CS252
Graduate Computer Architecture
Lecture 9

Instruction Level Parallelism: Potential?
Vector Processing

September 29, 2000

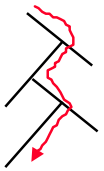
Prof. John Kubiatowicz

Review: Instruction Level Parallelism

- *Instruction level parallelism* (ILP)
  - potential of short instruction sequences to execute in parallel
  - Often measured by IPC (Instructions per cycle) instead of CPI (cycles per instruction)
- Superscalar and VLIW: CPI < 1 (IPC > 1)
  Dynamic vs Static Issue:
  - All about increasing issue and commit bandwidth: IPC limited by the rate of inflow and exit from pipeline
  - More instructions issue at same time => larger hazard penalty
  - Limitation is often number of instructions that you can successfully fetch and decode per cycle ⇒ "Flynn barrier"
- SW Pipelining
  - Symbolic Loop Unrolling to get most from pipeline with little code expansion, little overhead
- Branches, branches, branches: How to keep feeding *useful* instructions to the pipeline???
  - Since 1 in 5 instruction is a branch, must predict either in software or hardware

Review: Trace Scheduling

- Parallelism across IF branches vs. LOOP branches
- Two steps:
  - *Trace Selection*
    - » Find likely sequence of basic blocks (*trace*) of (statically predicted or profile predicted) long sequence of straight-line code
  - *Trace Compaction*
    - » Squeeze trace into few VLIW instructions
    - » Need bookkeeping code in case prediction is wrong



- This is a form of compiler-generated branch prediction!
  - Make "common-case" fast at expense of less common case
  - Compiler must generate "fixup" code to handle cases in which trace is not the taken branch
  - Needs extra registers: undoes bad guess by discarding

Limits to Multi-Issue Machines

- Inherent limitations of ILP
  - 1 branch in 5: How to keep a 5-way VLIW busy?
  - Latencies of units: many operations must be scheduled
  - Need approx Pipeline Depth x No. Functional Units of independent operations to keep all pipelines busy.
  - Difficulties in building HW
- Complexity:
  - Easy: More instruction bandwidth from L1 cache
  - Easy: More execution bandwidth
    - » Duplicate FUs to get parallel execution
  - Hard: Increase ports to Register File (bandwidth)
    - » VLIW example needs 7 read and 3 write for Int. Reg. & 5 read and 3 write for FP reg
  - Harder: Getting useful instructions to pipeline (branch prediction)
  - Harder: Increase ports to memory (bandwidth)
  - Harder: Latency to memory
  - Decoding Superscalar and impact on clock rate, pipeline depth?

Limits to ILP: Limit Studies

- Conflicting studies of amount: 2? 1000?
  - Benchmarks (vectorized Fortran FP vs. integer C programs)
  - Hardware sophistication
  - Compiler sophistication
- How much ILP is available using existing mechanisms with increasing HW budgets?
- Do we need to invent new HW/SW mechanisms to keep on processor performance curve?
  - Intel MMX
  - Motorola AltaVec
  - Supersparc Multimedia ops, etc.
  - Reinvent vector processing (IRAM)
  - Something else? Neural nets? Reconfigurable logic?

Limits to ILP: Specifications for a "perfect" machine

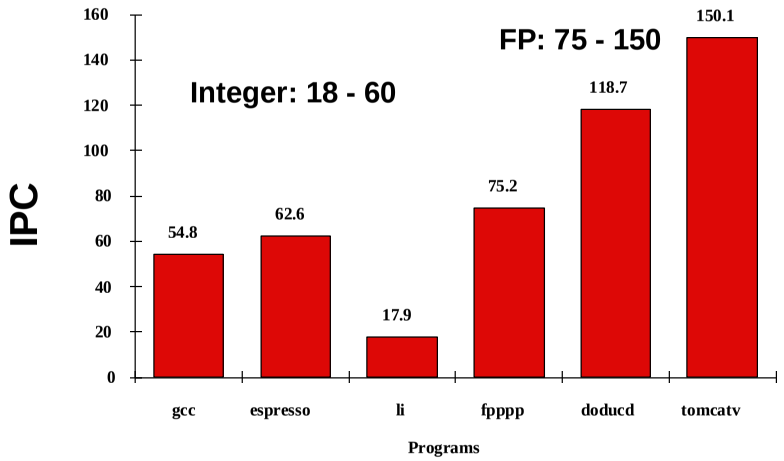
Assumptions for ideal/perfect machine to start:

