MACC: Mercurium ACCelerator Model

Guray Ozen, Eduard Ayguade, Jesus Labarta
Barcelona Supercomputing Center (BSC-CNS), Barcelona, Spain
Universitat Politecnica de Catalunya (UPC-BarcelonaTECH)
{name.surname}@bsc.es

Abstract- *GPU Offloading is emergent programming model. OpenMP includes in its latest 4.0 specification the accelerator model. In this paper we present a newly implementation of this specification while generationg "native" GPU code in the OmpSs programming model developed at the Barcelona Supercomputing Center. Focused on targeting NVIDIA GPUs, our work is based on an OpenMP 4.0 implementation in the Mercurium source-to-source compiler infrastructure referred MACC[1] (Mercurium ACCelerator compiler). Finally, the paper is also discuss challenges od code generation for GPUs.*

I. Introduction

The use of accelerators has been gaining popularity in the last years due to their higher peak performance and performance per Watt ratio when compared to homogeneous architectures based on multicores. However, the heterogeneity they introduce (in terms of computing devices and address spaces) makes programming a difficult task even for expert programmers.

The OmpSs[2] proposal has been evolving during the last decade to lower the programmability wall raised by multi/many-cores, demonstrating a task-based data flow approach in which offloading tasks to different number and kinds of devices, as well as managing the coherence of data in multiple address spaces, is delegated to the runtime system. This paper presents the latest implementation of OmpSs which includes partial support for the accelerator model in OpenMP 4.0 specification. We adopted those functionalities that are necessary to specify computational kernels in a more productive way.

II. MACC: Mercurium ACCelerator Compiler

A new compilation phase (MACC is abbreviation for "Mercurium Accelerator Compiler".) has been included in the Mercurium[3] compiler supporting the OpenMP 4.0 accelerator model with the OmpSs runtime. MACC takes care of kernel configuration, loop scheduling and appropriate use of the memory hierarchy for those tasks whose *device* is set to *acc* in the *target* clause. Some of the OpenMP 4.0 directives for accelerator *target data*, *target update* directives and *map* clause) are simply ignored because we delegate their functionality to the runtime system. Others have been extended to better map with the OmpSs model or to provide additional functionalities.

A. Kernel Creation

MACC offloads all code inside of *target* directive into GPU. Basically it generates kernel code and when code reaches that point, it invokes device kernel. Since MACC is based OmpSs, it behaves *target* directive as a *task*. It is novel approach of MACC and as well as according to OpenMP 4.1 drafts this specification will bring that approach soon. By this approach, MACC could put together advantages of *task* and *target* at the same time. The main advantages of that usage different from standard OpenMP 4.0 are device-to-device transfer became available and MACC could schedule multi-gpu task.

B. Kernel Thread Hierarchy Generation

When generating kernel code MACC needs to decide: 1) the dimensionality of the resources hierarchy (one-, two- or three-dimension teams and threads) and 2) the size in each dimension (number of teams and threads). In order to support the organization of the threads in multiple dimensions MACC allows the nesting of *parallel for* directives inside a *target* region.

C. Loop Directives and Data Clauses

MACC performs a cyclic mapping of loop iterations and tries to eliminate redundant "one-iteration" loops and simplifies increment expressions for induction variables in order to improve kernel execution time. MACC also makes use of shared memory for threads in a team based on the specification of *private* and *firstprivate* data structures in the *distribute* directive, so that each team allocates a private copy in its own shared memory.

However, for very large private arrays this is not possible to apply. For these cases we have implemented tree new clauses (*dist_private*, *dist_firstprivate* and *dist_lastprivate*); with these clauses and the chunk size provided in the *dist_schedule(static,chunk_size*) clause in the *distribute* directive the compiler just allocates a portion of the arrays to each team and performs the copies.

D. Reduction

Reduction is a very important of GPU parallelism as it was in the CPU side. When generating reduction code, MACC uses different reduction algorithms depending upon primitive type in order to obtain best speedup. It uses __shfl intrinsic (by this instruction it can obtain significant speedup) and doing reduction by using shared memory for thread level reductions. As for Thread Block Level Reduction, it uses atomic operations of GPUs and calling second kernel approaches.

III. Performance Evaluation

A. MACC Reduction on Jacobi

Jacobi is a simple iterative program to get an approximate solution of a linear system A*x=b. The structure of the code is shown in Figure-1, with two different annotations that correspond to two versions.

