# EDPEPPS*: A Toolset for the Design and Performance Evaluation of Parallel Applications

T. Delaitre, M.J. Zemerly, P. Vekariya, G.R. Justo, J. Bourgeois, F. Schinkmann, F. Spies, S. Randoux, and S.C. Winter

Centre for Parallel Computing,
Cavendish School of Computer Science,
University of Westminster
115 New Cavendish Street, London W1M 8JS
Email: edpepps-all@cpc.wmin.ac.uk
Web: http://www.cpc.wmin.ac.uk/~edpepps

**Abstract.** This paper describes a performance-oriented environment for the design of portable parallel software. The environment consists of a graphical design tool based on the PVM communication library for building parallel algorithms, a state-of-the-art simulation engine, a CPU characteriser and a visualisation tool for animation of program execution and visualisation of platform and network performance measures and statistics. The toolset is used to model a virtual machine composed of a cluster of workstations interconnected by a local area network. The simulation model used is modular and its components are interchangeable which allows easy re-configuration of the platform. Both communication and CPU models are validated.

## 1 Introduction

A major obstacle to the widespread adoption of parallel computing in industry is the difficulty in program development due mainly to lack of parallel programming design tools. In particular, there is a need for performance-oriented tools, and especially for clusters of heterogeneous workstations, to allow the software designer to choose between design alternatives such as different parallelisation strategies or paradigms. A portable message-passing environment such as Parallel Virtual Machine (PVM) [12] permits a heterogeneous collection of networked computers to be viewed by an application as a single distributed-memory parallel machine. The issue of portability can be of great importance to programmers but optimality of performance is not guaranteed following a port to another platform with different characteristics. In essence, the application might be re-engineered for every platform [26]. Traditionally, parallel program development methods start with parallelising and porting a sequential code on the target machine and

* The EDPEPPS (Environment for the Design and Performance Evaluation of Portable Parallel Software) project is funded by an EPSRC PSTPA Grant No.: GR/K40468 and also by EC Contract Nos.: CIPA-C193-0251 and CP-93-5383.

running it to measure and analyse its performance. Re-designing the parallelisation strategy is required when the reached performance is not satisfactory. This is a time-consuming process and usually entails long hours of debugging before reaching an acceptable performance from the parallel program. Rapid prototyping is a useful approach to the design of (*high-performance*) parallel software in that complete algorithms, outline designs, or even rough schemes can be evaluated at a relatively early stage in the program development life-cycle, with respect to possible platform configurations, and mapping strategies. Modifying the platform configurations and mappings will permit the prototype design to be refined, and this process may continue in an evolutionary fashion throughout the life-cycle before any parallel coding takes place.

The EDPEPPS toolset described here is based on a rapid prototyping philosophy and comprises four main tools:

- A graphical design tool (PVMGraph) for designing of parallel applications.
- A simulation utility (SES/Workbench [24]) based on discrete-event simulation.
- A CPU performance prediction tool (Chronos) which characterises computational blocks within the C/PVM code based on basic operations in C.
- A visualisation tool (PVMVis) for animation of program execution using traces generated by the simulator and visualisation of platform and network performance measures and statistics.

Other tools used in the environment for integration purposes are:

- A trace instrumentation utility (Tape/PVM) [19].
- A translator (SimPVM) [7] from C/PVM code to queueing network graphical representation.
- A modified version of the SAGE++ toolkit [3] for restructuring C source code. In the original version the C files were passed through the C-preprocessor (*cpp*) and then processed by the SAGE++ parser (*pC++2dep*), which creates a parse tree containing nodes for the individual statements and expressions in the code (stored in .dep files). The modification is needed because pC++2dep does not understand preprocessor directives such as #define and #include. Therefore, rules for these directives were added to the grammar of pC++2dep and new node types for them were introduced in the parse tree.
- A translator from existing C/PVM parallel applications into PVMGraph graphical representation (C2Graph) based on the modified SAGE++ toolkit. This is provided in order to allow already written parallel applications to experiment with the toolset.

