Asynchronous In Situ Processing with Gromacs: Taking Advantage of GPUs

Monica Liliana Hernandez Ariza, Matthieu Dreher, Carlos Jaime Barrios-Hernandez, Bruno Raffin

Asynchronous In Situ Processing with Gromacs: Taking Advantage of GPUs

Monica L. Hernandez[1,2], Matthieu Dreher[2,3], Carlos J. Barrios[1], and Bruno Raffin[2]

[1] Universidad Industrial de Santander, Colombia
[2] INRIA, University Grenoble Alpes, France
[3] Argonne National Laboratory, USA

Abstract. Numerical simulations using supercomputers are producing an ever growing amount of data. Efficient production and analysis of these data are the key to future discoveries. The in situ paradigm is emerging as a promising solution to avoid the I/O bottleneck encountered in the file system for both the simulation and the analytics by treating the data as soon as they are produced in memory. Various strategies and implementations have been proposed in the last years to support in situ treatments with a low impact on the simulation performance. Yet, little efforts have been made when it comes to perform in situ analytics with hybrid simulations supporting accelerators like GPUs. In this article, we propose a study of the in situ strategies with Gromacs, a molecular dynamic simulation code supporting multi-GPUs, as our application target. We specifically focus on the computational resources usage of the machine by the simulation and the in situ analytics. We finally extend the usual in situ placement strategies to the case of in situ analytics running on a GPU and study their impact on both Gromacs performance and the resource usage of the machine. We show in particular that running in situ analytics on the GPU can be a more efficient solution than on the CPU especially when the CPU is the bottleneck of the simulation.

Keywords: In situ Analysis; FlowVR; Graphics Processing Units; Gromacs.

1 Introduction

Large scale simulations are an important tool for scientists in various domains such as biology, fluid dynamic, material science or astrophysics. Yet, it is becoming more and more challenging to analyze the ever growing amount of data produced by these simulations. In 2010 already, a turbulence simulation (GTC [13]) was producing 260GB of data every 2 minutes using only 16384 cores [33]. More recently, in biology, the complete atomistic model of the HIV capsid has been determined [32]. Several simulations, each producing about 50TB of data for a total of 1PB, were required to build this model.

In the Exascale era, it is estimated that less than 1% of the data produced by simulation will be saved because of bandwidth constraints [19]. Writing raw data to disks will no longer be viable because of the resulting loss of information.

The in situ paradigm is emerging as one promising solution to this problem. The principle is to process data as close as possible to their source while data still reside in memory [31]. Both the simulation and analytics benefit from this

approach since they do not have to write/read to/from the file system. Although this approach was initially designed for I/O, numerous applications are possible: live visualization, statistics generation, feature tracking, simulation monitoring, etc. However, setting up such analysis can be challenging. As both simulation and analysis run concurrently, contention for the computational and network resources can lead to significant degradations of the simulation performance. Several strategies and systems have been proposed to mitigate this penalty in the last few years.

Another benefit from in situ processing is to improve the global resource usage of the machine. Usually, simulation codes cannot fully use and scale with all the computational resources available on large parallel machines [35]. For instance, running the GTS code on 512 cores using only 3 out of 4 cores per socket reduces the simulation performances by only 2.7% compare to using all the available cores [36]. For these cases, it can be more efficient to use the fourth core to run in situ analytics to accelerate the analysis phase and therefore shorten the time to discovery.

Efficiently using hybrid computational resources such as CPUs and accelerators is even more challenging for simulation codes. In the last years, various leadership parallel machines such as Tianhe-2 or BlueWaters have integrated accelerators (GPUs, Xeon PHI). The future 150+ petaflop machine Summit at Oak Ridge National Laboratory will also integrate GPUs in its architecture. These accelerators offer a high Flops/Watt ratio that is required to reach Exascale. Several codes such as NAMD [20] or Gromacs [11] have been adapted to benefit from these accelerators and lead to significant speedups.Yet, in most cases, not all the computations are performed on the accelerator. Consequently, there are some periods during the simulation execution where the accelerator is idle leading to underused resources.

Significant efforts have been made by the community to propose in situ systems with a low impact on the simulation performance. Yet, most of them focused on simulations and analytics running only on CPUs [5, 7, 29, 35]. In this article, we study current in situ strategies applied to hybrid simulations. Gromacs, a well established molecular dynamics simulation package supporting multi-GPUs, is our application target. We first study the usage of the computational resources by Gromacs native during classical runs. Then we study the resource usage when in situ analytics are running on the CPU using asynchronous time-partitioning and helper core strategies. We rely on FlowVR [3, 7], a middleware for asynchronous in situ/in transit applications, to implement these strategies. Finally, we adapt these two strategies for in situ analytics running on GPUs and analyze the resource usage. We show in particular that running in situ analytics on the GPU can be a more efficient solution than on the CPU especially when the CPU is the bottleneck of the simulation.

The rest of the article is organized as follow: we first discuss the related work (Sec. 2); then, we present Gromacs and the framework used to perform our experiments (Sec. 3). We present our experimental results (Sec. 4) and summarize our findings (Sec. 5).

2 Related Work

In this section, we first present the systems and strategies to perform in situ treatments. Then we present some use cases of treatments with GPUs in the context of in situ or off-line processing.

2.1 In situ Systems

One key design decision for in situ systems is the placement of analytics processing.

