Design and Monitoring Systems for Parallel Programming

A.J.Katalov, V.J.Katalov, V.K.Nikolaev

Abstract

In this paper we consider computer-based systems for designing, debugging, tuning and optimizating parallel programs. The development of such systems is complicated and labour-intensive. Despite this, in the last few years many interesting projects have been developed which can be effectively used to design debug programs for parallel architectures. We analyse the current state in this area and the various approaches are compared.

1. Introduction

There are now a range of powerful parallel computing systems commercially available, whose raw price-performance characteristics are better than conventional supercomputers. Nevertheless it is impossible to speak about wide use of parallel systems. One of the reasons is the difficulties inherent in designing, debugging and optimising a new programs and parallelizing the existing sequential programs for parallel architectures.

Many attempts have recently been undertaken to develop easy-to-use toolkits making it possible to to do this effectively, however none have completely solved the problem. We analyse the current state in this area.We consider designing a system, including the parallelizing compilers, and monitoring, tuning and optimising systems.

The considered systems conform to the one of the following architectures:

- Workstation clusters.
- Multiprocessor systems using i860 or transputer processors.
- Multiprocessor supercomputers: Sequent Symmetry, Alliant FX/8, BBN Butterfly, Ardent Titan and so on.

Practically all considered systems run under the UNIX operating system.

This paper is organised as follows. Section II contains a comparison of various design systems for parallel architectures. The characteristics of various systems of optimisation, tuning and analysis of behaviour of the parallel programs are compared in section III. Section IV contains brief conclusions. Appendix A presents a brief list of design systems and parallelisation compilers. Appendix B contains a brief list of debuggers and monitoring, optimisation and tuning systems.

2. Design systems

Design systems are divided into two groups: systems for transformation the sequential programs in parallel - parallelizing compilers, and systems which design new parallel programs, probably using some sequential codes. The difference between these approachesis that transformation systems transformthe existing sequential program into a parallel one, versus the user himself choosing a set of processes the topology of any channels used, and communication protocols between processes. Then user determines the programs which will be executed on each process, thus, as a rule, there is an opportunity to use old sequential program code.

2.1. Parallelizing compilers - the analytical approach

Parallelizing compilers, which can be characterised as analytical tools, use a sequential program, which can consist of several modules, to create the parallel program and data distribution. In some cases, a toolkit is involved as well. The code transformations are usually carried out in several steps and as a rule the target architecture is fixed. One of the most advanced designs is the system CAPTools [5], which has incorporated many recent advances.

Parallelizing compilers consist of the following main components:

- A detailed control and dependence analysis.

The implementation of such toolkit is very labour consuming process, and the underlying algorithms require significant computing resources. It involves an analysis of dependencies between various iterations of cycles is executed, and is NP-complete, so it is unlikely that the situation will change in the near future. On the other hand in practically all works the same small group algorithms is mentioned: Banerjee and Omega-test. However in the theory of linear programming there are plenty of other algorithms and methods [54] and it would be useful to check up on some of them.

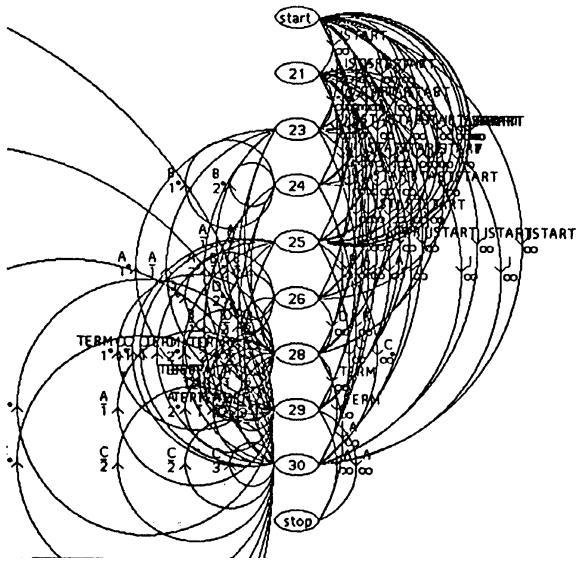

Figure 1

Practically of all the authors are recognize that it is impossible to analyse dependencies without attraction of the user. The development of the project ParaScope [9] is very indicative in this respect: at first its parallelizing compiler PFC worked as a "press the button and receive result" system, but after numerous user requests it was transformed into interactive system. Usually a program is analysed iteratively; after the generation of a dependency graph the user can enter or cancel restrictions, about which he probably has substantial additional knowledge. However, the size of such dependency graphs makes this very difficult (see Figure 1) – many authors make this point. Here, probably, the choice of a data representation method which is more convenient for the user could improve the situation. For example, the dependencies between loop iterations could be represented not as a graph but as a matrix, thus a hierarchical, multilevel representation of the data can be used. If everyone agrees that the participation of the user at this stage is vital, it seems obvious that the search for such data representation methods, which the user can process is essential. lets it is important to note that for successful work the user needs substantial knowledge of the system which he transforms.

- User definition of the parallelisation strategy.

Practically all systems use the same concept: SPMD (single program multiple data), in which each processor of the computer network executes the same program, but operates on its local? data only. As a rule, the only type of data distribution used is the decomposition of a mesh, and the mesh is usually regular. Certainly, such approach allows processing successfully with many scientific applications, but all the same it is an essential limitation .

- Automatic adaptation of the scalar code to equivalent parallel code.

The system usually has some set of predefined code transformations and searches for fragments of sequential program which can be parallelized using them. The system effectiveness therefore depends directly on size of this set. This set has increased over the last few years for example, in the works of J.Xue [21]. It is clear that for the successful design of such transforms it is necessary to recruit specialists in the theory of numbers and linear programming.

On the other hand such approach has a fundamental contradiction, which is connected to the dependencies which the parallelisation of sequential programs frequently brings into the programs, which were absent in the initial algorithm. Therefore during parallelisation frequently have to work around this kind of problem. There are a lot of caseswhere the development of the parallel program can be executed more effectively if it is carried out using a parallel algorithm, than from the transformation of a sequential program into parallel, because the sequential program has deformed initial character of these dependencies.

An interesting message is contained in [5]: after the analysis of dependencies was executed in algorithm LSOR, it became clear that the program cannot be transformed into parallel form. However this analysis suggested a change initial algorithm to the designers, with the result that an effective parallelisation became possible. The new algorithm converges more slowly, but this loss is overridden by an effective parallel realization. It seems, that this case further can become a common rule: with the help of system of the analysis of dependencies it is possible to transform the sequential programs in parallel, changing the algorithms in the process.

- Automatic insertion of «parallel» control and dataflow structures.

Having to insert conditions into the program follows from the SPMD architecture. These conditions ensure the correctness of access to the data: each computing unit must carry out transformations only on the data that is nominated for it. Usually it is done with the help of a mechanism of masking of the operators. Secondly, when there are data dependencies between processes exist, the operators ensure the transfer of the data between processors and synchronization of processes.

One of the well known methods which increases the performance of a distributed computer network is the insertion of the buffers for the storage of several copies of the input and output data– as shown in Figure 2. This buffering allows the system firstly to simultaneously carry out calculations and transfer of data which are to some extent parallel, and secondly provides each computing unit with a stock of the data, which reduces the latency of the data. Experts in transputer programming well know that such buffers usage and the use of high priority communication processes can decisively improve the productivity of a system. Therefore we think that parallel transformations of the sequential programs should use such constructions.

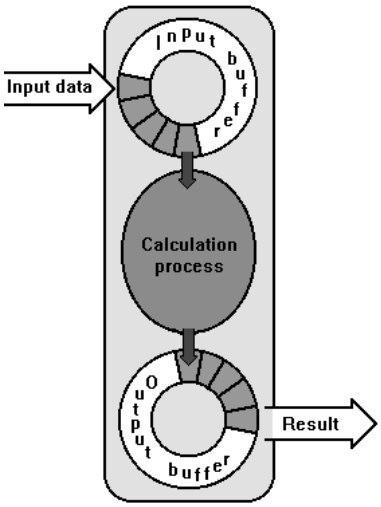

Figure 2.

Despite the difficulties some projects have approached a commercial standard for their products. So for example with the help of the CAPTool system[5] a software package having 4500 lines of Fortran 77 containing 36 procedures was successfully parallelized. In doing so the speedup on an 8 processor i860 system was found to be 5.8.

2.2. Design system - the synthetic approach

This approach enables the user to choose a graph of processes and communication channels connecting processes among themselves, to assign the code for each process, and to examine the structure of dependencies between the programs which will be executed on different nodes. Thus as well as in the analytical approach there is an opportunity to use existing sequential code; the parallel program is projected from top to bottom to some depth, from which available sequential modules can be used.

The design systems consist of the following components:

- The graphic editor.

With the help of the graphic editor the user can set the topology of a graph of processes. There are two approaches: in the first one the edges of a graph are the communication channels, and in the second they are dependencies on the data and control. Some systems include a graphic language, elements of which are the images of processes. Some systems allow the user to select a fragment of the text to show the corresponding dependency graph. In parallel programming as a rule the cycle of debugging is much longer than in for sequential systems. In addition the behaviour of a parallel system does not become clear at once, therefore we think that such immediate reaction of design system to changes made by the user is a very attractive feature of such an approach.

