Update on
Tools for Parallel Programming at the CNSF

Donna Bergmark

March, 1993

Not many CNSF users undertake the arduous task of parallelizing their programs. Of course, training, education, and available hardware will have a lot to do with changing this situation, but we feel that tools also have an important role to play. In 1989 we wrote:[1]

> At the present time, the general lack of parallel programming tools is an inhibitor to parallel programming at the Cornell National Supercomputer Facility (CNSF). The Technology Integration Group (TIG) is evaluating a number of tools designed to make parallel programming easier, including tools for source analysis, program development and execution analysis. The more effective tools will be "mainstreamed", i.e. turned over to users, integrated into workshops, and consulted on by staff.

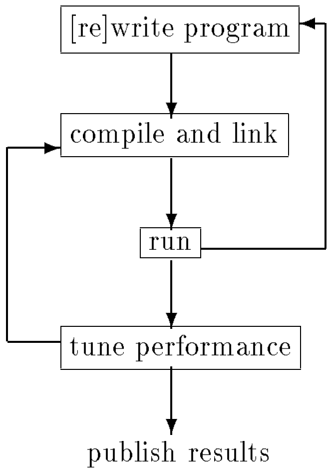
It is time to update that status report. Three years have passed since the original report on parallel tools was written. In that time we moved from a 6-processor IBM 3090/E to a 6-processor ES/9000, we added a "parallel cluster" of IBM RS/6000 machines, and we acquired two 64-cell KSR machines. So we have lots more parallel machinery. Do we have lots more parallel programming tools?

First, let us take a look at parallel programming tools as a concept. Creating a parallel program occurs in stages (Fig 1(a)): the source program *describes* the algorithm in some language, hopefully one with ability to specify concurrency at a comfortably high level. That description is transformed into one or more executable programs, a process usually called *compiling* and *linking*; the result is *run* on a parallel processor with some mechanism for observing program behavior; and the answers are checked for correctness (*testing*) and *debugging* occurs until they do appear to be correct. Each of these programming stages has tools associated with it (Fig 1b).

Tools are not strictly necessary, of course; the programmer can simply reason about the parallelism rather than get an analysis of the program dependences; debuggers are unnecessary if the program runs correctly right away, or if the bug is completely obvious; execution tracing is not necessary if performance is acceptable, or if it is clear where the bottleneck is. But in parallel programs, each of these activities can be very complex.

Unfortunately, the tools are also complex; they are at least as hard to use as writing the parallel program in the first place. Most parallel programming tools have steep learning curves. Why should one bother with the intellectual challenge of using them? The answer is that the more concerned one is with program *correctness* and *performance*, the more interested one should be in using the tools.

[1] For a copy of the 1989 report, please write to Donna Bergmark at bergmark@tc.cornell.edu.



(a) The Stages of Programming

source analyzers, restructurers, graphical program builders

languages, compilers, libraries

timers, instrumentation, debuggers

tracing, monitoring, profilers

(b) Tools for Parallel Programming

Figure 1: Stages of Parallel Programming

At the same time, tool developers must be absolutely dedicated to making their tools accurate and robust, and they must strive to give helpful information and not just obscure data. Unfortunately there are many counter-examples: debuggers that crash the system, source analyzers that generate incorrect programs; preprocessors that "pessimize" code; timers too imprecise for supercomputers; profilers that fail as soon as they are run with a parallel program. Improving this situation is our overall motivating concern as we collect, evaluate, and deploy parallel programming tools at the CNSF.

The major section of this paper is Tools for Parallel Programming, which is divided into 12 categories. Each category is summarized in pretty much the same way as in 1989, and then new status and prospects are discussed. The paper concludes with some comments on hybrid program development systems and the workstation environment. Appendix I contains a table of all the tools (divided into 12 categories). Appendix II lists acronyms, names, and institutions.

Summarized below, for each category of tool, is how the tool is used for parallel programming, what we have in-house, what we need, and what we are looking at. The careful reader will find that in some areas, such as parallel debuggers, impressive progress has been made; other areas, such as locality analysis, have lagged behind. Again, if you see an acronym that you don't understand, please refer to Appendix II.

Tools for Parallel Programming

1 *Languages*

When Fortran and C were invented, parallelism at the application level for scientific programming did not exist. Although many research languages have since been developed that encompass many aspects of parallel programming, scientific computing is still done largely in Fortran. Thus Fortran (and C) had to be extended by new language features in order to support parallelism. There has been a surprising amount of activity in parallel language development over the last three years, and we at the CNSF are reaping the benefit of some of this work. New features include syntax recognized directly by the compiler, automatic parallelization by the compiler, source-to-source translation (prior to compilation), directives and run-time libraries.

Current Status. The early PRPQ languages from IBM (PF and CF), and SCHEDULE from Argonne, are gone. In their place is a diversity of parallel languages to go along with the increased number of parallel processing platforms.

