*Astronomical Data Analysis Software and Systems XII*
*ASP Conference Series, Vol. 295, 2003*
*H. E. Payne, R. I. Jedrzejewski, and R. N. Hook, eds.*

# Efficient Distribution of Computational Load on a Beowulf-Like Cluster

Luca Fini, Marcel Carbillet

*INAF—Osservatorio Astrofisico di Arcetri, Firenze, I-50125, Italy*

**Abstract.** The CAOS Application Builder is a Graphical Programming Environment which allows the building of complex simulation applications by putting together elementary blocks. The resulting simulation programs are often very heavy in computational needs and could be profitably run on Beowulf-like clusters, provided the computational load can be efficiently distributed on the CPUs. In the paper we describe a project to provide the CAOS Application Builder with software tools which allow the user to optimize the distribution of blocks on a multi-CPU machine and show a few preliminary results.

## 1. Introduction

CAOS and AIRY are two sets of tools designed to allow the building of complex simulation programs specifically targeted to adaptive optics (CAOS) and interferometric image restoration (AIRY).

Each package consists essentially of a number of modules which can be assembled together in a simulation program (Carbillet 2001, Correia 2002).

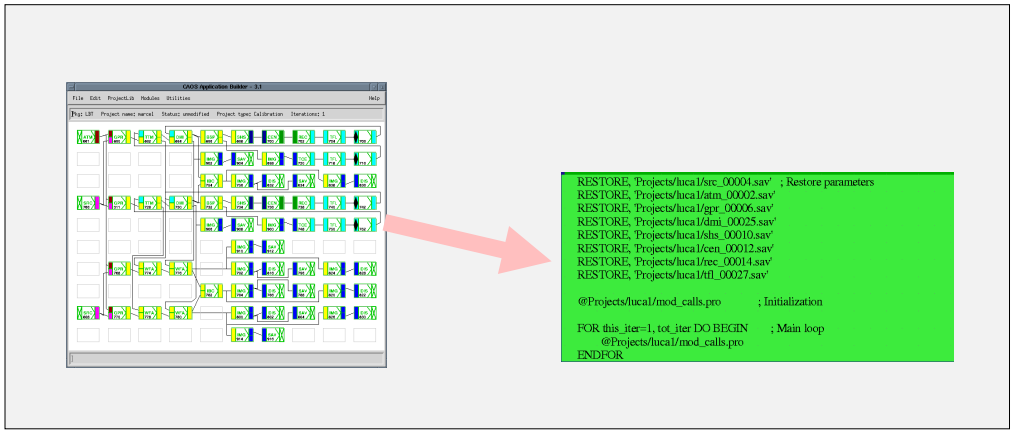


Figure 1. The CAOS Application Builder generates program code

A module is actually a single routine and the simulation program is usually built by assembling a sequence of calls to proper modules and wrapping everything within an iteration loop. In order to simplify the design of application

programs by hiding the actual code of the simulation algorithms, the packages are provided with the Application Builder[1] (Fini 2001), a Graphic Programming Environment where application programs can be assembled in a graphical manner and the actual simulation code is generated automatically (see Figure 1).

The CAOS/AIRY systems and the Application Builder have been developed under IDL and are currently targeted to IDL programs, although the same techniques could be applied to other programming environments as well.

Simulation programs are typically very CPU intensive, thus the ability to run production simulations on a high performance parallel machines is quite appealing.

Unfortunately most current programs (and notably both CAOS and AIRY) have been designed for scalar machines and should be rewritten, at least partially, to exploit the capabilities of parallel architectures and, moreover, programming for parallelism is not usually an easy task.

## 2. Common Parallelization Strategies

Parallelizing programs is in principle a simple job: the program is subdivided into partially independent tasks which can be executed concurrently on various CPUs. The actual improvement in performance, however, depends critically on how the program is divided into tasks and on the architecture of the parallel machine and its actual performance. To divide a program into tasks one can follow essentially three strategies:

- **Parallelize core algorithms**. This is done by identifying the "core" of the program, i.e., the portions of code where it spends most of the running time and implementing a parallel version of that part of the program (D'Amore 2003). This approach is well suited whenever the program actually contains a few and well identified cores, which is not always the case.
- **Subdivide input domain**. Simulation programs are usually structured as iterations on the input domain space. It is sometimes possible to identify one of the space dimensions along which iterations are independent from each other and then subdivide the input along that dimension so that each CPU of a parallel machine operates on a subset of the input domain.
- **Parallelize the program as a whole**, i.e., analyze the program's structure and redesign it carefully to identify concurrent tasks in some optimal way. This approach is a very general one, and is substantially independent of the architecture of the program, but is also the most complex.