- System of transformation of sequential codes in parallel.

The transformations which are to be executed with the sequential programs for transformation them in parallel, in this case are a little bit easier than in case of a parallelisation of the sequential programs. Here it is required to ensure reception of the input data and sending the results. Usually the fragments of a code are added to the text of the sequential program which provide performance of these functions. Such fragments often are named as wrappers. Some systems have sets of such wrappers for the several programming languages.

- System of search of optimum mapping the processes onto processors.

The systems having such tool usually also have a subsystem for configuring the computer network. T mapping can be done manually or automatically. In manual mode the programmer clicks on process in a graph of processes and nominates a processor. In automatic mode the system builds a tree describing of distribution of processes on processors, with the aim that (as well as can be determined) the computing load will be uniform, and the volume of communications will be minimal. In spite of the fact that generally this task is NP-complete, some systems find the suboptimum solution in a reasonable time.

3. Parallel debuggers and monitoring systems

The works at this theme are divided in two large groups: debuggers, which work in a mode on-line and systems, which work in postmortem mode: the sensors assembling a trace of performance of the program are built into the program which later is reproduced on host system. Actually the tendency of development of these systems is that it is difficult to carry out a precise line between them - the development can result in mutual penetration of methods and ideas, and creation of toolkit allowing to execute functions of both

approaches eventually can be elaborated. As of now the difference between these approaches lies in the fact that on-line the debuggers are focused in providing of access to a low level information: a condition of the registers, memory, observation of the moments of creation of new processes and so on., and the works of the second group provide processing of the information on more high level.

3.1. On-line debuggers

It is well known, what even the insignificant intrusion into the parallel program can result in radical change of character of its behaviour. This fact is one of the basic difficulties in development and use of debuggers of such type. The reproducibility of results is the necessary requirement to on-line to a debugger, and the implementation of this requirement is complicated problem. LeBlank has offered one of possible ways of overcoming of this difficulty. He has suggested to divide debugging into two passes. On the first pass the sensors are inserted into the program which write down sequence of accesses of all processes to shared resources - this information has rather small volume. At the subsequent program executions, during which the user receives access to the information are carried out so that the sequence of accesses to shared resources in accuracy repeated the trace, written down on the first pass. Originally this mechanism was applied to systems with shared memory, later it was adapted to systems with the distributed memory.

3.2. Monitoring, tuning and optimisation systems

On this theme there are a lot of works. Most widespread is the scheme, at which the sensors are inserted in the program to pick up the information, specified by the user. After program finished this information kept in local memory of each processor was unloaded to the host-machine. Then in a off-line mode this information is displayed in a lot of views.

This approach has the following advantages:

1. It is easier to implement.
2. It brings into the program less intrusions, as there is no necessity to occupy channels for a transmission of the assembled information during program execution.
3. Is more economic in the sense that the user occupies the expensive equipment in an monopoly mode less.
4. Supposes a batch mode at the collection of the information - it is possible to collect traces at once of several variants.
5. Allows to compare performance traces of several variants of the same program.

The tendencies of development of this approach are those:

- Automation of process of embedding a sensor. The user can specify group events of interest (input/output in procedures, sending/receiving of the data, change of a semaphores conditions and so on.) and the system itself will execute transformations of the program text necessary for the collection of the appropriate information.
- Representation to the user of the large set of the points of view, that is, the capability of system to select and to display the information according to a part, interesting for the user. There are systems capable to show the following views: all temporary diagram of performance of the program; the selected group of processes and exchanges, connected to them, with the more detailed information; condition of memory and registers in the points, specified by user; time distributions of counting, waiting and transmission of the data on the specified sites of program execution and so on.

- Maintenance of continuous connection with the program text. The user has an opportunity, studying some point of view, to establish connection between displayed events and appropriate sites of the text of the program. Such variants of this connection are possible, at which the changes in the program, are being carried out in reply to the certain actions of the user with the displayed information. For example, it is possible to enable the user to insert the sensors into the program in places, which he chooses on the temporary diagram and so on.

- Openness of the system. An input format of the description of a trace execution is developing as flexible, even programmable. It enables the user to connect the own points of view, to form combinations of the various images to extract from a trace the information, being required for him.

- Intellectualization of toolkit. In several works the systems are offered which not simply display the information, but analyze it. For example, the search of the reasons of low efficiency of the program is being executed and even the ways of its increase are offered. There is a system, which are capable to search the reasons of deadlocks, to determine the processes, which too long are waiting for the messages and so on.

In this direction the further development is possible. For example, the debugging of the parallel programs frequently is carried out in conditions, when the results of performance of each stage of the program beforehand are known. In such cases a tool can be very useful, which on one image combines together data, which should be received, which have been received actually and information on distribution of the data on processors. Such way of a presenting the material in many cases allows not only to find out a mistake from one sight, but to understand its reason.

The need of inverse analysis of computation process is frequently arises during debugging. For example, the user has found out, that some variable has incorrect meaning, then after carrying out restart of a debugger, he establishes the operator which has done it, that is, the user made one step upwards on a tree of dependencies, trying to find the node with incorrect variable. It is possible to make such tool: for given variable to build and to display on reasonable depth a tree of dependencies together with meanings of the variables, located on its nodes. Then the user receives at once all this tree and can, by examining its nodes, quickly to find a source of a mistake. Actually this tool is necessary and at usual sequential programming. For the parallel programs this task becomes more complicated because the data can come from the outside and their transformations should be traced in other processes. The systems with such opportunity to us are unknown.

4. Conclusions

The elaboration of debugging and monitoring systems for the parallel programming has the purpose to create such environment of development, the work in which would be so effective and convenient, as in some systems of development of the sequential programs. The works in this area are conducted on different directions.

In the field of construction of systems of monitoring and tuning almost at once was found the successful decisions. As a result the concept of the multilevel debugging was elaborated. This approach allows effectively from the different points of view to carry out study of behaviour of the parallel programs on various architecture. In a number of cases the toolkit appears even more convenient and advanced in comparison with similar systems for sequential programming.

In the field of designing the parallel programs there is an active search of convenient and effective means. At all a variety of the offered approaches they have essential common features.

Appendix A. Design systems and parallelizing compilers

ADDAP - Automatic data distribution and parallelisation - automatic distribution of the data and a parallelisation for MIMD.

University of Saarlandes, Germany. [12], [13].

Input language Pascal, resulting program - on Ñ.

Intel iPSC/860.

The limited class of distribution of the data is considered: each dimension of an array is divided on equal segments, so multidimensional the array breaks up on N of blocks. At distribution of the data each processor has one such block. The processors thus form multidimensional grid so the neighboring blocks are on the neighboring processors, and the grid is mapped on hypercube iPSC/860. Thus the program can on a level of the most external cycles be broken on some parts, and for each part to choose the distribution of the data. The data reference graph is defined, with which help for each distribution the estimation of intensity of a flow of the data arising because of necessity to exchange by the absent data turns out. At construction data reference graph is absent the complete analysis: the access to arrays is possible only through linear combinations of variable cycles. The arising restrictions are rather essential: Is forbidden indirect addressing, there is no interprocedural analysis.

The results received at a parallelisation Livermore Loops are provided: from 24 cycles 9 have appeared non-parallelized because of the limited capabilities at the analysis of dependencies on the data. On 5 cycles the speedup less than 3 times (on 32 processors) is received, on 7 - the speedup is less than 20 times, on 2 - there are less than 25 times and on one program the speedup in 31 times is achieved.

CAPTools - Computer Aided Parallelisation Tools - automated parallelizing compiler.

University of Greenwich, London. [5]. Greenwich.ac.uk.

FORTRAN 77.

Platforms: i860 + T805, SUN clusters Ethernet based communication

The very large and advanced work. In Parallel Computing, 1996, Vol 22, N 2 4 large articles are published at once, in which the approach of the authors is submitted.

By development of the project the following purposes were put:

- The maximum complete analysis of dependencies.
- The greatest possible attraction of the user during a parallelisation.
- The maximal connection with the initial sequential program.

As program model is chosen SPMD (Single Program Multiple Data), thus resulting parallel program is always one, and the data are distributed between processors. The work is made in an interactive mode under the following circuit:

- Definition of dependencies between variable. The authors use Banerjee test and Omega test and write, that the complete analysis of dependencies is one of most strengths of their system. The separate article in [5] is devoted to this theme.
- For some parameters, is especial for those, which values are entered outside, CAPTools requests the user about a probable range of values. The ranges of probable values are distributed from one variable to another with the help of the powerful tool - SIDA (symbolic inequality disproof algorithm). The dialogue with the user is organized iteratively: inquiry - specification, inquiry - specification and so on.

- After the answers to questions the system for each cycle gives out to the user the information on an capability it to be parallelized, thus the user receives a quantitative estimation and has an capability to receive the information on each expression resulting in restriction an capabilities to parallelize a cycle. At this stage, the user again can specify some parameters of the program. The final decision on what cycles will be parallelized the user accepts.

