Lecture 6: Vector Processing

Professor David A. Patterson

Computer Science 252

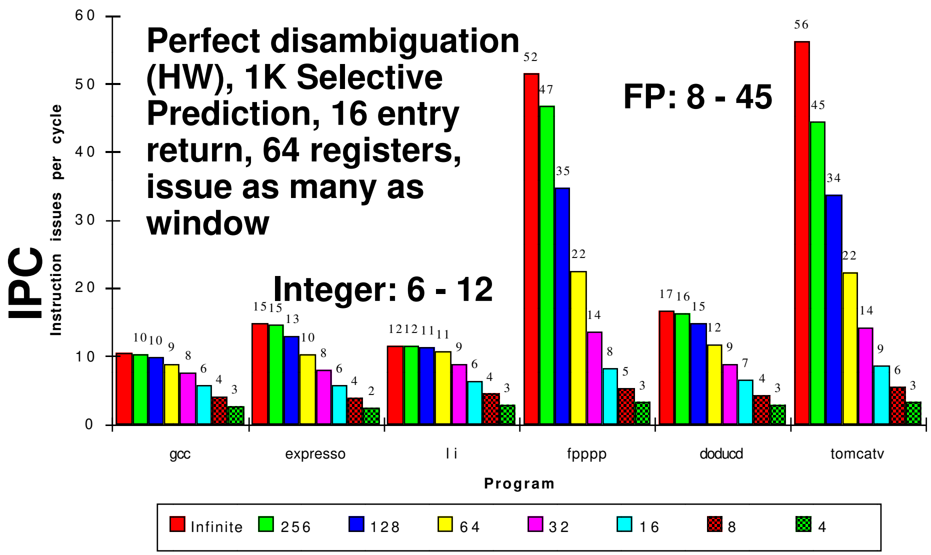
Spring 1998

Review

- Speculation: Out-of-order execution, In-order commit (reorder buffer+rename registers)=>precise exceptions
- Branch Prediction
  - Branch History Table: 2 bits for loop accuracy
  - Recently executed branches correlated with next branch?
  - Branch Target Buffer: include branch address & prediction
  - Predicated Execution can reduce number of branches, number of mispredicted branches
- Software Pipelining
  - Symbolic loop unrolling (instructions from different iterations) to optimize pipeline with little code expansion, little overhead
- Superscalar and VLIW("EPIC"): CPI < 1 (IPC > 1)
  - Dynamic issue vs. Static issue
  - More instructions issue at same time => larger hazard penalty
  - # independent instructions = # functional units X latency

Review: Theoretical Limits to ILP?

(Figure 4.48, Page 332)



Infinite 256 128 64 32 16 8 4

Review: Instructon Level Parallelism

- High speed execution based on *instruction level parallelism* (ilp): potential of short instruction sequences to execute in parallel
- High-speed microprocessors exploit ILP by:
  1) pipelined execution: overlap instructions
  2) superscalar execution: issue and execute multiple instructions per clock cycle
  3) Out-of-order execution (commit in-order)
- Memory accesses for high-speed microprocessor?
  - Data Cache, possibly multiported, multiple levels

Problems with conventional approach

- Limits to conventional exploitation of ILP:

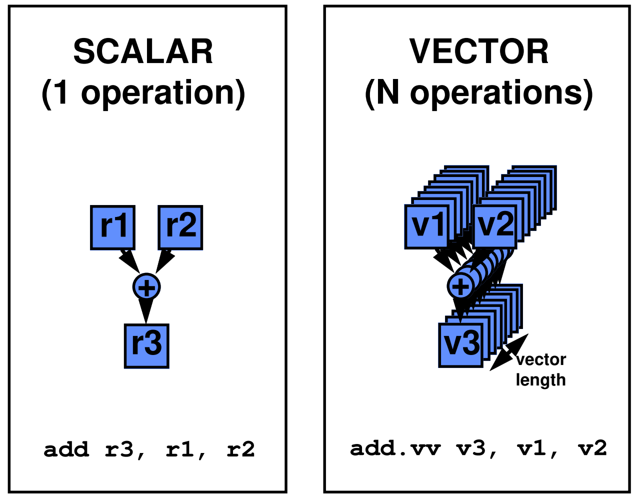
  1) ***pipelined clock rate***: at some point, each increase in clock rate has corresponding CPI increase (branches, other hazards)

  2) ***instruction fetch and decode***: at some point, its hard to fetch and decode more instructions per clock cycle

  3) ***cache hit rate***: some long-running (scientific) programs have very large data sets accessed with poor locality;
  others have continuous data streams (multimedia) and hence poor locality

Alternative Model: Vector Processing

- Vector processors have high-level operations that work on linear arrays of numbers: "vectors"



Properties of Vector Processors

- Each result independent of previous result
  => long pipeline, compiler ensures no dependencies
  => high clock rate
- Vector instructions access memory with known pattern
  => highly interleaved memory
  => amortize memory latency of over ≈ 64 elements
  => no (data) caches required! (Do use instruction cache)
- Reduces branches and branch problems in pipelines
- Single vector instruction implies lots of work (≈ loop)
  => fewer instruction fetches

Operation & Instruction Count: RISC v. Vector Processor

(from F. Quintana, U. Barcelona.)

