Interfacing Computer Aided Parallelization and Performance Analysis

Gabriele Jost[1*], Haoqiang Jin[1], Jesus Labarta[2], and Judit Gimenez[2]

[1] NAS Division, NASA Ames Research Center, Moffett Field, CA 94035-1000 USA
{gjost,hjin}@nas.nasa.gov
[2] European Center for Parallelism of Barcelona-Technical University of Catalonia (CEPBA-UPC), cr. Jordi Girona 1-3, Modul D6,08034 – Barcelona, Spain
{jesus,judit}@cepba.upc.es

Abstract. When porting sequential applications to parallel computer architectures, the program developer will typically go through several cycles of source code optimization and performance analysis. We have started a project to develop an environment where the user can jointly navigate through program structure and performance data information in order to make efficient optimization decisions. In a prototype implementation we have interfaced the CAPO computer aided parallelization tool with the Paraver performance analysis tool. We describe both tools and their interface and give an example for how the interface helps within the program development cycle of a benchmark code.

1 Introduction

During the last decades a large amount of time and money has been spent on the development of large-scale scientific applications which expose an insatiable appetite for floating point operations per second. High performance computer architectures have evolved to satisfy this demand, however, the structure of the existing codes is not always suitable to fully exploit the parallelism provided by the hardware. Considering the enormous investment in these codes and the fact that scientific applications are a niche market, there is a strong incentive not to re-implement the applications from scratch, but rather employ a conversion process to produce versions of existing codes that are optimized for the current most powerful computer architecture.

In this paper we will focus on shared memory parallel computer architectures which provide multiple processing units and a globally shared address space. High performance is achieved by multithreaded execution of loops. If the iterations of a loop can be carried out independently, each thread executes a subset of the iteration space in parallel. In order to determine whether a loop can be parallelized the array indices have to be analyzed for potential data dependencies. The technique of dependence analysis is well understood [19] and has been used for compiler optimization. Ideally the com-

[*] The author is an employee of Computer Sciences Corporation.

piler would determine if parallelization of a loop is possible, making the exploitation of the parallel hardware transparent to the user. In practice, however, the question of whether a loop can be executed in parallel is difficult to answer either due to the complexity of the code or the fact that it depends on values that are only known at runtime. By its nature the compiler has to be conservative and has to assume the existence of a dependence if it can not proof otherwise. Therefore parallelization completely left to compiler will often results in poor performance.

Most compilers for shared memory parallel architectures support parallelization in form of compiler directives such as those provided by the OpenMP standard [13]. This allows the user to indicate to the compiler information such as which loops are parallel and whether variables are private to a thread or shared among the threads. The program developer has to determine where to insert the directives and has to specify the scope of the variables. Although this is much easier than explicitly writing multi-threaded code, the task is still time consuming and error prone.

The CAPO [6] parallelization support tool was developed at the NASA Ames Research Center to aid the program developer in this task. CAPO automates the insertion of OpenMP directives into existing Fortran codes and allows user interaction for an efficient placement of the directives. When parallelizing an existing application, the program developer will go through several cycles of placement of OpenMP directives into the code, analysis of the performance, and optimization. To facilitate this process we have interfaced CAPO with a performance analysis tool. Our goal is to provide the user with an environment where he can jointly navigate through program structure and performance data and correlate the two. This requires a high level of flexibility and extensive analysis capabilities on behalf of the performance analysis tool.

Paraver [14] was developed at CEPBA-UPC to provide the user with means to obtain a qualitative global perception of the application behavior as well as a detailed quantitative analysis of program performance. Paraver allows the user to visually inspect trace files collected during program execution. Optionally available with the Paraver distribution is the OMPItrace module [12], which allows dynamic instrumentation and tracing of applications with multiple levels of parallelism.

The CAPO-Paraver interface is a first step towards the development of a programming environment for scientific applications. The purpose of this paper is to present the basic motivation and the general idea of how the tools should be integrated. We also describe our initial prototype implementation and give a demonstration of its potential use.

The rest of the paper is structured as follows: In Sections 2 and 3 we describe CAPO and Paraver. We discuss the interface between the tools in Section 4 and give an example parallelization and optimization session in Section 5. In Section 6 we discuss related work and draw our conclusion in Section 7 where we also elaborate on our future plans.