The advantage of the EDPEPPS toolset is that the cyclic process of design-simulate-visualise is executed within the same environment. Also the EDPEPPS toolset allows generation of code for both simulation and real execution to run on the target platform if required. The toolset is also modular and extensible to allow modifications and change of platforms and design as and when required.

This paper describes the various tools within the EDPEPPS environment and presents a case study for illustration and validation of the models used. In the next section we describe several modelling tools with similar aims to EDPEPPS and we highlight the differences between them. In section 3 we describe the different tools in the EDPEPPS toolset. In section 4 we present results obtained from the case study. Finally, in section 5 we present conclusions and future work.

## 2 Parallel System Performance Modelling Tools

The current trend in parallel software modelling tools is to support all the software performance engineering activities in an integrated environment [22]. A typical toolset should be based on at least three main tools: a graphical design tool, a simulation facility and a visualisation tool [22]. The graphical design tool and the visualisation tool should coexist within the same environment to allow information about the program behaviour to be related to its design. Many existing toolsets consist of only a subset of these tools but visualisation is usually a separate tool. In addition, the modelling of the operating system is usually not addressed.

The HAMLET toolset [23] supports the development of real-time applications based on transputers and PowerPCs. HAMLET consists of a design entry system (DES), a specification simulator (HASTE), a debugger and monitor (INQUEST), and a trace analysis tool (TATOO). However, the tools are not tightly integrated as in the case of EDPEPPS but are applied separately on the output of each other. Also no animation tool is provided.

HeNCE (Heterogeneous Network Computing Environment) [1] is an X-window based software environment designed to assist scientists in developing parallel programs that run on a network of computers. HeNCE provides the programmer with a high level of abstraction for specifying parallelism as opposed to real parallel code in EDPEPPS. HeNCE is composed of integrated graphical tools for creating, compiling, executing, and analysing HeNCE programs. HeNCE relies on the PVM system for process initialisation and communication. HeNCE displays an event-ordered animation of application execution.

The ALPSTONE project [17] comprises performance-oriented tools to guide a parallel programmer. The process starts with an abstract, BACS (Basel Algorithm Classification Scheme), description [5]. This is in the form of a macroscopic abstraction of program properties, such as process topology and execution structure, data partitioning and distribution descriptions, and interaction specifications. From this description, it is possible to generate a time model of the algorithm which allows performance estimation and prediction of the algorithm runtime on a particular system with different data and system sizes. If the prediction promises good performance implementation and verification can start. This can be helped with a skeleton definition language (ALWAN or PEMPI-Programming Environment for MPI), which can derive a time model in terms of the BACS abstraction, and a portability platform (TIANA), which translates the program to C with code for a virtual machine such as PVM.

The VPE project [21] aims to design and monitor parallel programs in the same tool. The design is described as a graph where the nodes represent sequential computation or a reference to another VPE graph. Performance analysis and graph animation are not used here, but the design aspect of this work is elaborate.

The PARADE project [27] is mainly oriented on the animation aspects. PARADE is divided into a general animation approach which is called POLKA, and specific animation developments such as PVM with PVaniM, Threads with GThreads and HPF. This work does not include any graphical design of parallel programs, thus, the predefined animations and views can decrease the user understanding. One of the most important aspects of POLKA is the ability of classification between general and specific concepts.

In the SEPP project [6] (Software Engineering for Parallel Processing) a toolset based on six types of tools has been developed. There are static design tools, dynamic support tools, debugging tools, behaviour analysis tools, simulation tools and visualisation tools [14, 10, 15]. These tools are integrated within the GRADE environment [16, 11]. The GRAPNEL application programming interface currently supports the PVM message passing library.

