Lecture 4: Modeling Sparse Matrix-Vector Multiply

William Gropp

www.cs.illinois.edu/~wgropp
Sustained Memory Bandwidth

- Measure the rate at which data can be copied from within a program:
  \[
  t = \text{mysecond}() \\
  a(1) = a(1) + t \\
  !$OMP \text{ PARALLEL DO} \\
  \text{DO 30 j = 1,n} \\
  \text{  c(j) = a(j)} \\
  30 \text{ CONTINUE} \\
  t = \text{mysecond}() - t \\
  c(n) = c(n) + t \\
  \text{times}(1,k) = t
  \]

- This is the STREAM COPY Benchmark

- STREAM contains multiple tests (not just copy), and contains multicore versions
  - Extensive historical information available on the web site
Example Results
(My Laptop in 2008)

<table>
<thead>
<tr>
<th>Function</th>
<th>Rate (MB/s)</th>
<th>Avg time</th>
<th>Min time</th>
<th>Max time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Copy:</td>
<td>2900.3744</td>
<td>0.0115</td>
<td>0.0110</td>
<td>0.0121</td>
</tr>
<tr>
<td>Scale:</td>
<td>2752.9018</td>
<td>0.0121</td>
<td>0.0116</td>
<td>0.0137</td>
</tr>
<tr>
<td>Add:</td>
<td>3241.4521</td>
<td>0.0156</td>
<td>0.0148</td>
<td>0.0188</td>
</tr>
<tr>
<td>Triad:</td>
<td>3265.9560</td>
<td>0.0151</td>
<td>0.0147</td>
<td>0.0165</td>
</tr>
</tbody>
</table>
Example Results (My newer Laptop in 2015)

<table>
<thead>
<tr>
<th>Function</th>
<th>Best Rate MB/s</th>
<th>Avg time</th>
<th>Min time</th>
<th>Max time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Copy:</td>
<td>16970.7</td>
<td>0.009641</td>
<td>0.009428</td>
<td>0.010048</td>
</tr>
<tr>
<td>Scale:</td>
<td>13321.1</td>
<td>0.012168</td>
<td>0.012011</td>
<td>0.012475</td>
</tr>
<tr>
<td>Add:</td>
<td>13147.8</td>
<td>0.018488</td>
<td>0.018254</td>
<td>0.019308</td>
</tr>
<tr>
<td>Triad:</td>
<td>13101.7</td>
<td>0.019142</td>
<td>0.018318</td>
<td>0.019389</td>
</tr>
</tbody>
</table>
Aside on Trends

• Raw numbers for performance improvement look good
  ♦ And they are!

• But the ratio is about a factor of 4 in 6 years
  ♦ A “mere” 26% improvement every year
  ♦ Much less than a doubling in performance every 2 years or less
Sparse Matrix-Vector Product

• Common operation for optimal (in floating-point operations) solution of linear systems

• Sample code:

```c
for row=1,n
    m   = i[row] - i[row-1];
    sum = 0;
    for k=1,m
        sum += *a++ * x[*j++];
    y[row] = sum;
```

• Data structures are a[nnz], j[nnz], i[n], x[n], y[n]
Sample Code in Fortran

• Arrays are \(ia(n+1), ja(nnz), a(nnz), x(n), y(n)\)
• Offset = 0
  Do row=1,n
    \(m = ia(row+1) - ia(row)\)
    sum = 0
    do k=1,m
      sum = sum + a(offset+k) * x(ja(offset+k))
    enddo
    y(row) = sum
    offset = offset + m
  enddo
• This is called CSR (Compressed Sparse Row) format
Question

• Don’t look at the next slide yet. See if you can estimate the performance of this operation:
  ♦ How many floating point operations are there?
  ♦ How many load operations?
  ♦ How many store operations?
There are nnz steps in the loop
- Each performs 2 floating point operations
- Each loads 3 values: A(k), ja(k), x(ja(k))
  - We’ll assume ja is half the size of A, x, and y
- Each stores 1 value: y(row)
- Also load n values: ia(row)
- We assume “sum” is stored in a register and is not written to memory