The most direct approach is to host the simulation and the analytics on the same computational resources. This strategy, called time-partitioning, is usually the easiest to implement. It can also enable the simulation and the analytics to share data structures leading to a reduced memory footprint. Ma et al. [31] integrated a volume rendering engine directly into the code of a turbulence simulation. About 10% of the total execution time is spent in the rendering step. Common visualization tools like Visit [30] or Paraview [9] have lightweight libraries to instrument the simulation. Their main purpose is to convert the simulation data format to the VTK format before executing an analysis pipeline. Tu et al. [28] propose a fully integrated solution with an earthquake simulation and the Hercules framework to perform in situ visualization. The only output is a set of JPEG images. For these systems, the time spent running the analytics is directly added to the simulation time. This approach can be very costly in both time and memory. A study has been proposed with Catalyst [18] on industrial simulations. With commonly used analysis scenarios, they observed up to 30% of increased execution time and up to 300% increased memory consumption because of data conversion requirements. Goldrush [35] tackles the problem of the global execution time by making the treatments asynchronous. To limit the impact of the asynchronous treatments on the simulation run time, the treatments are scheduled when the simulation is not using all the available cores (outside of an OpenMP section).The goal is to improve the global resource usage of the machine by scheduling the analysis when the resources are underused.

Other works propose dedicated resources to perform in situ analytics. This approach, called space-partitioning, allows asynchronous in situ analytics execution, avoids some contention on the computational resource but requires at least one data copy. Some systems, like Damaris [5], use dedicated cores (called *helper cores*) on each simulation node to execute asynchronously the in situ analytics. Data are copied from the simulation into a shared-memory space. Analytics can then read and process data from this space asynchronously. Applications such as I/O or scientific visualization with Visit [6] are then possible. The *helper core* strategy has also been used by Functional Partitioning [16] and GePSeA [22] mainly to focus on I/O operations.

Other systems propose to use a separate set of nodes (called *staging nodes*) to execute analytics (called in transit analytics). PreData [33] is built within the ADIOS framework [17] and allows to execute lightweight in situ operations before moving the data to *staging nodes*. Data are then processed in transit using a Map/Reduce like model. DataTap is used to schedule the data transfer when the simulation is in a computation phase and is not using the network card extensively. HDF5/DMS [23] uses the HDF5 interface to capture the data and store them in a distributed shared memory space. Other applications can then read the data with the read HDF5 API usually on a different set of nodes. DataSpaces [4] implements a distributed publish/subscribe system. The simulation pushes data in a indexed distributed space and other treatments retrieve the necessary data.

More recently, hybrid systems combining both in situ (synchronous or not) and in transit treatments have emerged. Fheng et al. [34] highlight the necessity of placement flexibility of analytics and propose an analytical model to evaluate the cost of the placement strategies. Glean [29] allows synchronous in situ

treatments and asynchronous in transit treatments. FlexIO [36] is built on top of ADIOS and allows asynchronous in situ treatments on dedicated cores and asynchronous in transit treatments.The system monitors the performance of the simulation and can migrate the analytics at runtime or slow them down if they are impacting too much the simulation performance. FlowVR [3, 7] allows describing a data-flow between components (simulation, analysis modules). Treatments are performed asynchronously from the simulation. The user can specify the location of the treatments: on a set of helper cores, staging nodes, or on the same resources as the simulation.

Our work in this paper follows the work done on helper core approaches and asynchronous time-partitioning approaches. We extend these works to the domain of multi-GPU simulations and in situ analytics using GPUs. To implement our approach, we rely on the FlowVR middleware to host and coordinate the in situ analytics execution.

2.2 Treatments with GPU

An implementation of in situ systems using GPUs is the work presented by R. Hagan et al. [10] who propose a load balancing method for in situ visualization in a multiGPU system. This method is based on an asynchronous space sharing strategy where N/2 GPUs are used as dedicated GPUs for visualization, N being the number of GPUs in the system. The other N/2 GPUs perform the N-body simulation and transfer the data processed to RAM. Once in the memory, the data are transferred to the dedicated GPU to perform rendering task through a ray tracing visualization algorithm. Each GPU is managed with separate buffers on the CPU side in order to write/read the data to/from memory asynchronously.

Performing off-line processing on GPUs is a growing field of interest. VMD [12] is a widely used tool for visualization and analysis of biological systems such as proteins and nucleic acids. Over the last years, many visualizations and analytics have been adapted to support GPUs using CUDA. The Quicksurf algorithm [14], for instance, has been proposed to visualize molecular surfaces of large ensembles of atoms. It has been recently used to visualize the full model of the HIV capsid on BlueWaters [27]. Other analysis such as radial distribution functions [15], fitting [26] and many others [24] are accelerated with GPUs. Although VMD does not have a full support for in situ analysis, some interactive applications combining simulation and live visualization are possible such as Interactive Molecular Dynamic simulations [25].

3 Framework Description

3.1 Gromacs

We describe here the features of Gromacs that are needed to understand its behavior and performance. The reader can refer to [11, 21] for complementary details.

Gromacs is a commonly used parallel molecular dynamics simulation package. It is able to scale to several millions of atoms on several thousands of cores by using MPI, OpenMP, and CUDA. The internal organization of Gromacs is a master/slave approach. The master process is responsible for maintaining a global state when necessary and performing the I/O operations.

Atoms are distributed in a irregular grid where each cell of the grid is managed by one MPI process. We call *home atoms* of an MPI process the atoms belonging to its cell. The cell sizes of the grid are adjusted at runtime by a dynamic load-balancing system. The main computation part is that of the forces: bonded interactions between atoms sharing a link and non-bonded interactions for the distant atoms. Non-bonded interactions are the most computationally expensive operations because they require N-to-N communications. Performance timings are monitored during this phase to load-balance the simulation.

