POSTER – hVISC: A Portable Abstraction for Heterogeneous Parallel Systems

Prakalp Srivastava
University of Illinois at
Urbana-Champaign
psrivas2@illinois.edu

Rakesh Komuravelli
Qualcomm Technologies Inc.
rakesh.komuravelli
@qti.qualcomm.com

Maria Kotsifakou
University of Illinois at
Urbana-Champaign
kotsifa2@illinois.edu

Vikram Adve
University of Illinois at
Urbana-Champaign
vadve@illinois.edu

Matthew D. Sinclair
University of Illinois at
Urbana-Champaign
mdsincl2@illinois.edu

Sarita Adve
University of Illinois at
Urbana-Champaign
sadve@illinois.edu

ABSTRACT

Programming heterogeneous parallel systems can be extremely complex because a single system may include multiple different parallelism models, instruction sets, and memory hierarchies, and different systems use *different combinations* of these features. We propose a carefully designed parallel abstraction of heterogeneous hardware – a *hierarchical dataflow graph with shared memory and vector instructions* – that is able to capture the parallelism in a wide range of popular parallel hardware. We use this abstraction, which we call *hVISC*, to define a Virtual Instruction Set Architecture (ISA) that aims to address both functional portability and performance portability across heterogeneous systems. *hVISC* is more general than existing virtual instruction sets such as PTX, HSAIL and SPIR, e.g., it can capture both streaming parallelism and general dataflow parallelism.

1. MOTIVATION

Programming heterogeneous systems is extremely challenging because of the diversity of the underlying hardware components. *Developers* must be able to develop portable algorithms and reason about their performance and scalability; write efficient, yet portable source-level programs; and tune program performance for a range of different hardware.

At a fundamental level, these challenges arise from the diversity in hardware parallelism models, memory architectures, and hardware instruction sets. A promising approach to address these challenges is to develop a *parallel model that captures these diversities* and thus can be mapped down to a wide range of different parallel hardware. In particular, a suitable unified set of abstractions can enable parallel algorithm development, parallel language design, compiler optimizations and code generation for parallel programs, performance tuning, and both source and object code portability.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

PACT '16 September 11-15, 2016, Haifa, Israel

© 2016 Copyright held by the owner/author(s).

ACM ISBN 978-1-4503-4121-9/16/09.

DOI: <http://dx.doi.org/10.1145/2967938.2976039>

2. PARALLEL ABSTRACTION DESIGN

In this work, we propose a parallel abstraction we call *hVISC* that abstracts away differences between parallelism models in hardware. In *hVISC*, a program is represented as a hierarchical dataflow graph. The nodes represent computations and may use shared memory and vector instructions. The edges capture explicit data transfers between nodes, while ordinary load and store instructions can be used to perform implicit communication via shared memory.

For example, Figure 1 shows an *hVISC* example of a Laplacian estimate computation for a greyscale image. The computation is represented by three dataflow nodes – Dilation Filter, Erosion Filter and Linear Combination – connected by edges. The code for the Linear Combination filter is shown, and includes load and store instructions accessing shared memory as well as explicit vector instructions with a parametric vector length [6].

A single static dataflow node or edge in a graph may represent multiple *dynamic instances* of the node or edge. The instances of a node are required to be independent of each other, i.e., can be executed fully in parallel. This succinctly captures fine-grain data parallelism, which can be targeted to a range of parallel hardware efficiently. Moreover, dataflow graphs in *hVISC* can be hierarchical: each node in a graph may itself contain a full-fledged dataflow graph (called parent and child graph respectively). In Figure 1, Dilation Filter, Erosion Filter and Linear Combination are child graphs to parent Laplacian Estimate graph. Typically, the top-level graph captures parallelism and explicit data transfers across major compute units in a heterogeneous system, while lower-level dataflow graphs capture fine-grain parallelism within individual compute kernels, which can be mapped to a GPU, vector hardware or a multicore host processor. The graph hierarchy is also an effective way to capture tiling of computations, useful in exploiting cache hierarchy or GPU scratchpad memory.

Finally, a dataflow graph edge may be either a one-time data transfer edge or a “streaming” edge that executes repeatedly. Streaming edges enable pipelined parallelism on long input streams to be expressed naturally and translated effectively to a wide range of hardware. In Figure 1, the node Laplacian Estimate is one stage in an image processing pipeline and the dashed arrows for I , I_d , I_e and L show streaming edges representing streams of images.