The cluster. PVM from Oak Ridge is currently the main choice for parallel programming on the cluster. The PVM library includes routines that support multiple copies of executables, all communicating via message passing, across several machines. While it is free and has few entry points, it has all the difficulties inherent in message-passing programming. Alternatives exist (Linda, Express) but they are not yet generally available on the cluster. We did look at P4 but decided not to pursue it further for various reasons.

KSR1. There is a single Fortran compiler on the KSR machines to handle both serial and parallel programs, called KSR Fortran. Parallelism is supported by directives and by library calls, and the flavor is reminiscent of the X3H5 standard for shared-memory parallelism. It is particularly rich in high-level directives. Also available on the KSR is **tcgmsg**, based on sockets. It implements message passing between simultaneously running Unix processes. PVM has been ported to the KSR, but it uses network message passing. We hope for a more efficient version in the future, which is based on communications via shared memory.

IBM mainframe. IBM has succeeded in reducing the number of its parallel Fortran dialects to one: VS FORTRAN, called APF in parallel AIX. APF follows the emerging X3H5 language standard and allows one to run multiple threads of execution in a single PAIX virtual machine.

Under Evaluation: Fortran 90 (F90) has become a standard, the X3H5 (PCF) standards effort is still in progress, and a new standards effort, High Performance Fortran, has come to the fore. HPF is based on F90 and it is oriented towards data parallel programming. We are tracking this closely since we feel it will become quite important for scalable parallel computing. In any case, we will see F90 gradually becoming available during the coming year. We will be evaluating Express, and probably Linda, for the cluster.

2 Graphical Program Development Tools

Graphical tools are far more important for parallel programming than for serial, because parallel programs have many blocks of sequential code that could run concurrently and there must be ways to depict interactions among them. Graphical tools allow the programmer to deal more effectively with the complexity involved in concurrent activity.

Current Status. There has been healthy growth in visual programming tools in the graphics world (AVS, Data Explorer and SGI Explorer) but the programs specified by them do not usually *really* run in parallel. These systems do, however, capture the user interface required for graphical program development.

Under Evaluation: While no known graphical tools presently exist for preparing parallel C or Fortran programs, HeNCE targets PVM systems and includes a tool called “build”, which lets you construct a program graph, from which a PVM program can be generated.

3 *Source Analyzers and Restructurers*

Source analysis (beyond that provided by the compiler) is crucial for parallel programs. We are not at the stage where compilers can detect data use conflicts, thus it is necessary for the user to analyze the source to make sure that correct directives are given to the compiler. Source analyzers assist in this process. The ideal source analyzer is interactive and informative.

The difference between an analyzer and a restructurer is that a source restructurer actually rearranges the sequential Fortran program to make it safe to parallelize. Also in this category are structural editors, which make sure the program conforms to the parallel language syntax. Automatic restructuring proceeds without help from the user; interactive restructuring is a give-and-take between human and machine. The ideal tool will automatically restructure code, and will accept partially parallelized code.

Current Status. Three years ago we relied on PF and PTOOL, both in CMS. They have now been replaced by Unix equivalents. PAT, then under evaluation, has been mainstreamed. This tool from Georgia Tech., good for all platforms, reorganizes the Fortran program under guidance from the user. It points out the dependences that prohibit parallelism, and then suggests ways to remove them. Once the user selects the method of choice, PAT goes ahead and changes the program's source, creating an updated file suitable for compilation (currently supported are IBM's APF, Cray's micro- and macro- tasking, and KSR tiling).

We have also acquired Baseline FORGE90 from Applied Parallel Research. Like PAT, it is interactive and fairly intelligible. This is an all-purpose Fortran analysis tool providing call graphs, use/def chains, instrumented code for timing purposes, and in-depth consistency checking. FORGE90 is for analysis only and does not emit parallel code.

The cluster. We have nothing as yet for the cluster of RS/6000's, though one could use a tool like PAT to determine where data need to be exchanged as messages. FORGE DMP has been ordered as a potential tool for creating parallel programs based on message-passing systems such as PVM. HeNCE will generate a PVM program, given a graph of the program's data flow and dependencies in a C program, but most people will find this too complicated. Also it does not currently work for Fortran programs on IBM RS/6000's so it is of limited use in the cluster.

KSR1. KAP is a preprocessor which analyzes loops and generates KSR Fortran parallel directives. Like PAT, it is helpful in telling the user which data dependences exist in each loop. Unlike PAT, it is not an interactive tool. Finally, for the user who wants to tile loops by hand (without KAP), the combination of PAT and FORGE can be particularly useful. ParaScope from Rice University is another restructurer that is under evaluation.

IBM mainframe. Both the VAST-2 preprocessor and the APF compiler will automatically parallelize some loops if there are no dependences. Both will print out extensive analysis of each loop. PAT is perfect for the IBM because it can interactively generate source input for the APF compiler from a serial Fortran program, as well as read in partially parallelized code. We have three or four tutorials online to help users use it.

