Declarative Framework for GPUs: DEF-G

Dissertation Proposal

Robert Senser
August 15, 2013

PhD Comprehensive Exam Committee
Tom Altman (advisor)
Gita Alaghband
Boris Stilman
Presentation Outline

• Motivation for Work
• Background: Graphical Processing Units (GPUs) and OpenCL
• Declarative Framework for GPUs: DEF-G
• Diverse GPU Applications using DEF-G
  – Sorting Roughly Sorted Data
  – Iterative Matrix Inversion
  – Breadth-First Search
  – Sobel Image Filter
Presentation Outline

• Proposed Dissertation Work Plan
  – Construct DEF-G Version 2
  – Apply DEF-G to Diverse GPU Applications
  – Describe the DEF-G Design Patterns
  – DEF-G as a Research Platform
    • Make wider GPU use of declarative approach?
    • Articulate the limits to this GPU development approach
    • Expose other approaches
Motivation for Work

• GPUs can provide high throughput
  – Radeon HD 7990: 2 (double-precision) TFLOPS
• Developing parallel HPC software is difficult
• Parallel development for GPUs is even more difficult
• GPU HPC software development requires:
  – Understanding of unique GPU hardware characteristics
  – Use of specialized algorithms
  – Use of GPU-specific, low-level APIs
    • OpenCL
    • CUDA
Background: GPUs and OpenCL

• Graphical Processing Unit (GPU)
  – Highly specialized coprocessor
  – Hundreds of cores
  – Thousands of hardware-managed threads
  – SIMT: *Single Instruction, Multiple Thread*
    • Variant of the common Single Instruction, Multiple Data (SIMD) model
    • Threads not on the execution path pause
  – Code executed in a “kernel”

• Common GPU programming environments
  – OpenCL, which is Open Source
  – CUDA, which is NVIDIA proprietary

• DEF-G is designed for OpenCL [2]
High-Level GPU Architecture

CPU

Cache

RAM

Virtual Memory

GPU

Cache?

Local

Global RAM

PCIe bus

GPU Characteristic:

- Processors often connected by Peripheral Component Interconnect Express (PCIe) bus
- Usually has own fast Global RAM
- Threads have some very fast local memory
- May/may not have a cache
- Usually many hardware-controlled threads
- Lacks CPU-style predictive branching, etc.
OpenCL Overview

• Specification provided by Khronos Group
• Open Source, multi-vendor
• Hardware device support
  – GPUs
  – CPUs
  – Digital signal processors (DSPs)
  – Field-programmable gate arrays (FPGAs)
• Device kernel normally written in C
• CPU-side code
  – C/C++
  – Third-party bindings for Java, Python, ...
GPU Applications

• Three components
  – Application algorithms
  – CPU-side code
    • Moves kernel code to GPU
    • Manages GPU execution and errors
    • Moves application data between CPU and GPU
    • May contain a portion of application algorithms
  – GPU kernel code
    • Can have multiple kernels per application
    • Each kernel usually contains an algorithm or algorithm step
    • Kernel code often uses GPU-specific techniques
GPU Performance

• Some of the Issues with GPU Performance
  – Kernel Instruction Path Divergence
    • Occurs with conditional instructions (ifs, loops, etc.)
    • Causes some threads to pause
  – High Memory Latency
    • Each RAM access can consume time of 200-500 instructions
    • Accesses to RAM should be coalesced
    • Bank conflicts can occur with local thread memory

• Rob Farber GPU suggestions [1]:
  – “Get the data on the GPU and leave it”
  – “Give the GPU enough work to do”
  – “Focus on data reuse to avoid memory limits”
Dissertation Contribution

• Declarative Framework for GPUs (DEF-G)
  – Simplifies construction of the CPU portion of the GPU application
  – Required CPU actions and resources are declared
  – DEF-G generates the needed CPU code
  – DEF-G provides a research platform

• DEF-G Applicability
  – Four diverse problem areas to show universality
    • Sorting Roughly Sorted Data
    • Iterative Matrix Inversion
    • Breadth-First Search with Very Large Irregular Graphs
    • Sobel Image Filter

• CPU-side Design Patterns

• Delineation of DEF-G Limits

• Insights into innovative new approaches
Existing Proof-of-Concept DEF-G Version [2]

