Mesa: Automatic Generation of Lookup Table Optimizations

Chris Wilcox
Michelle Strout
James Bieman
Colorado State University
Problem

- Scientific codes often require extensive tuning to perform well on multicore systems.
- Performance optimization consumes a major share of development effort on multicore systems.
- Manual tuning, including parallelization, is inefficient and can obfuscate application code.
Context

- Many scientific apps are performance limited by the evaluation of elementary function calls.
- Lookup table (LUT) optimizations are often coded by hand to accelerate elementary functions.
- Optimizations must be compatible with parallel execution of the application.

Table 1: Performance of elementary function instructions. (Intel Core 2 Duo E8300, 2.83GHz, 6MB L2 cache, single core)

<table>
<thead>
<tr>
<th>x86 Instruction</th>
<th>Execution Time</th>
<th>Math Library</th>
<th>Execution Time</th>
<th>Relative Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>FSIN</td>
<td>35.2ns</td>
<td>sin</td>
<td>36.5ns</td>
<td>+3.6%</td>
</tr>
<tr>
<td>FCOS</td>
<td>33.9ns</td>
<td>cos</td>
<td>36.9ns</td>
<td>+8.8%</td>
</tr>
<tr>
<td>FPTAN</td>
<td>72.9ns</td>
<td>tan</td>
<td>51.8ns</td>
<td>-28.9%</td>
</tr>
<tr>
<td>FSQRT</td>
<td>8.1ns</td>
<td>sqrt</td>
<td>1.8ns</td>
<td>-77.7%</td>
</tr>
</tbody>
</table>
Lookup Tables

- Replace expensive expression and function evaluation with accesses to table of previously computed results.
- Table optimizations involve a fundamental tradeoff between performance and accuracy.

\[ f(\theta) = \text{original function}, \quad l(\theta) = \text{table approximation}, \quad e(\theta) = \text{absolute error} \]
Results (Manual)

• From our Small Angle X-ray Scattering (SAXS) simulation code based on Debye’s equation:

\[ I(\theta) = 2\sum_{i=1}^{N-1}\sum_{j=i+1}^{N} F_i(\theta) F_j(\theta) \sin(4\pi r_{ij}\theta) / (4\pi r_{ij}\theta) \]

<table>
<thead>
<tr>
<th>Run Time</th>
<th>Cumulative Factor</th>
<th>Delta Factor</th>
<th>Version Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1496s</td>
<td>1X</td>
<td>1.0X</td>
<td>Original code</td>
</tr>
<tr>
<td>167s</td>
<td>9X</td>
<td>8.9X</td>
<td>Removed redundancy</td>
</tr>
<tr>
<td>27.5s</td>
<td>53X</td>
<td>6.0X</td>
<td>LUT optimization</td>
</tr>
<tr>
<td>14.3s</td>
<td>105X</td>
<td>1.9X</td>
<td>Parallel code, dual-core *</td>
</tr>
<tr>
<td>7.7s</td>
<td>194X</td>
<td>1.9X</td>
<td>Parallel code, quad-core *</td>
</tr>
</tbody>
</table>

* Parallel version uses OpenMP pragmas
Approach

- Automate the tedious and error prone elements of LUT optimization via the Mesa tool.
- Help programmers to improve performance with clear knowledge of the effect on accuracy.
Methodology

1) **Identify functions** and expressions for LUT optimization.

2) **Profile** the **domain** and distribution of LUT input values.*

3) **Determine** the LUT **size** based on domain and granularity.

4) **Analyze** the **error** characteristics and memory usage of LUT. *

5) **Generate** structures and **code** to initialize and access LUT data. *

6) **Integrate** the generated LUT **code** into the application. *

7) **Compare performance** and accuracy of original vs. optimized.

* automated by Mesa tool
Error Analysis

- Allows the programmer to control the tradeoff between domain, error, and performance.
- Mesa analyzes the error over the entire table using exhaustive traversal or stochastic sampling.
- Error decreases in proportion to LUT size, but the relationship is not always linear.
// Iterate steps (outer loop)
for (step = 0; step < 1000; ++step) {
    // Iterate atoms (middle loop)
    for (atom1 = 0; atom1 < vecAtoms.size(); ++atom1) {
        // Iterate atoms (inner loop)
        for (atom2 = atom1; atom2 < vecAtoms.size(); ++atom2) {

            // Compute distance between atoms
            float fDistance = distance(atom1, atom2);
            // Compute scattering angle
            float fTheta = m_fStep * (float)(step + 1);
            // Combine parameters to scatter
            float rTheta = fDistance * fTheta;

            // Optimize subexpression shown below
            #pragma LUTOPTIMIZE
            fIntermediate = sinf(FOURPI * rTheta) / (FOURPI * rTheta);
        }
    }
}
Results (Automated)

• Results from using Mesa to automate optimization of the dominant expression in the inner loop.
• Current performance matches that of the manually developed code for identical LUT size.

Table 2: SAXS performance and error comparison.

