Towards Interactive Visual Exploration of Parallel Programs using a Domain-Specific Language

Tobias Klein
KAUST, TU Wien
tobias.da.klein@gmail.com

Stefan Bruckner
University of Bergen
stefan.bruckner@uib.no

M. Eduard Gröller
TU Wien
groeller@cg.tuwien.ac.at

Markus Hadwiger
KAUST
markus.hadwiger@-
kaust.edu.sa

Peter Rautek
KAUST
peter.rautek@-
kaust.edu.sa

ABSTRACT

The use of GPUs and the massively parallel computing paradigm have become wide-spread. We describe a framework for the interactive visualization and visual analysis of the run-time behavior of massively parallel programs, especially OpenCL kernels. This facilitates understanding a program's function and structure, finding the causes of possible slowdowns, locating program bugs, and interactively exploring and visually comparing different code variants in order to improve performance and correctness. Our approach enables very specific, user-centered analysis, both in terms of the recording of the run-time behavior and the visualization itself. Instead of having to manually write instrumented code to record data, simple code annotations tell the source-to-source compiler which code instrumentation to generate automatically. The visualization part of our framework then enables the interactive analysis of kernel run-time behavior in a way that can be very specific to a particular problem or optimization goal, such as analyzing the causes of memory bank conflicts or understanding an entire parallel algorithm.

1. INTRODUCTION

Debugging tools like NVIDIA NSight [5], Visual Profiler [6] and VampirTrace [2] support debugging, profiling and analysis of parallel programs. These tools provide facts on application-level such as idle times of the processors, timings of memory transactions, as well as facts on the kernel level, such as warp divergence, memory bank conflicts, occupancy. These statistics indicate potential program bottlenecks. Essentially, the above mentioned tools show "*What* is going wrong". In contrast, the aim of our research project [3, 4] is to allow programmers to test their hypotheses on "*Where* and *Why* is something going wrong".

2. APPROACH

We have developed a system that conceptually consists of three major components:

Domain Specific Language: A domain specific language (DSL) that extends OpenCL with additional keywords forms an integral part of the system. Our DSL provides concise annotations of kernel code that trigger the generation of additional data during kernel execution. This meta data is read back to the host and prepared for interactive visualization by the runtime component.

Compiler and Runtime Component: A source-to-source compiler takes a program written in our DSL as input and generates instrumented OpenCL code. The runtime component triggers the just-in-time compilation of the code, binds the buffers and sets the kernel arguments that are necessary to run the kernel.

Visualization Framework: The resulting data that was generated during the execution of the kernel is visualized using a powerful framework capable of static as well as interactive visualizations. By linking the visualizations with source code, we provide insight into the program's structure, execution, and memory access patterns.

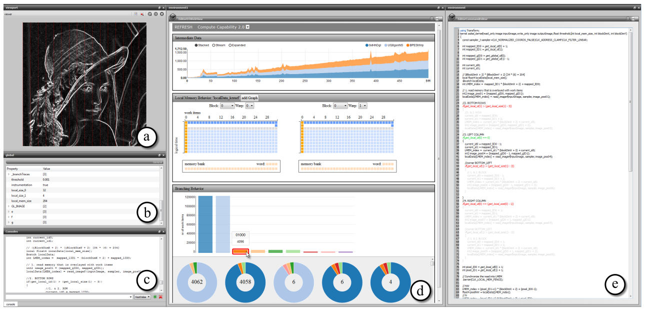
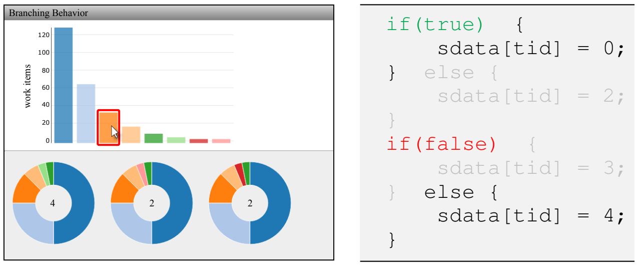


Figure 1: The interface of our integrated system: (a) shows a domain view that depicts the output of a Sobel filter and highlights corresponding to the hovered visual element, (b) shows a global variable view, (c) shows a console, (d) shows the visual explorer with different analysis views, (e) shows the source code editor with highlights that also corresponding to the hovered visual element.

