Advanced Computer Architecture I Week 8

ECE 587/687
Fall Quarter 2005

Dan Hammerstrom

PORTLAND STATE UNIVERSITY

Monday November 21:

PORTLAND STATE UNIVERSITY

- There will be a short quiz on the Week 8 readings and lectures and Week 9 readings
- Part 4 of the project is due
  - Part 5 is for extra credit only
  - There will be no additional extensions, hand in what you have, working or not
  - Each team will give a short 5-10 minute presentation on their project 1 implementation
- Project 2 (and the final project) will be assigned
- There will be no class on Wednesday Nov. 23 (and no make-up lecture this time)

PORTLAND STATE UNIVERSITY

Week 8 (11/14, 11/16)

- Topics:
  - Vector processors
- Required Reading:
  - Vector Processors, Krste Asanovc (from H&P Appendix G) hnp_appendix_g.pdf
  - "Vector Vs. Superscalar and VLIW Architectures for Embedded Multimedia Benchmarks," Kozyrakis & Patterson, vector_vs_scalar.pdf
  - The Cell Processor, cell_v2.pdf
  - "Cell Technology for Graphics and Visualization," Bruce D'Amora, cell1.pdf
  - "Broadband Engine (Cell Processor) power-efficient and cost-effective high-performance processing for a wide range of applications," J. J. Porta, cell2.pdf

PORTLAND STATE UNIVERSITY

Reference Only:

- "Overcoming the Limitations of Conventional Vector Processors", Kozyrakis & Patterson, vector_limits.pdf
- "VIRAM1: A Media-Oriented Vector Processor with Embedded DRAM", Gebis et al. viram_ucb.pdf

Vector Processing

PORTLAND STATE UNIVERSITY

Review: Instruction Level Parallelism

PORTLAND STATE UNIVERSITY

- 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 multi-ported, multiple levels of caching

Problems with Conventional Approach

PORTLAND STATE UNIVERSITY

- Limits to conventional exploitation of ILP:

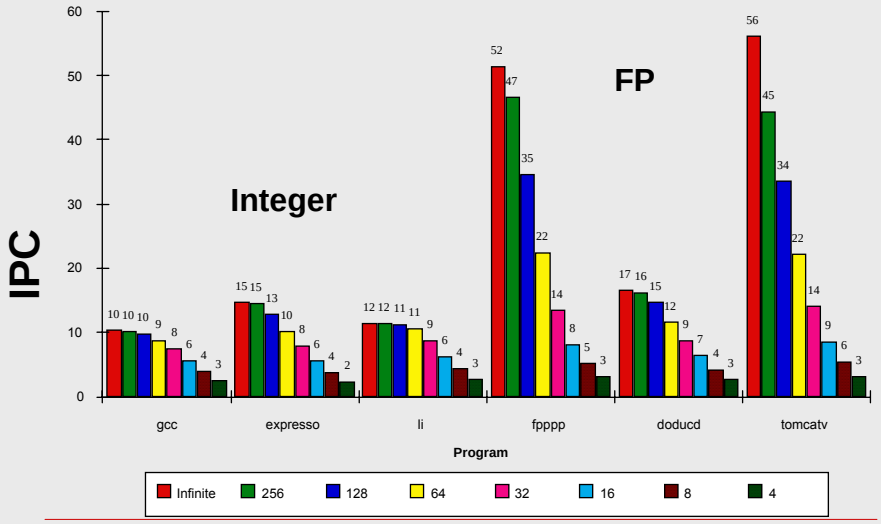
  1) *pipelined clock rate*: at some point, each increase in clock rate has a potential corresponding IPC decrease (branches, other hazards)

  2) *instruction fetch and decode*: it becomes increasingly expensive (non-linearly) 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

Review: Theoretical Limits to ILP?

