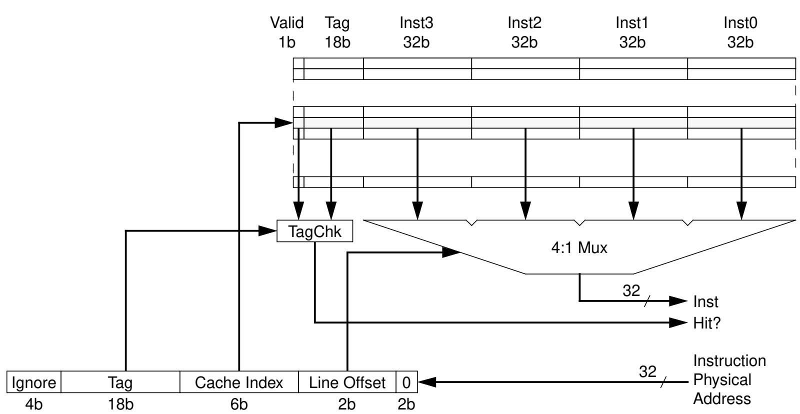
Copies end of one vreg to start of another vreg.
Speed of extract very important for reduction operations.

Executes in vector memory pipeline.

If extract index multiple of 8, use sneak path within lane, move 8 elements per cycle.

Otherwise, use memory crossbar for inter-lane communication (treated like 32-bit store and load occuring simultaneously), move 4 elements per cycle.

Instruction Cache



- 1KB, direct mapped, 16B lines
- Small, because off-chip memory fast
- Autonomous cache refill in F stage during D or X stage interlocks
- Miss steals only one cycle from ongoing vector memory instruction
- Prefetch when memory port idle reduces miss penalty from 3 to 2 cycles
- Ignores high 4b of address => can only map 256MB of instruction memory

Scalar Unit

NOT a commercial core, designed from scratch!

Advantages include:
- Low latency and high bandwidth coupling between scalar and vector
- Removes startup overhead hence allows shorter vector registers
- Simplifies interrupt and exception handling
- Avoids nasty workarounds of awkward interfaces on standard cores
- No time spent reading inaccurate documentation
- Cheap!
- Only our own bugs!

Disadvantages:
- Design time
- Our own bugs!

Scalar Unit (cont.)

MIPS-II compatible 32-bit integer RISC
- runs SGI's Irix4 standard C library object code!

Some MIPS-II instructions omitted (trapped/emulated in kernel):
- Misaligned load/store (not generated by gcc, but present in some assembler libraries - rewrote the libraries)
- Trap on condition (for Ada - not generated by gcc)
- Floating-point coprocessor (too expensive)
- Load-linked/Store-conditional (only for multiprocessors)

Main changes from conventional five-stage RISC core:
- Can send two scalar registers to vector unit in one cycle (base+stride, operand+insert index, operand+config)
- Merged scalar/vector memory pipe requires sixth pipe stage (2 load delay slots)
- Separate dedicated address adder (so vector memory can run in parallel with scalar ALU)

Vector Register File

Each vector arithmetic unit needs 2 reads plus 1 write port.
Vector memory unit needs 1 read and 1 write port.
=>Total requirement, 5 read ports and 3 write ports.

With differential writes and single-ended reads
=>8 address lines, 11 bit lines, 79.5 x 104.5 λ^2 per bit

We used a double-pumping scheme, reads on first phase, writes on second phase of clock
=>5 address lines, 6 bit lines, 57 x 72 λ^2 per bit

Needs tricky self-timed circuit but gave 2x saving in area.
Area limited # vector registers to 16 (Torrent ISA allows 32).

(see Day 3 for other ways to save VRF area)

