GPUMixer: Performance-Driven Floating-Point Tuning for GPU Scientific Applications

Ignacio Laguna, Paul C. Wood, Ranvijay Singh, Saurabh Bagchi

ISC High Performance 2019
June 17th, 2019, Frankfurt, Germany

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 (LLNL-PRES-777665)
Performance-Driven Floating-Point Tuning for GPU Scientific Applications

Run 1
LULESH
6 digits of accuracy, 10% speedup

Run 2
FP64 (double precision)
Mixed-Precision (FP64 & FP32)
3 digits of accuracy, 46% speedup
Floating-Point Precision Levels in NVIDIA GPUs Have Increased

![Graph showing FP64:FP32 Performance Ratio](image)

- **Tesla** (2008): 1:8
- **Fermi** (2009): 1:8
- **Kepler** (2012): 1:24
- **Maxwell** (2014): 1:32

**FP64:FP32 Performance Ratio**

- **2006**
  - FP32 (single only)
  - Compute capability 1.2
- **2008 - 2019**
  - FP32, FP64
  - Compute capability 1.3

Mixed-Precision Programming is Challenging

- Scientific programs have many variables
- \{\text{FP32, FP64}\} precision: \(2^N\) combinations
- \{\text{FP16, FP32, FP64}\} precision: \(3^N\) combinations

```c
double compute(...) {
  ...
  return r;
}

int main() {
  double x;
  x = compute() + ...;
}
```
Example of Mixed-Precision Tuning

**Force computation kernel in n-body simulation (CUDA)**

```c
__global__ void bodyForce(double *x, double *y, double *z, double *vx, double *vy, double *vz, double dt, int n)
{
    int i = blockDim.x * blockIdx.x + threadIdx.x;
    if (i < n) {
        double Fx=0.0; double Fy=0.0; double Fz=0.0;
        for (int j = 0; j < n; j++) {
            double dx = x[j] - x[i];
            double dy = y[j] - y[i];
            double dz = z[j] - z[i];
            double distSqr = dx*dx + dy*dy + dz*dz + 1e-9;
            double invDist = rsqrt(distSqr);
            double invDist3 = invDist * invDist * invDist;
            Fx += dx*invDist3; Fy += dy*invDist3; Fz += dz*invDist3;
        }
        vx[i] += dt*Fx; vy[i] += dt*Fy; vz[i] += dt*Fz;
    }
}
```

**Error of particle position (x,y,z)**

\[
\frac{|x-x_0|}{x} + \frac{|y-y_0|}{y} + \frac{|z-z_0|}{z}
\]

(x,y,z): baseline position
(x₀,y₀,z₀): new configuration
Example of Mixed-Precision Tuning (2)

**Force computation kernel in n-body simulation (CUDA)**

```c
#include <cuda_runtime_api.h>

__global__ void bodyForce(double *x, double *y, double *z, double *vx, double *vy, double *vz, double dt, int n)
{
    int i = blockDim.x * blockIdx.x + threadIdx.x;
    if (i < n) {
        double Fx = 0.0; double Fy = 0.0; double Fz = 0.0;
        for (int j = 0; j < n; j++) {
            double dx = x[j] - x[i];
            double dy = y[j] - y[i];
            double dz = z[j] - z[i];
            double distSqr = dx*dx + dy*dy + dz*dz + 1e-9;
            double invDist = rsqrt(distSqr);
            double invDist3 = invDist * invDist * invDist;
            Fx += dx*invDist3; Fy += dy*invDist3; Fz += dz*invDist3;
        }
        vx[i] += dt*Fx; vy[i] += dt*Fy; vz[i] += dt*Fz;
    }
}
```