1. *Branch prediction*–perfect; no mispredictions
2. *Register renaming*–infinite virtual registers and all WAW & WAR hazards are avoided
3. *Memory-address alias analysis* – addresses are known in advance & a store can be moved before a load provided addresses not equal
4. *Window Size* – machine with perfect speculation & an unbounded buffer of instructions available

1 cycle latency for all instructions; MIPS compilers; unlimited number of instructions issued per cycle

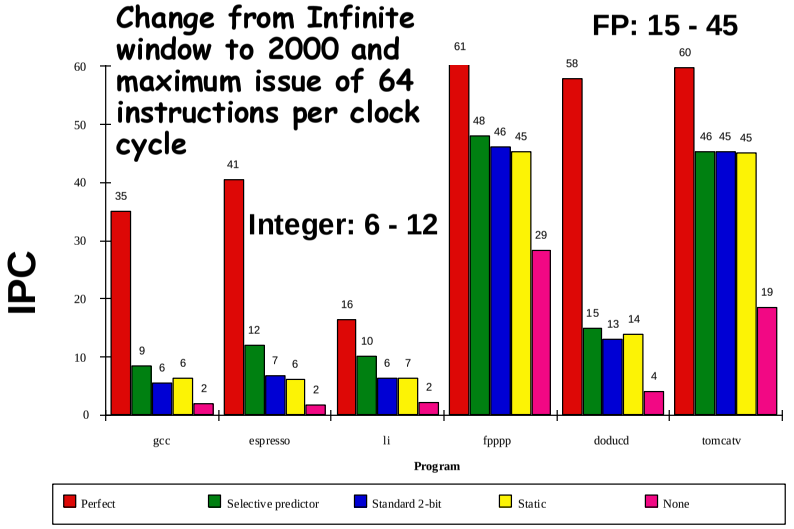
Upper Limit to ILP: Ideal Machine

(Figure 4.38, page 319)