Spec92fp	Operations (Millions)			Instructions (M)		
Program	RISC	Vector	R / V	RISC	Vector	R / V
swim256	115	95	1.1x	115	0.8	142x
hydro2d	58	40	1.4x	58	0.8	71x
nasa7	69	41	1.7x	69	2.2	31x
su2cor	51	35	1.4x	51	1.8	29x
tomcatv	15	10	1.4x	15	1.3	11x
wave5	27	25	1.1x	27	7.2	4x
mdljdp2	32	52	0.6x	32	15.8	2x

Vector reduces ops by 1.2X, instructions by 20X

Styles of Vector Architectures

- *memory-memory vector processors*: all vector operations are memory to memory
- *vector-register processors*: all vector operations between vector registers (except load and store)
  – Vector equivalent of load-store architectures
  – Includes all vector machines since late 1980s: Cray, Convex, Fujitsu, Hitachi, NEC
  – We assume vector-register for rest of lectures

Components of Vector Processor

- *Vector Register*: fixed length bank holding a single vector
  – has at least 2 read and 1 write ports
  – typically 8-32 vector registers, each holding 64-128 64-bit elements
- *Vector Functional Units (FUs)*: fully pipelined, start new operation every clock
  – typically 4 to 8 FUs: FP add, FP mult, FP reciprocal (1/X), integer add, logical, shift; may have multiple of same unit
- *Vector Load-Store Units (LSUs)*: fully pipelined unit to load or store a vector; may have multiple LSUs
- *Scalar registers*: single element for FP scalar or address
- Cross-bar to connect FUs , LSUs, registers

"DLXV" Vector Instructions

Instr.	Operands	Operation	Comment
• `ADDV`	`V1,V2,V3`	V1=V2+V3	vector + vector
• `ADDSV`	`V1,F0,V2`	V1=F0+V2	scalar + vector
• `MULTV`	`V1,V2,V3`	V1=V2xV3	vector x vector
• `MULSV`	`V1,F0,V2`	V1=F0xV2	scalar x vector
• `LV`	`V1,R1`	V1=M[R1..R1+63]	load, stride=1
• `LVWS`	`V1,R1,R2`	V1=M[R1..R1+63*R2]	load, stride=R2
• `LVI`	`V1,R1,V2`	V1=M[R1+V2i,i=0..63]	indir.("gather")
• `CeqV`	`VM,V1,V2`	VMASKi = (V1i=V2i)?	comp. setmask
• `MOV`	`VLR,R1`	Vec. Len. Reg. = R1	set vector length
• `MOV`	`VM,R1`	Vec. Mask = R1	set vector mask

Memory operations

- Load/store operations move groups of data between registers and memory
- Three types of addressing
  – Unit stride
    » Fastest
  – Non-unit (constant) stride
  – Indexed (gather-scatter)
    » Vector equivalent of register indirect
    » Good for sparse arrays of data
    » Increases number of programs that vectorize

DAXPY (Y = a * X + Y)

Assuming vectors X, Y are length 64

Scalar vs. Vector

```
LD     F0,a       ;load scalar a
LV     V1,Rx      ;load vector X
MULTS  V2,F0,V1   ;vector-scalar mult.
LV     V3,Ry      ;load vector Y
ADDV   V4,V2,V3   ;add
SV     Ry,V4      ;store the result
```

```
      LD     F0,a
      ADDI   R4,Rx,#512  ;last address to load
loop: LD     F2, 0(Rx)   ;load X(i)
      MULTD  F2,F0,F2    ;a*X(i)
      LD     F4, 0(Ry)   ;load Y(i)
      ADDD   F4,F2, F4   ;a*X(i) + Y(i)
      SD     F4 ,0(Ry)   ;store into Y(i)
      ADDI   Rx,Rx,#8    ;increment index to X
      ADDI   Ry,Ry,#8    ;increment index to Y
      SUB    R20,R4,Rx   ;compute bound
      BNZ    R20,loop    ;check if done
```

578 (2+9*64) vs. 321 (1+5*64) ops (1.8X)

578 (2+9*64) vs. 6 instructions (96X)

64 operation vectors + no loop overhead

also 64X fewer pipeline hazards

Example Vector Machines

Machine	Year	Clock	Regs	Elements	FUs	LSUs
Cray 1	1976	80 MHz	8	64	6	1
Cray XMP	1983	120 MHz	8	64	8	2 L, 1 S
Cray YMP	1988	166 MHz	8	64	8	2 L, 1 S
Cray C-90	1991	240 MHz	8	128	8	4
Cray T-90	1996	455 MHz	8	128	8	4
Conv. C-1	1984	10 MHz	8	128	4	1
Conv. C-4	1994	133 MHz	16	128	3	1
Fuj. VP200	1982	133 MHz	8-256	32-1024	3	2
Fuj. VP300	1996	100 MHz	8-256	32-1024	3	2
NEC SX/2	1984	160 MHz	8+8K	256+var	16	8
NEC SX/3	1995	400 MHz	8+8K	256+var	16	8

Vector Linpack Performance (MFLOPS)

Machine	Year	Clock	100x100	1kx1k	Peak(Procs)
Cray 1	1976	80 MHz	12	110	160(1)
Cray XMP	1983	120 MHz	121	218	940(4)
Cray YMP	1988	166 MHz	150	307	2,667(8)
Cray C-90	1991	240 MHz	387	902	15,238(16)
Cray T-90	1996	455 MHz	705	1603	57,600(32)
Conv. C-1	1984	10 MHz	3	--	20(1)
Conv. C-4	1994	135 MHz	160	2531	3240(4)
Fuj. VP200	1982	133 MHz	18	422	533(1)
NEC SX/2	1984	166 MHz	43	885	1300(1)
NEC SX/3	1995	400 MHz	368	2757	25,600(4)

