LA-UR-21-28323
Approved for public release; distribution is unlimited.

Title:	Task Parallelism to Optimize Performance of Environmental Modeling Software
Author(s):	Denlinger, Althea Rebekah
Intended for:	Report
Issued:	2021-08-19

Task Parallelism to Optimize Performance of Environmental Modeling Software

Althea R. Denlinger

University of New Mexico

August 17, 2021

Abstract

Climate modeling is an integral part of environmental research, from studying rare phenomena to predicting future climate trends. The need for more accurate models is only growing, but as climate modeling capabilities advance, existing workflows require optimization to recoup performance [1]. A solution comes in the form of task parallelism, a novel programming capability that provides an opportunity for optimization at execution time by allowing tasks to be executed in parallel, reducing runtime significantly. Using Parsl, an intuitive and scalable parallel scripting library for Python, we implement task parallelism within support software to aid in the continuous advancement of climate modeling technology.

In June of 2021, I began working at Los Alamos National Laboratory (LANL) under the mentorship of Dr. Xylar Asay-Davis, Dr. Darren Engwirda, and Dr. Luke Van Roekel. I joined the team behind the Energy Exascale Earth System Model (E3SM) project, a collaboration between eight different DOE laboratories to advance earth system modeling technology. Earth and climate models like E3SM are expansive simulation systems that require massive amounts of data to represent realistic conditions. Many existing elements of E3SM need to be updated and optimized continuously to compensate for newly added data and functionalities [1]. Illustrating this, E3SM simulation testing needs to increase in frequency, but the runtime requirements of model components are too extensive for this to be feasible. Thus, a project was created to reconstruct the execution strategies within E3SM. This project planned to utilize task parallelism, an unconventional and highly advanced programming practice that would allow multiple processes to run simultaneously, barring any dependencies between them. Dr. Asay-Davis recognized the Model for Prediction Across Scales (MPAS) ocean and land ice components of E3SM as a cohesive starting point, and the beginning of my internship marked the project's commencement.

MPAS itself was not the target for this modification, but instead the Configuration of MPAS Setups (COMPASS) software that initializes and executes MPAS test cases. Since task execution began within COMPASS, parallel task execution would need to as well [1]. Task parallelism is an unconventional branch of programming that can be challenging to learn, especially for programmers without an advanced computer science background. However, the parallel scripting library Parsl for Python was recognized as a promising tool for making parallel programming more accessible to our team. Parsl heavily utilizes Python’s decorator construct [2], an advanced programming concept which takes a polymorphic approach with nested

functions and other objects [3]. Thus, an advanced understanding of Python was required to begin planning for Parsl integration, and my first two weeks focused heavily on learning Python itself. Shortly thereafter, we began getting familiar with Parsl, creating several COMPASS-like test programs to simulate the development environment. The methodology behind this was to identify any major issues before serious work began, as we were conscious of the possibility that Parsl could present unexpected challenges for the project. Our subsequent testing revealed no major warning signs, so we began strategizing using Parsl's main constructs.

The first thing to consider was resource specification, which became a significant motivator of our decisions throughout the project. Parsl uses a construct called a "block" to refer to a single call for resources to a workload manager. The strategies for resource management are either static or dynamic. With a static strategy, a single block encompasses the entire execution, and the available resources – the number of requested nodes and the time limit – are specified at execution time. In a dynamic strategy, any number of blocks may be involved, and blocks are added and removed as needed during execution [2]. Initially, the dynamic strategy was appealing to us as it would automatically allow for the most efficient use of resources. In practice, however, our High-Performance Computing (HPC) systems would likely fare poorly when handling multiple calls to the workload manager for a single job. This is due to queuing for computation time, especially on busier machines. Each creation of a new block would be added to the back of the queue, adding significantly to the total runtime of the process. Considering the queue times on COMPASS-supported machines such as Cori (at the National Energy Research Scientific Computing Center) can be hours or even days, we decided that the static strategy would be the only logical course of action for COMPASS.

Following resource management, we began to consider execution. Executors are Parsl's execution managers, utilizing their specific strategies to conduct the overarching execution of the program. We examined three of Parsl's executors: ThreadPoolExecutor (TPEX), HighThroughputExecutor (HTEX), and WorkQueueExecutor (WQEX). TPEX initially looked like an ideal executor for local workflows, as it took advantage of the threading capabilities of even low-spec machines to speed up runtime. We realized quickly, however, that COMPASS and other MPAS software would need significant refactoring to become thread safe. Thus, we noted the possibility for threading as a future advancement to MPAS systems, but shifted our focus to HTEX and WQEX, which would support both local and HPC workflows. HTEX had the advantage of being the primary large-scale executor within Parsl, suggesting reliable performance. However, the resources for tasks managed by HTEX are determined during initialization of the workflow, meaning this approach is primarily envisioned for thousands to millions of similar tasks. In contrast, WQEX, currently in beta, introduced the ability to individually specify resources for each execution of a task [2]. This functionality was highly favored for our project, as COMPASS tasks range significantly in resource requirements. Still, WQEX's beta status has been the cause of some ongoing issues, specifically with running Parsl on Mac OS. This confirmed the necessity for a version of COMPASS without Parsl continuing to be available until these issues are resolved.

Our final consideration was workload management. Providers are Parsl's constructs for interfacing with workload management systems within running processes. We utilized two of Parsl's providers, the LocalProvider for local workflows and the SlurmProvider for HPC machines [2]. We are currently exploring this aspect of Parsl implementation, and considering

the extent to which refactoring will need to take place. Our primary focus is to preserve existing workflows while minimizing the complexity of new test case development.

This was the extent of our planning and development for the summer, though there is much additional work to be completed. Work will continue within the implementation phase to complete Parsl integration into COMPASS, with the secondary goal of familiarizing the team with Parsl for use with future projects. Many possible next steps for implementing task parallelism with Parsl exist, including additional systems within MPAS and other E3SM components. This work will increase efficiency in E3SM in significant ways, leading to more robust climate research capabilities and, ultimately, greater scientific advancements.

References

[1] E3SM – Energy Exascale Earth System Model. e3sm.org

[2] Parsl – Parallel Scripting Library. parsl.readthedocs.io/en/stable/index.html

[3] PEP 318 – Decorators for Functions and Methods. www.python.org/dev/peps/pep-0318/