Assignments

♡ By next class read about buses:
- Cragon 2.2.8

- Supplemental reading:
  - Hennessy & Patterson 6.3
  - Gustavson: Bus Tutorial (library)
  - Borill bus comparison (library)
  - Siewiorek & Koopman Appendix A

- Supplemental for lecture after that:
  - Skim USB overview slides by Kosar Jaff (see web page)
  - TI PCI technical briefing (see web page)
Where Are We Now?

- Where we’ve been:
  - Vector architecture

- Where we’re going today:
  - Vector performance

- Where we’re going next:
  - Multiprocessor Coherence
  - Fault tolerance

Preview

- Elements of vector performance
  - Peak & sustained performance
  - Amdahl’s Law again

- Benchmark performance
  - Linpack as an example

- Alternatives to vector computing?
  - Hardware alternatives
  - Software alternatives

- Eleven rules for supercomputer design
ELEMENTS OF VECTOR PERFORMANCE

Generic Vector Architecture
Important Factors in Vector Performance

- **FLOP = “FLoating point OPeration”**
  - FLOPs = “FLoating point OPerations”
  - MFLOPS = Million FLOPs per Second
- **Latency**
  - When are vectors so small that using a scalar unit is faster than waiting for a deeply pipelined vector unit?
- **VRF characteristics**
  - How big is the VRF (number of vectors, vector length)?
  - How many ports available to the VRF?
- **Bandwidth available to memory**
  - FLOPs per memory access is a measure of data locality
  - Bandwidth available to memory may limit performance
  - How many VAG units are available for simultaneous accesses?
  - How effective is the memory system at avoiding bank conflicts?

Measures of Vector Computer Performance

- **MFLOPS is the “MIPS” of the supercomputer world**
  - Peak MFLOPS = “guaranteed not to exceed performance rate”
  - Sustained MFLOPS, typically for 100x100 LINPACK
    - May be quoted for 1000x1000 LINPACK on some machines -- why?
- **Performance Measures**
  - Peak performance -- all operations from vector register file (streaming rate limited by execution limits)
  - \( R_* \) -- MFLOPS rate on an infinite-length vector (memory port streaming rate)
  - \( N_{1/2} \) -- vector length needed to reach one-half of \( R_* \)
  - \( N_V \) -- vector length needed to make vector mode at least as fast as scalar mode (“vector/scalar cross-over”)
- **Benchmarks**
  - Linpack
  - Livermore Loops
  - Floating point SPECmarks
  - ...
Peak Performance

- Peak performance typically limited by floating point execution units
  - DSPs typically quote multiply-accumulate as peak
  - Supercomputers typically quote peak DAXPY from register file
- Assume independent multiplier & adder
  - Peak performance is typically 1 multiply + 1 add per clock; peak MFLOPS = 2x clock rate (in MHz)
- But, there is an assumption of balance
  - DAXPY consumes 1 scalar + 2 vectors for each 1 result vector
  - Need holding register for scalar
  - Need 2 read ports + 1 write port to VRF to achieve peak rate

<table>
<thead>
<tr>
<th>Processor</th>
<th>Clock Period</th>
<th>Exec. Units</th>
<th>Peak MFLOPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cray-1</td>
<td>12.5 ns</td>
<td>2</td>
<td>160</td>
</tr>
<tr>
<td>Cray X-MP</td>
<td>9.5 ns</td>
<td>2</td>
<td>210</td>
</tr>
<tr>
<td>NEC SX-2</td>
<td>6.0 ns</td>
<td>8</td>
<td>1333</td>
</tr>
<tr>
<td>Titan 1</td>
<td>125 ns</td>
<td>2</td>
<td>16</td>
</tr>
</tbody>
</table>

Practical Effects of Vector Register Length

- Vectors bigger than vector registers must be strip-mined
  - Introduces additional overhead (preceding data ignored this)
  - Must make sure to have two sets of vector register available to “ping-pong” results and avoid false dependencies