2 The CAPO Parallelization Support Tool

The main goal of developing parallelization support tools is to eliminate as much of the tedious and sometimes error-prone work that is needed for manual parallelization of serial applications but leaving the possibility for user interaction to allow for optimization.

2.1 The CAPO Parallelization Process

CAPO [6] was developed to automate the insertion of OpenMP compiler directives with nominal user interaction. This is achieved largely by use of the very accurate interprocedural analysis from CAPTools [4] developed at the University of Greenwich which provides a fully interprocedural and value-based dependence analysis engine. To exploit loop level parallelism CAPO goes through the following three stages:

1. Identification of parallel loops and parallel regions based on an extensive dependence analysis.
2. Optimization of parallel regions and parallel loops achieved by merging together parallel regions where there is no violation of data usage and inserting the NOWAIT clause on successive parallel loops if possible.
3. Code transformation and insertion of OpenMP directives where the call graph is traversed to place OpenMP directives within the code and the scope of the variables is defined such as `SHARED`, `PRIVATE`, and `REDUCTION`.

More details on the CAPO parallelization process can be found in [6].

2.2 The Knowledge Data Base

The results of the dependence analysis performed during a CAPO session are stored in a data base. The data base contains the dependence graph and a list of unresolved questions that occurred during the analysis. Unresolved questions indicate conservatively defined dependencies, often due to unknown information about values of variables. The user has the possibility to browse the list of questions and provide assertions to make a more precise dependence analysis possible.

2.3 The CAPO Directives Browser

The CAPO directives browser is designed to display information gathered during the parallelization. For instance, the browser provides information on the reasons for loops to be parallel or serial and the relevant variables. For each subroutine the user can retrieve information about the loops it contains. The browser also provides interfaces for the user to declare certain variables as shared or private or to explicitly remove dependences.

3 The Paraver Visualization and Analysis System

The Paraver system allows performance analysis of process level, thread level and hybrid parallel programs. It consists of two major components: A tracing package and a graphical user interface to visually examine the traces which includes an analysis module for the calculation of various statistics.

3.1 The OMPItrace Instrumentation Module

The Paraver distribution optionally provides its own tracing package, OMPItrace. Depending on the computer system, OMPItrace allows for the dynamic instrumentation of certain runtime libraries. It allows for tracing of programs with multiple processes and multiple threads. Examples for OpenMP related information that is automatically traced on our development platform (SGI Origin 3000) are:

- entry and exit of OpenMP runtime library routines,
- entry and exit of compiler generated routines containing the body of parallel loops,
- two hardware counters
- the state of a thread (running, idle, synchronization, or in fork/join overhead).

User routines require manual instrumentation to be traced.

3.2 Trace Views and Configuration Files

The trace collected during the execution of a program contain a wealth of information, which as a whole is overwhelming. The user needs to filter information to gain visibility of a critical subset of the data. This can be done through timeline graphical displays or by histograms and statistics. Paraver provides great flexibility in composing displays of trace data. The Paraver object model is structured in a three level hierarchy: Application, Task, and Thread. Each record of trace is tagged with these three identifiers. The timelines can be displayed at the level of individual threads but also at the level of tasks. In the latter case, the values for each thread are combined to produce a value for the task. A user can specify through the Paraver GUI how to compute a given performance index from the records in the trace and then save it as a configuration file. These configuration files can then be used to immediately display a view of the selected performance index.

3.3 The Paraver Analysis Module

In general performance analysis starts by examining some basic metrics and statistics such as timelines and profiles of user functions. Further investigation may require information about hardware counters (e.g. cache misses or instructions per cycle) in terms of an absolute value, the rate per time, or the ratio of two counters. An extensive analysis will often raise the need to define new detailed metrics. The quantitative analysis module in Paraver allows the display of profile data for each thread in the form of tables or histograms. It also provides mechanisms to correlate different types of profile data with each other. Typical statistics such as average, minimum, maximum values or standard deviation can be computed for any section of the timeline views. It provides great flexibility in defining and calculating new performance metrics which results in a very powerful profiling capability.

4 The CAPO-Paraver Interface

An important step in this process is to determine, whether the directives have been placed for efficient parallelization. When working with large application packages the program developer is confronted with the question which parts of the code to focus on. We are trying to address this question by the use of trace information and an interface to a performance analysis tool. Although there are many aspects to performance optimization we will focus at this point on the parallelization aspect. The design of the interface between computer aided parallelization and performance analysis is summarized in Figure 1.

