The scalability impact of a component-based software engineering framework on a growing SAMR toolkit: a case study

Benjamin A. Allan * and Jaideep Ray

We examine a growing toolkit for parallel SAMR-based simulations and the impact of component-based software engineering on the scalability of the development process as well as the scalability of the run-time performance. The CBSE approach enables a wide range of contributions to the toolkit in a short time. We also examine the question of publications: Does writing good software hurt productivity?

1. Introduction

In this case we set out to build a reacting flow modeling toolkit [4] based on structured adaptive mesh refinement (SAMR) following the Common Component Architecture (CCA) [1]. This is a work in progress. Thus we are performing a test of the CCA concepts as well as generating a simulation code. Most of the costs and issues involved in building scientific research codes are rarely discussed in the science literature that presents the results of the simulations. As the complexities of studied phenomena are exploding, however, we must start examining the underlying problems in building codes complex enough to simulate these phenomena.

1.1. The research software problems

Creating a CFD toolkit as part of a research project presents many practical issues:

- The limited time available requires the use of externally controlled legacy libraries, of mixed language programming (since rewriting all the legacy code in a more favored language is not an option), and of a team of developers of mixed abilities many of whom may be only transiently involved in the project.
- The lead developer (usually also the software architect and a hero programmer) must be able to rigorously enforce subsystem boundaries and to control boundary changes.
- The team is usually required to continue publishing while coding, if they wish to obtain funding for their next projects.
- In a high-performance computing (HPC) code, coping with shifting platforms (either hardware or software) is required.
- For better scientific validation, swapping out developed package elements for alternative code (possibly developed by competing research groups) is often required.
- Over the course of any project, requirements change and various package elements will have to be recoded. Less frequently, but usually more expensively, an interface between elements may need to be extended or redefined.

*Sandia National Laboratories, MS9158, Livermore, CA 94550-3425
baallan,jairay@ca.sandia.gov

1.2. The CCA approach to HPC CBSE

Our development process and software products follow the Common Component Architecture (CCA) [5] methods. The CCA defines, among other things, a set of generic rules for composing a single-program-multiple data (SPMD) parallel application from many software components. Each component is a black-box object that provides public interfaces (called *ports*) and uses ports provided by other components. In object-oriented terms, a port is merely the collection of methods into an interface class. The CCA leaves the definition of SAMR-specific ports to us.

A component is never directly linked against any other. Rather, at run-time each component instance informs the framework of which ports it provides and which it wishes to use. At the user's direction, the framework exchanges the ports among components. A component sees other components only through the ports it uses. Implementation details are always fully hidden. All needed components are combined into a single SPMD code.

2. Software development scalability

In this section we present summary information about the software produced to date. The task flow of the project, broken down by products and developers, shows that the CCA approach enables diverse contributors to produce all the required products as a team. The raw data, in the form of a Gantt chart, is too large to present here, but appears in [3]. Over the four year course of the project eight developers have participated, with the average head-count at any moment being four.

2.1. How the CCA model manifests as a SAMR framework

The component design of the CFRFS project [6] allows encapsulation and reuse of large legacy libraries written in C and FORTRAN. Of the 61 components produced in the project so far, 13 require some aspect of parallel computing, while the rest handle tasks local to a single compute node. As with most research projects, the lead developer is critical: in the work-flow chart we see that 35 of the components were created by developer A, 13 by developer B, 8 by developer C, and the remaining components by others.

Port interfaces get created to handle new functionality; the present count is 31. Four of the eight developers contributed port definitions. In defining ports, the lead developer often plays a consulting role, but the details of a port encapsulating new functionality are usually dictated by an expert on the underlying legacy libraries rather than by the lead developer.

As we see in figure 2, the typical component source code is less than 1000 lines of C++ (excluding any wrapped legacy code); the exceptions being the GrACE [7] library wrapping component at almost 1500 lines and the adapter for Chombo [2] inter-operation at almost 2700 lines. The average component is therefore reasonably sized to be maintainable. The best measure of the legacy codes encapsulated by the components may be obtained by examining the size of the resulting component libraries in figure 1. The Chombo, HDF, and thermochemical properties libraries are the three giants at the end of the histogram; most components compile to under 100 kilobytes. An often proposed metric of quality for an object-oriented code is the number of lines per function. Figure 3 shows that the distribution of lines per *port* function varies widely across components, being particularly high in those components which are complicated application drivers or wrappers which hide complexity in an underlying legacy library.