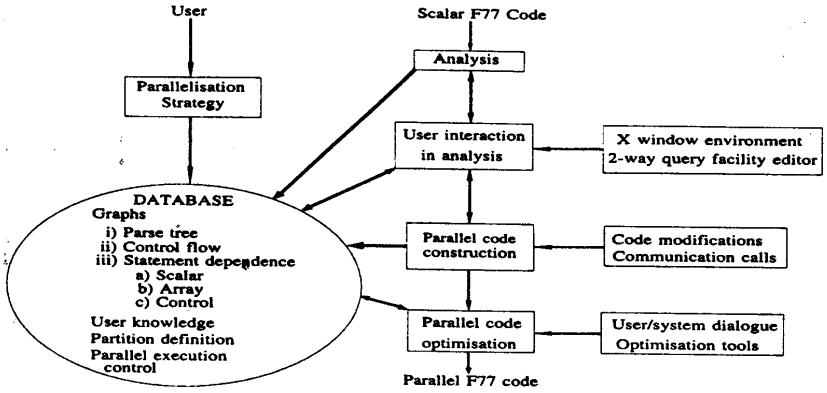

Figure 3.

- The arrays, which are indexed by the parameters of parallelized cycles, are distributed among processors, so the parameters of these cycles change according to distribution of the data. Besides for some operators the masks of the control are entered, so masked operators are carried out only on some (can be on one) processors.

- In cases, when data possessed by some processor and determined by it are required to another one, in the appropriate places of the program the operators of an exchange are inserted. These operators subsequently can be displayed in those or other concrete system calls (PVM, MPI, PARMACS).

Some examples of system work are provided, for example, the program consisting of 4500 lines and 36 subroutines (FORTRAN 77) was parallelized for 4 days. Thus on 8 processors i860 the speedup - 5.8 is received. The program consisting of 3300 lines and 17 subroutines is parallelized after some hours and on 16 processors i860 has speedup in 10 times.

Maximum quantity of processors, on which the received programs - 16 were tested. The programs, on which the testing was carried out, belong to a class 'scientific calculations'. It is interesting, that for one of the programs the system to improve quality of a parallelisation, has offered to the user to change iterative algorithm of the decision of system of the linear equations, so in result other, known iterative circuit with a little bit smaller speed of convergence has turned out.

CODE - Computation-Oriented Display Environment. Language of the description of a dependencies graph, which is considered as model of parallel programming. The purpose - to develop a means of designing of the parallel programs independent both from architecture, and from the programming language.

University of Texas at Austin. [1].

The nodes of a graph refer to as unit, and are the sequential programs in languages (Ada, C, Fortran, Pascal) or subgraphs. Transformation of the data and law of activation (firing rule), that is condition, at which performance unit begins to be executed, characterize every unit. Elementary such condition - all arguments are determined, that is are calculated.

Edges of a graph - dependence on the data or management (instead of channels of communication as in TRAPPER). Both the nodes and edges can be arrays.

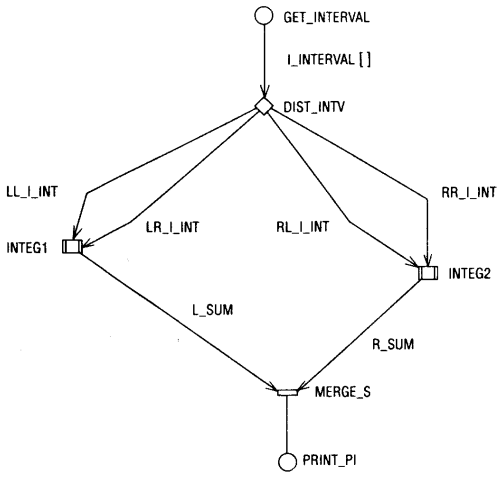

Figure 4.

To form a graph it is possible with the help of the graphic editor, the transformation of the data is set by the appropriate program, and law of activation - filling of the predetermined tables. There is ROPE (Reusability-Oriented Programming Environment) - library of functions on CODE. The input information for system is the substantial description of a task, and the system enables to express dependencies how they are seen by the user, but does not give any tool, for representation of a task in a parallel kind.

The generated task on CODE then can be translated in one of supported languages.

The simple example allowing to understand of the basic idea and engineering of work with CODE is disassembled. It shows what to master system not so simply.

The system will be convenient at designing the applications having the form of the pipeline.

Faust - Environment of development of the parallel programs including the editor, compiler, debugger, graphic editor of Make-files and toolkit for an estimation of efficiency.

University of Illinois. [4].

Supported languages: FORTRAN 8.x, Cedar Fortran, C.

Platforms: Alliant FX/8, BBN Butterfly, Ardent Titan.

The basic concept of system - project, in which communicates the information concerning the developed application. Project besides a usual component includes also graph of the program, the tables of dependencies generated by system, trace of performance of the program.

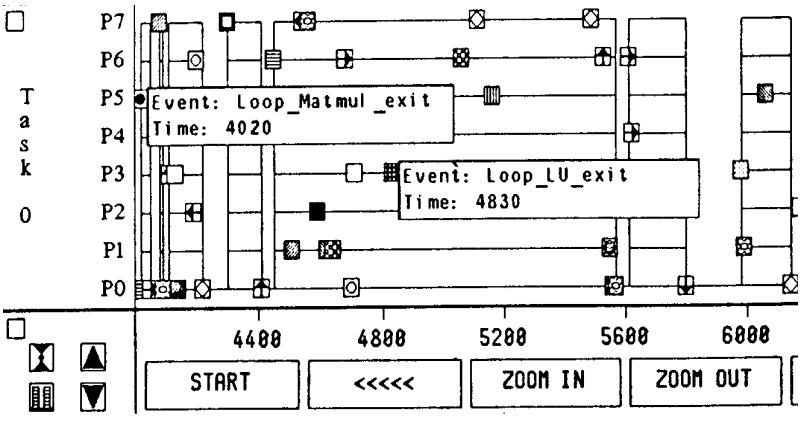

Figure 5.

The central Faust tool - specific editor Sigma. With its help the user can allocate interesting sites of the text of the program (it is the so-called local analysis) and to receive the information on a graph of dependencies for this site and estimation of its efficiency in flop.

The tool Impact, included in Faust, allows to collect and to display a trace of performance of the program. The user has an capability in a special file to connect with each event the certain icon and on

graphic representation of a trace, those events will be submitted only, for which such linkage is given. Such capability allows to examine a trace from the various points of view and is absent in ParaGraph, in which all images are predetermined.

HeNCE - Graphical development tools for network-Based Concurrent Supercomputing.

Oak Ridge National Laboratory, University of Tennessy, [38].

PVM, Isis, Cosmic, Linda.

Not very enhanced but very clear and attractive work.

HeNCE is implemented on top of a PVM - a software package that allows the utilization of a heterogeneous network of parallel and serial computers as a single computational resource. The strength of HeNCE is its ability to tie together drastically different architectures into a virtual parallel supercomputer.

The HeNCE philosophy of parallel programming is to have the programmer *explicitly* specify the parallelism of a computation and to aid the programmer as much as possible in the task of designing, compiling, executing, debugging and analyzing the parallel computation. In HeNCE, the programmer specifies graph, where the nodes represent procedures and the edges denote dependency. There are six types of constructs in HeNCE graph: subroutine nodes, simple dependency arcs, conditional, loop, fan and pipe constructs. Subroutine node represent subroutine written in either Fortran or C. A subroutine node has no state other than its parameter list. That is it cannot read any global information from other subroutine nodes. Arcs from one node to another represent the fact that the tail node of the arc must run before the node at the head of the arc. During the execution of a HeNCE graph, procedures are automatically executed when their predecessors, as defined by dependency arcs, have completed.

The parameter-passing interface is one of the strength of HeNCE: programmer need only specify which parameters are to be used to invoke each subroutine node. These parameters are specified when the programmer attaches a subroutine to a node in the graph. By automatically passing parameters, HeNCE programs are easy to build from pieces of code that have already been written.

The HeNCE tool provides an interface for creating graphs, configuring virtual machines, visualizing trace information, compiling, executing, and analyzing HeNCE program. At the stage of configuring PVM HeNCE requires that user specifies a cost matrix where entry a_{ms} is an integer that specifies an estimated cost for running subroutine s on machine m.

ParaScope - Parallelizing editor for shared memory systems.

Rice University, Houston, USA. [9].

FORTRAN 77, IBM Parallel FORTRAN, FORTRAN for Sequence Symmetry.

Historically ParaScope is the successor of three systems:

Ptools - browser of dependencies developed by the order of Los Alamos National Laboratory. Is established at many centers including in Cornell National Supercomputer Facility.

Rn - interprocedural analyzer and optimizer.

PFC - parallelizing compiler.

Originally PFC was completely automatic parallelizer, but at the numerous requests of the users ParaScope there was interactive - it allows the user (as well as CAPTools) to participate during a parallelisation.