![Graph](Image)
### R. -- Memory-Limited Peak Performance

- Limited by whether execution units can be kept fully fed from memory
  - Assume infinite-length vector, with no re-use of results
  - Equivalent to 100% cache miss rate in a scalar processor
- Balance required for full-speed operation
  - Assume DAXPY operation
    - Read 2 numbers and produce 1 result for each multiply + add
    - Need 3 memory ports (VAGs) for each 2 execution units for full speed
  - Need enough VRF capacity to assure no false dependencies from sharing vector registers

<table>
<thead>
<tr>
<th>Processor</th>
<th>Clock Period</th>
<th>Exec. Units</th>
<th>Peak MFLOPS</th>
<th>Memory Ports</th>
<th>R&lt;sub&gt;*&lt;/sub&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cray-1</td>
<td>12.5 ns</td>
<td>2</td>
<td>160</td>
<td>1</td>
<td>26.6</td>
</tr>
<tr>
<td>Cray X-MP</td>
<td>9.5 ns</td>
<td>2</td>
<td>210</td>
<td>3</td>
<td>210</td>
</tr>
<tr>
<td>NEC SX-2</td>
<td>6.0 ns</td>
<td>8</td>
<td>1333</td>
<td>12</td>
<td>1333</td>
</tr>
<tr>
<td>Titan 1</td>
<td>125 ns</td>
<td>2</td>
<td>16</td>
<td>3</td>
<td>14.5</td>
</tr>
</tbody>
</table>

### N<sub>1/2</sub> -- Performance on “Small” Vectors

- N<sub>1/2</sub> gives a feel for how well shorter vectors perform
  - N<sub>1/2</sub> measured with respect to R<sub>*</sub>,
    (how big a vector to get half R<sub>*</sub> performance)
  - Smaller N<sub>1/2</sub> means that machine achieves a good percentage of peak throughput with short vectors
- N<sub>1/2</sub> determined by a combination of:
  - Vector unit startup overhead
  - Vector unit latency
  - (varies depending on operation being performed)

<table>
<thead>
<tr>
<th>Processor</th>
<th>Clock Period</th>
<th>Exec. Units</th>
<th>Peak MFLOPS</th>
<th>Memory Ports</th>
<th>R&lt;sub&gt;*&lt;/sub&gt;</th>
<th>N&lt;sub&gt;1/2&lt;/sub&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cray-1</td>
<td>12.5 ns</td>
<td>2</td>
<td>160</td>
<td>1</td>
<td>26.6</td>
<td>10-20</td>
</tr>
<tr>
<td>Cray X-MP</td>
<td>9.5 ns</td>
<td>2</td>
<td>210</td>
<td>3</td>
<td>210</td>
<td>10-25</td>
</tr>
<tr>
<td>NEC SX-2</td>
<td>6.0 ns</td>
<td>8</td>
<td>1333</td>
<td>12</td>
<td>1333</td>
<td>large</td>
</tr>
<tr>
<td>Titan 1</td>
<td>125 ns</td>
<td>2</td>
<td>16</td>
<td>3</td>
<td>14.5</td>
<td>18</td>
</tr>
</tbody>
</table>
**N_V -- Use Scalars Instead?**

- N_V is the vector/scalar crossover point
  - Vector length < N_V -- scalars are faster
  - Vector length > N_V -- vectors are faster
  - Depends on latency & startup overhead for vector unit (varies by operation)

**Titan-1 N_V = 2 BUT**
- Titan-1 didn’t have a MIPS R2010 scalar floating point chip; it wasn’t ready in time; so scalars used vector unit as well
- MIPS M/120-5 used R2010 and got about the same performance with no vector unit
- Titan-2 included an R2010...