More Realistic HW: Branch Impact

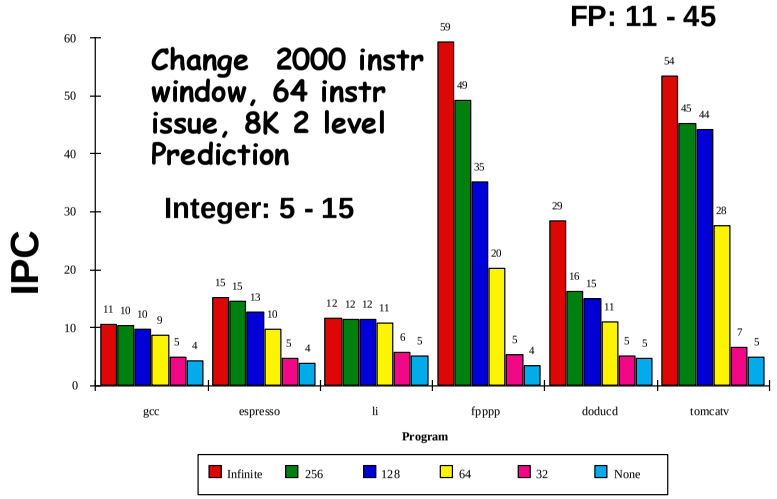
Figure 4.40, Page 323



Perfect | Pick Cor. or BHT | BHT (512) | Profile | No prediction

More Realistic HW: Register Impact

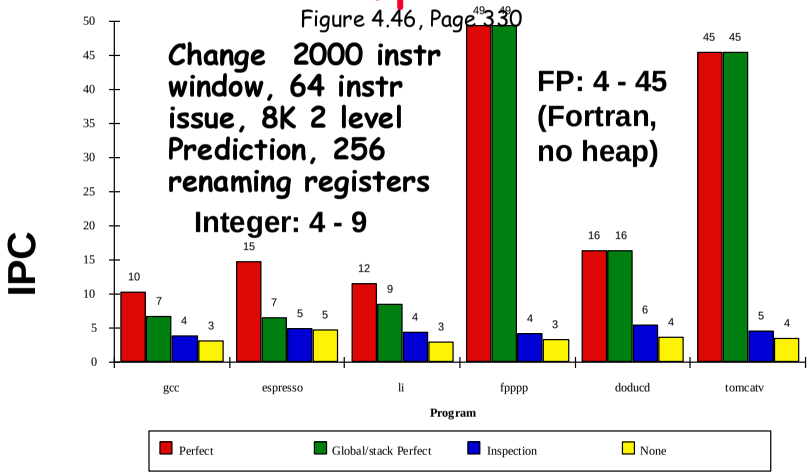
Figure 4.44, Page 328



Infinite 256 128 64 32 None

More Realistic HW: Alias Impact

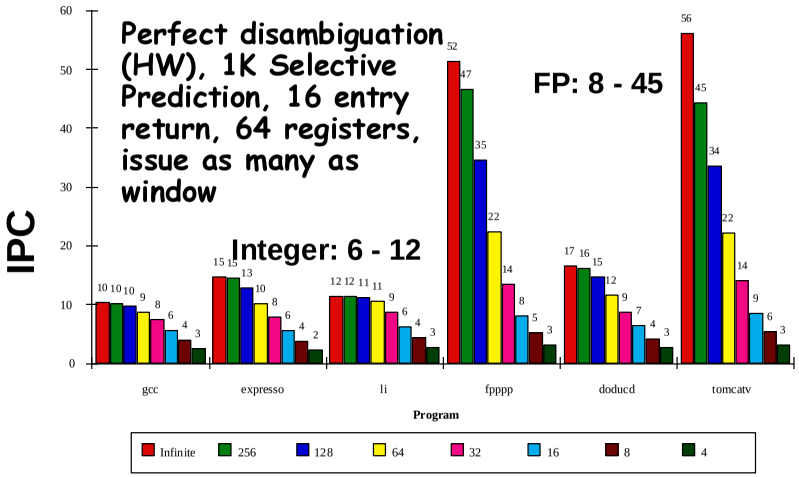
Figure 4.46, Page 330



Perfect Global/Stack perf; heap conflicts Inspec. Assem. None

Realistic HW for '9X: Window Impact

(Figure 4.48, Page 332)

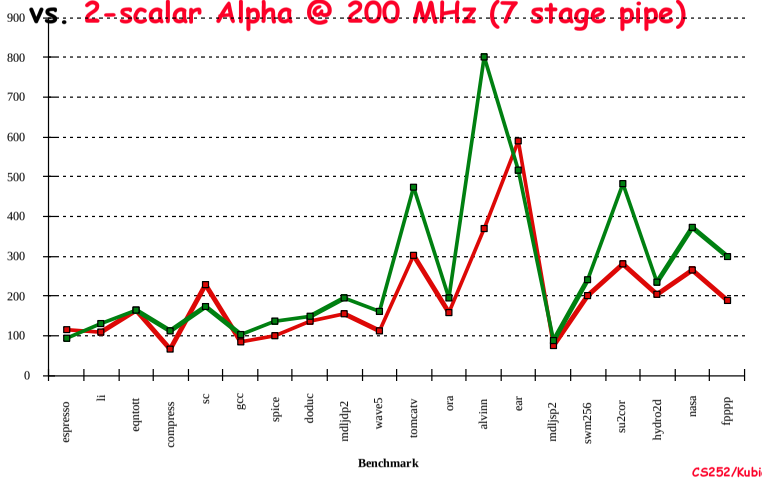


Infinite 256 128 64 32 16 8 4

Braniac vs. Speed Demon(1993)

- 8-scalar IBM Power-2 @ 71.5 MHz (5 stage pipe)

vs. 2-scalar Alpha @ 200 MHz (7 stage pipe)



Problems with scalar approach to ILP extraction

- Limits to conventional exploitation of ILP:
  - *pipelined clock rate*: at some point, each increase in clock rate has corresponding CPI increase (branches, other hazards)
  - *branch prediction*: branches get in the way of wide issue. They are too unpredictable.
  - *instruction fetch and decode*: at some point, its hard to fetch and decode more instructions per clock cycle
  - *register renaming*: Rename logic gets really complicate for many instructions
  - *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