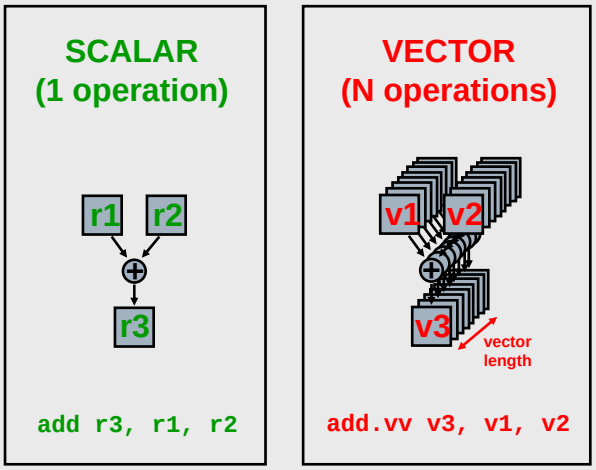
PORTLAND STATE UNIVERSITY



An Alternative Model: Vector Processing

PORTLAND STATE UNIVERSITY

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



Properties of Vector Processors

PORTLAND STATE UNIVERSITY

- Each result is independent of the previous result
  => long pipeline, compiler ensures no dependencies
  => high clock rate
- Vector instructions access memory with known patterns
  => highly interleaved memory
  => amortize memory latency over 64+ elements
  => no (data) caches (they usually have an instruction cache)
- Reduces branches and branch problems in pipelines
- Each single vector instruction can perform lots of work (the inner loop)
  => fewer instruction fetches
- The programmer (or compiler) explicitly specifies the ILP

Operation & Instruction Count: RISC v. Vector Processor

(from F. Quintana, U. Barcelona.)

PORTLAND STATE UNIVERSITY

Spec92fp Program	Operations (Millions) RISC	Vector	R/V	Instructions (M) 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

PORTLAND STATE UNIVERSITY

- *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
  - Here we assume vector-registers

Components of a Vector Processor

PORTLAND STATE UNIVERSITY

- *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, can start a 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
  - Can have optimized memory subsystem
- *Scalar registers*: single element for FP scalar or address
- High performance cross-bar to connect FUs , LSUs, registers

Typical Vector Instructions

PORTLAND STATE UNIVERSITY

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
LVV1	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

PORTLAND STATE UNIVERSITY

- 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
    - May have some performance penalty

DAXPY (Y = a * X + Y)

PORTLAND STATE UNIVERSITY

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

Vector Surprise

PORTLAND STATE UNIVERSITY

- 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

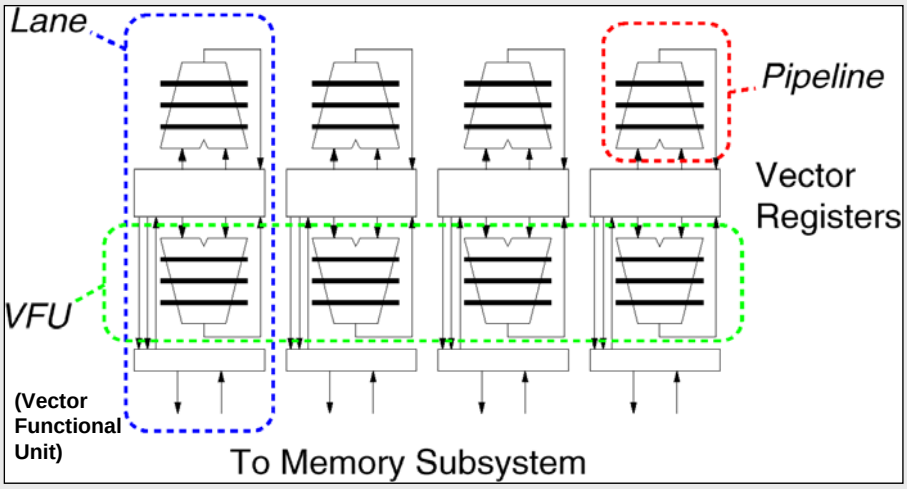
Vector Implementation

PORTLAND STATE UNIVERSITY

- Vector register file
  - Each register is an array of elements
  - Size of each register determines maximum vector length
  - Vector length register determines vector length (less than or equal to the number of elements in the vector register) for a particular operation
- Multiple parallel execution units = "lanes" (sometimes called "pipelines" or "pipes")

