Establishing a Miniapp as a Programmability Proxy

Andrew I. Stone
Colorado State University
stonea@cs.colostate.edu

John M. Dennis
National Center for Atmospheric Research
dennis@ucar.edu

Michelle Mills Strout
Colorado State University
mstrout@cs.colostate.edu

Abstract

Miniapps serve as test beds for prototyping and evaluating new algorithms, data structures, and programming models before incorporating such changes into larger applications. For the miniapp to accurately predict how a prototyped change would affect a larger application it is necessary that the miniapp be shown to serve as a proxy for that larger application. Although many benchmarks claim to proxy the performance for a set of large applications, little work has explored what criteria must be met for a benchmark to serve as a proxy for examining programmability. In this poster we describe criteria that can be used to establish that a miniapp serves as a performance and programmability proxy.

Categories and Subject Descriptors D.2.8 [*Metrics*]: Software Science

General Terms Languages, Measurement, Performance

Keywords Programmability Proxy, miniapp, benchmark, POP, conjugate gradient, parallel programming

1. Introduction

Miniapps are applications on the order of 1000 lines of code, that include a simple build system, and accurately model the performance of a larger application. In [4], Heroux et al. suggest that miniapps can be used to compare programming models and improve compilers. We contribute to the miniapp concept by introducing criteria for evaluating how well miniapps serve as a programmability proxy of a larger application.

A true measure of how well a miniapp acts as a programmability proxy is whether application developers would use results from the miniapp to evaluate possible changes in the full application. We believe three requirements must be fulfilled for application developers to find miniapps useful; these are that a miniapp include important code, is representative of the application, and is designed to allow for incremental change.

We use these criteria to evaluate a miniapp of the Conjugate Gradient solver for a large and heavily-used climate simulation code called the Parallel Ocean Program (POP). This miniapp is called CGPOP [5].

2. Performance and Programmability Proxies

In [4] Heroux et al. introduce the concept of a miniapp. They state that a miniapp should on the order of 1000 SLOC and include a simple build process to enable easy porting. These size and simplicity requirements enable researchers to benchmark in immature environments. In this section we describe what criteria must be met for a miniapp to be considered a *performance proxy* and introduce additional requirements that must be met to show that a miniapp is a *programmability proxy*.

To be considered a *performance proxy*, a miniapp should accurately model the performance bottleneck of the full application at the range of cores that the full application typically targets. Also, the miniapp should be able to scale the problem size in a way that matches the performance of the full application when such scaling occurs. Once the baseline version of the miniapp has been shown to be performance proxy for the full app, modified versions of the miniapp can be compared to the baseline miniapp instead of the full application.

To be considered a *programmability proxy*, a miniapp should include the code from the full application that has the largest impact on performance (*code importance*), should enable incremental improvement to the full application (*incremental*), and should include a representative cross section of the implementation details (such as data distribution) that permeate the full application (*representative*). The programmability requirements for a miniapp are designed so that programmability conclusions made about the miniapp apply to the full application.

The *code importance* requirement often ensures that the code from the full application that is in the miniapp is code that is either often modified or dominates execution time. Often, the performance bottleneck in an application will require the most maintenance in the form of performance porting and debugging.

The *incremental* requirement aids in the realistic evolution of the full application. To be incremental, a miniapp should define a clean interface between the full application and itself to enable plugging in different versions of the miniapp while not negatively impacting performance due to any overhead introduced by the interface.

The *representative* requirement specifies that implementation details that affect the entire application should be included within the miniapp. This requirement suggests that simplifications to cross cutting data structures for details such as discretizations should not be done for a miniapp. The trade-off between this criterium and providing a small, modular miniapp for reintegration into the full application must be carefully managed.

3. The CGPOP Miniapp

The Parallel Ocean Program (POP) was developed at Los Alamos National Laboratory and is an important multi-agency code used for global ocean modeling and is a component within the Community Earth System Model (CESM) [1]. POP has been actively developed on for over 18 years, and is nearly 71,000 lines of Fortran 90 and MPI code. Due to its continued use, maintenance of the application in terms of porting it to new architectures is an ongoing issue. Several characteristics of POP and other earth system models present challenges to introducing changes in the application.

The POP application developers would like to experiment with new algorithms, data-structures, and programming models to improve performance and programmability given that new machines, with ever increasing numbers of cores, are being developed. However, experimentation is difficult to do within the context of the full application. The motivation for creating a miniapp for the POP developer team is that it will enable them to ensure that performance portability of the most critical portion of the application is maintained while also testing new algorithms, data-structures, and programming models.

