Automated Generation of Message-Passing Programs — An Evaluation Using CAPTools

Michelle R. Hribar, Haoqiang Jin and Jerry C. Yan *
M/S T27A-2, NASA Ames Research Center, Moffett Field, CA 94035-1000
Tel: (650) 604-2872 E-mail:{hribar, hjin, yan}@nas.nasa.gov

Abstract: Scientists at NASA Ames Research Center have been developing computational aeroscience applications on highly parallel architectures over the past ten years. During that same time period, a steady transition of hardware and system software also occurred, forcing us to expend great efforts into migrating and re-coding our applications. As applications and machine architectures become increasingly complex, the cost and time required for this process will become prohibitive. In this paper, we present the first set of results in our evaluation of interactive parallelization tools. In particular, we evaluate CAPTool's ability to parallelize computational aeroscience applications. CAPTools was tested on serial versions of the NAS Parallel Benchmarks and ARC3D, a computational fluid dynamics application, on two platforms: the SGI Origin 2000 and the Cray T3E. This evaluation includes performance, amount of user interaction required, limitations and portability. Based on these results, a discussion on the feasibility of computer aided parallelization of aerospace applications is presented along with suggestions for future work.

1 Introduction

1.1 Motivation

High performance computers have evolved rapidly over the past decade. Although new advances in architecture have increased overall performance, they have also created limitations on programs' portability. Hand tuning applications for each new machine achieves the best performance results, but at a high cost of time and effort. At NASA Ames Research Center, high performance computing hardware is constantly updated to keep pace with new technology. This translates to an average machine life span of 3 years. In the past, our scientists expended a very large effort for every new machine in an attempt to fully utilize their computing performance potential. Currently, NASA is also working on an Information Power Grid Initiative to produce a computational grid that will work in concert with computational grids being assembled at PACI [15, 16]. In anticipation of a widely distributed and heterogeneous computing environment, together with the increasing complexity of future applications, we may not be able to afford to continue our porting efforts every three years. To protect investments in code maintentance and development, the parallelization process needs to require less time and effort.

1.2 A Spectrum of Parallel Programming Alternatives

Generally speaking, four major approaches have been used to mount applications on parallel architectures:

1. Parallelization by hand;
2. Using semi-custom building blocks (*PETSc* [4], *NHSE* software [14]);
3. Data parallel languages and parallelizing compilers (*HPF* [8], *FORTRAN-D* [1], *Vienna FORTRAN* [5], *ZPL* [18], *pC++/Sage++* [11], *HPC++* [10]); and

*MRJ Technology Solutions, Inc. M/S T27A-2, NASA Ames Research Center, Moffett Field, CA 94035-10000

4. Computer aided parallelization tools and translators (*KAP/Pro Tool-Set* [13], *SUIF* [20], *FORGExplorer* [12] and CAPTools [7]).

Parallelization by hand is usually the most effective method of achieving good performance on a given machine. However, it is also the most time-consuming and expensive, often requiring the close cooperation among a team of application scientists, system programmers and the machine vendor's support staff. Not surprisingly, the hand-coded parallel code is often optimized for a particular machine and does not perform equally well on a wide range of architectures. The use of custom building blocks provides portability but limits the expression of the parallel program; the programmer has to describe the problem solving process in terms of functions already provided in the software library. This requires restructuring existing codes to make use of pre-defined functions offered in mathematical libraries. The data parallel languages and compilers generate parallel code quickly, but often at the expense of performance. Automated tools and translators promise good performance without the work of hand-coding, but can lack stability, especially for large unstructured applications.

1.3 Evaluation of Parallelization Approaches

To determine which, if any, of the parallelization approaches are appropriate for aerospace applications, we first need to establish our parallelization needs, namely: 1) good performance, 2) less coding effort than hand parallelization, 3) general enough to handle large, unstructured aerospace applications and 4) portability across heterogenous computing platforms. The second parallelization need, reduced coding effort, limits the use of semi-custom building blocks since that approach often requires significant rewrite of the source code. The fourth need, portability, limits the use of parallelizing compilers for data parallel languages since those are often machine dependent and work primarily on closely coupled and homogeneous multiprocessors. Therefore, we began our evaluation of parallelization approaches with computer aided parallelization tools and translators, which is the subject of the study reported in this paper.

CAPTools[7], a computer aided parallelization tool, was selected to evaluate this computer aided approach for parallelizing aerospace applications. We use serial versions of the NAS parallel benchmarks and ARC3D, a computational fluid dynamics (CFD) application, as our test suite representing the very basic requirements of aerospace applications. For each of the test applications, we investigate what performance we can achieve and how much user interaction is required. In addition, we need to verify that a computer aided approach can handle aerospace applications; for this reason, we identify the limitations of the tool. Finally, we evaluated the portability of this parallelization approach ; we tested CAPTools generated parallel code on two different platforms: the shared memory SGI Origin 2000 and the distributed memory Cray T3E. Based on these initial experiments, we will make observations about the feasibility of computer aided parallelization for aerospace applications.

1.4 Outline of the Paper

In this paper, we first describe the experimental environment in Section 2. We describe CAPTools, the computer aided parallelization tool used in this study, the applications that were parallelized: serial versions of the NAS Parallel Benchmarks version 2.3 and ARC3D, and the two test platforms used: the SGI Origin 2000 and the Cray T3E. In Section 3, we present our experiences with CAPTools in the following areas: 1) performance, 2) user effort required, 3) limitations and 4) portability. Finally, in Section 4, we provide conclusions about the significance of this study for the parallelization needs for CFD applications and how we plan to extend this research in the future.

2 Experimental Set-up

2.1 Computer Aided Parallelization Tool

As discussed in Section 1.2, the use of automated approaches for parallel programming is not a new idea. We want to explore approaches that do not require extensive manual modification of the code, and that cater for execution on distributed heterogeneous systems. Based on our initial survey, these concerns eliminate the semi-custom building blocks, the data parallel languages and compilers, and the translators for shared memory machines.

Further elimination among the remaining candidates (FORGE Explorer DMP [12] and CAPTools [7]) was carried out. FORGE Explorer DMP generates message-passing parallel code, but does not perform extensive dependency analysis. Therefore, the user must still perform this step in order to generate efficient parallel code. CAPTools, however, does perform extensive dependency analysis. For this important reason, we chose this tool as the basis of our evaluation of computer aided parallelization.