• The DEF-G framework generates the CPU code
  – Input: declarative statements
  – Uses design patterns
  – Output: OpenCL code

• Proof-of-Concept DEF-G Compiler
  – ANTLR-Based Parser
  – Intermediate XML Document
  – TinyXML2 Parser
  – Code Generator written in C++
Actual DEF-G Code Sample

01. declare application sobel
02. declare integer Xdim (0)
03. declare integer Ydim (0)
04. declare integer BUF_SIZE (0)
05. declare gpu gpuone ( any )
06. declare kernel sobel_filter SobelFilter_Kernels ( [[ 2D,Xdim,Ydim ]] )
07. declare integer buffer image1 ( $BUF_SIZE )
08. integer buffer image2 ( $BUF_SIZE )
09. call init_input (image1(in) $Xdim (out) $Ydim (out) $BUF_SIZE(out))
10. execute run1 sobel_filter ( image1(in) image2(out) )
11. call disp_output (image2(in) $Xdim (in) $Ydim (in) )
12. end

... status = clSetKernelArg(sobel_filter, 1, sizeof(cl_mem), (void *)&buffer_image2);
  if (status != CL_SUCCESS) { handle error }
  // *** execution
  size_t global_work_size[2]; global_work_size[0] = Xdim ; global_work_size[1] = Ydim ;
  status = clEnqueueNDRangeKernel(commandQueue, sobel_filter, 2, NULL, global_work_size, NULL, 0,
                                  NULL, NULL);
  if (status != CL_SUCCESS) { handle error }
  // *** result buffers
  status = clEnqueueReadBuffer(commandQueue, buffer_image2, CL_TRUE, 0, BUF_SIZE * sizeof(int), image2, 0,
                                NULL, NULL);
...
Analysis of Proof-of-Concept DEF-G

• Presented Proof-of-Concept DEF-G Analysis
  – Conference: Parallel and Distributed Processing Techniques and Applications (PDPTA’13) [2]
  – DEF-G was well received

• Analysis
  – Three available OpenCL applications converted to DEF-G
    • CPU-side recoded in DEF-G and used existing GPU kernels
    • Applications:
      – Breadth-First Search (BFS)
      – All-Pairs Shortest Path (APSP/FW)
      – Sobel Image Filter (SOBEL)
    • Verified that results matched
  – Comparisons between “reference” and DEF-G
    • Lines-of-Code Comparison
    • Run-time Performance Comparison
### Lines-of-Code Comparison

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>DEF-G Source</th>
<th>DEF-G Gen.</th>
<th>REF.</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOBEL</td>
<td>12</td>
<td>208</td>
<td>442</td>
</tr>
<tr>
<td>BFS</td>
<td>33</td>
<td>291</td>
<td>364</td>
</tr>
<tr>
<td>FW</td>
<td>12</td>
<td>238</td>
<td>478</td>
</tr>
</tbody>
</table>

On average, the DEF-G code is 7.7 percent of the generated code.
• Shown are original run times (average of 10 runs)
• Later, made manual changes to understand timing differences
• FW reference version was slow due to a *vendor* coding error (which I fixed)
• BFS showed need for buffer optimization in DEF-G
• Likely CPU-based BFS-4096 was fast due to CPU’s cache
• Summary: DEF-G provides comparable performance
DEF-G Version 2

• DEF-G Benefits and Features
  – Provides CPU-side OpenCL functionality with fewer lines of code
  – Will be a production-quality tool
  – Will include support for multiple GPU cards
  – Will include support for Anytime algorithms
  – Will provide a research platform
    • Identify CPU-side design patterns
    • Help identify new approaches to GPU software development
      – How can the GPU-side also benefit from a declarative approach?
      – How can this approach be applied to other application domains?
  – Full list of features given later under Work Plan
Diversity of DEF-G Applications

