Data Provenance for Experiment Management of Scientific Applications on GPU

Sejin Kim
The Department of Computer Science
Sookmyung Women's University
Seoul, Korea
wonder960702@gmail.com

Jisun Oh
The Department of Computer Science
Sookmyung Women's University
Seoul, Korea
jsoh8088@gmail.com

Yoonhee Kim
The Department of Computer Science
Sookmyung Women's University
Seoul, Korea
yulan@sookmyung.ac.kr

***Abstract*— Graphics Processing Units (GPUs) are getting popularly utilized for multi-purpose applications in order to enhance highly performed parallelism of computation. As memory virtualization methods in GPU nodes are not efficiently provided to deal with diverse memory usage patterns for these applications, the success of their execution depends on exclusive and limited use of physical memory in GPU environments. Therefore, it is important to predict a pattern change of GPU memory usage during runtime execution of an application. Data provenance extracted from application characteristics, GPU runtime environments, input, and execution patterns from runtime monitoring, is defined for supporting application management to set runtime configuration and predict an experimental result, and utilize resource with co-located applications. In this paper, we define data provenance of an application on GPUs and manage data by profiling the execution of CUDA scientific applications. Data provenance management helps to predict execution patterns of other similar experiments and plan efficient resource configuration.**

Keywords— data provenance, GPU, scientific workflow

I. Introduction

Graphics Processing Units(GPUs) are recently used for multi-purpose applications due to their high computability stems from enabling parallelism of computation and low energy consumption. High-level language frameworks such as Compute Unified Device Architecture(CUDA) and Open Computing Language(Open CL) have improved GPU programmability. These language frameworks provide memory virtualization features such as Unified Memory and virtual address spaces, which can overcome GPU memory limitation issues while keeping applications maintaining high performance. However, GPU memory virtualization solutions often fail to show comparably better performance of applications running on GPUs to CPUs. A number of challenges exist on making a success of application's executions, which relies on scheduling schemes in GPU environments [4]. Therefore, the prediction of applications' execution patterns has been a key metric for an application scheduling scheme. It helps to decide how long each process should wait for a free resource. That means it provides insight about resource availability [5].

There has been a lot of research focusing on discovering a workflow pattern that frequently appears and predicting the completion of a specific workflow by applying data mining techniques on data provenance. The execution patterns deriving from accumulated data provenance leads to good utilization of resource [6]. A workload composed of many applications of which have uniform execution patterns, results in diverse and unexpected results depending on the condition of input data and features of an execution environment. Therefore, besides of data provenance, management such as runtime monitoring and adjustment of an execution plan is needed to deploy into a GPU execution environment. The management resolves Out Of Memory(OOM) failure of a workload, caused by the limit of GPU memory capacity. This perspective shows a method of workflow scheduling and optimization. In this paper, we define data provenance of an application running on GPUs and collect its data by profiling the execution of CUDA scientific applications such as LAMMPS [7] and GROMACS [8]. Also we perform experiments to show that management with data provenance can resolve OOM failure and enhance overall workload performance by utilizing GPU resource.

The organization of this paper is as follows. In Section II, we discuss some related works. Section III defines data provenance for experiments to predict execution patterns and the succeeding section shows tracing provenance of scientific applications from experiments. Section V shows experiments that workload is optimized based on provenance information, and we conclude the paper in Section VI.

II. Related works

[5] presents their approach to predicting execution time of programs running on Grid environments, where processing of resources is complicated due to the distribution of resources, by using data provenance. They recorded three types of data provenance including pre-execution, execution, and environment provenance feature set. The performance prediction is implemented by training machine learning models using this data. However, characteristics of GPU applications and GPU resources are not included in the feature set since they don't consider GPU applications. Therefore, a definition of provenance information is needed that reflects characteristics of GPU applications and GPU resources for predicting execution patterns accurately.

[13] proposes a provenance scheme for a cloud computing environment, which is based on virtualization. They present this provenance framework of the dynamic and abstract environment. However, the impact on background running workload is not considered in this work, since the execution time of GPU workload is greatly affected by the execution of other GPU workloads. Our approach to provenance framework reflects the effect of background load and GPU resources that are not addressed in this paper.