CAPTools[7] , a software package developed by researchers at the University of Greenwich, performs interactive parallelization of serial code using graphical interfaces. Version 1.3 beta was used for this study. CAPTools takes as input sequential FORTRAN 77 code and generates parallel SPMD message-passing code as output. Communication within the parallel code is performed via calls to the CAPTools communication library. Versions of this library, which are based on libraries such as MPI and PVM, are available for many different high performance computing platforms. The user must guide the parallelization process by deciding on the level of dependency analysis, the dat partitioning strategy, etc. This user interaction will be described in detail in Section 3.2.

In order to facilitate this evaluation, the Automated Instrumentation and Monitoring System, or AIMS [21] was also used to assist in the collection, analysis and visualization of performance data.

2.2 Test Applications

In this study, we wished to use applications that are representative of aeroscience applications. Because this study represents our initial experiences, we started with the simple, basic benchmarks and applications--the NAS Parallel Benchmarks (NPB) and ARC3D. The NPB's contain many of the basic routines and data structures contained in large CFD applications. ARC3D is a full CFD application.

2.2.1 NAS Parallel Benchmarks

The NAS Parallel Benchmarks (NPB's) [2, 3, 6] were developed by researchers at the Numerical Aerodynamic Simulation (NAS) Division at NASA Ames Research Center. These benchmarks were derived from CFD codes. They were designed to compare the performance of highly parallel computers and are widely recognized as a standard indicator of computer performance. NPB consists of five kernels and three simulated CFD applications derived from important classes of aerophysics applications. These five kernels mimic the computational core of five numerical methods used by CFD applications. The simulated CFD applications reproduce much of the data movement and computation found in full CFD codes.

For this case study, we used six of the NPB's: LU, SP, BT, FT, MG and CG.

LU is a simulated CFD application that uses symmetric successive over-relaxation (SSOR) to solve a block lower triangular-block upper triangular system of equations resulting from an unfactored implicit finite-difference discretization of the Navier-Stokes equations in 3D.

SP is a simulated CFD application that uses an implicit algorithm to solve the 3D compressible Navier-Stokes equations. The finite differences solution to the problem is based on a Beam-Warming approximate factorization that decouples the x, y and z dimensions. The resulting system has scalar pentadiagonal bands of linear equations that are solved using Gaussian elimination without pivoting. Within the algorithm, this solution is performed by the ADI solver (alternating direction implicit) which solves the three sets of systems of equations sequentially along each dimension.

BT is a simulated CFD application that solve systems of equations resulting from approximately factored implicit finite-difference discretization of the Navier-Stokes equations in three dimensions. The BT code solves block-tridiagonal systems of 5x5 blocks, but the solution algorithm has the same structure as SP.

FT contains the computational kernel of a three-dimensional Fast Fourier Transform (FFT)-based spectral method. FT performs 3 one-dimensional FFT's, one for each dimension.

MG uses a multigrid method to compute the solution of the three-dimensional scalar Poisson equation. It requires highly structured long distance communication and tests both short and long distance data movement. The algorithm solves for the solution over the grid of data points that is recursively made coarse and then fine again. This changing grid size requires that the arrays in the code are indirectly accessed.

CG uses a conjugate gradient method to compute an approximation to the smallest eigenvalue of a large, sparse, symmetric positive definite matrix. This kernel tests unstructured grid computations and communications by using a sparse, unstructured matrix with randomly generated locations of entries. CG, also requires indirect array accesses.

2.2.2 ARC3D

In addition to using the benchmarks, we also use a moderate size application, ARC3D[17], a well-known CFD code. ARC3D was developed by Thomas Pulliam at NASA Ames Research Center. It solves Euler and Navier-Stokes equations in three dimensions using a single rectilinear grid. A Beam-Warming algorithm is used to approximately factorize an implicit scheme of finite difference equations, which is then solved alternatively in three directions. The implemented Alternating Direction Implicit (ADI) solver sweeps through each of the cardinal directions one at a time, with partial updating of the fields after each sweep. ARC3D is similar in structure to SP, but is a more realistic application. It includes a curvelinear coordinate system, turbulent model and more realistic boundary conditions.

2.3 Test Platforms

Two computing platforms were used in our experiments: a SGI Origin 2000 and a Cray T3E. Because CAPTools generates communication based on a portable communications library, the same code can be tested on both platforms. The Origin 2000 is a shared memory machine that has 64 processors sitting on top of 16GB of shared memory. The processors are arranged in pairs with 512MB local memory on a single card. Each processor has a MIPS R10000 64-bit CPU (195 MHZ) with two 32KB primary caches and one 4MB secondary cache. The Cray T3E at NASA/Goddard Space Flight Center consists of 512 processor elements (PEs), 64 gigabytes of globally addressable memory, and 480-gigabyte disk. Each PE is a 64-bit DEC Alpha microprocessor (300 MHz) with an 8 KB primary data cache and a 96 KB secondary cache that is three-way set associative.

3 Evaluation of Computer Aided Parallelization

3.1 Performance

The execution times of the CAPTools generated parallel versions of the benchmarks: LU, BT and SP are presented in Figures 1, 2 and 3. (The other 3 benchmarks could not be parallelized using CAPTools; the reasons for this will be discussed in detail in Section 3.3.) Figures 1 and 2 compare the performance of the CAPTools code to the hand coded MPI on the Origin 2000 for problem sizes Class A and Class B, respectively. Figure 3 provides the same comparison on the Cray T3E for Class A. Figure 4 presents the performance of ARC3D on the Origin 2000 and the Cray T3E. Because a hand-coded parallel version of ARC3D does not exist, the CAPTools generated code was compared against an ideal linear speedup curve, which is based on the execution time of the serial code.

Several general observations can be made. First, the CAPTools code for the NPB's has similar performance as the hand-coded for one processor, indicating that the serial code is close to NPB 2.3. Next, for all applications, the CAPTools code scales well; almost as well as the hand-coded for the NPB's and close to linear for ARC3D.

More specifically, for LU, the CAPTools code performance is very close to that of NPB 2.3 (within 10% of the execution time) because CAPTools is able to generate code that is almost identical to the hand-coded version.

