Improving data reuse in co-located applications with progress-driven scheduling

Kavitha Chandrasekar[†] Bala Seshasayee[‡] Ada Gavrilovska[†] Karsten Schwan[†]

[†] *Georgia Institute of Technology*, [‡] *Intel Federal*

Abstract

Exascale computing environments – with their large compute and memory capacities – afford a usage model where multiple applications are run concurrently, to utilize the resources effectively. A common scenario in such a setting is for a high-end producer simulation application to run alongside a consumer data analytics kernel, continually inspecting the former's output data to perform *in-situ analysis* on it. As these simulations are expected to operate on a huge volume of data, increasing the data reuse between the applications is expected to yield significant energy savings due to decreased data movement requirements, which is a significant contributor to the overall energy consumption of future Exascale systems. While applications can be developed with the explicit goal of maximizing data reuse, applying this technique to existing applications with data sharing presents a practical challenge, since the runtime needs to be aware of the data dependence requirements between the independently developed applications in order to enable reuse. In this paper, we explore the utility of a *progress-based* approach to application scheduling for maximizing data reuse. To extract progress notions, in the current prototype the runtime relies on programmed application *hints*, obtained through simple demarcation of algorithmic steps in the application, thus requiring non-invasive changes. More generally, we envision that progress metrics can be derived through compiler-assisted technology or from other information available to the runtime. The progress metric is used by the runtime scheduler to (i) assess the data dependence between the applications, and (ii) manage data reuse by appropriate scheduling decisions. Our implementations of this technique on the Open Community Runtime (OCR) lead to performance improvement on an x86 platform and demonstrate energy savings of more than 40% on an experimental Exascale architecture.

1. Introduction

Next generation exascale machines are engendering new usage models in which multiple co-running and symbiotic applications share the resources assigned to a single job. An example is a long-running simulation co-running with data analytics continually processing simulation output to assess validity, ensure progress, extract valuable application insights [14], or even enable application steering [6, 7].

For peta- and exa-scale machines, extensive prior work has developed support infrastructures for online data analytics applied to I/O pipelines, to jointly scale the I/O and the analytics and visualization codes operating on such data. This includes running analytics concurrently with simulations – "in-situ" [16] – and in I/O staging areas – "in-transit" [1, 5] – on the high end machine itself and/or extending to auxiliary analytics clusters [4]. More generally, this implies that at any one time, there will be multiple, cooperating applications (e.g., visualization or analytics vs. those running a simulation) co-running on the same high end machines. For exascale machines like Traleika Glacier (TG) [3], supporting cooperative, loosely coupled applications entails support for resource partitioning by the runtime, given that such architectures will host a light-weight no-OS solution [9], where the runtime would handle system services in addition to handling application resource requirements based on goals or optimization opportunities.

For cooperating applications running on the same high end machines, a basic problem to be solved is for them to continually make appropriate levels of 'progress' in order to jointly obtain high performance. A specific case in point is a simulation producing output data consumed by co-running data analytics, both desiring high joint performance with limited resources available for temporary storage of output data. Previous work on ensuring progress for coupled applications has considered controlled production vs. consumption rates for multi-media applications coupled via shared buffers [10, 13], used overprovisioning of staging areas, and attempted to leverage idle cycles on the nodes where a simulation is run to run analytics codes [16]. Such prior work has identified two key issues that arise and must be addressed for ensuring mutual progress. First, there must be cooperation between the applications, via runtime assistance or underlying operating system, so that both can be aligned in terms of the scheduling decisions being made. Second, there are additional effects of co-location on the same node that must be taken into account, such as interference between simulation and analytics due to their use of shared node resources like its cache and memory [11, 16].

In exascale systems, with extreme NUMA characteristics like TG [3], the problem of limited temporary storage worsens considering that data movement costs are expected to be the primary drivers of energy consumption, hence data locality and reuse become key factors in ensuring application performance and efficient energy consumption [2].

This paper considers multiple applications in a single, task-based runtime – for the simulation and for analytics – co-running on the same machine, both demanding appropriate levels of continuous progress. The runtime has resource partitioning techniques using which it allocates appropriate resources to each application's tasks. In our scenario, we allocate resources statically for simulation and analytics to make progress and allow for dynamic re-allocation based on observed variations in progress rate for both applications. In contrast to prior work, we explore next generation highly asynchronous, task-based runtimes like Charm++ [8] and the Open Community Runtime (OCR) [12] as the vehicles for running simulation and analytics codes. Such runtime systems provide increased scope for solutions for progress detection and then, mitigation, that can go beyond the black-box methods used elsewhere, which for detection, rely on inspection of low-level system metrics, like IPC and LLC misses, and for prevention or mitigation, throttle analytics via explicit scheduling. Specifically, in our case, it becomes possible to link the application and runtime-level information to jointly in-

form runtime scheduling to ensure adequate progress for coupled applications.