Vector Terminology: 4 lanes, 2 vector functional units

PORTLAND STATE UNIVERSITY



Vector Execution Time

PORTLAND STATE UNIVERSITY

- Time = func(vector length, data dependencies, 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 approximation 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
```

Can go in same convoy since they don't depend on each other

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

PORTLAND STATE UNIVERSITY

- *Start-up time*: pipeline latency time (depth of FU pipeline); another sources of vector 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
2. MULV, LV	12+n	12+n+12	23+2n
3. ADDV	24+2n	24+2n+6	29+3n
4. SV	30+3n	30+3n+12	41+4n

Why startup time?

PORTLAND STATE UNIVERSITY

- 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

PORTLAND STATE UNIVERSITY

- Start-up overhead usually longer for Loads and Stores
- Memory system must sustain (# lanes x word) /clock cycle
- Many Vector Processors use banks (vs. simple interleaving):
  1) support multiple loads/stores per cycle
     => multiple banks & address banks independently
  2) support for non-sequential accesses
- Note: No. memory banks > memory latency to avoid stalls
  - *m* banks => *m* words of memory latency
  - If *m* < *n (vector length)*, then gap in memory "pipeline":

clock:	0	...	*n*	*n*+1	*n*+2	...	*n*+*m*-1	n+m	...	2*n*
word:	--	...	0	1	2	...	*m*-1	--	...	*m*

  - A large vector processor may have 1024 banks or more in SRAM

Vector Length

PORTLAND STATE UNIVERSITY

- What do we do when the 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! And what if *n* > Max. Vector Length (MVL)?

Strip Mining

PORTLAND STATE UNIVERSITY

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

PORTLAND STATE UNIVERSITY

- R_∞: MFLOPS rate on an infinite-length vector
  - (R_n is the MFLOPS rate for a vector of length n)
  - Vector "speed of light"
  - Real problems do not have unlimited vector lengths, and the start-up penalties encountered in real problems will be larger
- $N_{1/2}$: The vector length needed to reach one-half of R_∞
  - A good measure of the impact of start-up cycles
- 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

PORTLAND STATE UNIVERSITY

- Suppose adjacent elements are not stored sequentially in memory (for example, column access)

```
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)

Vector Optimization #1: Chaining

PORTLAND STATE UNIVERSITY

- 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 register read/write ports
- As long as there is enough HW, this increases the convoy size

Vector Optimization #2: Conditional Execution

PORTLAND STATE UNIVERSITY

- 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 the *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 a clock even if the result is not stored

Vector Optimization #3: Sparse Matrices

PORTLAND STATE UNIVERSITY

- 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 => a nonsparse vector in a vector register

- 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 K_i elements distinct, no dependencies; by compiler directive

- Use **CVI** to create index 0, 1xm, 2xm, ..., 63xm

Sparse Matrix Example

PORTLAND STATE UNIVERSITY

- 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	17 MFLOPS	125 (7.3)

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

Vector Example with Dependency

PORTLAND STATE UNIVERSITY

```
/* Matrix Multiply a[m][k] * b[k][n] to get c[m][n] */

for (i=1; i<m; i++)
{
    for (j=1; j<n; j++)
    {
        sum = 0;
        for (t=1; t<k; t++)
        {
            sum += a[i][t] * b[t][j];
        }
        c[i][j] = sum;
    }
}
```

- element at a time from a vector register and putting it in the scalar unit
- Called a "reduction" (we saw reduction in MPI)

Novel Matrix Multiply Solution

PORTLAND STATE UNIVERSITY

- You don't need to do reductions for matrix multiply
- You can calculate multiple independent sums within one vector register
- Vectorize the j loop to perform 32 dot-products at the same time

Original Vector Example with dependency

PORTLAND STATE UNIVERSITY

```
/* Multiply a[m][k] * b[k][n] to get c[m][n] */

for (i=1; i<m; i++)
{
    for (j=1; j<n; j++)
    {
        sum = 0;
        for (t=1; t<k; t++)
        {
            sum += a[i][t] * b[t][j];
        }
        c[i][j] = sum;
    }
}
```

Optimized Version

PORTLAND STATE UNIVERSITY

```
/* Multiply a[m][k] * b[k][n] to get c[m][n] */

for (i=1; i<m; i++)
   {
   for (j=1; j<n; j+=32)/* Step j 32 at a time */
      {
      sum[0:31] = 0; /* Initialize a vreg to zeros */
      for (t=1; t<k; t++)
         {
         a_scalar = a[i][t]; /* Get scalar from matrix */
         b_vector[0:31] = b[t][j:j+31];
         /* Get vector from b matrix */
         prod[0:31] = b_vector[0:31]*a_scalar;
         /* Do a vector-scalar multiply */
         /* Vector-vector accumulate */
         sum[0:31] += prod[0:31];
         }
      /* Unit-stride store of vector of results */
      c[i][j:j+31] = sum[0:31];
      }
   }
```


