Porting the Opacity Client Library to a CPU-GPU Cluster Using OpenMP 4.5

Jason S. Kimko[1], Michael M. Pozulp[2], Riyaz Haque[2], Leopold Grinberg[3]
[1] College of William & Mary, [2] Lawrence Livermore National Laboratory, [3] IBM Research

Abstract

This poster exhibits our experience porting the Opacity client library [1] to IBM's "Minsky" nodes [2] using OpenMP 4.5. We constructed a GPU-friendly container class that mimics existing library functionality. We benchmarked our implementation on Lawrence Livermore National Laboratory's (LLNL) RZManta [3], a Minsky cluster. In our benchmarks on a single POWER8 CPU and Tesla P100 GPU, we observed up to 4x speedup including CPU-GPU data transfers, and up to 30x speedup excluding data transfers. Optimizing to reduce register pressure and increase occupancy may improve speedups. Our results demonstrate a successful and beneficial library port to the CPU-GPU architecture.

Introduction

KULL [4] is one of LLNL's multi-physics simulation codes that models high energy density physics applications such as inertial confinement fusion. KULL uses the Opacity client library to obtain opacity data for radiation transport simulations and aims to run on CPU-GPU clusters in the future.

The Opacity client library performs bilinear interpolations and various extrapolations to provide material opacities given density and electron temperature pairs.

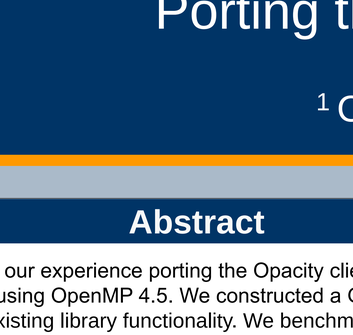

Figure 1. Bilinear interpolation returning P, where x and y are inputs and x_i, y_i, and Q_{ij} are known.

OpenMP is used to achieve a portable, single source code solution, and since version 4.0, it has provided support for GPU offloading through its device constructs. OpenMP 4.5 provides additional features such as the `target enter/exit data` construct, which is used in our implementation.

Since the library runtime requires low FLOPs, most of the required work consists of providing offloading support such that KULL may increase the throughput of library calls (Figure 2).

```
#pragma omp target teams distribute parallel for map(…)
for (i=0; i<n; i++) {
    gpu->Lookup(density[i], temperature[i], &opacity[i])
}
```

Figure 2. Expected kernel structure to maximize throughput, where `Lookup()` performs interpolations and extrapolations.

Lawrence Livermore National Laboratory

Implementation

Interpolants are read and distributed to many classes that often utilize the C++ Standard Template Library and virtual functions, making data transfer from host to device memory and generation of PTX instructions for GPUs more challenging.

We created a GPU-compatible, C-styled container class that directly stores interpolant values and implements methods to maintain original library functionality while minimizing changes to the user code (Figure 3).

```
OP::Enable_GPU(1);  // Enable data mapping underneath existing
                    // function calls during setup
    ⋮
gpu = cpu->Get_GPU_Obj();     // Extract GPU-compatible object
                              // before kernel launch
#pragma omp target teams distribute parallel for map(…)
for (i=0; i<n; i++)
    gpu->Lookup(); // Swap original object for extracted object
    // cpu->Lookup();
```

Figure 3. Changes to user code.

Currently, this port is not optimized and profiling reveals high register pressure and low occupancy (Figure 4), but how these values affect performance and how they can be optimized is beyond the scope of this work.

Grid Size	[128,1,1]
Block Size	[512,1,1]
Registers/Thread	102
Shared Memory/Block	908 B
Efficiency	
Local Memory Overhead	71.8%
Occupancy	
Achieved	24.9%
Theoretical	25%
Limiter	Registers

Grid Size	[128,1,1]
Block Size	[1024,1,1]
Registers/Thread	56
Shared Memory/Block	924 B
Efficiency	
Local Memory Overhead	82.9%
Occupancy	
Achieved	50%
Theoretical	50%
Limiter	Registers

Figure 4. Output provided by NVIDIA Visual Profiler [5] depicting high register pressure and low occupancy for Clang (left) and XL (right).

Results

We ran a driver on an IBM Power System S822LC [2], which features 2 ten-core POWER8 prime CPUs, 4 NVIDIA Tesla P100 GPUs, and NVLink 1.0 interconnects.

