Performance Analysis of DroughtHPC and Holistic HPC Workflows

Extended Abstract

Yasodha Suriyakumar
Portland State University
P.O. Box 751
Portland, OR 97207-0751
yasodhan@pdx.edu

Karen L. Karavanic
Portland State University
P.O. Box 751
Portland, OR 97207-0751
karavan@pdx.edu

Hamid Moradkhani
University of Alabama
P.O. Box 751
Portland, OR 97207-0751
hmoradkhani@ua.edu

ABSTRACT

We present the results of a performance study of a newly developed drought prediction code called DroughtHPC. Holistic view helps to identify bottlenecks and identify areas for improvement in the software. The significant performance bottlenecks identified include: the overhead of calls to the VIC hydrologic model from within a python loop; VIC code structures that precluded parallelization with OpenMP; and significant file accesses. We observed challenges in diagnosing the performance of the code due to the use of an external modeling code in combination with python, a fairly common scenario in scientific computation. To address these challenges we designed PPerfG, a tool for visualizing Holistic HPC Workflows. We have implemented an initial prototype of PPerfG.

CCS CONCEPTS

• General and reference---Cross-computing tools and techniques---Performance

KEYWORDS

Parallel performance, data assimilation, performance visualization

1 INTRODUCTION

This work is part of an effort to develop a more effective drought prediction code. We report our initial efforts to analyze the performance of DroughtHPC [1], a drought prediction application developed at Portland State University. DroughtHPC improves prediction accuracy for a target geographical area, using data assimilation techniques that integrate data from hydrologic models [2,3], and satellite data. Input data includes: soil conditions, snow accumulation, vegetation layers, canopy cover and meteorological data. After development of an initial prototype in python, the goal was to scale the application to do finer-grained simulations, and also to simulate a larger geographical area.

The land surface of the target geographical area is modeled as a grid of uniform *cells*, we group sets of 25 cells as one *job*.

For a job that simulates 50 meteorological samples over a one month time period, the input data size is 144.5 MB, with the satellite data 132 MB; and the runtime is approximately two hours with the initial Python prototype.

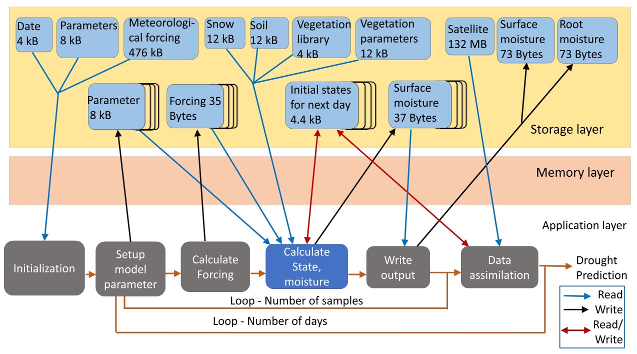

Figure 1: Holistic workflow diagram of DroughtHPC: Files accessed in simulation of a single cell, with multiple meteorological data (Number of samples) and time period (Number of days). Gray sections refer to Python code, and blue refers to the VIC hydrologic model. VIC accesses 57 files to compute each call.

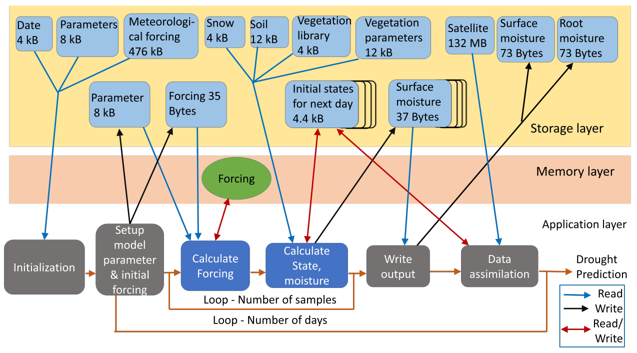

Figure 2: Holistic workflow diagram of DroughtHPC with updated VIC organization: the code is changed to minimize forcing data's movement between storage layers, and reduce invocation cost.

