Lecture 7: Vector Processing

Professor David A. Patterson

Computer Science 252

Spring 1998

Review: 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

Review: 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*

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
```

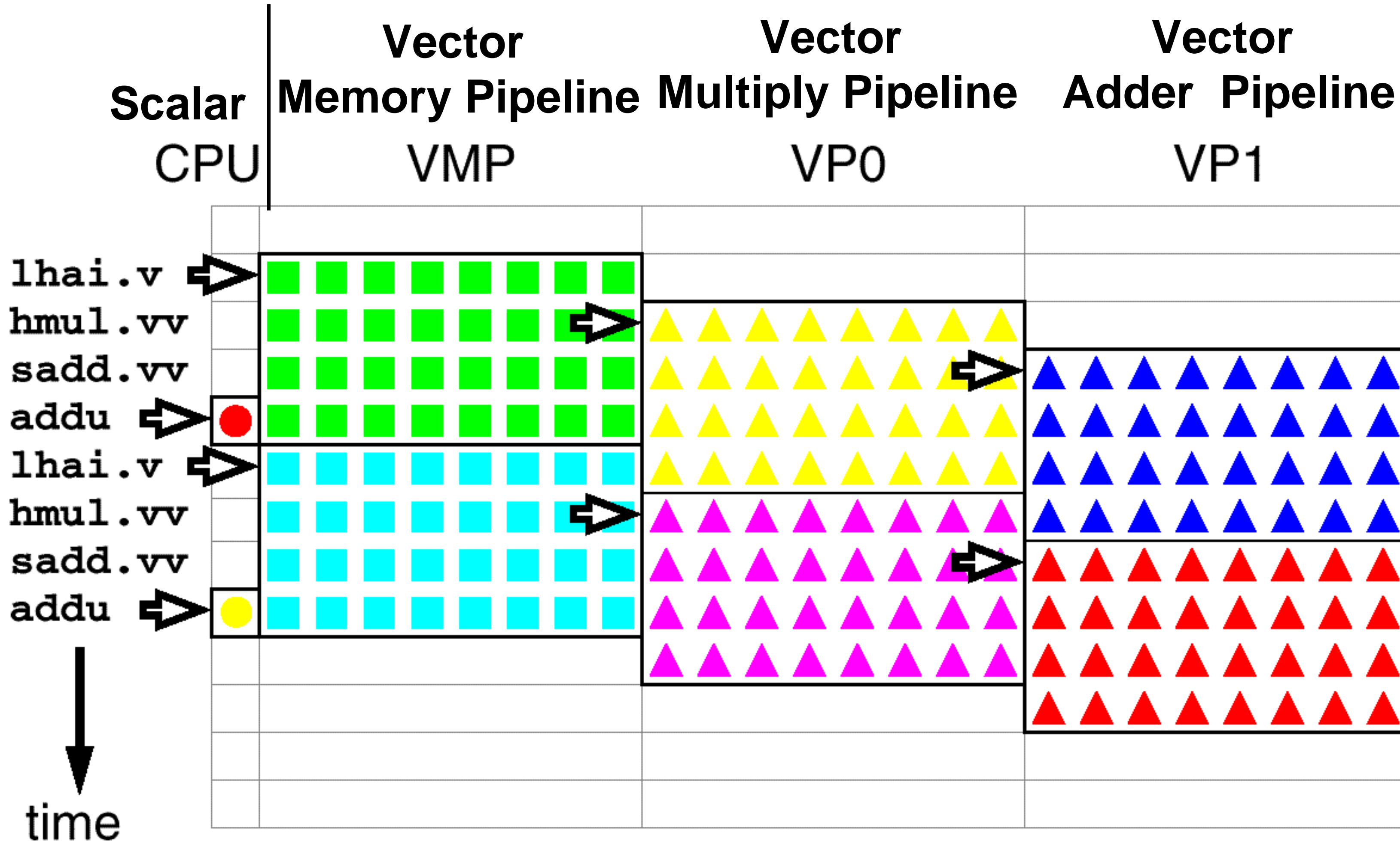
578 (2+9*64) vs.
321 (1+5*64) ops (1.8X)

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

64 operation vectors + no loop overhead

also 64X fewer pipeline hazards

Example Execution of Vector Code



8 lanes, vector length 32, chaining

Operations
Instruction issue

Review: 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

Review: 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?

Outline

- Hard vector example
- Vector vs. Superscalar
- Krste Asanovic's dissertation: designing a vector processor issues
- Vector vs. Superscalar: area, energy
- Real-time vs. Average time

Vector Example with dependency