• Construction of four very diverse GPU applications
  – *Sorting*: Sorting Roughly Sorted Data
    • Construction of novel sorting approach
    • Will use parallel prefix calculations in sorting optimization
  – *Numerical*: Iterative Matrix Inversion using Mieczyslaw Altman’s Method
    • Application of *Anytime* algorithms and OpenCL APPML BLAS (Basic Linear Algebra Subprograms)
    • When a measure is met, *Anytime* algorithm stops the process
  – *Graph Theoretic*: Breadth-First Search with Very Large Irregular (VLI) graphs
    • Novel use of prefix sum to avoid GPU data locking
    • Showcase for multiple GPU card option
  – *Image Processing*: Sobel Image Filter
    • Showcase for multiple GPU card option

• These four applications will show the DEF-G applicability
Sorting Roughly Sorted Data

• Goal: Improve on $O(n \log n)$ sorting bound when sequence is partially sorted
• Based on the prior sorting work by T. Altman et al. [3]
• $k$ is a measure of “sortedness”
• A sequence is $k$-sorted if no element is more than $k$ positions out of sequence
• Knowing $k$ allows for sorts of $O(n \log k)$
• If $k$ is small then we get a substantial improvement
• The $k$-sorted trait can be GPU exploited
  – Prefix sum in calculating $k$
  – Parallel sorts of sub-sequences
Status of Roughly Sorting

• Developed a Roughly Sorting serial version
• Reviewed existing GPU sorts
  – Requires a sorting capability for sub-sequences
  – Considering use of radix sort from AMD SDK
  – Prefer a comparison-based sort due to merging, when using divide, process, merge pattern
• Next steps:
  – Choose an existing OpenCL sort for use
  – Add support for multiple GPU cards using DEF-G’s divide, process, merge pattern
  – Performance analysis of GPU Roughly Sorting
Iterative Matrix Inversion (IMI)

• DEF-G iterative matrix inversion application using M. Altman’s method [4]

• Use the Anytime-algorithm approach to manage the iterative inversion
  – Inversion is stoppable at “anytime”
  – Can balance run time against accuracy
  – Anytime management in DEF-G, not the application

• Requires GPU matrix operations
  – Expect to use OpenCL APPML
  – APPML integration into DEF-G is a big “plus”
Basic M. Altman IMI Approach

The initial inverse approximation, that is $R_0$, can be formed by:

$$R_0 = \alpha I$$

where $\alpha = 1 / \| A \|$

$\| A \|$ being the Euclidean norm of $A$

and $I$ is the identity matrix.

To invert matrix $A$, each iteration calculates:

$$R_{n+1} = R_n (3I - 3AR_n + (AR_n)^2)$$

with the result in $R_{n+1}$.

- Better $R_0$ estimate provides for quicker convergence
- Method is self correcting
- DEF-G Anytime facility stops the iterations
  - When inversion quality measure is met
  - When max iterations have occurred
  - When max run time has occurred
Status of IMI

• Developed a C++ IMI version using Boost BLAS
• Further developed a proof-of-concept GPU version using CUDA’s cuBLAS
  – cuBLAS is a CUDA-specific BLAS facility
  – Suspended due to DEF-G’s OpenCL focus
• Use multiple levels of parallelism
  – Start multiple inversions with different $\alpha$ values
  – After a set criteria is met, finish only inversion instance with best intermediate results
Status of IMI

• Next steps:
  – Addition of Anytime support to DEF-G
  – IMI application using DEF-G and OpenCL APPML
  – Analysis of application
    • Use *University of Florida Sparse Matrix Collection* [7]
    • Run-time performance
    • Inversion accuracy
    • Application usability
      – Maximum matrix size
      – Ability to handle sparse matrices
Breadth First Search (BFS)

• Well-studied graph-theoretic problem
• Focus: BFS with Very Large Irregular (VLI) Graphs
  – Social Networking, Routing, Citations, etc.
• Many published GPU BFS approaches, starting with Harish [8]

- Harish used “Dijkstra” BFS
- Vertex frontier as a Boolean array
  • 1 = vertex on frontier
- A GPU thread assigned to each vertex
- Resulted in very poor thread utilization
Status of BFS

• Identified VLI Graph Repositories:
  – Stanford Network Analysis Package (SNAP) [5]
  – Center for Discrete Mathematic and Theoretical Computer Science [6]
  – University of Florida Sparse Matrix Collection [7]