Wall times were collected for the `Lookup()` loop on the host using 40 hardware threads over one socket. Running on one P100, timings include and exclude time required to map the query data (Figure 7). Speedup is calculated as $\frac{time_{CPU}}{time_{GPU}}$.

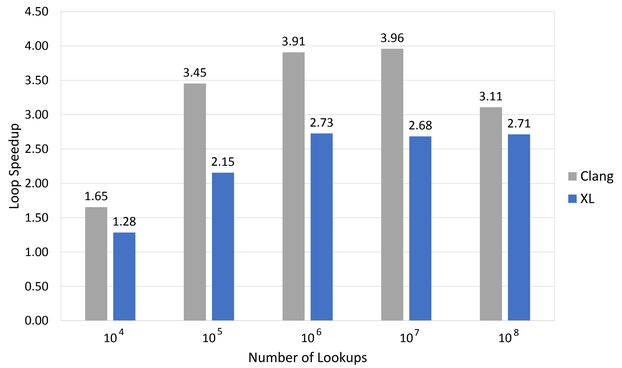

Figure 5. Loop Speedup vs. Number of Lookups, including data mapping

Results (cont.)

Currently, KULL needs to map query data before calling `Lookup()`. Accounting for this data movement, we see speedups from 1.3x to 4x depending on the number of lookups and the compiler (Figure 5).

In the future, we expect KULL to initialize the query data in GPU memory. Thus, this data transfer will become unnecessary, resulting in speedups from 1.4x to 30x (Figure 6).

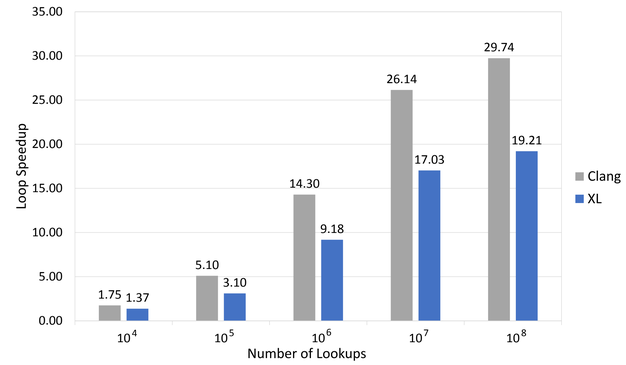

Figure 6. Loop Speedup vs. Number of Lookups, excluding data mapping

```
#pragma omp target data map(…)      // data movement
{
double wtime = omp_get_wtime();
#pragma omp target teams distribute parallel for
for (i=0; i<n; i++)
    gpu->Lookup(density[i], temperature[i], &opacity[i])
wtime = omp_get_wtime() - wtime;
}
```

Figure 7. Example structure of timing code that *excludes* time for data movement.

Conclusion

With the ever increasing importance of the CPU-GPU architecture, many scientific applications will want to claim the advantages therein. With an unoptimized OpenMP 4.5 port, which features at its core a C-styled container class, performance gains were still observed and very attainable, peaking at 4x when including CPU-GPU data transfers and 30x when all of the data can be kept in GPU memory. We are integrating the ported library into KULL, and we hope to experiment with different runtime configurations and code structures to optimize the utilization of GPU resources.

References

[1] LLNL Support Libraries. Web Page. Accessed Wed Jul 19, 2017. https://wci.llnl.gov/simulation/support-libraries
[2] A.B. Caldeira, V. Haug and S. Vetter, "IBM Power System S822LC for High Performance Computing Introduction and Technical Overview," IBM Redbooks, October 2016.
[3] LLNL RZManta. Web Page. Accessed Fri 21, 2017. https://hpc.llnl.gov/hardware/platforms/RZManta
[4] J.A. Rathkopf, D.S. Miller, J.M. Owen, L.M. Stuart, M.R.Zika, P.G. Eltgroth, N.K. Madsen, K.P. McCandless, P.F. Nowak, M.K. Nemanic, N.A. Gentile, N.D. Keen and T.S. Palmer, "KULL: LLNL's ASCI Inertial Confinement Fusion Simulation Code," January 2000.
[5] NVIDIA Visual Profiler. Web Page. Accessed Tue 25, 2017. https://developer.nvidia.com/nvidia-visual-profiler

National Nuclear Security Administration

LLNL-POST-735790

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344