Subjects

• Definitions
• Barcelona Overview
• Branch Predictor + Conditional Moves
• Software Pipelining
• Vector Operations
• How to Take Advantage of Given Optimizations
SIMD and Vector

**SIMD**

\[
\begin{array}{cccc}
1 & 2 & 3 & 4 \\
1 & 2 & 3 & 4 \\
1 & 2 & 3 & 4 \\
1 & 2 & 3 & 4 \\
\end{array}
\]

+ \\

\[
\begin{array}{cccc}
1 & 2 & 3 & 4 \\
1 & 2 & 3 & 4 \\
1 & 2 & 3 & 4 \\
1 & 2 & 3 & 4 \\
\end{array}
\]

= \\

\[
\begin{array}{cccc}
1 & 2 & 3 & 4 \\
1 & 2 & 3 & 4 \\
1 & 2 & 3 & 4 \\
1 & 2 & 3 & 4 \\
\end{array}
\]

**Vector**

\[
\begin{array}{cccc}
1 & 2 & 3 & 4 \\
1 & 2 & 3 & 4 \\
1 & 2 & 3 & 4 \\
1 & 2 & 3 & 4 \\
\end{array}
\]

+ \\

\[
\begin{array}{cccc}
1 & 2 & 3 & 4 \\
1 & 2 & 3 & 4 \\
1 & 2 & 3 & 4 \\
1 & 2 & 3 & 4 \\
\end{array}
\]

= \\

\[
\begin{array}{cccc}
1 & 2 & 3 & 4 \\
1 & 2 & 3 & 4 \\
1 & 2 & 3 & 4 \\
1 & 2 & 3 & 4 \\
\end{array}
\]
Software Pipelining references attempts to manually guarantee the existence of superscalar code within the Lookahead window.
Barcelona Overview

Three level cache architecture

• L1/L2 – High speed computational caches (256 and 128 bits per cycle respectively)

SIMD Instruction Set

• Limited “Vector” operations
• Three SIMD pipelines

This presentation focuses on single precision
Branches

Four sub-topics

• SSE conditional moves
• Integer comparators
• Branch prediction mechanism
• Case study: CORDIC Arctangent
SSE Conditional Moves

Branches that can be expressed as moves conditional on simple floating point comparisons

SSE friendly

if(a < b) {
    sgn = 1.0;
} else {
    sgn = -1.0;
}

ans = c + sgn * d;

if(a < b) {
    ans = c + d;
} else {
    ans = c – d;
}
Integer Comparison Unit

As recommended in the AMD 10h optimization guide, floating point comparisons can be performed in integer units (chart taken from guide):

<table>
<thead>
<tr>
<th>Comparison Against Zero</th>
<th>Replaces</th>
</tr>
</thead>
<tbody>
<tr>
<td>if(*((unsigned int *)&amp;f) &gt; 0x80000000U)</td>
<td>if(f &lt; 0.0f)</td>
</tr>
<tr>
<td>if(*((int *)&amp;f) &lt;= 0)</td>
<td>if(f &lt;= 0.0f)</td>
</tr>
<tr>
<td>if(*((int *)&amp;f) &gt; 0)</td>
<td>if(f &gt; 0.0f)</td>
</tr>
<tr>
<td>if(*((unsigned int *)&amp;f) &lt;= 0x80000000U)</td>
<td>if(f &gt;= 0.0f)</td>
</tr>
</tbody>
</table>
Branch Predictor

Dynamic branches predicted based on 2-bit Global History Bimodal Counters

<table>
<thead>
<tr>
<th>Value</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action</td>
<td>No Branch</td>
<td>No Branch</td>
<td>Branch</td>
<td>Branch</td>
</tr>
</tbody>
</table>

Counters updated as branch conditionals evaluated
Branch Predictor cont.

Heuristic improved by addressing GHBC with Global Branch History

<table>
<thead>
<tr>
<th>Global Branch History</th>
<th>GHBC</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 0 0</td>
<td>➔ 3</td>
</tr>
<tr>
<td>1 0 1 1</td>
<td>➔ 0</td>
</tr>
</tbody>
</table>

Unfortunately, loops waste half of the available bits!
### Branch Predictor cont.

| ... | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 |

- **Dots** – Instruction address
- **Red** – Loop branch bits
- **Green** – Legitimate branch bits

Unrolling loops can help improve the Legitimate/Loop ratio and make the GHBCs work better.
Branch Benchmarks

Branch Speed Comparison

- Branch
- Block Branch
- Int Branch
- Int Block Branch
- SSE Conditional Move (x1)
- SSE Conditional Move (x4)

