Alternative open-source tools for high-performance software: Integrating parallel global address space languages into traditional applications

Sophia Lefantzi[1] and Benjamin A. Allan[2]

[1] lefantzi@comcast.net,
http://www.cca-forum.org/~lefantzi
[2] Sandia National Laboratories, Livermore, CA 94550, USA,
baallan@ca.sandia.gov,
http://www.cca-forum.org/~baallan

Abstract. Parallel global address space (PGAS) languages are a hot topic in scientific computing research, as the computer science community seeks combined hardware and software architectures that yield near-peak performance. However, computational science is evolutionary rather than revolutionary in its software development. Adoption of PGAS tools will require interoperation with conventional language tools. We survey the integration of open-source PGAS languages, including Titanium and UPC, with existing applications and language interoperability tools. Their impact on source code, software build processes, and run-time issues is presented for parallel application test cases.

1 Introduction

After years of struggle, and with no end to its difficulty in view, MPI has become the standard for achieving parallel programming portability between the workstation clusters accessible to most researchers and the time-shared high-end machines based on non-commodity processors and networks. Better ways to specify programs on parallel machines, beyond the reuse of specialized high-level application libraries, remain a significant research topic, as evidenced by the focus on languages in programs such as the United States multi-agency HPCS[1] effort.

As better commodity hardware and open-source networking software are expanding to support one-sided communications (remote memory reads and writes), the simpler parallel global address space (PGAS) programming model [2] has become a viable alternative for portable programming. Current compiler development efforts by the open-source and high-end-computing vendor communities have shown performance comparable or superior to MPI on microbenchmarks as well as full scientific application frameworks [3]. Co-array FORTRAN [4, 5] is not readily available to the open-source community yet.

Scientific computing is increasingly the province of multi-institutional software teams instead of hero-programmers [6, 7]. For the most challenging multi-scale, multi-domain applications MPI allows each submodel to have its own communicator object and permits the submodel programmer to code in a logically isolated fashion. When the submodels are assembled into full simulations with any given submodel running only on a subset of the total processors available to the overall model, collective MPI operations are possible in the scope of the submodel. A similar facility is not generally available in the current PGAS languages, though Cray Fortran provides a *team* concept which may allow for submodel computations.

2 Integration

Hybrid codes will be the norm, rather than the exception, as computational science simply cannot afford to throw out all the codes and start over when building multi-domain, multi-discipinary simulations. Critical to forming these integrated codes is the ability to combine libraries written in new languages with existing MPI-based libraries and with existing libraries in conventional C, C++, and Fortran.

2.1 Integration with MPI applications

Next we examine a number of open-source PGAS languages and integration of objects written in those languages into MPI-based programs.

MuPC The Michigan MPI+pthreads implementation of UPC is intended to be portable. It does not claim to support mixing MPI and UPC calls in the user's code. Its basic architecture is an application thread in which the user's UPC code runs and a service thread given entirely to managing data-exchanges through compiler-generated MPI calls.

Berkeley UPC The Berkeley UPC compiler permits cooperation with MPI code in a barrier-sychronized way. The design is that at any given time, the entire parallel application must be in *UPC mode* or *MPI mode*. This approach is feasible for many applications, but requires any higher-level driver controlling multiple libraries to know which libraries use which communications underneath and coordinate changing modes.

Titanium We do not yet have the results for integrating Titanium and MPI at the application level. We expect the results to be dependent on how the hidden back-end (gasnet) layer is selected and implemented.

2.2 Integration with other languages

None of the PGAS language definitions presently includes explicit interoperability between the PGAS language compiler and conventional Java, C, C++, or FORTRAN compilers. Generally, some provision is made in the PGAS implementations for interoperation with C, but this feature is not presented as a stable interface for production coding efforts. The Cray Co-Array Fortran compiler produces objects which can be integrated with Cray UPC, C++, and other compilers in the X1 hardware environment, which does not appear to include a Java to binary compiler.