• Developed proof-of-concept DEF-G version

• Performance experiments performed
  – Explored GPU BFS Vertex Reordering
    • Grouped work-items into work-groups by vertex degree
    • Good results with test graphs, poor with SNAP graphs
  – Explored GPU BFS Frontier Adaptation
    • Dynamic switching between frontier array and queue
    • Multi-threaded queue requires locking
    • GPU locking tends to be expensive (slow)
    • Showed promise with certain graphs and GPU cards
BFS Vertex Frontier

• Merrill approach to vertex buffer management [9]
  – Have a queue or buffer with multiple update threads
  – Uses prefix sum to allocate cells

• Generalize this buffer management in DEF-G
  – Likely provided as a library of kernel functions

• Useful for shared buffers with multiple GPU cards
Status of BFS

• Next steps:
  – DEF-G generalization of Merrill approach
    • Prefix-scan based buffer allocation
    • Use of virtual pointers (between GPU cards)
  – Enhancement of DEF-G BFS application
  – Test application against LVI graphs
    • Test graphs from SNAP and DIMACS repositories
    • Very large graph datasets: millions of vertices and edges
  – Analysis of BFS application
    • Characteristics and capabilities
      – Maximum graph sizes
    • Run-time performance
      – Single-card versus multi-card performance
      – Performance relative to well-known BFS implementations
Sobel Image Filter

- Sobel operator detects edges in images
- Pixel gradient calculated from 3x3 mask
- Uses a single GPU kernel, called once
- Base-line test application for multiple GPUs
- DEF-G processing:
Status of Sobel Image Filter

• Developed proof-of-concept DEF-G version
• Created a proof-of-concept, hand-written, multiple GPU version
  – Using two cards doubled throughput
  – Complexity in managing overlapped sub-images
• Next steps:
  – DEF-G multiple GPU version
    • Due to 3x3 mask, sub-images overlap
    • Uses *overlapped split, process, concatenate* design pattern
  – Testing and analysis with large images and more than one GPU
  – Expect to find (or emulate) a four-GPU environment
Proposed Dissertation Work Plan

• DEF-G Enhancements
  – Three existing design patterns
    • Execute kernel *once*, execute *N* times, and *loop-while*
    • The current *loop-while* syntax is too “procedural”
  – New design patterns
    • Anytime algorithm support
    • Multiple GPU Support
      – *divide*, *process*, *merge*
      – *overlapped split*, *process*, *concatenate*
    • Explicit Parallel
  – Add other interesting DEF-G design patterns
Proposed Dissertation Work Plan

• More DEF-G Enhancements
  – Extend compiler and add buffer optimization
  – Add high-performance data loaders, result displays, and debugging aids (logging)
  – Introduce additional run modes and function calling options
  – Implement run-time support for multiple GPU card operation and Anytime algorithm support
Proposed Dissertation Work Plan

• New design patterns
  – *Divide, process, merge*
    • Used with Rough Sorting
    • The input is separated, processed, and merged
  – *Overlapped split, process, concatenate*
    • Used with Sobel Image Filter
    • The input is split with overlaps, processed, and reconnected
  – Anytime algorithm
    • Used with Iterative Matrix Inversion
    • Use of well-defined quality measures to monitor progress
    • Can terminate processing early (quality versus run time)

• Some design patterns can be combined
  – Example: Execute $N$ times + *divide, process, merge* + Anytime
Proposed Dissertation Work Plan

• Roughly Sorting
  – Choose an existing OpenCL sort
  – Support for multiple GPU cards using *divide, process, concatenate* pattern
  – Performance analysis of GPU Roughly Sorting

• Iterative Matrix Inversion
  – Addition of Anytime support to DEF-G
  – Develop IMI application using DEF-G and APPML
  – Analysis of application
    • Run-time performance
    • Inversion accuracy
    • Application usability
    • Use *University of Florida Sparse Matrix Collection* for testing
Proposed Dissertation Work Plan

• Breadth-First Search Summary
  – DEF-G generalization of Merrill approach
    • Prefix-scan based buffer allocation
  – Analysis of BFS application using LVI graphs
    • Characteristics and capabilities
    • Run-time performance

• Sobel Filter Summary
  – DEF-G Multiple GPU version
    • Due to 3x3 mask, sub-images overlap
    • Use *overlapped split, process, concatenate* design pattern
  – Analysis with large images and multiple GPUs
Proposed Dissertation Work Plan