The TOPSYS (TOols for Parallel SYStems) project [2] aims to develop a portable environment which integrates tools that help programmers cope with every step of the software development cycle of parallel applications. The TOPSYS environment contains tools which support specification and design, coding and debugging, and optimisation of multiprocessor programs. The TOPSYS environment comprises: a CASE environment for the design and specification of applications (SAMTOP) including code generation and mapping support, a multi-processor operating system, a high level debugger (DETOP), a visualiser (VISTOP), and a performance analyser (PATOP). The tools are based on the MMK operating system which has been implemented for Intel's iPSC/2 hypercube. The tools were later ported to PVM in [18]. A more detailed review of parallel programming design tools and environments can be found in [8].

## 3 Description of the Integrated Toolset

The advantages of the EDPEPPS toolset over traditional parallel design methods are that it offers a rapid prototyping approach to parallel software development, allows performance analysis to be done without accessing the target platform, helps the user to take decisions about scalability and sizing of the target platform, offers modularity and extensibility through layered partitioning of the model and allows the software designer to perform the cycle of design-simulate-analysis in the same environment without having to leave the toolset.

Figure 1 shows the components of the EDPEPPS toolset. The process starts with the graphical design tool (PVMGraph), step (1) in the figure, by building a graph representing a parallel program design based on the PVM programming model. The graph is composed of computational tasks and communications. The

tool provides graphical representation for PVM calls which the user can select to build the required design.

The software designer can then generate (by the click of a button) C/PVM code (.c files) for both simulation and real execution. The toolset also provides a tool (C2Graph), step (0), to translate already developed parallel applications onto graphical representation suitable for PVMGraph. The software designer can then experiment with the toolset by changing the parallelisation model or other parameters, such as the number of processors or processor types to optimise the code.

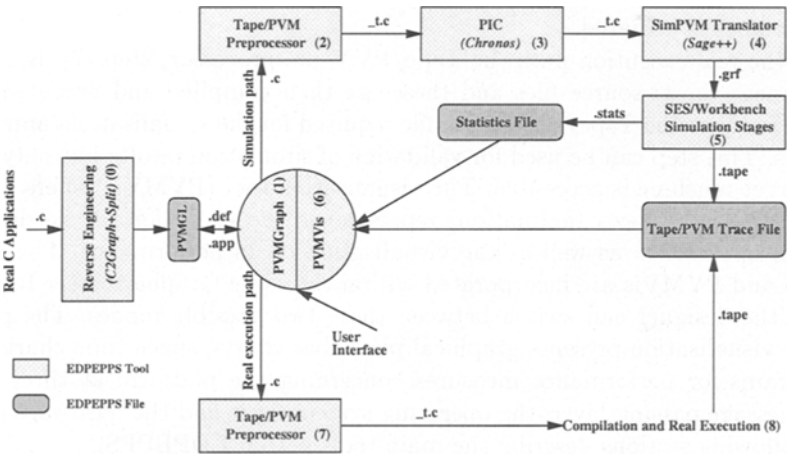


**Fig. 1.** The EDPEPPS Integrated Environment.

In the simulation path each C/PVM source code obtained from the PVMGraph is instrumented using a slightly modified version of the Tape/PVM trace pre-processor, step (2), [19]. The output is then parsed using the Program Instruction Characteriser (PIC), step (3), which forms a part of the CPU performance prediction tool called "Chronos" which inserts *cputime* calls at the end of each computational block. The instrumented C source files are translated using the SimPVM Translator [7], step (4), into a queueing network representation suitable for Workbench graph (.grf file). SES/Workbench, step (5), translates the graph file into the Workbench object oriented simulation language called SES/*sim* [25] using an SES utility (*sestran*). The *sim* file is then used to generate an executable model using some SES/Workbench utilities, libraries, declarations and the PVM platform model. The simulation is based on discrete-event modelling. SES/Workbench has been used both to develop and simulate the platform models. Thus the Workbench simulation engine is an intrinsic part of the toolset. All these simulation actions are hidden from the user and are executed from the PVMGraph window by a click on the simulation button and hence shown in the EDPEPPS environment in Figure 1 in one box under "Sim-

ulation Stages". The simulation executable is carried out by using three input files containing parameters concerning the target virtual environment (e.g. number of hosts, host names, architecture, the UDP communication characteristics and the timing costs for the set of instructions used by Chronos [4]). The UDP model and the instruction costs are obtained by benchmarking (benchmarks are provided off-line) the host machines in the network.

