Design of a Hybrid MPI-CUDA Benchmark Suite for CPU-GPU Clusters

Tejaswi Agarwal and Michela Becchi
University of Missouri - Columbia
tejaswi.agarwal2010@gmail.com, becchim@missouri.edu

ABSTRACT

In the last few years, GPUs have become an integral part of HPC clusters. To test these heterogeneous CPU-GPU systems, we designed a hybrid CUDA-MPI benchmark suite that consists of three communication- and compute-intensive applications: Matrix Multiplication (MM), Needleman-Wunsch (NW) and the ADFA compression algorithm [1]. The main goal of this work is to characterize these workloads on CPU-GPU clusters. Our benchmark applications are designed to allow cluster administrators to identify bottlenecks in the cluster, to decide if scaling applications to multiple nodes would improve or decrease overall throughput and to design effective scheduling policies. Our experiments show that inter-node communication can significantly degrade the throughput of communication-intensive applications. We conclude that the scalability of the applications depends primarily on two factors: the cluster configuration and the applications characteristics.

Keywords

Benchmark; CUDA-MPI; clusters; GPU

1. INTRODUCTION

GPUs have been widely adopted on standalone systems and are now finding their way into high performance clusters. However, there has been little work on developing benchmark suites consisting of hybrid MPI-CUDA applications which could give insights to administrators on how to use CPU-GPU clusters effectively. Our benchmark applications use MPI to distribute work among nodes and CUDA to offload computation to GPUs. These applications alternate inter-node communication and CPU-GPU computation phases (including data transfers between CPU and GPU). As far as CPU-GPU communication is concerned, we tested 3 alternatives: (1) use of unpinned host memory, (2) use of pinned host memory and (3) use of CUDA-Aware MPI.

2. BENCHMARKS

Matrix Multiplication (MM) and Needleman Wunsch (NW): MM and NW [2] are iterative applications that work on *I* datasets of size *D*. At the beginning of every iteration, each dataset is uniformly distributed among the *P* available processes using the `MPI_Scatter` primitive. Each process is executed on a single GPU. For every iteration, each process transfers *D/P* portions of the dataset from CPU to GPU, performs computation and copies the result from GPU to CPU. At the end of each iteration, the results are transferred to the root process using the `MPI_Gather` primitive, which also provides implicit barrier synchronization among processes. The size of the dataset and number of iterations are configurable. In MM and NW the inter-process communication dominates the computation time, making them communication sensitive. *Amortized time−bandwidth DFAs (ADFA):* ADFA is a technique to compress Deterministic Finite Automata that accept large sets of regular expressions. In our hybrid MPI-CUDA ADFA implementation we maintain a work queue containing *N* input DFA files. In every iteration, *P* DFAs are removed from the work queue and each of them is assigned to a different MPI process for ADFA compression. At the end of each iteration, an `MPI_Gather` primitive transfers the result of ADFA compression into a single array to the root process. The application terminates once all DFAs from the work queue have been processed. ADFA is designed similar to the above applications, except that it does not involve `MPI_Scatter` at the beginning. The inter-process communication is limited: differently from MM and NW, ADFA is compute-intensive.

Table 1: Summary of benchmark characteristics.

	MM	NW	ADFA
Number of Processes and dataset size	Configurable	Configurable	Configurable
Communication Type	MPI_Scatter, MPI_Gather,	MPI_Scatter, MPI_Gather	MPI_Gather
Sensitivity	Communication sensitive	Communication sensitive	Computation sensitive
Size of MPI Transfer per iteration	800 MB	100-200 MB	8 MB
Number of iterations	2-8	15-30	1-2

3. RESULTS

Fig. 1 shows the performance obtained by configuring the 3 applications as shown in Table 1. As can be seen, MM and NW fail to scale over multiple nodes since the inter-process communication dominates the computation time. On the other hand, ADFA scales over multiple nodes, though not linearly. We observed similar trends on a low-end cluster with 8 nodes. Thus, heterogeneity within the cluster and the type of application play a significant role in deciding scheduling policies for a cluster administrator. For example, if the communication dominates the execution time of the application, it is preferable to adopt a policy that co-locates the application processes on one node.

Next, we want to evaluate the impact of using pinned memory and CUDA-aware MPI on the performance of the considered workloads.

PACT'14, August 24–27, 2014, Edmonton, AB, Canada.
ACM 978-1-4503-2809-8/14/08.
http://dx.doi.org/10.1145/2628071.2671423

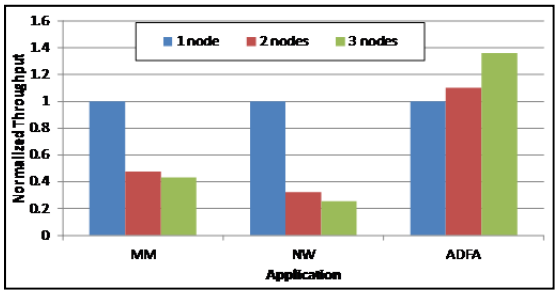

Fig. 1: Performance of MM, NW and ADFA on 1, 2 and 3 nodes. The throughput has been normalized over the 1 node implementation.