Since version 4.6, Gromacs supports GPUs with CUDA, where no bond interactions are transferred to the GPUs while the bonded-interactions are executed in parallel on the CPU. Gromacs also supports multi-GPUs: each GPU is assigned to an MPI process; OpenMP is used to fill the rest of the cores when more cores than GPUs are available on a node. Since the bonded and non-bonded computations are performed concurrently, a balance must be found between CPU and GPU computations. The dynamic load-balancing system monitors the difference of computation time between the CPU and the GPU and adjusts the grid dimensions accordingly.

3.2 FlowVR

FlowVR [7] is our middleware to create asynchronous in situ applications. It allows describing an application as a graph, where nodes are data operations and edges are communication channels. A node is called a module and is equipped with input and output ports. A module runs an infinite loop. At each iteration, a module can receive data, process them, and send computed data to the rest of the application. The loop is implemented with three main functions: *wait, get*, and *put*. *Wait* blocks the module until there is at least one message in all input ports. *Get* returns the oldest message from an input port's queue. *Put* sends a message to an output port. Both *Get* and *Put* functions are nonblocking.

A module has no knowledge of the data source and destination. The data channels are described in a Python script that declares the global application and creates the links between the modules. Each module is assigned to a host and possibly to a set of cores on the host. A daemon is hosted on each node in the application. It hosts a shared memory segment in which the modules are reading and writing their data. If two modules are on the same host, the daemon does a simple exchange of pointers in the shared memory space. Otherwise, the local daemon sends the message to the daemon hosting the remote module, which will write the data in its shared memory space and pass the pointer to its module.

FlowVR does not impose any restrictions on the resources used by a module. Any module is free to use OpenMP or GPUs to accelerate its treatment. However, FlowVR does not provide any protections in case several modules are using intensively the same resources.

For more details, the reader can refer to [7].

3.3 Gromacs-FlowVR Interaction

We have instrumented the simulation code Gromacs with a similar method than our previous works [7, 8]. For each MPI process, we declare one module with one output port to extract the atom positions. This approach allows us to preserve the same level of parallelism of the data, which can be used later by in situ analytics.

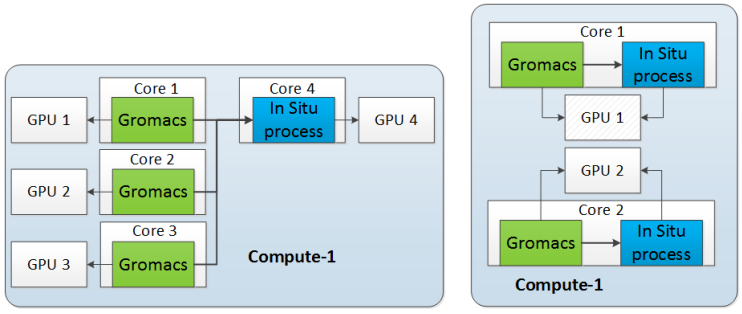
We modified the main loop of Gromacs to periodically extract the atom positions. Every x iterations, with x a user-defined parameter, each module performs a FlowVR *wait()*, copies the positions of the *home atoms* inside a FlowVR message, and puts the message. The atom positions are then sent to the rest of the application, if the output ports of the modules are connected to other modules such as in situ analytics. Otherwise, the data are erased because they are not used anymore by any module. Note that because the Gromacs modules do not have any input ports, the *wait()* will return immediately and not block the simulation. In order to minimize any noise in the simulation performance, we have also disabled the native writing system of Gromacs.

3.4 Benchmark Framework Description

We implemented a framework to perform in situ analytics based on two different placement strategies: *helper core* and *overlapping*. For all strategies, the simulation and the in situ analytics have the possibility to use GPUs.

The *helper core* strategy reserves one core per node to perform in situ analytics (see Figure 1(a)). Data are gathered on each node asynchronously and sent to one or several in situ tasks hosted on the helper core. We assigned one GPU per simulation process and one GPU for the analytics.

The *overlapping* strategy runs on the same resources as the simulation (see Figure 1(b)). In our case, we instantiated as many in situ tasks as there are MPI processes per node. Therefore, each MPI process of the simulation sends the data to the in situ task located on the same core as the MPI process. Note that the in situ tasks are running asynchronously with the simulation. Each GPU is shared between one simulation process and one in situ task. At runtime, both simulation and analytics kernels run concurrently on each GPU.



(a) (b)

Fig. 1. (a) Helper core strategy. A dedicated core is allocated for in situ analytics. Data are gathered by the node and sent asynchronously to the in situ tasks (b) overlapping strategy. One in situ task is instantiated for each MPI process of the simulation.

The in situ analytics are triggered each time the simulation outputs data. The different analytics used are described in the next section. In these two particular setups, the communication channels are simple FIFO channels between the simulation and analytics modules. Overflows can occur if the analytics do not follow the pace of the simulation. For this framework, this is an acceptable situation since

the data produced are relatively small and just a few output steps are performed. For real application scenarios, special components can be added to sample the output data from the simulation. It is also possible to block the simulation at the next output step if the previous output step has not been analyzed yet.

3.5 Benchmarks

We designed this framework to evaluate the impact of in situ CPU/GPU tasks on Gromacs performance and the resource usage of the machine. We adopted the same approach as in [35]. We implemented several micro benchmarks, each designed to stress specific parts of a multi-GPU parallel machine. Each of these benchmarks is available for *overlapping* and *helper core* strategies.

