Program Transformation for Automatic GPU-Offloading with OpenMP

Alok Mishra[1], Abid Malik (mentor)[2], Barbara Chapman (advisor)[1,2]

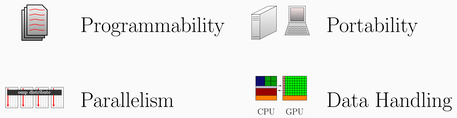
alok.mishra@stonybrook.edu, amalik@bnl.gov, barbara.chapman@stonybrook.edu

[1]Stony Brook University – USA, [2]Brookhaven National Laboratory – USA

Abstract

In this research, we design and build a compiler framework that can automatically discover OpenMP kernels, recommend several potential OpenMP variants for offloading that kernel to the GPU, and using a novel static neural network-based compile time cost model, predict and return the most optimal of those variants. We divide our framework into 3 modules, each of which functions independently. Module 1 detects and analyzes an OpenMP kernel and suggests several variants, by applying various potential code level transformations, for offloading that kernel to a GPU. In module 2, we define COMPOFF, which employs ML techniques (for the first time in OpenMP) to predict the Cost of OpenMP OFFloading statically. In module 3, we modify the original source code using the analysis and prediction from the other modules to modify the source code and returns newly generated code that supports GPU offloading. Our preliminary findings indicate that our framework will aid scientists transfer their programs to the new heterogeneous computing environment.

Challenges



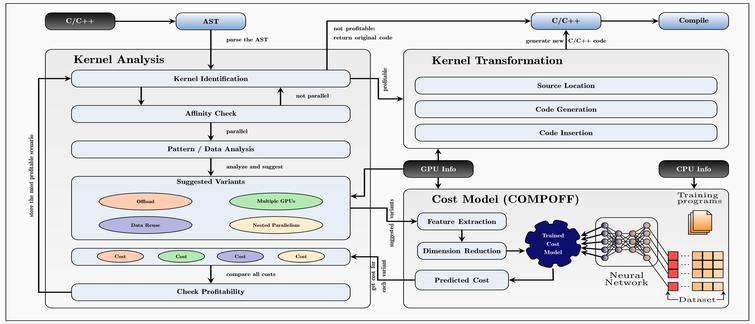
OpenMP

- Defacto standard for parallel programming
- More portable
- Easier to code
- Multiple compiler support
- **Supports GPU offloading**
- Unfortunately, effectively "pragmatizing" each kernel is a repetitive and complex task.

Proposal

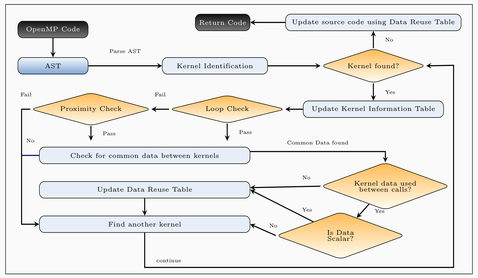
- Design & build a compiler framework to automatically offload profitable regions of code to GPUs
- Kernel detection – OpenMP parallel regions
- Patterns / Data Analysis
- Evaluate several variants of kernels resulting in different OpenMP options
  - Choose most profitable variant using Cost Model
- Novel and adaptive Cost Model
  - Profitability of offloading each kernel variant
  - Using Machine Learning
- Code Generation
  - Insert pertinent OpenMP code

High Level Flow



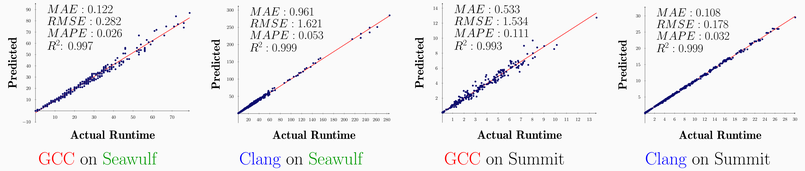
In the backend we use LLVM/Clang tools. Analysis Pass determine the "proximity" of kernel calls and evaluate the degree of data reuse between adjacent calls and suggests several variants for offloading. Cost model is a trained neural network to predict the **C**ost of **O**pen**MP** **OFF**loading statically. Kernel Transformation adds pertinent OpenMP code to the kernel to support offloading.

Data Reuse Analysis



Moving only required data between the CPU and GPU, and reusing data between subsequent kernels improved the performance of code.

