Leveraging Polyhedral Compilation in Chapel Compiler

Sahil Yerawar, Siddharth Bhat, Michael Ferguson, Philip Pfaffe, Ramakrishna Upadrasta

IIT Hyderabad, IIIT Hyderabad, Cray Inc. , Karlsruhe Institute of Technology, IIT Hyderabad

Introduction

- Chapel is an emerging parallel programming language developed with the aim of providing better performance in High-Performance Computing as well as accessibility to the newcomer programmers in order to provide a relatively smoother learning curve in Parallel Computing.
- It is known that Chapel uses LLVM as one of its backups to leverage its optimization toolset.
- We propose to integrate Polly's Optimization passes in Chapel. In today's world, it is well known that High-Performance Computing involves nested loops as one of their most compute intensive parts which are efficiently handled by Polly-based optimizations, thereby making HPC more productive

Motivation

- Chapel has been constantly reaping benefits of optimization from LLVM backend and its robust ongoing set of inclusion of optimizations
- In a HPC setup, there are always instances of compute-intensive loops which take up most parts of the program in terms of compute time. (Hot Regions)
- Apart from the set of optimizations Polly provides, it also has its own built-in PPCG-NVPTX backend to generate native CUDA code for GPU's

Challenges

- Initially Polly failed to recognize simple Chapel Loops involving Multi-Dimensional Array Accesses. On further investigation, found out that Polly was unable to detect inductively loop-invariant accesses.
- The above approach didn't handle more generalized cases which Chapel analyzed. Thus alternative approaches were discussed (one of them is present in the next section)
- Aiming for both polly-codegen and polly-codegen-ppcg, we first tried to teach the existing ppcg-codegen about our new intrinsic.
- Issues faced during array allocation in CUDA due to mismatch of data given by Chapel and what Polly expects.
- Polly's GPU runtime checks were improper due to wrong evaluation of Invalid context of SCoP.
  - Due to the constants of invalid context were wrongly considered as LargeInts which affected RTC's

Multi-Dimensional Array Indexing Intrinsic

There are primarily one of the two views of array indexing used for multiple dimensions case
For example:

```
int ex1(int n, int m, B[n][m],
        int x1, int x2, int y1, int y2) {
    assert(x1 != x2);
    assert(y1 != y2);
    B[x1][y1] = 1;
    printf("%d", B[x2][y2]);
    exit(0);
}
```

If we consider C language based array indexing, they are just concerned if the flattened representations of the array accesses coincide with same array location or not.

But if we consider the following tuples:

```
n = m = 3
x1 = 1, y1 = 2; B[x1][y1] = nx1+y1 = 3*1+2=5
x2 = 0, y2 = 5; B[x2][y2] = nx2+y2 = 3*0+5=5
```

Here the second tuple represents an illegal access in C perspective which is perceived to be legal when considered in flattened format

However, there are some languages(Julia, Fortran, Chapel) which tuple-indexed semantics where one can infer that

```
[x1][y1] != [x2][y2] iff x1 != x2 || y1 != y2
```

As of now, LLVM's getelementptr() is concerned with only flattened representations. Due to dangers of alias of illegal accesses, Polly considers the multidimensional expressions, use and evaluates them at runtime, causing a runtime performance hit. This can cause Polly to bail out on most of the cases.

Ideally one would like to convey the multi-dimensional semantics directly to Polly, making code faster and easier to analyze.

This can be achieved by introducing a new intrinsic "multidim_array_index", allowing to represent multidimensional array accesses without the compulsion for flattening

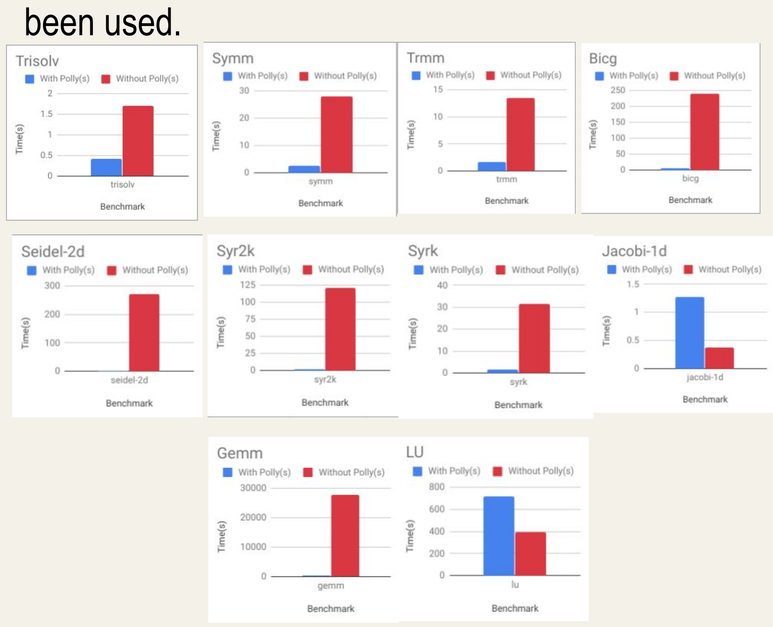
Use cases

- This methodology has been used in an experimental branch of Polly and was used on the COSMOS climate weather model. This greatly helped increase the accuracy of Polly's analysis, due to elimination of guessing here
- Molly also uses this kind of idea
- Implemented as a part of unifying GSoC effort between Chapel and Polly

Evaluation and Results

During the GSoC Period, much of the analysis of integrating Polly within Chapel was done by only using the examples of 2D array initialization and 2d matrix multiplication. For a more extensive testing, some of the Polybench benchmarks were ported to Chapel.

In this evaluation, only the LARGE_DATASET has been used.



Conclusion

- Push forward for integrated Chapel-Polly pipeline
- Working for inclusion of multidim_array_index() method could be introduced within LLVM. RFC will be developed shortly
- Extending Polly coverage to more general Chapel Arrays like strided Arrays, Arrays with custom indices and so on.
- Covering data-parallel forall() loops in addition to the simpler for loops.

Acknowledgments

I would like to thank the GSoC Organization for allowing me to collaborate with Polly and Chapel on this unique project with my mentors.

I would like to extend my special thanks to Michael Kruse for always being available and also for allowing reference of Molly used in our proposal

Again, special thanks to Tobias Grosser for allowing me to use the reference related to COSMOS climate weather model for substantiating our claim regarding multi-dimensional array indexing

Contact Information

Sahil Yerawar: cs15btech11044@iith.ac.in

Siddharth Bhat: siddharth.bhat@research.iiit.ac.in

Michael Ferguson: mppf@cray.com

Philip Pfaffe: philip.pfaffe@kit.edu

Ramakrishna Upadrasta: ramakrishna@iith.ac.in

www.meteconferences.org