CS 252 Administrivia

- Get your photo taken by Joe Gebis! (or give URL)
- Exercises for Lectures 3 to 7
  - Due Thursday Febuary 12 at 5PM homework box in 283 Soda (building is locked at 6:45 PM)
  - 4.2, 4.10, 4.19, 4.14 parts c) and d) only, B.2
  - Done in pairs, but both need to understand whole assignment; Anyone need a partner?
  - Study groups encouraged, but pairs do own work
  - Turn in (copy of)photo with name on it (phonetic spelling, if useful)
- Select projects by next Monday! Need partner too. Send email to TA, me saying what and who
  - Start now doing small things to get setup done

Computers in the News

- IBM researchers announced (at ISSCC '98) they have demonstrated the world's first experimental CMOS microprocessor that can operate at 1000 MHz
- The chip contains 1 million transistors and uses 0.25-micron circuit technology
- Integer only, 4 stage pipeline, + caches; innovations include:
  - A multifunctional unit, which combines addition and rotation operations into a single circuit
  - An innovative cache design, which combines the address calculation with the array access function
  - A dynamic circuit approach that reduced the number of stages through which signals must propagate
- Experimental only (like 4Gbit DRAM prototypes)

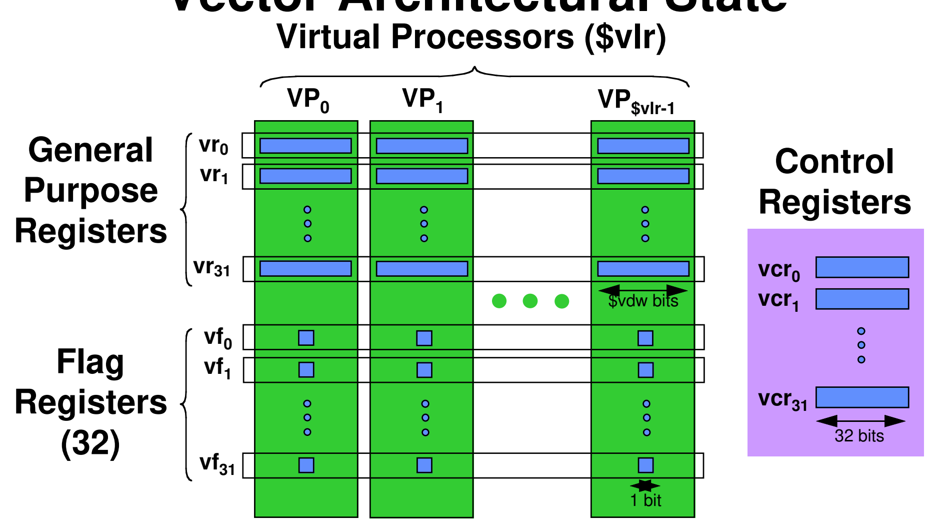
Vector Surprise

- Use vectors for inner loop parallelism (no surprise)
  - One dimension of array: A[0, 0], A[0, 1], A[0, 2], ...
  - think of machine as, say, 32 vector regs each with 64 elements
  - 1 instruction updates 64 elements of 1 vector register
- and for outer loop parallelism!
  - 1 element from each column: A[0,0], A[1,0], A[2,0], ...
  - think of machine as 64 "virtual processors" (VPs) each with 32 scalar registers! (≈ multithreaded processor)
  - 1 instruction updates 1 scalar register in 64 VPs
- Hardware identical, just 2 compiler perspectives

Virtial Processor Vector Model

- Vector operations are SIMD (single instruction multiple data)operations
- Each element is computed by a virtual processor (VP)
- Number of VPs given by vector length
  - vector control register

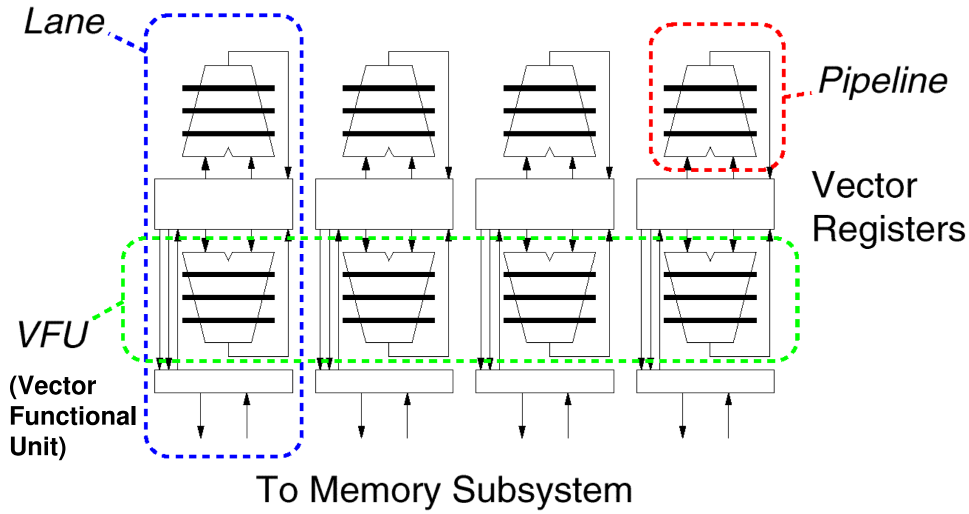
Vector Architectural State



Vector Implementation

- Vector register file
  - Each register is an array of elements
  - Size of each register determines maximum vector length
  - Vector length register determines vector length for a particular operation
- Multiple parallel execution units = "<u>lanes</u>" (sometimes called "<u>pipelines</u>" or "<u>pipes</u>")

