Language and Compiler Extensions for Heterogeneous Computing

Albert Sidelnik (UIUC), María J. Garzarán (UIUC), David Padua (UIUC), Brad Chamberlain (Cray Inc.)

asideln2@illinois.edu garzaran@illinois.edu padua@illinois.edu bradc@cray.com

Introduction

Heterogeneous systems have many advantages:

- Very high bandwidth and FLOP count
- Performance / (Cost and Power) ratio is high
- Low-level existing programming models such as CUDA or OpenCL have made it somewhat easier to program compared to earlier efforts

Downsides to programming accelerators:

- Tremendous effort required to optimize
- Lack of automatic scheduling
- Explicit data transfers
- Different programming models for each component (CUDA, OpenMP, MPI, etc..) co-exist in the same program leading to a loss of portability

Goals of this project

- Write **portable** codes: The same program can be **compiled** and **optimized** for different classes of architectures, with the focus being on systems containing *accelerators*
- Leverage the language **Chapel**. The notion of parallelism and locality is built into Chapel from the ground up rather than using ad-hoc libraries or directives. Take advantage of the data parallel primitives of Chapel
- Show that it is possible to write accelerator code in a **high-level** language and achieve similar results as one would in a language such as CUDA

Method

Chapel Background

- Part of the Darpa led HPCS program intended to increase programmer productivity on high-end systems
- Supports task-, data-, and nested-parallelism
- Parallel concepts influenced by **ZPL** and **HPF**
- Domains are first-class objects in Chapel to describe a multi-dimensional index space
  - E.g. **var** dom = [1..100];

Language Extensions

- Arrays that need to be used on the accelerator are declared using the distribution *GPUDist*
- Depend on ***forall*** as the main support for data-parallelism on a device
- Leverage Chapel's support for user-defined distributions[1]
- Low-level support for different accelerator memory spaces including shared and constant memory
- Support for both explicit and implicit data transfers
- Current ongoing work in supporting whole array assignments and operations
- Support for reduction and scans

Compiler Support

- Currently generates CUDA code for the accelerator kernels and C for the host
- Compiler analysis for implicit data transfers between host and device
- Memory optimizations

Motivation – HPCC Stream

```
config const m = 1000, tbSizeX = 256;

const alpha = 3.0;

const ProbDist = new GPUDist(rank=1, tbSizeX);

const ProbSpace: domain(1) distributed ProbDist = [1..m];

var A, B, C: [ProbSpace] real;


forall (a,b,c) in (A,B,C) do
  a = b + alpha * c;
```

New configuration constant to specify # threads per block

Distribution changed to target a GPU

Loops and arrays using this domain are implemented on the GPU

No changes required to the computation for other architectures

•The above code can run on either a GPU, Multi-core, or Distributed memory architecture by plugging in a different distribution such as Block, Cyclic, Block-Cyclic, etc.

•Even though a performance comparison is not shown (due to lack of space), this version of Stream matches the performance the CUDA implementation of Stream.

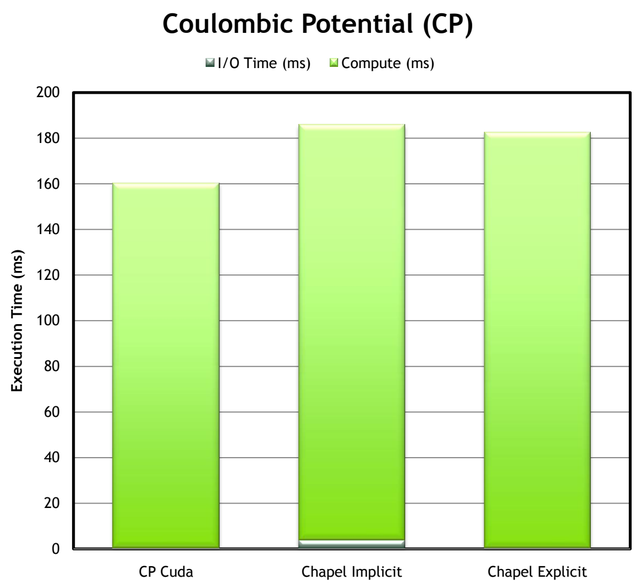
Motivation #2 – Jacobi Method 2D

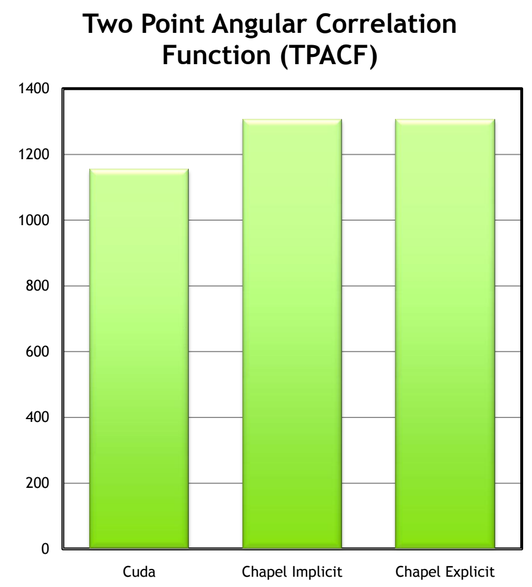
```
config const n = 200, epsilon = 0.0001;
const gpuDist = new GPUDist(rank=2, tbSizeX=16, tbSizeY=16);
const ProbSpace: domain(2) distributed gpuDist = [1..n,1..n];
const BigDomain: domain(2) distributed gpuDist = [0..n+1,0..n+1];
var X, Xnew: [BigDomain] real;
X[n+1, 1..n] = 1.0;
var iteration = 0, delta: real;
const north = (-1,0), south = (1,0), east = (0,1),  west = (0,-1);
do {
  forall ij in (gpuProbSpace) do
    Xnew(ij) = (Xnew(ij+north)+Xnew(ij+south)+ Xnew(ij+west)+Xnew(ij+east))/ 4.0;


   delta = max reduce abs(Xnew[ProbSpace]- X[ProbSpace]);
} while (delta > epsilon);
```