UPC Berkeley UPC and MuPC both generate conventional C code with added communication directives from UPC sources. Thus it is possible to use libraries in other conventional languages callable from C for purely local (thread-local) computations. C++, Fortran, and GNU Java are thus integrable with libraries from these UPC implementations. Berkeley UPC can also be initialized and used from a main() function written in other languages, provided necessary command-line options are accessible.

Titanium As with Berkeley UPC, Titanium generates code for a conventional compiler with added gasnet directives to manage communication. At present, C and other C callable languages are only accessible through Titanium's equivalent of JNI (Java native interface). Support for embedding the Titanium run-time in a larger application is not documented.

3 Experiments on a hybrid application

The simulation model We present some results on a small problem run on a small cluster. The problem is to solve a large set of coupled ordinary differential equations combining a conventional solver with models expressed in the PGAS languages.

The source code results Integrating with Titanium requires source management gymnastics, particularly if one decides to adjust a method signature. This is typical of Java integration in general, however. Integrating UPC and MPI codes may require judicious use of copying.

The build results Building and linking are tricky, but no more so than with other compilers under development. The bulk of the cost is in the up-front effort to install and verify the experimental compilers on our Infiniband-based cluster.

The run-time results None of the compilers tested claim to be performant; rather they are reference implementations of new languages. Nevertheless, the results are encouraging.

4 Conclusions and future work

PGAS languages are presently too early in development to use as the base for a multi-domain production code with strong portability requirements. However, we are encouraged that the computer science community is moving in the correct direction. A key feature that remains to be addressed by the PGAS language development community is the ability of the module developer to code as if their module sees the whole machine when in fact it is used on only a subset of the threads available; the MPI equivalent is the developer who writes code using a given communicator object. Without this ability to use subsets of the thousands of threads available on the worlds largest machines, multi-domain simulations and optimization over bag-of-tasks style submodels will remain outside the scope of PGAS-based software.

References

1. HPC Productivity: an Overarching View, *IJHPCA* **18** (2004).
2. High Performance Programming in the Partitioned Global Address Space Model, `http://www.cs.berkeley.edu/~yelick/talks/upc-caf-ti-tutorial-siampp04.pdf`, 2004.
3. Katherine Yelick, Partitioned Global Address Space Languages, in *SIAM conference on parallel processing for scientific computing*, 2006.
4. Fortran language reference manual, `http://docs.cray.com/cgi-bin/craydoc.cgi?mode=View;id=S-3694-54;idx=books_search;this_sort=;q=;type=books;title=Fortran%20Language%20Reference%20Manual%2c%20Volume%203`, 2006.
5. Cristian Coarfa, Yuri Dotsenko, John Mellor-Crummey, Francois Cantonnet, Tarek El-Ghazawi, Ashrujit Mohanty, YiYi Yao, and Daniel Chavarria-Miranda, An Evaluation of Global Address Space Languages: Co-Array Fortran and Unified Parallel C, in *PPoPP'05 Principles and Practice of Parallel Programming*, pages 36–47, ACM Press, 2005.
6. N. Collins, G. Theurich, C. DeLuca, M. Suarez, A. Trayanov, V. Balaji, P. Li, W. Yang, C. Hill, and A. da Silva, Design and Implementation of Components in the Earth System Modeling Framework, *IJHPCA* (2005).
7. Lois Curfman McInnes, Benjamin A. Allan, Robert Armstrong, Steven J. Benson, David E. Bernholdt, Tamara L. Dahlgren, Lori Freitag Diachin, Manojkumar Krishnan, James A. Kohl, J. Walter Larson, Sophia Lefantzi, Jarek Nieplocha, Boyana Norris, Steven G. Parker, Jaideep Ray, and Shujia Zhou, Parallel PDE-Based Simulations Using the Common Component Architecture, in *Numerical Solution of Partial Differential Equations on Parallel Computers*, Lecture Notes in Computational Science and Engineering, Vol. 51, 2006, editors: Are Magnus Bruaset and Aslak Tveito, Springer-Verlag, chapter 10; also available as Argone National Laboratory technical report ANL/MCS-P1179-0704.