Arithmetic Pipelines

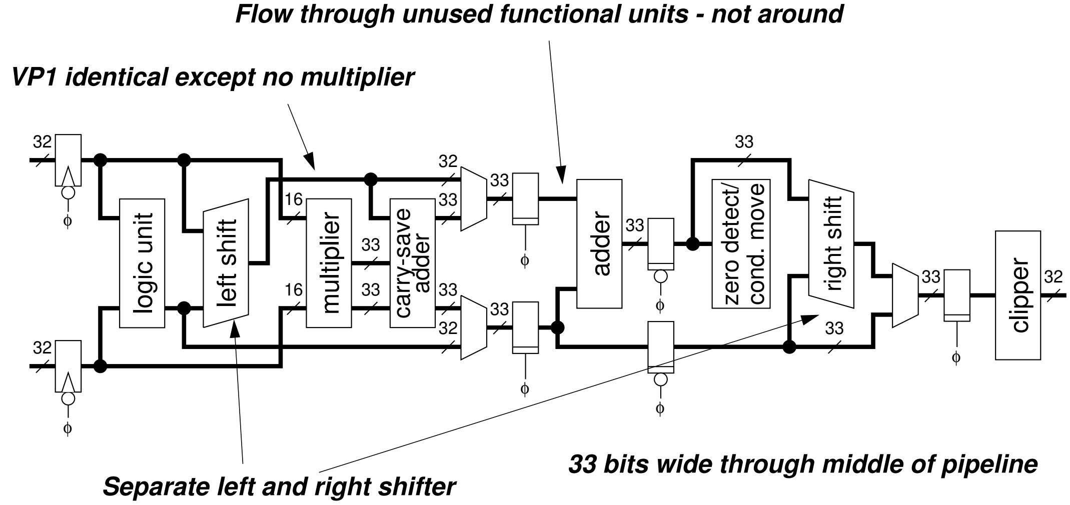
Primary goal: 8 multiply-adds/cycle

16bx16b->32b multiplies and 32b accumulators,

but with fixed-point scaling, rounding, and saturation.

Also wanted basic integer arithmetic, logical, shift operations.

VP0 Arithmetic Pipeline Layout



All units under control of scalar configuration register, e.g.,
logic + left_shift + add + right_shift + clip + conditional_write
=> 6 operations in one vector instruction!

Why Two Asymmetric Arithmetic Units?

Why not 16 lanes with one arithmetic unit each?

Because multiplier array is large: VP0 is >twice area of VP1.

Multiply-adds very common.

Want memory system to run at same speed as arithmetic to simplify chaining (assume doubling memory bandwidth impossible).

Why Not Partitioned Datapaths?

We considered partitioning datapaths to give 16x16b lanes.

Double throughput for some image processing codes with ~10% area overhead.

But 10% too much - already at full reticle.
Our primary application required 32-bit datapaths.

Chaining

T0 has most flexible chaining of any extant vector machine:

- Chains Read-After-Write, Write-After-Read, and Write-After-Write hazards on vector registers.
- Chains between vector instructions running at different rates.
- Chains at any time in instruction execution (no "chain slot" time)

All chaining through VRF storage --- no bypass muxes

- would add area
- would increase complexity of conditional moves
- would only reduce latency, vectors usually bandwidth limited

Control circuit similar to RISC register forwarding/interlock, required only 23 register number comparators.

Made easier by single-chip design with short latencies and fully multiported vector register file.

Day 1, Session D:
Results and Retrospective

Results on benchmark tasks

Things we learned

Things we did right

Things we did wrong

Spinoff projects

Design Results

Near-industrial quality design

- Clock rate 45MHz in 1.0µm
  (compare Intel i860 40MHz in 1.0µm, Cypress SPARC 40MHz in 0.8µm)
- Area comparable to industrial designs
- No bugs in first-pass silicon

