THE COMMON COMPONENT ARCHITECTURE (CCA) APPLIED TO SEQUENTIAL AND PARALLEL COMPUTATIONAL ELECTROMAGNETIC APPLICATIONS

DANIEL S. KATZ, E. ROBERT TISDALE, CHARLES D. NORTON

Jet Propulsion Laboratory, California Institute of Technology, 4800 Oak Grove Drive, Pasadena, CA, 91109, USA

E-mail: {Daniel.S.Katz, E.Robert.Tisdale, Charles.D.Norton}@jpl.nasa.gov

The development of large-scale multi-disciplinary scientific applications for high-performance computers today involves managing the interaction between portions of the application developed by different groups. The CCA (Common Component Architecture) Forum is developing a component architecture specification to address high-performance scientific computing, emphasizing scalable (possibly-distributed) parallel computations. This paper presents an examination of the CCA software in sequential and parallel electromagnetics applications using unstructured adaptive mesh refinement (AMR). The CCA learning curve and the process for modifying Fortran 90 code (a driver routine and an AMR library) into two components are described. The performance of the original applications and the componentized versions are measured and shown to be comparable.

1 Introduction

The work described in this paper was undertaken to answer the following questions regarding the Common Component Architecture (CCA):

- How usable is the CCA software? What work is involved for a scientist to take previously written software and turn it into components, particularly for parallel components?
- Once the components exist and are linked together, how does performance of the componentized version of the application compare with that of the original application, again, particularly for parallel components?

The paper does not deal with the question of why one might choose to use components. It assumes that the reader has an interest in using components, and wants to understand the implications of choosing to use the CCA software for this purpose.

The remainder of this paper will describe the initial software, describe the componentization process, and provide and analyze the timing measurements, and finally summarize the answers to the questions.

2 The Common Component Architecture (CCA)

The CCA Forum [1] was founded in January 1998, as a group of researchers from the U.S. National DoE labs and academic institutions committed to defining a standard Component Architecture for High Performance Computing. The CCA Forum noticed that the idea of using component frameworks to deal with the complexity of developing interdisciplinary HPC applications was becoming increasingly popular. Such

systems enable programmers to accelerate project development through introducing higher-level abstractions and allowing code reusability, as well as provide clearly specified component interfaces which facilitate the task of team interaction. These potential benefits encouraged research groups within a number of laboratories and universities to develop, and experiment with prototype systems. However, these prototypes do not interoperate.

The need for component programming has been recognized by the business world and resulted in the development of systems such as CORBA, DCOM, Active X and others. However, these systems were designed primarily for sequential applications and do not address the needs of HPC.

The objective of the CCA Forum is to create a standard that both a framework and components must implement. The intent is to define a minimum set of conditions needed to allow high performance components built by different teams at different institutions to be used together, and to allow these components to interoperate with one of a set of frameworks, where the frameworks may be built by teams different from those building the components. The CCA forum members are developing implementations of the standard as well, both components and frameworks.

3 The Non-Componentized Software

The original JPL software consisted of two units. The first was the 2-dimensional, parallel version of the Pyramid unstructured Adaptive Mesh Refinement (AMR) library [4], developed at JPL over the last few years. Pyramid uses the MPI library for its interprocessor communication. The second was a driver routine for this library [2]. The driver is also parallel, but it does not have any communications routines, since they are all handled within Pyramid. All of the original software was written in Fortran 90, though Pyramid requires an additional library called ParMetis, that determines a repartitioning for the parallel version of the Pyramid library. ParMetis was only used as a binary library, and was not modified in any way in this work. The function of the software is to read in a mesh resulting from an electromagnetic problem, and to (possibly repeatedly) refine a region of this mesh.

4 Componentization of the Software

The initial work on this task [3] included development of simple single component and two component example applications. After these were developed, the only problem that had to be overcome to componentize the sequential software was building a C++ wrapper for the Fortran Pyramid library, and translating the driver code into C++, as the CCA framework (Ccaffeine) required components to be written in C++.

The CCA model for parallel applications is a Single Component, Multiple Data (SCMD) model. In this model, one process of each component exists on all processors. In a given processor, one components

communicates with another component through the framework. Intercomponent communication takes place as expected, using a library such as MPI. A component in one processor cannot communicate directly with a different component in a different processor.