Under Investigation: Other than ParaScope, no code restructurers are under evaluation. We looked at E/SP, but since it runs only on Apollos we had no further interest in it.

4 *Locality Analyzers*

Locality analysis is used in serial programming to reduce paging and cache misses, based on analysis of subscripts and strides. Locality analysis is more complex for parallel programs, because different execution units share the same data. Here, locality is needed to minimize storage conflicts, as well as to minimize paging and cache misses. Locality analysis is also helpful in determining which pieces of code could profitably be run in parallel.

Current Status. Despite the universally acknowledged need for such tools, there are very few locality analysis tools out there. We looked at a few prototypes during the past three years, but found nothing that was very usable. We still are on the lookout for more tools that can be used to help partition program data.

The cluster. Nothing is available at present, though there are some interesting system trace tools on the RS/6000 that require you to be "root" to run them.

KSR1. The `pmon` library (described below under execution analysis) records cache misses and generally provides information helpful in organizing data effectively. Some users have found it to be invaluable for organizing their data to achieve more locality. Other than that, locality on the KSR is achieved by following some rules of thumb and knowing the cache line sizes.

IBM mainframe. The output of the APF trace facility (see below) will give you an idea of how often common areas were copied back and forth.

Under Evaluation: There is a tool from Rice that analyzes locality with respect to hierarchical memories which might be useful. The MAPI/MAPA Facility from Argonne automatically instruments Fortran programs to record accesses to arrays; the generated trace output can be viewed graphically. HPF will give the programmer more control over locality, but help will be needed to analyze locality. APR has also announced a locality-related tool; we will look at this when it becomes available.

5 *Canned Parallel Code.*

Third party software that has already been parallelized gives users a jump-start on parallel processing. Parallel libraries that can be called from serial programs also help. In the three years since the last report, however, very few third party programs have been parallelized on any platform. A few exceptions are listed below.

Current Status. ESSL and LAPACK both contain parallel routines that can be used in applications. They are on the cluster, but they are not documented; furthermore, they are coded in terms of IMCS, a special communications package for optical fiber and switch. COLORS is a QCD program on the IBM mainframe, but it has not been well documented; the KSR has the richest set of codes including parallel make, a parallel random number generator, and some of computational chemistry packages: AMBER, CEDAR, X-PLOR, ECEPP, and DISCOVER. There is also a parallel matrix multiplier which people can use.

Under Evaluation: We will be receiving an early evaluation copy of Gaussian-92 parallelized for the KSR.

6 *Debuggers.*

Three years ago there was no hint of parallel debuggers anywhere. People used intermediate write statements in both serial and parallel programs. For serial programs, there was also the IBM supplied interactive debugging aid (called IAD). Fortunately, the intervening years have seen a lot of progress on the parallel debugger front, and some of the debuggers are beginning to be mature enough for real-world use.

Current Status. We have a parallel debugger on each one of our parallel platforms:

The cluster. On the cluster it is PVM programs we want to debug . **Xab**, which will display each PVM library call as it occurs, can be used as a limited debugger. You must link your program with a specially instrumented version of the PVM library to have this happen. At the present time, one cannot use **dbx**, the standard Unix debugger on PVM programs, because you cannot "attach" the debugger to executables at run time. However, you could use a proprietary new parallel debugger from IBM called **pdbx** to do this. It is also the case that **ndb**, the Express debugger, can be attached to running processes.

KSR1. The KSR has a production-quality parallel debugger called **udb**. This debugger comes in both line mode and as an X client, and is very reminiscent of **gdb** from the FSF. It is a very full-functioned debugger, including two different ways to handle parallel break-points and the ability to monitor user-defined subsets of threads.

IBM Mainframe. A new debugger called **pdd** (Parallel and Distributed Debugger) has recently completed staff test. This full-screen, X-based debugger has been under development at IBM for several years. Not all debugger features are yet implemented, but **pdd** has the advantage of showing the source program that is being debugged. You can use the mouse to go from one thread of execution to another, and the source code and local data display is scrolled accordingly.

Under Evaluation: Evaluation of the Express debugger has not yet begun.

7 *Execution Analyzers.*

Execution analyzers tell the user 'what happened when' during the execution of a program. Execution analysis is an activity very similar to debugging, but relates more to performance than to locating errors. Tools are needed to help the user understand what happened while running a parallel program because parallel execution is so much more complex than serial. Tracing tools are also useful for load balancing, and for selection of coarse vs. fine-grained parallelism. The tools range from static to animated displays, ascii to graphic output, and from summary to detailed. It should also be noted in passing that sequential profiling also has a role to play in parallel programming: hot-spot analysis almost always preceeds the reprogramming process. Standard Unix tools such as **prof** as well as Baseline FORGE90 's profiling can be used for this purpose and will not be discussed further.