OpenACC(non-optimised)	MACC (auto Optimized)

while (…) {
#acc kernels\
copyin(u) copyout(uold)
for (…)
// <..computation..>
 	
#acc kernels \
copyout(u) copyin(uold)\
reduction(+:err)
for (…)
// <..computation..>
}
	while (…) {
	#omp target device(acc)
	#omp task in(u) out(uold)
	#omp parallel for
	for (…)
	// <..computation..>
	#omp target device(acc)
	#omp task out(u) in(uold)
	#omp parallel for \
	reduction(+:err)
	for (…)
	// <..computation..>
	}

Figure-1

Figure-2 shows benchmark between 3 versions. There are two OpenACC(HMPP) versions baseline and optimized. Optimized means when programmers use explicitly data directives, OpenACC keeps data into device. Since MACC behaves code will be offloaded as a task, programmers doesn't need to indicate any extra directives.

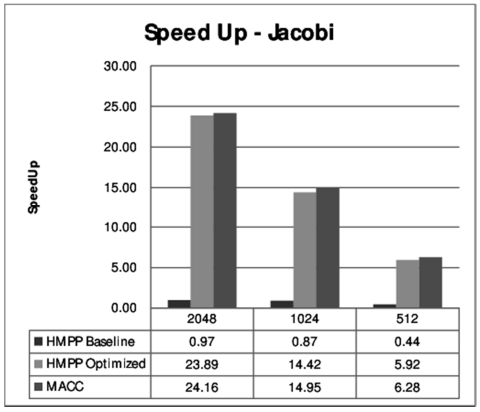

Figure-2

B. MACC Shared Memory performance on DG-kernel

DG is a kernel version of a climate benchmark developed by National Center for Atmospheric Research dg-kernel. The code consist single target. Data directionalities and characteristics are indicated at figure-3. By using new MACC's clauses, partly using shared memory is enabled. Additionally array B could be taken back from shared memory. This usage corresponds opt-1 at figure-4.

```
#omp target device(acc)
#omp task in(A[0:SMALL],C[0:HUGE]) inout(B[0:HUGE)
out(D[0:BIG])
#omp teams distribute parallel for first_private(A)
    dist_first_private([CHUNK]C) \
    dist_first_last_private([CHUNK]B)
 for(...)
  <<..Computation..>>
```

Figure-3

Besides, when MACC build kernel hierarchy equals to iteration size, it removes redundant for iteration in order to avoid thread-divergence. At figure-4 this optimization denotes opt-2.

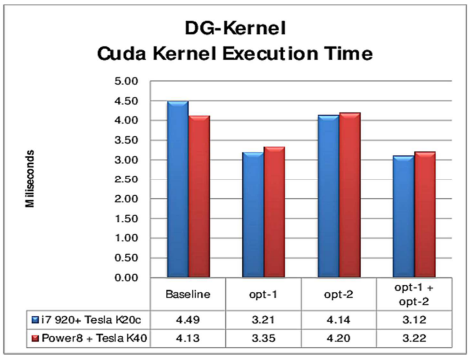

Figure-4

C. MACC multi-GPU scheduling on NAS-CG

The last code we have selected for the experimental evaluation in this paper is NAS CG. The main computational part of the application contains several loops that can be made tasks and offloaded to a device or executed on the host. To execute this loop we want to use the two GPUs available in the node. We could enable multi gpu by dividing most expensive iteration into chunks, since nanox can schedule them automatically. Figure-5 shows comparison graph.

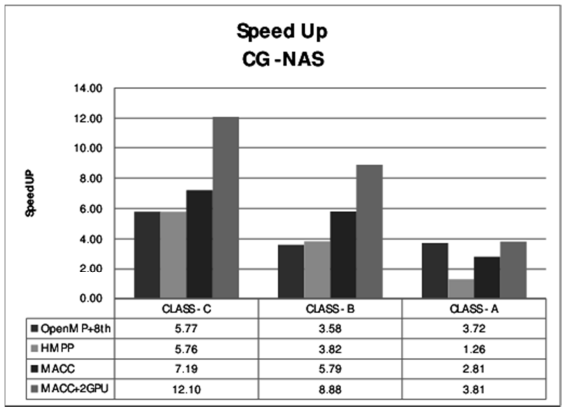

Figure-5

REFERENCES

[1] Guray Ozen, Eduard Ayguadé, Jesús Labarta: On the Roles of the Programmer, the Compiler and the Runtime System When Programming Accelerators in OpenMP. IWOMP 2014:

[2] The OmpSs programming model. http://pm.bsc.es

[3] Mercurium Compiler. http://pm.bsc.es/mcxx