Fusion of Linear Algebra Kernels and the Memory Subsystem

Ian Karlin
Joint work with
Elizabeth Jessup, Erik Silkensen,
Geoffrey Belter and Jeremy Siek

University of Colorado at Boulder
Linear Algebra Used in Many Applications

- Climate Modeling
- Magnetic Field Simulation
- Starburst Event Modeling
Memory Gap Growing

- FLOPS
- Memory Speed

Year:
- 1980
- 1985
- 1990
- 1995
- 2000
- 2005
- 2010

Speed:
- 1
- 10
- 100
- 1,000
- 10,000
- 100,000
- 1,000,000
- 10,000,000

The graph shows the growing gap between FLOPS and Memory Speed over the years from 1980 to 2010.
Libraries Limit Loop Fusion

\[
\begin{align*}
q &= Ap \\
s &= A^Tr
\end{align*}
\]

\[
\text{DGEMV('n', m, n, alpha, a, lda, p, l, beta, q, l);} \\
\text{DGEMV('y', m, n, alpha, a, lda, r, l, beta, s, l);} \\
\text{for (i=0;i<n;++i)} \\
\text{for (j=0;j<m;++j) \{ \}
\text{q[j] += p[i]*A[i][j];} \\
\text{s[i] += r[j]*A[i][j];} \\
\text{\}}
\]
\( q = Ap \)
\( s = A^T r \)

for (i=0; i<N; ++i)
for (j=0; j<M; ++j) {
  \( q[j] += p[i] * A[i][j]; \)
  \( s[i] += r[j] * A[i][j]; \)
}
Subset of MATLAB

Generate Loops

Enumerate Optimizations

Analyze

Generate C Code
Goals

- Stress test compiler to expose bugs
- What happens when you fuse a lot
Test Code

DGEMV2
in
  u0 : vector, u1 : vector, A : row matrix
out
  v0 : vector, v1 : vector
{
  v0 = A * u0
  v1 = A * u1
}
### Defining Terms

<table>
<thead>
<tr>
<th>outer loops</th>
<th>all loops</th>
</tr>
</thead>
<tbody>
<tr>
<td>for (i = 0; i &lt; n; i++)</td>
<td>$v_0[i] += A[i][j] \times u_0[j]$</td>
</tr>
<tr>
<td>for (j = 0; j &lt; n; j++)</td>
<td>$v_1[i] += A[i][j] \times u_1[j]$</td>
</tr>
<tr>
<td>$v_0[i] += A[i][j] \times u_0[j]$</td>
<td>$v_1[i] += A[i][j] \times u_1[j]$</td>
</tr>
</tbody>
</table>

- **No fusion**
- **Outer loops**
- **All loops**

### Accounting for Registers

To be able to account for register usage in the model, the number of registers was determined, the bandwidth between the level - cache and processor calculated, and the model code modified to account for how the native compiler allocates registers. The following section first describes how to incorporate registers into the model and then shows how adding them improves performance prediction.

#### 5.1. The Changes Needed to Add in Registers

The first step in including registers to the model is to represent the registers as a memory structure. To do so on the Opteron system described in Section 
we first determined that there are eight general purpose registers. However, because one is reserved for the stack pointer and another for the base pointer, only six registers can be used for general purpose computation. Then we determined the bandwidth between the level - cache and the processor by means of the DAXPY benchmark in STREAM and stored it as the cost of register misses. The next step involved modifying the code to account for the fact that registers are not allocated in a least recently used fashion. Instead, the native compilers attempt to allocate registers in a manner that reduces the number of reads into registers. To figure out which variables remain in registers, the following heuristics are used in the model to mimic the native compiler's allocation. The iterate of an inner loop is stored in a register. A variable that is accessed within an inner loop more than once is stored in a register if one is available. Finally, when a register
Test Results

matrix order = 6500

- all outer loops fused
- fully fused
- predicted fully fused
Interesting Results

- Too much inner loop fusion detrimental as nvecs increases
- Model inaccurate for large nvecs
- Too much outer loop fusion detrimental as matrix order increases
Test Results

matrix order = 6500

- all outer loops fused
- fully fused
- predicted fully fused
Not a cache Problem

L1 Cache

L2 Cache
Or the TLB
It’s the Registers

<table>
<thead>
<tr>
<th>nvecs = 4</th>
<th>nvecs = 5</th>
<th>nvecs = 6</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>faddl (%edi,%eax,8)</code></td>
<td><code>faddl (%esi,%eax,8)</code></td>
<td><code>mov 0x5c(%esp),%edi</code></td>
</tr>
<tr>
<td><code>fstpl (%edi,%eax,8)</code></td>
<td><code>fstpl (%esi,%eax,8)</code></td>
<td><code>faddl (%edi,%eax,8)</code></td>
</tr>
<tr>
<td><code>faddl (%esi,%eax,8)</code></td>
<td><code>mov 0x4c(%esp),%edi</code></td>
<td><code>fstpl (%edi,%eax,8)</code></td>
</tr>
<tr>
<td><code>fstpl (%esi,%eax,8)</code></td>
<td><code>faddl (%edi,%eax,8)</code></td>
<td><code>fstpl (%edi,%eax,8)</code></td>
</tr>
</tbody>
</table>
Fixing the Model

- Add registers to model
- Keep in mind how compilers allocate registers
That's better
Large Sizes
It’s the Cache

Chart showing that its L2 cache performance that causes dropoff

Matrix order

L2 misses per flop

nvecs = 8

GotoBLAS
fully composed
all outer loops
Two Options

Do not fuse all outer loops

Cache block
What’s the Best Amount of Fusion?

for for statement A statement B statement C
for statement D statement E statement F
Application to Parallel Codes

- Advantages
  - Multi-core = more registers
  - Multi-core = more cache to core bandwidth

- Disadvantages
  - SMP/NUMA best data layout = tall and skinny
  - Still memory bound

Distributed memory data layout
Conclusions

- Fusion produces positive and negative memory effects
- We can model these effects
- Our compiler infrastructure makes it easy to test hunches
Future Work

- Partitioning
- Cache Blocking
- Parallel Codes
- Modeling of the above
- Reduce search space
Questions

Special thanks:
Erik Silkensen

Contact:
Ian.Karlin@colorado.edu

http://ecee.colorado.edu/wpmu/btoblas/