Current Status. Three years ago we were looking at building our own program execution animation tool, called *Speedo.* We also had SCHEDULE's trace animation facility. These have fallen into disuse. We now have other execution analysis tools on all our machines, some of which came for free when we switched to Unix (**prof**, **gprof**, **time**, etc.). The remaining execution analysis tools are in early stages of development, but are still quite useful.

The cluster. PVM programmers will find that running the PVM daemon interactively (use the `-i` flag) is a good way to examine periodically the state of the various PVM program instances during execution. For a more complete trace, you can link your program with XAB's instrumented PVM libraries and run XAB's monitor process, **abmon**, concurrently with your program. You can view the resulting trace via the **xab** visualizer or you can translate the trace to PICL format and see its animation via ParaGraph.

KSR1. The KSR has an environment variable, which when set, will output a log of all thread events. The variable is PL_LOG and the monitoring is handled by the `presto` run-time library. More recently, the **pmon** facility has been added to the operating system; calling its routines can give a complete dump of system and application activities, such as cache misses. This library relies on built-in hardware support to summarize execution events on a per-thread basis.

IBM mainframe. The old PF trace facility and TAT have been replaced by the new APF Parallel Trace and TMVIEW. In order to trace a parallel program, the user merely adds the `-trace` option at execution time, or the `-f'etrace'` option at compile time. This provides a binary output file containing all the parallel events that occurred during execution of an APF program. TMVIEW is an X client which gives the user a graphical interpretation of the output and a more intuitive feel for what happened during execution. Or, one can use **apftrmap** which formats the trace file into readable ascii. In addition, locally-written timer routines plus a routine that returns the *id* of the running process can also be used to give non-graphic output describing program execution.

Under Investigation: We intend to have a closer look at Pablo, an execution analysis tool from the U. of Illinois (Dan Reed's group).

7.1 Execution Predictors

This is an emerging class of tools which tell the user 'what will happen when'. Like simulators, these sorts of tools are often table driven, where the table contains parameters describing a particular target system or machine. By simulating the instructions in the program, the execution predictor can estimate the performance of the program when run on the real machine. Such a tool is useful only if so much of the execution can be elided that the simulation takes less time than the actual run, or prior to arrival of the actual hardware.

8 Performance Monitoring.

Performance monitoring is slightly different from execution analysis, in that execution analysis is devoted more to *what* happened, whereas performance monitoring indicates *efficiency* and *speed* of what happened. Timers, of course, are key. Profilers are also useful. The FORGE profiler, for example, works on all our platforms (but currently on serial programs only) and gives the elapsed user CPU time of each subroutine and each loop within the subroutine. The next version (8.7) will provide a similar level of profiling for parallel programs.

Current Status. A number of performance monitoring tools are available.

The cluster. ParaGraph provides some summary charts that give you a clue to the performance of your program. One of the more useful charts, the `gantt` chart, shows for each processor the total idle time, system time, and user time. Other than that, you are pretty much left to calling timers judiciously. The most precise timer on the cluster of RS/6000's is a one-hundredth second timer (`mclock`).

KSR1. The **pmon** library on the KSR counts up to 18 different kinds of system events and is invaluable for performance monitoring. The system events include times, cache misses, and context switches. **Prof** and **gprof** works for multi–threaded programs so they actually are useful for monitoring the performance of a parallel program, as well.

IBM mainframe. We produced some timers locally that can be used with parallel programs, and these are currently available in the `faux` library. You can get an idea of the concurrency actually achieved by your parallel program by using TMView to look at a trace.

Under Investigation: Pablo (mentioned above) might have some tools for analyzing performance of the program as well as watching parallel events as they occur. As mentioned above, we will be receiving a parallel program profiler with Forge90 dmp.

9 Hazard Detection.

Given the probe effect of parallel debuggers, it is especially important to have tools that point out potential hazards (race conditions) in parallel programs. Of potential interest are tools which, presented with a set of data and a program to run on that data, will list anomalies if any exist. These tools can be used to guarantee that you have no race conditions. They do not work for cases where data are not reproducible, such as real-time applications.

Current Status. The data dependence tools (pat, kap) point out data dependences. This is a limited form of hazard detection.

Under Evaluation: ParaScope

10 Interprocedural Analysis (IPA).

For parallel programs, interprocedural analysis is extremely important if one wishes to achieve optimal coarse-grained parallelism (what routines can run in parallel?) as well as optimal fine-grained parallelism (can loop iterations that call subroutines be executed in parallel?). It is essential for modern parallel languages such as HPF.

Current Status. Pat and Forge90 both do IPA, depending on how much of the source program you give them.

KSR-1 KAP can inline code in order to perform IPA and thus expose more parallelism.

IBM mainframe. Vast-2 is supposed to do in-line expansion, which is a limited solution to the problem.

Under Evaluation: ParaScope.

