Spacehog: Evaluating the costs of dedicating resources to in situ analysis

Rebecca Kreitinger
University of New Mexico
Albuquerque, New Mexico
rkreitinger@unm.edu

Scott Levy, Kurt B. Ferreira, Patrick Widener
Center for Computing Research
Sandia National Laboratories
Albuquerque, New Mexico
sllevy|kbferre|pwidene@sandia.gov

ACM Reference format:
Rebecca Kreitinger and Scott Levy, Kurt B. Ferreira, Patrick Widener. 2017. Spacehog: Evaluating the costs of dedicating resources to in situ analysis. In *Proceedings of SC17 conference, Denver, Colorado USA, November 2017 (SC17),* 2 pages.
DOI: 10.475/123_4

1 INTRODUCTION

Making sense of the enormous volumes of data generated by scientific simulation codes requires domain scientists to use sophisticated analysis tools. However, the time and energy costs of exchanging data between simulation and analytics codes through a filesystem is becoming prohibitive. Workflows that use *in situ* analytics to process output data are therefore becoming more common. *in situ* techniques require sharing computational resources between simulation and analysis. Two common approaches for sharing resources are time-sharing and space-sharing. Time-shared simulation and analysis use the same set of resources; each is granted intervals of exclusive access to the resources before yielding (or being forced to yield) to the other. Space-sharing partitions the computational resources allocated to a job, and both the simulation and the analysis are given exclusive use of a subset of those resources. Some shared resources cannot be (easily) partitioned (cache/DRAM memory; memory, network and filesystem bandwidth), and so simulation and analysis tasks may possibly contend for those resources. In this poster, we examine how space-sharing HPC applications with a set of analytics tasks impacts application time-to-solution.

The contributions of this poster include: (1) a description of a software library that facilitates the evaluation of the performance impact of space-sharing MPI applications with analysis tasks; (2) an examination of how space-sharing MPI applications with different analysis task may impact their time-to-solution; (3) an analysis of how the execution environment influences the time-to-solution of MPI applications that are space-shared with analytics tasks.

2 APPROACH

We developed a software library, `libspacehog`, to model analytics tasks. `libspacehog` uses the MPI profiling interface to intercept all MPI calls transparently, allocating a user-configurable number of processes to run analytics tasks and running an HPC application on the remaining processes. The profiling interface allows `libspacehog` to run the application on a subset of the resources of the allocation while concealing this fact from the application. We created nine different analysis tasks, shown in Table 1: five microbenchmarks that exercise a single resource and four analysis proxies that mimic analytics tasks. `libspacehog` has several configuration settings, including: (i) the fraction of processes to reserve for the analytics tasks; (ii) which of the analytics tasks to run on these processes; and (iii) task-specific parameters for analytics tasks. When the application completes, `libspacehog` intervenes to shutdown the analytics tasks before relinquishing the allocation. Our study does not model explicit coupling between the analytics tasks and the application.

3 EXPERIMENTAL RESULTS

We ran three sets of experiments, each set examining a different dimension of how simulation workloads might be space-shared with analysis codes (results in poster Figures 1-3). We ran these experiments for each of the five workloads described in poster Table 2P. Each experimental workload used 84 application processes. We conducted our experiments on two systems, *Volta* and *Chama* (poster Table 3P). We collected results on *Volta* except where otherwise specified. All results are normalized to the application time-to-solution obtained by space-sharing with the `busy wait` analytics task.

3.1 Impact of Computational Kernel

Applications were most significantly impacted by the `copy`, `bursty copy`, and `allreduce` microbenchmarks, with HPCG most affected. These results suggest that HPCG is sensitive to contention for both memory bandwidth and communication resources. We also observed that miniAMR is sensitive to contention for communication resources. Impact is more modest for the other three applications.

These results show that time-sharing applications with our microbenchmark tasks (e.g., `copy`, `bursty copy`, and `allreduce`) can significantly increase application time-to-solution. However, time-sharing our applications with the file I/O microbenchmark task or any of the analysis tasks (e.g., `parallel FFT`, `parallel k-means`, `voronoi`) has a very modest impact on application performance.