**Table 1.**

<table>
<thead>
<tr>
<th>No.</th>
<th>Variables in FP32</th>
<th>Error</th>
<th>Speedup(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>All</td>
<td>15.19</td>
<td>53.70</td>
</tr>
<tr>
<td>2</td>
<td>invDist3</td>
<td>4.08</td>
<td>5.78</td>
</tr>
<tr>
<td>3</td>
<td>distSqr</td>
<td>1.93</td>
<td>-43.35</td>
</tr>
<tr>
<td>4</td>
<td>invDist3, invDist, distSqr</td>
<td>1.80</td>
<td>11.69</td>
</tr>
</tbody>
</table>

The example illustrates that some configurations can produce low performance speedup or even performance degradation; the goal of our approach is to find via static analysis configurations such as 3 and 5 that improve performance and discard cases such as 4.
**Configuration**: set of operations $N = n_1 + n_2$

- $N$: total program operations
- $n_1$: precision level 1
- $n_2$: precision level 2

### Most previous work

- Satisfy accuracy
- Improve Performance

### Our work

#### Config 1
- $+$, FP64
- $-$, FP64
- $/$, FP64

#### Config 2
- $+$, FP32
- $-$, FP64
- $/$, FP64

#### Config 3
- $+$, FP32
- $-$, FP64
- $/$, FP32
Program Variables are not the Right Level to Define Configurations

What if the optimal configuration is?
GPUMixer Overview

GPU Program

- \( \text{kernel}_1 \)
- \( \text{kernel}_2 \)
- \( \text{kernel}_3 \)

Profiling Run (Optional)

Compiler Static Analysis

Fast Mixed-Precision Configurations

Accuracy-Driven Analysis

GPU program
- Performance speedup
- Accuracy constraints satisfied

Dynamic analysis
Fast Imprecise Sets (FISets) for Mixed-Precision

- Type cast operations are costly
- Performance model
  - Arithmetic-to-cast ratio
  \[
  r_{ac} = \frac{O}{C}
  \]
  \[O = \text{arithmetic operations}\]
  \[C = \text{type casting operations}\]
  \[r_{ac} \gg 1.0\]
Algorithm for FISet Identification

**Step 1**: Arithmetic-to-Cast Operations Ratio = 1:3

\[ r_{ac} = \frac{0}{c} = \frac{1}{3} \]

FP64

FP32
Algorithm for FISet Identification (2)

Step 2: Arithmetic-to-Cast Operations Ratio = 2:3

\[ r_{ac} = \frac{0}{C} = \frac{2}{3} \]
Algorithm for FISet Identification (After N Steps)

- 3 type cast operations
- 4 arithmetic operations

Step N: Arithmetic-to-Cast Operations Ratio = 4:3

\[ r_{ac} = \frac{0}{C} = \frac{4}{3} > 1 \]
Calculating FISets in Loops

- **Model**: $L_0 > L_1 > L_2 > ...$
  - $L_0$ encloses $L_1$, $L_1$ encloses $L_2$, ...
- **Case 1**: all nodes of the FISet are in the same $L_x$
  - No special treatment

**Case 2**

$L_x > L_y$

$L_x$: Arithmetic operations

$L_y$: Casting operations

**Case 3**

$L_x > L_y$

$L_x$: Casting operations

$L_y$: Arithmetic operations

Loop $L_x(...)$
{
  a*b
  Loop $L_y(...)$
  {
    Cast
  }
}

Arithmetic operations will executed equal or more times than casting

Loop $L_x(...)$
{
  Cast
  Loop $L_y(...)$
  {
    a*b
  }
}

Casting operations may be executed more times than arithmetic operations
Compilation Process is Based on Clang/LLVM

- GPU Program
  - kernel
  - kernel
  - kernel

- clang
- FISets Code Transformation
- CUDA IR
- PTX
- Object
- nvcc
- GPU Executable
Shadow Computations are Used to Calculate the Error Introduced in a FISet

- FISets introduce error
- Techniques exist on serial code
- Could not use any on CUDA
- Our method:
  - Shadow computations
  - Used before in serial code
Overview of Shadow Computations

Program Execution

CUDA Kernel

FISet 1
(FP64)

\[
\begin{align*}
a &= x + y \\
b &= a \ast z \\
\cdots
\end{align*}
\]