```
/* 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;
     }
}
```

Straightforward Solution

- **Must sum of all the elements of a vector besides grabbing one element at a time from a vector register and putting it in the scalar unit?**
- **e.g., shift all elements left 32 elements or collapse into a compact vector all elements not masked**
- **In T0, the vector extract instruction, vext.v. This shifts elements within a vector**
- **Called a "reduction"**

CS 252 Administrivia

- Get your photo taken by Joe Gebis! (or give URL)
- 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
- Upcoming events in CS 252

18-Feb DSP/Multimedia Processors #1 (Wed)

20-Feb DSP/Multimedia Processors #2 (Fri)

23-Feb HW #2 due by 5:00 PM (Mon)

25-Feb Memory Hierachy: Caches; Meeting signup

26-Feb Project Survey due (Thurs)

27-Feb Memory Hierarchy Example; Proj. Meetings

4-Mar Quiz 1 (5:30PM - 8:30PM, 306 Soda) (Wed)

CS252 Projects?

- P1: Investigating algorithms, circuits, and floorplan for the VIRAM-1 vector unit.
- P2: Algorithms/Benchmarks on VIRAM
  - Port the NESL Language to VIRAM-1
  - Port the BDTI DSP Benchmarks to VIRAM-1
- P3: TLB design for the VIRAM-1 vector unit
- P5: Explore energy implications of reconfigurable implementation of compute kernels (BRASS)
- P4: Characterize Architecture Metrics for Multiple Commercial Databases (kkeeton@cs)
- P6: Very Large Scale Backup (eklee@pa.dec.com)
- P7. Doing better than SPEC(mash@mash.engr.sgi.com)
- P8. Fixing Knuth Volume 3 & "Algorithms and Data Structures" courses
- P9. I/O nightmares on the way

CS252 Projects?

- **P13. Evaluating New WinTEL I/O Standard "I2O"**
- **P12. Networks vs. Busses (Gray@Microsoft.com)**
- **P14. Evaluating Embedded Processors (kozyraki@CS)**
- **P10. I/O Benchmarks, multimedia,(pfister@us.ibm.com)**
- **P11. Application Performance: Messages vs. CC-NUMA**

Review: 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, 16 vector regs each with 32 elements
  - 1 instruction updates 32 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 32 "virtual processors" (VPs) each with 16 scalar registers! (≈ multithreaded processor)
  - 1 instruction updates 1 scalar register in 64 VPs
- Hardware identical, just 2 compiler perspectives

Novel Matrix Multiply Solution

- You don't need to do reductions for matrix multiply
- You can calculate multiple independent sums within one vector register
- You can vectorize the j loop to perform 32 dot-products at the same time
- Or you can think of each 32 Virtual Processor doing one of the dot products
- (Assume Maximul Vector Length is 32)
- Show it in C source code, but can imagine the assembly vector instructions from it

Original Vector Example with dependency

```
/* 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 Vector Example

```
/* 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 vector
                          register to zeros. */
        for (t=1; t<k; t++)
        {
         a_scalar = a[i][t]; /* Get scalar from
                                a 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. */
```

Optimized Vector Example cont'd

```
            /* Vector-vector add into results. */
            sum[0:31] += prod[0:31];
        }

        /* Unit-stride store of vector of
            results. */
        c[i][j:j+31] = sum[0:31];
    }
}
```

Novel, Step #2

- It's actually better to interchange the i and j loops, so that you only change vector length once during the whole matrix multiply
- To get the absolute fastest code you have to do a little register blocking of the innermost loop.