Complete working system, still in production use

- With 1990 technology, still faster than 1998 workstations on some apps

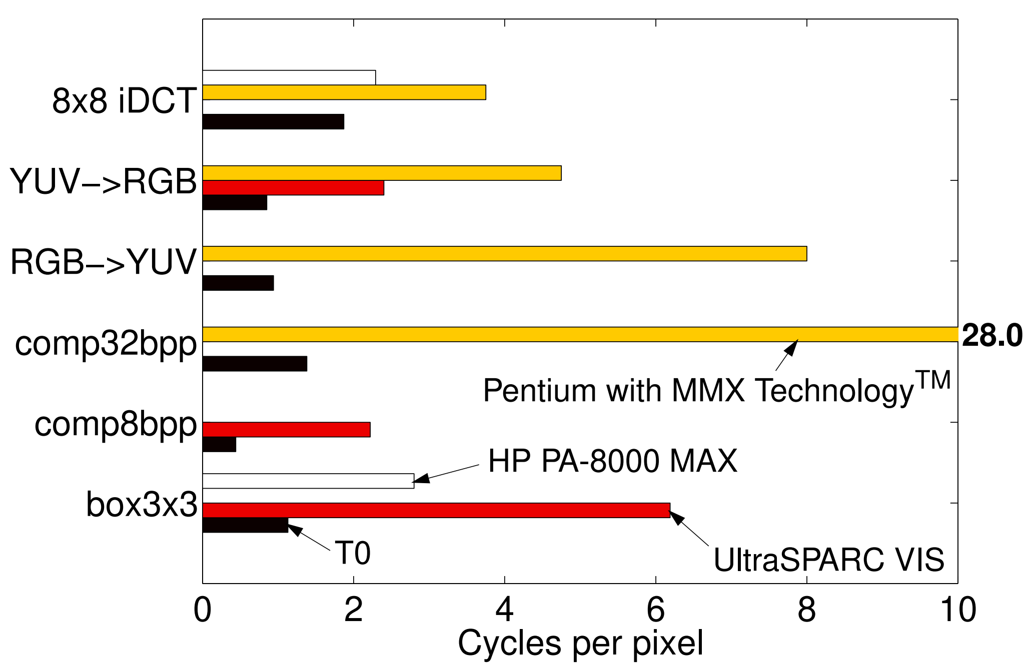
Spert-II Boards at Work

Site	Country	Number of Boards
Faculte Polytechnique de Mons	Belgium	2
Cambridge University	England	4
Sheffield University	England	3
Duisburg University	Germany	1
INESC	Portugal	1
IDIAP	Switzerland	2
ICSI	USA	21
Oregon Graduate Institute	USA	1
UC Berkeley	USA	1

Train 400,000 weight artificial neural network for speech recognition:

Sun Ultra-1/170 takes ~20 days
Spert-II takes ~20 hours

T0 vs. MiMX^{TM}: Image Kernels



Vectors vs. MiMX^{TM}

Instruction issue bandwidth/dependency checking

- 1 T0 instruction specifies 32x32b = 1024b of datapath work
- 1 MiMX^{TM} instruction specifies 64b of datapath work

Alignment/Packing

- Vector machine: Load vector
- MiMX^{TM}: Load surrounding words, align, unpack

Registers

- Vector Machine: Vector length **multiplies** number of registers available
- MiMX^{TM}: Loop unrolling **divides** number of registers available

Vector Length?

Non-Unit Stride and Scatter/Gather?

Other Application Examples

IDEA Cryptography

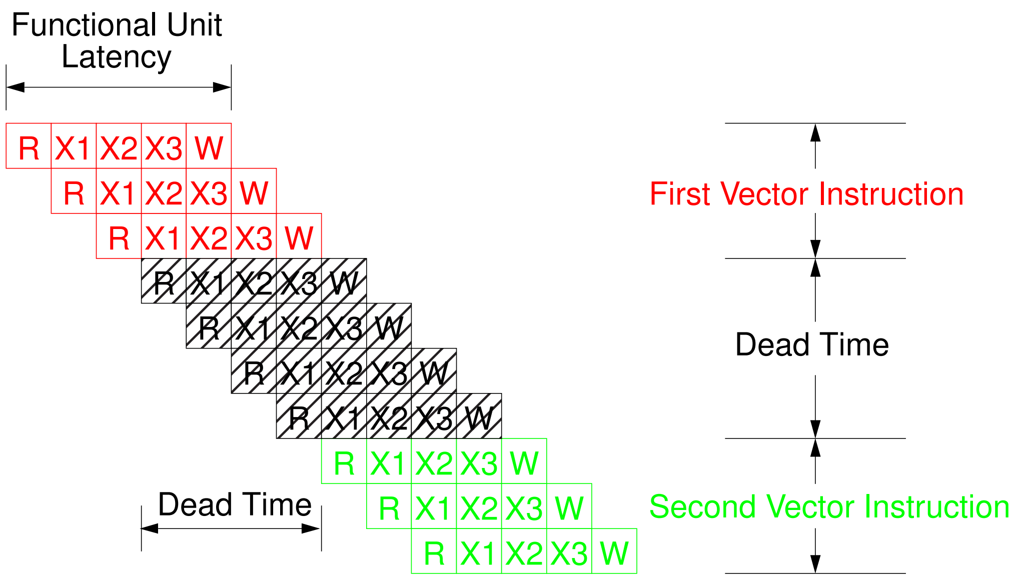
	Decryption rate (MB/s)
T0 @ 40MHz	13
Alpha 21164 @ 500MHz	4

Additive Audio Synthesis

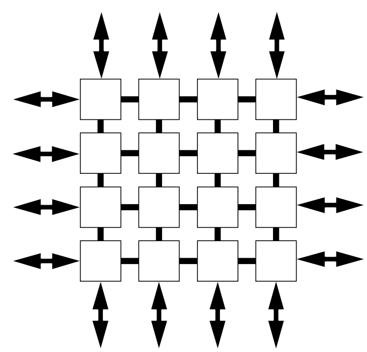
	# Real-time oscillators
T0 @ 40MHz	600
MIPS R10K @ 180MHz	1000