(FP64)

Shadow Operations

\[
\begin{align*}
a_\_ &= x_\_ + y_\_ \\
b_\_ &= a_\_ \ast z_\_ \\
\cdots
\end{align*}
\]

(FP32)

Relative Error

\[
\text{error}_a, \text{error}_b, \cdots
\]

Total Error
(for FISet 1)

\[
\left| \frac{v_{FP64} - v_{FP32}}{v_{FP64}} \right|
\]

\[v_{FP64} : \text{result of FP64 operation}\]

\[v_{FP32} : \text{result of FP32 operation}\]
Our CUDA Runtime System Keeps Track of the Per-Thread Error

```c
main() {
    kernel1<<<N,M>>>();
    kernel2<<<N,M>>>();
    kernel3<<<N,M>>>();
}
```

- Keep global-memory data structure `total_error [INST][THREADS]`
  - `INST`: number of static instructions
  - `THREADS`: kernel threads
- No synchronization needed
- Single location may be executed several times
  - Every error is aggregated
- **Total error** aggregated at the end of kernel
## Trial Runs are Sorted by FISet Total Error (Default Search)

<table>
<thead>
<tr>
<th>Program Configurations (1 for each FISet)</th>
<th>Total Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>FISet a</td>
<td>0.00012</td>
</tr>
<tr>
<td>FISet b</td>
<td>0.00448</td>
</tr>
<tr>
<td>FISet c</td>
<td>0.00619</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

### Process Steps:
1. **Run Program**
2. **Check if output satisfies accuracy constraints**
Evaluation

- **Comparison approach:** Precimonious [Rubio-Gonzalez et al. SC’13]
  - Uses generic search algorithm: delta-debugging
  - Original version doesn’t consider parallel code
  - Implemented our own version for CUDA (called Precimonious-GPU)

- Three CUDA programs: LULESH, CoMD, CFD (Rodinia)

- LLNL system
  - NVIDIA Tesla P100 GPUs
  - Clang 4.0
  - CUDA 8.0
Three Modes of Operation

1. User specifies **accuracy threshold**
   - Search based in FISet total error

2. User specifies **accuracy threshold** and **performance speedup**
   - Accuracy has priority
   - Search like in mode 1
   - Ends when both constraints are satisfied

3. User specifies **accuracy threshold** and **performance speedup**
   - Performance has priority
   - Search based on the ratio $r_{ac}$ (start with the largest ratio)
   - Ends when both constraints are satisfied

Example of 3 digits of accuracy
FP64: 3.1415
Mixed-Precision: 3.1479
Overhead of Shadow Computations and Threshold Settings

- Overhead of shadow computation analysis: $24\times$ average
  - LULESH: $61\times$
  - CoMD: $1.5\times$
  - CFD: $11\times$
  - It is run only once for a given input

- Accuracy levels: 3, 6, 9 digits
- Performance speedups levels: 5%, 10%, 15%, 20%
GPUMixer Results – Performance Speedup

<table>
<thead>
<tr>
<th>Error Thold. (digits)</th>
<th>Mode 1 Performance Threshold</th>
<th>Mode 2 Performance Threshold</th>
<th>Mode 3 Performance Threshold</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>5%</td>
<td>10%</td>
<td>15%</td>
</tr>
<tr>
<td>LULESH 3</td>
<td>9.8% (1)</td>
<td>9.8% (1)</td>
<td>30.4% (2)</td>
</tr>
<tr>
<td>LULESH 6</td>
<td>0.3% (12)</td>
<td>8.4% (79)</td>
<td>-</td>
</tr>
<tr>
<td>LULESH 9</td>
<td>0.3% (12)</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>CoMD 3</td>
<td>24.2% (1)</td>
<td>24.2% (1)</td>
<td>24.2% (1)</td>
</tr>
<tr>
<td>CoMD 6</td>
<td>24.2% (1)</td>
<td>24.2% (1)</td>
<td>24.2% (1)</td>
</tr>
<tr>
<td>CoMD 9</td>
<td>2.3% (3)</td>
<td>19.7% (62)</td>
<td>19.7% (62)</td>
</tr>
<tr>
<td>CFD 3</td>
<td>8.3% (1)</td>
<td>8.3% (1)</td>
<td>13.3% (3)</td>
</tr>
<tr>
<td>CFD 6</td>
<td>8.34% (1)</td>
<td>8.3% (1)</td>
<td>13.3% (3)</td>
</tr>
<tr>
<td>CFD 9</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