2 Performance Study Results

To evaluate the performance of the DroughtHPC prototype, we stepped through a series of measurements for single cell, single node, and multiple node performance. We used a variety of measurement strategies, spanning from higher level tools such as the python profiler [6] and gprof [4], to targeted tools such as dtrace; then we used the PerfTrack performance database [5] to collect all of the data. The significant performance bottlenecks identified include: the overhead of calls to the VIC hydrologic model from within a python loop; and significant file creation, reads, and writes. Each run of the VIC modeling code inputs and outputs 25 files; and VIC is called once per sample; the data assimilation code accesses 200 files for each simulated day. We are currently evaluating modifications to VIC to reduce the call overhead and the number of files.

The MPI version of the code is embarrassingly parallel with the work divided by job; we determined a best fit for our particular platform to be one job per core, where one job comprises 25 land cells. A significant improvement was achieved by configuring the use of local disks for storing the intermediate files; this reduced contention for the shared NFS file server. In addition, we have identified several minor modifications to VIC that will allow each MPI rank to be multithreaded using OpenMP.

3 Holistic HPC Workflows and PPerfG

During our study we observed the challenges of the performance analysis related to the structure of the code, that combined a python implementation of data assimilation with existing hydrologic modeling codes. The data moved between the two via writing and reading of files. Although this pattern is not uncommon in scientific applications, it is not easily handled by any one performance tool. In particular, the performance bottleneck related to the call pattern of VIC from python was inefficient but took a good deal of time to isolate.

To address this need we are developing an approach for performance diagnosis called Holistic HPC Workflows. The goal is to merge data from different layers of the runtime system (and therefore frequently different tools) for a single diagnosis. Our starting point is the development of PPerfG. PPerfG is a visualization tool for Holistic HPC Workflows. It captures the data movement behavior between storage layers, and between different stages of an application. Challenges include determining the best metrics, and efficient measurement techniques. We show a sketch of PPerfG in Figures 1 and 2. (Note: these are sketches and not screenshots.)

4 CONCLUSIONS

As a result of a detailed performance study, we identified several bottlenecks in our prototype approach. In our single-cell simulations, the bottleneck is the overhead of the VIC hydrologic model call from Python. In our parallel single-node version, we determined a best fit on our platform of one job per logical core; we explored changes to VIC for Intel Xeon Phi; and we are developing a version of VIC that eases integration with individual science codes such as data assimilation. As we scaled to multi-node simulations with MPI, the performance was dominated by the filesystem access pattern.

We designed PPerfG for visualizing Holistic HPC Workflows, in order to conduct analysis across layers of the runtime traditionally separated into disparate performance tools.

ACKNOWLEDGMENTS

Henry Cooney, Tu Le, Jiaqi Luo and Kristina Frye contributed ideas and discussions and implemented software used in this project. This material is based upon work supported by the National Science Foundation under Grant No. 1539605.

REFERENCES

[1] H. Yan and H. Moradkhani, "Combined assimilation of streamflow and satellite soil moisture with the particle filter and geostatistical modeling", Advances in Water Resources, vol. 94, pp. 364-378, 2016.

[2] University of Washington, "VIC hydrology model", http://www.hydro.washington.edu/Lettenmaier/Models/VIC/index-old.shtml, 2014. Accessed: 2018-02-01.

[3] U.S. Department of the Interior - U.S. Geological Survey, "Precipitation Runoff Modeling System", https://wwwbrr.cr.usgs.gov/projects/SW_MoWS/PRMS.html , 2016. Accessed: 2018-02-01.

[4] S. L. Graham, P. B. Kessler, and M. K. McKusick, "Gprof: A call graph execution profiler," SIGPLAN Not., vol. 39, pp. 49 - 57, Apr. 2004

[5] Karen L. Karavanic, John May, Kathryn Mohror, Brian Miller, Kevin Huck, Rashawn Knapp, and Brian Pugh, "Integrating Database Technology with Comparison-based Parallel Performance Diagnosis: The PerfTrack Performance Experiment Management Tool", In Proceedings of the 2005 ACM/IEEE conference on Supercomputing (SC '05).

[6] The Python Standard Library, chapter 27 Debugging and Profiling, available at https://docs.python.org/3/library/.