HPC Workflow Management with Chapel

Benjamin Albrecht
benjamin.albrecht@hpe.com
Hewlett Packard Enterprise

Abstract

Coordinating many runs of monothilic high performance computing (HPC) applications is a challenge faced by much of the HPC user community. This includes domains such as datascience, bioinformatics, astronomy, and computational chemistry. For simple cases, users tend to rely on shell scripts that interact with the system workload manager to launch their applications. However, more advanced workflows can require complexity beyond what can reasonably be accomplished in a shell script. A more productive programming language is needed to tackle these more complex tasks. Cray HPO is a blackbox hyperparameter optimization framework written in Chapel. The framework employs many advanced workflow features, such as parallel launching, time budgets, and variable node counts. This talk will explore some HPC workflow design patterns encountered in the development Cray HPO and demonstrate why Chapel works well in this area.

To launch applications onto an HPC system, the workflow program must call out to the system workload manager. In Chapel, this can be done by utilizing Chapel's Spawn module to run the workload manager commands as spawn calls. In Cray HPO, an internal library was created to provide a clean interface to allocating, launching, and querying information through the workload manager. The Chapel forall loop can be used to drive the main loop that calls the workload manager launch commands. The domain being iterated over can represent the applications to launch, while the number of parallel tasks can represent the available units of nodes. If the application only runs on a single node, then the number of parallel tasks is equal to the number of nodes. By having each task block on the spawn call that initiates the launch, the forall loop acts as a local scheduler for the application launches. As soon as an application completes, the task becomes unblocked and ends, freeing up the next task to launch. Some other patterns that will be explored in this talk are syncronization between nodes, a time budget shared across all launches, and launching a distributed application with variable numbers of nodes.

Lastly, this talk will address the advantages and disadvantages of using Chapel in the workflow development space. A large advantage is having parallelism as a first class language feature. This makes parallelizing and syncronizing job launching straightforward. For some large scale workflows, performance and the ability to move to a distributed model can become valuable. In addition, Chapel's native C and Python interoperability are beneficial if there is a larger workflow framework that the Chapel application needs to hook into. The drawback is that Chapel's compiler speed will not match the edit-compile-run cycle of a language with an interpreter such as Python. Taken together, Chapel is a strong candidate for development of HPC workflow tools.