Vector Terminology: 4 lanes, 2 vector functional units



Vector Execution Time

- Time = f(vector length, data dependicies, struct. hazards)
- *Initiation rate*: rate that FU consumes vector elements (= number of lanes; usually 1 or 2 on Cray T-90)
- *Convoy*: set of vector instructions that can begin execution in same clock (no struct. or data hazards)
- *Chime*: approx. time for a vector operation
- <u>*m* convoys take *m* chimes</u>; if each vector length is n, then they take approx. *m* x *n* clock cycles (ignores overhead; good approximization for long vectors)

```
1:  LV      V1,Rx        ;load vector X
2:  MULV    V2,F0,V1     ;vector-scalar mult.
    LV      V3,Ry        ;load vector Y
3:  ADDV    V4,V2,V3     ;add
4:  SV      Ry,V4        ;store the result
```

4 conveys, 1 lane, VL=64
=> 4 x 64 ≈ 256 clocks
(or 4 clocks per result)

DLXV Start-up Time

- *Start-up time*: pipeline latency time (depth of FU pipeline); another sources of overhead
- Operation Start-up penalty (from CRAY-1)
- Vector load/store 12
- Vector multply 7
- Vector add 6

Assume convoys don't overlap; vector length = n:

Convoy	*Start*	*1st result*	*last result*	
1. LV	0	12	11+n (12+n-1)	
2. MULV, LV	12+n	12+n+12	23+2n	*Load start-up*
3. ADDV	24+2n	24+2n+6	29+3n	*Wait convoy 2*
4. SV	30+3n	30+3n+12	41+4n	*Wait convoy 3*

Why startup time for each vector instruction?

- Why not overlap startup time of back-to-back vector instructions?
- Cray machines built from many ECL chips operating at high clock rates; hard to do?
- Berkeley vector design ("T0") didn't know it wasn't supposed to do overlap, so no startup times for functional units (except load)

Vector Load/Store Units & Memories

- Start-up overheads usually longer fo LSUs
- Memory system must sustain (# lanes x word) /clock cycle
- Many Vector Procs. use banks (vs. simple interleaving):

  1) support multiple loads/stores per cycle
  => multiple banks & address banks independently

  2) support non-sequential accesses (see soon)
- Note: No. memory banks > memory latency to avoid stalls
  - *m* banks => *m* words per memory lantecy *l* clocks
  - if *m* < *l*, then gap in memory pipeline:

clock:	0	…	*l*	*l*+1	*l*+2	…	*l*+*m*- 1	<u>l+m</u>	…	2 *l*
word:	--	…	0	1	2	…	*m*-1	--	…	*m*

  - may have 1024 banks in SRAM

Vector Length

- What to do when vector length is not exactly 64?
- *vector-length register* (VLR) controls the length of any vector operation, including a vector load or store. (cannot be > the length of vector registers)

```
     do 10 i = 1, n
10   Y(i) = a * X(i) + Y(i)
```

- Don't know n until runtime!
  n > Max. Vector Length (MVL)?

Strip Mining

- Suppose Vector Length > Max. Vector Length (MVL)?
- *Strip mining*: generation of code such that each vector operation is done for a size $\leq$ to the MVL
- 1st loop do short piece (n mod MVL), rest VL = MVL

```
  low = 1
  VL = (n mod MVL)  /*find the odd size piece*/
  do 1 j = 0,(n / MVL)  /*outer loop*/
     do 10 i = low,low+VL-1  /*runs for length VL*/
            Y(i) = a*X(i) + Y(i)  /*main operation*/
10   continue
     low = low+VL  /*start of next vector*/
     VL = MVL  /*reset the length to max*/
1    continue
```

Common Vector Metrics

- R_∞: MFLOPS rate on an infinite-length vector
  - vector "speed of light"
  - Real problems do not have unlimited vector lengths, and the start-up penalties encountered in real problems will be larger
  - (R_n is the MFLOPS rate for a vector of length n)
- $N_{1/2}$: The vector length needed to reach one-half of R_∞
  - a good measure of the impact of start-up
- N_V: The vector length needed to make vector mode faster than scalar mode
  - measures both start-up and speed of scalars relative to vectors, quality of connection of scalar unit to vector unit

Vector Stride

- Suppose adjacent elements not sequential in memory

```
do 10 i = 1,100
    do 10 j = 1,100
          A(i,j) = 0.0
          do 10 k = 1,100
10                  A(i,j) = A(i,j)+B(i,k)*C(k,j)
```

- Either B or C accesses not adjacent (800 bytes between)
- *stride*: distance separating elements that are to be merged into a single vector (caches do <u>unit stride</u>) => `LVWS` (load vector with stride) instruction
- Strides => can cause bank conflicts (e.g., stride = 32 and 16 banks)
- Think of address per vector element

Compiler Vectorization on Cray XMP

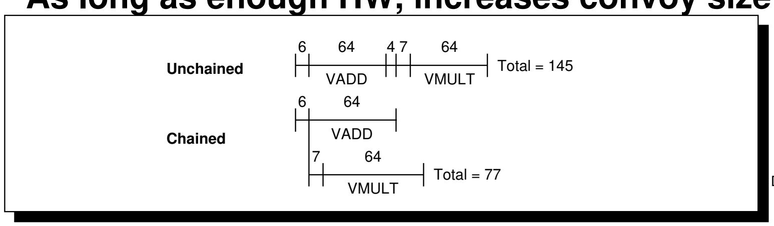
Benchmark	%FP	%FP in vector	
ADM	23%	68%	
DYFESM	26%	95%	
FLO52	41%	100%	
MDG	28%	27%	
MG3D	31%	86%	
OCEAN	28%	58%	
QCD	14%	1%	
SPICE	16%	7%	(1% overall)
TRACK	9%	23%	
TRFD	22%	10%	

