1 Modeling SMVM Using PAPI Data

We instrumented the register profile collection code with PAPI. Collecting a register profile consists of executing Sparsity’s register blocked $r \times c$ SMVM routines on a dense, 1000×1000 matrix stored in blocked CSR (BCSR) format.

Profiles were collected on various platforms using wall-clock timers. A summary of the characteristics of each platform appears in Table 1; the profiles themselves are shown in Figure 1. The irregular structure of these plots is striking, as is the lack of an obvious function of $r$ and $c$ to explain the structure. Our goal in this section is to use the PAPI-instrumented versions to try and understand these differences.

1.1 Sanity-checking the raw PAPI data

First, we verify that PAPI’s count of load and store instructions matches our expectations. (For simplicity, we omit ceiling and floor operations.) The data below was collected on a Sun Ultra 2i platform as described in Table 1.

Let $k$ be the number of non-zeros in the matrix. For an $m \times n$ dense matrix stored in $r \times c$ BCSR format, there are $k/rc$ blocks, requiring storage for $k$ double-precision non-zero values, $k/rc$ column indices (one for each non-zero block), $m+1/r$ row pointer indices. The matrix data must be loaded once. Furthermore, for each $r \times c$ block, we must load the $c$ corresponding elements of the source vector; thus, there are $k/rc \times c = k/r$ source vector loads, assuming that there are a sufficient number of registers to perform the multiplication of a block without spills. Finally, we assume that for each block row, we can load the elements of the destination vector into registers once, and keep them in registers for the duration of the block-row multiply; this requires $m$ loads and $m$ stores in all.

Thus, the total number of load operations is

$$\frac{k}{rc} + \frac{k}{r} + \frac{m+1}{r} + \frac{k}{rc} + \frac{m}{rc}$$

Note that for the register profile, the matrix size is actually $\lceil 1000 \rceil r \times \lceil 1000 \rceil c$. Sparsity collects the register profile in this way to obtain a raw estimate of the peak performance of each register blocked routine. For the block sizes considered, the true matrix size differs from the 1000×1000 case by no more than 2%.
<table>
<thead>
<tr>
<th>Property</th>
<th>Sun Ultra 2i</th>
<th>Pentium III</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clock rate</td>
<td>333 MHz</td>
<td>500 MHz</td>
</tr>
<tr>
<td>Peak Memory Bandwidth</td>
<td>500 MB/s</td>
<td>680 MB/s</td>
</tr>
<tr>
<td>Peak Flop Rate</td>
<td>667 Mflop/s</td>
<td>500 Mflop/s</td>
</tr>
<tr>
<td>DGEMM (1000×1000)</td>
<td>425 Mflop/s</td>
<td>331 Mflop/s</td>
</tr>
<tr>
<td>DGEMV (1000×1000)</td>
<td>58 Mflop/s</td>
<td>96 Mflop/s</td>
</tr>
<tr>
<td>L1 data cache size</td>
<td>16 KB</td>
<td>16 KB</td>
</tr>
<tr>
<td>L1 line size</td>
<td>16 B</td>
<td>32 B</td>
</tr>
<tr>
<td>L1 latency</td>
<td>2 cy</td>
<td>3 cy</td>
</tr>
<tr>
<td>L1 associativity</td>
<td>direct</td>
<td>4-way</td>
</tr>
<tr>
<td>L2 cache size</td>
<td>2 MB</td>
<td>512 KB</td>
</tr>
<tr>
<td>L2 line size</td>
<td>64 B</td>
<td>32 B</td>
</tr>
<tr>
<td>L2 latency</td>
<td>7 cy</td>
<td>19 cy</td>
</tr>
<tr>
<td>TLB entries</td>
<td>64</td>
<td>64</td>
</tr>
<tr>
<td>Page size</td>
<td>8 KB</td>
<td>4 KB</td>
</tr>
<tr>
<td>TLB latency</td>
<td>36 cy</td>
<td>8 cy (?)</td>
</tr>
<tr>
<td>Memory latency</td>
<td>66 cy</td>
<td>60 cy (?)</td>
</tr>
<tr>
<td>sizeof(double)</td>
<td>8 B</td>
<td>8 B</td>
</tr>
<tr>
<td>sizeof(int)</td>
<td>4 B</td>
<td>4 B</td>
</tr>
<tr>
<td>Compiler</td>
<td>Sun cc v6.1</td>
<td>Intel C v5</td>
</tr>
</tbody>
</table>