ParaScope during processing the text of the program carries out the following actions:

- Construction of dependencies on the data and on management. The specialized editor allows to look through, to analyze, to sort dependencies and so on. Thus the dependencies are displayed in a textual kind, and continuously connection with the text of the program is supported.

- The analysis of dependencies. A question on that whether are depended two references to the same array are very involved: the answer depends on, whether these references can address the same element of an array. Frequently system simply is not capable to answer this question and then it considers, that the dependence is - such dependence refers to as false.

- Classification of a variable cycle. Every variable is classified: private or shared. The opportunity to parallelize the loop is the greater the less number of shared variables are in the loop body.

- The interprocedural analysis.

- Synchronization. If some loop can be parallelized, and between its iterations are dependencies - that the dependent operators were carried out in the correct order in the program the operators post and wait - generation of event and its waiting are inserted.

- Transformation of the text of the program. There is a set of transformations of the program, which the system can carry out under the control of the user. Simultaneously system transforms of the dependencies graph. The purpose of these transformations - elimination of dependencies and increase of a degree of a parallelisation of the program. After a choice of transformation the system gives an estimation of a prize at parallel performance or warning that the order of performance of the operators is broken or can be broken. Exists 4 such as transformations (for each transformation there are references to the literature):

    * Reordering transformation: loop distribution, interchange, skewing, reversal, adjusting, fusion.

    * Dependence breaking transformation: scalar expansion, array renaming, loop peeling, loop splitting.

    * Memory optimizing transformation: strip mining, scalar replacement, unrolling, unroll and jam.

    * Others: sequential - parallel, statement addition, preserved dependence, constant replacement.

In article, the good example of transformation of a cycle is resulted. The authors write, that their system differs from similar by the large set of transformation of the text of the programs.

During the answers to questions on ranges of the variable value user can make a mistake. The authors assume to keep the protocol of the answers on such questions, and during debugging to check, whether there are meanings variable for these borders.

ParaScope can be used for development of the new applications, parallelizing sequential codes and verification of the existing programs. The evolution of a product is instructive: from completely automatic parallelizer to the program with the expressed interactive mode.

SUIF - Stanford University Intermediate Format.

Stanford University, USA, suif.Stanford.EDU. [10].

FORTRTAN, C.

SGI, Stanford DASH multiprocessor, Kendall Square Research KSR1.

The system the authors name compiler infrastructure. Language is developed, in which the FORTRAN and C programs are translated, and for this language the powerful and structured toolkit is created, with which help it is possible to convert the program into SUIF. Then received program on SUIF it is possible to translate on the necessary language for the necessary platform: with shared or distributed memory. Among capabilities of system, the authors name optimization MIPS of a code and Verilog.

Structurally SUIF consists of a small kernel and toolkit.

Functions of a kernel:

- Definition of internal representation of the program: SUIF the format makes representation of the program by independent from language.

- Functions allowing to manipulate with objects in internal representation.

- Maintenance of the interface between various passes of the compiler. The system is designed in such a manner that all passes of the compiler have the same interface. The user can free combine these passes, achieving the necessary purpose. It is not effective, but it is very convenient.

Functions of toolkit:

Data input: FORTRAN and C are supported, thus the programs on FORTRAN at first are translated in C with the help of AT&T's f2c.

- Parallelisation of cycles and local optimization. As well as in CAPTool as model is chosen SPMD - by result of work is one program working with the distributed data. Except for a standard set of means, the system has an capability of a parallelisation of reductions, that is such calculations, at which the arrays turn to a constant: the sums, products, minim and so on. The algorithm from [35] is applied to definition of dependencies.

- Optimization MIPS of a code.

- Toolkit for development of compilers. The set of functions frequently used by development of compilers, including algorithm Fourier-Motzkin.

- The simplified variant of system for the students.

The system has been successfully tested on the Multiflow test suite, Perfect benchmark, SPEC92 benchmark and NAS parallel benchmark.

The system has volume 200.000 lines C ++, is distributed freely and perfectly is documentary.

TDFL - Task-Level Dataflow Language - Graphic language for architectural - independent parallel programming.

University of Southern California, IBM T.J. Watson Research Group, University of Texas at Austin, University of Singapore, [23].

CODE.

CDC Cyber 170.

By development TDFL the authors followed three following principles:

- Specification of computing and communication parts of the application one should be separated from another.

- At creation of computing procedures use of standard compilers should be supposed.

- At language there should be following designs: a cycle alternative choice (as ALT in OCCAM), duplication of processes etc.

The basic concept TDFL reminds system HeNCE: the parallel program is represented as a graph, which units are the consistently executed procedures, and the edges specify dependence on the data and on management. Thus, as well as in HeNCE, each unit some rule of inclusion - boolean a condition is attributed at which performance there is solved a performance of procedure appropriate to this unit.

However set primitives at these systems different, their images essentially differ, the graph in TDFL is not acyclic. In TDFL the generation of new processes (nodes) is supposed during performance of the program (graph). Elementary constructors of TDFL are:

- Simple node - sequential procedure.

- Merge node - nondeterminism - analogue a costructor ALT in OCCAM.
- Loop node - cyclic operator - recurrence while is carried out given boolean a condition.
- DoAll Node - parallel performance - analogue of the costructor PAR in OCCAM.
- Case node - choice on a condition.
- Self-loop arc - arc of dependence outgoing and included in same unit.

At realization of language TDFL the graphic interface of system CODE was used. However both from the text of an article and from the given examples it is impossible to understand how with the help CODE primitives, which set is much poorer, it is possible to express constructors of language TDFL. Not clearly also what for it was done and what for then TDFL is necessary.

TRAPPER - Integrated graphic environment of development of the parallel programs.

Daimler-Benz AG Research and Technology, German National Research Center for Computer Science, Germany, [7].

PVM, MPI, Parmacs.

The system consists of four parts:

- Designtool - graphic editor allowing to set and to edit structure of the parallel program as a graph. The nodes of a graph represent processes, and edge - channels their connecting. The user has an capability 'to enter' in any node and by appearing in the text editor to edit the program appropriate it to unit. Node can represent and subgraph. The system does not give any means for automation of process of a parallelisation (as well as CODE [1]) - it enables in a convenient kind to present the application, which parallel realization is already generated. On the other hand convenient representation of the parallel project facilitates understanding of interrelations between its parts and accelerates development.

- Configtool - graphic editor allowing to set architecture of parallel system, for which is projected the application. There is a library of standard products: B008 and so on. Same part includes the program allowing optimally to map a graph of processes on a graph of architecture.

- Vistool - system of visualization of behaviour of the application during performance. There are on-line and off-line modes. The viewing of a trace is possible in two variants. First with use of an axis of time, so displayed parameter is postponed on other axis. The authors write, that all diagrams from ParaGraph are present. Secondly with the help of the image of a graph of processes generated with the help of Designtool, so flows of the data and the activity of processes is displayed as animation of this graph. The system supports the Analysis of a Critical Branch (see. IPS-2).

- Perftool - system of display of efficiency of use of the computing system. Vistool displays behaviour of the program, and Perftool - the efficiency of use of the calculator and allows to connect one with another.

TRAPPER is realized on C ++, has volume 80000 lines and is carried out on UNIX stations. A strength of system is its user simplicity.

VERDI/Raddle Graphic environment for development of the programs for the distributed systems and language of designing.

University of Texas; MCC, NCR, Austin, [16].

The basic concept VERDI is action, which are formed of five primitives: code_block, parallel, interaction, cast and return with the help of the following rules of association: choice, sequence and iterator. The icons correspond to the primitives and rules of formation - rules of connection of icons on the image.

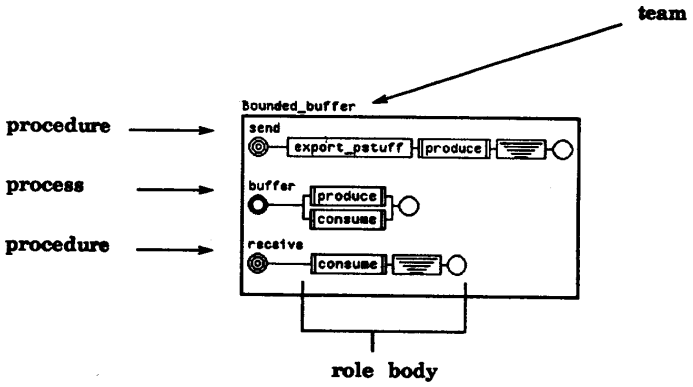

Figure 6.

Each expression in VERDI can be supplied with boolean condition, which is a condition of its execution. From action the procedures and processes are formed which can be united in team. Synchronize and communication mechanism VERDI is n-party. The icons of interaction (forming the special class), taking place within the framework of same party and having the same name, can cooperate, if each of them is in a condition of readiness. The interaction consists in parallel performance of the operators corresponding to these icons. Thus the assignments can be carried out only in local variable by each of them. If in any moment some interactions can be executed, they are executed substantially, the order of performance gets out casually.

VERDI does not support dynamic creation of processes and has not global variable.