Vector Opt #1: Chaining

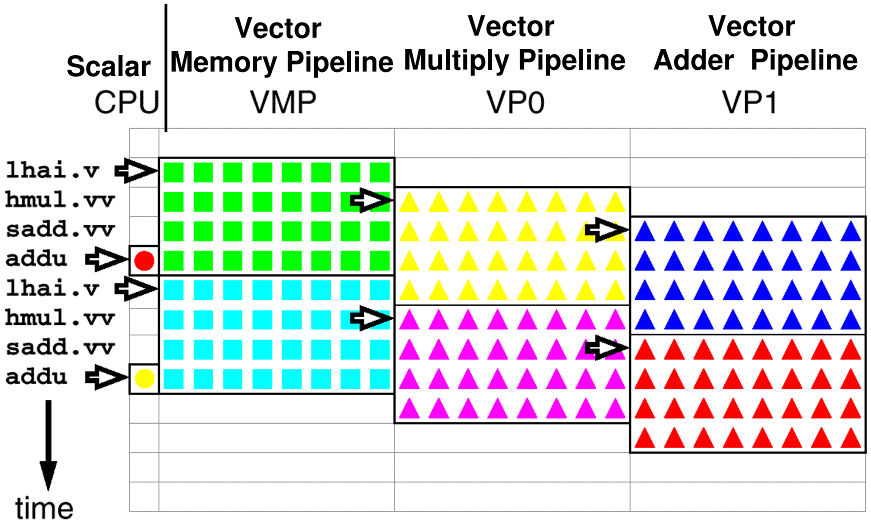
- Suppose:

  MULV <u>V1</u>,V2,V3

  ADDV V4,<u>*V1*</u>,V5 ; separate convoy?
- *chaining*: vector register (V1) is not as a single entity but as a group of individual registers, then <u>pipeline forwarding can work on individual elements of a vector</u>
- *Flexible chaining*: allow vector to chain to any other active vector operation => more read/write port
- As long as enough HW, increases convoy size



Example Execution of Vector Code



8 lanes, vector length 32, chaining

Operations

Instruction issue

Vector Opt #2: Conditional Execution

- **Suppose:**

```
      do 100 i = 1, 64
             if (A(i) .ne. 0) then
                    A(i) = A(i) – B(i)
             endif
100 continue
```

- ***vector-mask control*** **takes a Boolean vector: when** ***vector-mask register*** **is loaded from vector test, vector instructions operate only on vector elements whose corresponding entries in the vector-mask register are 1.**
- **Still requires clock even if result not stored; if still performs operation, what about divide by 0?**

Vector Opt #3: Sparse Matrices

- **Suppose:**

```
              do   100 i = 1,n
100           A(K(i)) = A(K(i)) + C(M(i))
```

- ***gather*** **(`LVI`) operation takes an** ***index vector*** **and fetches the vector whose elements are at the addresses given by adding a base address to the offsets given in the index vector => <u>a nonsparse vector in a vector register</u>**
- **After these elements are operated on in dense form, the sparse vector can be stored in expanded form by a** ***scatter*** **store (`SVI`), using the same index vector**
- **Can't be done by compiler since can't know Ki elements distinct, no dependencies; by compiler directive**
- **Use `CVI` to create index 0, 1xm, 2xm, ..., 63xm**

Sparse Matrix Example

- **Cache (1993) vs. Vector (1988)**

	IBM RS6000	Cray YMP
Clock	72 MHz	167 MHz
Cache	256 KB	0.25 KB
Linpack	140 MFLOPS	160 (1.1)
Sparse Matrix (Cholesky Blocked)	17 MFLOPS	125 (7.3)

- **Cache: 1 address per cache block (32B to 64B)**
- **Vector: 1 address per element (4B)**

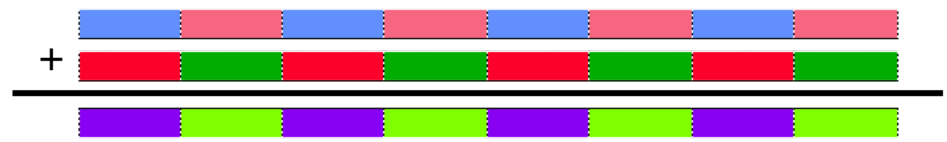
Applications

Limited to scientific computing?

- **Multimedia Processing** (compress., graphics, audio synth, image proc.)
- **Standard benchmark kernels** (Matrix Multiply, FFT, Convolution, Sort)
- **Lossy Compression** (JPEG, MPEG video and audio)
- **Lossless Compression** (Zero removal, RLE, Differencing, LZW)
- **Cryptography** (RSA, DES/IDEA, SHA/MD5)
- **Speech and handwriting recognition**
- **Operating systems/Networking** (`memcpy`, `memset`, parity, checksum)
- **Databases** (hash/join, data mining, image/video serving)
- **Language run-time support** (stdlib, garbage collection)
- **even SPECint95**

Vector for Multimedia?