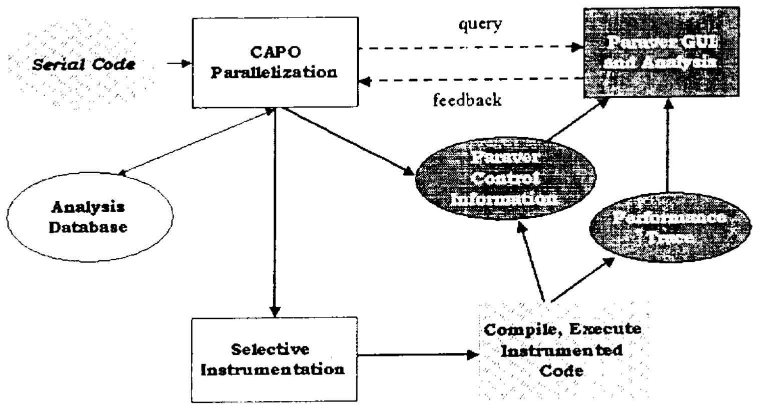


Fig. 1: The CAPO-Paraver Interface. The dashed lines indicate that our prototype implementation requires user interaction.

4.1 Selective Source Code Instrumentation

We have extended the source code transformation capability of CAPO to automatically insert calls to the OMPItrace library. The OMPItrace module allows for dynamic tracing of parallel loops and parallel regions, but does not automatically trace entry and exit to user routines on our development platforms (SGI Origin 3000). Even though the tracing of user level routines is possible on some platforms, it may not always be desirable to automatically trace all of the subroutines because of large instrumentation overhead. In our prototype instrumentation we use the following simple heuristics to insert calls to the tracing library. We automatically instrument:

- routines that are not contained within a parallel region or a parallel loop,
- routines which contain at least one DO loop.

During the selective instrumentation process CAPO generates a file containing control information which will be used by Paraver to relate the performance trace data with the routine names. The instrumented, parallelized code can now be compiled and run to obtain a trace file which can be processed by Paraver.

4.2 Display of Performance Metrics

We have designed a set of configuration files which show statistics related to the information displayed in the directives browser. The CAPO directives browser presents the user with a list of routines and means to examine the loops within the routines. To determine where to focus his efforts on, the user will need to know:

- Where is the time spent?
- Is the time spent "efficiently"?

To address these questions the user has the possibility to invoke Paraver from within CAPO. Paraver will start up, loading the previously generated trace file. The first view presents the user with a diagram containing the percentage of time that each of the routines took of the total traced time. The time is presented on a task level, averaging over all threads. Reported is the exclusive time for a function call.

To give an indication of the quality of the parallelization we display the percentage of useful time for each thread. By useful time of a thread we mean time spent not in fork/join operations, blocked or idle time. The Paraver analysis module allows us to calculate the percentage of useful time for each thread within the instrumented routines, as well as their mean and standard deviation. This view gives a first basic indication on the quality of the parallelization. The average useful time for each thread should be high and the standard deviation low.

In a third view the user is presented with the average time that each parallel loop or parallel region takes within a selected routine. CAPO automatically generates control information which is then used by Paraver for calculating the performance metrics and their display.

This set of standard views enables even the novice Paraver user to determine where to focus on during the parallelization process. An expert user has the full Paraver capability available for further investigation.