PI (CPU) The PI benchmark, used by Zheng et al. in [35], stresses the floating point units of the processors. When PI is triggered, x iterations, with x an user-defined parameter, are performed to estimate the value of π. For both strategies, *overlapping* and *helper core*, we execute the same total number of PI iterations. With the *helper core* strategy, only one in situ process computes all the iterations (x). In the case of the *overlapping* strategy, N in situ processes are used. The x iterations are then distributed evenly among all the in situ tasks (x/N).

This benchmark perturbs the CPU while both the CPU and GPU are intensively used by the simulation. The simulation load-balances both the CPU and GPU computations. Therefore, perturbing the CPU should impact both the CPU and GPU computations from the simulation.

Bandwidth (GPU) The bandwidth Nvidia CUDA kernel stresses the communications between the CPU and the GPU by sending and receiving data packages several times. The message sizes s are user-defined. For the *helper core* strategy, one GPU receives messages of size s. In the case of the *overlapping* strategy, each of the N GPUs receives messages of size s/N.

Data exchanges are frequently performed between the CPU and the GPU during the simulation. Perturbing the GPU data transmission can delay the GPU computations of the simulation waiting for their data transfers.

Matrix Multiplication (GPU) The multMatrix Nvidia CUDA kernel is a compute-intensive kernel multiplying 2 matrices several times. The sizes of the two matrices are 640x320 and 320x320, respectively. The number of multiplications y performed during one iteration of the benchmark is user defined. With the *helper core* strategy, one in situ process does all the matrix multiplications (y). In the case of the *overlapping* strategy, N in situ processes performed y/N.

This benchmark occupies the processing units of the GPU. The kernels of the simulation will have less multiprocessors available to be scheduled leading to delays of the simulation computations.

The impact of the benchmarks on the simulation performance depends not only on the benchmark but also on the balance of GPU/CPU computations adopted by the simulation. If the CPU is the limiting factor of the simulation, the GPU benchmark should be less damaging and vice versa if the GPU is the limiting factor.

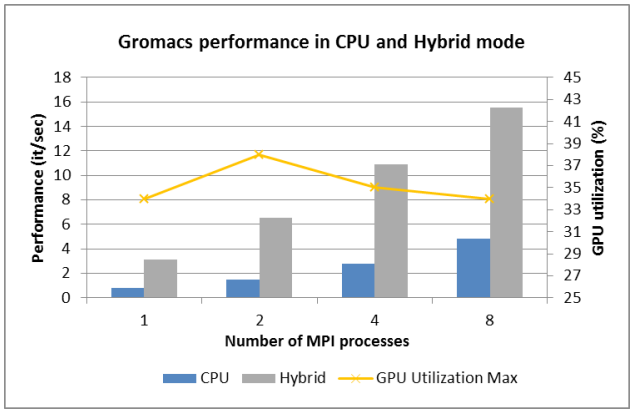


Fig. 2. Native Gromacs performance and GPU utilization when increasing the number of MPI process

4 Experiments

4.1 Experimental context

We ran our experiments on the cluster *GUANE-1 (GpUs Advanced eNviromEnt)* at *Universidad Industrial de Santander*. Each node is a 8-core Intel® Xeon® CPU E5640 @ 2.67 GHz (two sockets with 4 cores each one) with hyper-threading activated (16 logical cores), 103 GB of RAM and 8 Nvidia Tesla M2050 GPUs (448 cores each). Interprocess communication is done through a 1 GB Ethernet network. For all experiments, Gromacs runs a Martini simulation (simulation of atom aggregates) with a patch of 54000 lipids representing about 2.1 million particles in coarse grain [1]. Gromacs is very sensitive to the quality of the network due to its high frequency [2]. Therefore, we preferred to avoid intranode communications and used only one node for our experiments. For all experiments, the native writing method of Gromacs is disabled.

We measured both the simulation performance and the GPU utilization for each experiment. The performance metric is iterations per second (higher is better). Each simulation lasted at least 1000 iteration steps to avoid performance jittering due to the dynamic load-balancing at the initialization phase. The GPU utilization is measured with the tool nvidia-smi[4] from Nvidia. The GPU utilization indicates the percent of time over the past second during which one or more kernel were executed on each GPU. We took the highest GPU utilization that we found from every experiment (five measures with nvidia-smi per experiment).

4.2 Gromacs native

We first benchmarked Gromacs stand alone (no code modification) to determine which configuration (number of threads, MPI processes, GPUs) provides the best performance on our node.

Figure 2 presents the results for Gromacs running on the CPU only and in hybrid mode using GPUs. As a reminder we use 1 GPU for each MPI process. For

[4] https://developer.nvidia.com/nvidia-system-management-interface

both cases, we used 2 OpenMP threads per MPI process with the 2 threads mapped to the same physical core (hyperthreading). The hybrid version outperforms the CPU version in all cases by a factor from 3.2 for 8 MPI processes to 4.38 for 2 MPI processes.

During the simulation, CPU and GPU computations are performed concurrently. The computations are balanced at runtime by Gromacs. According to Gromacs internal logs, for each hybrid configuration, the CPU is not waiting for the GPU. This can indicate that the GPU is idle while the CPU completes its computation.

The GPU utilization of Figure 2 indicates the maximum percentage of time where at least one CUDA kernel is active. In the best case, for 2 MPI processes, the GPU is used at most only 38% of the time. Moreover, when the simulation kernels are running, the GPU occupancy is only of 60%[5].