Designing a Vector Processor

- Changes to scalar
- How Pick Vector Length?
- How Pick Number of Vector Registers?
- Context switch overhead
- Exception handling
- Masking and Flag Instructions

Changes to scalar processor to run vector instructions

- Decode vector instructions
- Send scalar registers to vector unit (vector-scalar ops)
- Syncrhonization for results back from vector register, including exceptions
- Things that don’t run in vector don’t have high ILP, so can make scalar CPU simple

How Pick Vector Length?

- **Vector length => Keep all VFUs busy:**

$$\text{vector length} \geq \frac{(\#\ \text{lanes}) \times (\#\ \text{VFUs})}{\#\ \text{Vector instructions/cycle}}$$

How Pick Vector Length?

- **Longer good because:**
  1) Hide vector startup
  2) lower instruction bandwidth
  3) tiled access to memory reduce scalar processor memory bandwidth needs
  4) if know max length of app. is < max vector length, no strip mining overhead
  5) Better spatial locality for memory access
- **Longer not much help because:**
  1) diminishing returns on overhead savings as keep doubling number of element
  2) need natural app. vector length to match physical register length, or no help

How Pick Number of Vector Registers?

- **More Vector Registers:**

1) Reduces vector register "spills" (save/restore)

- **20% reduction to 16 registers for su2cor and tomcatv**
- **40% reduction to 32 registers for tomcatv**
- **others 10%-15%**

2) agressive scheduling of vector instructinons: better compiling to take advantage of ILP

- **Fewer:**

Fewer bits in instruciton format (usually 3 fields)

Context switch overhead

- Extra dirty bit per processor
  - Ifvector 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: External

- If external exception, can just put pseudo-op into pipeline and wait for all vector ops to complete
  - Alternatively, can wait for scalar unit to complete and begin working on exception code assuming that vector unit will not cause exception and interrupt code does not use vector unit

Exception handling: Arithmetic

- **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

- Page Faults must be precise
- Instruction Page Faults not a problem
- Data Page Faults harder
- Option 1: Save/restore internal vector unit state
  - Freeze pipeline, dump vector state
  - perform needed ops
  - Restore state and continue vector pipeline

Exception handling: Page Faults

- Option 2: expand memory pipeline to check addresses before send to memory + memory buffer between address check and registers
- muiltiple queues to transfer from memory buffer to registers; check last address in queues before load 1st element from buffer.
- Pre Adress Iinstruction Queue (PAIQ) which sends to TLB and memory while in parallel go to Address Check Instruction Queue (ACIQ)
- When passes checks, instruction goes to Commited Instruction Queue (CIQ) to be there when data returns.
- On page fault, only save intructions in PAIQ and ACIQ

Masking and Flag Instructions

- Flag have multiple uses (conditional, arithmetic exceptions)
- Alternative is conditional move/merge
- Clear that fully masked is much more effiecient that with conditional moves
  - Not perform extra instructions, avoid exceptions
- Downside is:

1) extra bits in instruction to specify the flag regsiter

2) extra interlock early in the pipeline for RAW hazards on Flag registers

Flag Instruction Ops

- **Do in scalar processor vs. in vector unit with vector ops?**
- **Disadvantages to using scalar processor to do flag calculations (as in Cray):**

1) if MVL > word size => multiple instructions; also limits MVL in future

2) scalar exposes memory latency

3) vector produces flag bits 1/clock, but scalar consumes at 64 per clock, so cannot chain together