11 Program Database.

Given the importance of interprocedural analysis combined with the extra burdens placed on the compiler of a parallel language, one needs a sophisticated database system in order to have incremental compilation. Program databases are also becoming increasingly important in the area of instrumentation and performance analysis.

Current Status. Three years ago the best we could offer our users for a program database was the ICA function for the IBM serial compilers. We were looking at PTRAN, but that never really got past the prototype stage. Now we have FORGE90 which is a complete database for programs. You can include as much or as little of your source as you want; you can include timing information if you want; FORGE90 performs incremental analysis. It does not do incremental compilation, since it is not a compiler.

Under Evaluation: ParaScope will support integrated program databases, and it is also a compiler.

12 *Portability Tools.*

Portability is achieved for serial programs primarily by coding in Fortran and using standard libraries. Things are nowhere near as rational in the world of parallel programming, where each vendor has its own dialect, and a great coding chasm divides shared memory systems from distributed memory systems. A common parallel Fortran language is still needed.

Current Status: The need for portability has not gone away in the last three years. The major change we have seen is that F90 has become the Fortran standard, the HPF has proposed a definition for a data parallel F90-based language, and the ANSI X3H5 has been meeting to establish a standard parallel language for shared memory machines. Coding in PVM and plain f77 Fortran is currently the most portable way we have of writing parallel programs, because both can be found on many computers.

FORGE90 is a portability tool as well, in that you can generate a message-passing program from your serial (Fortran 77) program. This message-passing program can then be linked with any number of standard message-passing libraries, such as PVM, to run on a specific system. This portability depends, however, on your having purchased at least one copy of FORGE90 DMP and on each potential target having licensed APR run-time library, and having another "real" message-passing system to support that library.

Hybrid Program Development Systems

We do not yet have any hybrid static/execution analysis tools, but they show promise. The idea is to share information gleaned from source analysis of the program (which is static) with the run-time system, and vice versa.

For example, the adversary scheduling system proposed for the ParaScope programming environment would pick out a "dangerous path" through one's program based on possible conflicts and race conditions as determined by source analysis; then the program would be forced to take this execution path. If all works well, the program should execute correctly under any other schedule as well.

Examples of information flowing from execution back into compilation included PTRAN and PFDE, which collected run-time statistics and stored them in the program database. These statistics would be available to the compiler when it next compiled part of the program. This could help the compiler optimize the program. These were prototype systems, however, and did not survive the evaluation period. Since then, nothing else has come along to replace them. Compilers still remain quite ignorant of run-time statistics. An exception to this is FORGE90 (which is not really a compiler); it can take run-time statistics into account to determine which loop to parallelize next.

The Workstation Environment

The growing interrelationship between workstations and parallel programming tools has been explosive. Three years ago we wrote:

> The role of the workstation in parallel programming cannot be overlooked. The more graphical the programming tools, the more likely it is that a scientific workstation is used as the display device. On the other hand, source and locality analysis are sufficiently compute-intensive that it may be desirable to carry out this portion of the process on the supercomputer itself. The compiler will almost certainly reside on the supercomputer. Thus workstations and the supercomputer will be used in conjunction with one another.

This statement remains true today, except that the single supercomputer has been joined by multicomputers and massively parallel machines. The parallel tools reside all over the network, with some pieces residing on the parallel platform and other pieces on a workstation or the user's own system. The growing ubiquity of X servers has allowed this. Three years ago,

> ... all *production* CNSF tools are mainframe-based (using 3270 graphics terminals for display) while most of the tools being *evaluated* are workstation-based. TIG is gambling on a large increase in the numbers of workstations on scientists' desks.

Our gamble paid off. Plenty of users are able to use graphical tools; a much faster network has been put in place across the nation; X has become a default standard practically everywhere. 3270 graphics terminals have disappeared. Another expectation we had three years ago was that

> Unix will be the program development system of choice. We expect that the emergence of good programming tools at the CNSF will be directly correlated with the emergence of AIX and increased ubiquity of workstations. Fortunately, the computer science establishment has given us a head start on preparing for these eventualities.

One issue that seems to have been settled by time and experience is the toolkit vs. environment approach. The difference is one of degree and interchangeability; toolkits are mix and match, environments are complete and contained and stand alone. We investigated complete environments such as PTRAN and $\mathbf{R}^n$, and individual tools such as PAT. Combinations of tools seem to work best. The Scientist's Workbench became popular because it allows the user to mix and match tools within a consistent environment. We have conceived of a "Parallel Programming Toolkit" based on the SWB, and have produced one example (the PVM Workbench), but have not yet had the resources to pursue this further.

Another issue is the inconvenience associated with preparing one's program on one computer (the workstation) to run on another computer (the parallel processor). Many find this across-machine program preparation inconvenient. One motivation for the Scientist's Workbench project is to make this process more transparent to the user. For example, one can hope that with better tools, the programmer will never knowingly use "ftp" in the program preparation process.