<table>
<thead>
<tr>
<th>Processor</th>
<th>Clock Period</th>
<th>Exec. Units</th>
<th>Peak MFLOPS</th>
<th>Memory Ports</th>
<th>R*</th>
<th>N_1/2</th>
<th>N_V</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cray-1</td>
<td>12.5 ns</td>
<td>2</td>
<td>160</td>
<td>1</td>
<td>26.6</td>
<td>10-20</td>
<td>1.5-2.5</td>
</tr>
<tr>
<td>Cray X-MP</td>
<td>9.5 ns</td>
<td>2</td>
<td>210</td>
<td>3</td>
<td>210</td>
<td>10-25</td>
<td>~2</td>
</tr>
<tr>
<td>NEC SX-2</td>
<td>6.0 ns</td>
<td>8</td>
<td>1333</td>
<td>12</td>
<td>1333</td>
<td>large</td>
<td>large</td>
</tr>
<tr>
<td>Titan 1</td>
<td>125 ns</td>
<td>2</td>
<td>16</td>
<td>3</td>
<td>14.5</td>
<td>18</td>
<td>2^{I/2}</td>
</tr>
</tbody>
</table>

**BENCHMARK PERFORMANCE**
Benchmarks

- **Linpack**
  - Gaussian elimination using DAXPY operation
  - 100x100 Linpack measures a good balance of overhead & sustained speed

- **Floating point SPECmarks**
  - Combination of 10 scientific programs; emphasis on finite element computation (workstation computing)

- **Livermore Loops**
  - Collection of scientific computation “kernels”
  - Uses harmonic mean to penalize outliers reduces benefit from single peak performance loops
  - Replaced by NAS & PERFECT

- **NAS Parallel Benchmarks (NASA)**
  - 8 programs for parallel supercomputers; computational fluid dynamics

- **Perfect Club Benchmarks (Univ. Illinois)**
  - 13 executable programs for scientific computing

LINPACK as an Example

- **100 x 100 Matrix Gaussian Elimination**
  - Iteratively, a row is multiplied by constant and added to rows below it
  - Each iteration adds a column of leading zeros, reducing next row size by 1

- **Inner loop is DAXPY operation**
  - 1st iteration is vector length 100
  - 2nd iteration is vector length 99 ...
**Linpack Excerpt:**

```c
for (kb = 0; kb < n; kb++)
{
    k = n - (kb + 1);
    b[k] = b[k]/a[lda*k+k];
    t = -b[k];
    daxpy(k,t,&a[lda*k+0],1,&b[0],1);
}
```

```c
void daxpy(int n, REAL da, REAL dx[], int incx, REAL dy[], int incy)
...
for (i = 0;i < n; i++) { dy[i] = dy[i] + da*dx[i];}
```

**Linpack Performance**

- **Balance required for DAXPY peak performance:** \( Y = aX + Y \)
  - Assume \( a \) loaded into a holding register
  - On every clock: load \( X \), store intermediate;
    load intermediate, load \( Y \), store \( Y \)
    - 3 read ports and 2 write ports
  - Internal (to the functional units) chaining can reduce bandwidth requirements:
    load \( X \), load \( Y \), store \( Y \)
    - 2 read ports and 1 write port
    - Intermediate not written to VRF (implements a logical or actual “DAXPY” instruction)

- **Peak performance might be one DAXPY result per clock**
  - 3 memory “touches”, 2 FLOPs per clock
  - Linpack performance limited by available memory bandwidth!
  - Balanced system: 3 concurrent memory pipes + multiplier + adder
Linpack Performance: Multiprocessing

Stardent Titan-1 Data:

<table>
<thead>
<tr>
<th>Array Size</th>
<th>Linpack MFLOPS</th>
<th>Number of Processors</th>
</tr>
</thead>
<tbody>
<tr>
<td>100 x 100</td>
<td>6.5 9.1 11.0 11.7</td>
<td></td>
</tr>
<tr>
<td>300 x 300</td>
<td>9.0 13.4 14.7 15.0</td>
<td></td>
</tr>
<tr>
<td>1000 x 1000</td>
<td>10.5 15.0 15.6 15.7</td>
<td></td>
</tr>
</tbody>
</table>

Linpack is limited by bus bandwidth!