• Describe the DEF-G capabilities
  – Identify domains of appropriate use
  – Establish the limits of DEF-G use
  – Strive for a rigorous theoretical basis

• Use DEF-G as a research platform
  – Quickly create GPU application
  – Enable wider GPU use of declarative approach?
  – Articulate the limits to this GPU development approach
  – Help introduce other, perhaps better, approaches
Proposal Summary

• DEF-G
  – Inputs declarations
  – Uses declared design patterns
  – Generates CPU-side OpenCL code
• Summarized the Proof-of-Concept DEF-G
• Described the Version 2 DEF-G enhancements
• Described the diverse DEF-G applications
  – These show the DEF-G applicability and flexibility
  – Each is a full application implementation
• Addressed DEF-G research goals
Additional Slides
Raw Performance Numbers

<table>
<thead>
<tr>
<th></th>
<th>CPU</th>
<th>GPU-GT 430</th>
<th>GPU-Tesla T20</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DEF-G</td>
<td>Ref.</td>
<td>DEF-G</td>
</tr>
<tr>
<td>BFS-4096</td>
<td>4.7</td>
<td>2.6</td>
<td>27.2</td>
</tr>
<tr>
<td>BFS-65536</td>
<td>40.9</td>
<td>14.2</td>
<td>143.9</td>
</tr>
<tr>
<td>FW</td>
<td>115.6</td>
<td>152.0</td>
<td>17.9</td>
</tr>
<tr>
<td>SOBEL</td>
<td>23.0</td>
<td>24.8</td>
<td>11.1</td>
</tr>
</tbody>
</table>
Sample DEF-G Code Showing a Sequence

01. declare application floydwarshall
02. declare integer NODE_CNT (0)
03. declare integer BUF_SIZE (0)
04. declare gpu gpuone ( any )
05. declare kernel floydWarshallPass FloydWarshall_Kernels ( [[ 2D,NODE_CNT ]])
06. declare integer buffer buffer1 ( $BUF_SIZE )
07. integer buffer buffer2 ( $BUF_SIZE )
08. call init_input (buffer1(in) buffer2(in) $NODE_CNT(out) $BUF_SIZE(out))
09. sequence $NODE_CNT times
10. execute run1 floydWarshallPass ( buffer1(inout) buffer2(out) $NODE_CNT(in) $CNT(in) )
11. call disp_output (buffer1(in) buffer2(in) $NODE_CNT(in))
12. end
Sample DEF-G Code Showing a Loop-While

declare application  bfs
declare integer NODE_CNT (0)
declare integer EDGE_CNT (0)
declare integer STOP (0)
declare gpu gpuone ( any )
declare kernel kernel1 bfs_kernel ( [[ 1D,NODE_CNT ]] ) 
  kernel kernel2 bfs_kernel ( [[ 1D,NODE_CNT ]] )
declare struct (4) buffer graph_nodes ( $NODE_CNT ) 
  integer buffer graph_edges ( $EDGE_CNT ) 
  integer buffer graph_mask ( $NODE_CNT ) 
  integer buffer updating_graph_mask ( $NODE_CNT ) 
  integer buffer graph_visited ( $NODE_CNT ) 
  integer buffer cost ( $NODE_CNT )
// note: init_input handles setting "source" node
call init_input (graph_nodes(out) graph_edges(out) graph_mask(out) updating_graph_mask(out) graph_visited (out) cost (out) 
  $NODE_CNT(out) $EDGE_CNT(out))
loop
  execute part1 kernel1 ( graph_nodes(in) 
    graph_edges(in) 
    graph_mask(in) 
    updating_graph_mask(out) 
    graph_visited(in) 
    cost(inout) 
    $NODE_CNT(in) )
  // set STOP to zero each time thru...
  set STOP (0)
  // note: STOP value is returned...
  execute part2 kernel2 ( graph_mask(inout) 
    updating_graph_mask(inout) 
    graph_visited(inout) 
    $STOP(inout) 
    $NODE_CNT(in) )
  while $STOP eq 1
    call disp_output (cost(in) $NODE_CNT(in))
end
References