The performance bottleneck in POP is a conjugate gradient (CG) computation. Conjugate gradient is a frequent target for benchmark and miniapp development. Examples include NPB CG [3] and HPCCG [4]. The main performance bottleneck in CG is the sparse matrix vector multiplies, which is a kernel that has also received significant attention in terms of performance [2, 6]. Although NAS CG has been referred to as a miniapp [4], based on the criteria we set forth it is not a miniapp relevant to POP because it is not a performance or programmability proxy for POP. Given that NAS CG does not fulfill the proxy requirements of a miniapp a new miniapp that models the Parallel Ocean Program has been developed. This miniapp is CGPOP. Whereas NAS does not fulfill the proxy requirements CGPOP does by design:

3.0.1 CGPOP Includes Important Code

The conjugate gradient solver within POP is the performance bottleneck when the application is executed on more than a thousand of cores. In a scientific application, the performance bottleneck code is often the most critical, and it consumes most of the developers' programming and maintenance time.

3.0.2 CGPOP is Incremental

We consider a miniapp to be incremental if a different version of the miniapp can be incorporated back into the full application. Incrementality would be threatened if reintegrating an implementation of the CGPOP miniapp back into the full POP application required conducting changes outside of the Conjugate Gradient solver routine that it is meant to model. We do not believe this to be a concern for CGPOP miniapp due to the fact that the difference between the base and studied CGPOP implementations predominantly lie in the construction of communication metadata and a subroutine (`UpdateHalo`) that performs the communication. Because communication metadata is solely used by the `UpdateHalo` routine and the fact that this routine is exclusively used by the conjugate-gradient algorithm, changes to this metadata or the `UpdateHalo` routine will not invasively impact any code outside of them when reintegrated into POP.

3.0.3 CGPOP is Representative

For the POP application, the most important implementation details are the discretization of the earth's surface including the exclusion of land, the domain decomposition for parallelization, the interaction between communication and computation, and the metadata generated to represent the communication schedule.

The discretization and the domain decomposition are used throughout the POP application and as such these important implementation details are part of the input for the CGPOP miniapp. Code within CGPOP uses the same domain decomposition as in the full POP application.

The interaction between communication and computation and the communication metadata are representative implementation details in POP that have been encapsulated within CGPOP. Thus programmability impacts for those implementation details are isolated within CGPOP. This makes it easier to reintegrate such changes back into the full application while making CGPOP a more representative programmability proxy.

4. Conclusions

Miniapps that serve as performance and programmability proxies can be used to evaluate the hypothetical impact a changes would have in a larger application. In this poster we discuss what criteria must be fulfilled for an application to be a performance and programmabiliy proxy. We then use these criteria to evaluate whether the CGPOP miniapp is a proxy for the Parallel Ocean Program (POP).

Acknowledgements

This work was supported by Department of Energy Early Career Award #DE-SC3956. This work was financially supported through National Science Foundation Cooperative Grant NSF01 which funds the National Center for Atmospheric Research (NCAR), and through the grant: #OCI-0749206.

References

[1] Community Earth System Model. `http://www.cesm.ucar.edu/`.

[2] K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer, D. A. Patterson, W. L. Plishker, J. Shalf, S. W. Williams, and K. A. Yelick. The landscape of parallel computing research: A view from Berkeley. Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley, Dec 2006.

[3] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, H. D. Simon, V. Venkatakrishnan, and S. K. Weeratunga. The NAS parallel benchmarks. Technical report, The International Journal of Supercomputer Applications, 1991.

[4] M. A. Heroux, D. W. Doerfler, P. S. Crozier, J. M. Willenbring, H. C. Edwards, A. Williams, M. Rajan, E. R. Keiter, H. K. Thornquist, and R. W. Numrich. Improving performance via mini-applications. Technical Report SAND2009-5574, Sandia National Laboratories, 2009.

[5] A. I. Stone, J. M. Dennis, and M. M. Strout. Evaluating coarray fortran with the CGPOP miniapp. In *Proceedings of the Fifth Conference on Partitioned Global Address Space Programming Models (PGAS)*, October 15, 2011.

[6] S. Williams, L. Oliker, R. Vuduc, J. Shalf, K. Yelick, and J. Demmel. Optimization of sparse matrix-vector multiplication on emerging multicore platforms. In *Proceedings of the 2007 ACM/IEEE conference on Supercomputing*, SC '07, pages 38:1–38:12, New York, NY, USA, 2007. ACM.