III. Definition of Provenance information for Experiments

We propose a method of collecting common/specific data for defining provenance information of GPU applications and classifying data provenance sets from collected data.

A. Collecting Data for Defining GPU Provenance Information

Provenance information of applications is captured and recorded. Meanwhile, Data that has an impact on an execution pattern of applications is also collected. Data provenance defined in this paper can be applied for every GPU's manufacturer. As the concept of warp is only limited to CUDA programming model, data provenance about warp is additional data for NVIDIA GPU.

We use NVIDIA System Management Interface(nvidia-smi)[9] and nvprof profiling tool[10] to collect related data. Nvidia-smi is a utility to monitor and manage capabilities, based on top of the NVIDIA Management Library(NVML)[9]. Nvidia-smi provides information on gpu utilization, which shows the percentage of each kernel consumes that executing on the GPU, utilization of Streaming Multiprocessor(SM) and so forth. Nvprof is a profiling tool that can trace CUDA activities. Nvprof is able to profile memory transfer throughput, occupancy defined as the ratio of active warps on an SM to the maximum number of active warps supported by the SM and so forth.

B. Grouping Data provenance Sets from Collected Data

Data provenance is related to an execution pattern of applications, input data, and execution environment. It can be defined to predict execution pattern. To perform the collection of provenance data, we employed monitoring tools, which have been described in the section 3.A. We recorded runtime processing data by using nvprof. To monitor GPU resource usage, we utilized nvidia-smi. The data sets from collected data are grouped into three sets : execution features, pre-execution features, environment features.

1) Set 1: Execution features: Execution features contain runtime information about applications. Allocated memory by each process and GPU utilization are recorded every 5 seconds by using nvidia-smi. Host to Device memory copy throughput, Device to Host memory copy throughput, SM efficiency, which is the percentage of time at least one warp is active on a specific multiprocessor, occupancy and ratio of the average active threads per warp to the maximum number of threads per warp supported on a multiprocessor(warp efficiency) are reported when an application completes its execution. Their mean value is used as the provenance data since the variance of them is small enough. We calculate the total runtime of a process by using the formula

$$\text{execution time} = \text{timestamp}_{\text{finish}} - \text{timestamp}_{\text{start}}.$$

Important features comprising execution features are shown in columns 1 and 2 of Table I.

2) Set 2: Pre-execution features: Pre-execution features are collected before the actual execution is started, which consists of input data onto process and its size. The denotation of input parameters varies depending on applications because the diversity of required information on each application, which contains input data, mesh size, etc. Considerable features involved in pre-execution features are shown in column 3 of Table I.

3) Set 3: Environment features: Environment features categorized into two types: static features and dynamic features. Static features refer to system hardware information. Dynamic features indicate data that changes over time of execution, and hence are recorded at intervals. Static features comprise total installed GPU memory, GPU speed, an architecture of GPU and PCI-e bandwidth that measures how fast the GPU connection is to the CPU's PCI-e controller, which controls the GPU's access to system memory or RAM. They also contain the maximum number of warps, thread blocks, threads and the maximum size of shared memory per SM that influence on concurrent application Execution. As an execution pattern of a process of an idle machine and a machine with background load differs considerably, we recorded background load on GPU in dynamic features. The background load includes the percentage of SM allocated by an active process and PCIe throughput between CPU and GPU. We recorded them every 5 seconds exploiting nvidia-smi daemon. Columns 4,5 and of Table I show a list of important static and dynamic environment features.

TABLE I. PROVENANCE DATA SET

Environment features		
Static		*Dynamic*
GPU memory total	Warps per SM	Free GPU memory
GPU speed	Thread blocks per SM	SM used
GPU architecture	Shared Memory per SM	PCIe Rx throughput
PCIe bandwidth	Threads per SM	PCIe Tx throughput
Execution features		***Pre-execution features***
GPU memory used	Device to Host Throughput	Input size
Execution time	Occupancy	Input parameters
GPU utilization	SM efficiency	
Host to Device Throughput	Warp execution efficiency	

IV. TRACING PROVENACE OF SCIENTIFIC APPLICATIONS

In this section CUDA scientific application LAMMPS and GROMACS are profiled to collect provenance information. To collect provenance information these applications are executed on configuration of Intel(R)Core(TM) i7-5820K and Nvidia Titan XP.