Although Gromacs greatly benefits from using multiple GPUs, these resources are underused by the simulation. These results indicate that, in the case of Gromacs, the CPU is the limiting factor in the computation.

4.3 Gromacs Instrumented with FlowVR

We measured the performances of both the native Gromacs and our FlowVR-instrumented version. For each MPI process, we allocate 2 OpenMP threads and 1 GPU as previously. For the FlowVR-instrumented version, we extracted the data every 100 simulation iterations.

At most, our instrumentation cost increases the simulation time by 0.5% in the case of 2 MPI processes. The impact on the GPU utilization is also negligible. These results demonstrate that our instrumentation method does not impact the simulation behavior. This cost is significantly lower than our previous report [7]. This is explained by a much lower output frequency. Previously, we extracted data every 10 ms. For this study, we only extract the data every 6 seconds. As the instrumentation blocks the simulation for about 0.2 ms at each output step (*Wait()* and copy of the atom positions), the cost of instrumentation is negligible

4.4 Gromacs with CPU analytics

We measured the performances of Gromacs while running asynchronous in situ analytics on the CPU. We used the PI benchmark described in Section 3.5 in both *helper core* and *overlapping* strategies. Gromacs outputs data every 100 simulation iterations. For each Gromacs output, we triggered y iterations of PI in total. Figure 3 shows the simulation performance and GPU utilization for both strategies while varying y.

The *overlapping* strategy gives the best performance as long as the extra computations are not intense. For less than 10^8 PI iterations, the simulation is slowed by less than 4% while the GPU utilization stays at the same level as Gromacs native. However, for a larger number of PI iterations, the simulation performance is dropping as y is increasing. For 10^9 iterations, the performance degradation is higher than 30% while the GPU utilization drops by 3%. For 10^{10} iterations, the degradation of both simulation performance and GPU utilization is even higher.

[5] measured with nvprof

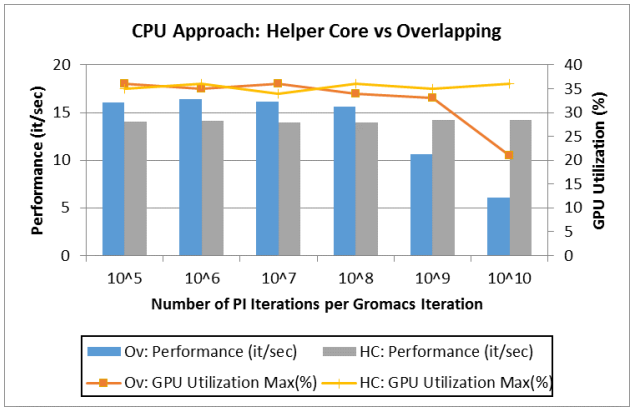


Fig. 3. Helper core (HC) and overlapping (Ov) strategies when increasing the number of PI iterations performed per Gromacs iteration.

The *helper core* strategy displays a more stable behavior. The initial cost with a low number of PI iterations is higher than the *overlapping* strategy. This is expected since one core and one GPU are removed from the simulation resources. However, as the in situ tasks are not hosted on the same resources as the simulation, the increasing computational charge is not affecting the simulation performance. Figure 3 shows that between 10^8 and 10^9 iteration, the *helper core* strategy becomes more efficient than the *overlapping* strategy.

The GPU utilization is reduced by the in situ analytics although the PI benchmark does not use the GPU. For the *helper Core* strategy, 1 GPU is not being used during all the simulation whereas for *overlapping*, the GPU utilization is lower than 36% for all the tests. Gromacs balances its computations between the CPU and GPU. However, previous results (Sec.4.2) showed that the CPU is the bottleneck of the simulation. As the PI benchmark stresses the CPU, Gromacs requires more time to launch the GPU kernels, leading to more idle time on the GPU with the *overlapping* strategy.

In summary, traditional in situ analytics running on the CPU fail to improve the global resource usage of Gromacs in hybrid mode. With the *helper core* strategy, one GPU is not used. With the *overlapping* strategy, the bottleneck of the simulation is more stressed by the analytics leading to more idle time on the GPUs. Others strategies are necessary to improve the global usage of resource.

4.5 Gromacs with GPU analytics

Rather than stressing the CPU, which is already the bottleneck of our simulation, we propose to perform in situ analytics on the GPU. We first used the multMatrix benchmark described in Section 3.5. As previously, Gromacs outputs data every 100 iterations. For each Gromacs output, y matrix multiplications are performed in total.

Figure 4 shows the simulation performance and GPU utilization for both strategies. As for the CPU benchmark, the *overlapping* strategy is more efficient for light computations. For less than y=3120 matrix multiplications per Gromacs output, the simulation frequency is reduced by less than 12%, while the GPU utilization stays at the same level as Gromacs native. However, for larger numbers

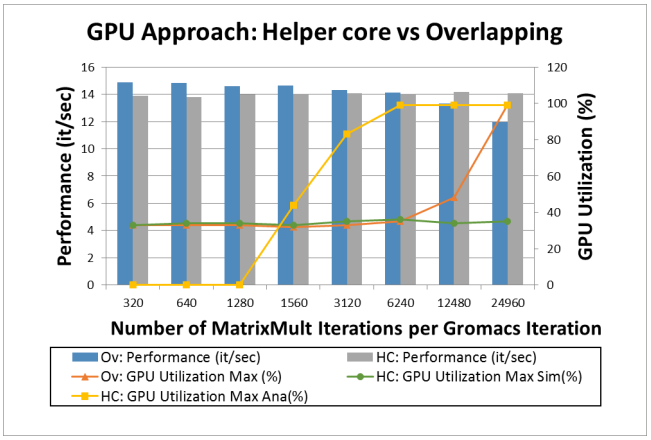


Fig. 4. Helper core (HC) and overlapping (Ov) strategies when increasing the number y of matrix multiplications perform per Gromacs Iteration. The GPU utilization for helper core strategy is split in two curves. GPU Utilization Max Sim indicates the maximum utilization of the 7 GPUs used by the simulation. GPU Utilization Max Ana is the GPU utilization for the GPU used by the in situ multMatrix.

of multiplications, the performance drops up to 20%, but the GPU utilization is increasing up to 99%.