The rest of the paper is organized as follows. Section 2 motivates the need for progress-rate driven scheduling. Section 3 introduces the runtime and describes the implementation of resource scheduling based on progress-based hints. Section 4 shows early results on cholesky and $O(n^2)$ analytics. Section 5 explores related work. We conclude the paper with Section 6.

2. Motivation

There are several motivating factors for progress-rate driven scheduling, especially for coupled co-located cooperative applications like simulation and analytics.

The primary reason is achieving increased true "in-situ" processing of the simulation output, where data is processed in place – memory modules or even caches – where it is originally placed by the simulation. The rationale is that by preventing analytics from lagging further behind the simulations they are coupled with, the likelihood of data getting moved down the memory hierarchy is reduced – e.g., from on-chip, near memory like stacked DRAM to off-chip DRAM or to high-capacity but slower non-volatile memories, like PCM, envisioned to be part of exascale system designs. Reducing the need for data movement leads to performance improvements – due to access to data in faster/nearer memory, and to energy improvements – by both avoiding the interconnect as well as actual read/write device operations. A related opportunity is that, by being progress-aware, the scheduler actions can more easily take data-locality metrics into consideration, and schedule analytics where the data was actually produced, on CPUs 'closest' to the memory component where data resides.

For these reasons, in this paper we are particularly concerned with the impact of progress-aware scheduling on 'data reuse' as a key factor contributing to performance and energy efficiency.

Additional goals of the scheduling policy include improving overall time-to-answer, including the total execution time for the simulation and analytics code, as well the iteration latency producing intermediate output produced by individual time step minimizing application wait time, minimizing resource contention for platform-wide shared resources like interconnect and memory controllers, maximizing the use of machine resources, and in general, the energy efficiency of the platform.

In addition to the need for progress-rate driven scheduling, another aspect that makes such scheduling appealing is the ability to track progress-rate in a task-based runtime system, which provides more access and visibility into application characteristics and runtime behavior like the type of tasks, the number of tasks, current resource usage, queued tasks waiting to be run etc. Runtime APIs available to applications can be used to explicitly inform the runtime of application phases, like algorithmic timesteps, that can be used by the runtime to derive progress-rate.

3. Design and Implementation

The general workflow of simulation and analytics co-located on the same node is shown in Figure 1. In our experiments, the primary representative application for simulation that we consider is Cholesky and we use an $O(n^2)$ as a representative analytics operation on the data that is output by the simulation - which is common for many analytical codes.

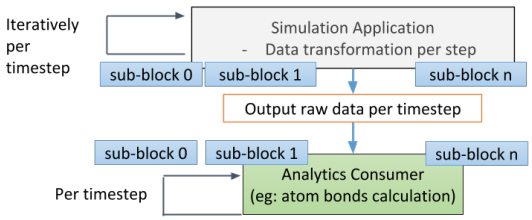

Figure 1: Flowchart illustrating the control flow and data dependences of simulation & analytics

Both applications are modeled to be co-run using Open Community Runtime (OCR), an asynchronous event-driven task-based programming model which has been co-designed to run on Exascale systems like TG. The programming model allows for applications' codes to be modeled as tasks and their data-block dependences, providing the runtime with explicit control over the placement of these entities to improve performance and energy usage. However, the data dependence between the two individual co-located applications is not explicitly modeled as data-blocks.

In order to support our work on progress-based scheduling, we implement two primary components, described briefly below.

Introspection hooks. These APIs allow application code to inform the runtime of its algorithmic *phases*, like timesteps, so the runtime can derive the current progress of the application from this context. The introspection API used in our prototype implementation is shown below:

```
u8 reportStartOfTimestep(u64 app_tag,
                         u64 timestep_index);
u8 reportEndOfTimestep(u64 app_tag,
                       u64 timestep_index);
```

Scheduler support.

OCR on x86: For supporting progress-rate based scheduling in OCR on x86, we augment the work-stealing scheduler with a resource partitioning scheme. This scheme partitions worker resources to selectively schedule tasks, either from simulation or from analytics to allow for sustained progress of both. For this purpose, we implement workers to have two double-ended queues (deques), one each for tasks from simulation and analytics, to enable their selection and scheduling as they become runnable.

