Heterogeneous Habanero-C (H2C): A Portable Programming Model for Heterogeneous Processors

Deepak Majeti
Advisor: Vivek Sarkar
Rice University

1. INTRODUCTION

1.1 Motivation

With the end of Dennard scaling [9], hardware manufacturers are resorting to heterogeneous multicore processors to deliver increased performance. Heterogeneity is everywhere—from mobile phones to largest supercomputers. Heterogeneous systems with their massive computational capabilities have opened opportunities to solve computational problems which include DNA sequencing, medical imaging, big-data analytics, human brain simulation, and particle simulations. The current top two super computers: Tianhe-2 from China and Titan from United States have heterogeneous hardware.

However, a key challenge is that, existing programming languages used to develop software applications are not able to utilize the full potential of these newer and faster processors. Consequently, application programmers have to deal with new low-level programming languages; furthermore, these languages involve non-trivial learning and training. Extensive training has always been a barrier for the adoption of any new language. Furthermore, legacy software applications as well as libraries need re-targeting for these newer hardware causing a portability challenge.

Heterogeneous architectures also differ in their architectural features. For example, CPU+GPU systems are the dominant heterogeneous architecture found today where the CPU contains a small number of "fat" cores, and the GPU contains a much larger number of "thin" cores. Further, the memory hierarchy and cache structures are very different on the CPU and GPU sides. With such diverse characteristics, it is not only hard to program these systems in a portable manner, but also very challenging to optimize them.

An effective approach to tackles the aforementioned problems is to extend existing widely-used programming languages by adding a few carefully selected high-level constructs to express the programmer's intent and develop compiler technologies to efficiently map the program on the particular target hardware.

1.2 Contributions and Impact

My research addresses the aforementioned programmability problems by extending the familiar C programming language. The concepts are applicable to any other language. As part of my research in the Habanero Extreme Scale Software project led by Prof. Vivek Sarkar I developed Heterogeneous Habanero-C (H2C), a programming framework to build applications on modern heterogeneous hardware. This involved the design and development of high-level language constructs, compiler transformations, and runtime support.

A summary of my research contributions is as follows:

- Extended Habanero-C [8] (an extension of the popular C programming language) to target heterogeneous CPU, GPU, APU and DSP architectures. The goal is to enable a common programming platform for domain experts, software developers and ninja parallel programmers while also providing portability, performance and productivity.
- Designed intuitive language constructs to express heterogeneous parallelism. The extensions are minimal in order to keep it as close to the base language as possible. The programmer only needs to specify the parallelism in his application in a machine independent manner.
- Developed a compiler and runtime framework to efficiently map the user program to the underlying hardware. H2C compiler produces code tuned to a particular target architecture, paying attention to its memory hierarchy among other architectural specifics.
- Developed Shared Virtual Memory (SVM) to support sharing of pointers between CPU and GPU.
- Developed a data layout framework to automatically generate code with best data layout suited for a given architecture.

My H2C framework made significant contributions to a few applications as summarized below:

- *Parallelization of Lattice Boltzmann Method Simulation*: In this project, we parallelized a sequential C program of the Lattice Boltzmann Method (LBM) simulation provided to us by Halliburton Services. When I ported the sequential program to H2C, the generated program took 18 minutes to process an image of size $300 \times 300 \times 300$ for 10000 timesteps compared to 4 days taken by the original sequential C program.
- *Enhancing Medical Imaging Pipeline*: By using the data layout framework, we demonstrated a 2x improvement using H2C compared to the hand-written OpenCL program. The medical imaging pipeline is used in the detection of cancer cells.
- *Search for a rational curve on Del Pezzo surface*: The existence or absence of a rational curve on a Del Pezzo surface has not been proved. In this project, we use