PORTLAND STATE UNIVERSITY

- Ignore dividing it into 64-element blocks and different size dimensions, the basic algorithm is
  - Row b_{1-} is fetched and placed into a vector register
  - Scalar element a_{11} multiplies vector b_{1-} and the result is added to the vector c_{1-} in another vector register, "accumulating the sum"
  - Then fetch row b_{2-} and use a_{12} to do it again …

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} x \begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \\ b_{41} & b_{42} & b_{43} & b_{44} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & c_{13} & c_{14} \\ c_{21} & c_{22} & c_{23} & c_{24} \\ c_{31} & c_{32} & c_{33} & c_{34} \\ c_{41} & c_{42} & c_{43} & c_{44} \end{bmatrix}$$

Matrix Multiply is $O(n^3)$, if you can do $O(n)$ ops at a time, then MM becomes $O(n^2)$

Vector Length?

PORTLAND STATE UNIVERSITY

- Vector length => Keep all VFUs busy:
- Being longer is good because:
  1) It spreads out the vector startup costs
  2) There is a lower instruction bandwidth
  3) Tiled access to memory reduces scalar processor memory bandwidth needs
  4) If you know that the max length of app. is < max vector length, then there is no strip mining overhead
  5) Better spatial locality for memory access
- But being longer may not help much because:
  1) Diminishing returns on overhead savings as you keep doubling number of elements
  2) Need natural app. vector length to match physical register length

Number of Vector Registers?

PORTLAND STATE UNIVERSITY

- More Vector Registers:

  1) Reduces vector register “spills” (save/restore)

  2) aggressive scheduling of vector instructions: better compiling to take advantage of ILP

  Fewer bits in instruction format (usually 3 fields)

Context switch overhead

PORTLAND STATE UNIVERSITY

- Extra dirty bit per processor
  - If vector registers not written, don't need to save on context switch
- Extra valid bit per vector register, cleared on process start
  - Don't need to restore on context switch until needed

Exception handling

PORTLAND STATE UNIVERSITY

- If there is an external exception, we can just put a pseudo-op into pipeline and wait for all vector ops to complete
  - Alternatively, we can wait for the scalar unit to complete and begin working on exception code assuming that vector unit will not cause an exception and exception handling code does not use the vector unit
- Arithmetic traps harder
- Precise interrupts => large performance loss
- Alternative model: arithmetic exceptions set vector flag registers, 1 flag bit per element
- Software inserts trap barrier instructions from SW to check the flag bits as needed
- IEEE Floating Point requires 5 flag bits

Exception handling: Page Faults

PORTLAND STATE UNIVERSITY

- Page Faults must be precise
- Instruction Page Faults are generally not a problem
- Data Page Faults have a greater impact
- Option 1: Save/restore internal vector unit state
  - Freeze pipeline, dump vector state
  - perform needed ops
  - Restore state and continue vector pipeline
- Option 2: expand memory pipeline to check addresses before send to memory + memory buffer between address check and registers

Parallelism From Vectors Is Less Expensive Than Implicit ILP

PORTLAND STATE UNIVERSITY

Scalar

- N ops per cycle
  $\Rightarrow O(N^2)$ circuitry
- HP PA-8000
  - 4-way issue
  - reorder buffer: 850K transistors
    - incl. 6,720 5-bit register number comparators

Vector

- N ops per cycle
  $\Rightarrow O(N + \varepsilon N^2)$ circuitry
- T0 vector micro
  - 24 ops per cycle
  - 730K transistors total
    - only 23 5-bit register number comparators

Vectors Are More Power Efficient

PORTLAND STATE UNIVERSITY

Single-issue Scalar

- One instruction fetch, decode, dispatch per operation
- Arbitrary register accesses, adds area and power
- Loop unrolling and software pipelining for high performance increases instruction cache footprint
- All data pass through cache; wastes power if no temporal locality
- One TLB lookup per load or store
- Off-chip access in whole cache lines

Vector

- One instruction fetch, decode, dispatch per vector
- Structured register accesses
- Smaller code for high performance, less power in instruction cache misses
- Bypass cache
- One TLB lookup per group of loads or stores
- Move only necessary data across chip boundary

Superscalar Energy Efficiency Even Worse!