Vector Startup Latency

Two Types: Functional Unit Latency and Dead Time



Vector Mainframe Design Dilemma



Single Chip CPU
Low Intra-CPU Latencies
but
Low Memory Bandwidth

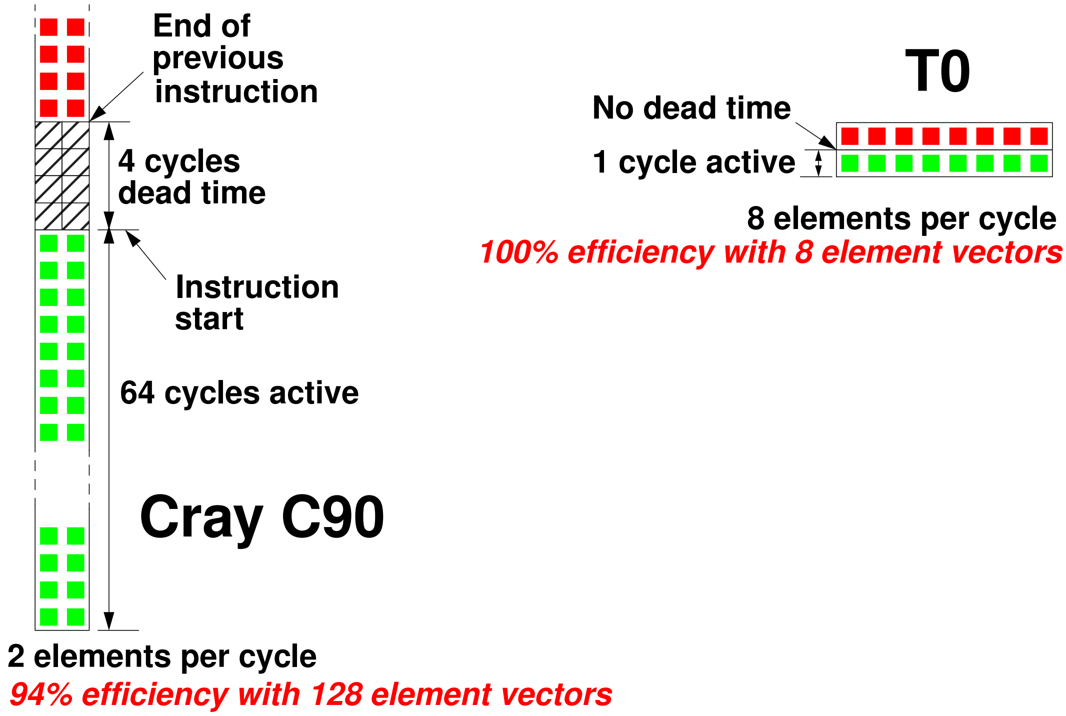
Greater Performance/$

Multi-Chip CPU
High Intra-CPU Latencies
but
High Memory Bandwidth

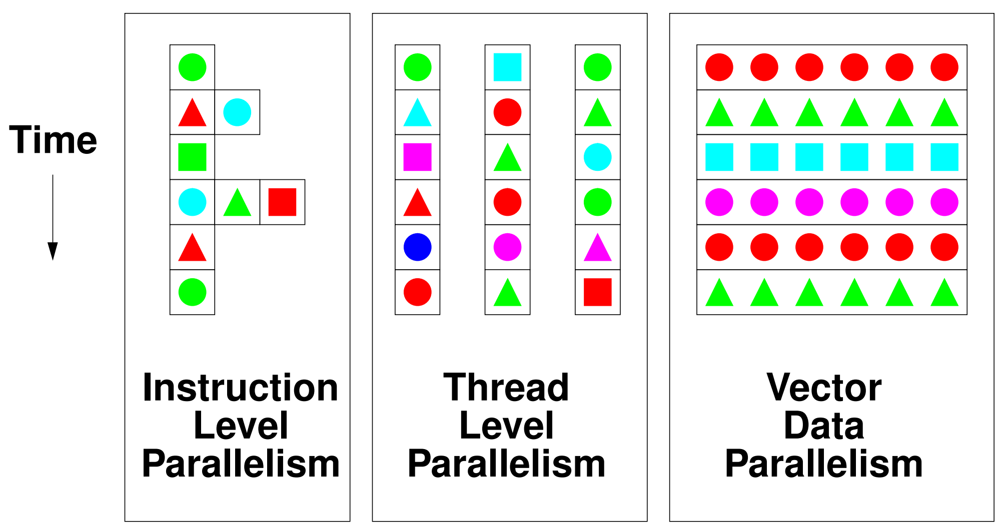
Greater Single CPU Performance

Supercomputer Customers' Choice

No Dead Time => Shorter Vectors



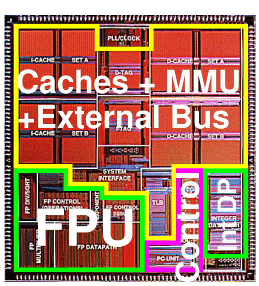
Forms of Processor Parallelism



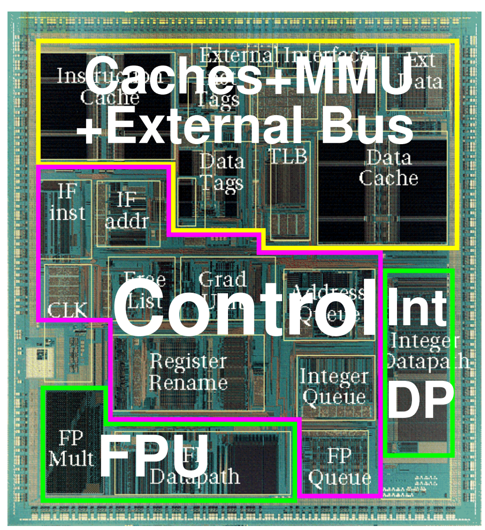
MIPS R5K vs. R10K on SPECint95

	SGI O2 R5K	SGI Origin-200 R10K	R10K/R5K Ratio
Clock Rate	180 MHz		
Process	0.35 µm		
L1 cache I/D	32KB/32KB, 2-way		
L2 cache	512KB 1-way, 90MHz	1MB 2-way, 120MHz	
Compiler	MIPSPRO7.1		
Execution	In-Order	Out-of-Order	
Branch Prediction?	No	Yes	
Non-Blocking Caches	No	Yes	
Integer insts/cycle	**1**	**3**	**3**
SPECint95 (base)	**4.76**	**7.85**	**1.65**
SPECint95 (peak)	**4.82**	**8.59**	**1.78**
Die Area (mm^2)	**87**	**298**	**3.43**
CPU Area (mm^2)	**~33**	**~162**	**~4.9**