- Intel MMX: 57 new 80x86 instructions (1st since 386)
  - similar to Intel 860, Mot. 88110, HP PA-71000LC, UltraSPARC
- 3 data types: 8 8-bit, 4 16-bit, 2 32-bit in 64bits
  - reuse 8 FP registers (FP and MMX cannot mix)
- ≈ short vector: load, add, store 8 8-bit operands



- Claim: overall speedup 1.5 to 2X for 2D/3D graphics, audio, video, speech, comm., ...
  - use in drivers or added to library routines; no compiler

MMX Instructions

- Move 32b, 64b
- Add, Subtract in parallel: 8 8b, 4 16b, 2 32b
  - opt. signed/unsigned saturate (set to max) if overflow
- Shifts (sll,srl, sra), And, And Not, Or, Xor in parallel: 8 8b, 4 16b, 2 32b
- Multiply, Multiply-Add in parallel: 4 16b
- Compare = , > in parallel: 8 8b, 4 16b, 2 32b
  - sets field to 0s (false) or 1s (true); removes branches
- Pack/Unpack
  - Convert 32b<–> 16b, 16b <–> 8b
  - Pack saturates (set to max) if number is too large

Vectors and Variable Data Width

- Programmer thinks in terms of vectors of data of some width (8, 16, 32, or 64 bits)
- Good for multimedia; More elegant than MMX-style extensions
- Don't have to worry about how data stored in hardware
  - No need for explicit pack/unpack operations
- Just think of more virtual processors operating on narrow data
- Expand Maximum Vector Length with decreasing data width:
  64 x 64bit, 128 x 32 bit, 256 x 16 bit, 512 x 8 bit

Mediaprocesing: Vectorizable? Vector Lengths?

Kernel	Vector length
• Matrix transpose/multiply	# vertices at once
• DCT (video, communication)	image width
• FFT (audio)	256-1024
• Motion estimation (video)	image width, iw/16
• Gamma correction (video)	image width
• Haar transform (media mining)	image width
• Median filter (image processing)	image width
• Separable convolution (img. proc.)	image width

(from Pradeep Dubey - IBM,
`http://www.research.ibm.com/people/p/pradeep/tutor.html)`

Vector Pitfalls

- Pitfall: Concentrating on peak performance and ignoring start-up overhead: N_V (length faster than scalar) > 100!
- Pitfall: Increasing vector performance, without comparable increases in scalar performance (Amdahl's Law)
  - failure of Cray competitor from his former company
- Pitfall: Good processor vector performance without providing good memory bandwidth
  - MMX?

Vector Advantages

- Easy to get high performance; N operations:
  - are independent
  - use same functional unit
  - access disjoint registers
  - access registers in same order as previous instructions
  - access contiguous memory words or known pattern
  - can exploit large memory bandwidth
  - hide memory latency (and any other latency)
- Scalable (get higher performance as more HW resources available)
- Compact: Describe N operations with 1 short instruction (v. VLIW)
- Predictable (real-time) performance vs. statistical performance (cache)
- Multimedia ready: choose N * 64b, 2N * 32b, 4N * 16b, 8N * 8b
- Mature, developed compiler technology
- Vector Disadvantage: Out of Fashion

Vector Summary

- Alternate model accomodates long memory latency, doesn’t rely on caches as does Out-Of-Order, superscalar/VLIW designs
- Much easier for hardware: more powerful instructions, more predictable memory accesses, fewer harzards, fewer branches, fewer mispredicted branches, ...
- What % of computation is vectorizable?
- Is vector a good match to new apps such as multidemia, DSP?

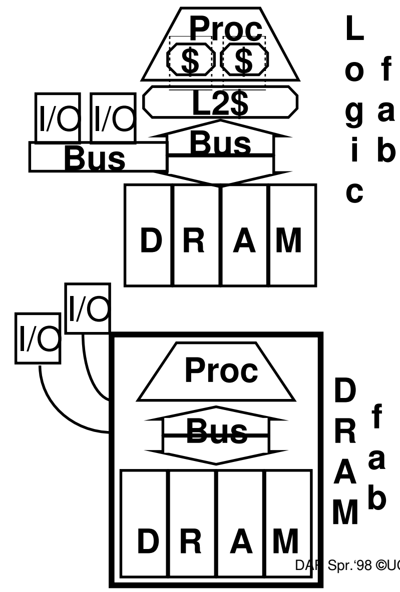
Project Overviews

- IRAM project related
- BRASS project related
- Industry suggested

IRAM Vision Statement

Microprocessor & DRAM on a single chip:

- on-chip memory latency 5-10X, bandwidth 50-100X
- improve energy efficiency 2X-4X (no off-chip bus)
- serial I/O 5-10X v. buses
- smaller board area/volume
- adjustable memory size/width



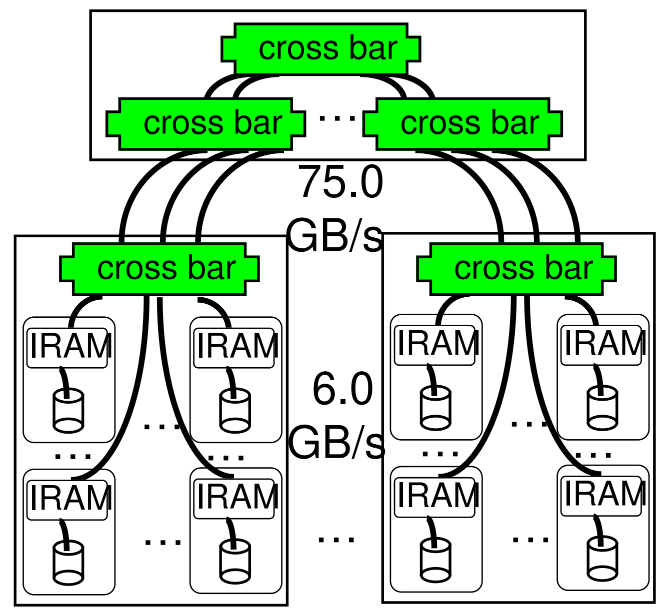
App #1: Intelligent PDA (2003?)