The simulation outputs are the execution time, a Tape/PVM trace file and a statistics file about the virtual machine. These files are then used by the visualisation tool (PVMVis), step (6), in conjunction with the current loaded application to animate the design and visualise the performance of the system. The design can be modified and the same cycle is repeated until a satisfactory performance is achieved.

In the real execution path the Tape/PVM pre-processor, step (7), is used to instrument the C source files and these are then compiled and executed, step (8), to produce the Tape/PVM trace file required for the visualisation/animation process. This step can be used for validation of simulation results but only when the target machine is accessible. The visualisation tool (PVMVis) offers the designer graphical views (animation) representing the execution of the designed parallel application as well as the visualisation of its performance. The PVMGraph and PVMVis are incorporated within the same Graphical User Interface where the designer can switch between these two possible modes. The performance visualisation presents graphical plots, bar charts, space-time charts, and histograms for performance measures concerning the platform at three levels (the message passing layer, the operating system layer and the hardware layer). The following sections describe the main tools within EDPEPPS.

### 3.1 PVMGraph

PVMGraph is a graphical programming environment to support the design and implementation of parallel applications. PVMGraph offers a simple but yet expressive graphical representation and manipulation for the components of a parallel application. The main function of PVMGraph is to allow the parallel software designer or programmer to develop PVM applications using a combination of graphical objects and text. Graphical objects are composed of boxes which represent tasks (which may include computation) and arrows which represent communications. The communication actions are divided into two groups: input and output. The PVM actions (calls) are numbered to represent the link between the graph and text in the parallel program. Also different types and shapes of arrows are used to represent different types of PVM communication calls. Parallel programs (PVM/C) can be automatically generated after the completion of the design. Additionally, the designer may enter PVM/C code directly into the objects. The graphical objects and textual files are stored separately to enable the designer to re-use parts of existing applications [13].

A PVMGraph on-line[1] demonstration is available on the Web.

[1] `http://www.cpc.wmin.ac.uk/~edpepps/demo_html/ppf.html`

### 3.2 PVMVis

The main objective of this tool is to offer the designer graphical views and animation representing the execution and performance of the designed parallel application from the point of view of the hardware, the design and the network.

The animation is an event-based process and is used to locate an undesirable behaviour such as deadlocks or performance bottlenecks. The animation view in PVMVis is similar to the design view in PVMGraph except that the pallet is not shown and two extra components for performance analysis are added: barchart view and platform view. The barchart view shows historical states for the simulation and the platform view shows some statistics for selected performance measures at three levels: the message passing layer, the operating system layer and the hardware layer.

### 3.3 The CPU Performance Prediction Tool: Chronos

The Program Instruction Characteriser (PIC) is called only in the simulation path to estimate the time taken by computational blocks within a parallel algorithm. PIC characterises a workload by a number of high-level language instructions (e.g. float addition) [4] taking into account the effect of instruction and data caches. Assumptions have been made to reduce the number of possible machine instructions to 43 (see [4] for more details on these assumptions). The costs associated with the various instructions are kept in a file in the hardware layer accessible by the SES utilities. These costs are obtained by benchmarking the instructions on different machines.

PIC first parses an instrumented C/PVM program using the modified pC++2dep (mpC++2dep) tool from the SAGE++ toolkit In the second stage, PIC traverses the parse tree using the SAGE++ library and inserts *cputime* calls with the number of machine instructions within each sequential C code fragment.

The *cputime* call is a simple function with a fixed number of parameters (a total of 31). This is different from the number of machine instructions because the instruction cache duplicates some of the instructions (hit or miss).