Superscalar Has High Control Complexity

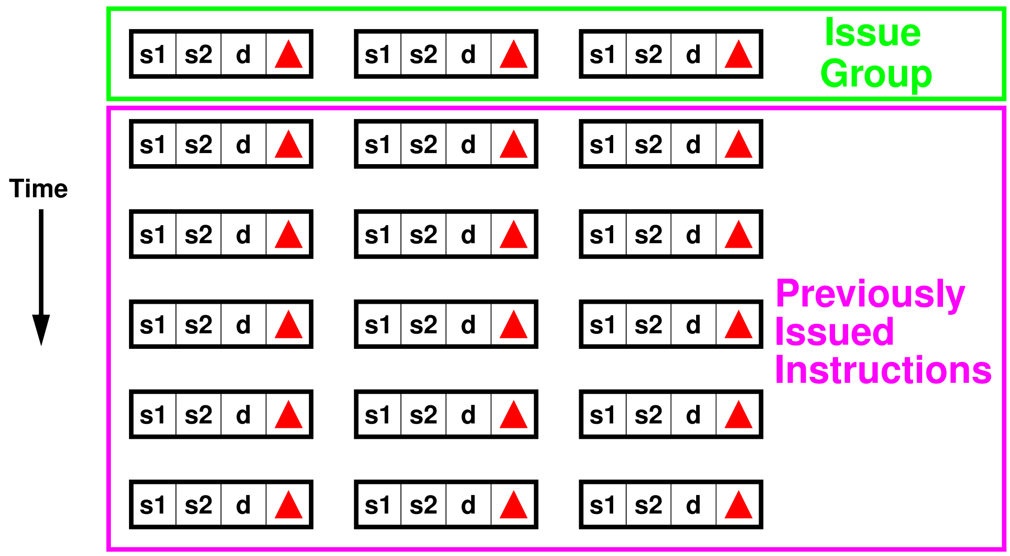


R5000

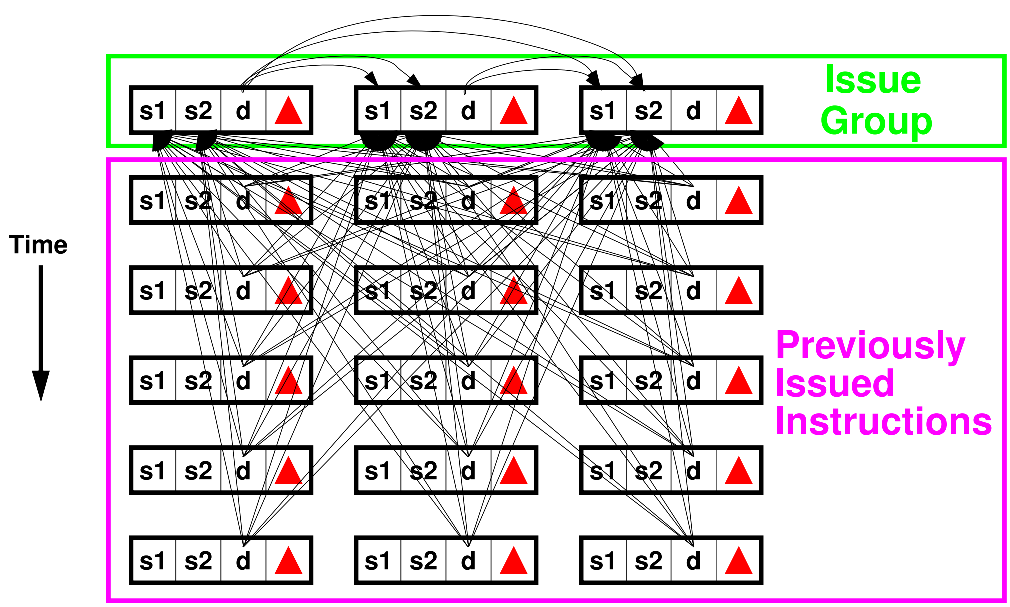


R10000

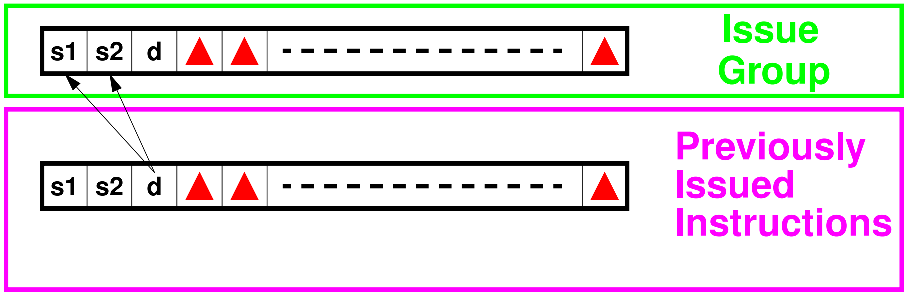
Superscalar Control Complexity



Superscalar Control Complexity

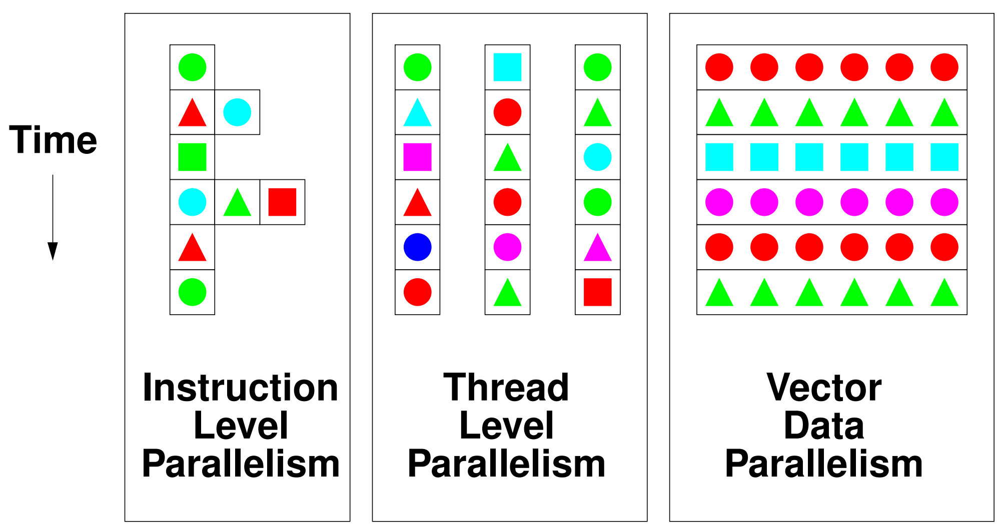


Vectors Have Low Control Complexity