4.3 CAPO-Paraver Feedback Mechanism

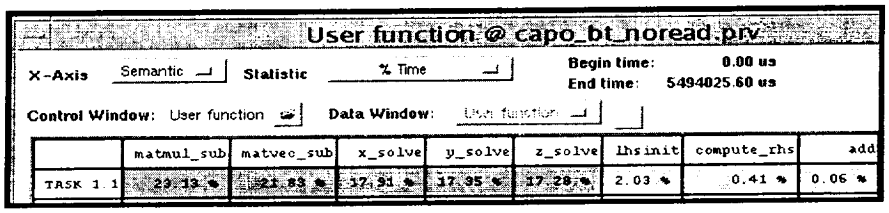
CAPO supports querying the performance trace by generating control information and invoking Paraver. In our prototype implementation it is required that the user visually inspects the performance metrics calculated by Paraver and then provides feedback to CAPO. This is indicated by the dashed lines in Figure 1. Our goal is to have CAPO query Paraver for performance metrics and retrieve the information without user interaction. We are currently working on a non-GUI based Paraver which will allow retrieving Paraver performance metrics in batch mode. With this, CAPO will be able to automatically query Paraver and correlate the outcome of the query with its loop analysis information. It might then be possible to automatically optimize the code or to prompt the user for assertions, e.g. to eliminate conservative dependences. At the very least the user can be pointed to specific performance problems. Our current prototype implementation allows us to conduct experiments to gain experience on what the nature of the queries should be and how the outcome should affect the optimization process. The use of Paraver to provide profile data has the advantage that at any point in time the user can enter in the process, inspect the metric requested by CAPO and carry out a more elaborate analysis.

5 An Example Parallelization and Optimization Session

To demonstrate a parallelization session using the CAPO-Paraver interface we consider the BT benchmark from the NAS Parallel Benchmarks (NPB) [2]. The BT benchmark solves three systems of equations resulting from an approximate factorization that decouples the x, y and z dimensions of the 3-dimensional Navier-Stokes equations. These systems are block tridiagonal consisting of 5x5 blocks. Each spatial dimension is alternatively swept.

The development platform is an SGI Origin 3000. To develop a parallel version of BT, the user starts a CAPO session loading the source code. Then CAPO performs the parallelization and generates the database. The user saves the CAPO generated source code containing OpenMP directives and the selective instrumentation as described in Section 4. Compiling and running the instrumented OpenMP code generates a performance trace file. In our demonstration the application runs on 4 threads.

Now Paraver is invoked via the link provided by the CAPO user interface. Paraver will load the performance trace file and a configuration file which calculates the percentage of time spent in each of the instrumented routines. The fragment of a snapshot is shown in Figure 2.


User function @ capo_bt_noread.prv

X-Axis Semantic Statistic % Time Begin time: 0.00 us End time: 5494025.60 us

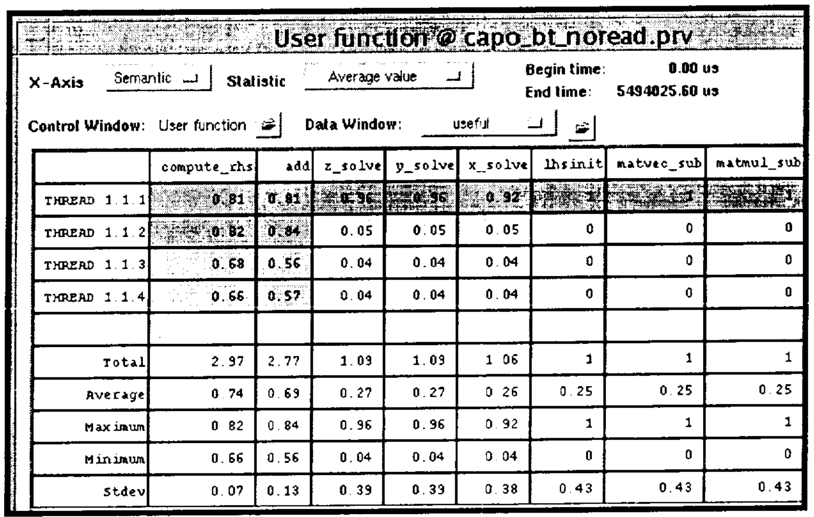
Control Window: User function Data Window: User function

	matmul_sub	matvec_sub	x_solve	y_solve	z_solve	lhsinit	compute_rhs	add
TASK 1.1	23.13 %	21.83 %	17.91 %	17.35 %	17.28 %	2.03 %	0.41 %	0.06 %

Fig. 2: Fragment of an analysis histogram displaying the percentage of time.

The useful time for each thread can be displayed by loading a second configuration file into Paraver, which is shown in Figure 3. The value of 0.96 for thread 1 in routine `z_solve` indicates that thread 1 was running user code 96% of the time. The value of 0.05 for thread 2 in routine `z_solve` indicates that the thread was running only 5%

of the time. Routines matmul_sub and matvec_sub are obviously executed by only one of the 4 threads. The imbalance in useful time of the threads is an indicator for inefficient parallelization. All of these routines are major time consumers as can be seen in Figure 2. Selecting for example routine z_solve in the directives browser and inspecting the parallelized loops reveals that the inner loops of the routine were chosen for parallelization (see Figure 4). Routines matmul_sub and matvec_sub are called from within routine z_solve, outside of the parallelized loop. For the outer loops, dependences had to be assumed.