Each parameter of the *cputime* function represents the number of times each instruction is executed within the sequential C code fragment. The only exception is the last parameter, which determines whether the instruction cache is hit or miss for the code fragment in question.

### 3.4 C2Graph

As mentioned before, this tool allows existing PVM applications to be converted into the EDPEPPS format. The C/PVM application files are first parsed with mpC++2dep to get the .dep files. These files are then traversed by the C2Graph translator using the SAGE++ library.

The translator also takes into account the PVM calls in the original code and generates their corresponding graphical representation in the PVMGraph files.

The translator then determines the master process, positions it with the other tasks by calculating appropriate coordinates for them in the PVMGraph screen, and writes the PVMGraph definition files (.def) for each task. The translator finally writes the application file (.app) required for PVMGraph.

### 3.5 SimPVM Translator

From PVMGraph graphical and textual objects, executable and "simulatable" PVM programs can be generated. The "simulatable" code generated by PVM-Graph is written in a special intermediary language called SimPVM, which defines an interface between PVMGraph and SES/Workbench [7].

To simulate the application, a model of the intended platform must also be available. Thus, the simulation model is partitioned into two sub-models: a dynamic model described in SimPVM, which consists of the application software description and some aspects of the platform (e.g. number of hardware nodes) and a static model which represents the underlying parallel platform.

The SimPVM language contains C instructions, PVM and PVM group (PVMG) functions, and simulation constructs such as computation delay and probabilistic functions.

### 3.6 The EDPEPPS Simulation Model

The EDPEPPS simulation model consists of the PVM platform model library and the PVM programs for simulation. The PVM platform model is partitioned into four layers: the *message passing layer*, the group functions layer which sits on top of the message passing layer, the *operating system layer* and the *hardware layer*. Modularity and extensibility are two key criteria in simulation modelling, therefore layers are decomposed into modules which permit a re-configuration of the entire PVM platform model. The modelled configuration consists of a PVM environment which uses the TCP/IP protocol, and a cluster of heterogeneous workstations connected to a 10 Mbit/s Ethernet network.

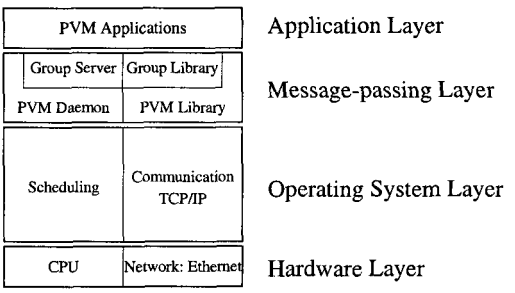


**Fig. 2.** Simulation model architecture.

The message-passing layer models a single (parallel) virtual machine dedicated to a user. It is composed of a daemon, which resides on each host making up the virtual machine, a group server and the libraries (PVM and PVMG), which provide an interface to PVM services. The daemon and the group server act primarily as message routers. They are modelled as automatons or state machines which are a common construct for handling events. The LIBPVM library allows a task to interact with the daemon and other tasks. The PVM library is structured into two layers. The top layer includes most PVM programming interface functions and the bottom layer models the communication interface with the local daemon and other tasks. Only this layer needs to be modified if the Message Passing Interface (MPI [20]) model is to be supported.

The major components in the operating system layer are the System Call Interface, the Process Scheduler, and the Communication Module. The Communication Module is structured into 3 sub-layers: the Socket Layer, the Transport Layer and the Network Layer. The Socket Layer provides a communications endpoint within a domain. The Transport Layer defines the communication protocol (either TCP or UDP). The Network Layer models the Internet Protocol (IP).

The Hardware Layer is comprised of hosts, each with a CPU layer, and the communications subnet (Ethernet). Each host is modelled as a single server queue with a time-sliced round-robin scheduling policy. The communications subnet is Ethernet, whose performance depends on the number of active hosts and the packet characteristics. Resource contention is modelled using the CSMA/CD (Carrier Sense Multiple Access with Collision Detection) protocol. The basic notion behind this protocol is that a broadcast has two phases: propagation and transmission. During propagation, packet collisions can occur. During transmission, the carrier sense mechanism causes the other hosts to hold their packets.