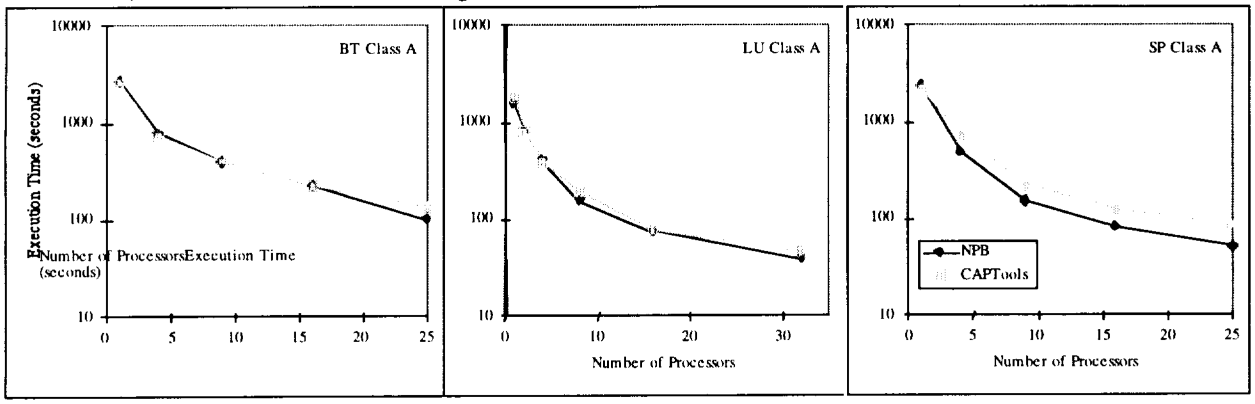

Figure 1: Execution time of LU, BT, SP Class A on Origin 2000

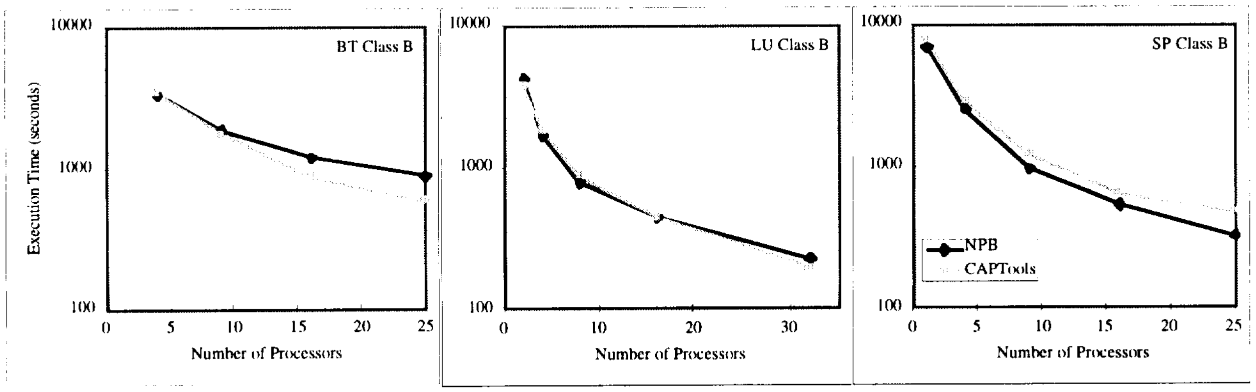

Figure 2: Execution time of LU, BT, SP Class B on Origin 2000

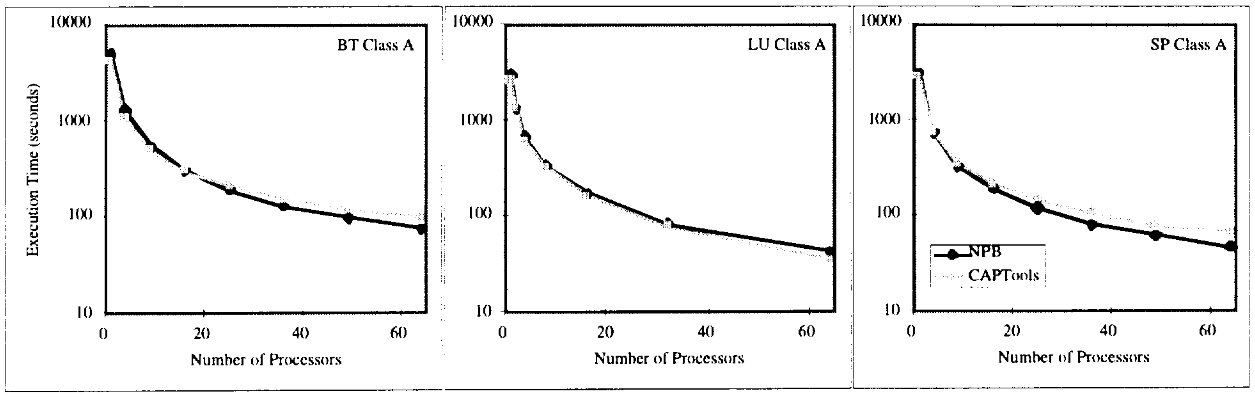

Figure 3: Execution time of LU, BT, SP Class A on Cray T3E

For BT, CAPTools code actually performs better than the hand-coded version for small numbers of processors. This is because the serial code, which is the basis for the CAPTools generated code, has better cache optimization than NPB 2.3. For larger number of processors, however, the hand-coded version has better performance because the hand-coded version uses a highly specialized partitioning strategy, multiparitioning[19], to reduce communication. It should be noted that while multipartitioning results in good performance for BT, it would not be applicable for a typical CFD application. On the Origin 2000, it appears as though the CAPTools code outperforms the NPB 2.3 code for Class B. For this case, the Origin 2000 system was not isolated; therefore, the execution of the NPB 2.3 code probably had interference from other processes. We plan to test these codes again to confirm our supposition. In general, the CAPTools code for BT was always within 20% of the execution time of the NPB 2.3.