Cost-performance of simple vs. OOO

MIPS MPUs	R5000	R10000	10k/5k
• Clock Rate	200 MHz	195 MHz	1.0x
• On-Chip Caches	32K/32K	32K/32K	1.0x
• Instructions/Cycle	1(+ FP)	4	4.0x
• Pipe stages	5	5-7	1.2x
• Model	In-order	Out-of-order	---
• Die Size (mm^2)	84	298	3.5x
– without cache, TLB	32	205	6.3x
• Development (man yr.)	60	300	5.0x
• SPECint_base95	5.7	8.8	1.6x

CS 252 Administrivia

- Exam: Wednesday 10/18
  Location: TBA
  TIME: 5:30 - 8:30
- This info is on the Lecture page (has been)
- Meet at LaVal's afterwards for Pizza and Beverages
- Assignment up now
  - Due in two weeks
  - Done in pairs. Put both names on papers.
  - Make sure you have partners! Feel free to use mailing list for this.
- Computers in the news? Sony playstation hard to manufacture! Expected to be a serious shortage.

Architecture in practice

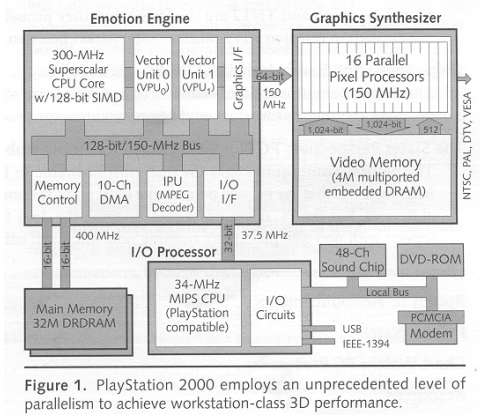


Figure 1. PlayStation 2000 employs an unprecedented level of parallelism to achieve workstation-class 3D performance.

Figure 2. PlayStation 2000 screenshot. (Source: Namco)

- (as reported in Microprocessor Report, Vol 13, No. 5)
  - Emotion Engine: 6.2 GFLOPS, 75 million polygons per second
  - Graphics Synthesizer: 2.4 Billion pixels per second
  - Claim: *Toy Story* realism brought to games!

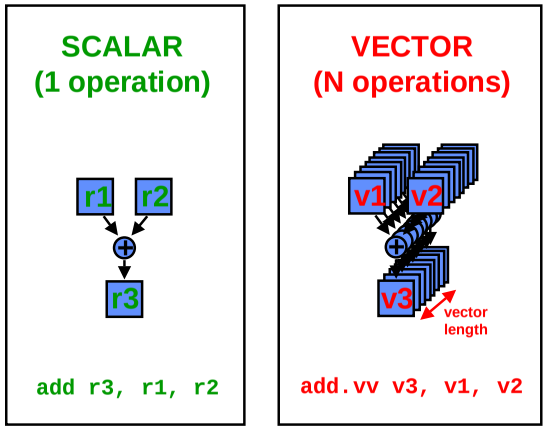
Complexity of Superscalar Processors

- In class discussion of "Complexity effective superscalar processors"

  Subbaro Palacharla, Norman Jouppi, and Jim Smith

Alternative Model: Vector Processing

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



"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

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

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

DAXPY (Y = a * X + Y)

Assuming vectors X, Y are length 64

Scalar vs. Vector

```
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
```

```
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
```

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

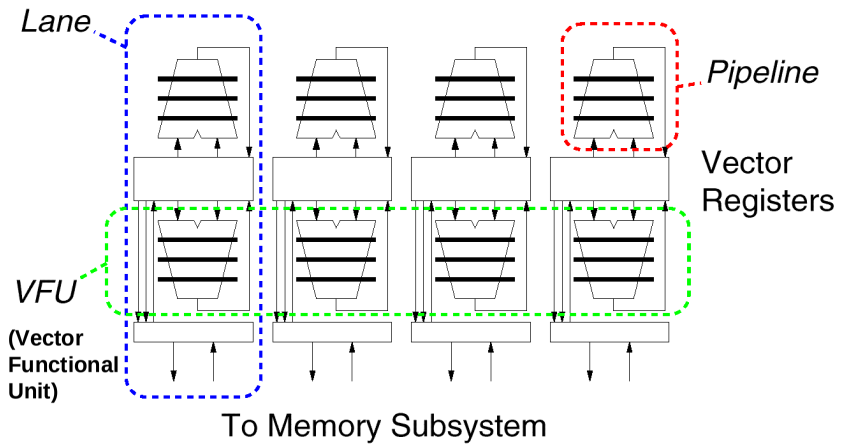
Example Vector Machines

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 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 = "lanes" (sometimes called "pipelines" or "pipes")