Figure 1: Non-linear Laplacian computation in *hVISC*

In this work, we use *hVISC* to define a *virtual instruction set* (ISA) that is an extension of the LLVM virtual instruction set [4]. *hVISC* can be used to ship programs as “virtual object code,” and each DF node in a program can be translated to different hardware compute units either at install-time or run-time. This is similar to PTX [5], HSAIL [1] and SPIR [3]. However, *hVISC* provides more powerful and general abstractions for parallelism, such as (regular or irregular) dataflow graphs or pipelined, streaming codes (in addition to the SIMT and short-vector SIMD parallelism targeted by HSAIL or SPIR), which can be used to represent a much wider range of parallel programs.

3. COMPILER INFRASTRUCTURE

We have designed and implemented “back-end” code generators from *hVISC* to three different classes of parallel hardware: GPUs (using NVIDIA’s PTX), vector SIMD (Intel’s AVX), and multicore CPUs (using Posix threads). We believe *hVISC* can also support other kinds of hardware well, e.g., FPGAs, but code generation for such targets is left to future work. We have also implemented an efficient run-time system that supports data transfers between compute units, buffering for streaming edges, and optimizations to avoid redundant transfers by tracking the “last location” for data objects identified via shared memory pointers.

4. EVALUATION

Figure 2 shows that *hVISC* achieves performance close to or on par with *hand-coded* performance on both GPU and vector platforms. We use the *same hVISC* code for seven applications from the Parboil [7] benchmark suite, and compare them against the best performing OpenCL implementation of each one in the Parboil suite *on its respective target*, compiled using nVidia’s proprietary OpenCL compiler or the Intel OpenCL compiler [2] respectively. *hVISC* achieves performance near that of hand-tuned OpenCL for all benchmarks except bfs, where the overhead is 20% due to lack of support for global barriers. In the vector case, the performance of *hVISC* is within 7% in the worst case.

Figure 3 shows that *hVISC* can (a) capture pipelined, streaming computations effectively; and (b) is flexible enough to allow a wide range of pipeline mapping configurations with different performance characteristics from a

Figure 2: GPU and Vector Experiments

single code. We use Edge Detection in grey scale images, a six-stage image processing pipeline; we can map each stage to one of three targets (GPU, vector or a CPU thread), allowing a total of $3^6 = 729$ different configurations, all generated from a single *hVISC* code. The graph shows seven of the configurations, and illustrates the dramatic differences in performance that can result from different mappings. This flexibility of *hVISC* enables a (future) run-time scheduler to choose configurations based on properties of the code, available hardware resources, and energy constraints.

5. ACKNOWLEDGEMENTS

This work was supported in part by the National Science Foundation (grant number CCF 13-02641), and by C-FAR, SRC STARnet Center sponsored by MARCO and DARPA.

Figure 3: Frame rates of different configurations through single *hVISC* object code.

6. REFERENCES

- [1] HSAIL. <http://www.hsafoundation.com/standards/>. Accessed: August 1, 2016.
- [2] Intel. <https://software.intel.com/en-us/intel-opencl>. Accessed: August 1, 2016.
- [3] Khronos Group. SPIR 1.0 Specification for OpenCL. <http://www.khronos.org/registry/cl/specs/spir-spec-1.0-provisional.pdf>.
- [4] C. Lattner and V. Adve. LLVM: A compilation framework for lifelong program analysis and transformation. In *Proc. Conf. on Code Generation and Optimization*, pages 75–88, San Jose, CA, USA, Mar 2004.
- [5] Nvidia Compute. PTX: Parallel thread execution isa version 2.3. http://developer.download.nvidia.com/compute/DevZone/docs/html/C/doc/ptx_isa_2.3.pdf, 2011.
- [6] P. Srivastava, M. Kotsifakou, M. Sinclair, R. Komuravelli, V. Adve, and S. Adve. hVISC: A portable abstraction for heterogeneous parallel systems. Technical report, Department of Computer Science, University of Illinois at Urbana-Champaign, 2016.
- [7] J. A. Stratton, C. Rodrigues, I.-J. Sung, N. Obeid, L.-W. Chang, N. Anssari, G. D. Liu, and W.-M. W. Hwu. Parboil: A revised benchmark suite for scientific and commercial throughput computing. *Center for Reliable and High-Performance Computing*, 2012.