- We can find good configurations only in a few runs (1-3 runs)
- Mode 1 takes only a few runs (but can’t find very good cases)
- Mode 2 finds configurations with better performance than mode 1
- Highest performance improvements are found with mode 3
Precimonious-GPU Results

### Table 4: Results of using FISets and shadow computations: performance speedup (% of maximum ideal speedup) for three error thresholds, four performance thresholds and three modes of operation; number of runs in parenthesis.

<table>
<thead>
<tr>
<th>Error Mode</th>
<th>Performance Threshold</th>
<th>Thold. (digits)</th>
<th>5%</th>
<th>10%</th>
<th>15%</th>
<th>20%</th>
</tr>
</thead>
<tbody>
<tr>
<td>LULESH</td>
<td></td>
<td>3</td>
<td>11.6% (11)</td>
<td>11.4% (11)</td>
<td>17.4% (32)</td>
<td>20.7% (34)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6</td>
<td>11.5% (11)</td>
<td>11.4 (11)</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td></td>
<td>9</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>CoMD</td>
<td></td>
<td>3</td>
<td>12.6% (2)</td>
<td>12.9% (2)</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6</td>
<td>13.6% (2)</td>
<td>12.7% (2)</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td></td>
<td>9</td>
<td>5.4% (24)</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>CFD</td>
<td></td>
<td>3</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td></td>
<td>9</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>

### Table 5: Precimonious-GPU results: performance speedup (% of maximum ideal speedup) for the error thresholds and performance thresholds; number of runs are in parenthesis. See Fig. 4 for the maximum speedup reported for each approach.

<table>
<thead>
<tr>
<th>Error Thold. (digits)</th>
<th>Performance Threshold</th>
</tr>
</thead>
<tbody>
<tr>
<td>LULESH</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>11.6% (11)</td>
</tr>
<tr>
<td></td>
<td>11.4% (11)</td>
</tr>
<tr>
<td></td>
<td>17.4% (32)</td>
</tr>
<tr>
<td></td>
<td>20.7% (34)</td>
</tr>
<tr>
<td>CoMD</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>12.6% (2)</td>
</tr>
<tr>
<td></td>
<td>12.9% (2)</td>
</tr>
<tr>
<td></td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>–</td>
</tr>
<tr>
<td>CFD</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>–</td>
</tr>
</tbody>
</table>

The use of Precimonious-GPU results showed:

- Useful in finding configurations quickly
- It didn’t find configurations with higher speedup than GPUMixer
Comparison of Maximum Performance Speedup Between GPUMixer and Precimonious-GPU

- GPUMixer can find configurations with up to 46% of ideal speedup
- Versus 20.7% for Precimonious-GPU
- Two FISet configurations
  - FISet 1, low $r_{ac} = 2.08$
  - FISet 2, high $r_{ac} = 6.90$

- Speedups stabilize for large inputs

- FISet 1 has higher digits of accuracy
  - FISet 1 has fewer FP32 operations
In Summary

1. Automatic mixed-precision tuning can improve performance in GPU applications.

2. We present the first framework for automatic FP tuning in GPUs; we focus on performance improvements.

3. GPUMixer gets performance improvements of up to 46% of ideal speedup (20% only in state-of-the-art).

4. FISets can be found via static analysis; can be implemented in a compiler.

Thank you for the nomination!