OCR on TG: A similar scheme is implemented for OCR on TG. The TG architecture consists of several low-power single-issue specialized cores to execute application code – termed execution engines (XEs), controlled by fewer x86-based control engines (CEs) to orchestrate task scheduling, data mapping, and monitoring. Instead of two local deques per worker as in the x86 implementation, on TG we have two global deques per CE, to store runnable tasks, that can be executed by XE workers. Paritioning is similar to OCR on x86, i.e, we partition workers into sets, each working on a different (simulation or analytics) application's EDTs. On TG, it is relatively harder to extract time per timestep, since its functional simulator has a logical timing model. Hence we allocate resources based on prior knowledge to demonstrate progress-based scheduling.

Scheduling Algorithm: The scheduler determines the tasks to be executed based on per-application progress notion, expressed as time taken by the recent timesteps to finish tasks. The runtime maintains information on time taken per timestep by measuring and storing time on invocation of introspection APIs reporting start and end of each timestep. A one-to-one correspondence in timesteps across applications is assumed and relative progress is assessed by comparing time taken per timestep. In the current implementation we use a threshold-based policy on progress to trigger resource re-allocation to increase resources for analytics application, whenever its progress relative to the simulation falls below a certain threshold.

4. Experimental Evaluation

We demonstrate benefits from progress-driven scheduling by evaluating the mechanism on OCR on x86 and OCR on TG.

Testbed. OCR on x86 is tested on dual-socket, hex-core Intel Westmere platform with a total of 48 GB RAM, half on each socket. OCR on TG is run on FSim, a functional simulator for TG.

The observations are listed below:

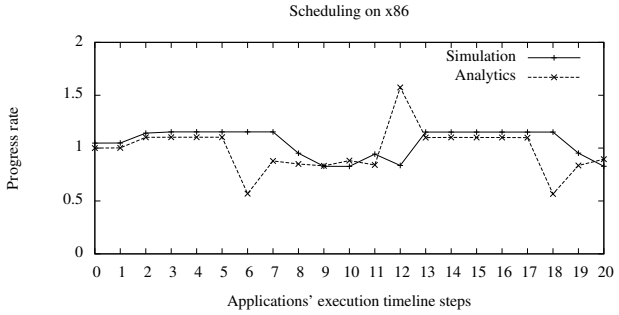

Figure 2: Adaptation Accuracy on x86

1. Adaptive Scheduling - In Figure 2, we show the ability of our scheduler to adapt to time step variations in progress rate. The x-axis is the timeline of execution of both applications. The y-axis is the progress rate per timestep. If the application, in this case analytics, shows variability, then by increasing or decreasing workers assigned, the progress rates are matched to ensure that simulation and analytics make instep progress.

2. Experimental results on x86 - In Figure 3, we show improvements from progress based scheduling for CoMD - O(n^2) analytics on x86. The benefits are derived from better utilization of idle resources and reduced cache thrashing due to timestep alignment. The reason for the relatively modest gains from progress-based scheduling is that, in x86, it is harder to see benefits of data reuse from LLC due to hardware prefectching. Furthermore, in this experimental platform, memory capacity does not present a bottleneck for these applications. In exascale systems, we foresee caches to be replaced by scratchpad without hardware prefecthing capabilities. In addition, the challenges posed by the "memory wall" in exascale systems are well-recognized in the community. Therefore, to emulate a scenario with no-prefectching and higher compute-to-memory ratio, we used a memory hogging utility and show effects of bounded buffer problem in DRAM in Figure 4, which results in paging, if timesteps are unaligned on x86 preventing in-memory data reuse. Multiple runs with median values are show in the figure, confirming the benefits of progress based scheduling. We also show MB of sampled page swapping performed for unaligned case, in Figure 4.

3. Experimental results on TG - For experimentation on TG, we measure benefits of time-step aligned scheduling by means of tracking data accesses using FSim, instead of tracking exact timing related measurements. The rationale behind the measurements is that we seek to demonstrate improvements in terms of increased near memory (BSM) accesses and reduced far memory (DRAM) accesses. We observe a reduction of over 8-15X in DRAM accesses for realistc and synthetic workloads, in Figure 6 and Figure 5. We observe about 40% energy usage reduction in energy requirement for realistic workloads as can be seen from Figure 7. The primary intent is to demonstrate reduction in DRAM energy usage, which tends to be the dominant contributor. The graph is normalized against energy used by progress-based scheduling.