The performance of the CAPTools parallel code for SP is not as similar to the hand-coded versions as those for LU and BT. For small numbers of processors, the difference in performance between CAPTools and NPB 2.3 is within 20%. However, as the number of processors increases, this difference increases to up to 100%. Similar to BT, the multipartitioning strategy used by NPB 2.3 SP has much better performance than the one or two dimensional partitioning of CAPTools. The impact of the partitioning is greater on the performance of SP because it requires twice as much communication as BT does.

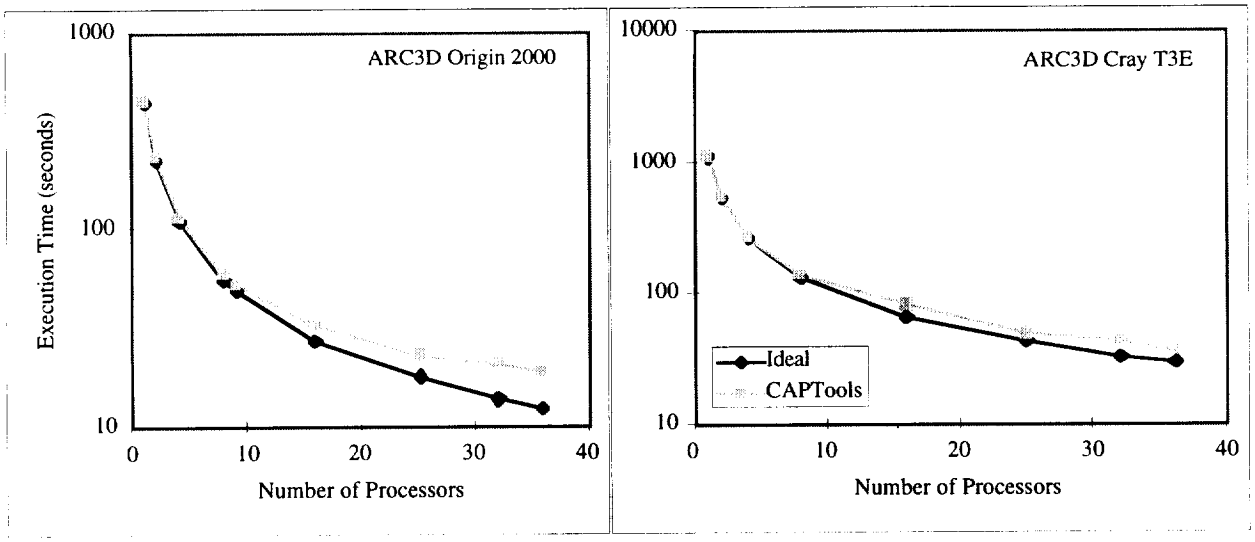

Figure 4: Execution Time of ARC3D on Origin 2000 and Cray T3E

For ARC3D, the results were similar. CAPTools generates code that scales well, especially for the Cray T3E.

For all the test applications, the CAPTools code performs very well. We recognize that this is an important observation; however, there are other considerations for our evaluation. User interaction, limitations and portablility are important concerns as well.

3.2 User Interaction

In CAPTools, as with any computer aided parallelization tool, the user must provide input to guide the parallelization process. This includes both inputs to the tool as well as tuning of the resulting parallel code. In this section, we discuss the types of interaction performed for this parallelization study. We comment about the nature of the input and try to generalize beyond the specific tool that we used for our evaluation.

We note that the amount of time it took to generate good code using CAPTools was significantly less than that for hand-coding. In all cases, it took at most a day to generate code using CAPTools and another several weeks to tune it. In comparison it took several months to write the hand-coded versions of the NAS Parallel Benchmarks.

3.2.1 User Inputs

For all the applications, it was necessary to input the relative values of the major parameters of the application. In all cases, this corresponded to entering the relative values of the problem size in each of the dimensions. This user input was critical to achieving good performance; without this information, the tool cannot remove as many dependencies and generates extra, unnecessary communication.

While CAPTools requires that the user interact with the tool to provide this information, this may not be necessary for other tools. Providing this information in a simple file to be read by the tool would be sufficient. Furthermore, this information should be readily available from the user. Therefore, this input requirement does not represent a significant burden for users.

3.2.2 Choice of Partitioning

When using CAPTools, the user must select the arrays to be partitioned and in which dimension(s) partitioning takes place. Fortunately, the user does not need to specify every array to be partitioned; the tool will automatically partition any array that is dependent on the chosen partitioned array(s). In all applications, it was sufficient to select one major array to be partitioned and CAPTools partitioned the rest. Anyone with knowledge of his/her code should be able to select the appropriate array to be partitioned.

The choice of which dimensions to partition is not so obvious. In our study, we experimented with partitioning in one dimension versus two dimensions. To illustrate the importance of choosing a good partitioning scheme, we compare the performance of both one and two dimensional partitioning for SP. Similar results were obtained for LU, BT and ARC3D as well.

SP uses an ADI solver that solves for the x, y and z dimensions, one at a time. Therefore, computation in the partitioned dimension of the parallel code must be pipelined. Figure 5 shows that one-dimensional partitioning for SP results in a pipeline with a great amount of idle time. Only the z dimension is partitioned; the other two dimensions do not require any pipelines or communication among processors.