The *helper core* strategy (referred as HC), displays stable behavior like the CPU benchmark. The initial cost with a low number of matrix multiplications is higher than the *overlapping* strategy. because the in situ tasks are not hosted on the same resources as the simulation, however the increasing computational charge is not affecting the simulation performance further. Figure 4 shows that between 6,240 and 12,480 matrix multiplications, the *helper GPU* strategy keeps a fixed cost and outperforms the *overlapping* strategy. This strategy also allows the GPU utilization to increase up to 99%.

We observe the following general trends. First, *overlapping* and *helper core* have similar behavior with CPU and GPU in situ analytics. The *overlapping* cost increases with a growing number of in situ computations. The *helper core* strategy has a higher initial cost for a small number of multiplications but does not further impact the simulation performance for higher computational cost. Secondly, performing in situ analytics on the GPU improves the GPU utilization. Because the simulation does not fully use the GPUs, other kernels can be launched with a limited impact on the simulation performance.

The multMatrix benchmark performs computations but does not transfer data between the CPU and GPU. Only the computational units are stressed. However, when performing data intensive analytics, data transfers must also be considered.

On our nodes, the 8 GPUs are connected to 3 PCI express ports. They share the same bus to transfer data from/to the GPU. We used the Bandwidth benchmark described in Section 3.5 to evaluate the impact of intensive in situ data analytics on the simulation performance. As previously, Gromacs outputs data every 100 iterations. For each Gromacs output, five rounds of data transfers are performed each with a given message size.

Figure 5 shows the simulation performance for both strategies. For this benchmark, we do not indicate the GPU utilization since the benchmark does not use

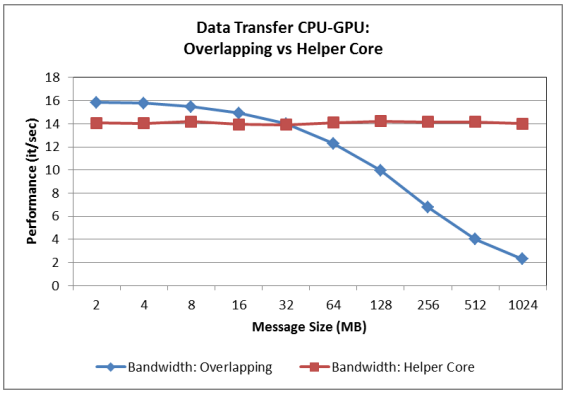


Fig. 5. Helper core and overlapping strategies when increasing the size of the message transferred.

the GPU. The *overlapping* strategy's impact on the simulation performance is less noticeable when the size of the message transferred is smaller than 32 MB. However, for a bigger message size, the performance drops up to 85% for message sizes of 1GB. The *helper core* strategy preserves a good isolation between the simulation and the in situ analytics and keeps a fixed cost. Two factors can explain this result. First, our GPUs are connected in a 3-3-2 pattern on the PCI express buses. We placed the dedicated GPU on the third bus which only hosts 2 GPUs. Therefore, the in situ analytics only disturb one GPU of the simulation on the bus which is the less stressed. Secondly, molecular dynamics codes are not data-intensive codes. The full molecular model represents only a few MB of data to transfer to/from the GPU. This leaves room on the buses to transfer data for other tasks.

4.6 Discussion

With these experiments, we have shown that we can apply the same placement strategies for GPU in situ analytics as for CPU strategies and observe similar behaviors. Moreover, in the case of Gromacs, using the GPUs for in situ analytics improves the GPU utilization while keeping a cost similar to the CPU strategies. This is possible for two reasons. First, the bottleneck of Gromacs is the CPU in our setup. This leaves more room on the GPU than on the CPU. Second, Gromacs is not a data-intensive application which makes it less sensitive to other data transfer.

Our goal in this article is to show that, in the case of hybrid simulations, there is also a need for placement flexibility to compute in situ analytics. For a pure CPU simulation, the placement strategy generally depends on the cost of the computation and the nature of the analytics. However, for a hybrid simulation, the balance of CPU/GPU computation is another parameter to take into account. Depending of whether the CPU or the GPU is the simulation bottleneck, in situ analytics should be performed on the less loaded resource if possible.

The simulation setup presented here leaves the GPUs idle a significant amount of time. This allows us to schedule heavy computational in situ tasks on the GPUs with the Overlapping strategy for a limited cost. However, other types of simulation might use the GPUs more intensively. For such scenarios, it is likely that for

the same computational in situ tasks, the helper core strategy becomes more efficient. We expect that, as the utilization of the GPUs by the simulation increases, the computational charge manageable at a reasonable cost by the Overlapping strategy would decrease in favor of the Helper core strategy.