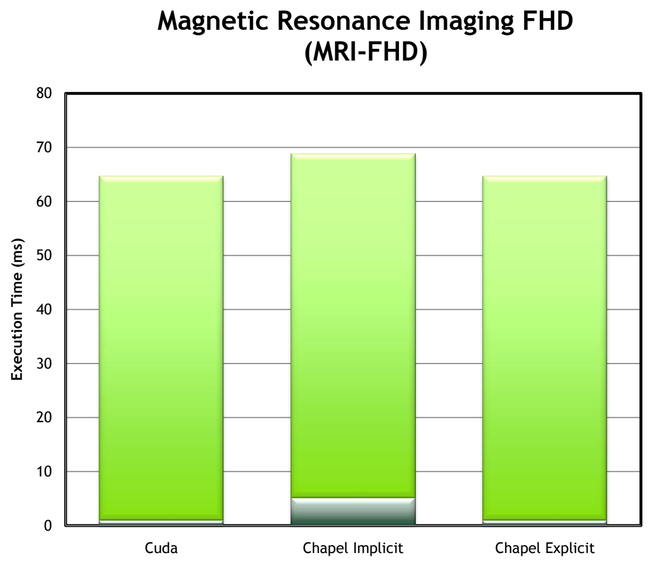
Results – Parboil Benchmark Suite http://impact.crhc.illinois.edu/parboil.php

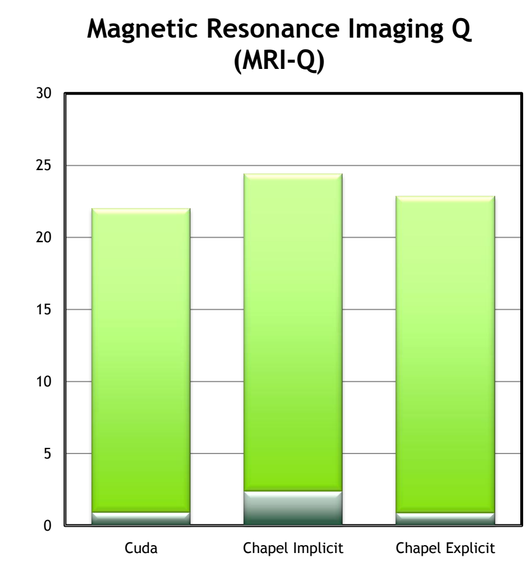
Experimental results using the **Parboil Benchmark Suite** based on real applications written in CUDA. Experiments were made using a Nvidia GTX280 GPU with the CUDA 2.3 environment.

For these benchmarks, we compare the original CUDA implementation, and the benchmarks ported to Chapel using both implicit and explicit data transfers.









- The difference in compute performance between the Chapel and CUDA for CP, TPACF, and MRI-Q are due to additional overhead-code being generated by the Chapel compiler. Ongoing work is being done to lower this overhead.

- TPACF is purely compute bound with very little data transfers between the host and device making time spent in I/O negligible.

Future and Ongoing Work

•Provide support for texture memory

•Implement support for whole-array operations and perform optimizations to fuse them into a single kernel

•Improve current conservative approach to implicit data transfers

•Develop compiler analysis to detect ideal forms of memory to place data into

•Enable through language and compiler support, overlapping computation and communication over an accelerator

•Extend the current *GPUDist* distribution to support clusters of accelerators or other heterogeneous nodes

•Develop autotuning techniques to study optimization strategies leading to programs self-adapting at run-time including scheduling decisions based on characteristics of the data

•Close the performance gap between Chapel and CUDA

•Generate OpenCL code in addition to CUDA for additional accelerator portability

Conclusions

•In this work we show that it is possible to target heterogeneous architectures using high-level languages that were built from the ground up to support parallelism without any substantial loss in performance when compared to a low-level programming model.

•If the user needs the low-level facilities offered by a model such as CUDA, Chapel provides low-level CUDA-like features for the user to leverage.

To learn more about Chapel, visit http://cray.chapel.com

Acknowledgments

This material is based upon work supported by the National Science Foundation under Awards CCF 0702260, CNS 0509432, the Defense Advanced Research Projects Agency under its Agreement No. HR0011-07-9-0001. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Defense Advanced Research Projects Agency.

[1] B. Chamberlain, S. Deitz, D. Iten, S. Choi
User-Defined Distributions in Chapel
In Proceedings of the 2nd USENIX Workshop on Hot Topics in Parallelism (*HotPar*) 2010