VERDI has both graphic, and text editor. The graphic editor allows collecting the project, specialized text editor - to set a name, condition of performance and operators of assignment for every primitive. At the assignments the system allows to use syntax of language C, but without use of the constructors 'while', 'for' and so on. Apparently it results to too fine-grained projects, as compels of the user everyone, even an only arithmetic cycle, to take out on a graphic level.

The system allows executing the assembled project. The performance can occur in several modes and is accompanied by animation and colour effects. VERDI has statistic toolkit allowing building in the project the sensors, to collect and to process the information, interesting for the user, on a course of performance of the project.

In article the examples of use VERDI are resulted: including together with Motorola the project connected to an optimum choice of telephone unit at movement of the mobile telephone.

The authors mark, that the system can be with success used at work with the customer at a stage of demonstration of a capability of realization of the project.

Appendix B. Parallel debuggers and monitoring systems.

In one of articles the results of interrogation are published, according to which 90 % of the programmers developing the parallel programs, do not use debuggers, preferring *print* and so on.

AIMS - Automated Instrumentation and Monitoring System. Performance tuning tool-set. AIMS is distributed freely.

NASA Ames Research Center, USA, [6], yan@nas.nasa.gov.

The purpose of this interesting work is not only to give, as the authors write, series 'pretty pictures', but to give the user the capability been beginning from attributes, seen by an eye, of an inefficiency of the program to understand their reasons with the help of quantitative estimations and even to receive from system the help - probable way of transformation of the program raising its efficiency.

The system consists of the following elements:

- VK - View Kernel - displays the dynamics of program execution using animation. This is more or less usual performance visualization tool.

- Xisk - Block of calculation of indexes of efficiency. At first is determined, what percent of time the application was engaged in calculations, communications, input\output data or was suspended by the system. These parameters AIMS are capable to determine for each processor, procedure or even for a fragment of the text of the program given by the user. Then these times are divided into a parts, for example, the communication time is subdivided into time of exchanges of a type a point - point and time of global exchange, these times are in turn divided into a send time and receive time and so on. These data form a tree of indexes. The user, thus, receives an capability, by noticing a parameter, not satisfying him, to pass on this tree up to last sheet and to find a site of the program, on which it stands idle most of all. But it yet not all - the authors on each sheet of a tree of indexes of efficiency, offer to the user the recipe allowing to improve efficiency. Sometimes such recipe looks a little trivial - 'change distribution of the data', but sometimes it happens is quite reasonable. For example, if the overhead charge is too great, the system considers, that in the program it is too much short packages and advises to block them in larger.

- MK - Model Kernel - system of a prediction of the program efficiency after the given changes of its structure or at execution on other parallel system. It rather new idea. The authors reason about bad scalability of the parallel programs and offer the tool of a prediction of the program efficiency on other architecture. For this purpose the trace is created during program execution, and then on this trace and on parameters (the speed and parameters of an exchange) other parallel architecture is carried out a prediction of an overall performance of the program on this architecture. The method thus appears is rather simple - the duration of sites of calculation and exchanges available in a trace are recalculated. The results of a prediction of an overall performance of the program on Intel Paragon are provided, proceeding from a trace taken off with iPSC/860. The mistake for the different programs changes from 0.2 up to 8.3 percents - not bad!

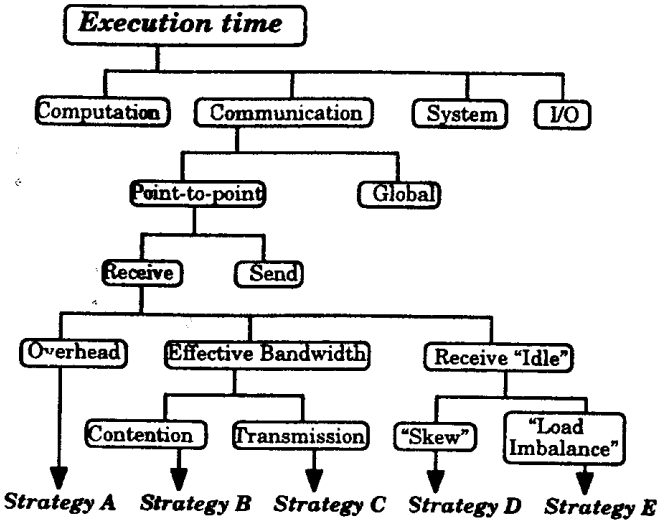

Figure 7.

In article the examples of use AIMS in different situations are provided. As against other works on this theme here there is a mention of indemnification of delay of the program execution as a result of measuring tools insertion.

CAPS - Coding Aid for the PASM (Partitionable SIMD/MIMD) system. System for the analysis of behaviour of the parallel program. CAPS - Coding Aid for the PASM (Partitionable SIMD/MIMD) system. System for the analysis of behaviour of the parallel program.

Purdue University, Indiana; University of Iowa. [19].

PASM - parallel processor developed in Purdue University [20].

UNIX System V.

The work is interesting to that effects arising at embedding into the program a sensors - sites of a code which form a execution trace are studied in it. The authors proceed from an event-action paradigm, which considers performance of the distributed program, as a sequence of transitions of processes from one state to another. The events generate actions, which can be finished by new events. The events are distributed from process to process through the messages transmitted on channels. Monitored and monitoring the systems are connected among themselves by channels for transmit the information on events occurred in monitored systems and representing interest for monitoring. Thus in monitored system are brought in of intrusion. The

measuring system refers to as intrusive, if as a result of measurements or the state of system are changed or the order of events varies because of delays connected to processing of a flow of events. The authors consider the following characteristics of measuring system:

- Δt^r - time necessary for transmission the event from monitored to monitoring system.

- Δt^l - time necessary for measuring system for definition of action, (such as processing) which must be executed with the occurred event.

- Δt^d - time necessary for monitoring system to process occurred event.

- ΔH^e - time necessary for monitored system for performance of action, caused by current event *e*.

With the help of these characteristics the authors determine conditions, at which the system can be non-intrusive and capable to register a flow of events occurring in monitored system. It is marked that the realization of non-intrusive monitoring system has both conceptual and technical difficulties. The majority of existing systems - intrusive, those are and CAPS.

The difficulties connected to determining the order of registered events, occurred in different processors are marked. These difficulties arise as in connection with impossibility of exact definition of the moment of approach of events, and that the sizes Δt^r are different for different ways from measured up to measuring of processes, and the order of arrival of the messages about events, can differ from the order of their realization.

The classification of events is offered: simple, complex, local, unlocal, global. Complexes the events refer to as which are considered occurred, only if some condition was executed.

It is offered multilevel models intrusive and non-intrusive of monitoring systems, in which each level cooperates only with next. At construction of non-intrusive system the additional equipment is necessary which should be capable to display a condition of processors on a condition of processes and to allocate from them registered events.

CAPS allows the user to receive from each processor the information on a condition of the registers, memory, to establish control points and so on. The programs on PASM are executed under the control of the resident monitor, on base Motorola's VMEbug. The system is integrated and allows executing a complete cycle of debugging.

DAD - Data Access Descriptor - way of data dependencies representation at parallel programming.

Caltech Concurrent Computation Program, Pasadena, Rice University [17].

In many tool systems (PTOOL, PFC, ParaScope …) the following circuit of work is accepted: the user requests system about this or that information, and the system gives out it to user in an evident kind and the user makes a decision. One of main such inquiries are the requirement to construct of the dependency graph. The experiments with such systems allow to make the following conclusions:

- The graph of dependencies is the information of rather low level and not always intuitively clear to user.

- It frequently has volume, with which user be not capable to consult.

- Toolkit, existing to the present moment, be not capable to build of the exact graph of dependencies and in complex case ads nonexistent dependencies.

- The complete analysis of dependencies frequently (for example, at the interprocedural analysis inside a cycle) becomes too long and expensive process.

- In the real programs, not all sites of the program can be parallelized.

The author offers the alternative approach to representation of dependencies of the data in a convenient and evident kind, at which the dependence of the data is treated as crossing of areas of access to the same array of several sites of the program. Each reference to an array is described with the help (DAD) Data Access Descriptor - 3 tuple: a reference template, area of access and traversal order.

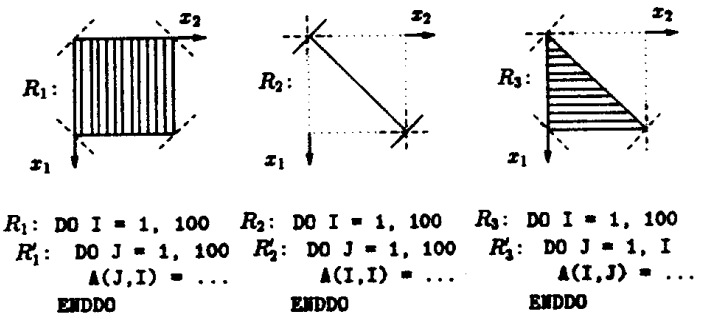

Figure 8.

The areas of access thus are approximately described with the help of the concept, entered by the author, simple section: the convex set limited hyperplanes of a kind xi = c, xi±xj = c, where xi, xj - coordinate of an array, access to which is described. It appears, that each simple section can given with the help largest of 2n2 of conditions, n - dimension of an array. The crossing of simple sections is simple section, and instead of simple sections union the smallest simple section containing the exact union is considered.