- **Proposal: separate Vector Flag Functional Units and instructions in VU**

Vectors Are Inexpensive

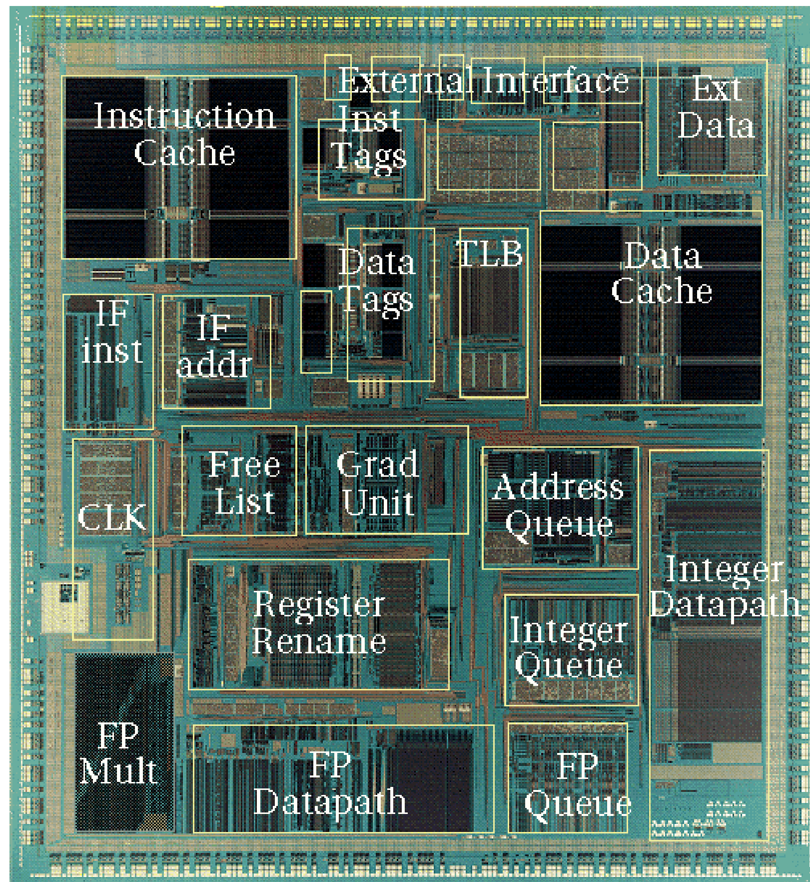
Scalar

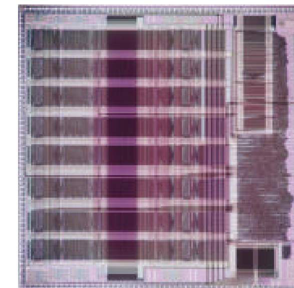
- N ops per cycle
  $\Rightarrow O(N^2)$ circuitry
- HP PA-8000
  - 4-way issue
  - reorder buffer:
    850K transistors
    - incl. 6,720 5-bit register number comparators

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
  - No floating point

MIPS R10000 vs. T0





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

Vectors Lower Power

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 passes through cache; waste 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