A. LAMMPS

LAMMPS is a classical molecular dynamics code based on material modeling. It's an acronym for Large-scale Atomic/Molecular Massively Parallel Simulator.

1) Execution provenance information: Execution provenance information that is recorded every 5 seconds is shown in Fig. 1.

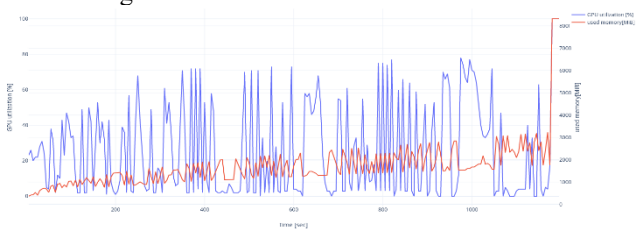

Fig. 1. Execution provenance features of LAMMPS

LAMMPS uses 363MB~8.3GB of memory during its execution. GPU utilization of the application is about 25%, as

data transfer between host and device appears frequently. 1176 seconds after the start of the execution, it is presented as compute-intensive using 8.3GB of memory and showing 100% utilization of GPU. The total runtime of the process is 1196 seconds, and the average memory throughput from Host to Device is 5.865GB/s, the average memory throughput from Device to Host is 8.561 GB/s. The average of SM efficiency is 82.12%, average warp efficiency is 90.10% and the average occupancy is 0.55.

2) Pre-execution provenance information: Input parameters of LAMMPS consist of used package of LAMMPS, input file, binsize that affects the building pairwise neighbor lists, and variables x, y, z which indicate dimensionality of LJ system and the number of creating particles. In this experiment, we used kokkos package, an input file of Leonard Jones 3D melt example [11]. We set the binsize to 2.8 and variables x, y, z to 8.

3) Environment provenance information: Static environment features of the experiment are shown as TABLE II.

TABLE II. STATIC ENVIRONMENT FEATURES

Static Environment features			
GPU memory total	11.91 GB	Warps per SM	64
GPU speed	1582 MHz	Thread blocks per SM	32
GPU architecture	TITAN Xp	Shared Memory per SM	96KB
PCIe bandwidth	32GB/s	Threads per SM	2048

Fig 2. is a graph of tracing total free memory, SM usage and PCIe bus throughput which is a background load that changes over time dynamically in execution.

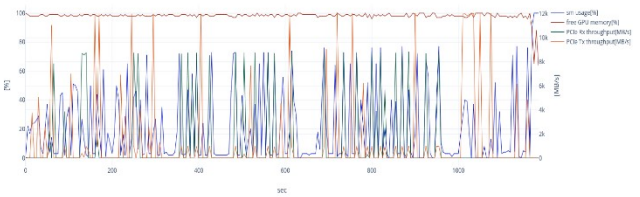

Fig. 2. Dynamic environment features of LAMMPS

B. GROMACS

1) Execution provenance information: Execution provenance information that is recorded every 5 seconds as shown in Fig. 3. In the case of GROMACS, its total runtime of the process is 636 seconds and it uses 171MB~537MB of memory. GROMACS is a compute-intensive application, as its average GPU utilization is 69% with frequent computation. The average memory throughput from Host to Device is 11.75GB/s and the average memory throughput from Device to Host is 12.81GB/s. The average of SM efficiency is 99.98%, average warp efficiency is 71.73% and the average occupancy is 0.49.