The traversal order sets the order in which expressions specifying index, change at change of a variable cycle, in which there is a considered reference to an array.

In article the pseudo-codes of the programs building Data Access Descriptor for given area and the references, building union and intersection of the DAD, transformation DAD from a body of procedure to the point of a call, transformation DAD are after reduction of number of the fixed variable cycle. The work contains many examples allowing to understand a detail of the offered approach.

The author offers to use DAD at debugging the parallel programs, at designing distribution of the data among processes, for visualization of distribution of the data.

Devise/Paradyn - integrated system for visualization of a trace of the parallel program execution.

University of Wisconsin. [14], [15].

COW - cluster SPARCstation20, consisting of 40 Ethernet-connected stations.

The system Devise allows to integrate in one image some traces received from different sources: from library functions (PVM or MPI); from systems of monitoring of the programs (Paradyn); from the manager of resources (Condor); from operational system (UNIX of the utility vmstat and iostat) or is direct from the application. Thus the joint viewing of several traces for their comparison is possible. The system is so open, that can be applied in other areas: the financial market, clinical medicine and so on.

The process of visualization in Devise is buffered, and it allows to look through the data practically of unlimited volume. The process of visualization consists of the following stages. The source of the information is considered as an input flow of records in a format TData, attributes, containing some. Such records are displayed on structure GData with the fixed fields: the size, colour, coordinate and so on. The user choosing the representation, necessary to him, of the data operates the process of display. The graphic filter is determined with what part of the image should be deduced on the screen. Then GData is converted in Vdata - bitmap of the image. Last stage - accommodation of the image on the screen, at which is possible to receive combinations of the several images. The input format Devise is capable to process MPI and PICL log-files. In system there is a cursor, which allows to receive the more detailed image interesting for the user of sites of the image.

Paradyn is tool system allowing to study efficiency of the parallel program: the collection of the information, automatic search of bottlenecks and visualization of the data. At work with Paradyn the user can choose from the predetermined list a subset of hypotheses about the reasons of low efficiency. After that system itself determines specification of parameters, which are necessary for measuring, carries out transformation of the investigated application for construction of a trace of its performance. The user has an capability to set parameters, which are necessary for measuring, by choosing the *metrics* and *focus* -

concepts, specific to systems determining that to measure and where to measure. Paradyn allows the user to add to system his images.

Mentioned above Condor is the manager of resources for stations cluster capable dynamically to redistribute of a task on cluster, providing their uniform loading. Condor in University of Wisconsin works 7 years. With the help Devise it is possible to visualize a log-file (which it was necessary to convert), assembled for these years.

As advantage of the work the authors consider an capability to display and to combine the data received from different sources.

Instant Replay - approach to principles of construction of debuggers for the parallel programs.

University of Rochester. [8]

BBN Butterfly.

The article of the 1987 that has put in pawn theoretical bases of construction of debuggers for parallel systems. So-called 'the trial effect' consists that the work of parallel system is unstable in the sense that after the minimal change of the parallel program the logic of its behaviour can change essentially. For example, let process A waits the message from processes B and C, so if by first there comes the message from B the program that is executed on one branch, and if from C - on another. Let at some data by first there comes the message from B, and during debugging the process B was stopped until as it has sent the message to process A. If the processes A and C are not stopped (and where them to stop - it is not known), the process A will receive the message from C. and the logic of work of all system will be perfect another - user after this stop should begin at first, differently he will look at a condition of system, which actually was not realized, and grew out of intrusion to the program, that is result 'of trial effect'.

The authors offer the following idea. At first to write down process history tape - for each process a sequence of accesses to shared data (message by consideration message-passing of systems), so thus current number of the version of each data set to which is fixed the access - writing-reading and number of the processes which have read each version is fixed. It appears, that this information is enough for exact replay (it and is Instant Replay) process of performance of the program, and volume of the written down data is not too great. Thus the user has an capability to change the program (to insert prints and so on.) - if only the circuit of the communications was not changed - Instant Replay guarantees exact reproduction of the written down variant of behaviour of the program. This is a very useful property.

The authors realized this idea and approved it for different, including very unstable programs. The not too complex idea, however work has become classical - everybody refers on it even those who simply ignore a problem 'of trial effect'.

IPS-2 - system of measurement of efficiency of the parallel and distributed programs.

University of Wisconsin-Madison, AT and T Bell Laboratory, DEC. [11].

VAX, DECstation, Sun 4, Sequent Symmetry multiprocessor.

4.3BSD Unix.

IPS-2 gives means for measurement of efficiency of the parallel program and help to the programmer at definition of bottlenecks. Measurements and their display can be carried out at the following levels: the program as a whole, loading of processors, processes, procedures and all measurements in aggregate.

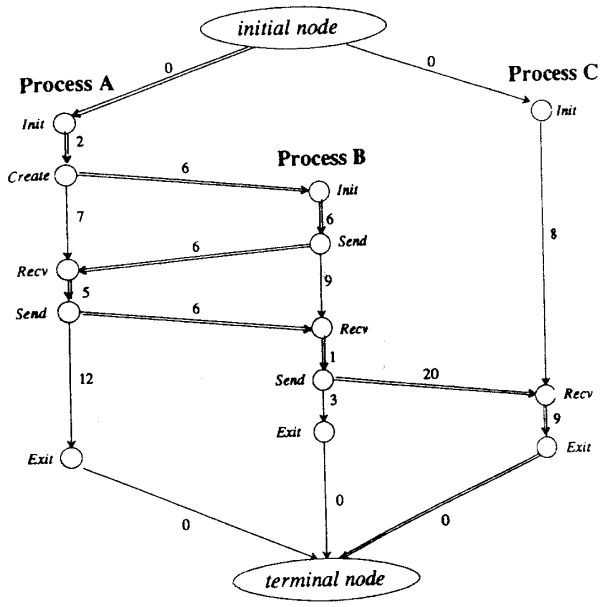

Figure 9.

Structurally IPS-2 consists of four parts: the tool of measurement, pool of the data (in local memory of each machine), local and global analyzers and interface of the user.

The programmer for reception of the program trace at all should not change: it is done automatically.

Many efforts are enclosed to minimizing an overhead charge from introduction sensors in the program: direct access to the timer (that is why 4.3BSD) and packing of a line. Each reference to shared memory is not fixed: the times of the reference to appropriate semaphore are measured only. The overhead charge at reception of a trace makes 10-45%.

For work with system the graphic interface is developed, where the simple editor allowing to describe architecture of the system and each processor to map the executed program. After the program was executed the image of each executed program turns to a tree of processes/calls, which can be edited, leaving only necessary information.

The system helps the user in search of bottlenecks - reasons of low efficiency. For this purpose two techniques are developed:

The analysis of a Critical Branch - determining the branch, which consumes the most of time. Then on this branch the sites (sequence of events) are determined, which a lot of time is spent and so on. This branch is displayed on the Graph of Activity of the Program, which nodes are the moments of creation of processes and interprocess interaction, and the edges represent time between them. There is a tool allowing approximately to estimate of change in behaviour of the program, provided that the Critical Branch will be changed definitely.

The analysis of Phase Behaviour. The authors define a phase, as the periods of time, during which distribution of loading of the processor, frequency of the communications, waiting time and so on, are about identical. The system determines phases of performance of the program and has an capability to analyze them separately.

One of the strengths of system is the complete automation of preparation of the program to measurement of efficiency.

Moviola/Multiple Views - the analyzer of the program performance, in which at various stages of debugging and designing is used the most appropriate tool.

University of Rochester, New York; Rice University, Houston. [18].

The type of the processor MC68000 MC68020 is informed only.

Development of toolkit for the analysis of behaviour of the parallel programs is based on two methodological principles:

- The analysis of behaviour of the program follows stages of its development from search of mistakes up to the analysis of efficiency and optimization.

- The analysis is carried out from above - downwards from a general sight on the program as a whole up to details of performance of separate procedure on one processor.

The offered methodology of debugging and analysis of behaviour of the program has four phases:

- Determining of the reasons (deviating) behaviour of the program.

- Reception of general characteristics of efficiency of the program: coefficient of acceleration and so on.

- Program optimization.

For each of the listed stages some of the points of view on process of performance of the program and its detail is offered. The debugging of the program sees as step-by-step process from one point of view to another, so the user, being in some point, can choose most suitable for him in the given moment the next point of view. The list of all types of the offered points of view is resulted below:

- *Program view* - point of view at which behaviour of the program is visible as a whole and the result of work of the program is seen.

- *Anonymous process view* - program is represented as a collection of processes. This point of view allows approximately estimating a degree of parallelism of the program.

- *Interprocess communication view* - shows the communications between processes.

- *Data parallel view* - illustrates the relation between processes and data, for example distribution of the data between processes.

- *Processor view* - shows the relation between processes and processors - for example degree of loading of each processor.

- *External view* - shows process from the point of view of its communications with the external world.

- *Infrastructure view* - contains the information on calculations that are executed by process.