Cross Validation of COMPOFF



GCC on Seawulf | Clang on Seawulf | GCC on Summit | Clang on Summit

Since training and testing are done on numerous separate portions, K-Fold Cross Validation has a significant advantage in terms of avoiding selection bias. Because we've trained and tested our model on so many distinct sub-datasets, we may be even more confident in its resilience. For our experiments, we did a 10-Fold cross validation and got over 95% accuracy in all of our models.

Output Examples

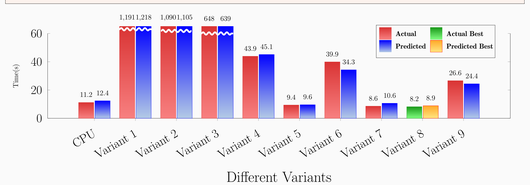
Input Code – Multiplication of 3 matrices A, B and C, using an intermediate matrix C1, to get the resultant matrix D.

```
/* Kernel 1 assigning array C1 */
#pragma omp parallel for
   for(int i=0; i<N; i++)
      for(int j=0; j<N; j++)
         for(int k=0; k<N; k++)
            C1[i][j] += A[i][k] * B[k][j];
/* Kernel 2 using array C1 */
#pragma omp parallel for
   for(int i=0; i<N; i++)
      for(int j=0; j<N; j++)
         for(int k=0; k<N; k++)
            D[i][j] += C1[i][k] * C[k][j];
```

Output Variant 7 – Auto-generated code taking advantage of nested parallelism by collapsing loops

```
#pragma omp target data map(A[N][N], B[N][N], C1[N][N])
#pragma omp target teams distribute parallel for collapse(2)
   for(int i=0; i<N; i++)
      for(int j=0; j<N; j++)
         for(int k=0; k<N; k++)
            C1[i][j] += A[i][k] * B[k][j];

#pragma omp target data map(D[N][N], C[N][N], C1[N][N])
#pragma omp target teams distribute parallel for collapse(2)
   for(int i=0; i<N; i++)
      for(int j=0; j<N; j++)
         for(int k=0; k<N; k++)
            D[i][j] += C1[i][k] * C[k][j];
```



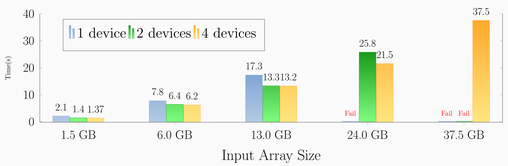
GPU Offload – Multiplying three 3000x3000 matrices on Summit Supercomputer using NVIDIA V100 GPU. We suggest several transformation that can be used to offload and predict the best variant using COMPOFF.

Output Variant 4 – Auto-generated code to offload the two kernels to GPU, distributing threads over different teams

```
#pragma omp target data map(A[N][N], B[N][N], C1[N][N])
#pragma omp target teams distribute parallel for
   for(int i=0; i<N; i++)
      for(int j=0; j<N; j++)
         for(int k=0; k<N; k++)
            C1[i][j] += A[i][k] * B[k][j];
#pragma omp target data map(D[N][N], C[N][N], C1[N][N])
#pragma omp target teams distribute parallel for
   for(int i=0; i<N; i++)
      for(int j=0; j<N; j++)
         for(int k=0; k<N; k++)
            D[i][j] += C1[i][k] * C[k][j];
```

Output Variant 8 – Auto-generated code from Input Code 1 to reuse same data in multiple kernels. We identify whether data need to be send to or from the device.

```
#pragma omp target data map(to:  A[N][N], B[N][N], C[N][N]), \
      map(alloc:  C1[N][N]) map(tofrom:  D[N][N])
  {
#pragma omp target teams distribute parallel for collapse(2)
      for(int i=0; i<N; i++)
         for(int j=0; j<N; j++)
            for(int k=0; k<N; k++)
               C1[i][j] += A[i][k] * B[k][j];
#pragma omp target teams distribute parallel for collapse(2)
      for(int i=0; i<N; i++)
         for(int j=0; j<N; j++)
            for(int k=0; k<N; k++)
               D[i][j] += C1[i][k] * C[k][j];
  }
```



Multi-GPU – Copying a large array. Based upon different input sizes split the computation into multiple devices upon availability. We offload computation to 1, 2 and 4 devices.

This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of two U.S. Department of Energy organizations (Office of Science and the National Nuclear Security Administration) responsible for the planning and preparation of a capable exascale ecosystem, including software, applications, hardware, advanced system engineering and early testbed platforms, in support of the nation's exascale computing imperative.