Dataflow-Driven GPU Performance Projection For Multi-Kernel Transformations

Jiayuan Meng
Vitali A. Morozov
Venkatram Vishwanath
Kalyan Kumaran

Argonne National Laboratory
Application Developers

Hardware Designers

Leadership Computing Facility, Argonne National Laboratory, SC 2012, Jiayuan Meng, jmeng@alcf.anl.gov
Application Developers

• What HW to choose?
• What’s the benefit?
• How to get there?

Hardware Designers

• What app to test/inspire?
• What behavior is typical?
Goal

Find out how would an application adapt to a hardware

- Without acquiring actual hardware
- Without actual execution or tedious tuning

Leadership Computing Facility, Argonne National Laboratory, SC 2012, Jiayuan Meng, jmeng@alcf.anl.gov
Approach

Model both the hardware and software

Allow them to interact with each other to project the best performance
A Case Study on GPU

- A sequence of legacy parallel for loops
  - Goal: To project
    - Optimized GPU performance
    - Transformations needed to achieve the performance

- Assumption:
  - Data is already on the GPU

- Transformations
  - Intra-kernel
  - Inter-kernel
GPU Performance Factors

- **Good for compute-intensive workloads**
- **Challenging for data-intensive workloads**
  - Data movement is key in GPU optimization
    - Reuse
    - Locality
- **Dataflow is key**
  - Data access + Control flow + Dependency analysis
Kernel Fusion
Steps

- **Understand** the application
- **Transform** the application
- **Project** the performance of transformations
  - pick the best
Understand the application using code skeletons

- A code skeleton includes:
  - Control structure
  - Data accesses
  - Computation intensity
  - Domain knowledge
  - Input

- Notes:
  - Written by user
  - One-time effort

```plaintext
forall n = 1:50  
  read C[n]  
  for m = 1:50  
    write A[n][m]  
  end

forall k = 1:50  
  read C[k]  
  read A[k][k]  
  write A[k][k]  
  write C[k]
```
Transform an Application

- **Build**
  - a dependency graph among kernels
- **Fuse dependent kernels**
- **Optimize data movement**

```
for n = 1:100
  ...
for m = 1:200
  read A[n][m]
  write B[n][C[m]+4]
```

Data Analysis using Bounded Regular Sections

```
A[n<1:100>][m<1:200>] (read)
B[n<1:100>][C[m<1:200>]] (write)
```
The Challenges

1. Loops may be unaligned
   - different # of indices
   - different loop domains

2. Dataflow may hide in Hierarchies
   - data may be produced and consumed within separate code blocks
Fusing Loop Partitions

- Find dependency among individual iterations
- Backpropagate consumer-producer partitions
- Assemble the loop partition for the fused kernel
Hierarchical Dataflow Analysis/Transformations

- **Challenges**
  - Multiple, imperfectly nested loops
  - Questions: What data to cache & When
  - Constraints: L1 storage size

\[
\text{threads } n = 1:50 \{ \\
\quad \text{read } C[n] \\
\quad \text{for } m = 1:50 \\
\quad \quad \text{write } A[n][m] \\
\quad \text{// 2\textsuperscript{nd} knl} \\
\quad \text{read } C[n] \\
\quad \text{read } A[n][n] \\
\quad \text{write } A[n][n] \\
\quad \quad \text{write } C[n] \\
\}\]

Leadership Computing Facility, Argonne National Laboratory, SC 2012, Jiayuan Meng, jmeng@alcf.anl.gov
Transform a kernel: Hierarchical Dataflow Analysis

- For every block
  - Calculate dataflows at each level
  - Decide WHAT to cache and WHERE

threads $n = 1:50$

```plaintext
read C[n]
for m = 1:50
  write A[n][m]
// 2^{nd} knl
read C[n]
read A[n][n]
write A[n][n]
write C[n]
```

Leadership Computing Facility, Argonne National Laboratory, SC 2012, Jiayuan Meng, jmeng@alcf.anl.gov
Transform a kernel: Cache Footprint

- Record data lifespan at outermost level, emulate allocation and deallocation
  - Alloc: +     Dealloc: -

- Accumulate, find the Peak cache usage
Project Performance

Input:
- Threads per block
- Active blocks
  - No. of blocks
  - No. of warps
  - Cache usage
- No. of Memory instructions
  - Coalesced
  - Non-coalesced
- No. of computation instructions

GPU Performance Model
(S. Hong and H. Kim, 2009)