A question often asked by people considering adopting the component approach is "how

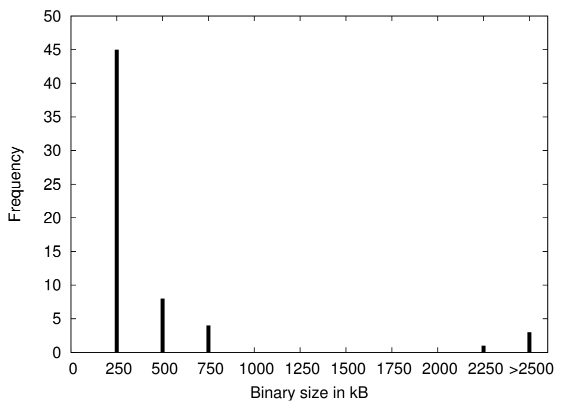

Figure 1. Histogram of component binary library sizes.

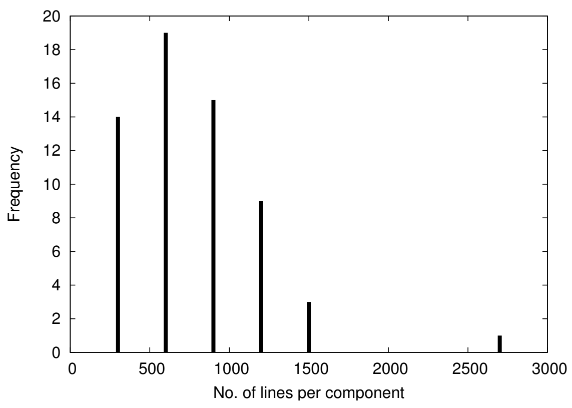

Figure 2. Histogram of component source code sizes.

complex should my ports be?", by which they mean how many functions are in one port. The correct answer is "it depends on what the port does", but in figure 4 we see that most ports are small with fewer than 10 functions. The largest port, with 39 functions is the AMRPort which encapsulates the bulk of the GrACE functionality.

CCA component technology is often described as providing plug-and-play functionality such that alternative implementations can be plugged in for testing and optimization. We test this claim in the CFRFS context by examining how many different components implement each kind of port, in figure 5. In this combustion work, the most often implemented port turns out to be the diffusion functionality, with 10 components providing alternate models.

In CFD modeling, there is often a central mesh concept that is reused in nearly all code modules. This is apparent in our case from figure 6, where the most frequently used port is the AMRPort with 29 clients. The next most frequently used port is the properties port which allows a driver component to tune any component that exports the properties port.

2.2. Productivity

We have seen the impact of the CCA on the CFRFS software development, but what of the project intellectual productivity? One of the principal goals of the project is to produce a reusable software toolkit, of course, but what of other metrics of career success? Software developers in the course of the work have produced: six scientifically interesting in-house test applications, 11 conference papers, a book chapter, and 3 journal papers all related directly to this work. A SAMR framework interoperability standard has also been drafted and tested with the Chombo framework. Rather than distracting the development team from the intellectual issues the software is designed to address, the component structure of the SAMR framework enables faster testing of new ideas because any needed changes are usually isolated to one or two components.

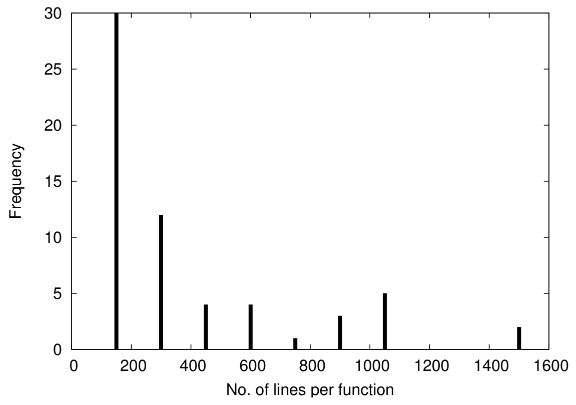

Figure 3. Histogram of function size.