5 Conclusion

We presented a study of the impact of in situ analytics in the case of the Gromacs simulation running with GPUs. We first showed that Gromacs is not natively able to fully use the available GPUs leading to underused resources. Then we showed that common in situ placement strategies such as using a dedicated core or asynchronous time-partitioning can amplify this phenomenon in the case of Gromacs. As an alternative approach, we used the GPUs to process analytics and applied the same placement strategies as for the CPU in situ analytics. CPU and GPU in situ analytics impact the simulation performance in a similar way with the same placement strategy. However, GPU analytics improve the global GPU utilization. We showed that, when considering hybrid simulation using GPUs, the balance between CPU and GPU computation should be taken into account when selecting a placement strategy for in situ analytics.

Our future work will focus on building real-case applications combining an hybrid simulation with hybrid in situ analytics. Tools such as VMD are available to perform analytics on the GPU. We will also extend this study to the case of In-Transit analytics. Some supercomputers such as *BlueWaters* have hybrid architectures combining CPU nodes and GPU nodes that can bring new trade-offs in the analytics placement. We will also study the recent feature in GPUs with compute capability 4.0 to launch CUDA kernels with a priority level. This feature can bring new opportunities especially for *Overlapping* strategies to perform in situ analytics at a lower priority than the simulation.

Acknowledgments. Experiments presented in this paper were carried out using the GridUIS-2 experimental testbed, being developed under the Universidad Industrial de Santander (SC3UIS) High Performance and Scientific Computing Centre, development action with support from UIS Vicerrectoria de Investigacin y Extension (VIE-UIS) and several UIS research groups as well as other funding bodies (http://www.sc3.uis.edu.co).

References

1. `http://philipwfowler.wordpress.com/2013/10/23/gromacs-4-6-scaling-of-a-very-large-coarse-grained-system/`
2. `http://www.hpcadvisorycouncil.com/pdf/GROMACS\_Analysis\_AMD.pdf`
3. Allard, J., Gouranton, V., Lecointre, L., Limet, S., Melin, E., Raffin, B., Robert, S.: FlowVR: A Middleware for Large Scale Virtual Reality Applications. In: Proceedings of Euro-par 2004. Pisa, Italia (August 2004)
4. Docan, C., Parashar, M., Klasky, S.: DataSpaces: an Interaction and Coordination Framework for Coupled Simulation Workflows. Cluster Computing 15 (2012)
5. Dorier, M., Antoniu, G., Cappello, F., Snir, M., Orf, L.: Damaris: How to Efficiently Leverage Multicore Parallelism to Achieve Scalable, Jitter-Free I/O. In: CLUSTER - IEEE International Conference on Cluster Computing. IEEE (Sep 2012)

6. Dorier, M., Sisneros, Roberto, R., Peterka, T., Antoniu, G., Semeraro, Dave, B.: Damaris/Viz: a Nonintrusive, Adaptable and User-Friendly In Situ Visualization Framework. In: LDAV - IEEE Symposium on Large-Scale Data Analysis and Visualization. Atlanta, United States (Oct 2013)
7. Dreher, M., Raffin, B.: A Flexible Framework for Asynchronous In Situ and In Transit Analytics for Scientific Simulations. In: Cluster, Cloud and Grid Computing (CCGrid), 2014 14th IEEE/ACM International Symposium on (May 2014)
8. Dreher, M., Piuzzi, M., Ahmed, T., Matthieu, C., Baaden, M., Férey, N., Limet, S., Raffin, B., Robert, S.: Interactive Molecular Dynamics: Scaling up to Large Systems. In: International Conference on Computational Science, ICCS 2013. Elsevier, Barcelone, Spain (Jun 2013)
9. Fabian, N., Moreland, K., Thompson, D., Bauer, A., Marion, P., Geveci, B., Rasquin, M., Jansen, K.: The Paraview Coprocessing Library: A Scalable, General Purpose In Situ Visualization Library. In: Large Data Analysis and Visualization (LDAV), 2011 IEEE Symposium on (Oct 2011)
10. Hagan, R., Cao, Y.: Multi-gpu load balancing for in-situ visualization. In: The 2011 International Conference on Parallel and Distributed Processing Techniques and Applications (2011)
11. Hess, B., Kutzner, C., van der Spoel, D., Lindahl, E.: GROMACS 4: Algorithms for Highly Efficient, Load-Balanced, and Scalable Molecular Simulation. Journal of Chemical Theory and Computation (2008)
12. Humphrey, W., Dalke, A., Schulten, K.: VMD – Visual Molecular Dynamics. Journal of Molecular Graphics 14, 33–38 (1996)
13. Klasky, S., Ethier, S., Lin, Z., Martins, K., Mccune, D., Samtaney, R.: Grid-Based Parallel Data Streaming implemented for the Gyrokinetic Toroidal Code. In: In Supercomputing Conference (SC 2003. IEEE Computer Society (2003)
14. Krone, M., Stone, J.E., Ertl, T., Schulten, K.: Fast Visualization of Gaussian Density Surfaces for Molecular Dynamics and Particle System Trajectories. In: EuroVis 2012 Short Papers. vol. 1 (2012)
15. Levine, B.G., Stone, J.E., Kohlmeyer, A.: Fast analysis of molecular dynamics trajectories with graphics processing units Radial distribution function histogramming. Journal of Computational Physics 230(9) (2011)
16. Li, M., Vazhkudai, S.S., Butt, A.R., Meng, F., Ma, X., Kim, Y., Engelmann, C., Shipman, G.: Functional Partitioning to Optimize End-to-End Performance on Many-core Architectures. In: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis. SC '10, IEEE Computer Society, Washington, DC, USA (2010)
17. Lofstead, J.F., Klasky, S., Schwan, K., Podhorszki, N., Jin, C.: Flexible IO and Integration for Scientific Codes Through the Adaptable IO System (ADIOS). In: 6th international workshop on Challenges of large applications in distributed environments (2008)
18. Lorendeau, B., Fournier, Y., Ribes, A.: In-Situ Visualization in Fluid Mechanics Using Catalyst: A Case Study for Code Saturne. In: Large-Scale Data Analysis and Visualization (LDAV), 2013 IEEE Symposium on (Oct 2013)
19. Moreland, K.: Oh, $#! Exascale! The Effect of Emerging Architectures on Scientific Discovery. In: High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion: (Nov 2012)
20. Phillips, J.C., Braun, R., Wang, W., Gumbart, J., Tajkhorshid, E., Villa, E., Chipot, C., Skeel, R.D., Kal, L., Schulten, K.: Scalable molecular dynamics with NAMD. Journal of Computational Chemistry 26(16), 1781–1802 (2005)
21. Pronk, S., Pall, S., Schulz, R., Larsson, P., Bjelkmar, P., Apostolov, R., Shirts, M.R., Smith, J.C., Kasson, P.M., van der Spoel, D., Hess, B., Lindahl, E.: Gromacs 4.5: a high-throughput and highly parallel open source molecular simulation toolkit. Bioinformatics (2013)

