Kernel Fusion/Decomposition for Automatic GPU-Offloading

Alok Mishra*, Martin Kong†, Barbara Chapman*†
alok.mishra@stonybrook.edu, mkong@bnl.gov, barbara.chapman@stonybrook.edu
*Stony Brook University – USA, †Brookhaven National Laboratory – USA

***Abstract*—The massively parallel architecture of GPU accelerators are being harnessed to expedite computational workloads in cutting edge scientific research. Unfortunately writing applications for GPUs requires extensive knowledge of the underlying architecture, the application and the interfacing programming model. Moreover, (re-)writing kernels using lower-level programming models such as CUDA and OpenCL is a burden for application scientists. A more appealing strategy is to leverage a programming model layered on directive-based optimization: OpenMP, whose recent specification significantly extends its accelerator functionalities.**

Despite this, it is still quite challenging to optimize large scale applications, since "pragmatizing" each kernel is a repetitive and complex task. In large scale applications most of the operations could be small, don't have enough computational work to justify a GPU execution, deeply buried in the library specification, or evenly spread throughout the application. Thus, we seek to design and build a compiler framework that can automatically and profitably offload regions of code with these characteristics.

The driving principle of our work resides in generating numerous kernel variants that result from fusing and/or decomposing existing function bodies. We analyze the program's call graph to determine the "proximity" of kernel calls and evaluate the degree of data reuse among adjacent or "close-enough" calls. When such patterns are detected we generate several scenarios, until producing a single variant whose footprint is near the capacity of the GPU.

To compare the potential performance among the various kernel variants generated, we are designing an adaptive cost model. The precision of this cost model will depend upon the analyzability of the program. We are also building upon existing cost models like Baghsorkhi et al.'s model which proposed a work flow graph based analytical model and a recent Hong et al.'s model which propose the use of abstract kernel emulations to help identify the performance bottlenecks of a GPU program execution. Along with these we introduce GPU initialization and data transfer cost to the model.

Once the profitable kernel variants are detected, we automatically insert pertinent OpenMP directives and provide a newly generated code supporting GPU offloading.

I. Extended Abstract

The massively parallel architecture of GPU accelerators are being harnessed to expedite computational workloads in cutting edge scientific research. More researchers and developers desire to port their applications to GPU-based clusters, due to their extensive parallelism and energy efficiency.

Unfortunately porting and writing applications for accelerators, such as GPUs, requires extensive knowledge of the underlying architecture, the application/algorithm and the interfacing programming model (e.g. CUDA). Moreover, (re-)writing kernels using lower-level programming models such as CUDA and OpenCL is a burden for application scientists. Clearly, this impacts productivity. A more appealing strategy is to leverage a programming model layered on directive-based optimization: OpenMP. Since version 4.5, OpenMP provides device offloading capabilities, and its most recent specification (Nov 2018) [1] significantly extends its accelerator functionality.

Despite the variety of programming models available, it is still quite challenging to optimize large scale applications consisting of tens-to-hundreds of thousands lines of code. Effectively, "pragmatizing" each kernel is a repetitive and complex task. An example of this is type of applications is the Grid library [2], which is a C++ data-parallel library, consisting roughly 40K lines of code. It is used in applications of high-energy physics, and in particular, for Lattice Quantum Chromodynamics (QCD). Grid provides a core set of mathematical operators with several back-end implementations. However, most of the operations are small, don't have enough computational work to justify a GPU execution, are deeply buried in the library specification, or are evenly spread through the application using Grid. Thus, we seek to design and build a compiler framework that can automatically and profitably offload regions of code with these characteristics.

To address these issues, our framework performs the following tasks:

1) detect potentially offloadable kernels;
2) identify common code invocation patterns arising in a program;
3) evaluate several kernel variants resulting from fusion/decomposition of kernels;
4) evaluate the profitability of each kernel variant via a novel and adaptive cost model;
5) insert pertinent compiler directives to perform offloading.

Recurring Function Call Patterns. The driving principle of our work resides in generating numerous kernel variants that result from fusing and/or decomposing existing function bodies. We analyze the program's call graph to determine the "proximity" of kernel calls and evaluate the degree of data reuse among adjacent or "close-enough" calls. When such patterns are detected we generate several scenarios, until producing a single variant whose footprint is near the capacity of the GPU. Listing 1 shows two such scenarios which we are currently targeting. More such patterns will be detected in future work.