When DCE becomes a reality, this issue may be settled once and for all. In the meantime, a client/server approach to parallel programming tools is indicated. While client/server computing is still a new idea to many people, and systems staffs are reluctant to set up "server" userids, we plan to work toward setting up a parallel programming tools server that can be transparently accessed from all our parallel platforms, and perhaps indeed from elsewhere within the MetaCenter. Optionally, a parallel programming workbench could be used to organize these tools.

Summary

The major advance made in the last three years has been in debugging for parallel programs. From being a glaring omission from the list of tools for parallel programming, it has gone to availability on all our parallel platforms (at least in a prototype stage). Why the progress? Because parallel debugging was an interesting problem, and many research groups worked on it. Along the way, tool developers began picking up experience with graphical techniques and user interfaces. The coincident explosion in the popularity of X11 meant the development of many graphical tools, not just for debugging. Who knows: perhaps in another three years, even compilers will be graphics based!

It is still true that for every ten tools we encounter, we probably look at five rather closely, bring in three for evaluation, and put one into production. As part of this process, we still have the problem that once the tool is moved into production, how do we train users to use it? How do we get consultants to play with the tools and become familiar with them? How do we motivate users to use the tools, which are not yet as user-friendly as we would like? The answers to these questions still need to be discussed within the CNSF at large, but clear progress here can be seen.

Parallel programming tools *have* become a standard part of the curriculum, and more consultants *are* learning them. But using them is still a major intellectual challenge. Properly interpreting the trace of a parallel execution is still just about as difficult as doing the parallel program in the first place. The solution to this problem lies in the hands of the trainers and educators as much as the tool developers. More feedback from users to developers will be the key to a better programming environment in the years to come.

Acknowledgements

Thanks to Marcia Pottle for rounding up all the information on KSR parallel tools, to Ros Leibensburger for helpful suggestions to improve the clarity of the presentation, and to David Presberg and Steven Hotovy for recommendations on content. Especial thanks to all our users who have used and been abused by the parallel programming tools at the CNSF and provided their feedback.

The LaTeX source for this document is on
eagle:f7f/PAR-TOOLS/parallel.tools and on
snake:/nfs/snake/home/bergmark/reports/PAR-TOOLS/parallel.tools

Appendix I: Tools - Status and Prospects

Type of Tool	In Production and Available to Users	Here – Under Evaluation	Not yet here, but Will Be	Possible in the Future
Languages	APF PVM KSR Fortran	EXPRESS tcgmsg	F90 versions of Fortran	HPF Linda
Graphical Program Develop. Tools			HeNCE	
Source Analyzers and Restructurers	APF, PAT KAP Baseline FORGE	FORGE DMP		ParaScope VAST-2 (new)
Locality Analyzers				MAPI/MAPA
Canned parallel code	LAPACK (parallel BLAS), AMBER		ESSL	IMSL,NAG,ANSYS, FIDAP
Parallel Debuggers	udb	pdd pdbx		adversary scheduling instant replay
Execution Analysers non-graphic	iproc(), abmon, APF Trace, pmon, timers, apftrmap	FORGE DMP's profiler		
graphical -static	TMView		Pablo	graphical interface to pmon output
-dynamic	ParaGraph Xab			
Performance Monitoring	APF trace and TMView, prof, gprof, pmon, timers	FORGE DMP's profiler		Pablo's instrumentation package
Hazard Detection	PAT KAP			ParaScope's debugger
Interprocedural Analysis	FORGE PAT			ParaScope
Program Database	FORGE90			ParaScope
Portable Coding Tools	FORGE90 PVM			PCF Fortran HPF

Appendix II: Acronyms, Names, and Institutions

abmon	Part of the Xab package; captures the trace records from an instrumented PVM library and sends them to standard output or to a file.
AIX	IBM's family of Unix operating systems, based on the OSF standard.
APF	The parallel Fortran compiler for IBM's PAIX operating system, which runs on ES/9000 supercomputers. Accepts the X3H5 emerging standard Fortran dialect.
APR	Applied Parallel Research, the vendor of the FORGE programming product.
BLAS	Basic Linear Algebra Subroutines are the basic building blocks of many linear algebra subroutine packages, including LAPACK. BLAS-3 is for matrix–matrix manipulation and is especially well-suited for parallel machines.
BUILD	Generated SCHEDULE programs from user-drawn flow charts, put out by Argonne, used SunViews. Notable for being one of the first two graphically based parallel programming tools. Has envolved into one of the components of HeNCE.
CF	Clustered FORTRAN, designed to support coupled 3090 mainframes. Now obsolete.
CIO	A communications library that supports a fast optical fiber switch from IBM. Replaces IMCS in AIX 3.2 on the RS/6000, but is not as fast.
CNSF	Cornell National Supercomputer Facility, part of the Cornell Theory Center; where the author works.
CSRD	Center for Supercomputing Research and Development, at U. of illinois.
DCE	Distributed Computing Environment, an emerging OSF standard that includes Kerberos, the Andrew File System, and other features for network-based computing.
E/SP	An environment for parallel processing, developed by the Concurrent Computer Corporation in 1990, based partially on Jim Browne's work at U. Texas, Austin. A unique feature of this environment was a graphical interface for displaying data dependences.
ESSL	A scientific subroutine library available from IBM, notable for including a set of 6 or 8 parallelized subroutines. These are not yet available in PAIX, however.
EXPRESS	A message passing communications library from ParaSoft, Inc. Notable for inclusion of topological communication routines as developed originally for the Crystalline operating system at Cal Tech. Now for heterogeneous networks of computers. Also includes tools for debugging and tracing.