PORTLAND STATE UNIVERSITY

Superscalar

- Control logic grows exponentially with issue width
- Control logic consumes energy regardless of available parallelism
- Speculation to increase visible parallelism wastes energy

Vector

- Control logic grows linearly with issue width
- Register / Function unit crossbar can grow non-linearly, but is small part of total
- Vector unit switches off when not in use
- Vector instructions expose parallelism without speculation
- Software control of speculation when desired:
  - Whether to use vector mask or compress/expand for conditionals

Applications

PORTLAND STATE UNIVERSITY

Are Vectors forever 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
- Remember the Intel Tera-Era paper!

Typical Vector Pitfalls

PORTLAND STATE UNIVERSITY

- 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 (CDC Star)
- Pitfall: Good processor vector performance without providing good memory bandwidth

Vector Advantages

PORTLAND STATE UNIVERSITY

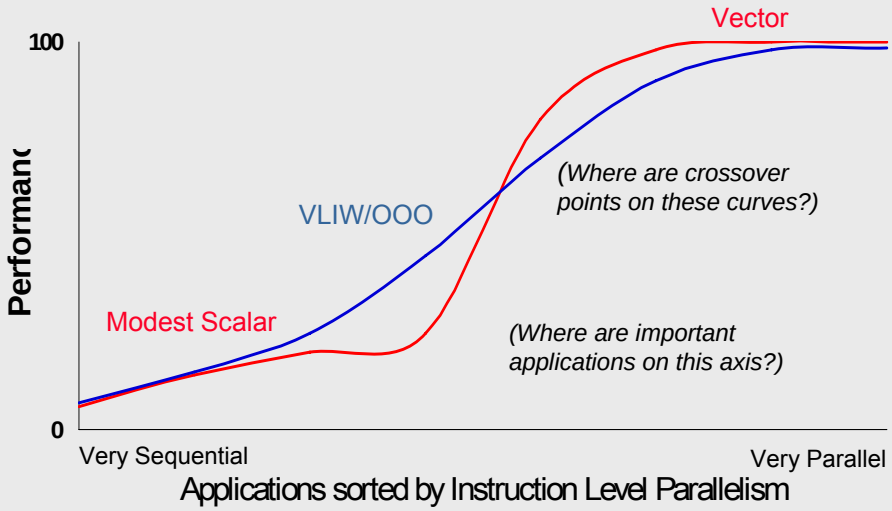
- Easy to get high performance; N operations:
  - are independent
  - use same functional unit
  - access disjoint registers
  - access registers in same order as previous instructions
  - access contiguous memory words in a known pattern
  - can exploit large memory bandwidth
  - can hide memory latency (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)

PORTLAND STATE UNIVERSITY

- Multimedia ready: choose N * 64b, 2N * 32b, 4N * 16b, 8N * 8b
- Mature, developed compiler technology
- Not all that difficult to think in terms of vectors and matrices
  - Much easier than more general MIMD (MPI) distributed processing
- Vector Disadvantages:
  - Not all applications vectorizable, though the applications that need to be sped up often are
  - Out of Fashion, outré

VLIW/Out-of-Order vs. Modest Scalar+Vector

PORTLAND STATE UNIVERSITY



Cost-performance of Simple vs. OOO

PORTLAND STATE UNIVERSITY

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

Vector Summary

PORTLAND STATE UNIVERSITY

- Vector architectures accommodate long memory latency, don't rely on caches as does Out-Of-Order, superscalar/VLIW designs
- Simpler hardware: more powerful instructions, more predictable memory accesses, fewer hazards, fewer branches, fewer mispredicted branches, ...
- But the big question always is, what % of the computation is vectorizable?
- However, vector may be a good match to new apps such as multimedia, digital signal processing

The Cell Processor

Porta-CellOverview.pdf

damora-cell4graphicsandviz-gh05.pdf – slides #16-#31

PORTLAND STATE UNIVERSITY

PORTLAND STATE UNIVERSITY

Week 9 (11/21)

- There will be a short quiz on the Week 8 readings and lectures and Week 9 readings
- Part 4 of the project is due
  - Each team will give a short 5-10 minute presentation on their project 1 implementation
- Project 2 (and the final project) will be assigned

- Topics:
  - Data parallel computing, SIMD architectures
  - Adaptive Solutions CNAPS
- Reading:
  - Introduction to SIMD, intro_simd.pdf
  - "Image Processing Using One-Dimensional Processor Arrays," Dan Hammerstrom and Dan Lulich, *The Proceedings of the IEEE*, Vol. 84, No. 7, July 1996, pp. 1005-1018, ieeeproc.pdf

- There will be no class on Wednesday Nov. 23 (and no make-up lecture this time)