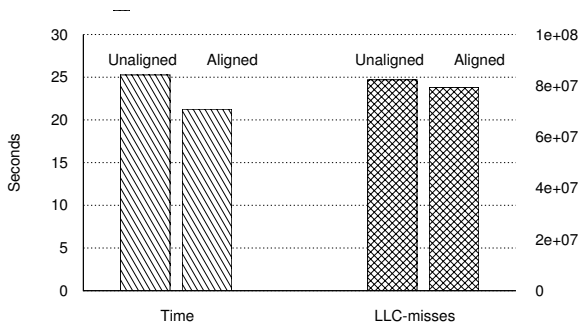

Figure 3: Simulation - Analytics (CoMD-O(n^2)) on x86

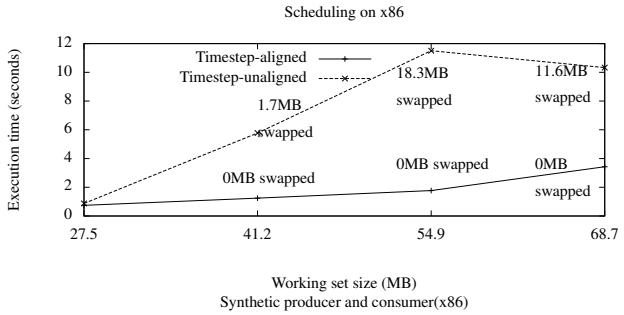

Figure 4: Simulation - Analytics on x86 (Bounded buffer in memory).

5. Related work

There is much previous work on feedback-based scheduling, one of the earliest being that in [10], where the progress of communicating processes is estimated every 'X' periods dynamically, and CPU proportions are increased or decreased accordingly. Progress estimates are based on the state of the shared communication resource. Our primary method of estimation is time-based. The progress metric used in Zaharia et. al [15] is task execution time, the purpose being to estimate delays and determine straggler tasks in order to quickly re-execute stragglers. Its simple methods of linearly extrapolating execution time and assigning scores to nodes to detect slow nodes is suitable for its more homogenous execution model consisting of only two types of tasks, namely, map and reduce tasks, which have more deterministic phase characteristics than the tasks in highly asynchronous runtimes like OCR.

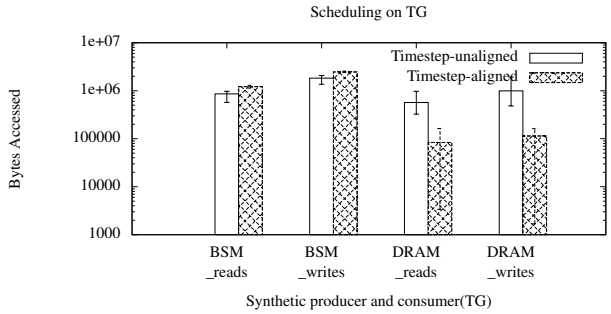

Figure 5: Synthetic producer and consumer. The baseline is when the runtime does not adapt to lack of progress from consumer analytics.

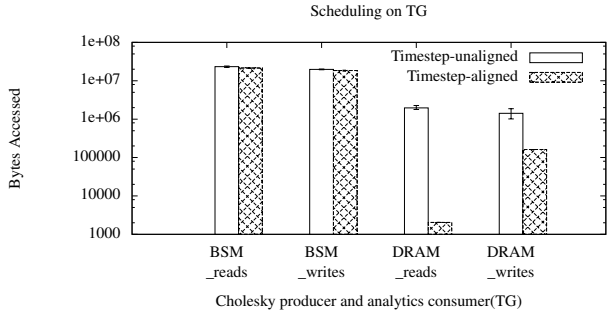

Figure 6: Cholesky producer and O(n^2) analytics consumer.

6. Conclusion

We show that data reuse can yield high benefits for future Exascale systems and data locality aware scheduling is key in energy consumption on such extreme scale systems. We demonstrate application progress is an important metric that can lead to understanding data access patterns for co-located coupled applications and incorporating such hints can lead to significant benefits for HPC application on future exascale machines. These insights apply for both single runtimes on Exascale architecture like TG and other OS-based systems like Kitten, which are concerned with co-locating applications in separate enclaves to provide a more isolated execution environment.

7. Acknowledgment

This work is supported through research grants provided under the Office of Science ExaOS/R and XStack programs.