User function @ capo_bt_noread.prv

X-Axis Semantic   Statistic Average value   Begin time: 0.00 us   End time: 5494025.60 us

Control Window: User function   Data Window: useful

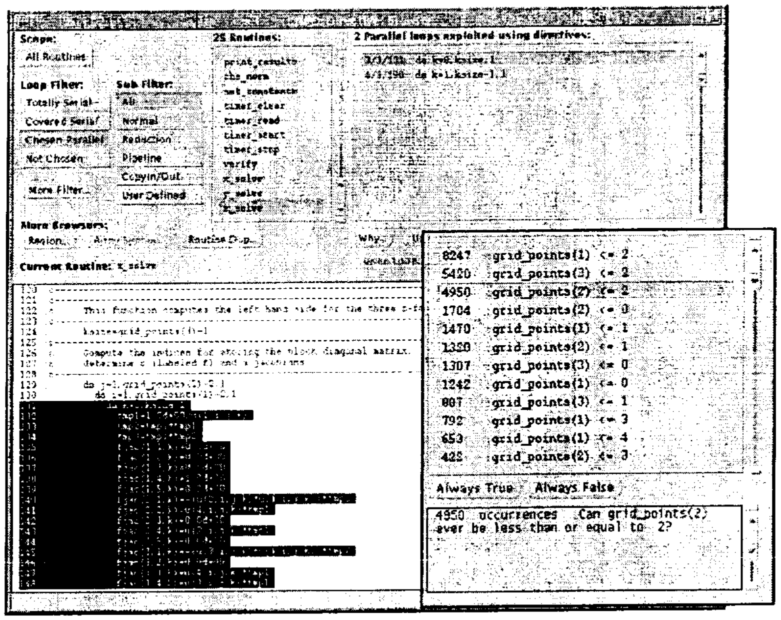
	compute_rhs	add	z_solve	y_solve	x_solve	lhsinit	matvec_sub	matmul_sub
THREAD 1.1.1	0.81	0.81	0.96	0.96	0.92	1	1	1
THREAD 1.1.2	0.82	0.84	0.05	0.05	0.05	0	0	0
THREAD 1.1.3	0.68	0.56	0.04	0.04	0.04	0	0	0
THREAD 1.1.4	0.66	0.57	0.04	0.04	0.04	0	0	0
Total	2.97	2.77	1.09	1.09	1.06	1	1	1
Average	0.74	0.69	0.27	0.27	0.26	0.25	0.25	0.25
Maximum	0.82	0.84	0.96	0.96	0.92	1	1	1
Minimum	0.66	0.56	0.04	0.04	0.04	0	0	0
Stdev	0.07	0.13	0.39	0.39	0.38	0.43	0.43	0.43

Fig. 3: Paraver analysis results showing the useful time per routine for each thread. Darker shading indicates a higher percentage of useful time.

Inspecting the knowledge database (see Figure 4) shows that there are conservative dependences because the values of entries in gridpoints are not known at compile time. These are the number of grid points for each dimension which are provided as input. They determine the loop lengths of the outer loops surrounding the parallelized loop. Since the user knows that the number of grid points is greater than 5 for each dimension, he can provide this information and repeat the analysis. The OpenMP directives are now placed on the outer loops of the solver routines. Note that the provided user knowledge benefits all three solver routines. Saving, compiling, and running the new instrumented OpenMP code results in a much better balance of useful time between the threads which is reflected in an overall performance increase of the benchmark.

6 Related Work

There are a number of commercial and research parallelizing compilers and tools for automatic parallelization that have been developed over the years.



```
c     This function computes the left hand side for the three x-f
c     Compute the indices for storing the block diagonal matrix.
c     determine c (labeled f) and s jacobians
```

```
8247  grid_points(1) <= 2
5420  grid_points(3) <= 2
4950  grid_points(2) <= 2
1704  grid_points(2) <= 0
1470  grid_points(1) <= 1
1350  grid_points(2) <= 1
1307  grid_points(3) <= 0
1242  grid_points(1) <= 0
887   grid_points(3) <= 1
792   grid_points(1) <= 3
653   grid_points(1) <= 4
425   grid_points(2) <= 3
```

```
4950 occurrences  Can grid_points(2)
ever be less than or equal to  2?
```

Fig. 4: The CAPO directives browser and knowledge database.