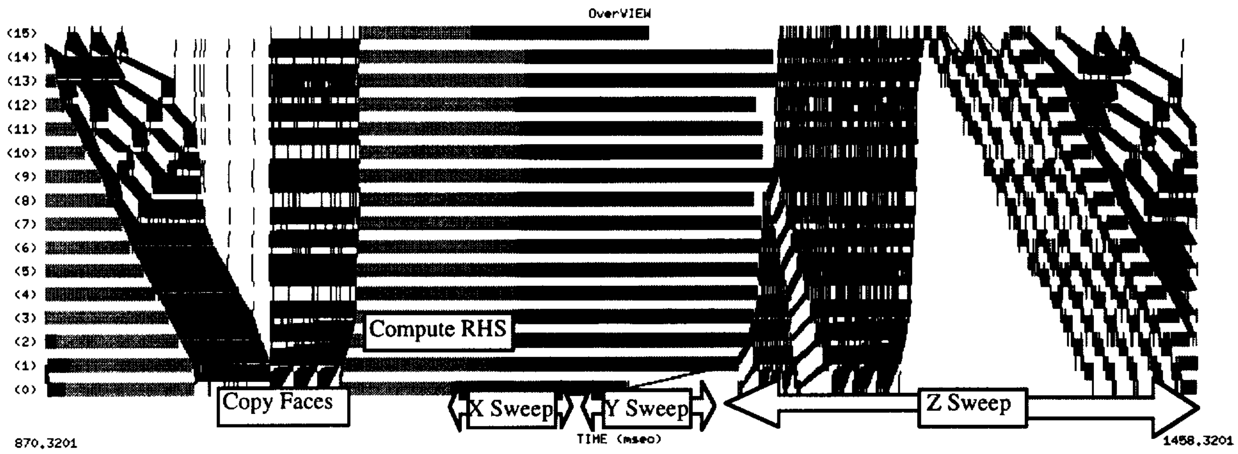

Figure 5: CAPTools-generated One-Dimensional Pipelining in SP

Figure 6 shows the two-dimensional partitioning for SP. Now there are pipelines and communication in two dimensions, but the pipelines are shorter and result in less idle time than the one-dimensional partitioning. There is still significant idle time, however.

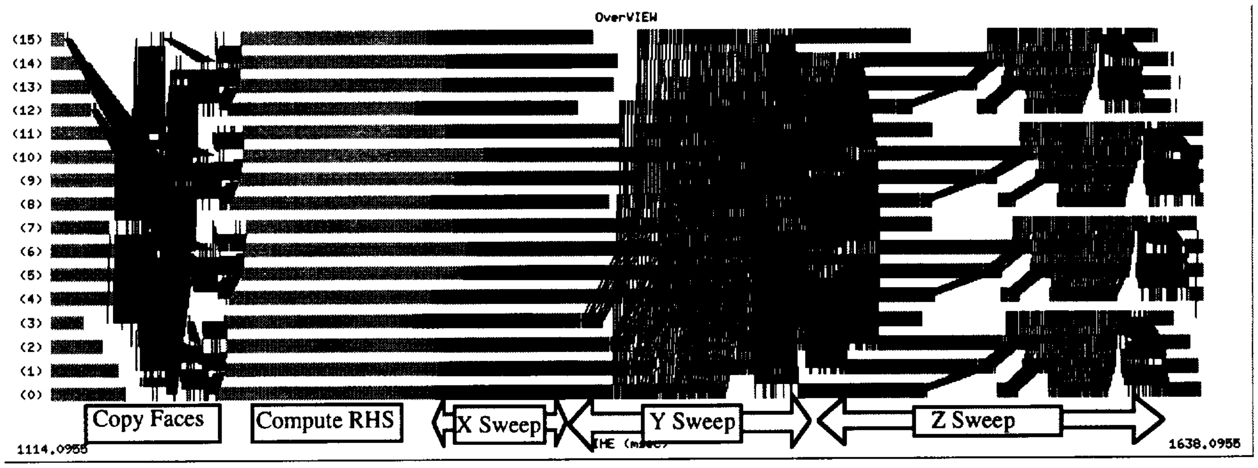

Figure 6: CAPTools-generated Two-Dimensional Pipelining in SP

In general, all the codes we parallelized perform pipelined computation. For this reason, partitioning in two dimensions is more effective. On the other hand, if the serial code did not use pipelined computation, partitioning in one dimension may be more effective since it would result in fewer messages. Being able to select the best partitioning strategy requires a good understanding of the code and often trial and error. This can be somewhat time consuming; however, the advantage to using automated parallelization is that we can quickly generate code that uses different partitioning and verify which scheme is best.

3.2.3 Communication Tuning

In CAPTools, not a lot of input is required from the user for generating communication. The user does have the choice of buffering communication. In tuning the code, however, large amounts of communication identify bottlenecks and lead to further revisions of the code. Tuning communication can require a lot of effort by the user, as will be shown by the following example. This type of tuning is somewhat unavoidable; it is required by most parallelization approaches.

It is important that communication is done carefully because it critically affects performance. In this section, we show how we interact with CAPTools, based on feedback from AIMS visualizations, to eliminate unnecessary communication. Although only the results of tuning on LU is presented here, these were similar to both SP and BT.

The first version of the code was generated automatically. The execution of the code on four processors on the SP2 is illustrated in a space-time diagram as shown in Figure 7. Horizontal bars represent computation in different subroutines (each has a different color), while the lines between the blocks represent communication and white spaces represent idle times. It can be seen that subroutines `erhs` and `rhs` generate significant communication.

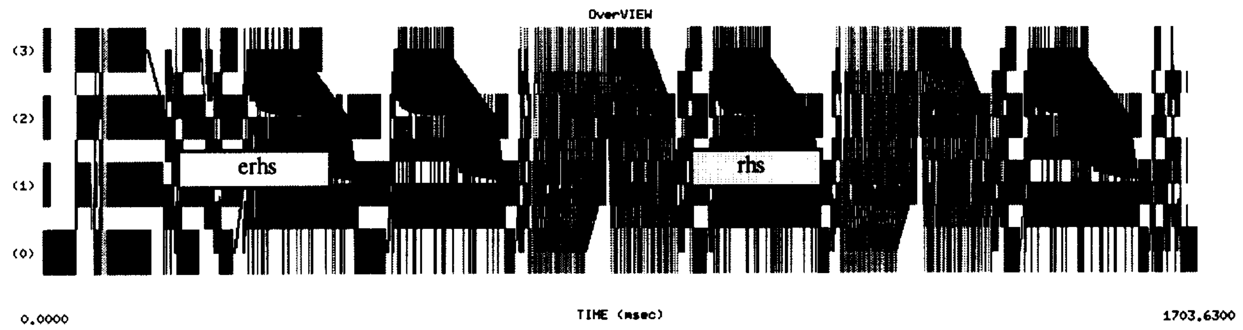

Figure 7: Automated parallelized version of LU with no tuning

An inspection of Figure 7 shows that subroutines `erhs` and `rhs` communicate boundary values for several local arrays, however, this communication is unnecessary since these values can be computed. To correct this, the user asked CAPTools to remove partitioning performed for local arrays (local arrays being any arrays that are not dependent on any data on another processor for their assignment), so that the boundaries of the array will be computed instead of communicated. As shown in Figure 8, the total execution time decreased by 38% because of the reduction in communication. Large amount of communication still exist in `erhs` and `rhs` due to large numbers of small messages. We chose the "buffered communication" option and were able to reduce total execution time by another 8% as shown in Figure 9. The number of messages in the `erhs` and the `rhs` subroutines is reduced which results in better performance.