Superscalar

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

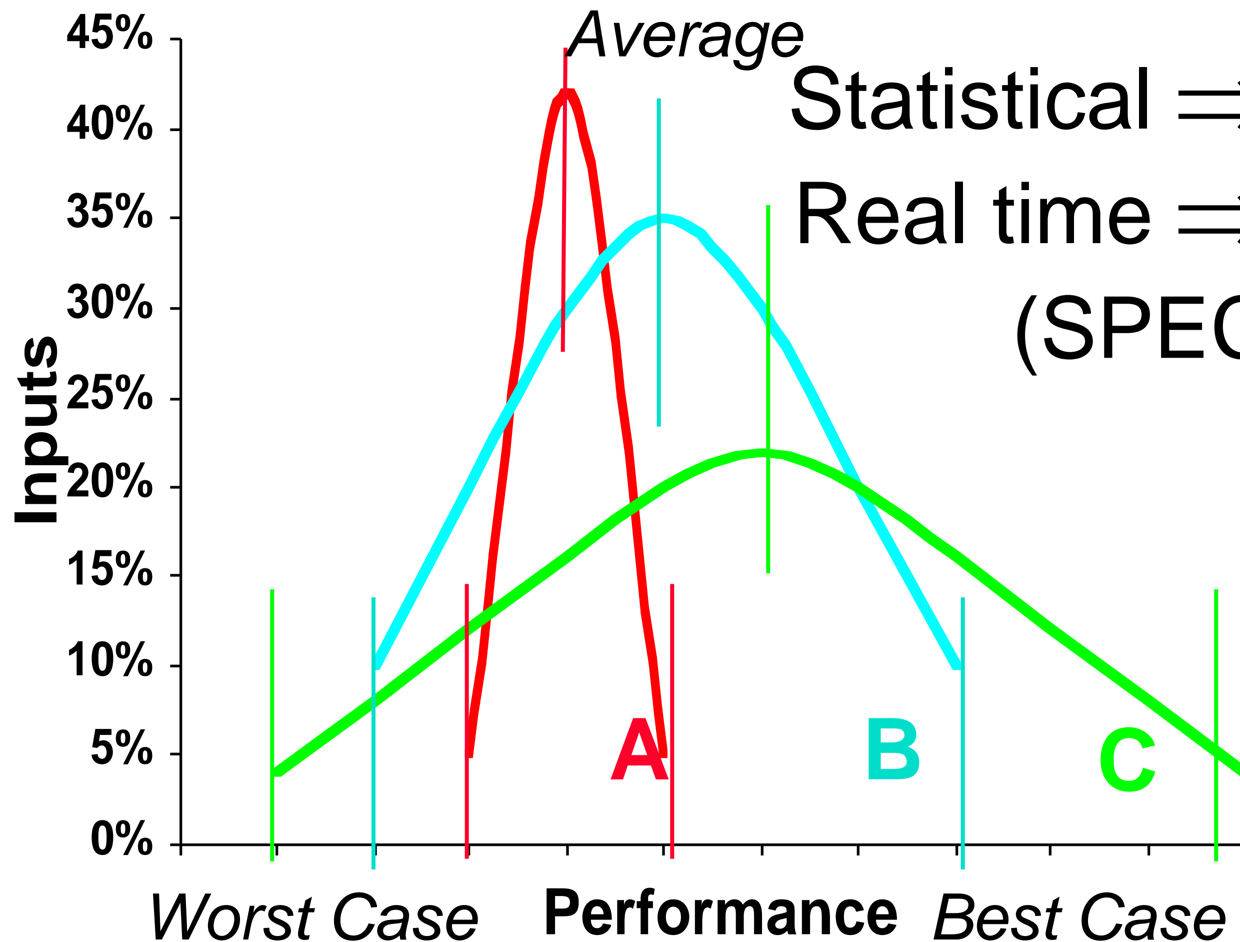
Vector

- Control logic grows linearly with issue width
- 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

New Architecture Directions

- "…media processing will become the dominant force in computer arch. & microprocessor design."
- "... new media-rich applications... involve significant real-time processing of continuous media streams, and make heavy use of vectors of packed 8-, 16-, and 32-bit integer and Fl. Pt."
- Needs include high memory BW, high network BW, continuous media data types, real-time response, fine grain parallelism
  - "How Multimedia Workloads Will Change Processor Design", Diefendorff & Dubey, *IEEE Computer* (9/97)