Time = nnz(2c + 2.5r) + n(0.5r+w)

However, this is too pessimistic. We need a slightly better model
Main memory contains the program data

Cache memory contains a *copy* of the main memory data

- Cache is *faster* but consumes more space and power
- Cache items accessed by their address in main memory

Registers contain working data only

- Modern CPUs perform most or all operations only on data in register
Improved Performance Model

• Assume values are only loaded once
  ♦ Because nnz > n, and there are only n values of X, X is only loaded n times, not nnz times

• Assumes that after the first time:
  – X is in cache
  – Cache memory is infinitely fast
Simple Performance Analysis

- Memory motion - Loads:
  - \( \text{nnz} \) (\( \text{sizeof(double)} \) + \( \text{sizeof(int)} \)) + \( n \) (\( \text{sizeof(double)} \) + \( \text{sizeof(int)} \))
  - Assume a perfect cache (never load same data twice)

- Memory motion – Stores:
  - \( n \) (\( \text{sizeof(double)} \))

- Computation
  - \( \text{nnz multiply-add (MA)} \)
Sparse Matrix-Vector Multiply
Performance Expectations

• Assume nnz >> n
  ♦ Then load ja(k) and a(k) (typically 4 + 8 = 12 bytes)
    for each multiply and add operation
• Roughly 12 bytes per MA
• Typical workstation node can move 1-4 bytes/MultiplyAdd
  ♦ Thus we can estimate a bound on the maximum possible performance:
    ♦ 4 bytes moved/12 bytes needed for operation is 33% of peak
    ♦ 1 byte move/12 bytes needed for operation is 8% of peak
• Thus, maximum performance is 8-33% of peak
More Performance Analysis

• Instruction Counts:
  ♦ $\text{nnz} (2*\text{load-double} + \text{load-int} + \text{mult-add}) + n (\text{load-int} + \text{store-double})$

• Roughly 4 instructions per MA

• Maximum performance is 25% of peak (33% if MA overlaps one load/store)
  ♦ (wide instruction words can help here)

• Changing matrix data structure (e.g., exploit small block structure) allows reuse of data in register, eliminating some loads ($x$ and $j$)

• Implementation improvements (tricks) cannot improve on these limits

• Details of the estimate depend on the details of the *execution model* (what does the model hardware provide) and the fidelity of that execution model to the real hardware.
Realistic Measures of Peak Performance
Sparse Matrix Vector Product
One vector, matrix size, $m = 90,708$, nonzero entries $nz = 5,047,120$

Theoretical Peak
Mem BW Peak
Oper. Issue Peak
Observed

Thanks to Dinesh Kaushik; ORNL and ANL for compute time
Realistic Measures of Peak Performance
Sparse Matrix Vector Product
one vector, matrix size, \( m = 90,708 \), nonzero entries \( nz = 5,047,120 \)

Theoretical Peak
Oper. Issue Peak
Mem BW Peak
Observed
Observations

• Clock rate based performance analysis is often not useful
• Models that make use of sustained memory bandwidth can provide a better prediction of performance
• Both models provide upper bounds on performance
  ♦ In this example, most of the data was accessed in a regular way
    • Good fit to cache design
    • Operations are close to STREAM model
    • Not always so simple
Question

• Assume a processor with a 2.8 GHz clock, and able to perform one floating point operation per clock cycle

♦ What is the peak performance of the processor, defined as the maximum number of floating point operations per second?
Question

- Assume that the sustainable memory bandwidth is 12 Gbytes/second. For a DAXPY operation, what is the maximum possible performance, using the same analysis as we used for the Sparse matrix-vector multiply. A DAXPY is

- Do $i=1,n$

  \[ y(i) = \alpha \times x(i) + y(i) \]

  enddo

- What is the ratio of the performance for DAXPY and the peak performance for the processor?