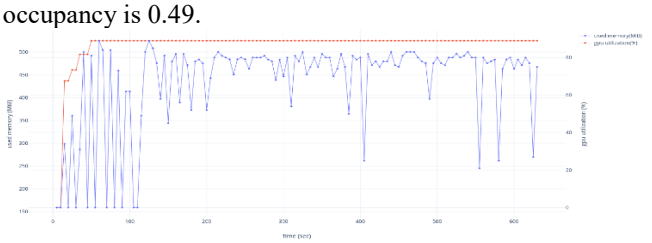

Fig. 3. Execution provenance features of GROMACS

2) Pre-execution provenance information: Required parameters are determined by which of the programs is performed. In this experiment, we performed mdrun engine that simulates Molecular Dynamics and set the number of steps to conduct 10000 steps. We used water GMX50 bare benchmark data [12] as input data and its size is 72.017KB.

3) Environment provenance information: Since the execution environment of GROMACS and LAMMPS are the same, static environment features are as shown in TABLE II. Fig 4. is a graph of tracing total free memory, SM usage and PCIe bus throughput which is a background load that changes over time dynamically in execution.

Fig. 4. Dynamic environment features of GROMACS

V. EXPERIMENT

We perform experiments by predicting execution patterns through provenance information traced in section IV. We compared GPU resource utilization and execution time of identical workload based on the presence or absence of provenance information.

Experiment environment, applications, provenance information for this experiments are explained in prior section. Workloads using for experiments are 1) consisted of two LAMMPS applications and 2) consisted of a LAMMPS application and two GROMACS applications. As a baseline of application deployment, we choose random deployment of applications.

A. Consisted of two LAMMPS applications

Fig. 5. shows result of deploying applications randomly without provenance information. As memory usage of applications is unknown, two LAMMPS applications are launched concurrently. It can be seen that only one task terminates normally with Out Of Memory(OOM) occurring approximately 1625 seconds after the start of the experiment since LAMMPS uses 8.3GB memory 1176 seconds after its launch and GPU total memory of experiment environment is 11.91GB.

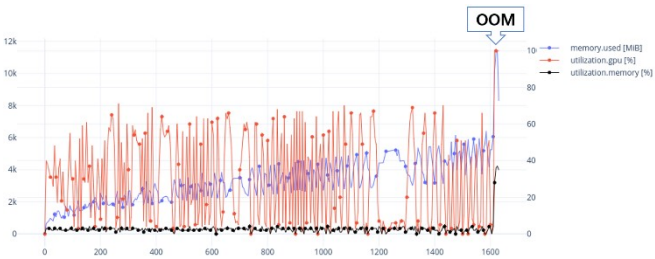

Fig. 5. Random application deployment of first workload

Through provenance information, OOM was predicted in advance and executed by giving a time difference between

launch of two tasks. Figure 6 shows that both tasks are successfully terminated. A time difference of 60 seconds between the two tasks was given, and after 1690 seconds of experimentation, both tasks are finished normally.

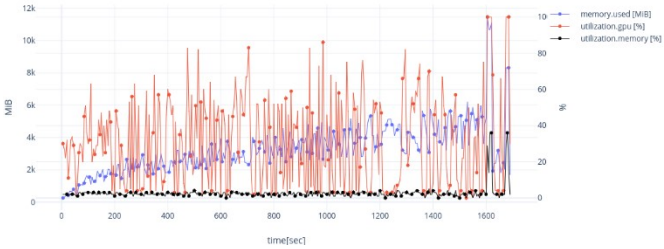

Fig. 6. Application deployment of first workload with provenance information

B. *Consisted of a LAMMPS application and two GROMACS applications*

Figure 7 shows the result of a random placement for three tasks of the second workload. The sequence of tasks through the random placement is that two GROMACS launch simultaneously, and LAMMPS is subsequently executed. The two GROMACS jobs executed simultaneously end in 1380 seconds. Simultaneous launch of two GROMACS jobs takes longer than two jobs performed sequentially. This is because it was executed without considering compute-intensive characteristic of GROMACS. The LAMMPS application is dispatched after the end of the two GROMACS applications. The entire workload took 2576 seconds to run.