<table>
<thead>
<tr>
<th>Protein</th>
<th>Version</th>
<th>Time</th>
<th>Speedup</th>
<th>$E_{MAX}$</th>
<th>$E_{AVG}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1xib</td>
<td>Original</td>
<td>167.0s</td>
<td>-</td>
<td>0.00068%</td>
<td>0.00012%</td>
</tr>
<tr>
<td></td>
<td>Optimized</td>
<td>29.7s</td>
<td>5.62x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1hbb</td>
<td>Original</td>
<td>346.2s</td>
<td>-</td>
<td>0.00071%</td>
<td>0.00015%</td>
</tr>
<tr>
<td></td>
<td>Optimized</td>
<td>61.4s</td>
<td>5.64x</td>
<td>0.00071%</td>
<td>0.00015%</td>
</tr>
<tr>
<td>4gcr</td>
<td>Original</td>
<td>42.0s</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Optimized</td>
<td>7.6s</td>
<td>5.53x</td>
<td>0.00092%</td>
<td>0.00021%</td>
</tr>
</tbody>
</table>

Intel Core 2 Duo CPU (E8300), 2.83GHz, 6MB L2 cache
Mesa Tool

- Example of Mesa expression optimization via an inserted pragma, with exhaustive error analysis:

```plaintext
./Mesa ScatterOriginal.cpp ScatterOptimized.cpp
   -exhaustive -pragma -lutsizel 100000
Mesa LUT optimization started
Variable: rTheta
Lower Bound: 2.000000e-02
Upper Bound: 2.000000e+01
Granularity: 1.998000e-04
Lut size (lut): 100000
Error analysis: exhaustive
Table: Emax: 5.476423e-04, Eavg: 1.884966e-05
Mesa LUT optimization completed
```
Mesa Results

- SAXS scattering code benefits from LUT optimization, until incurring L2 cache penalties.
Other Results

- Expression optimization is highly effective, can improve application performance if computation dominates.

<table>
<thead>
<tr>
<th>Expression Name</th>
<th>Expression Description</th>
<th>Parameter Domain</th>
<th>Table Size</th>
<th>Estimated Error</th>
<th>Measured Performance</th>
<th>Performance Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gamma Correction</td>
<td>$I' = I^{(1/\gamma)}_{\text{ORIGINAL}}$</td>
<td>intensity = [0, 1]</td>
<td>$1.0 \times 10^5$</td>
<td>$1.3 \times 10^{-3}$</td>
<td>$14.6$ ns</td>
<td>0.34 ns</td>
</tr>
<tr>
<td>Cauchy's Equation</td>
<td>$t = 1.458 + (.00354/\lambda^2)$</td>
<td>$\lambda = [0.0, 1.8]$</td>
<td>$1.0 \times 10^3$</td>
<td>$7.0 \times 10^{-4}$</td>
<td>$0.66$ ns</td>
<td>0.17 ns</td>
</tr>
<tr>
<td>Normal Distribution</td>
<td>$Z = 1/\sqrt{2\pi}e^{-x^2/2}$</td>
<td>$x = [0, 10]$</td>
<td>$1.0 \times 10^5$</td>
<td>$1.2 \times 10^{-5}$</td>
<td>$15.3$ ns</td>
<td>0.34 ns</td>
</tr>
<tr>
<td>Logistic Curve</td>
<td>$P(t) = 1/(1 + e^{-t})$</td>
<td>$t = [0, 10]$</td>
<td>$1.0 \times 10^3$</td>
<td>$1.3 \times 10^{-3}$</td>
<td>$4.7$ ns</td>
<td>0.17 ns</td>
</tr>
</tbody>
</table>
Parallel Performance

- LUT optimization and parallelization are complementary.

Parallel Execution of SAXS Code

- Speedup vs. Cores (Parallel Efficiency)
- Ideal (Optimized)
- Discrete (Optimized)
- Ideal (Original)
- Discrete (Original)

Graph showing speedup percentages for different numbers of cores.

- 96.0% speedup with 1 core
- 82.0% speedup with 2 cores
- 77.5% speedup with 4 cores
- 63.8% speedup with 8 cores
- 72.0% speedup with optimized discrete model
Parallel Efficiency

- LUT optimization does not compromise parallel efficiency.

Cray XT6m, AMD Opteron 6100, 512KB L2 cache, 12mb L3 cache
Conclusions

Our methodology and associated tool improves the LUT optimization process:

- Our Mesa tool supports LUT optimization of elementary functions and expressions.
- We show that LUT optimizations can be applied without extensive manual tuning.
- We show that LUT optimization is complementary to code parallelization.
- Code is freely available at our website: http://www.cs.colostate.edu/saxs
Acknowledgments

- Award Number 1R01GM096192 from the National Institute Of General Medical Sciences.
- Grant number DE-SC0003956 from the Department of Energy.

Additional support comes from seed funding from the Vice President of Research and the Office of the Dean of the College of Natural Sciences at Colorado State University and from a Department of Energy Early Career grant.
Related Work

- [Tang91] Seminal work that presents the use of lookup table algorithms to approximate elementary functions, including detailed error analysis.
- [Schulte93] Lookup table based algorithms for high precision elementary function implementations in hardware context.
- [Deng09] Optimization of hardware lookup table implementations, including automatic analysis of power, space, and performance tradeoffs.
- [Zhang10] Special purpose compilers to generate multicore lookup table optimization code for function evaluation in software.