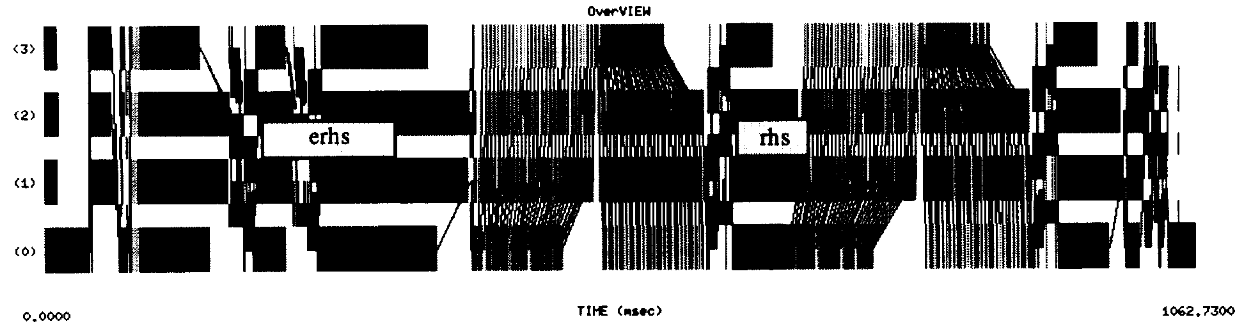

Figure 8: CAPTools generated LU without local array partitioning

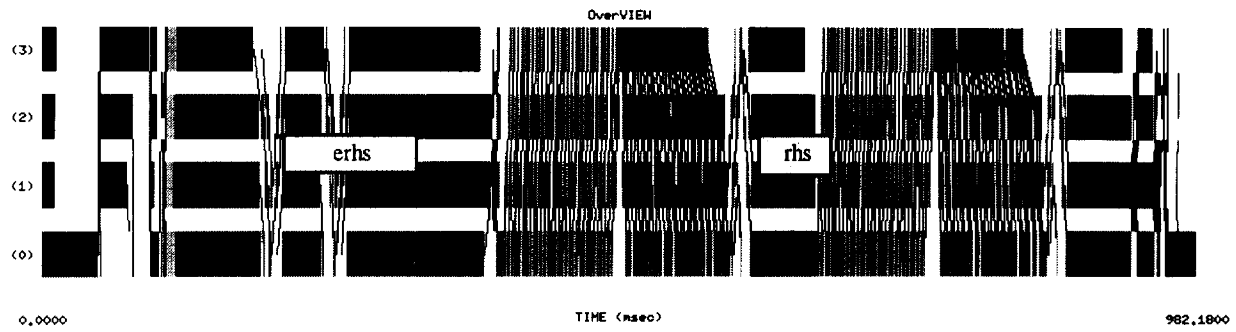

Figure 9: CAPTools generated LU based on "buffered communication"

3.2.4 Routine Copy

The final type of interaction we encountered was the necessity of copying routines. The reason that this is necessary is that one routine is often called with different dimensions of a given array. When the array is partitioned, one version of the routine can no longer handle unpartitioned dimensions and each of the partitioned dimensions. For example, in ARC3D, the VPENTA3 and VPENTA routines needed to be copied to handle the different data partititons for the one and two dimensional partitionings.

Copying routines requires some user knowledge of the code, but must be performed in order for the arrays to be partitioned. CAPTools implements this as a user decision; however, it would be possible to automate this process for simple cases such as the one encountered in ARC3D.

3.2.5 Rewriting Code

Finally, there is some tuning that cannot be performed using CAPTools; sometimes code must be rewritten by hand in order to achieve good performance. This type of optimization was not required by the NPB's; however, it was for ARC3D. In ARC3D, the solvers used for the x, y and z directions were originally written to solve for grid points on one 2 dimensional plane at a time. When solving for the direction with partitioned data, the computation is pipelined across the processors for each two dimensional plane being solved. This is portrayed in Figure 10, Version 1. The AIMS trace diagram for Version 1 during the "Z sweep" shows many pipelines, each incurring idle time (shown by the white space). In order to reduce this idle time, the computation was regrouped so that the three dimensional grid points were solved at once, instead of two dimensions at a time. This results in one large pipeline and much less idle time than with multiple small pipelines; this is illustrated in Figure 10, Version 2.