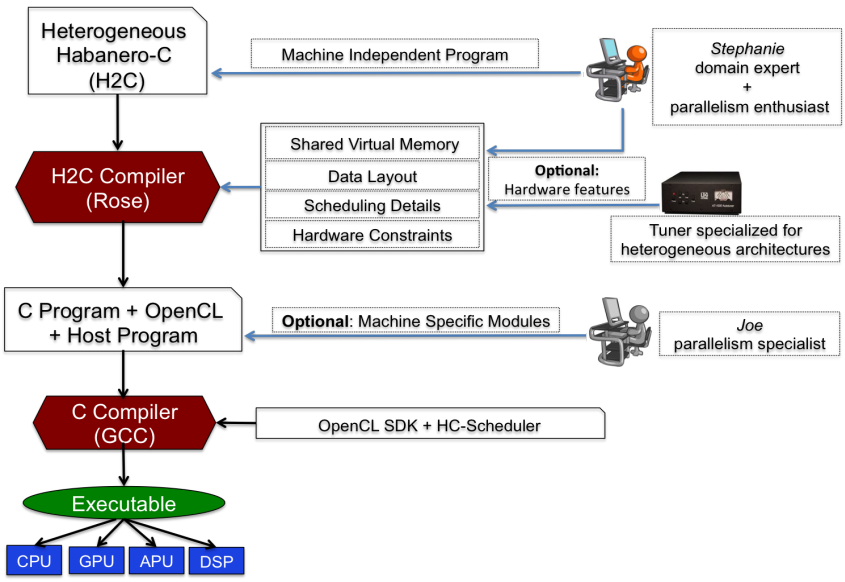

Figure 1: Overall H2C Compilation Framework

computation techniques and state of art GPUs to show that a solution does not exist upto degree 12.

The rest of this document is organized as follows. Section 2 describes the design and implementation of the H2C framework. Section 3 discusses the project in detail where H2C played an important role. Section 4 describes a few current and future research goals. Section 5 compares H2C with some related work. Section 6 finally concludes.

2. HETEROGENEOUS HABANERO-C (H2C) FRAMEWORK

This section describes the Heterogeneous Habanero-C (H2C) language, compiler and runtime extensions targeted towards heterogeneous architectures.

2.1 Background

The Habanero-C language extends the C language with parallel constructs which include, **forasync**, **async**, **finish**, **next**, **single**, **isolated**, **futures**, **at** and **await** to help achieve asynchronous fine grain task parallelism. In our experience we found that these constructs are enough to express most of the parallelism patterns. H2C programming language extends these constructs and added a few new clauses to support heterogeneous CPU, GPU, APU and DSP hardware. Below is the description of the constructs extended.

- **async** *copyin*<args> *copyout*<args> *at*<device>: Asynchronously copy data specified by the arguments to and from the *device*.
- **forasync** *point*<args> *range*<args> *at*<device>{Body}: Multi-dimensional data/task parallel loop. The loop indices are specified by the *point* clause. The loop bounds are specified by the *range* clause and *at* clause is used to specify the mapping of the kernel to the devices. The loop can be mapped to multiple devices. There is no implicit barrier at the end of the forasync construct.
- **finish** {Body}: Ensures tasks spawned inside *body* are completed.
- **phaser-next**: Phasers [14] are fine-grain synchronization constructs introduced by the Habanero programming model. Supporting phasers on certain architectures like GPUs requires compiler transformations.

2.2 H2C Compiler

An application developer writes a program in H2C, oblivious to the underlying hardware. The H2C compiler performs a two step compilation.

- In the first phase, it applies high-level transformations such as communication and computation overlap and identifies parallelism patterns. It then uses a *machine description* specification and automatically generates OpenCL code tuned to a particular hardware.
- In the second phase, a standard C compiler(say GCC) compiles the code to a target specific executable. At this phase, an expert programmer can choose to insert customized modules for the given architecture.

The machine description consists of a light-weight profile of the target hardware. The important hardware features include (i) micro-architectural details such as the type and number of execution units available, the number of registers, pipeline units and other specific capabilities, (ii) memory hierarchy and (iii) hardware constraints such as functional limitations. Figure 1 shows the overall H2C compilation framework.

The H2C compiler also supports embedded DSLs for stencils and reuse patterns. It automatically generates code to take advantage of special scratch-pad memory buffers available (eg: local shared memory is commonly used on the GPUs).

Supporting phasers on GPUs is a challenging task since it requires blocking mechanisms. Since there is no context switching on GPU hardware, we use techniques like kernel splitting and smart scheduling to achieve blocking semantics.

2.3 H2C Runtime

The H2C runtime performs the following tasks:

- Handles the mapping and scheduling of the kernels. The programmer or an auto-tuner can specify the mapping.
- Seamlessly manages the memory handles of various devices on a given hardware with the help of fat-pointers on the host.
- Identifies the target hardware and optimizes the communication.
- Automatically selects the best memory allocation (pinned vs. non-pinned).

2.4 H2C Highlights

Some novel features of H2C include shared virtual memory (SVM) [6] and data layout framework [10].