By combining these three components into one integrated visual exploration system, we enable fine-grained and specialized analysis. Due to the interactive nature of our system a programmer can quickly iterate over a large number

IWOCL '16 April 19–21, 2016, Vienna, Austria

ACM ISBN 978-1-4503-4338-1/16/04.
DOI: http://dx.doi.org/10.1145/2909437.2909459



```
if(true)  {
    sdata[tid] = 0;
}  else {
    sdata[tid] = 2;
}
if(false)  {
    sdata[tid] = 3;
}  else {
    sdata[tid] = 4;
}
```

(a) *(b)*

Figure 2: The code path view (a) shows an overview different code paths that occur during the execution of the program and reveals warp divergence. The code snippet (b) illustrates the highlighting of code when the user hovers over a visual element (red rectangle) that represents one specific code path.

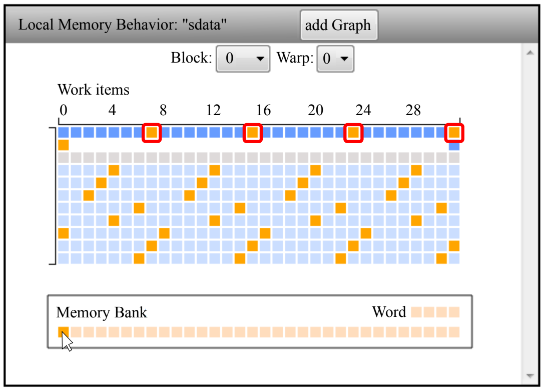


Figure 3: Screenshot of the local memory view, which depicts memory accesses and memory banks. Hovering over a memory bank reveals the corresponding memory accesses. Multiple accesses of the same memory bank (red rectangles) in one instruction of a warp indicate a potential bank conflict.

of source code variations. The immediate interactive visualization aids in developing a better understanding of the program's intrinsics.

Figure 1 depicts the graphical user interface of our integrated system. The domain view is shown in Figure 1 (a), which depicts, in this instance, the output of a Sobel filter. For convenience, our system also integrates a global variable view shown in Figure 1 (b) and a console shown in Figure 1 (c). An immediate visual response is shown in the visual exploration panel in Figure 1 (d). The editor shown in Figure 1 (e) is used for the implementation of kernels in our DSL, as well as for the generation of test input data and the computation of kernels. When the program is run, it automatically translates the DSL to OpenCL code, interprets the commands for the input generation, enqueues the kernel, waits for the result, and visualizes the additional data that was generated during the execution of the kernel.

The visualizations are generated using the D3 [1] visualization framework and facilitate interactive exploration capabilities, such as highlighting and linking. For instance, in Figure 1, a mouse event triggers the highlighting of a code path determined by a specific control flow, which is reflected with the highlighting in the source code and corresponding elements in the visual explorer.

3. VISUALIZATION VIEWS

Parallel programs are typically designed for high-throughput and performance. The complexity of the underlying hardware architecture directly impacts the complexity of the code. In many cases multiple implementations of the same algorithm compete for optimal performance and memory usage. Our visualizations aim to enhance the understanding of parallel programs, their competing implementations, and their possible impact on performance.

Control flow statements in parallel programs potentially lead to warp divergence, significantly affecting the performance. Figure 2 shows the visualizations of different code paths that occur during the execution of the program and the occurrence of warp divergence.

In order to visualize the memory behavior, our system traces the accesses of specified variables during the execution of a program. The tracing results in a sequence of events that are recorded for each thread of an execution. In the memory view (see Figure 3) memory accesses as well as synchronization barriers constitute events that are visualized with colored blocks. This view facilitates the comparison of different memory access patterns. Additionally, the depiction of related hardware components reveals the cause of common performance issues, such as memory bank conflicts.

4. CONCLUSION

We present a tool for the visual exploration of parallel programs executed on the GPU. Our integrated system features just-in-time compilation of kernels that collect additional data about their execution. This information is visualized using a powerful visualization framework. It provides the programmer with immediate feedback and therefore enables the quick exploration of a large number of variations of a kernel.

5. REFERENCES

[1] M. Bostock, V. Ogievetsky, and J. Heer. D3: Data-Driven Documents. *IEEE Trans. Visualization & Comp. Graphics (Proc. InfoVis)*, 2011.

[2] M. Jurenz, R. Brendel, A. Knüpfer, M. Müller, and W. E. Nagel. Memory allocation tracing with vampirtrace. In *Computational Science–ICCS 2007*, pages 839–846. Springer, 2007.

[3] T. Klein. Towards interactive visual exploration of parallel programs using a domain-specific language. Master's thesis, 2015.

[4] T. Klein. Towards interactive visual exploration of massively parallel programs using a domain-specific language. Talk at NVIDIA GPU Technology Conference (GTC), 2016.

[5] NVIDIA. Nsight. http://www.nvidia.com/object/nsight.html. Accessed: 07/04/2016.

[6] NVIDIA. Visual Profiler. https://developer.nvidia.com/nvidia-visual-profiler. Accessed: 07/04/2016.