Simplifying HPC Application Development with OpenMP Cluster and Unified Memory

Jhonatan Cléto[1], **Hervé Yviquel**[1], **Marcio M. Pereira**[1], **Guido Araújo**[1]

[1]Institute of Computing – State University of Campinas (UNICAMP)

`j256444@dac.unicamp.br`

Abstract. *As accelerators such as GPU and FPGA become more common in HPC systems, programming for these systems becomes more challenging due to, for example, the additional layer of memory management. This paper presents an extension to the OpenMP Cluster that integrates CUDA's Unified Memory management. Evaluation using a synthetic benchmark reveals that while this extension simplifies the development of GPU-based OMPC applications, further optimization is required to reduce its impact on performance.*

1. Introduction

High Performance Computing (HPC) systems are becoming increasingly heterogeneous [Meuer et al. 2014]. To deliver the computational capacity required by modern (scientific and machine learning) applications, the number of accelerators (e.g., GPU and FPGA) on a system node has increased. This heterogeneous architecture increases the complexity of HPC application programming, which already suffered from the combination of several languages and parallel and distributed programming models.

OpenMP Cluster (OMPC) [Yviquel et al. 2022] is an innovative programming model for HPC clusters that build upon task parallelism. It effectively extends the popular OpenMP API, allowing for the balanced distribution of computationally intensive scientific tasks across nodes in an HPC cluster. This enables users to easily take advantage of the power and scalability of HPC clusters without requiring specialized knowledge or expertise in distributed computing. OMPC enables the utilization of various APIs for tasks on accelerators. However, it doesn't handle buffers like variables and arrays allocated by these APIs. Consequently, programmers must manually transfer data between APIs for use in other tasks, increasing the complexity of programming the application.

In this work, we developed an extension to OMPC, making it capable of managing address spaces allocated in both CPU and GPU. This extension aims to facilitate the programming of OMPC applications that use GPUs.

2. Background

Memory management is a challenge in programming heterogeneous architectures due to complexities in managing memory regions between cluster nodes and the addition of accelerator memory. Heterogeneous nodes typically have main memory (RAM) for host processors and device memory for accelerators. While accelerator APIs include data transfer primitives, it is the programmer's responsibility to use them when needed. Some parallel programming models offer mechanisms to abstract different addressing spaces into a single virtual space. The data transfer between physical addresses is done behind the scenes by the programming model's runtime, taking this responsibility off the programmer.

3. OMPC Unified Memory Extension

NVIDIA's Compute Unified Device Architecture (CUDA) is a parallel computing platform that allows developers to use Graphics Processing Units (GPUs) for general computing. It features Unified Memory, which combines different memory spaces into one accessible by any processor in the system.

The OMPC's Data Management (DM) module ensures data consistency across cluster nodes by examining task dependencies and tracking buffer locations with data maps for each node. When a task is assigned to a node, DM checks these maps and handles the necessary data forwarding. OMPC also has an event handling system that oversees memory allocation and removal, as well as data submission, retrieval, and forwarding between nodes.

We've modified OMPC's allocation and removal events in our extension to manage buffers in both main and device memory. We've shifted main memory allocations to the CUDA Unified Memory system. This allows all memory addresses managed by OMPC to be accessible to all CPUs and GPUs in the cluster nodes, enabling data transfer between different nodes and devices without the need for explicit copying. This is facilitated by CUDA's management of data transfers between host and device.

It is important to note that the modification in OMPC is related only to the data allocation and transfer mechanism used by the runtime. The extension does not modify the programming model. However, buffers allocated only in the main memory or in a device's memory are reallocated to the unified address space when they are used in target regions or mapped in the map clause of the OpenMP target directives. In the following sections of this paper, this extended version is referred to as OMPC+UM.