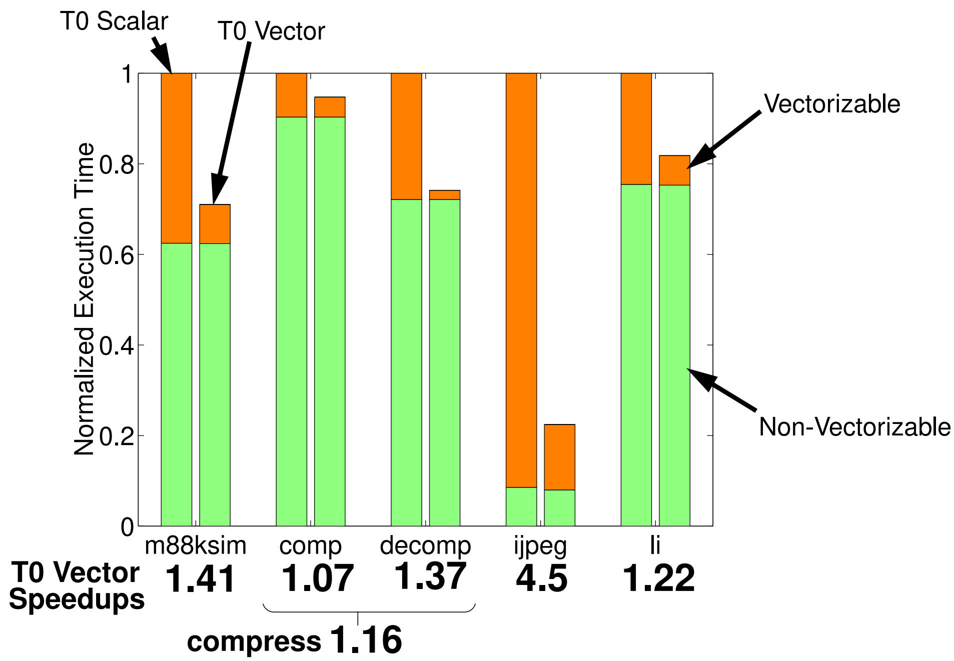
	T0	HP PA-8000
Operations/Cycle	24	6
5-bit Register Comparators	23	6,720

Forms of Processor Parallelism



Combine all forms of parallelism for best cost/performance

Vectorizing SPECint95

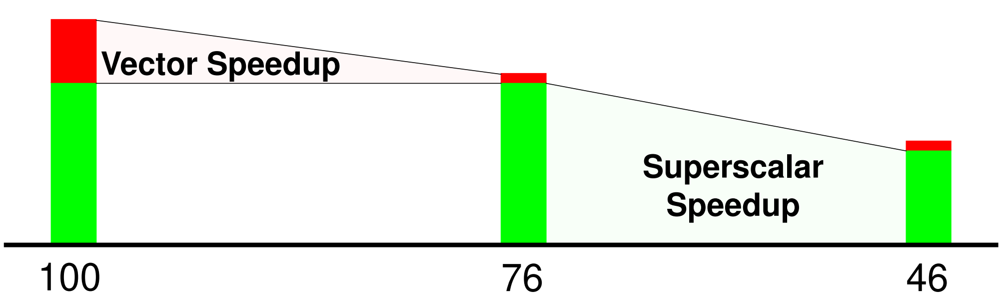


Mean vector speedup over all 8 SPECint95 benchmarks: **1.32**