978-1-7281-1436-1/19/$31.00 © 2019 IEEE

CGO 2019, Washington, DC, USA
Student Research Competition

Code 1: Two kernels using same data

```
/* Kernel 1 using array A */
for(...) { ... }
/* Kernel 1 End */
/*Serial Code Start
... // no dependency on any kernel
Serial Code End*/
/* Kernel 2 using array A */
for(...) { ... }
/* Kernel 2 End */
```

Code 2: Kernel with large data

```
float func(struct object x) {
 return ...;
}
void func2(int N, struct object B[N]) {
   ...
   // Kernel with data larger than the GPU memory can support
   for(int i=0; i<N; i++) {
      A[i] = func(B[i]);
   }
}
```

Code 3: Code auto-generated from Code 1 to fuse two kernels reusing data and moving some serial code out of kernel

```
// Kernels fused together
#pragma omp target teams map(A)
{
   /* Parallel Loop 1 using array A */
   #pragma omp distribute parallel for
   for(...) { ... }

   /* Parallel Loop 2 using array A */
   #pragma omp distribute parallel for
   for(...) { ... }
}

/*Serial Code Start
...  // Moved out of the kernels
Serial Code End*/
```

Code 4: Code auto-generated from Code 2 for decomposing a large kernel and splitting the computation over multiple devices

```
#pragma omp declare target
float func(struct object x) { return ...; }
#pragma omp end declare target
void func2(int N, struct object B[N]) {
   ...
   // Decomposing kernel over multiple devices
   #pragma omp target enter data map(alloc:A[0:N/2]) \
                                 map(to:B[0:N/2]) device(0)
   #pragma omp target teams distribute parallel for device(0)
   for(int i=0; i<N/2; i++) { A[i] = func(B[i]); }

   #pragma omp target enter data map(alloc:A[N/2:N/2]) \
                                 map(to:B[N/2:N/2]) device(1)
   #pragma omp target teams distribute parallel for device(1)
   for(int i=N/2; i<N; i++) { A[i] = func(B[i]); }
}
```

Listing 1: Example codes for fusing and decomposing kernels

Profitability of a Kernel Variant. To compare the potential performance among the various kernel variants generated, we are designing an adaptive cost model. The precision of this cost model will depend upon the analyzability of the program. For instance, if the kernel body is affine and fits the standard requirements of the polyhedral model [3], we can detect the exact set of live-in and live-out data, compute the number of parallel loop dimensions and easily extract its operational intensity. If however a kernel doesn't fit into the standard requirements, over-approximation techniques can be leveraged to compute less precise information [4]. Being able to compute the live-in and live-out dataset of kernels is useful when the kernel parameters are complex or deep data structures. We are also building upon existing cost models like the model of Baghsorkhi et al. [5] which proposed a work flow graph (WFG) based analytical model and a recent model by Hong et al. [6] which propose the use of abstract kernel emulations to help identify the performance bottlenecks of a GPU program execution. Along with these we introduce GPU initialization and data transfer cost to the model.

Automatic Kernel Generation. Once a profitable kernel variant is detected, we proceed to generate its body. As discussed previously, if the involved kernels are affine (or close to affine) we can further optimize the code for locality, multi-level parallelism and automatically generate the necessary set of pragmas to enable the offloading. Tools such as DawnCC [7] and Polly [8], with their advanced scheduling frameworks and code generation capabilities are currently being considered for this task.

To conclude, GPUs are increasingly important to the HPC industry, but GPU offloading is a strenuous and laborious job. We aims at achieving automatic source-to-source code translation to support GPU offloading, by analyzing several code variants resulting from kernel fusion/decomposition.

References

[1] O. Board, "Openmp 5.0 specification," 2018. [Online]. Available: https://www.openmp.org/wp-content/uploads/OpenMP-API-Specification-5.0.pdf

[2] P. Boyle, A. Yamaguchi, G. Cossu, and A. Portelli, "Grid: A next generation data parallel c++ qcd library," *arXiv preprint arXiv:1512.03487*, 2015.

[3] P. Feautrier, "Dataflow analysis of array and scalar references," *International Journal of Parallel Programming*, vol. 20, no. 1, pp. 23–53, 1991.

[4] M.-W. Benabderrahmane, L.-N. Pouchet, A. Cohen, and C. Bastoul, "The polyhedral model is more widely applicable than you think," in *International Conference on Compiler Construction*. Springer, 2010, pp. 283–303.

[5] S. S. Baghsorkhi, M. Delahaye, S. J. Patel, W. D. Gropp, and W.-m. W. Hwu, "An adaptive performance modeling tool for gpu architectures," in *ACM Sigplan Notices*, vol. 45, no. 5. ACM, 2010, pp. 105–114.

[6] C. Hong, A. Sukumaran-Rajam, J. Kim, P. S. Rawat, S. Krishnamoorthy, L.-N. Pouchet, F. Rastello, and P. Sadayappan, "Performance modeling for gpus using abstract kernel emulation," in *Proceedings of the 23rd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. ACM, 2018, pp. 397–398.

[7] G. Mendonça, B. Guimarães, P. Alves, M. Pereira, G. Araújo, and F. M. Q. Pereira, "Dawncc: automatic annotation for data parallelism and offloading," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 14, no. 2, p. 13, 2017.

[8] T. Grosser, H. Zheng, R. Aloor, A. Simbürger, A. Größlinger, and L.-N. Pouchet, "Polly-polyhedral optimization in llvm," in *Proceedings of the First International Workshop on Polyhedral Compilation Techniques (IMPACT)*, vol. 2011, 2011, p. 1.