As mentioned above, Pyramid uses the MPI library to communicate, and the driver component does not do any communication. Thus the only real differences between the sequential and parallel versions of the application are in launching the framework in parallel and ensuring that the components are also started in parallel. For the current framework, it can be launched on multiple processors by simply starting it with `mpirun -np $number $path_to_ccaffeine`. Since the driver code and the Pyramid library were both written in such a way that they can run on one or more processors, no changes needed to be made to the driver or Pyramid components.

5 Timing Results

For each run of the application, two times were measured, the maximum time from the before the first call to the library to after the last call to the library over the set of processors in a given run, and the wall clock time. These two times were not significantly different for any run. Figure 1 shows the results from the parallel experiments. (Sequential results are not shown in the interest of space.) Each result is the average of 5 to 10 runs.

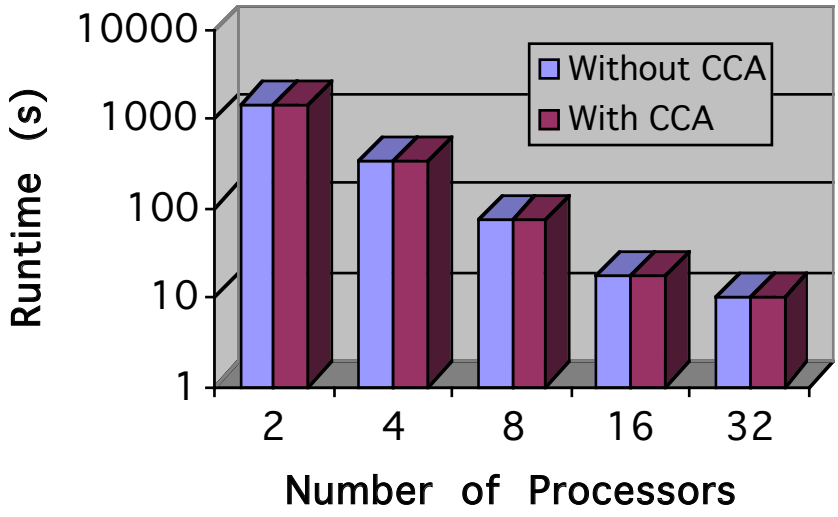

Figure 1.Timing results for the parallel component vs. driver/library application.

These results show an insignificant difference between the speed of the component application and the driver/library application on 2 to 32 processors. In some cases, the component application is slightly faster, in others, the driver/library application is slightly faster. The key point is that

the scalability is unchanged between the versions; the CCA framework has no effect on how the parallel application scales.

6 Conclusions

The lessons learned in this work are:

- There was initially a fair amount of learning associated with use of the CCA Forum's technology, including the CCAFEINE framework. It took 2-3 months to componentize the first application, though the second was componentized fairly quickly. Once the sequential application was componentized, proceeding to the parallel application was simple.
- The lack of a means to write Fortran90 components is a serious shortcoming for many science applications. It is possible to get around this shortcoming, but this introduces additional work for the componentizer and adds the chance for additional errors to come into the application.
- Once an application is componentized, if the amount of work done in each component call is large when compared with the time needed to make a function call, it is likely that the componentized version of the application will perform well.

The authors' knowledge of ongoing work within the CCA Forum leads them to believe that the first issue has been mostly resolved, and the second issue will be resolved in time, most likely in less than 9 months. Once this is done, the CCA model will be a promising method for building large single-processor and parallel applications.

In the next year, an effort will be undertaken to continue to resolve the first two issues above (flattening of the CCA learning curve and ensuring the Fortran90 components can be used in CCA.) Additionally, plans exist to turn a climate application into a CCA application.

References

1. Armstrong R., Gannon D., Geist A., Keahey K., Kohn S., McInnes L. C., Parker S., Smolinski B., Toward a Common Component Architecture for High-Performance Scientific Computing, Proceedings of High Performance Distributed Computing, (1999) pp. 115–124.
2. Cwik T., Coccioli R., Wilkins G., Lou J. and Norton C., Multi-Scale Meshes for Finite Element and Finite Volume Methods:Active Device and Guided-Wave Modeling, *Proc. of AP2000 Millennium Mtg*. (2000).
3. Katz D. S., Tisdale E. R., and Norton C. D., A Study of the Common Component Architecture (CCA) Forum Software, *Proceedings of High Performance Embedded Computing (HPEC-2002)*, (2002).
4. Norton C. D., Lou J. Z., and Cwik T., Status and Directions for the PYRAMID Parallel Unstructured AMR Library, *8th Intl. Workshop on Solving Irregularly Structured Problems in Parallel (15th IPDPS)*, (2001).