Table 1: Basic data for the machines used in our experiments. Performance figures for the BLAS on the Sun Ultra 2i platform are the best of Sun’s performance library v6.0 and ATLAS 3.2.0; on the Pentium III platform, figures reported are the best of Intel’s MKL v5.2, ATLAS 3.2.0, and ITXGEMM 1.1.
Figure 1: Sparsity register blocking profiles collected on several different platforms. Each register profile shows the performance of $r \times c$ register blocked routines on a dense 1000\times1000 matrix stored in BCSR format. The parameters that characterize each platform is shown in Figure 1. Observe that the performance profiles between any two machines is qualitatively quite different.

$$= k \left( 1 + \frac{1}{rc} + \frac{1}{r} \right) + \frac{m + 1}{r} + m$$ (1)

and the total number of store operations is $m$. In Figure 2 (left), we show PAPI’s raw load instruction count for all $r, c$; in Figure 2 (right), we show the ratio of the raw count to the preceding analytic count. The figure confirms that the data matches our expectations.

Figures 3 (left, right) show the analogous data for the stores. Note that at 10\times4, there is a large increase in the number of stores. Inspection of the assembly code revealed extra stores due to register spilling at this particular register block size.

1.2 Modeling Cache Misses

We show the raw L1 cache miss count reported by PAPI in Figure 4.

Let $l_1$ be the L1-cache line size, in double-precision words; let $\gamma$ be the ratio of double-precision word size to integer index word size. We expect to incur one cold-start L1 read miss for every matrix element (value and index) and destination vector element. The source vector miss count is more complicated to predict, however, because it depends on the relative placement in memory of the various arrays. Since the source vector size (1000 elements, or 8000 bytes) is less than the L1 cache size, in the best case we would incur only $n$ cold-start misses for the source vector. Thus, a lower bound $M_{\text{lower}}^{(1)}$ on L1 misses is

$$M_{\text{lower}}^{(1)} = \frac{1}{l_1} \left[ k \left( 1 + \frac{1}{\gamma rc} \right) + n + \frac{m + 1}{\gamma r} + m \right].$$ (2)
Figure 2: (Left) Raw load instruction count produced by PAPI for the instrumented register profile code on the Sun Ultra 2i platform. Each square is the load count for an $r \times c$ blocked SMVM routine, colored by the load count. (Right) Ratio of PAPI’s load count to the analytic model in expression (1). Ideally, the ratio would be 1 for all blocks; the largest ratio is 1.025, and occurs at $10 \times 4$. PAPI passes the first sanity check.

Figure 3: (Left) Raw store instruction count produced by PAPI for the instrumented register profile code on the Sun Ultra 2i platform. Each square is the store count for an $r \times c$ blocked SMVM routine, colored by the store count. (Right) Ratio of PAPI’s store count to the analytic model in expression (1). Ideally, the ratio would be 1 for all blocks; the ratio very close to 1 for all implementations except at $10 \times 4$. Inspection of the assembly code revealed extra stores due to register spilling.
The factor of \( \frac{1}{l_1} \) accounts for the L1 line size, and the factor of \( \frac{1}{\gamma} \) accounts for the difference in word size between a double-precision word and an integer index.

In the worst case, we will miss on every access to the source vector; thus, an upper bound \( M_{\text{upper}}^{(1)} \) on L1 misses is

\[
M_{\text{upper}}^{(1)} = \frac{1}{l_1} \left[ k \left( 1 + \frac{1}{\gamma rc} \right) + \frac{k}{r} + \frac{m + 1}{\gamma r} + m \right].
\]  

We compare these analytic bounds to the actual L1 miss count reported by PAPI in Figure 4 (right). The upper bound appears to be more accurate than the lower bound when \( c \) is small, and vice-versa as \( c \) increases. We do not know of an obvious explanation for this phenomenon.

We can repeat the above analysis for the L2-cache. Figure 5 shows the analogous results. The lower bound appears to be a better match to the actual data than the upper bound, indicating the assumption that the source vectors remains in cache is valid for the L2-cache. We expect the assumption is more likely to hold for L2 than L1 because the source vector size of 8 KB is half the size of the L1 cache, but only a small fraction (less than 1%) of the L2 cache size.