4. Experiments

To assess the performance of OMPC+UM, we conducted experiments that compared it to the original OMPC. We selected two applications for testing and ran each experiment ten times, reporting the average result. The first application, Task Bench, is a benchmarking framework that allows the comparison of parallel and distributed programming models based on tasks [Slaughter et al. 2020]. Task Bench employs a graph abstraction that uses vertices to represent tasks and edges to represent data dependencies between them. For the experiments, we implemented two benchmark versions [1] for the original OMPC and OMPC+UM. Both versions perform computational tasks exclusively on the GPU.

The experiment involved a $N \times N$ square matrix multiplication application, computing $C = AB$ by partitioning matrices into blocks and calculating matrix C blocks using $C_{qr} = \sum_{i=1}^{s} A_{qi}B_{ir}$, where q, r, and s are valid partitions over the rows and columns of matrices A and B, respectively. Block multiplications were implemented using cuBLAS [2] library as tasks for cluster nodes. We compared two application versions: one using the original OMPC with explicit buffer copies and the other using OMPC+UM without explicit buffer copies. Comparing the two versions allowed us to evaluate the impact of OMPC+UM on performance in a scenario where data transfers between CPU and GPU are critical.

[1] Available at https://gitlab.com/ompcluster/task-bench/

[2] The cuBLAS library provides optimized implementations for many common linear algebra operations on GPUs.

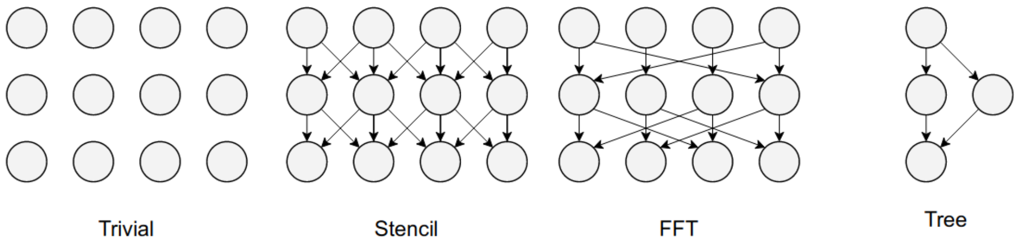

Figure 1. Task Bench dependency types used in experiments.

4.1. Environment Setup

The experiments were performed on the Santos Dumont supercomputer at LNCC [LNCC 2023]. The configuration consisted of 5 nodes with Intel Xeon Cascade Lake Gold 6252 CPUs, 384 GB RAM, and Nvidia Volta V100 GPUs with 32 GB of VRAM. To ensure consistency and reproducibility, Singularity containers were used. The software stack included Clang v14.0, NVCC v11.2, OMPC v14.9.0, MPICH v3.4.2, and UCX v1.11.2. The experiments were managed and scheduled by the SLURM task manager.

4.2. Experiments Results

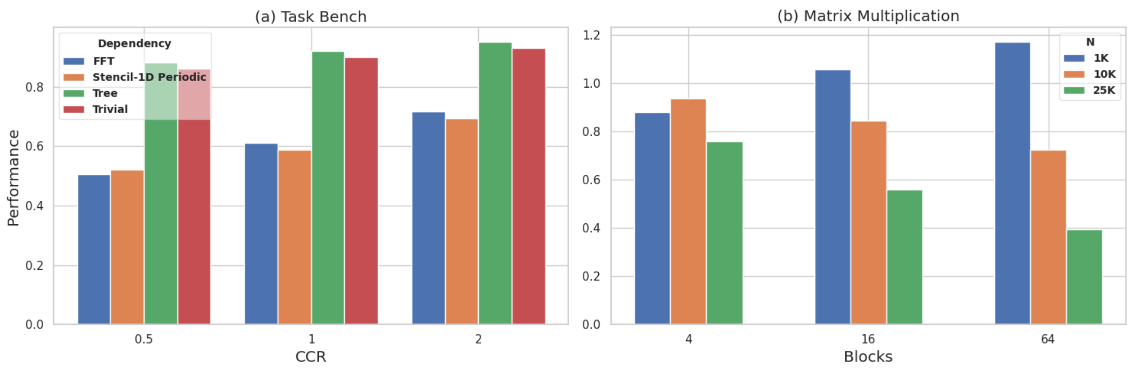

Figure 2. Performance relative to OMPC of OMPC+UM in (a) GPU computing benchmarking for some task dependencies and (b) Block Matmul Application.