22. Singh, A., Balaji, P., Feng, W.c.: GePSeA: A General-Purpose Software Acceleration Framework for Lightweight Task Offloading. In: Proceedings of the 2009 International Conference on Parallel Processing. ICPP '09, IEEE Computer Society, Washington, DC, USA (2009)
23. Soumagne, J., Biddiscombe, J.: Computational Steering and Parallel Online Monitoring Using RMA Through the HDF5 DSM Virtual File Driver. In: Proceedings of the International Conference on Computational Science, ICCS 2011. vol. 4. Singapore (Jun 2011)
24. Stone, J.E., Hardy, D.J., Ufimtsev, I.S., Schulten, K.: Gpu-accelerated molecular modeling coming of age. Journal of Molecular Graphics and Modelling 29(2) (2010)
25. Stone, J.E., Kohlmeyer, A., Vandivort, K.L., Schulten, K.: Immersive molecular visualization and interactive modeling with commodity hardware. In: ISVC'10. Springer-Verlag (2010)
26. Stone, J.E., McGreevy, R., Isralewitz, B., Schulten, K.: Gpu accelerated analysis and visualization of large structures solved by molecular dynamics flexible fitting. Faraday Discuss. 169 (2014)
27. Stone, J.E., Vandivort, K.L., Schulten, K.: Gpu-accelerated molecular visualization on petascale supercomputing platforms. In: Proceedings of the 8th International Workshop on Ultrascale Visualization. UltraVis '13, ACM, New York, NY, USA (2013)
28. Tu, T., Yu, H., Ramirez-Guzman, L., Bielak, J., Ghattas, O., Ma, K.L., O'Hallaron, D.: From Mesh Generation to Scientific Visualization: An End-to-End Approach to Parallel Supercomputing. In: SC 2006 Conference, Proceedings of the ACM/IEEE (Nov 2006)
29. Vishwanath, V., Hereld, M., Papka, M.: Toward Simulation-Time Data Analysis and I/O Acceleration on Leadership-Class Systems. In: Large Data Analysis and Visualization (LDAV), 2011 IEEE Symposium on (Oct 2011)
30. Whitlock, B., Favre, J.M., Meredith, J.S.: Parallel In Situ Coupling of Simulation with a Fully Featured Visualization System. In: Proceedings of the 11th Eurographics Conference on Parallel Graphics and Visualization. EGPGV '11, Eurographics Association (2011)
31. Yu, H., Wang, C., Grout, R., Chen, J., Ma, K.L.: In situ visualization for large-scale combustion simulations. Computer Graphics and Applications, IEEE (2010)
32. Zhao, G., Perilla, J.R., Yufenyuy, E.L., Meng, X., Chen, B., Ning, J., Ahn, J., Gronenborn, A.M., Schulten, K., Aiken, C.: Mature HIV-1 Capsid Structure by Cryo-electron Microscopy and All-Atom Molecular Dynamics (2013)
33. Zheng, F., Abbasi, H., Docan, C., Lofstead, J., Liu, Q., Klasky, S., Parashar, M., Podhorszki, N., Schwan, K., Wolf, M.: PreDatA - Preparatory Data Analytics on Peta-Scale Machines. In: Parallel Distributed Processing (IPDPS), 2010 IEEE International Symposium on (2010)
34. Zheng, F., Abbasi, H., Cao, J., Dayal, J., Schwan, K., Wolf, M., Klasky, S., Podhorszki, N.: In-situ I/O Processing: A Case for Location Flexibility. In: Proceedings of the Sixth Workshop on Parallel Data Storage. PDSW '11, ACM, New York, NY, USA (2011)
35. Zheng, F., Yu, H., Hantas, C., Wolf, M., Eisenhauer, G., Schwan, K., Abbasi, H., Klasky, S.: 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, ACM (2013)
36. Zheng, F., Zou, H., Eisenhauer, G., Schwan, K., Wolf, M., Dayal, J., Nguyen, T.A., Cao, J., Abbasi, H., Klasky, S., Podhorszki, N., Yu, H.: FlexIO: I/O Middleware for Location-Flexible Scientific Data Analytics. In: Proceedings of the 2013 IEEE 27th International Symposium on Parallel and Distributed Processing. IPDPS '13, IEEE Computer Society (2013)