Figure 4: (Left) L1-cache miss count reported by PAPI on the Sun Ultra 2i platform. The count has been normalized by the true flop count for each implementation. (Right) Compares the L1-cache miss count to the analytic bounds described in expressions (2) and (3). The \( r \times c \) implementations have been sorted in lexicographic order; implementations with the same \( r \) and increasing \( c \) appear consecutively. The reported miss counts lie within the analytic bounds. Note that \( l_1 = 2 \) and \( \gamma = 2 \) for this platform (see Table 4).
Figure 5: (Left) L2-cache miss count reported by PAPI on the Sun Ultra 2i platform. The miss count has been normalized by the true flop count for each implementation. (Right) Compares the L2-cache miss count to analytic bounds. The reported miss counts lie within the analytic bounds, though they appear to be much closer to the lower bound than the upper bound. Note that $l_2 = 8$ for this platform.

1.3 Modeling Execution Time

1.3.1 Analytic Bounds

We can compare the execution time of the sparse routines to several reference points based on hardware parameters and dense library routines:

- **Peak flops rate**: The processor’s peak floating point performance (in, say, Mflop/s) provides the lowest bound on execution time. We expect this bound to be unrealistic for a memory bandwidth-limited kernel like SMVM on a typical workstation.

- **GEMM**: The performance of the best dense double-precision matrix-matrix multiply BLAS routine, DGEMM, frequently executes at near peak machine speed.

- **Peak memory bandwidth**: In the best case, the source and destination vectors are cached and we only have to bring the matrix through memory.

- **GEMV**: The performance of the best dense double-precision matrix-vector multiply BLAS routine, Dgemv, is a more “practical” time estimate.

If we know the number of hits $h_i$ and misses $m_i$ at cache level $i$, then a simple model of execution time (in cycles) is

$$T = h_1\alpha_1 + h_2\alpha_2 + m_2\alpha_{mem},$$

where $\alpha_i$ is the access time (in cycles) at cache level $i$ and $\alpha_{mem}$ be the memory access time. This expression assumes a perfect nesting of the caches, so that a miss at level $i$ is an access at level $i + 1$.  
However, on the Sun Ultra 2i, we cannot accurately predict the memory access term, \( m_2 \alpha_{\text{mem}} \), due to two factors: interaction with the TLB and the lack of TLB counters. We can obtain upper and lower bounds, however. In the best case, assume that an L2 miss incurs a miss penalty + TLB access time; in the worst case, an L2 miss incurs the full memory access time \( \alpha_{\text{mem}} \). So the previous expression for \( T \) gives us an upper bound \( T_{\text{upper}} \), and the lower bound is \( T_{\text{lower}} \):

\[
T_{\text{lower}} = h_1 \alpha_1 + h_2 \alpha_2 + m_2 \alpha_{\text{TLB}}.
\]  

(5)

1.3.2 Predicting cycles on the Sun Ultra 2i platform

Using our analytic expressions for the number of loads (equation 1) and lower bounds on L1 and L2 miss rates (equations 2–3), we can compute \( T_{\text{lower}} \). Using the analytic upper bounds on miss rates, we can also compute \( T_{\text{upper}} \). We compare \( T_{\text{lower}} \) and \( T_{\text{upper}} \) to the actual cycle time in Figure 6 (top). The analytic bounds envelop the true cycle count.

Moreover, we can see how close the actual cycle time is to the lower bound, as shown in 6 (bottom). For the block sizes up to 6×6 (the most frequently selected, as indicated by prior experience), the true cycle time is between 14% and 34% higher than the lower bound.

In Figure 7, we refine the cycle-time bounds using the actual hit and miss counter data reported by PAPI. The data-based lower bound is nearly identical to the model-only lower bound; the data-based upper bound is tighter than its model-only counterpart. Using actual data captures the envelope of the true cycle time, but not the detailed features.

Finally, in Figure 8, we show the breakdown of the three terms—L1, L2, and memory access—in the cycle time model of equation (4).

1.3.3 Predicting cycles on the Pentium III platform

Repeating the same analyses on the Pentium III platform as in the previous section, we find similar results. The same bounds on L1 and L2 cache misses shown in equations (2–3) apply, as shown in Figure 4. Note that the Pentium III cache line sizes are reflected in the change of scale in the figures compared to those of the Sun Ultra 2i (Figures 4–5).

In Figure 10, we show the data-based cycle time bounds for the Pentium III. The lower bound appears to break down as row sizes increase beyond 8. The model does not apply there because the Pentium III only has 8 double-precision floating point registers, which is insufficient to hold entire blocks of the destination and source vectors as \( r + c \) exceeds 8. Figure 11 (left) shows the increase in L1 accesses as \( r + c \) becomes approximately greater than 8. Figure 11 (right) shows the resulting increase in time spent performing L1 accesses.