All the three strategies, alone or even used together, can yield improvements in execution time, but in any case the need for data exchange among tasks must be carefully considered. On most parallel machines, and notably on Beowulf clusters, task-to-task data communication is performed through a comparatively

[1]Due to historical reasons the name "CAOS Application Builder" is often used, but the tool can be used for assembling AIRY related programs too.

slow channel so that the related overhead can easily spoil the time improvement gained with parallelization.

## 3. Parallelization Made Easy

The third parallelization strategy quoted above is the most general, but also the most challenging in that the complexity of the program structure may notably be increased when the code is redesigned for parallelism. Moreover parallel programming skills are pretty unusual in the common curricula of physicists or engineers who are the main users of the CAOS system.

The aim of this work is thus to exploit the automatic code generation capabilities of the Application Builder to implement a code generator which can automatically produce source code optimized for a target Beowulf architecture.

What is most important is that all the essential tools to analyze the structure of the program and to generate the equivalent code are already there because they are also needed for the scalar version of the Application Builder.

## 4. How Do We Proceed?

In order to allow the Application Builder to generate optimized parallel code, it must be augmented with three tools:

- Code Profiler. It is essentially a modification of the Application Builder code generator which includes calls to profiling routines in the program's code. During a sample run of the program, data on execution time and on the amount of data exchanged between modules are gathered and stored in a profile table.
- Enhanced code generator. Based on profile table data, linear programming techniques are used to subdivide modules into tasks in an optimal manner, so that the code is suitable to be executed on a grid of computers. The code generator also includes calls to proper data exchange routines based on the the standard MPI library for task-to-task communication.
- Cluster Evaluation Tool. A simple tool which characterizes the performances of a given Beowulf cluster in order to guide the optimization of the task allocation process.

A simulation program life cycle can thus be as follows: a) build the graphic representation of the simulation program by using the Application Builder just as before (or open an existing simulation program); b) select the profiling option and run a sample version of the program on a sensible but small subset of the input data using any scalar machine; c) evaluate the cluster characteristics by using the suitable tool (or get the same info from a previously stored table); d) let the Application Builder generate code for your target grid of computers. e) run the parallel version of the program to your satisfaction.

## 5. Conclusion

By exploiting the code generation capabilities of an existing Graphic Programming Environment it is possible to implement a system which is capable of generating code optimized for running on parallel system of the Beowulf class.

Most of the code needed is already part of the existing CAOS Application Builder and we have presented the architecture of the tools to be added to it for the purpose of parallel code generation. The coding is currently on the way and will yield a beta version in the near future.

More information about the CAOS and the AIRY packages can be found at:

`http://www.arcetri.astro.it/caos`

## References

Carbillet, M., Fini, L., Femenía, B., Riccardi, A., Esposito, S., Viard, E., Delplanke, F. & Hubin, N. 2001, in ASP Conf. Ser., Vol. 238, Astronomical Data Analysis Software and Systems X, ed. F. R. Harnden, Jr., F. A. Primini, & H. E. Payne (San Francisco: ASP), 349

Correia, S., Carbillet, M., Boccacci, P., Bertero, M., Fini, L. 2002, A&A, 387, 733.

D'Amore, L., Guarracino, M.R., Laccetti, G. 2003, "On the Parallelization of a Commercial PSE for Scientific Computation", to appear in "Proceedings of IEEE International Conference on Parallel and Distributed Processing", Genova, Italy.

Fini, L., Carbillet, M., & Riccardi, A. 2001, in ASP Conf. Ser., Vol. 238, Astronomical Data Analysis Software and Systems X, ed. F. R. Harnden, Jr., F. A. Primini, & H. E. Payne (San Francisco: ASP), 253