A Software Environment for Ultrahigh-Resolution ELM Development on GPUs

FRANKLIN EAGLEBARGER, Pellissippi State Community College, USA
DALI WANG, FENGMING YUAN, PETER SCHWARTZ, CHRIS LAYTON, ROBERT WEISKOPFF, DANIEL RICCIUTO, PETER THORNTON, Oak Ridge National Laboratory, USA

A software tool, called SPEL, has been developed to port and optimize the ultrahigh-resolution ELM (uELM) code onto GPUs within a functional unit test framework. To promote the widespread adoption of this approach for community-based uELM development, this poster presents a portable software environment that enables efficient development of the uELM code on GPUs. The standalone software environment, which utilizes Docker, contains all the necessary code, libraries, and system software required for uELM development using SPEL. The process involved in this study includes identifying a Docker image that supports GPU, configuring and simulating ELM at the site level, capturing reference solutions, testing uELM functional units, and generating and optimizing code that is compatible with GPUs. The effectiveness of this methodology is demonstrated through a case study.

CCS Concepts: • **Applied computing** → **Environmental sciences**.

Additional Key Words and Phrases: Exascale Energy Earth System Model, E3SM Land Model, Ultrahigh-resolution ELM, SPEL

ACM Reference Format:
Franklin Eaglebarger and Dali Wang, Fengming Yuan, Peter Schwartz, Chris Layton, Robert Weiskopff, Daniel Ricciuto, Peter Thornton. 2023. A Software Environment for Ultrahigh-Resolution ELM Development on GPUs. In . ACM, New York, NY, USA, 2 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Designing and refactoring complex scientific code, such as the E3SM land model (ELM), for new computing architectures poses significant challenges. ELM is a data-centric application with half a million lines of code. We have designed a novel method for the ultrahigh-resolution ELM (uELM) development on Exascale computers using compiler directives (OpenMP/OpenACC). The first version of uELM has been implemented using OpenACC, targeting the NVidia GPUs in the Summit supercomputer at Oak Ridge National Laboratory. During the implementation process, we have developed a Software tool for Porting ELM (SPEL), which has greatly facilitated the generation, verification, and performance tuning of the uELM code. The initial implementation of uELM for NVidia GPUs on Summit has been a success, with over 98% of the ELM code being automatically generated and tuned by SPEL. The GPU-based uELM have exhibited 2-time speedup compared to the original ELM code for CPUs on fully loaded Summit nodes.

2 METHOD

To develop a portable, standalone software environment for efficient community-based development of uELM code on GPUs using SPEL, we have developed a software environment (based on Docker) that utilizes OS-level virtualization to deliver all the code, libraries, and system software need for uELM development in following five steps:

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2023 Association for Computing Machinery.
Manuscript submitted to ACM

1. Identify a Docker image with Nvidia GPU support: First, we need a Docker image that supports GPUs. After attempting to customize docker images for Nvidia GPU support without complete success, we decide to focus on identifying and evaluating preconfigured NVidia-docker images (nvcr.io) that already include compatible NVHPC programming environments and the necessary CUDA driver support. This allows us to efficiently run our simulations and computations on Nvidia GPUs across multiple computational resources.

2. Customize a Docker image for ELM simulation: On the top of the base nvidia-docker image, we integrate the ELM code, including the Offline ELM Model Testbed (OLMT) for automatic ELM configuration and simulation at the site level. By leveraging our previous experience in developing customized docker images from the NGEE-Arctic project, we generate a new docker image for site-based ELM simulation on CPUs through a multi-step docker image creation.

3. Capture reference solution: To validate the accuracy and performance of our ELM model, we use SPEL functions to instrument code segment into the ELM source code. These code segments capture and save the input and output data streams from target ELM functional units from a reference ELM simulation on CPUs. These input and output data streams are then used for testing uELM functional units.

4. uELM functional unit testing using SPEL: Before porting the uELM code on GPUs, we use SPEL functions to create standalone functional unit modules on CPUs first. Technically. SPEL constructs a unit test driver that handles the initialization and reading of input parameters and data streams (from the previous step), executes a target uELM functional unit module, and saves the output data streams. Finally, SPEL compares the output data streams with those from the reference solution to verify the correctness and robustness of the uEML module in its standalone format.

5. GPU-ready code generation and optimization via SPEL: SPEL provides functions to generate and optimize uELM GPU code within the standalone functional unit testing framework. Users also have the flexibility to develop their own strategies for quickly porting the GPU code, efficiently utilizing GPU resources, and maximizing the performance of individual uELM functional units.

To showcase the effectiveness and efficiency of our proposed methodology within our portable software environment, the poster also presents a case study of the creating and optimizing of a specific uELM functional test called LakeTemperature. Several computing resources are used in this study, including Mac and Windows laptops/desktops, an NVidia DGX station (with 4 V100), and a Virtual Machine with GPU system (1 A100) in the ORNL Research Cloud using the OpenStack platform. Several Docker files and configuration files that define the environment for running applications in containers, as well as associated Docker images, are available on GitHub and DockerHub (docker.io).

3 RESULTS

This poster presents information about layered architecture and software components in docker images, such as the base operating system, libraries, and packages, along with the ELM software structure within the E3SM system. The poster also outlines the process of creating ELM cases and simulations using OLMT. Additionally, the poster mentions the workflow for functional unit testing using SPEL. Finally, as a demonstration, the poster provides the steps to launch and execute the LakeTemperature module on both CPUs and GPUs using SPEL.

ACKNOWLEDGMENTS

Mr. Franklin Eaglebarger is funded by the Community College Internships (CCI) program in the US Department of Energy (DOE). This research leverages previous efforts that were supported by E3SM and NEGG-Arctic, funded by DOE's Office of Biological and Environmental Research. This research used resources of the OLCF and CADES at the Oak Ridge National Laboratory, which are supported by the DOE's Office of Science under Contract No. DE-AC05-00OR22725.