# 4 Case Studies

## 4.1 Bessel Equation

The computational intensive application chosen here to validate the CPU model is the Bessel equation which is a differential equation defined by:

$$x^2 \frac{d^2y}{dx} + x\frac{dy}{dx} + (x^2 - m^2)y = 0$$

When $m$ is an integer, solutions to the Bessel differential equation are given by the Bessel function of the second kind [4], also called Neumann function or Weber function.

The Bessel function of the second kind is translated in C and executed 100000 times.The results for four machines are given in Figure 3. The average error is about 6.13%. The errors for the 486 machine are larger than those of other machines and this is probably due to the fact that there is only one cache used for both data and instructions in the 486 machine. However, for the superscalar

machines, Pentium and SuperSparc, the results are encouraging with errors of 1.7% and 3.4% respectively.

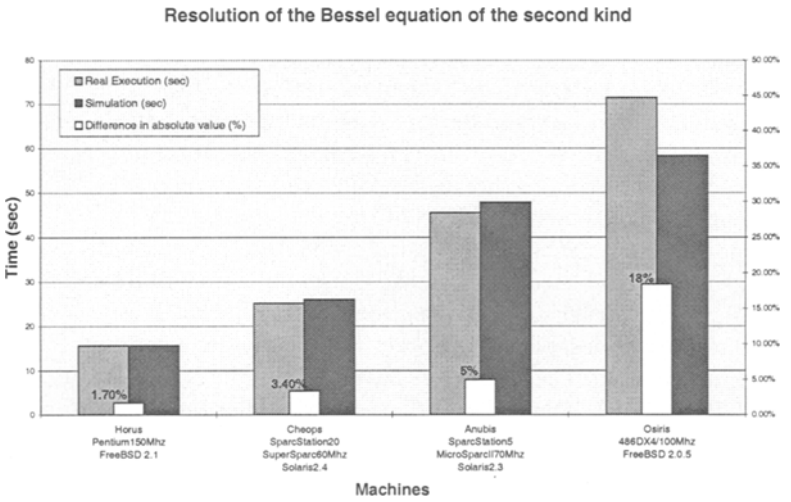


**Fig. 3.** Comparison between real execution and simulation.

## 4.2 CCITT H.261 Decoder

The application chosen here to demonstrate the capabilities of the environment in the search for the optimal design is the Pipeline Processor Farm (PPF) model [9] of a standard image processing algorithm, the CCITT H.261 decoder [9]. Figure 4 shows how the H.261 algorithm decomposes into a three-stage pipeline: frame initialisation (T1); frame decoder loop (T2) with a farm of 5 tasks; and frame output (T3).

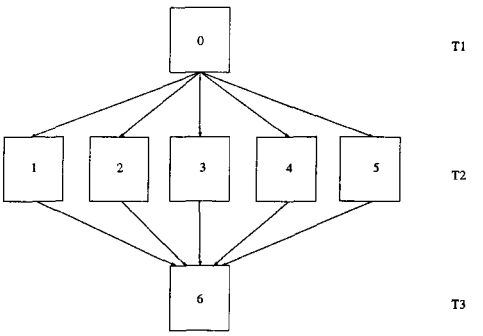


**Fig. 4.** PPF topology for a three-stage pipeline.

The first and last stages are inherently sequential, whereas the middle stage contains considerable data parallelism.

The same topological variation in the PPF model leads directly to performance variation in the algorithm, which, typically, is only poorly understood at the outset of design. One of the main purposes of the simulation tool in this case is to enable a designer to identify the optimal topology, quickly and easily, without resorting to run-time experimentation. Two experiments for 1 and 5 frames were carried out. The number of processors in Stage T2 is varied from 1 to 5 (T1 and T3 were mapped on the same processor). In every case, the load is evenly balanced between processors.