2.4.1 Shared Virtual Memory (SVM)

On certain architectures such as the Intel Sandy Bridge and AMD APU fusion, the CPU and GPU are integrated on the same die and the main memory is shared. The virtual memory, however is different for CPU and GPU, which prohibit many pointer-based applications from taking advantage of the GPU. The programmer has to translate the CPU pointers to GPU pointers manually.

SVM enables mapping of the virtual memory between the CPU and GPU via smart compiler techniques; this enables sharing of pointers across CPU and GPU. The H2C compiler generates code to translate the pointers between CPU to GPU.

With the help of SVM, the programmer can now run pointer intensive applications written in H2C on both the CPU and GPUs without any manual pointer translations. GPU core consumes less energy compared to the "fat" CPU cores. We tested SVM on applications including soft-body physics simulation, face detection, BTree, single-source shortest path, and breadth-first search. Results on an Intel IVB integrated GPU show an energy efficiency improvement of 3.5× and, on average, 69% improvement over multicore CPU executions.

2.4.2 Data Layout Framework

The data layout problem attempts to determine the best layout of data items in memory for a given application running on a given architecture. Many factors contribute to determining the optimal data layout of a program: (a) number of parallel hardware threads/contexts available on the target machine; (b) memory hierarchy of the target machine; (c) data access pattern in the program; (d) input size of the program. There are several ways a programmer can specify the layout of a data structure, e.g, Array-Of-Struct (AoS) and Struct-Of-Array (SoA) are two simple alternatives. In most scenarios, the programmer does not have any information on the underlying hardware on which the application is running. For example, a CPU usually performs well with an AoS layout because AoS can benefit prefetching and cache sharing, whereas the GPU performs well with a SoA layout because it can benefit from coalescing of memory loads. Thus, optimizing compilers should perform data layout transformations automatically, when possible. Compiler-driven data-layout reduces programming burden and at the same achieves portable performance across a wide-variety of platforms.

A theoretical contribution of my work is that we proved that the problem of finding the optimal data layout for a certain section of the program considering only Array-Of-Struct (AoS) and Struct-Of-Array (SoA) layouts is NP-hard. We also showed that if we know the best data layout for a given section of the program, then the optimal data layout for the entire program can be found in P-time. We extend the H2C compiler to automatically determine the best data layout for the input H2C program on a given hardware. The best layout could either be the same layout for the entire program or different layouts for different parts of the program and data remapping in between.

3. REAL-WORLD IMPACT OF H2C

This section highlights some of the impacts of my Heterogeneous Habanero-C framework.

3.1 Parallelization of Lattice Boltzmann Method Simulation

The goal of this project is to parallelize a sequential version of the Lattice Boltzmann Method (LBM) simulation from Halliburton Services by taking advantage of current heterogeneous CPU-GPU Hardware.

The sequential version of the program took around 6 seconds per time-step for an image size of $300 \times 300 \times 300$ and single floating point precision. I ported the sequential LBM code to H2C. The first step was to fuse two computationally intensive loops and linearize the array accesses. This reduced the sequential execution time to 1.2 seconds per time-step. The H2C compiler generated code further reduce this execution time to 0.018 seconds per time-step on an NVIDIA Tesla K20c GPU giving a performance improvement of 333×. The whole porting effort using H2C was a mere two days.

Another group (parallelism experts) in the applied mathematics department at Rice University hand-coded an OpenCL implementation by entirely re-writing the sequential LBM code which took them 3 weeks to debug and complete. The code generated by the H2C framework and this hand-coded version performed the same. The advantage of using H2C is that it is more portable and productive. An attractive feature of H2C is that one can debug the H2C version of the code since the H2C-compiler generates easy to debug CPU code variants.

3.2 Medical Imaging pipeline

The medical imaging pipeline is part of the CDSC [7] project. The goal is to improve the performance of existing medical imaging applications and make them feasible for real-time clinical usage. The pipeline consists of de-noising, registration and segmentation stages with image registration being the most time consuming phase. The image registration had 7 kernels. The original version was hand ported to

OpenCL [3] and used the same data layout for all the kernels. I ported the application to H2C. The automatic data layout framework in H2C analyzed the kernels and showed that different kernels required a different data layout and generated the corresponding code. We improved the performance of the registration stage by a factor of 2× compared to the original OpenCL program.