- Pilot PDA (todo,calendar, calculator, addresses,...)
- \+ Gameboy (Tetris, ...)
- \+ Nikon Coolpix (camera)
- \+ Cell Phone, Pager, GPS, tape recorder, TV remote, am/fm radio, garage door opener, ...
- \+ Wireless data (WWW)
- \+ Speech, vision recog.
- \+ Speech output for conversations

– Speech control of all devices

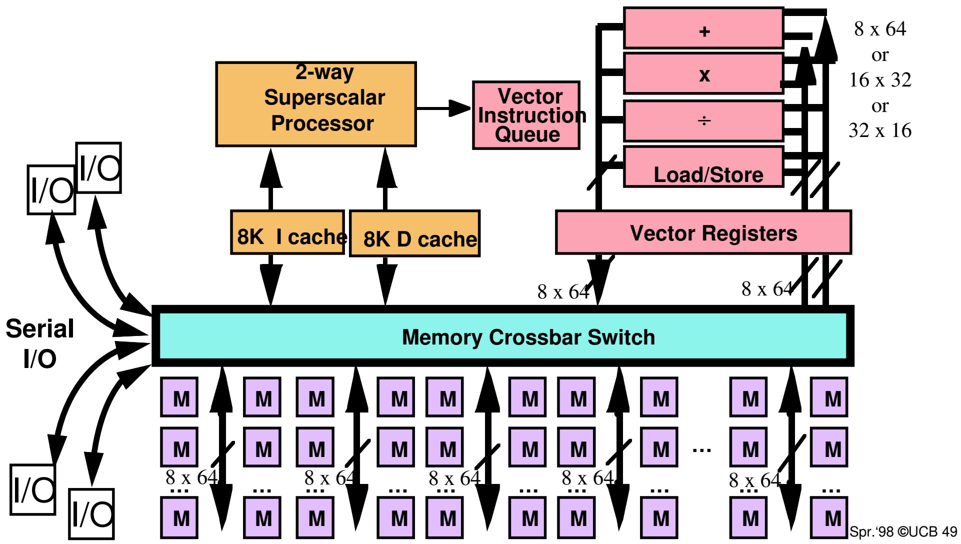
– Vision to see surroundings, scan documents, read bar codes, measure room

App #2: "Intelligent Disk"(IDISK): Scaleable Decision Support?

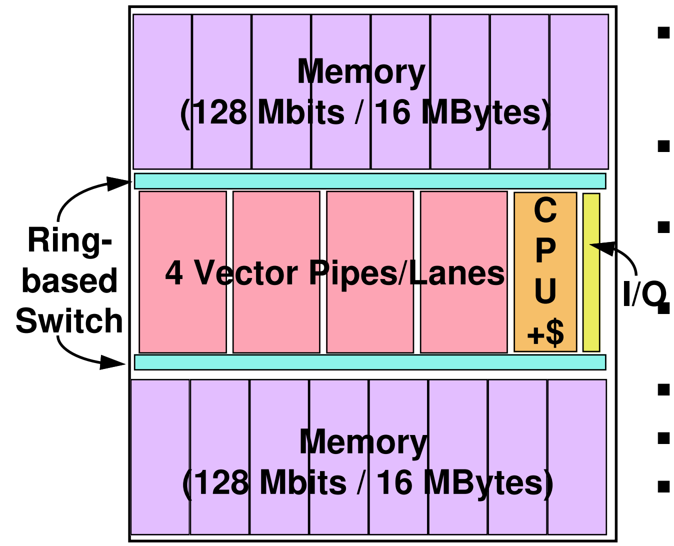


- 1 IRAM/disk + xbar + fast serial link v. conventional SMP
- Move function to data v. data to CPU (scan, sort, join,...)
- Network latency = f(SW overhead), not link distance
- Avoid I/O bus bottleneck of SMP
- Cheaper, faster, more scalable (≈1/3 $, 3X perf)

V-IRAM-2: 0.13 µm, Fast Logic, 1GHz 16 GFLOPS(64b)/64 GOPS(16b)/128MB



Tentative VIRAM-1 Floorplan



- 0.18 µm DRAM 32 MB in 16 banks x 256b, 128 subbanks
- 0.25 µm, 5 Metal Logic
- ≈ 200 MHz MIPS, 16K I$, 16K D$
- ≈ 4 200 MHz FP/int. vector units
- die: ≈ 16x16 mm
- xtors: ≈ 270M
- power: ≈2 Watts

Potential IRAM CS252 Projects?

- P1: Investigating algorithms, circuits, and floorplan for the VIRAM-1 vector unit.
  - Survey existing FPU implementations, then guess
  - Concentrate on multiplier; survey existing designs
  - Design of datapath with someone s low-level components
  - Take T0 fixed-point vector unit layout & apply changes
- P2: Algorithms/Benchmarks on VIRAM
  - Port the NESL Language to VIRAM-1
    - » Guy Blelloch's NESL language works well on vector machines, and he has several programs with irregular parallelism
  - Port the BDTI DSP Benchmarks to VIRAM-1
    - » VIRAM has DSP support; BDTI in Berkeley; vector better for DSP?
  - Code other algorithms: viterbi, lossless compression algorithms, MPEG2 encoding, GSM cellphone algorithms, encryption algorithms, texture mapping for 3D games.
- P3: TLB design for the VIRAM-1 vector unit.
  - 4 address translations per cycle? MicroTLB/lane?