The interface of the user is programmed and bases on Kyoto Common Lisp. It allows the user to build the own points of view, for example, specific to its application. Work contains an excellent example allowing understanding offered method of the parallel program debugging.

ParaGraph - construction and graphic display of a trace of performance of the parallel program.

University of Illinois, Oak Ridge National Laboratory. [3].

One of the first, but very successful work on this theme. Has a high citation index - practically all refer on it.

The following circuit of study of behaviour of the parallel program is offered. In the program in places, interesting for the user, sensors are built, These sensors write during the program execution some given parameters - such as times of passage of critical points (beginning and end of exchanges an so on.), the meanings of the given values and transfer them to the host-machine, where the trace of the program execution is formed. The effect from such intervention in the program is considered small, as the results of trace are stored in local memory of each processor and are unloaded only after end of the program execution. The trace is created with the help of PICL (Portable Instrumented Concurrent Library), developed in ORNL. Among systems, on which PICL is ported are mentioned Intel, N-Cube, Cogent and Symult.

The received trace of the program then is scrolled with the help of system of visualization - here it and is ParaGraph (X-Windows). ParaGraph is capable to display traces of systems having up to 512 or (some diagrams) up to 1024 processors. The trace of the program received with the help of PICL allows to determine a state of each processor in each moment of time: calculation, communication (the authors name it of overhead) and idle. The basic part of ParaGraph carries out display just of these value, thus 25 various diagrams are given, many of which appear very useful, interesting and unexpected.

ParaGraph has the tool allowing the user to create the own diagrams, if only the input information was in the appropriate format. The doubtless success of system has become possible in many respects due to two

of 'easy', professed by the authors - 'easy to understand' and 'easy to use' - the file Readme has only one page. ParaGraph and even its texts were freely distributed to the moment of the publication (1991).

PATOP/DETOP - PATOP - performance analyzer and DETOP - parallel debugger.

A commercial product, about 100 deliveries.

Technical University Munchen, Germany Parsytec Computer GmbH Aachen. [2]

Platforms: Parsytec GC/PowerPlus, PowerXplrer.

The debugger for C, as against many others, is designed for systems with the large number of processors - at least 100. As the further development is mentioned C++, HPF (High Performance Fortran) and version for PVM.

As the important advantage of the approach the authors consider an capability to work with architecture allowing to run several easy processes (thread) on the same processor. DETOP allows to work both with one such process, and with group selected with the user. Thus there is an capability to manipulate with dynamically created processes - to receive to them access at the moment of their creation.

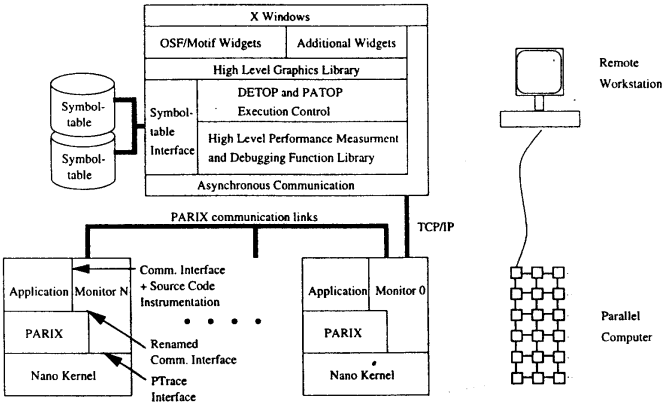

Figure 9.

It is a lot of attention and efforts are given to a question of a scalability of a product. The authors determine the scalability, as it - independence of time of reaction and volume of flows of the processors, from the number of processors in the system. DETOP works on-line. Parallel to debugged application the distributed monitor is loaded, which cooperates both with a debugger, and with the analyzer of efficiency. The separate units of the monitor cooperate among themselves at a level of operational system - PARIX/Nano Kernel. Thus there is no rigid protocol event - action, the user can map actions to the given combinations of events. There is an capability to combine and team of a debugger, setting complex conditions: conveyors and so on.

The interaction of the host-machine to the distributed system executes under protocol TCP/IP: on one of conferences DETOP was executed in California and debugged program - in Germany.

The debugger has three levels of work: display of behaviour of system as a whole, work with the selected group of processes and work inside process - (as in a usual debugger). As the important feature of a debugger the authors consider an capability of work from up to down, so at each level the user receives the information, which allows him to advance further. On this theme there are many reasoning, but what mechanisms realized this unconditional the important idea remained not clearly.

PATOP allows to receive and to display the diagram of the application performance, mean values of selected parameters and so on. In work there is a mention of trial effect (see [8]), but except for that the authors tried to lower volumes of flows of the data and sizes of the monitor anything concrete on this theme is not present. Apparently, this product is designed only for the applications, the logic of which work is very steady to external intrusion.

References

[1] J.C.Browne, M.Azam and S.Sobek, CODE: A Unified Approach to Parallel Programming, IEEE Software, 6(4) (1989) 10-18.

[2] R.Wismuller, M.Oberhuber, J.Krammer Olav Hansen, Interactive debugging and performance analysis of massively parallel applications, Parallel Computing, 22 (1996) 415-442.

[3] M.T.Heat, J.A.Eteridge, Visualizing the Performance of Parallel Programs, IEEE Software, 8(5) (1991) 29-39.

[4] V.A.Guarna, Jr, D.Gannon, D.Jablonowski, A.D. Malony and Y Gaur, Faust: An Integrated Environment for Parallel Programming, IEEE Software, 6(4) (1989) 20-26.

[5] C.S.Ierotheou, S.P.Johnson, M.Cross and P.F.Legett, Computer aided parallelisation tool (CAPTools) - conceptual overview and performance on the parallelisation of structured mesh codes, Parallel Computing, 22(2) (1996) 163-195.

[6] J.C.Yan, S.R.Sarukkai, Analyzing parallel Program performance using normalized performance indices and trace transformation technique, Parallel Computing, 22(2) (1996) 1215-1237.

[7] C.Scheidler, L.Schafers, O.Kramer-Fuhrmann, Software Engineering for Parallel Systems: The TRAPPER Approach, Proc. Of HICSS-28, Hawaii, 4-6 January, 1995, IEEE CS Press.

[8] T.J. Leblanc, M.Mellor-Crummey, Debugging Parallel program with Instant Replay, IEEE Transactions on Parallel and Distributed Systems, 36(4) 1978.

[9] K. Kennedy, K.S.McKinley, Chau-Wen Tsen, Interactive Parallel Programming Using the Parascope Editor, IEEE Transactions on Parallel and Distributed Systems, 2(3) (1991).

[10] R.P.Wilson, R.S.French, and other, SUIF: An Infrastructure for Research on Parallelizing and Optimizing Compiles, ACM SIGPLAN Notices, 29(12) (1994).

[11] B.P.Miller, M.Clark, J.Hollingworse, S Kierstead, S.Lim, T.Torzewski, IPS-2: The Second Generation of Parallel Program Measurement System, IEEE Transactions on Parallel and Distributed Systems, 1(2) (1990).

[12] A. Dierstein, R.Hayer, T.Rauber, The ADDAP System on the iPSC/860: Automatic Data Distribution and Parallelisation, Journal of Parallel and Distributed Computing, 1996, 32 (1996) 1-10.

[13] A. Dierstein, R.Hayer, T.Rauber, The ADDAP system on the iPSC/860: Automatic data distribution and parallelisation, Tech. Rep. 09-94, University of Saarbrucken, 1994.

[14] K.L.Karavanic, J.Myllymaki, M.Livny, B.P.Miller, Integrated visualization of parallel program performance data, Parallel Computing, Vol 23, (1997) 181-198.

[15] B.P.Miller, M.Callaghan, J.Cardille, J.Hollingworth, R.Irvin, K.Karavanic, K.Kunchithapadam, T.Newhall, The Paradyn: Parallel performance measurement tool, IEEE Computer 28 (1995).

[16] V.Y.Shen, C.Richter, M.L.Graf, J.A.Brumfield, VERDI: A Visual'Evironment for Designing Distributed Systems, Journal of Parallel and Distributed Computing, 5 (1990) 128-137.

[17] V.Balasundaram, A Mechanism for Keeping Useful Internal Information in Parallel Programming Tools: The Data Access Descriptor, Journal of Parallel and Distributed Computing, 9 (1990) 154-170.

[18] T.J.LeBlank, J.M.Mellor-Crummey, R.J.Fowler, Analizing Parallel Program Executions Using Multiple Views, Journal of Parallel and Distributed Computing, 9 (1990) 203-217.

[19] D.C.Marinescu, J.E.Lumpp, Jr, T.L.Casavant, H.J.Siagel, Models for Monitoring and Debugging Tools for Parallel and Distributed Software, Journal of Parallel and Distributed Computing, 9 (1990) 171-184.

[20] H.J.Siegel, L.J.Siegel, F.C.Kammerer, D.T.Mueller, H.E.Smaller, S.D.Smith, PASM: A partitionable SIMD/MIMD system for image processing and pattern recognition, IEEE Tranz. Comput. C-30 (Dec 1981), 934-946.