Combining Vector and Superscalar Speedups on SPECint95

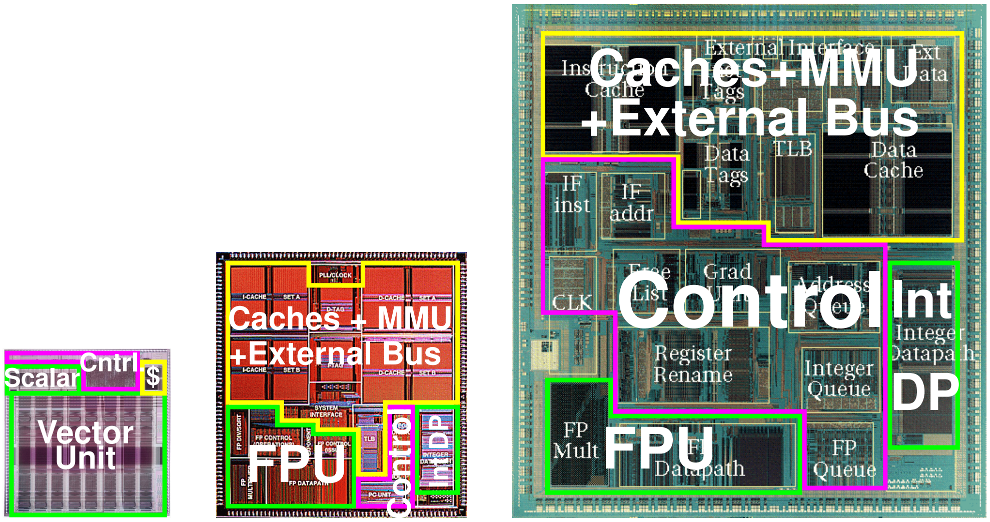
Vector unit (T0) speeds up 28% by factor of 8, speedup is **1.32**
Superscalar (R10K) speeds up 100% by factor of **1.7**

Vector + superscalar has combined speedup of **2.18**



1.28x greater than superscalar alone!