Which is Faster? Statistical v. Real time v. SPEC Performance

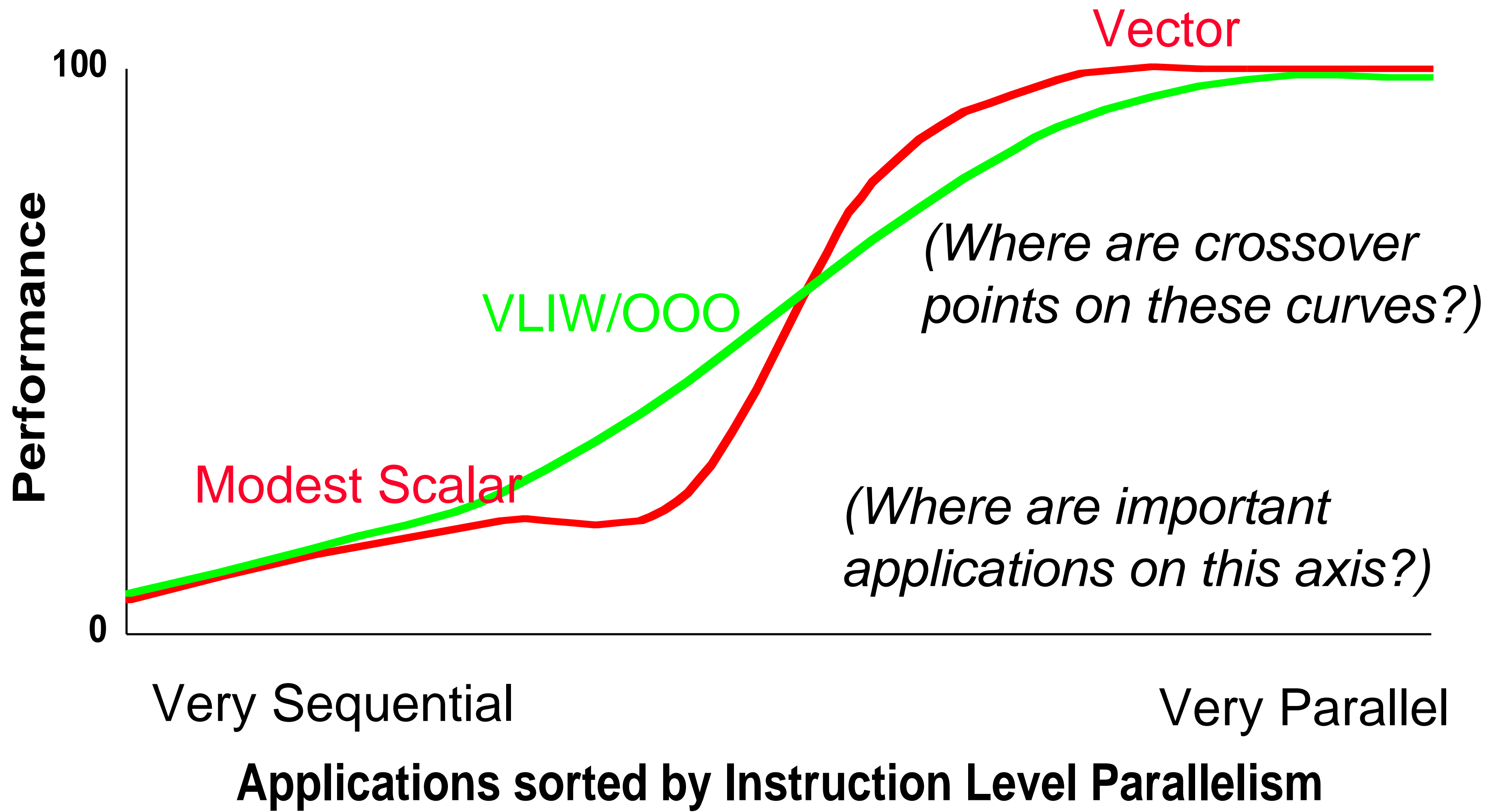


Statistical ⇒ Average ⇒ C

Real time ⇒ Worst ⇒ A

(SPEC ⇒ Best ⇒ C)

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



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

Summary

- Vector is alternative model for exploiting ILP
- If code is vectorizable, then simpler hardware, more energy efficient, and better real-time model than Out-of-order machines
- Design issues include number of lanes, number of functional units, number of vector registers, length of vector registers, exception handling, conditional operations
- Will multimedia popularity revive vector architectures?