Midas: A System for Achieving "Golden" Reproducibility in HPC

A.C. Minor & W. Feng | Dept. of Computer Science | Virginia Tech | acminor@vt.edu & wfeng@vt.edu

With the exponentially improving serial performance of CPUs from the 1980s and 1990s slowing to a standstill by the 2010s, the HPC community has seen parallel computing (via multi-core CPUs, many-core GPUs and TPUs, and even FPGAs) become ubiquitous, which, in turn, has led to a proliferation of parallel programming models, e.g., CUDA, OpenACC, OpenCL, and SYCL. This diversity in hardware platform and programming model has forced application users to port their codes from one hardware platform to another (e.g., CUDA on NVIDIA GPU to OpenCL on AMD GPU) and demonstrate reproducibility via ad-hoc unit testing. To more rigorously ensure reproducibility between codes, we propose Midas, a system to ensure that the results of the original code match the results of the ported code by leveraging the power of snapshots to capture the state of a system before and after the execution of a kernel.

1 Midas

To achieve reproducible cross-platform code, we created Midas, a system composed of three main subsystems: Golden, Midas Touch, and Flamel, as shown in Fig. 1.

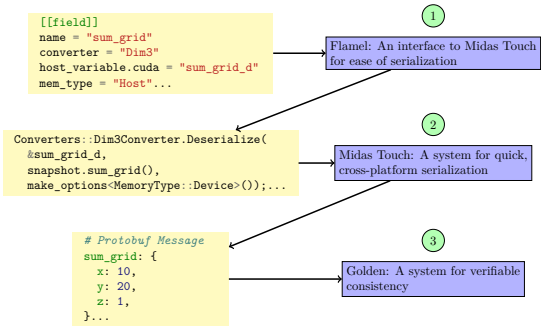

Figure 1: Midas System Overview

Golden serves as the foundation of Midas. It reliably snapshots data to disk and compares the snapshots. Its design was inspired by the C++ library Approval Tests [1] and uses the concept of "goldens." Goldens are a snapshot of what your data (input or output) should be for a given function. Current snapshots and goldens can then be compared to determine if your cross-platform program has changed its behavior. The general process for using Golden is to save a snapshot from a verified source and then use this to validate future snapshots.

To use Golden directly, snapshots must be handmade for each function that we want to snapshot. Because this process is tedious and error-prone, we created Midas Touch, an interface and collection of converters to handle common serialization tasks. If needed, users can write custom converters using the provided Midas Touch interfaces.

Flamel is a high-level description language to generate Midas Touch code. As Midas Touch is a generic library, it can be verbose to use. Also, it is embedded into a programming language, and hence, must follow the syntax for that language. Flamel abstracts this to a TOML interface. TOML [2] is a data-description language, similar to JSON, that allows one to specify metadata for a snapshot and not worry about the Midas Touch code generated for that snapshot.

This project was supported in part by NSF I/UCRC CNS-1822080.

2 Challenges

Non-determinism can change snapshots between runs. If the change was due to a bug across platform ports, we could validate said change and update the snapshot or port; but if it is due to non-determinism, then the snapshot will continuously change, whether a bug was introduced or fixed. A common source of non-determinism in parallel HPC code is atomic operations across threads.

Fig. 2 shows an example of a reversible kernel.[1] For this kernel, we are processing a set of 3D arrays. If an array cell has been previously updated, we move to the next array in the stack and process the same cell location. This forms a non-deterministic ordering of values along atomic time. Furthermore, for our kernel, the generated values are deterministic in value and contain the thread identification (i.e., id) that processed them. So, we have a non-deterministic ordering of deterministic values with thread id that can be used to impose an output ordering. That is, we sort array cells over time from smallest to largest thread, which represents a possible execution ordering. Thus, we can compare any two results via this deterministic transformation.

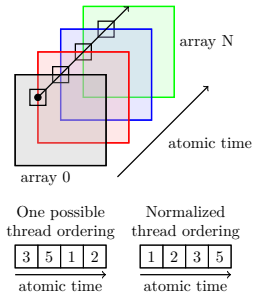

Figure 2: Reversible Non-determinism Example

3 Results

As a case study, we use FenZi, a molecular dynamics code [3], originally written in CUDA and then largely auto-translated to OpenCL [5] via CU2CL [4]. Midas validated the reproducibility of the OpenCL code from the original CUDA code. Our poster will show how.

References

[1] Approval Tests. https://github.com/approvals/ApprovalTests.cpp.

[2] T. Preston-Werner et al. TOML: Tom's Obvious Minimal Language. https://github.com/toml-lang/toml.

[3] N. Ganesan, M. Taufer, B. Bauer, and S. Patel. FenZi: GPU-Enabled Molecular Dynamics Simulations ... In *IEEE Int'l Parallel and Distributed Processing Symp. Workshops and PhD Forum*, 2011.

[4] G. Martinez, M. Gardner, and W. Feng. CU2CL: A CUDA-to-OpenCL Translator for Multi- and Many-Core Architectures. In *IEEE Int'l Conf. on Parallel & Distributed Systems*, 2011.

[5] P. Sathre, M. Gardner, and W. Feng. On the Portability of CPU-Accelerated Applications via Automated Source-to-Source Translation. In *Proc. Int'l Conf. on High Performance Computing in Asia-Pacific Region*, 2019.

[1] By reversible, we mean that the operation can be made deterministic through removing and/or imposing an ordering.