Appendix II: Acronyms, Names, and Institutions

F90	Fortran 90, the new standard for the Fortran programming language. Compilers are slowly emerging for this replacement to the FORTRAN 77 standard.
FORGE90	FORGE90 is an integrated collection of interactive Fortran programming tools from Applied Parallel Research. Baseline FORGE90 is designed for analysis of Fortran 77 source. Analyses include COMMON grids, variable tracing, call graphs, use-def chains, and instrumentation for loop timings.
FORGE90 DMP	The Distributed Memory Parallelizer (formerly called FORGE MIMDizer) is an "add-on" to the FORGE90 package which interactively with the user parallelizes a Fortran program for distributed memory systems. It generates an equivalent SPMD Fortran program studded with calls to APR's message passing library which in turn is implemented in one of PVM Express, P4, etc.
FSF	The Free Software Foundation is a self-supporting organization that produces high-quality software freely available to the public; this software is popularly called "Gnu software", which stands for "GNU's not Unix".
FTP	File Transfer Protocol. Also the command that does it.
HeNCE	A package of tools that support the writing of PVM programs, available from Oak Ridge National Laboratory.
HPF	High Performance Fortran, an extension of F90 that includes data parallel directives. The draft standard is expected in January 1993. Orthogonal to the emerging X3H5 standard.
HPFF	High Performance Fortran Forum, the body of people that met over the course of 1992 to propose the draft definition of HPF.
IAD	Interactive Debug aid available from IBM to go along with their VS Fortran compiler. Gives execution time profiles and supports interactive debugging. Not currently available with Parallel FORTRAN or Clustered FORTRAN.
ICA	Inter-compilation analysis, a feature for VS Fortran, allows one to save procedure headings and common declarations in a database, which could then be used at compile time to perform IPA. A set of functions was provided to manipulate this database.
IMCS	Inter-Machine Communication System, available from IBM, to support very fast communications over optical fibers. RS/6000 based.
IPA	Interprocedural Analysis. What aliases exist; constant Propagation; what variables are used, and redefined by subroutine calls. A technique that is *necessary* for optimal parallel applications.
KAP	A Fortran pre-processor from Kuck and Associates that inserts directives into a Fortran program to cause a parallelizing compiler to generate code for shared memory parallelism. We use KSR KAP, designed for the KSR1 machines.
KSR	Kendall Square Research, maker of scalablscalable parallel computers.

Appendix II: Acronyms, Names, and Institutions

LAPACK	A linear algebra package from Argonne designed especially for parallel machines.
Linda	A Fortran or C dialect for parallel computing, from David Gelernter of Yale University, marketed by Scientific Computing Associates. It is based on a novel parallel programming paradigm, based on tuple space and pattern matching.
MetaCenter	An initiative currently being undertaken by the four NSF supercomputer centers to define and implement a national machine room via high-speed cross-country communication links filled with a diversity of supercomputers and accessed by common programming methods and user interfaces.
Pablo	An execution time analysis system by Dan Reed of the University of Ill., the outgrowth of Tapestry and Picasso instrumentation systems. It supports program tracing and trace viewing.
PAIX	Parallel AIX, from IBM, that runs on ES/9000 parallel processors. It is a multi-threaded Unix operating system that supports parallel exeuction of individual jobs as well as parallel execution of the OS itself.
ParaGraph	A very well-known, highly regarded, very colorful, trace visualization system written by Michael Heath and Jennifer Estridge while at the Oak Ridge National Laboratory. The input to this visualizer is PICL traces.
Parallel Trace	An execution time feature supported by the old IBM Parallel FORTRAN compiler in CMS. It produced a trace of parallel events and was invoked by the PAR(TRACE option. This has now been replaced by the APF trace feature.
ParaScope	The successor to $\mathbf{R}^n$ for parallel programming. Includes **PED**, a structured program editor, and an **execution monitor** for debugging and performance monitoring. This software is under continuous development by Ken Kennedy and his group at Rice University. It also includes the beginnings of a Fortran D compiler.
PAT/START	A source analysis tool created by Bill Appelbe and Charlie McDowell, a former student of Appelbe's. Consists of a Static Analysis Anomaly Reporting Tool (START) and an interactive Parallelization Assistant (PAT). Plans are for this tool to target PF and PFC Fortran. One superior feature of PAT is that it will assist in parallelizing already partially parallelized code. Appelbe is at Georgia Tech, McDowell at the University of Santa Cruz.
PCF Fortran	Parallel Computing Forum Fortran is a standard parallel Fortran dialect that the PCF has been attempting to define for about three years now. Currently in X3H5 committee; if adopted, will standardize parallel FORTRAN across all vendors of shared memory parallel machines.