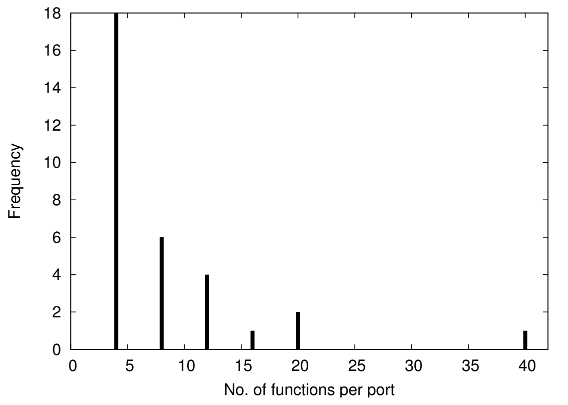

Figure 4. Histogram of port size in number of functions.

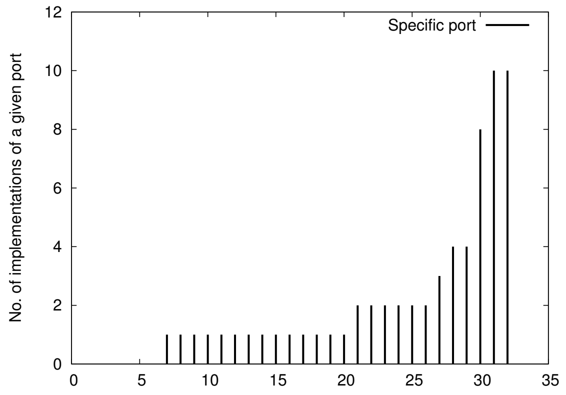

Figure 5. Number of components implementing each port type, sorted by increasing frequency.

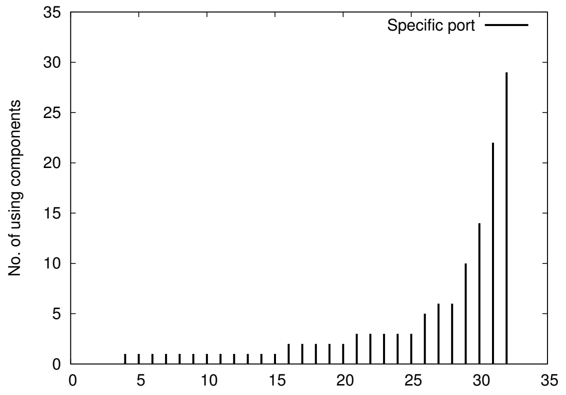

Figure 6. Number of components using each port type, sorted by increasing frequency.

3. Conclusions

In this abstract we have characterized the software structure and processes obtained from the application of the CCA high-performance component methodology to the structured AMR modeling of reacting flows. We have seen that the component methodology enables efficient development of both software and intellectual contributions.

4. Acknowledgments

This work was supported by the United States Department of Energy, in part by the Office of Advanced Scientific Computing Research: Mathematical, Information and Computational Sciences through the Center for Component Technology for Terascale Simulation Software and in part by the Office of Basic Energy Sciences: SciDAC Computational Chemistry Program. This is an account of work performed under the auspices of Sandia Corporation and its Contract No. DE-AC04-94AL85000 with the United States Department of Energy.

REFERENCES

1. CCA Forum homepage. `http://www.cca-forum.org/`, 2004.
2. P. Colella et al. Chombo – Infrastructure for Adaptive Mesh Refinement. `http://seesar.lbl.gov/anag/chombo`.
3. PCFD 2005 Gantt chart of CFRFS software development processes. `http://cca-forum.org/~baallan/pcfd05-snl/workflow`, 2005.
4. Sophia Lefantzi and Jaideep Ray. A component-based scientific toolkit for reacting flows. In *Proceedings of the Second MIT Conference on Computational Fluid and Solid Mechanics*, volume 2, pages 1401–1405, Boston, Mass., 2003. Elsevier Science.
5. 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 Are Magnus Bruaset, Petter Bjørstad, and Aslak Tveito, editors, *Numerical Solution of PDEs on Parallel Computers*. Springer-Verlag, 2005. invited chapter, submitted.
6. Habib N. Najm et al. CFRFS homepage. `http://cfrfs.ca.sandia.gov/`, 2003.
7. M. Parashar et al. GrACE homepage. `http://www.caip.rutgers.edu/~parashar/TASSL/Projects/GrACE/`, 2004.