The SUIF Explorer [8] developed at Stanford University is an interactive parallelization tool based on the SUIF [17] compiler. It performs extensive static dependence analysis and also includes a set of dynamic analyzers to provide runtime information. Runtime information includes checking dependencies and time profiles for loops and routines. The user can provide assertions via a graphical user interface. The most notable difference to our approach is that by interfacing to a full performance analysis system like Paraver, we have more performance metrics available than just the time. This allows us a more flexible and detailed analysis and greater opportunity to detect performance problems.

Commercial products for interactive parallelization are for example ForgeExplorer [1] and the KAP/Pro toolkit [7]. Both provide means to display dependence analysis information but do not allow extensive user interaction the way CAPO does.

An example for a commercial product for performance analysis is Vampir [16] which allows tracing and analysis of MPI codes. The CAPTools system provides an interface for generating VAMPIR traces and invoking VAMPIR from within CAPTools for performance analysis of message passing applications.

The Automated Instrumentation and Monitoring System (AIMS) [20] was developed at NASA Ames. It consists of number of components to facilitate the performance analysis of message passing programs but does not support OpenMP.

Paradyn [10], which is developed at the University of Madison, is a performance measurement tool for parallel and distributed programs. It includes dynamic code instrumentation and automatic searches for performance bottlenecks.

A general performance model for OpenMP is proposed in [11]. The model allows user functions and arbitrary code regions to be marked and performance measurements to be controlled using new OpenMP directives. Performance libraries based on this model have been developed for the TAU (Tuning and Analysis Utility) performance analysis framework ([11], [15]) and the EXPERT automatic event trace analyzer [18].

The advantage Paraver offers is the great flexibility in computing performance indices and statistics. This allows a broad exploration of metrics of interest and their corresponding influence on the parallelization choices.

7 Conclusions and Future Plans

We have interfaced the CAPO computer aided parallelization tool with the Paraver performance analysis system to support the program developer when parallelizing existing sequential codes. We have used the static analysis information available from the CAPO to instrument the code. By employing the statistical analysis module from Paraver, performance metrics for critical parts of the code can be calculated and displayed to the user. We have shown how the CAPO user interface displays loop analysis information and the performance statistics available from Paraver to point the user to parallelization problems.

As mentioned earlier, we are currently working on the design of a non-graphical interface to the Paraver analysis module to be used by CAPO directly, without user interaction. Many common problems could be detected by CAPO automatically and correlated to the loop analysis information and the knowledge data base. A high standard deviation, for example, in the useful thread time is often an indicator that an inner loop within a loop nest has been parallelized. In many cases CAPO might be able to determine that the reason for this is a conservative dependence in the outer loop. If the reason is missing information for a variable value, CAPO can prompt the user for an assertion. Another opportunity is the automatic detection of workload imbalance within a parallel loop which can be improved by changing the scheduling of the loop iterations from static to dynamic. We also plan to address scalability issues by automatically comparing traces for different numbers of threads.

It is needless to say that the performance problems in full-scale applications will not all be able to be solved automatically. At this point it is still not clear where to draw the line between automated and user guided responsibilities. By experimenting with our prototype implementation we hope to be able to identify more and more tasks that can be automated. Our approach is to successively develop an environment where the repetitive, cumbersome and error-prone tasks are left to the tools, allowing the user to focus on the tasks that require ingenuity. This will help to reduce the development time and/or increase the quality of the generated code.

Acknowledgements

The authors wish to thank the CAPTools team (C. Ierotheou, S. Johnson, P. Leggett, and others) at the University of Greenwich for their support on CAPTools. This work was supported by NASA contracts NAS 2-14303 and DTTS59-99-D-00437/A61812D with Computer Sciences Corporation/ AMTI, by the Spanish Ministry of Science and Technology, by the European Union FEDER program under contract TIC2001-0995-C02-01, and by the European Center for Parallelism of Barcelona (CEPBA).