Output:
- Performance bottleneck
- Projected performance
Experiment Setup

- **Benchmarks**
  - SRAD
    - Multi-array dependency
  - HotSpot
    - Repeated kernels
  - CFD
    - Multiple dependent kernels
  - Stassuij (from GFMC)
    - Different loops shapes
    - Indirect accesses

- **Graphics Processors**
  - FX5600
    - 16 Stream Multiprocessors
    - Mem. Bandwidth: 76.8 GB/s
  - C1060
    - 30 Stream Multiprocessors
    - Mem. Bandwidth: 104.2 GB/s
    - Improved coalescing
**Results (1)**

**SRAD: Spectral Removal Anisotropic Diffusion**

- Dependency caused by multiple arrays

```c
1 R = 4096, C = 4096
2 float a[R][C], b[R][C]
3 float c[R][C], d[4][R][C]
4 forall i=0:R, j=0:C
5 {
6    read a[i-1][j]
7    read a[i+1][j]
8    read a[i][j-1]
9    read a[i][j+1]
10   for k=0:4
11      write d[k][i][j]
12      write c[i][j]
13 }
14 forall i=0:R, j=0:C
15 {
16    read c[i+1][j]
17    read c[i][j+1]
18    read a[i][j]
19   for k=0:4
20      read d[k][i][j]
21   write b[i][j]
22 }
```
Results (2)
HotSpot: Iterative kernel

- Iterative stencil operations
- Double buffering

```c
R = 1024, C = 1024, N = 120
2 float a[R][C], b[R][C], pow[R][C]
/* double buffering */
4 src=a; dst=b
/* Iterate over a kernel */
6 for n=0:N
7 {
8     forall r=0:R, c=0:C
9         /* Need data from several 
10          * producer iterations 
11          */
12         read src[r][c]
13         read src[r][c+1]
14         read src[r][c-1]
15         read src[r-1][c]
16         read src[r+1][c]
17         read pow[r][c]
18         write dst[r][c]
19     }
20    swap=src; src=dst; dst=swap
22 }
```
Results (3)

CFD: Computational Fluid Dynamics

- Multiple dependent kernels
- Nested loops

```c
for all i=0:NELR
  for k=1:4
    /* Compound accesses w/t multiple loop indices */
    read src[i+k*NELR]
    write step_factors[i]
  }
for all i=0:NELR
  for j=0:NNR
    /* Indirect accesses */
    nb = neighbors[i + j*NELR]
    for k=0:NVAR
      read src[nb + k*NELR]
    for k=0:NVAR
      write fluxes[i + k*NELR]
  }
/* Depend on two producer loops */
for all i = 0:NELR
  read step_factors[i]
  for k = 0:NVAR
    read fluxes[i + k*NELR]
    write dst[i + k*NELR]
  }
```
Results (4):
Stassuij: Sparse Linear Algebra + Spectral Methods

- Indirect accesses
- Different loop shapes

```plaintext
1 NT = 132
2 NS = 2048
3 NSG = 512
4 ELEMS = 1848
5 avg_j_ntdt = 14
6 int J[NT+1], I[ELEMS], T[ELEMS]
7 float A[NT][NS][2]
8 float B[NT][NS][2]
9 float C[NT][NS][2]
10 for all j=0:NT, i=0:NS, r=0:2
11 { read J[j]
12   read J[j+1]
13   stream n = 0:avg_j_ntdt
14     read T[n]
15     read I[n]
16     read A[I[n]][i][r]
17   write B[j][i][r]
18 }
19 /* Loop with a different shape */
20 for all j=0:NT, ig=0:NSG, m=0:4, r=0:2
21 { read M[ig][m]
22   /* Indirect accesses */
23   read A[j][M[ig][m]][r]
24   read B[j][M[ig][m]][r]
25   write C[j][M[ig][m]][r]
26 }
```
Conclusions and Future Work

- An analytical application model can
  - Express potential performance behavior
  - Explore transformations

- Enable the App. model to interact with the HW model

Technical Contributions
- Model-based kernel fusion
- Hierarchical data flow analysis

Future Work
- Explore more kernels and applications
- Adopting more recent hardware models
Acknowledgements

- Steven Pieper
- Gail Pieper
- Anonymous Reviewers

- Sponsors:
  - The U.S. Department of Energy (contract DE-AC02-06CH11357).
  - Leadership Computing Facility at Argonne National Laboratory
  - Office of Science of the U.S. Department of Energy (contract DE-AC02-06CH11357)
Thank You!

Vitali A. Morozov

Venkatram Vishwanath

Kalyan Kumaran

GROPHECY (soon to be released):
www.alcf.anl.gov/perfengr/grophecy