Figure 2 (a) compares the performance of OMPC+UM and the original OMPC for four Task Bench graph configurations, as shown in Figure 1. This comparison considers the data exchanged between tasks and CCR[3]. The tests ran 10 million iterations (500 ms per task) on a task graph with a width of 4 and a height of 16. For graphs with fewer dependencies like Tree and Trivial, OMPC+UM achieves over 80% of the original OMPC's performance for all three CCR values. This performance drop is due to the overhead of CUDA runtime managing Unified Memory buffers. The performance drop is more significant for graphs with more dependencies like FFT and Periodic Stencil-1D, and it further decreases with higher communication costs.

[3]Communication to Computation Ratio measures the relationship between computation and communication costs.

Figure 2 (b) displays the matrix multiplication application experiment results, comparing OMPC+UM and OMPC for three square matrix sizes. Performance is shown as a function of block numbers from matrix partitionings (4, 16, and 64 blocks). For $N = 1K$, performance gains and speedup were observed with increased blocks, but for $N = 10K$ and $N = 25K$, performance loss occurred as block numbers increased, with the worst case ($N = 25K$ and 64 blocks) being below 40% of the original OMPC.

Our experiments suggest that the performance loss of OMPC+UM is largely due to the frequent data transfers between the main memory and GPU memory. These transfers occur when tasks are scheduled, and in the case of Task Bench, during a buffer validation on the CPU that takes place before the execution of each task. We speculate that these transfers could be minimized with an optimized OMPC+UM implementation that reduces the number of data transfers between the CPU and GPU.

5. Conclusion

In this paper, we present OMPC+UM, an extension to OMPC that simplifies programming OMPC applications with GPUs by managing buffers in both CPU and GPU and handling data transfers between host and device. However, our experiments reveal that OMPC+UM shows reduced performance compared to explicit data copy versions, particularly as data transfers increase. Therefore, further optimization of OMPC+UM is necessary, such as minimizing data transfers between CPU and GPU. Our implementation provides valuable insights into OMPC's behavior with other programming models, highlighting the potential of integrating OMPC and CUDA for accelerating scientific applications in heterogeneous HPC clusters.

Acknowledgments

This work is supported by Petrobras under grant 2018/00347-4. We would also like to thank the LNCC for granting the computational resources at the Santos Dumont supercomputer that were crucial to perform the experiments for this project.

References

LNCC (2023). Santos Dumont (SDumont) - Bull Sequana X1000, Xeon Gold 6252 24c 2.1GHz, Mellanox Infiniband EDR, Nvidia Tesla V100 SXM2.

Meuer, H. W., Strohmaier, E., Dongarra, J., and Simon, H. D. (2014). *The TOP500: History, Trends, and Future Directions in High Performance Computing*. Chapman & Hall/CRC, 1st edition.

Slaughter, E., Wu, W., Fu, Y., Garcia, N., Kautz, W., Marx, E., Morris, K. S., Cao, Q., Bosilca, G., et al. (2020). Task bench: A parameterized benchmark for evaluating parallel runtime performance. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–15. IEEE.

Yviquel, H., Pereira, M., Francesquini, E., Valarini, G., Gustavo Leite, P. R., Ceccato, R., Cusihualpa, C., Dias, V., Rigo, S., Sousa, A., and Araujo, G. (2022). The OpenMP Cluster Programming Model. *51st International Conference on Parallel Processing Workshop Proceedings (ICPP Workshops 22).*