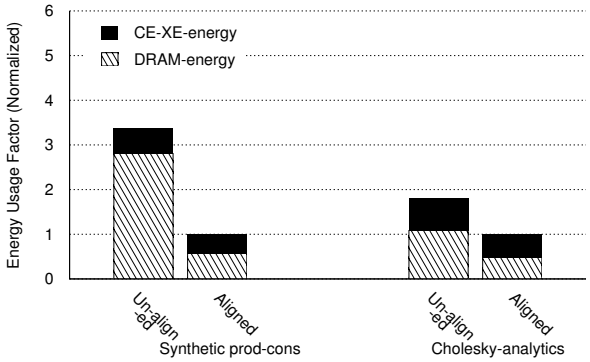

Figure 7: Energy savings & breakdown

References

[1] H. Abbasi, M. Wolf, G. Eisenhauer, S. Klasky, K. Schwan, and F. Zheng. Datastager: scalable data staging services for petascale applications. *Cluster Computing*, 13(3), 2010.

[2] S. Borkar. How to stop interconnects from hindering the future of computing. In *Optical Interconnects Conference, 2013 IEEE*, pages 96–97, May 2013.

[3] N. P. Carter, A. Agrawal, S. Borkar, R. Cledat, H. David, D. Dunning, J. Fryman, I. Ganev, R. A. Golliver, R. Knauerhase, R. Lethin, B. Meister, A. K. Mishra, W. R. Pinfold, J. Teller, J. Torrellas, N. Vasilache, G. Venkatesh, and J. Xu. Runnemede: An architecture for ubiquitous high-performance computing. In *Proceedings of the 2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*, HPCA '13, Washington, DC, USA, 2013. IEEE Computer Society.

[4] J. Dayal, D. Bratcher, G. Eisenhauer, K. Schwan, M. Wolf, X. Zhang, H. Abbasi, S. Klasky, and N. Podhorszki. Flexpath: Type-based publish/subscribe system for large-scale science analytics. In *14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGrid 2014, Chicago, IL, USA, May 26-29, 2014*, pages 246–255, 2014.

[5] C. Docan, M. Parashar, and S. Klasky. Dataspaces: An interaction and coordination framework for coupled simulation workflows. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, HPDC '10, pages 25–36, New York, NY, USA, 2010. ACM.

[6] W. Gu, G. Eisenhauer, E. Kraemer, K. Schwan, J. Stasko, J. Vetter, and N. Mallavarupu. Falcon: on-line monitoring and steering of large-scale parallel programs. In *Frontiers of Massively Parallel Computation, 1995. Proceedings. Frontiers '95., Fifth Symposium on the*, pages 422–429, Feb 1995.

[7] W. Gu, J. Vetter, and K. Schwan. An annotated bibliography of interactive program steering. *SIGPLAN Not.*, 29(9), Sept. 1994.

[8] L. V. Kale and S. Krishnan. Charm++: Parallel programming with message-driven objects. *Parallel Programming using C+*, pages 175–213, 1996.

[9] R. Knauerhase, R. Cledat, and J. Teller. For extreme parallelism, your os is sooooo last-millennium. In *Proceedings of the 4th USENIX Conference on Hot Topics in Parallelism*, HotPar'12, pages 3–3, Berkeley, CA, USA, 2012. USENIX Association.

[10] D. C. Steere, A. Goel, J. Gruenberg, D. McNamee, C. Pu, and J. Walpole. A feedback-driven proportion allocator for real-rate scheduling. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, OSDI '99, pages 145–158, Berkeley, CA, USA, 1999. USENIX Association.

[11] P. Tembey, A. Gavrilovska, and K. Schwan. Merlin: Application- and platform-aware resource allocation in consolidated server systems. In *Proceedings of the ACM Symposium on Cloud Computing*, SOCC '14, pages 14:1–14:14, New York, NY, USA, 2014. ACM.

[12] Open Community Runtime. https://01.org/open-community-runtime.

[13] R. West, I. Ganev, and K. Schwan. Window-constrained process scheduling for linux systems. In *In Proceedings of the 3rd Real-Time Linux Workshop*, 2001.

[14] M. Wolf, Z. Cai, W. Huang, and K. Schwan. Smartpointers: Personalized scientific data portals in your hand. In *In Proc. of SuperComputing 2002*, pages 1–16. Press, 2002.

[15] M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, and I. Stoica. Improving mapreduce performance in heterogeneous environments. In *Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation*, OSDI'08, pages 29–42, Berkeley, CA, USA, 2008. USENIX Association.

[16] F. Zheng, H. Yu, C. Hantas, M. Wolf, G. Eisenhauer, K. Schwan, H. Abbasi, and S. Klasky. Goldrush: Resource efficient in situ scientific data analytics using fine-grained interference aware execution. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '13, pages 78:1–78:12, New York, NY, USA, 2013. ACM.