3.2 Impact of Process Density

Poster Figures 1.1-1.5 show the results of changing the total number of processes running on each node. For the majority of the experiments, the impact of mapping 24 processes per node and

SC17, Denver, Colorado USA

DOI: 10.475/123_4

Analysis Task	Description	
Microbenchmark Tasks	Busywait	Just checks for simulation completion
	Allreduce	Performs a sequence of `MPI_Allreduce` operations.
	Copy	Continuously copies data from one memory allocation to another.
	Bursty Copy	Alternates intervals of sleep and intervals of copying memory.
	File I/O	Analysis tasks write random data to a POSIX file
Proxy Tasks	FFT	Parallel computation of the fast Fourier transform (FFT) on an array of random data
	k-means Clustering	Parallel clustering of random data
	Pi	Monte carlo computation of π
	Voronoi Tessellation	Parallel partition of a plane based on randomly selected points

Table 1: Descriptions of the analysis tasks used.

48 processes per node is modest. Using 24 processes per node, all processor cores are used, but only one thread runs per core. At 48 processes per node, both hyperthreads on each core are used. Reducing the number of processes per node to 12 significantly reduces the performance impact of the analysis workloads. This is consistent with our expectations since in this configuration half of the processor cores are entirely idle. As a result, there is much less contention for shared resources. An outlier is the set of experiments run with miniAMR and the `allreduce` benchmark task: in these, the performance impact of the analysis workload grows as the number of processes per node decreases. This suggests that contention for communication resources between miniAMR and the `allreduce` task causes performance degradation.

3.3 Impact of Number of Analysis Processes

Figures 2.1-2.5 of the poster show the results of changing the total number of analysis processes. In most cases, increasing the number of analysis processes has a modest impact on application performance. However, the `copy` and `bursty copy` microbenchmarks present a striking exception. Their results show that increasing the number of analysis processes to 28 significantly increases their impact on the application time-to-solution. This happens because increasing the number of analysis tasks causes more contention for memory bandwidth and potentially more cache pollution. Each application's memory usage characteristics dictate how sensitive it is to this sort of contention. For example, although the LAMMPS problems are relatively insensitive to memory bandwidth contention, significant performance degradation begins with 28 analysis processes executing the `copy` and `bursty copy` workloads.

3.4 Impact of System Architecture

Figures 3.1-3.5 of the poster show the results of running the experiments on different systems (*Volta* vs. *Chama*). In most cases, the differences between results from the two systems are relatively small. One exception is that there appears to be more contention for memory bandwidth on *Volta* than on *Chama*. The `copy` and `bursty copy` microbenchmarks tend to have a much larger performance impact on *Volta* than on *Chama*. On the other hand, there appears to be more contention for network resources on *Chama*. The `allreduce` task tends to degrade performance more significantly on *Chama* than on *Volta*.

4 RELATED WORK

Several existing studies examine the impact of resource contention on application performance, *see e.g.*, [2] (explicitly induced memory contention in applications to characterize memory requirements), [3] (memory contention with RDMA transfers), [1] (contention from network traffic from independent jobs). These existing studies consider resource contention between *independent* processes. In contrast, we examine resource contention between *inter-dependent* processes. For space-shared *in situ* analytics, general purpose strategies for reducing contention (e.g., improving scheduling decisions) may not be viable; more sophisticated approaches tailored to these composed workloads may be required. Moreover, instead of focusing our analysis on a detailed analysis of a particular resource (e.g., memory, network resources) we study how complete computational workloads might result in resource contention with HPC workloads.

5 CONCLUSION

As *in situ* analytic techniques become preferred tools in computational science workflows, the designers of such workflows need finer-grained understanding of the performance effects of those techniques. We have examined here how space-sharing resources between HPC applications and analytics tasks affects appliction time-to-solution. Our results indicate that contention for memory bandwidth is a greater concern than simply the number of analysis processes or overall process density. Application characteristics are also important; we found that reducing total process densities can have greater impact on a network-bandwidth-sensitive code like miniAMR.

REFERENCES

[1] Abhinav Bhatele, Kathryn Mohror, Steven H Langer, and Katherine E Isaacs. 2013. There goes the neighborhood: performance degradation due to nearby jobs. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. ACM, 41.

[2] Marc Casas and Greg Bronevetsky. 2014. Active measurement of memory resource consumption. In *Parallel and Distributed Processing Symposium, 2014 IEEE 28th International*. IEEE, 995–1004.

[3] Taylor Groves, Ryan E Grant, and Dorian Arnold. 2016. NiMC: Characterizing and eliminating network-induced memory contention. In *Parallel and Distributed Processing Symposium, 2016 IEEE International*. IEEE, 253–262.