3.3 Search for a rational curve on Del Pezzo surface

In mathematics, Del Pezzo surfaces represent an important class of two-dimensional algebraic varieties [11], i.e. sets of solutions to polynomial equations. By the recent result of Saldago, Testa and Varilly-Alvarado, every smooth Del Pezzo surface of degree two over finite field $\mathbf{F}_3$ with three elements is always unirational [13], except possibly in the following two cases:

$$\begin{aligned} X_1/\mathbf{F}_3 &: -w^2 = (x^2+y^2)^2 + y^3z - yz^3, \\ X_2/\mathbf{F}_3 &: -w^2 = x^4 + y^3z - yz^3 \end{aligned} \tag{1}$$

It is conjectured that above two surfaces are also unirational. This will, in particular, imply the existence of a rational curve on them, or equivalently, parametrization $w(t)$, $x(t)$, $y(t)$, $z(t)$ by nonconstant polynomials in one variable t with coefficients in field $\mathbf{F}_3 = \mathbf{Z}/3\mathbf{Z}$. The existence of parametrization that satisfies one of the equations in (1) is not known.

We implemented a polynomial solver in H2C to take advantage of heterogeneous architectures to prove the existence of such parameterization. In order to efficiently search for a polynomial solution, we used symmetry as well as some arithmetic properties to prune the search range. Our experiments show that no solution exists upto degree 12. We are in the process of further optimizing the implementation to increase the search range and efficiency.

4. ONGOING RESEARCH & FUTURE WORK

I am currently focusing on the following research problems:

4.1 Data Layout Framework

In order to determine the best data-layout for a given device, our H2C compiler computes an affinity graph between the array and object fields accessed in a particular parallel region of the program. Computing this affinity graph precisely has great implication on choosing the device of execution and its layout. At the same time, it is complicated since it depends on various access patterns such as read-/write, number and type of accesses. This affinity graph is then mapped onto the machine model to obtain machine specific costs. Without accurate affinity graph would lead to I am currently exploring other metrics such as hardware performance counters to effectively evaluate the affinity between different fields and also the machine model.

The H2C compiler currently only considers Array-Of-Struct (AoS) and Struct-Of-Array (SoA) layouts. We are extending the framework to evaluate complex data layouts such as Array-Of-Struct-Of-Array (AoSoA) and morton order.

I am also working on extending the data-layout framework for hybrid CPU-GPU scheduling. When the kernel is mapped to multiple devices such as the CPU and the GPU, the data needs to be partitioned and laid out accordingly to obtain best performance and energy-use. The main challenge now is to ensure that the output produced by multiple devices are consistent and correct.

4.2 Heterogeneous Scheduling and data-layout using Neural Networks

One of the key challenges in current and future heterogeneous architectures is to come up with an optimal mapping and scheduling strategy for a give workload onto the available processing cores. The problem is hard because one has to consider multiple features such as data layout, locality, communications costs, energy budget, and hardware features. Given these parameters, our approach is use a machine learning based approach to train a model based on a set of static features and subsequently use these features during real run to predict the best data layout and the device to execute it on. Neural networks are of particular interest because one can combine multiple input features and characterize the machine accurately.

4.3 Distributed Environment

Extending the H2C framework to support distributed heterogeneous CPU-GPU architectures is important for the high performance computing applications. The challenges involve efficiently transferring the data between the accelerators on distributed nodes. The idea is to come up with efficient compiler techniques to overlap communication and computation across different nodes. I am currently extending the *async* copy constructs to support distributed MPI communication.

5. RELATED WORK

Par4All [4] is an automatic parallelizing compiler for C and Fortran programs. It targets heterogeneous architectures and automatically generates CUDA or OpenCL. With the help of a polyhedral framework, it detects parallelism, identifies reduction parallelism, and performs optimizations like induction variable substitution, loop fusion and array linearization. The drawback of automatic parallelization is that the scope is limited to regular parallelism patterns. It has been shown that the problem of automatic parallelization is a hard problem and the correct approach is to allow the programmer to specify the parallelism. H2C language provides high level first class constructs to specify the parallelism. The language extensions are minimal to keep it as close as possible to the source language.

StarPU [5] is a task programming library for hybrid architectures. The programmer has to explicitly specify a codelet for each different architecture. This poses a severe constraint on productivity. The H2C compiler uses information about the target architecture to generate a machine specific executable.

C++ AMP [1] supports GPUs mapping closures and adding new language extensions for data movement. However, it does not have features to overlap communication and computation, does not take advantage of special hardware buffers and does not support hybrid CPU and GPU computation. H2C framework supports all these features.