Vectors are Cheap!



T0 R5000 R10000

Die scaled to same feature size

What We Learned

Vectors *are* inflexible - cheap, sufficient for many future tasks

Vectors same startup as scalar

Vectors short - just long enough to keep machine busy

Vector registers tiny

Vectors cheap

Overall:

Vectors are best way of executing data parallel code

Things We Did Right

Focus on one idea
- No multiprocessor support
- No threading

Use industry standard ISA
- Huge software advantage (compiler, assembler, linker, debugger)

Build our own MIPS core
- Could build exactly what we needed
- No imported bugs
- Simplified design flow

Design complete system (even at start of project)
- Make chip more complicated to simplify software and board design

General-purpose machine rather than application-specific
- Could get many more benchmark results
- Can accomodate unforseen changes in main application

Things We Did Wrong (Architecture)

Software-visible vector length
- Should allow stripmine loops to be written independent of vector length

Conditional move rather than masked execution
- Can't mask loads and stores
- Can't mask saturation/overflow events
- Takes up whole vector data register for flag vector

Fixed-point pipeline problems
- Should have had right shift before adder and no shift on multiplier output
- Should have logic unit data in series with shifter for reconfigurable ops
- Minor rounding inconsistency with variable shift of zero

Unit-stride auto address increment doesn't happen if vlr=0
- Should always happen regardless of vlr to avoid long path in control logic

Things We Did Wrong (Microarch)

Instruction fetch should have been more aggressive

- Could prefetch *next* sequential line when memory port idle
- Could avoid caching lines that we could prefetch

Aligned unit-stride loads should have extra cycle latency to make same as misaligned

- Would remove source of stall
- Would avoid need to check for WAW hazard between load and ALU

Should have made I/O path burst 32 bytes not just 16

- Would give higher I/O bandwidth with little increase in area

Should have put HPM counters on chip with software access

- Too much effort to add hardware and software path outside chip
- Never finished or used

Spin-Off Projects

Multi-Spert (Philipp Faerber, ICSI)

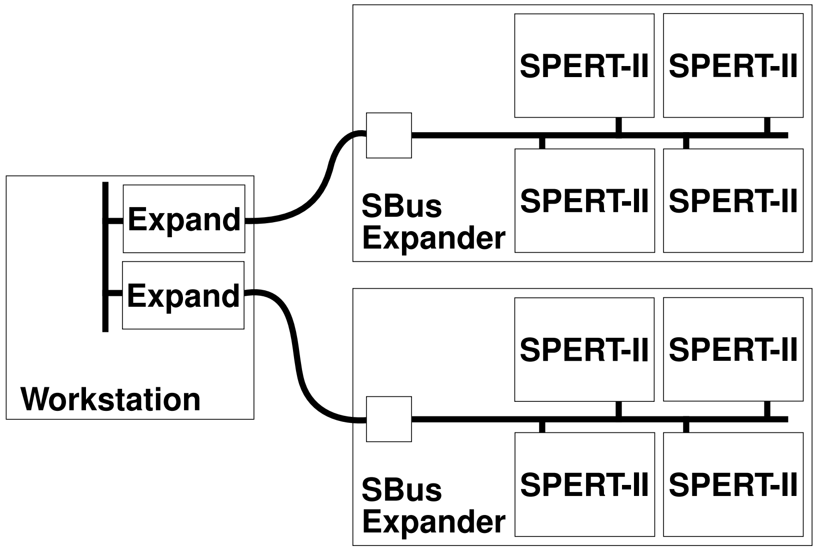
Vector IRAM (UCB IRAM group)

UCB RISC Core (Willy Chang, UCB)

Vector software studies:

- audio synthesis (Todd Hodes, UCB/CNMAT)
- hash-join (Rich Martin, UCB)
- speech decoding (Dan Gildea, UCB/ICSI)
- image processing (Chris Bregler, UCB/ICSI)

MultiSpert Configuration



ICSI running two 4-node and one 2-node Multi-Sperts.