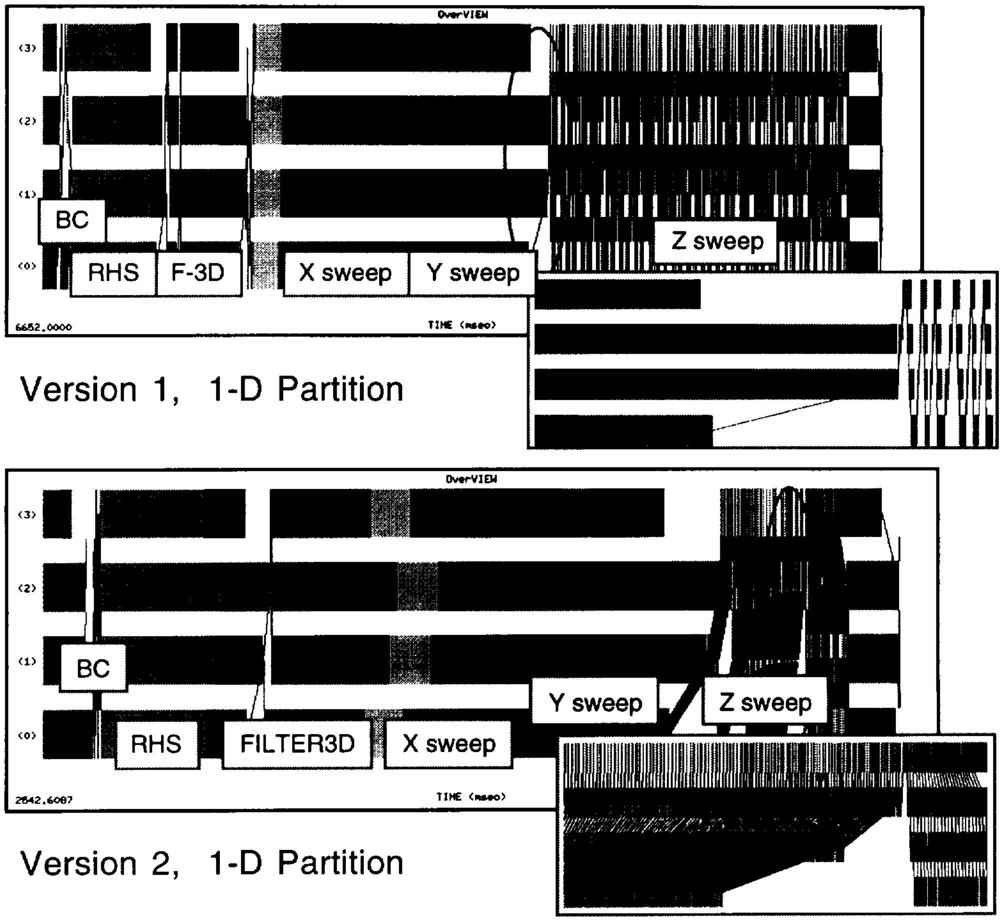

Figure 10: ARC3D: Comparison of versions

3.3 Limitations

Another important consideration of our evaluation is to identify the limitations of the tool. As mentioned previously, CAPTools could not parallelize all the benchmarks. In this section, we summarize the parallelization requirements of the applications in order to best identify the shortcomings of the tool.

Table 1 provides four major dimensions are critical to parallelizing the test applications: grid structure, array accessing, communication and data partitioning. The + sign represents features that CAPTools can currently handle; the – signs represent what it cannot. Therefore, if an application requires a feature not currently supported by CAPTools, it cannot be parallelized. MG, CG and FT all contain at least one unsupported feature; therefore, theycannot be parallelized. On the other hand, LU, SP, BT and ARC3D can be parallelized in a relatively straightforward manner; these applications all had structured grids with direct array accesses, communication with only nearest neighbors and static partitioning. This is arguably the simplest case for a parallelization tool to handle. While it is reassuring that CAPTools and computer aided parallelization in general can handle this simple case well, there are still more complex considerations that must be addressed.

Application	**Grid**		**Array Access**		**Communication**		**Partitioning**	
	Struct.	**Unstruct.**	**Direct**	**Indirect**	**Neigh.**	**Non-neigh**	**Static**	**Dynamic**
SP	+		+		+		+	
LU	+		+		+		+	
BT	+		+		+		+	
MG	+			–		–	+	
CG		–		–	+		+	
FT	+		+			–		–
ARC3D	+		+		+		+	

Table 1: Summary of Parallelization Requirements for test applications

First, in FT, the best parallelization approach involves distributing the data in such a way so that each one dimensional FFT can be performed without communication. For example, when computing the FFT in the x dimension, data should be distributed in the y and/or z dimensions so that there are no data dependencies between processors in the x dimension. Otherwise, the amount of communication required within the one dimensional FFT calculation is overwhelmingly prohibitive. The hand-crafted version in NPB 2.3 handles this data distribution problem by performing data transpositions. For example, first the data is partitioned in the z dimension. The one-dimensional FFT in the x and y dimensions can now be performed without any communication. Then, the data is transposed so that the partitioned dimension is now the x dimension. Consequently, the one-dimensional FFT in the z dimension can now be performed without requiring large amounts of communication. The transposition requires all-to-all communication, but it is much smaller than that required by a distributed FFT calculation. Unfortunately, since CAPTools does not handle dynamic re-partition of data, we were not able to produce a parallel FT with comparable performance at this point.

MG requires the grid to be distributed onto the nodes in a manner best described as "successively halved", starting with the z dimension, the y dimension and then the x dimension, and repeated until all processors are assigned. CAPTools cannot parallelize MG for two reasons. First, it performs only static, block data distribution, so the changing grid size results in large amounts of communication to non-neighboring processors. CAPTools can perform this communication only through global broadcasts. Secondly, CAPTools cannot handle the indirect accessing of arrays that is used in MG.

Finally, CG employs an unstructured grid and uses indirect array accesses which CAPTools does not support.

The unstructured grids, indirect array accesses and non-neighbor communication are critical elements of many large aeroscience applications. Therefore, any computer aided parallelization tool needs to address these elements. Unfortunately, this particular evaluation using CAPTools did not suggest that it could be done currently. Nevertheless, the underlying technology for handling these elements is already in place; it is the technology that handles the simple case. Furthermore, subsequent beta versions of CAPTools are currently being tested which handle unstructured grids, indirect array accesses and non-neighbor communication. These versions have been successful in handling these concerns in test programs and show promise for parallelizing large aeroscience applications.

3.4 Portability

Portability is another important concern because of our rapid turnover of high performance computers and in anticipation of the Information Power Grid. CAPTools' portability is based on the generation of parallel code that contains generic communication calls to a library, CAPLib. This CAPLib is unique to each architecture; however, since it is based on lower level libraries such as MPI, PVM or shared memory library calls, it is trivial to port. For example, the Origin 2000 CAPLib that is used in this study is based on MPI, but on the Cray T3E, we used the CAPLib based on shmem library calls. Furthermore, the CAPLib library is linked with the executable at compile time; therefore, the same version of the parallel code can be used on multiple architectures and platforms. Finally, the good performance demonstrated by results on both the Origin 2000 and the Cray T3E confirms that the CAPTools code is portable.

4 Conclusions and Future Research

Parallel programming is a difficult process; there are many challenges to generating efficient parallel code. In this paper, we evaluated one approach to parallelization using an automated parallelization tool, CAPTools. We used a test suite of applications including the NAS Parallel Benchmarks and a CFD application, ARC3D. We evaluated CAPTools in four areas: 1) performance, 2) user interaction, 3) limitations and 4) portability.