References

1. Applied Parallel Research Inc., "ForgeExplorer," http://www.apri.com/.
2. D. Baily, T. Harris, W. Saphir, R. Van det Wijngaart, A. Woo, and M. Yarrow, "The NAS Parallel Benchmarks 2.0", RNR-95-020, NASA Ames Research Center, 1995.
3. Blume W., Eigenmann R., Faigin K., Grout J., Lee J., Lawrence T., Hoeflinger J., Padua D., Paek Y., Petersen P., Pottenger B., Rauchwerger L., Tu P., Weatherford S. "*Restructuring Programs for High-Speed Computers with Polaris, 1996 ICPP Workshop on Challenges for*
4. C.S. Ierotheou, S.P. Johnson, M. Cross, and P. Leggett, "Computer Aided Parallelisation Tools (CAPTools) – Conceptual Overview and Performance on the Parallelisation of Structured Mesh Codes," *Parallel Computing*, 22 (1996) 163-195. http://captools.gre.ac.uk/
5. H. Jin, M. Frumkin, and J. Yan, "The OpenMP Implementations of NAS Parallel Benchmarks and Its Performance", NAS Technical Report NAS-99-011, 1999.
6. H. Jin, M. Frumkin and J. Yan. "Automatic Generation of OpenMP Directives and Its Application to Computational Fluid Dynamics Codes," in Proceedings of Third International Symposium on High Performance Computing (ISHPC2000), Tokyo, Japan, October 16-18, 2000.
7. 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, Assure/Guide Reference Manual," 1997.
8. Liao, S., Diwan, A., Bosch, R. P., Ghuloum, A., Lam, M., "*SUIF Explorer: An interactive and Interprocedural Parallelizer*", 7th ACM SIGPLAN Symposium on Principles & Practice of Parallel Programming, Atlanta, Georgia, (1999), 37-48.
9. A. Maloney, S. Shende, "*Performance Technology for Complex Parallel and Distributed Systems*", Proc. 3rd Workshop on Distributed and Parallel Systems, DAPSYS 2000, (Eds. G. Kotsis, P. Kacusk), pp. 37-46, 2000.
10. B.P. Miller, M.D. Callaghan, J. M. Cargille, J. K. Hollingsworth, R. B. Irvin, K.L. Karavanic, K. Kunchithhapdam and T. Newhall, "*The Paradyn Parallel Performance Measurement Tools*", IEEE Computer 28, 11, pp.37-47 (1995).
11. B. Mohr, A. D. Malony, S. Shende, and F. Wolf, "*Design and Prototype of a Performance Tool Interface for OpenMP*", Internal Report FZJ-ZAM-IB-2001-09, Research Centre Juelich, Germany, 2001.
12. OMPItrace User's Guide, https://www.cepba.es.paraver/manuals.htm
13. OpenMP Fortran/C Application Program Interface, http://www.openmp.org/.
14. Paraver, http://www.cepba.upc.es/tools.paraver/
15. TAU: Tuning and Analysis Utilities, http://www.cs.uoregon.edu/research/paracomp/tau.
16. VAMPIR User's Guide, Pallas GmbH, http://www.pallas.de.

17. Wilson R.P, French R.S,Wilson C.S, Amarasinghe S.P, Anderson J.M, Tjiang S.W.K, Liao S, Tseng C., Hall M.W, Lam M. and Hennessy J., *"SUIF: An Infrastructure for Research on Parallelizing and Optimizing Compilers"* Computer Systems Laboratory, Stanford University, Stanford, CA.
18. Wolf, F., Mohr, B., "Automatic Performance Analysis of SMP Cluster Applications", Tech. Rep. IB 2001-05, Research Centre Juelich, Germany, 2001.
19. Wolfe, M. , *"Optimizing Supercompilers for Supercomputers"*. Research Monographs in Parallel and Distributed Computing, MIT Press, MA, 1989.
20. Yan, J. C., Schmidt, M. and Schulbach, C., *"The Automated Instrumentation and Monitoring System (AIMS) – Version 3.2 User's Guide"*, NAS Technical Report, NAS-97-001, January 1997.
21. Zima H P, Bast H -J, and Gerndt H M, *"SUPERB- A Tool for Semi-Automatic MIMD/SIMD Parallelisation"* Parallel Computing, 6, 1988.