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
- *m convoys take m chimes*; 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 convoys, 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

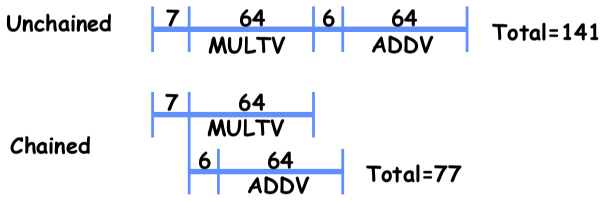
Operation	Start-up penalty (from CRAY-1)
Vector load/store	12
Vector multiply	7
Vector add	6

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

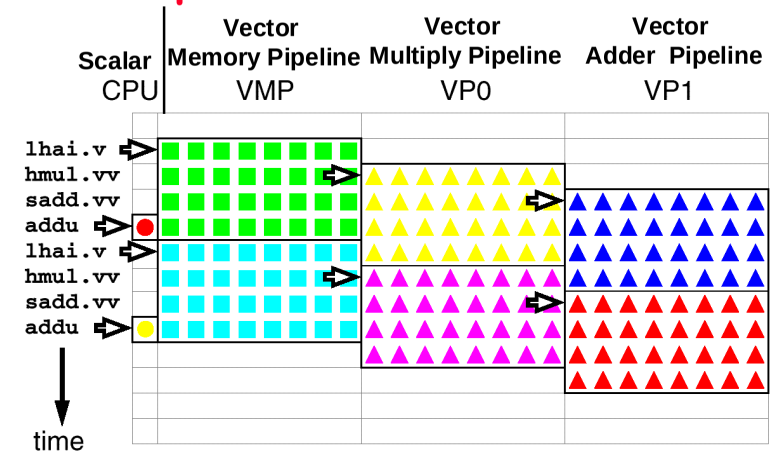
Convoy	*Start*	*1st result*	*last result*	
1. LV	0	12	11+n (=12+n-1)	
2. MULV, LV	12+n	12+n+7	18+2n	*Multiply startup*
	12+n+1	12+n+13	24+2n	*Load start-up*
3. ADDV	25+2n	25+2n+6	30+3n	*Wait convoy 2*
4. SV	31+3n	31+3n+12	42+4n	*Wait convoy 3*

Vector Opt #1: Chaining

- Suppose:
  MULV V1,V2,V3
  ADDV V4,V1,V5 ; separate convoy?
- *chaining*: vector register (V1) is not as a single entity but as a group of individual registers, then pipeline forwarding can work on individual elements of a vector
- *Flexible chaining*: allow vector to chain to any other active vector operation => more read/write ports
- As long as enough HW, increases convoy size



Example Execution of Vector Code



8 lanes, vector length 32, chaining

Operations
Instruction issue

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

Minimum resources for Unit Stride

- Start-up overheads usually longer for LSUs
- Memory system must sustain (# lanes x word) /clock
- 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
- 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 *l*+*m* … 2 *l*

    word: -- … 0 1 2 … *m*-1 -- … *m*
  - may have 1024 banks in SRAM

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 unit stride) => LVWS (load vector with stride) instruction
- Strides => can cause bank conflicts (e.g., stride = 32 and 16 banks)
  - Can use prime number of banks! (Paper for next time)
- Think of address per vector element

Vector Opt #2: 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 data from each address in the index vector
  - This produces a "dense" vector in the vector registers
- 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 figured out by compiler since can't know elements distinct, no dependencies
- 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)