- Multiprocessing helps by obtaining overhead concurrency
  - \( R_w = 14.5 \); but there is spare bandwidth on the write-back bus that can be used by a second CPU with Linpack
- Multiprocessing limited with short vectors because of parallelism overhead
- With long vectors, bus bandwidth limits everything
  - Titan aggregate bus bandwidth is 256 MB/sec
  - 256 MB/sec / (8 B/word * 3 words/2 FLOPs) = 21 MFLOPS theoretical limit
  - But, there are limitations due to multi-processor bank conflicts

ALTERNATIVES TO VECTOR COMPUTERS?
Alternative to Vector Registers: Use Cache

- Cache might be used instead of vector registers
  - Hardware can’t assume dependency-free access
  - Set associativity must be high (at least 3-way set associative to avoid conflicts with DAXPY)
    - Think of a cache set as a vector register
    - BUT, also have to worry about conflicts caused by large stride accesses

- Cache good for:
  - Vector working sets that fit all in cache
  - High FLOP/memory touch ratios
  - Combines scalar & vector mechanisms for cost savings

- But, it has problems:
  - Very long vectors can flush other data from cache (must use cache bypass instructions, which start looking like vector loads/stores)
  - Requires large number of pending memory references to be tracked from processor

Cache Linpack Performance

- Performance depends heavily on data set size
  - In-cache performance high
  - Out-of-cache performance poor

- Opposing performance forces
  - Bigger array reduces overhead
  - Bigger array doesn’t fit in cache
    - But, bottom/right part of array more heavily referenced than top left -- ~45KB array does will in 8 KB L1 cache

- Fine-grain performance variations
  - Different array sizes move in and out of phase with block size (partial block usage affects performance)
Vector Concepts: “General Purpose” Processing

- **Vector-like data abounds**
  - “Scientific” computing: weather forecasting, design optimization
  - Image processing (rows vs. columns); multimedia
  - Radar, Sonar, signal processing
  - Spreadsheets

- **Anticipatory fetching** of data streams
  - Data streams exhibit sequential locality much larger than cache block size
    - Once first cache miss occurs, can avoid future cache misses
    - Vector registers can be thought of as software-managed caches with no conflict misses

- **Strided data accesses**
  - Exploit structured accesses to data as an aid to prefetching
  - Cache blocks are hardware-supported accesses with stride 1
    - Automatic pre-fetching of data with stride 1 after a demand miss
    - Not necessarily well suited to data with large strides (high cache pollution)

Software Prefetching

- **Superscalar CPUs may permit “free” prefetching**
  - Schedule prefetch instructions to load data into cache just before it is needed
  - Can schedule these instructions in otherwise unused issue slots
  - Requires special instruction semantics -- does not generate data dependencies
    - Should not stall processor on cache miss
    - Can be “faked” with load instruction on nonblocking cache, although might incur TLB miss, etc.

- **But, not a perfect solution**
  - For large transport times may require extra instructions and registers for bookkeeping address arithmetic (can it all be scheduled for free?)
  - Large vectors will sweep the cache, clearing other data out
  - Large block sizes can inflate traffic ratios and cache pollution
Stream Buffers -- Run-Time Vectorization

- Data prefetch buffer to “stream” data into caches or CPU
  - Automatic detection of strided access by comparing successive load instructions -- a dynamically operated VAG!
  - Counter generates new addresses to load using speculative prefetching
    - But, be careful about page faults from accidental over-runs!
    - May want more than one buffer (perhaps 3 buffers for DAXPY)
  - These are speculative loads; keep in buffer awaiting cache miss (or, have no cache at all)

![Stream Buffer Diagram]

Palacharla & Kessler; an extension of a paper by Jouppi

Possible HW Solutions for Vector Access

- Sectored caches
  - *e.g.*, block size = 8 bytes to fit a dword; sector size = 64 bytes
    - Avoids cache pollution and limits traffic ratio
    - Still risks conflict misses with strided data -- only 25% of cache can be used
  - Could combine with software hints as to how many blocks to load within a sector upon encountering a cache miss