The target platform is a heterogeneous network of up to 6 workstations (SUN4's. SuperSparcs and PC's). Timings for the three computational stages of the algorithm were extracted from [9] and inserted as time delays. Figure 5 shows the simulated and real experimental results for speed-up.

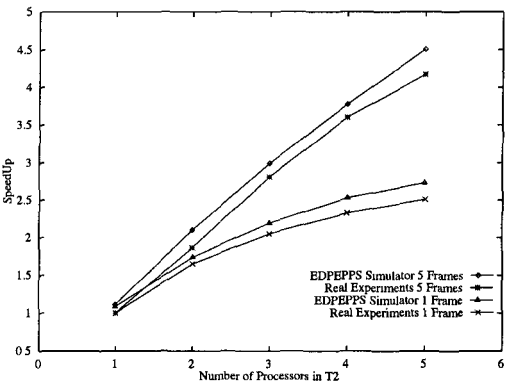


**Fig. 5.** Comparison between speed-ups of simulation and real experiments for the H.261 PPF algorithm.

As expected, the figure shows that the 5-frame scenario performs better than the 1-frame scenario, since the pipeline is fuller in the former case. The difference between simulated and real speed-ups is below 10% even though the PPF simulation results do not include packing costs.

## 5 Conclusion

This paper has described the EDPEPPS environment which is based on a performance-oriented parallel program design method. The environment supports graphical design, performance prediction through modelling and simulation and visualisation of predicted program behaviour. The designer is not required to leave the graphical design environment to view the program's behaviour, since

the visualisation is an animation of the graphical program description. It is intended that this environment will encourage a philosophy of program design, based on a rapid synthesis-evaluation design cycle, in the emerging breed of parallel programmers.

Success of the environment depends critically on the accuracy of the underlying simulation system. Preliminary validation experiments showed average errors between the simulation and the real execution of less than 10%.

An important future direction of our work is to extend the simulation model to support other platforms, such as MPI. The modularity and flexibility of our model will ensure that PVM layer may be re-used where appropriate in the development of the MPI model component. Another planned extension to our environment is the integration of a distributed debugger such as the DDBG [16].

## References


1. A. Beguelin, et. al. HeNCE: A Heterogeneous Network Computing Environment *Scientific Programming*, Vol. 3, No. 1, pp 49–60.
2. T. Bemmerl. The TOPSYS Architecture, In H. Burkhart, editor, *CONPAR90-VAPPIV Conf.*, Zurich, Switzerland, Springer, September 1995, Lecture Notes in Computer Science, 457, pp 732–743.
3. F. Bodin, et. al. Sage++: An Object-Oriented Toolkit and Class Library for Building Fortran and C++ Restructuring Tools, *Proc. 2nd Annual Object-Oriented Numerics Conf.*, 1994. http://www.extreme.indiana.edu/sage/docs.html.
4. J. Bourgeois. CPU Modelling in EDPEPPS EDPEPPS EPSRC Project (GR/K40468), D3.1.6, EDPEPPS/35, Centre for Parallel Computing, University of Westminster, London, June 1997.
5. H. Burkhart, et. al. BACS: Basel Algorithm Calssification Scheme, version 1.1, Tech. Report 93-3, Universität Basel, URZ+IFI, 1993.
6. T. Delaitre, et. al. Simulation of Parallel Systems in SEPP, in: A. Pataticza, ed., *The 8th Symposium on Microcomputer and Microprocessor Applications* 1 (1994), pp 294–303.
7. T. Delaitre, et. al. Final Syntax Specification of SimPVM, EDPEPPS EPSRC Project (GR/K40468) D2.1.4, EDPEPPS/22, Centre for Parallel Computing, University of Westminster, London, March 1997.
8. T. Delaitre, M.J. Zemerly, and G.R. Justo, Literature Review 2, EDPEPPS EPSRC Project (GR/K40468) D6.2.2, EDPEPPS/32, Centre for Parallel Computing, University of Westminster, London, May 1997.
9. A.C. Downton, R.W.S. Tregidgo and A. Cuhadar, Top-down structured parallelisation of embedded image processing applications, in: *IEE Proc.-Vis. Image Signal Process.* 141(6) (1994) 431-437.
10. G. Dozsa, T. Fadgyas and P. Kacsuk, A Graphical Programming Language for Parallel Programs, in: A. Pataricza, E. Selenyi and A. Somogyi, ed., *Proc. of the symposium on Microcomputer and Microprocessor Applications* (1994), pp 304–314.
11. G. Dozsa, P. Kacsuk and T. Fadgyas, Development of Graphical Parallel Programs in PVM Environments, *Proc. of DAPSYS'96*, pp 33–40
12. A. Geist, et. al. *PVM: Parallel Virtual Machine*, MIT Press, 1994.
13. G.R. Justo, PVMGraph: A Graphical Editor for the Design of PVM Programs, EDPEPPS EPSRC Project (GR/K40468) D2.3.3, EDPEPPS/5, Centre for Parallel Computing, University of Westminster, February 1996.