OpenACC [12] is a pragma based approach to target CPU-GPU architectures for C programs. The pragma model does not map well to heterogeneous hardware as the context is restricted to a parallel region in a given module and cannot span across different modules. H2C contexts are not limited to any single parallel region.

OpenCL [3] and CUDA [2] are low level programming languages which target GPUs. The issue with these languages is that they are not productive and require expert training to use them. H2C framework on the other hand provides high level parallel constructs and uses compiler optimization techniques to target a particular architecture.

Table 1 shows some differences between H2C and state-of-art high level programming models like C++ AMP and OpenACC.

Feature	C++ AMP	OpenACC	H2C
Shared pointers between CPU-GPU	No	No	Yes
Portable data layout framework	No	No	Yes
Communication + Computation overlap	Limited	Limited	Yes
Support scratchpad memory	Yes	No	Yes

Table 1: H2C vs. existing high-level programming models for heterogeneous parallelism

6. CONCLUSION

There is an urgent need to improve existing programming models to target the current and future heterogeneous architectures. In the H2C programming language, I have extended the popular C language with minimal language constructs with the ability to express various parallelism patterns. The H2C compiler incorporates start-of-the-art code transformation and optimization techniques such as SVM and data layout to generate code tuned to a particular architecture. The H2C runtime efficiently maps the various tasks to the available heterogeneous processors. The aggregate effect of H2C is that it makes programming for diverse architectures portable, productive and achieves high performance. We have demonstrated the effectiveness of H2C on three important applications.

7. REFERENCES

[1] C++ Accelerated Massive Parallelism. `http://msdn.microsoft.com/en-us/library/vstudio/hh265137.aspx`.

[2] NVIDIA Corporation. The CUDA Specification. `www.nvidia.com`.

[3] OpenCL:The open standard for parallel programming of heterogeneous systems. `http://www.khronos.org/opencl/`.

[4] Mehdi Amini, Onig Goubier, Serge Guelton, Janice Onanian Mcmahon, FranÃğois xavier Pasquier, GrÃľgoire PÃľan, and Pierre Villalon. Par4all: From convex array regions to heterogeneous computing.

[5] Cédric Augonnet, Samuel Thibault, Raymond Namyst, and Pierre-André Wacrenier. StarPU: A Unified Platform for Task Scheduling on Heterogeneous Multicore Architectures. *Concurrency and Computation: Practice and Experience, Special Issue: Euro-Par 2009*, 23:187–198, February 2011.

[6] R. Barik, R. Kaleem, D. Majeti, B. Lewis, T. Shpeisman, C. Hu, Y. Ni, and A. Tabatabai. Efficient Mapping of Irregular C++ Applications to Integrated GPUs. CGO'14, Florida, USA, 2014 (To Appear).

[7] Center for Domain Specific Computing. `www.cdsc.ucla.edu`.

[8] S. Chatterjee, S. TasÄśrlar, Z. Budimlic, V. Cave, M. Chabbi, M. Grossman, V. Sarkar, and Yonghong Yan. Integrating asynchronous task parallelism with mpi. In *IPDPS 2013*, pages 712–725, May 2013.

[9] R.H. Dennard, F.H. Gaensslen, V.L. Rideout, E. Bassous, and A.R. LeBlanc. Design of ion-implanted mosfet's with very small physical dimensions. *Solid-State Circuits, IEEE Journal of*, 9(5):256–268, Oct 1974.

[10] Deepak Majeti, Rajkishore Barik, Jisheng Zhao, Vivek Sarkar, and Max Grossman. Compiler Driven Data Layout Transformation for Heterogeneous Platforms. HeteroPar '13, Aachen, Germany, 2013. LNCS.

[11] Y.I. Manin. *Cubic Forms: Algebra, Geometry, Arithmetic*. North-Holland Mathematical Library. Elsevier Science, 1986.

[12] OpenACC. `www.openacc-standard.org`.

[13] C. Salgado, D. Testa, and A. Várilly-Alvarado. On the Unirationality of del Pezzo surfaces of degree two. *ArXiv e-prints*, April 2013.

[14] Jun Shirako, David M. Peixotto, Vivek Sarkar, and William N. Scherer. Phasers: A unified deadlock-free construct for collective and point-to-point synchronization. In *Proceedings of the 22Nd Annual International Conference on Supercomputing*, ICS '08, pages 277–288, New York, NY, USA, 2008. ACM.