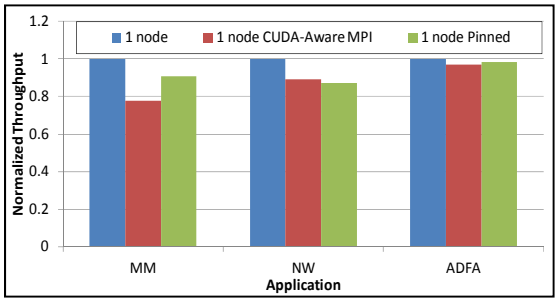

Fig. 2: Comparison of the unpinned memory, the pinned memory and the CUDA-Aware MPI versions on 1 node. The data are normalized over 1 node unpinned memory version.

Pinned memory is non-pageable host memory. When data are allocated using the `cudaMalloc` primitive, they are stored in pageable (or unpinned) memory on the host. However, data transfers between unpinned memory and device memory incur an overhead. When a CPU-GPU data transfer is required, since the GPU cannot directly access data from the pageable memory, the CUDA driver and OS first create a temporary pinned memory buffer and copy the data from pageable memory to this buffer, and then they perform the final data transfer from pinned memory to the device. Directly allocating pinned memory by the programmer avoids the cost of transferring the data between pageable and pinned memory [3]. However, allocation of too much pinned memory affects performance as the OS normally limits the amount of non-pageable memory.

CUDA-Aware MPI is a programming model that facilitates the programmability of hybrid MPI-CUDA applications. In standard MPI, only host memory pointers can be passed to MPI functions. However, with CUDA-Aware MPI GPU buffers can directly be passed to MPI functions. Other advantages of using CUDA-Aware MPI are that message transfer operations can be pipelined and GPUDirect can be utilized [4]. This provides programmer efficiency with reduced MPI code.

Figure 2 presents a performance comparison between the unpinned memory, the CUDA-Aware MPI and the pinned memory versions of the considered applications. The results are collected on 1 node and normalized over the performance reported by the unpinned memory version. We observe that, since MM and NW are data intensive, allocation of pinned memory for these applications reduces performance. Since ADFA is less data intensive, the overall performance of pinned ADFA is better as compared to the other two applications. In general, we observe that CUDA-Aware MPI improves programmer efficiency and it does not affect performance significantly.

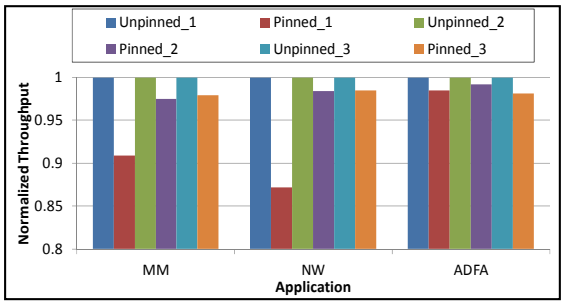

Fig. 3: Comparison between the pinned and unpinned memory versions on 1, 2 and 3 nodes. *Pinned_j*/*Unpinned_j* indicates that the experiment is run on *j* nodes. The data are normalized over the 1 node, unpinned memory implementation.

Figure 3 shows performance results for pinned and unpinned memory of the 3 applications over 1, 2 and 3 nodes. The results are normalized over unpinned memory version for all 3 nodes individually. As can be seen, pinned memory affects performance more significantly in MM and NW as compared to ADFA as these are data intensive applications. In addition, allocating too much pinned memory affects the application performance.

4. CONCLUSION

We designed an MPI-CUDA benchmark which can be useful for administrators to test scalability of their applications on HPC clusters. By changing the user-input, the applications can be configured to include longer- or shorter-running kernels, varying amount of inter-process communication and CPU duration. In the poster, we show more extensive results with different datasets, and hardware and software configurations. Our results show that application performance depends mainly on two factors: application characteristics (communication vs compute intensive) and compute capability of the nodes. We also conclude that, when dataset sizes are small, memory intensive applications benefits from the use of pinned memory. However, in case of large dataset sizes, allocating too much pinned memory affects application performance. In future, we will leverage this study to propose scheduling policies to effective manage hybrid workloads on heterogeneous HPC clusters.

5. ACKNOWLEDGEMENT

This work has been supported by NSF award CNS-1216756 and by a gift from NEC Laboratories America and equipment donations from Nvidia Corporation.

REFERENCES

[1] M. Becchi and P. Crowley, "A-DFA: A Time- and Space-Efficient DFA Compression Algorithm for Fast Regular Expression Evaluation," *ACM TACO,* vol. 10, no. 1, pp. 1-26, 2013.

[2] S. B. Needleman, and C. D. Wunsch, "A general method applicable to the search for similarities in the amino acid sequence of two proteins," *J. of Molecular Biology,* vol. 48, no. 3, pp. 443-453, 1970.

[3] How to Optimize Data Transfers in CUDA C/C++, *http://devblogs.nvidia.com/parallelforall/how-optimize-data-transfers-cuda-cc.*

[4] An Introduction to CUDA-Aware MPI, *http://devblogs.nvidia.com/parallelforall/introduction-cuda-aware-mpi.*