Appendix II: Acronyms, Names, and Institutions

pdbx A parallel debugger from IBM's HPSS Laboratory, under early beta test at the CNSF. Based on dbx.

pdd Parallel Distributed Debugger, a parallel debugger from IBM's Research organization. Notable for being a unified view of a parallel program, rather than just an n-way dbx debugger.

PF Parallel FORTRAN from IBM, based on an older version of VS FORTRAN. Also called the PRPQ language.

PFDE Parallel FORTRAN Development Environment was an X11-based system developed by Dan Pease, a Computer Science professor at the U. of Syracuse, under IBM funding. There was a version at Cornell, which ran on an RT in Caldwell. But it was buggy, and TIG was never given access to it.

PICL Portable Instrumented Communications Library, yet another message passing subroutine library, this one being one of the earliest. Most notable for producing a trace of the parallel execution, which can be graphically displayed by ParaGraph. Originally meant for the Intel iPSC hypercube computers.

PRPQ Please Request Price Quotation. An IBM designation for software they want to sell only to certain selected customers on a limited support basis.

PTOOL Installed for several years on the CornellF CMS machine, this tool analyzed each loop in a Fortran program to determine whether the loop could be parallelized and if not, to show which data dependencies would inhibit the parallelization. It had limited interprocedural analysis, and required a 3270-type terminal. It used a service virtual machine running **PSERVE**, a program analyzer. This has been functionally replaced by PAT, but PSERVE is still used by the ParaScope environment.

PTRAN A comprehensive environment for producing parallel programs developed at Yorktown Heights by Fran Allen and crew. Part of the system was evaluated by TIG but it never passed prototype stage. Being programmed in PL/I, it had to be dropped entirely when we moved to Unix. It was notable for using a very fast, large, and efficiently programmed database. It generated Parallel Fortran as well as code for the RP3 and ACE-1.

PVM Parallel Virtual Machine, another message passing library from Oak Ridge National Laboratory. Notable for its wide use at the current time and its relative simplicity. It runs over TCP/IP and supports a heterogeneous network of computers, usually workstations. Supports both Fortran 77 and C source programs.

$\mathbf{R^n}$ A programming environment for vectorizing Fortran code, produced by Ken Kennedy and his graduate students at Rice University. This has entirely been superseded by ParaScope, which runs under X11 and produces PF code.

Appendix II: Acronyms, Names, and Institutions

Schedule	A run-time library from Argonne National Lab authored by Dongarra and Sorenson to promote portable parallel programming in Fortran. You split your program up into subroutines, tell Schedule what partial order they must run in, and Schedule handles the rest. A VM/CMS implementation in PF was done locally at Cornell. Comes with two other subsystems: *trace facility* - animated replay of subroutines as they ran in execution. Currently available on elgin (a Sun). *build facility* - user graphically depicts order in which his subroutines must execute, and a proper Schedule program is built. This system has been pretty much replaced by HeNCE.
Speedo	A demonstration project in TIG based on the X Window System to show how programs executing on the supercomputer can be monitored on the workstation. It would have led to an instrumentation system like Pablo.
SPMD	Single Program, Multiple Data. A programming paradigm in which the same program runs in each process or processor, but on different data. SPMD programs that communicate in some "coordinated" topologically defined fashion, only with neighbors, for example, are potentially very scalable.
Tapestry	A Sun-workstation based system being developed by Dan Reed at CSRD to display use of memory for data storage.
TAT	Parallel FORTRAN Trace Analysis Tool, written at IBM St. Theresa, to display contents of voluminous trace output more simply in graphic form. Required a GDDM device for display, or GDDMXI and an X server. This program evolved directly into TMVIEW.
udb	Universal Debugger, the dbx-based debugger for the KSR1. Is rich in features and has an X11 interface as well as the more usual line-mode interface. Very similar to pdbx.
Xab	An X-based tracing system for PVM. It includes an instrumented version of the PVM libraries, a PVM-based program to capture the trace records at run-time, and an X client for graphically displaying those trace records. It also includes a handy awk script for translating the trace records into PICL format.
X11	A byte stream protocol to support two-dimensional graphics and communications between a client and a server. Available from your workstation vendor or MIT. (Public domain software).
X3H5	An Ansi standards committee that is turning Parallel Computing Forum Fortran into a standard Fortran dialect for shared memory parallel computers.