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 Ś 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
```


Vector Opt #3: Conditional Execution

- Suppose:

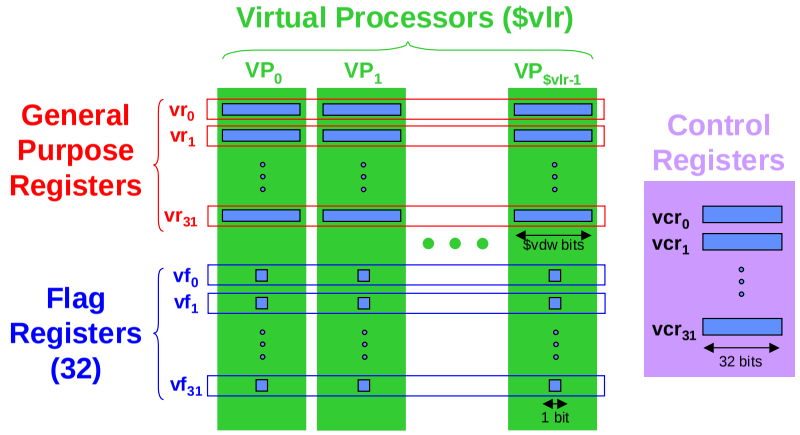
```
  do 100 i = 1, 64
      if (A(i) .ne. 0) then
         A(i) = A(i) – B(i)
      endif
100 continue
```

- *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?

Virtual Processor Vector Model: Treat like SIMD multiprocessor

- Vector operations are SIMD (single instruction multiple data) operations
  - Each virtual processor has as many scalar "registers" as there are vector registers
  - There are as many virtual processors as current vector length.
  - Each element is computed by a virtual processor (VP)

Vector Architectural State



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 Processing and Power

- If code is vectorizable, then simple hardware, more energy efficient than Out-of-order machines.
- Can decrease power by lowering frequency so that voltage can be lowered, then duplicating hardware to make up for slower clock:

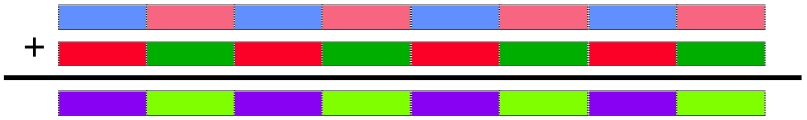
$$\text{Power} \propto CV^2 f$$

$$\left\{ \begin{array}{c} f = \frac{1}{n} f_0 \\ Lanes = n \times Lanes_0 \\ V = \delta V_0;\ \delta < 1 \end{array} \right\} \Rightarrow \left\{ \begin{array}{l} \text{Performance Constant} \\ \text{Power Change}: \delta^2 < 1 \end{array} \right\}$$

- Note that V_o can be made as small as permissible within process constraints by simply increasing "n"

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

Mediaprocessing: 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`)

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 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 (ETA) 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 by adding HW resources)
- Compact: Describe N operations with 1 short instruction
- Predictable: performance vs. statistical performance (cache)
- Multimedia ready: N * 64b, 2N * 32b, 4N * 16b, 8N * 8b
- Mature, developed compiler technology
- Vector Disadvantage: Out of Fashion?
  - Hard to say. Many irregular loop structures seem to still be hard to vectorize automatically.
  - Theory of some researchers that SIMD model has great potential.

Summary #1: Vector Processing

- Vector Processing represents an alternative to complicated superscalar processors.
- Primitive operations on large vectors of data
- Load/store architecture:
  - Data loaded into vector registers; computation is register to register.
  - Memory system can take advantage of predictable access patterns:
    - » Unit stride, Non-unit stride, indexed
- Vector processors exploit large amounts of parallelism without data and control hazards:
  - Every element is handled independently and possibly in parallel
  - Same effect as scalar loop without the control hazards or complexity of tomasulo-style hardware
- Hardware parallelism can be varied across a wide range by changing number of *vector lanes* in each vector functional unit.

Summary #2: ILP? Wherefore art thou?

- There is a fair amount of ILP available, but branches get in the way
  - Better branch prediction techniques? Probably not much room to go still: prediction rates already up in the 93% and above
  - Fundamental new programming model?
- Vector model accommodates long memory latency, doesn't rely on caches as does Out-Of-Order, superscalar/VLIW designs:
  - No branch prediction! Loops are implicit in model
  - Much easier for hardware: more powerful instructions, more predictable memory accesses, fewer hazards, fewer branches, fewer mispredicted branches, ...
  - But, what % of computation is vectorizable?
  - Is vector a good match to new apps such as multimedia, DSP?
- Right answer? Both? Neither? (my favorite)
- Next time: Prediction of everything but stock market