Brass Vision Statement

- The emergence of high capacity reconfigurable devices is igniting a revolution in general-purpose processing. It is now becoming possible to tailor and dedicate functional units and interconnect to take advantage of application dependent dataflow. Early research in this area of reconfigurable computing has shown encouraging results in a number of spot areas including cryptography, signal processing, and searching --- achieving 10-100x computational density and reduced latency over more conventional processor solutions.
- BRASS: Microprocessor & FPGA on single chip:
  - use some of millions of transitors to customize HW dynamically to application

Potential BRASS CS252 Projects?

- P5: Explore energy implications of reconfigurable implementation of compute kernels.
  - For some common kernels, collect the data activity and estimate the actual energy consumed on a processor and an FPGA implementation.
  - The goal would be to understand the source of potential benefits for the reconfigurable architecture and quantify typical effects.
- Suggested by André DeHon(amd@CS.Berkeley.edu)

Other Projects: Database Study

- P4: Characterize Architecture Metrics for Multiple Commercial Databases
  - About 40% of sales of servers are for data base applications, yet little has been published on comparing multiple databases on a single SMP.
  - Use the builtin hardware performance tools of either the the 4-way SPARC SMP or the 4-way Intel Pentium II SMP to recreate the Kim Keeton's study across several commercial databases.
  - Does architectural support vary by database?
- Kim Keeton (kkeeton@cs) would be willing to help

Petabyte Backup?

- P6: Very Large Scale Backup
  - Automatic, reliable backup for large scale storage systems should be done at the device rather than file level.
  - Designed such that one never has to do a full backup. Only incrementals should be necessary.
  - Backup should be "consistent" and online.
  - Users shouldn't have to wait for the entire restore (≈Petabyte) to finish, just most frequently used (≈Terabyte).
  - Edward K. Lee (eklee@pa.dec.com) of DEC SRC Research labs would be willing to give advice

Other Projects: Mashey/SGI

- John R. Mashey (mash@mash.engr.sgi.com)
- P7. Doing better than SPEC
  - Continue to propose/analyze new benchmarks
  - No new data, more correlation (product lines, cache sizes)
  - Synthetic benchmarks (predict spec, measure latency)
- P8. Fixing Knuth Volume 3 & "Algorithms and Data Structures" courses
  - Classical algorithm analysis meets modern machines
  - Algorithm 1 meets algorithm 2, in presence of cache
  - Evil pointers, linked lists, out-of-order machines; convert lists, binary trees to N-ary structures and measure impact
  - Algorithms, languages, object-oriented stuff; measure old and new style; suggest indirect jump fast

Other Projects: Mashey/SGI

- P9. I/O nightmares on the way
  - Disks are getting HUGE, fast, and algorithms are breaking (out) all over
  - if you believe TeraStor, expect to see 500GB 3.5" by 2003
  - unfortunately, reading a 500GB disk take 10,000 secs!
  - coming disks are likely to cause trouble for most current filesystems, and there'd be some serious rearchitecting.
  - file system evaluations/analysis at user level, and study key algorithms without having to invent everything.
  - John R. Mashey (mash@mash.engr.sgi.com) advises again

New I/O standard

- P13. Evaluating New WinTEL I/O Standard
  - "I2O" (I-to-O) is the big new I/O architecture definition from WiNTel, attempting to push more processing off of the main CPU's and onto the I/O cards.
  - Is this any faster than the "smart" IO subsystems people have been building for a while?
  - Will I2O open up new opportunities to move stuff off of the main CPU's, resulting in faster performance?
  - Establish the performance benefits of I2O subsystems compared to traditional "smart" IO subsystems and traditional "not smart" IO subsystems
  - Make suggestions of how to improve system performance further by perhaps off loading more IO processing into an IOP.
  - David Douglas (douglas@East.Sun.COM) and F. Balint (balintf@East.Sun.COM) from Sun Microsystems willing to advise

Other Projects

- P12. Networks vs. Busses
  - Measure Disk controller bandwidth and latency (via read/write of same cached block).
  - Measure Network Controller bandwidth and latency
  - Prediction: Network controller has lower latency and bandwidth. Why? (why can't we have the best of both?)
  - Jim Gray of Micrsoft (Gray@Microsoft.com) suggestion
- P14. Evaluating Embedded Processors
  - Run Spec95int and other micro-benchmarks (lmbench etc) on an embedded processor (recommended: StrongARM SA110 that we have available).
  - Use BDTI benchmarks or other DSP kernels to compare/ analyze the performance of an embedded processor (SA110) and a desktop processor (Sparc/Pentium).
  - Christoforos Kozyrakis (kozyraki@CS) will help

Other Projects

- Greg Pfister of IBM (pfister@us.ibm.com) suggestions
- P10. I/O
  - Many topics from last time could be revisited in an I/O context: Benchmarks of I/O, efficiency of I/O, etc.
  - How good is the memory system at block streaming multiple multimedia streams onto disk and/or a fast network?
  - How about OS overhead for lots of little transactions -- there certainly are imaginative ways it could be be reduced, and proposing/measuring the results could be a good project.
- P11. Application Performance: Messages vs. CC-NUMA
  - How about an application comparison of a low-overhead messaging scheme (like VIA) versus a shared-memory implementation using CC-NUMA?
  - Of course it should be based on measurement, so maybe SMP measurement plus some analytical modelling substitutes for CC-NUMA and some analytical work.