For our test cases, we found that the performance and portability of CAPTools is similar to that of hand-coded programs using MPI. The tool is capable of handling codes using structured grids; however, it cannot currently handle unstructured grids, indirect array accesses, non-neighbor communication or dynamic data distribution. In short, these findings confirm what is already publicly accepted opinion of computer aided parallelization: the performance is good for highly structured cases, but unstructured programs cannot currently be handled. Nevertheless, future versions of CAPTools which are currently being tested suggest that these concerns will be able to be addressed eventually using computer aided parallelization.

Furthermore, our study specifically identified the type of user interaction that is required by computer aided parallelization. Most of the interaction is simple and involves providing readily available information to the tool. Other interaction, such as partitioning choices and communication tuning, require trial and error and performance evaluation by the user. Even though the tuning process can be lengthy, the computer aided tool can significantly reduce the time by automating the generation of successive parallel versions of the code. For example, CAPTools allowed us to easily generate one and two dimensional partitionings for the same code so that we can determine the best approach. Finally, even though some rewriting of the code may be necessary to generate good code, the amount of user interaction required by the computer aide d parallelization approach is significantly less than that required by hand-coding.

In summary, our evaluation of the CAPTools shows that a computer aided parallelization approach performs very well in three of the four areas that we evaluated: performance, amount of user interaction required and portability. The fourth area, identifying the limitations of the tool, showed that CAPTools can handle structured grid problems. Tests of future versions of CAPTools will be needed to further evaluate whether this approach effectively handles large, unstructured problems.

A more extensive study is underway to incorporate performance of other parallelization tools/compilers (e.g. *HPF* [8], *SUIF* [20], FORTRAN-D[1], other architectures (network of workstations), and other more complex applications (e.g. OVERFLOW). The results of some of these experiments will be reported in an upcoming paper.

5 Acknowledgments

The authors wish to acknowledge Mark Cross, Steve Johnson, C.S. Ierotheou, Emyr Evans and Pete Leggett for their help with CAPTools. The authors also wishes to acknowledge the entire AIMS development team which includes Melisa Schmidt, Cathy Schulbach, and Michael Frumkin. The work described here is supported under NASA's High Performance Computing and Communications Program.

6 References.

[1] Vikram S. Adve, John Mellor-Crummey, Mark Anderson, Ken Kennedy, Jhy-Chun Wang, and Daniel Reed, "An Integrated Compilation and Performance Analysis Environment for Data Parallel Programs," presented at Supercomputing '95, San Diego, CA, 1995.

[2] D. Bailey, J. Barton, T. Lasinksi, and H. Simon, "The NAS Parallel Benchmarks," NASA Ames Research Center, Moffett Field, CA RNR-91-002, 1991.

[3] D. Bailey, T. Harris, W. Saphir, R. Van der Wijngaart, A. Woo, and M. Yarrow, "The NAS Parallel Benchmarks 2.0," NASA Ames Research Center, Moffett Field, CA RNR-95-020, 1995.

[4] Satish Balay, Bill Gropp, Lois Curfman McInnes, and Barry Smith, "PETSc Library" . Argonne National Laboratory: http://www.mcs.anl.gov/petsc/petsc.html.

[5] S. Benkner, "Vienna FORTRAN 90- An Advanced Data Parallel Language," presented at International Conference on Parallel Computing Technologies (PACT-95), St. Petersburg, Russia, 1995.

[6] NAS Division NASA Ames Research Center, "NAS-Parallel Benchmarks" , 2.3 ed. Moffett Field, CA: http://science.nas.nasa.gov/Software/NPB, 1997.

[7] M. Cross, C.S. Ierotheou, S.P. Johnson, P. Leggett, and E. Evans, "Software Tools for Automating the Parallelisation of FORTRAN Computational Mechanics Codes," *Parallel and Distributed Processing for Computational Mechanics*, 1997.

[8] High Performance FORTRAN Forum, "High Performance FORTRAN Language Specification Version 1.0," *Scientific Programming*, vol. 2, 1993.

[9] G.C. Fox, S. Hiranandani, K. Kennedy, C. Koelbel, U. Kremer, C.W. Tseng, and M. Wu, "The FORTAN D Language Specification," CRPC, Rice University CRPC-TR90079, 1990.

[10] Dennis Gannon, Peter Beckman, Elizabeth Johnson, Todd Green, and Mike Levine, "HPC++ and HPC++ Lib Toolkit," Indiana University, Bloomington, IN, 1997.

[11] Dennis Gannon, Shelby X. Yang, and Peter Beckman, "User Guide for a Portable Parallel C++ Programming System pC++," Department of Computer Science and CICA, 1994.

[12] Applied Parallel Research Inc., "FORGE Explorer" : http://www.apri.com.

[13] Kuck and Associates Inc., "Parallel Performance of Standard Codes on the Compaq Professional Workstation 8000: Experiences with Visual KAP and the KAP/Pro Toolset under Windows NT," , Champaign, IL.

[14] NHSE, "HPC-Netlib Software Catalog" : http://nhse.cs.utk.edu/rib/repositories/hpc-netlib/catalog.

[15] NPACI, "http://www.npaci.edu/Research" .

[16] PACI, "http://www.cise.nsf.gov/acir/paci.html" .

[17] T.H. Pulliam, "Solution Methods in Computational Fluid Dynamics," in *Notes for the von Karman Institute for Fluid Dynamics Lecture Series*. Belgium: Rhode-St-Genese, 1086.

[18] Lawrence Snyder, "A ZPL Programming Guide," University of Washington, 1998.

[19] R. Van der Wijngaart, "Efficient Implementation of a 3-dimensional ADI Method on the iPSC/860," NASA Ames Research Center, Moffett Field, CA, 1995.

[20] Robert P. Wilson, Robert S. French, Christopher S. Wilson, Saman P. Amarasinghe, Jennifer M. Anderson, Steve W.K. Tjiang, Shih-Wei Liao, Chau-Wen Tseng, Mary W. Hall, Monica Lam, and John Hennessy, "SUIF: An Infrastructure for Research on Parallelizing and Optimizing Compilers," Computer Systems Laboratory, Stanford University, Stanford, CA.

[21] J.C. Yan, S.R. Sarukkai, and P. Mehra, "Performance Measurement, Visualization and Modeling of Parallel and Distributed Programs using the AIMS Toolkit," *Software Practice and Experience*, vol. 25, pp. 429-461, 1995.