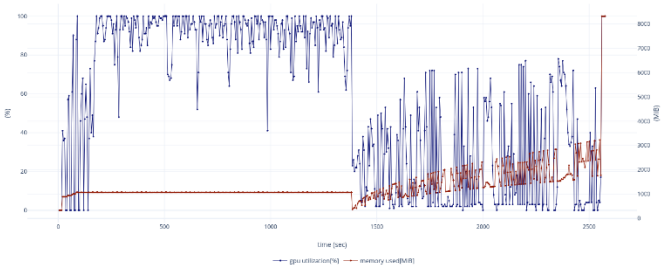

Fig. 7. Random application deployment of second workload

On the other hand, Figure 8 shows the results of experimenting with the simultaneous placement of LAMMPS application, which is the focus of memory operations, and GROMACS application, which is compute-intensive. The result shows that all three tasks took a total of 1,855 seconds to complete, a reduction of approximately 38% compared to a random application deployment. This shows that the two tasks of central use of different resources could be arranged together to reduce time by increasing the efficiency of resource use.

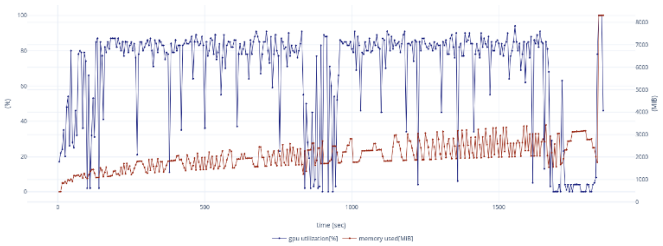

Fig. 8. Application deployment of second workload with provenance information

VI. Conclusion

In this paper, we defined data provenance for experiments with GPU applications and managed data by profiling the execution of CUDA scientific applications LAMMPS and GROMACS.

By defining provenance information, data from past experiments can be accumulated and managed systematically. Based on data provenance management, execution management allows us to minimize unnecessary experiments. In addition, it can be used for applications scheduling and planning efficient resource configuration [5,13]. In this paper, we show that management with data provenance can resolve OOM failure and enhance overall workload performance by utilizing GPU resource through experiments.

We are planning to collect provenance information about GPU CUDA scientific applications. In addition, we are going to propose a scheduling strategy including memory usage hint by predicting execution time based on provenance information.

Acknowledgment *(Heading 5)*

This research was supported by Next-Generation Information Computing Development Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT and Future Planning (2015M3C4A7065646) and This work has supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT)(No. NRF-2017R1A2B4005681).

References

[1] CUDA Toolkit, https://developer.nvidia.com/cuda-toolkit

[2] Open CL Overview, https://www.khronos.org/opencl/

[3] Beyond GPU Memory Limits with Unified Memory on Pascal, https://devblogs.nvidia.com/beyond-gpu-memory-limits-unified-memory-pascal/

[4] Ausavarungnirun, Rachata, et al. "Mosaic: a GPU memory manager with application-transparent support for multiple page sizes." Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture. ACM, 2017.

[5] Malik, Muhammad Junaid, Thomas Fahringer, and Radu Prodan. "Execution time prediction for grid infrastructures based on runtime provenance data." Proceedings of the 8th Workshop on Workflows in Support of Large-Scale Science. ACM, 2013.

[6] Suh, Young-Kyoon, and Ki Yong Lee. "A survey of simulation provenance systems: modeling, capturing, querying, visualization, and advanced utilization." Human-centric Computing and Information Sciences 8.1 (2018): 27.

[7] LAMMPS, https://lammps.sandia.gov/

[8] GROMACS, http://www.gromacs.org/

[9] NVIDIA System Management Interface, https://developer.nvidia.com/nvidia-system-management-interface

[10] nvprof, https://docs.nvidia.com/cuda/profiler-users-guide/index.html#nvprof-overview

[11] Leonard Jones 3D melt example, https://lammps.sandia.gov/inputs/in.lj.txt

[12] water GMX50 bare, ftp://ftp.gromacs.org/pub/benchmarks/water_GMX50_bare.tar.gz

[13] Imran, Muhammad, and Helmut Hlavacs. "Provenance in the cloud: Why and how?." CLOUD COMPUTING (2012): 106-112.