1.3.4 Linear Model

We also fit a parametric model of execution time using PAPI’s counter data as inputs and cycle time as the output. Let \( X \) be a matrix of counter values, where
Figure 6: (Top) Cycle time per true flop, measured by PAPI (“Actual”) and predicted by the various bounds, as a function of block size (linearized). Data is shown for the Sun Ultra 2i platform. Bounds shown as horizontal lines are computed assuming the volume of data or flops for a 1000×1000 dense matrix stored in sparse format. The bounds $T_{\text{lower}}$ and $T_{\text{upper}}$ shown here are computed using our analytic expressions. (Bottom) Ratio of the actual execution time (in cycles) to the predicted lower bound $T_{\text{lower}}$ shown above.
Figure 7: Same as Figure 6 (top), except that the lower and upper bounds are computed using the hit and miss counts reported by PAPI. This data-based lower bound is very similar to the model-only lower bound, but the data-based upper bound is tighter than the model-only upper bound.
Figure 8: Breakdown of the cycles spent accessing each level of the memory hierarchy as a function of block size (linearized) on the Sun Ultra 2i platform. The four lines shown correspond to the terms of equation (4): $h_1\alpha_1$, $h_2\alpha_2$, $m_2\alpha_{TLB}$ (lower bound, equation 5), and $m_2\alpha_{mem}$ (upper bound). As expected, memory access is the dominant cost.
Figure 9: Cache miss counts (L1-cache, top row; L2-cache, bottom row) compared to the analytic model for the Pentium III platform. These plots equivalent to those for the Sun Ultra 2i platform shown in Figures 4–5. Note that $l_1 = 4$, $l_2 = 4$, $\gamma = 2$ for this platform (see Table 4).
Figure 10: Cycle time per true flop, measured by PAPI (“Actual”) and predicted by the various bounds, as in Figure 6. Data is shown for the Pentium III platform. This plot is the analogue of Figure 7 for the Pentium III.

Figure 11: (Left) Ratio of observed L1 data cache accesses to predicted accesses on the Pentium III platform. The ratio exceeds 1 as the Pentium III’s 8 register are no longer sufficient to store the \( r + c \) source and destination vector elements for a given matrix block. (Right) Breakdown of the cycles spent accessing each level of the memory hierarchy as a function of block size (linearized) on the Pentium III platform. Register spills are evident in the increased contribution of the L1 access term, \( h_1 \alpha_1 \).
Table 2: Best fit parameters for the linear model in equation (6) applied to PAPI data for the Sun Ultra 2i platform (see Figure 12).

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>L2 cache misses (PAPI_L2_TCM)</td>
<td>58.91</td>
<td>cy/miss</td>
</tr>
<tr>
<td>L1 load misses (PAPI_L1_LDM)</td>
<td>5.09</td>
<td>cy/miss</td>
</tr>
<tr>
<td>L1 store misses (PAPI_L1_STM)</td>
<td>187.58</td>
<td>cy/miss</td>
</tr>
<tr>
<td>Cond branch mispredictions (PAPI_BR_MSP)</td>
<td>660.23</td>
<td>cy/branch</td>
</tr>
</tbody>
</table>

Each column of $X$ corresponds to a counter and each row corresponds to an $r \times c$ SMVM implementation. Let $y$ be the vector of measured cycle times for each of the $r \times c$ implementations. We computed a vector of parameters $\beta$ such that

$$y \approx X\beta.$$ (6)

We perform a least squares fit to compute $\beta$. The model was manually, successively refined to include only “reasonable” counter values. The resulting fitted model (i.e., predicted execution time) is shown in Figure 12 (top). The model parameter coefficients are given in Table 2.

The contribution of each term $\beta_i x_i$ (where $x_i$ is the $i$th column of $X$) in the model to the predicted time is shown in Figure 12 (bottom). As expected, the predominant contribution to time in the fitted model comes from L2-cache misses (main memory accesses).

Stuff to come:

- repeat analyses for an actual matrix
- more machines (Intel P4, Itanium; IBM Power3/4)
- ...
Figure 12:  (Top) Least squares fit of a linear model of execution time based on available PAPI counters. Data shown is for the Sun Ultra 2i platform. (Bottom) Contribution of each term in the linear model to the execution time, as a fraction of total predicted cycles. The term associated with L2-cache misses gives the largest contribution to the final time.