14. P. Kacsuk, P.Dozsa and T. Fadgyas, Designing Parallel Programs by the Graphical Language GRAPNEL, *Microprocessing and Microprogramming* 41 (1996), pp 625–643.
15. P. Kacsuk, et. al. Visual Parallel Programming in Monads-DPV, in: López Zapata, ed., *Proc. of the 4th Euromicro Workshop on Parallel and Distributed Processing* (1996), pp 344–351.
16. P. Kacsuk, et. al. A Graphical Development and Debugging Environment for Parallel Programs, *Parallel Computing*, 22:1747–1770, 1997.
17. W. Kuhn and H. Burkhart. The ALPSTONE Project: An Overview of a Performance Modelling Environment, In *2nd Int. Conf. on HiPC'96*, McGraw Hill 1996, pp 491–496.
18. T. Ludwig, et. al. The TOOL-SET - An Integrated Tool Environment for PVM, In *EuroPVM'95*, Lyon, France, September 1995. Tech. Rep. 95-02, Ecole Normale Superieure de Lyon.
19. E. Maillet, TAPE/PVM: An Efficient Performance Monitor for PVM Applications - User Guide. LMC-IMAG, ftp://ftp.imag.fr/ in pub/APACHE/TAPE, March 1995.
20. M. P. I. Forum. MPI: A Message Passing Interface Standard. *The Int. Journal of Supercomputer Applications and High-Performance Computing*, 8(3/4), 1994.
21. P. Newton, J. Dongarra, Overview of VPE: A Visual Environment for Message-Passing, *Heterogeneous Computing Workshop*, 1995.
22. C. Pancake, M. Simmons and J. Yan, Performance Evaluation Tools for Parallel and Distributed Systems, *Computer* 28 (1995), pp 16–19.
23. P. Pouzet, J. Paris and V. Jorrand, Parallel Application Design: The Simulation Approach with HASTE, in: W. Gentzsch and U. Harms, ed., *HPCN* 2 (1994), pp 379–393.
24. Scientific and Engineering Software Inc. SES/workbench Reference Manual, Release 3.1, Scientific Engineering Software Inc., 1996.
25. K. Sheehan and M. Esslinger, The SES/*sim* Modeling Language, *Proc. The Society for Computer Simulation*, San Diego CA, July 1989, pp 25–32.
26. A. Reinefeld and V. Schnecke, Portability vs Efficiency? Parallel Applications on PVM and Parix, in *Parallel Programming and Applications*, P. Fritzson and L. Finmo eds., (IOS Press, 1995), pp 35–49.
27. J.T. Stasko. The PARADE Environment for Visualizing Parallel Program Executions, Technical Report GITGVU-95-03, Graphics, Visualization and Usability Center, Georgia Inst. of Tech., 1994.