[21] Jingling Xue, Automating non-unimodular loop transformation for massive parallelism, Parallel Computing, 20 (1994) 711-728.

[22] J.J.P.Tsai, Y.Bi, S.J.Yang, Debugging for Timing-Constraint Violations, IEEE Software, March 1996, 89-99.

[23] P.A.Suhler, J.Biswas, K.M.Korner, J.C.Browne, TDFL: A Task-Level Dataflow Language, Journal of Parallel and Distributed Computing, 9 (1990) 103-115.

[24] J.-Y. Berthou, L.Colomber, Which approach to parallelizing scientific codes - That is a question, Parallel Computing, 23 (1997) 165-179.

[25] T.Hey, A.Dunlop, E.Hermandez, Realistic parallel performance estimation, Parallel Computing, 23 (1997) 5-21.

[26] H.El-Rewini, Scheduling Parallel Program Task onto Arbitrary Target Machines, Journal of Parallel and Distributed Computing, 9 (1990) 138-153.

[27] S.Arunkumar, R.Lal, R.Venkatagopal, SIMPAC-T: A Simulator for Multitransputer System, Microprocessing and Microprogramming, 35 (1992) 253-260, North-Holland.

[28] T.Brandes, S.Chaumette, M.C.Counilh, J.Roman, A.Darte, F.Desprez, J.C.Mignot, HPFIT: A set of integrated tools for the parallelisation of applications using High Performance Fortran. PART I: HPFIT and the TransTOOL environment, Parallel Computing, 23 (1997) 71-87.

[29] C.Chu, D.L.Carver, Parallelizing subroutines in sequential programs, IEEE Software, January 1994, pp. 77-85.

[30] L.J.Hendren, A.Nicolau, Parallelizing Programs with Recursive Data Structures, IEEE Transactions on Parallel and Distributed Systems, 1(1) (1990) 35-47.

[31] Z.Li, P.C. Yew, C.-Q.Zhu, An Efficient Data Dependence Analysis for Parallelizing Compilers, IEEE Transactions on Parallel and Distributed Systems, 1(1) (1990) 26-34.

[32] J.May, F.Berman, Retargetability and Extensibility in a Parallel Debugger, Journal of Parallel and Distributed Computing, 35 (1996) 142-155.

[33] W.Pugh, A Practical Algorithm for Extract Array Dependence Analysis, Communications of the ACM, 35(8) (1992) 102-114.

[34] S.E.Hambrusch, A.A.Khokhart, C3: A Parallel Model for Coarse-Grained Machies, Journal of Parallel and Distributed Computing, 37 (1996) 139-154.

[35] D.E.Maydan, S.P.Amarasinghe, M.S.Lam, Array Data Flow Analysis and its Use in Array Privatization, Proceedings of the ACN SIGPLAN'91 Conference on Programming Language of Design and Implementation, June 1991.

[36] K.Smith, W.Appelbe, PAT - An Interactive Fortran parallelizing assistant tool, Proc. 1988 Int Conf. Parallel Processing, St. Charles, IL, Aug, 1988.

[37] R.Sawdayi, G.Wagenbreth, J.Williamson, MIMDizer: Functional and data decomposition; creating parallel programs from scratch transforming existing Fortran programs to parallel, in Compilers and Runtime Software for Scalable Multiprocessors, J.Saltz, P.Mehrotra, Eds. Amsterdam, The Netherlands: Elsevier.

[38] A.Beguelin, J.J.Dongarra, G.A.Geist, R.Manchek, V.S.Sunderam, Visualization and Debugging in a Heterogeneous Environment, IEEE Computer, 26(6) (1993) 88-95.

[39] I.Gorton, I.Jelly, J.Gray, Parallel Software Engineering with RAPSE, Proceeding of COMPSAC'93, IEEE Computer and Software and Applications Conference, Nov. 1993, Phoenix, USA.

[40] M.R.Van Steen, Hamlet Application Design Language, Technical Report EUR _ CS-93-16, Department of Computer Science, Erasmus University, 1993

[41] T.Fahringer, Using the P3T to guide the parallelisation and optimization effort under the Vienna Fortrancmpilation system, Proc. Scalable High-Performance Computing Conf., Knoxvill, TN, 1994.

[42] J.Dongarra, D.Sorenson, SCHEDULE: Tools for developing and analyzing parallel Fortran programs, L.Jamieson, D. Gannon, R.Douglas, eds., The Characteristics of Parallel Algorithms, (MIT Press, Cambridge, MA, 1987).

[43] K.Ekanadham, V.K.Naik, M.S.Squillante, PET: Parallel Performance Estimation Tool, Proc. 7th SIAM Conf. On Parallel Processing for Scientific Computing, San Francisco, 1995.

[44] M.Aspnas, R.J.R.Back, T.Langbacka, A Progamming Environment Providing Visual Support for Parallel Programming, Proceedings of EWPC 1992, the European Workshop on Parallel Computing, Barselona, Spain.

[45] J.Mangee, N.Dulay, MP: A Programming Environment for Multicomputers, Proc. Of the IFIG WG 10.3 on Programming Environment for Parallel Computing, Edinburg, Scotland, April 6-8, 1992.

[46] E.Maehle, W.Obeloer, DELTA-T: A User-Transparent Software Monitoring Tool for Multi-Transputer Systems, Proc. EUROMICRO'92, Paris, Sept. 15-17, 1992

[47] J.K.Hollingsworth, B.P.Miller, Dynamic control of performance monitoring on large scale parallel systems, Proc. 7th ACM Int. Conf. On Supercomputing, Tokyo, Japan, July 1993.

[48] Parasoft Corporation, 2500 E, Foothill Blvd, Pasadena CA 91107, EXPRESS - Building Parallel and Distributed Programs, Version 3.2 (1992).

[49] S.Sistare, D.Allen, R.Bowker, K.Jourdenais, J.Simons, R.Title, A scalable debugger for massively parallel message-passing programs, Proc. SHPCC'94 Scalable High Performance Computing Conf., Knoxville, Tennessee, May 1994, pp. 825-832.

[50] H.Zima, H.-J. Bast, M, Gerndt, SUPERB - A tool for semiautomatic MIMD/SIMD parallelisation, Parallel computing, 6 (1988) 1-18.

[51] K.Ikudome, G.C.Fox, A.Kolawa, J.W.Flower, An automatic and symbolic parallelisation system for distributed memory parallel computers, Proceedings of the 5th of Distributed Memory Computing Conference, 1990, pp 1105-1114.

[52] H.M.Gerndt, Automatic parallelisation for distributed-memory multiprocessing system, Ph. D. Thesis, University of Illinois, Urbana-Champain, II., 1992

[53] C.Koelbel, P.Mehrotra, J.Rosendale, Supporting Shared Data Structures on Distributed Memory Architectures, ACM SIGPLAN Symposium on Principles and Practices of Parallel Programming, 1990, pp 177-186.

[54] A.Schrijver, Theory of linear and integer programming, Wiley 1990.

[55] R.Das, R.Ponnusami, J.Saltz, D.Mavriplis, Distributed memory compiler ' methods for irregular problems - Data copy reuse and runtime partitioning, Languages, Compilers and Run-Time Environment for Distributed Memory Machines, North-Holland, Amsterdam, pp. 185-219, 1992.

[56] M.Rosing, R.Schnabel, R.Weaver, Scientific programming languages for distributed memory'multiprocessors: Paradigm and research issues. Languages, Compilers and Run-Time Environment for Distributed Memory Machines, North-Holland, Amsterdam, pp. 17-37, 1992.

[57] J.Li, M.Chen, Index domain alignment: Minimizing costs of cross-referencing between distributed arrays, Third Symposium on the Frontiers of Massively Parallel Computation., pp. 424-433, 1990.

[58] M.Cupta, P.Banerjee, PARADIGM: A compiler for automatic data distribution on multicomputers, Int. Conf. On Supercomputing 1993, pp 87-96.

[59] J.Dongarra, D.Sorenson, SCHEDULE: Tools for developing and analyzing FORTRAN programs, Tech. Mem. 86, Argonne National Lab, 1986.

[60] A.L.Couch, Seecube user's manual. Tech Rep., CS Dept., Tufts University, 1987.

[61] O.Breuer, J.J.Dongarra, D.C.Sorensen, Tools for aid the analysis of memory access patterns for FORTRAN programs, Tech. Mem., 120, Argonne National Lab. 1988.

[62] A.Bar-Noy, S.Kipnis, Designing broadcasting algorithms in the postal model for message-passing systems, Proc. 4th ACM Symposium on Parallel Algorithms and Architectures, pp. 13-22, 1992.

[63] D.Culler, R.Karp, D.Patterson, A.Sahay, K.E.Schauser, E.Santos, R.Subramonian, T.vonEcken, LogP: Towards a realistic model of parallel computation, Proc. 4th ACM SIGPLAN Symposium on Principles and Practices of Parallel Programming, pp. 1-12, 1993.

[64] L.G.Valiant, Abridging model for parallel computation, Comm. ACM, 33(8) (1990) 103-111.