Managed by UT-Battelle for the U. S. Department of Energy
CORDIC Arctangent

• CORDIC algorithms popular for implementing trig functions with limited hardware
• Basic algorithm is SSE friendly, but includes a tricky conditional move
CORDIC Arctangent (SIMD)

```c
for(i = 0; i < N; i++) {
    if(y < 0.0f) {
        s = 1.0f;
    } else {
        s = -1.0f;
    }

    n_x = x - s * y * powf(2.0f, -(float)i); //powf and atanf are implemented in
    n_y = y + s * x * powf(2.0f, -(float)i); //implemented in
    n_z = z - s * atanf(powf(2.0f, -(float)i)); //lookup tables

    x = n_x;
    y = n_y;
    z = n_z;
}
```
CORDIC Arctangent (Vector)

for(i = 0; i < N; i++) {
    if(y < 0.0f) {
        s = 1.0f;
    } else {
        s = -1.0f;
    }
}

n_x = x + (-s) * y * powf(2.0f, -(float)i) ;
n_y = y + s * x * powf(2.0f, -(float)i) ;
n_z = z + (-s) * 1.0 * atanf(powf(2.0f, -(float)i)) ;

x = n_x;
y = n_y;
z = n_z;
CORDIC Performance

Cordic Conditional Performance

- Conditional Moves
- Floating Point Branches
- Integer Branches

- Single Element, Parallel
- Single Element, Vector
- Pipelined, Vector
CORDIC Performance Cont.

Cordic Conditional Performance cont.

- Pipelined, Vector
- Full SIMD, Parallel
- Full SIMD, Pipelined, Parallel
Software Pipelining

• Out-of-Order scheduling is built around the idea of automatically pipelining code
• Out-of-Order scheduler is effective while multiple independent blocks of code appear in its look ahead window

• Questions:
  • How far ahead does the scheduler look?
  • How effective is “effective”?
Out-of-Order Scheduler

 Blocked vs. Shuffled Instructions

<table>
<thead>
<tr>
<th>Instruction Block Size</th>
<th>Blocked</th>
<th>Shuffled</th>
<th>Gain</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>15.10GFLOPS</td>
<td>16.00GFLOPS</td>
<td>5.62%</td>
</tr>
</tbody>
</table>
Software Pipelining

• Scheduler is quite effective as long as instructions are available in its window.
• Instructions easily made “available” by two techniques:
  • Unrolling loops
  • Staging loops
Software Pipelining

Original

Unrolled

Staged
Software Pipelining

Sine Performance

- Staged, Blocked, Pipelined
- Staged, Shuffled, Pipelined
- Non-Staged, Blocked, Pipelined
- Non-Staged, Shuffled, Pipelined
- Staged, Non-Pipelined
- Non-Staged, Non-Pipelined

Managed by UT-Battelle for the U. S. Department of Energy
Vector Optimizations

• Some codes pre-shuffle values to make vector operations fit better to SIMD instruction sets
• For multiplication on complex numbers we can trade a dependency and a shuffle for a load
Vector Optimizations

Effects of Buffering (Barcelona)

Average Clock Cycles/Complex Op vs Buffer Block Size (floats)

Regular
Buffered
Vector Optimizations

Effects of Buffering (Athlon)

Average Clock Cycles/Complex Op vs Buffer Block Size (floats) for Regular and Buffered conditions.

Managed by UT-Battelle for the U.S. Department of Energy
Vector Optimizations

Comparison of Buffering Importance

Percent Gain of Buffered Implementation

Buffer Block Size (floats)

-20 -10 0 10 20 30 40

Barcelona
Athlon

Managed by UT-Battelle for the U. S. Department of Energy
How to Use This Stuff

• All these tests were written in C with the help of the Intel Assembly Intrinsics
  • Unfortunately, the intrinsics do not work well on PGI or Pathscale compilers (they do work in GNU, Intel, and Microsoft ones)
  • Intrinsics, while better than assembly, are clunky at best
How to Use This Stuff

for(i = 0; i < ITERS; i++) {
    n_two = _mm_set1_ps(-2.0f);
    m = (__m128)_mm_shuffle_epi32((__m128i)v1, 0x0);
    m = _mm_cmpgt_ps(m, zero);
    n_two = _mm_and_ps(n_two, m);
    n_two = _mm_add_ps(n_two, p_one);
    n_v = (__m128)_mm_shuffle_epi32((__m128i)v1, 0xF1);
    r1 = _mm_load_ps(&lin_lookup[i * 4]);
    n_v = _mm_mul_ps(n_v, r1);
    n_v = _mm_mul_ps(n_two, n_v);
    v1 = _mm_add_ps(v1, n_v);
}

Managed by UT-Battelle for the U. S. Department of Energy
Conclusions

• Conditionals
  • Unroll slow conditional loops
  • Use integer comparisons
  • Code for conditional moves
• Pipeline operations explicitly
• Separate staged computations
• Don’t worry about vector operations
Square Root/Reciprocal Time

![Bar chart](image)