- Run-time vector HW support
  - Automatic *strided access detection* to start prefetching data
  - Stream buffers to store speculative prefetches without polluting cache
  - “Uncached” loads to avoid sweeping out cache contents with one-time data accesses
    - These are often the loads that might be prefetched via stream buffers

- A new life for vector hardware?
  - *Software-controlled stream buffers* that prefetch data (VAGs)
    - VRF = cache; strided prefetching
    - VRF = prefetch buffer; strided speculative prefetching (doesn’t disrupt cache)
Possible SW Solutions for Vector Access

- Software-controlled prefetching
  - Especially effective with sectored cache
- Better algorithms...
  - A better algorithm wins over better hardware every time

LAPACK -- a better Linpack

- LAPACK = Linpack re-written with block optimizations
  - Loads intermediate results into vector registers
  - Increases FLOP/memory touch ratio
- Titan data demonstrates usefulness of blocked algorithms
  - LAPACK is limited by $N^{1/2}$ not $R$.  

<table>
<thead>
<tr>
<th>Linpack Array Size</th>
<th>MFLOPS</th>
<th>Number of Processors</th>
<th>Linpack Array Size</th>
<th>MFLOPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>100 x 100</td>
<td>6.5</td>
<td>9.1</td>
<td>100 x 100</td>
<td>6.1</td>
</tr>
<tr>
<td>300 x 300</td>
<td>9.0</td>
<td>13.4</td>
<td>300 x 300</td>
<td>10.4</td>
</tr>
<tr>
<td>1000 x 1000</td>
<td>10.5</td>
<td>15.0</td>
<td>1000 x 1000</td>
<td>13.1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LApack Array Size</th>
<th>MFLOPS</th>
<th>Number of Processors</th>
<th>LApack Array Size</th>
<th>MFLOPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>100 x 100</td>
<td>9.1</td>
<td>11.0</td>
<td>100 x 100</td>
<td>10.6</td>
</tr>
<tr>
<td>300 x 300</td>
<td>14.7</td>
<td>15.0</td>
<td>300 x 300</td>
<td>24.6</td>
</tr>
<tr>
<td>1000 x 1000</td>
<td>15.7</td>
<td>15.6</td>
<td>1000 x 1000</td>
<td>36.5</td>
</tr>
</tbody>
</table>
ELEVEN RULES OF
(SUPER)COMPUTER DESIGN

Gordon Bell’s Eleven Rules ...

1) **Performance, performance, performance.**
   - People are buying supercomputers for performance.

2) **Everything matters.**
   - The use of the harmonic mean for reporting performance on the Livermore Loops severely penalizes machines that run poorly on even one loop.

3) **Scalars matter the most.**

4) **Provide as much vector performance as price allows.**
   - Peak vector performance is often determined by a combination of bus bandwidth and vector register capacity.
   - Rule of thumb: enough bandwidth for two results per clock tick

5) **Avoid holes in the performance space.**
   - Divides are uncommon, but not unused.

6) **Place peaks in performance.**
   - Give Marketing something to brag about; be “the best in the world” at something.
... for Supercomputer Design

7) **Provide a decade of addressing.**
   - Support an extra 2 address bits every 3 years of product (10 years \(\approx 7\) bits)

8) **Make it easy to use.**
   - “Dusty deck” code (or Windows code) is what people want to run

9) **Build on other’s work.**
   - Don’t reinvent compiler technology if you’re in the hardware business

10) **Design for the next one, and then do it again.**
    - Pipeline design teams so after first success there is another product underway

11) **Have slack resources. Expect the unexpected.**
    - No matter how good the schedule, unscheduled events will occur.
Review

- **Elements of vector performance**
  - Peak performance is easy
  - Sustained performance requires bandwidth
    - But bandwidth is just money
  - Really good performance requires low latency
    - And latency requires creative design
  - Amdahl’s Law applies...

- **Benchmark performance**
  - Linpack as an example; LApack as a refinement

- **Alternatives to vector computing?**
  - Hardware alternatives -- tweaking cache; adaptive prefetching
  - Software alternatives -- nothing beats a better algorithm!

- **Eleven rules for supercomputer design**