# 26

# The Basel Tool Suite for Parallel Processing

*H. Burkhart, N. Fang, R. Frank, G. Hächler, W. Kuhn, and G. Prétôt*
*Universität Basel*
*Institut für Informatik, Mittlere Strasse 142, CH-4056 Basel, Switzerland.*
*Telephone: +41-61 321 99 67 Fax: +41-61 321 99 15*
*email:* `burkhart@ifi.unibas.ch`

**Abstract**

The Basel approach is characterized by a coordinated set of subprojects that use the same basic terminology but target for different goals. Emphasis is put on software engineering aspects such as increased programmability, portability, and interoperability. This integrated approach has benefits because different user groups (application programmers, performance analysts, students) can interact and profit from synergies by using common system elements. The Basel Tool Suite (developed at PUB*) currently consists of

- ALWAN, the language used for writing algorithmic skeletons, offering reuse-in-the-small constructs, and its compiler which produces portable source code skeletons for different target systems and programming languages.
- PEMPI, a library-based environment supporting structured parallel programming for the MPI message passing standard.
- ALPSTONE, a methodology and environment to make performance prediction and algorithmic benchmarking on different target architectures.

Prototypes for all components exist and are used in courses at the University. Public domain versions of this software are envisaged; potential users should contact the group.

**Keywords**

software engineering for parallel and distributed systems; structured parallel programming; mixed language programming; portability; MPI support; performance prediction; algorithmic benchmarking; parallelism courseware.

*Parallel laboratory, University of Basel

I. Jelly et al. (eds.), *Software Engineering for Parallel and Distributed Systems*

# 1 ALWAN : A PARALLEL COORDINATION LANGUAGE

ALWAN is a parallel language and programming environment developed at PUB. The design goals of ALWAN are to increase the programmability of parallel applications, enable performance portability, support the reuse of software components, and mixed-language programming. Parallel programs consist of (sequential) calculation and (parallel) coordination parts. To address the major difficulties in parallel programming, the ALWAN language provides high level constructs for the description of parallel coordination aspects such as data partitioning and distribution, process topology management and communication aspects. As ALWAN is intended to specify only the coordination of an algorithm, it provides an interface to other, widely used, sequential languages, such as C and FORTRAN. Coordination skeletons and sequential building blocks are processed by the programming environment (ALWAN compiler and support libraries) which can automatically generate programs for various parallel architectures.

## Approach

Program development within the ALWAN environment is outlined in the figure below to which the Roman numerals refer.

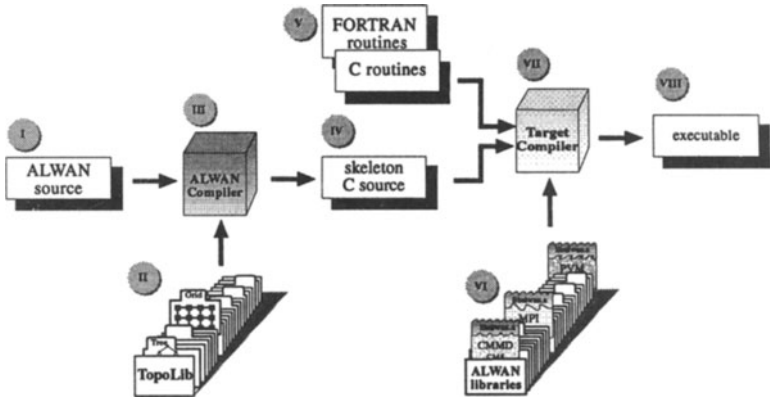


**Figure 1** system overview

A coordination description (I) written in ALWAN is transformed into a source code skeleton (IV) using the ALWAN compiler (III). Predefined ALWAN modules, where frequently used topologies or routines are collected, may be imported and re-used (II). Procedures declared as EXTERNAL define the interface to code written in other (sequential) languages (V). The high-level parallel coordination constructs are translated into ALWAN library (VI) calls with appropriate parameters. This library is implemented for various machines, interfacing to the virtual machine layers available on the given platform. Finally, all code parts are compiled (VII) and linked to form an executable program (VIII). Porting to a different platform only requires a recompilation on the target machine, thus replacing the ALWAN library with the appropriate new one. Compilation of the different source parts (I, IV, and V) and linking to the appropriate libraries (II and VI) is handled by a shell script and generated make files.

## Status and Future Work

The ALWAN compiler is implemented on various UNIX platforms. The full set of library routines has been implemented for PVM. To prove the feasibility of our approach, an intermediate version (no support of collective communication, inhomogeneous data, asynchronous communication, and run time checks) was implemented for PVM, MPI, CMMD, and NX and tested on CM5, SP1, Paragon and a workstation cluster containing NeXT and Sun workstations. The compiler supports mixed-language programming in that the external routines may be written in either C or FORTRAN 77.

A full ALWAN implementation for PVM and MPI will be available in the first quarter of 1996. Further implementations will follow and include support for virtual shared memory systems.

The ALWAN tool suite is currently used within different projects: ALWAN is used within the ALPSTONE performance prediction environment (described in section 3). A library of sample parallel algorithms was built to be used as teachware. This library contains matrix multiplication variants and stencil algorithms (both using a torus topology), bitonic sort using a hypercube topology, divide and conquer sort on a tree, gaussian elimination and transitive closure of graph (Warshall algorithm) both on a farm. Other members of the laboratory use ALWAN to solve application problems (CFD code; image processing for computer-aided surgery). Another project extends ALWAN to support the creation of parallel services for industry standard client server environments.

*Contact Person* : Robert Frank and Guido Hächler ({frank|haechler}@ifi.unibas.ch)
*Home Page* : http://www.ifi.unibas.ch/~alwan
*Reference* : Burkhart, H., Frank, R. and Hächler, G. (1996) Structured Parallel Programming: How Informatics Can Help Overcome the Software Dilemma. To appear in *Scientific Programming, 1996.*

# 2 PEMPI : PROGRAMMING ENVIRONMENT FOR THE MESSAGE PASSING INTERFACE

The PEMPI project helps the programmer in writing message passing programs by using higher abstraction functions and supporting tools. It aims to achieve three goals in a unified approach: *obtain portability by employing the MPI standard, achieve performance through the machine best-fit implementation, and increase programmability by exploiting higher abstractions and taking advantage of supporting tools.*
As a design feature, PEMPI allows the application programmer to work within the context of the higher-level functions to solve regular problems but also to jump into the system level, i.e., MPI level, to solve irregular or performance-critical problems if necessary.

## Architecture and Functionality

PEMPI is a two-layer architecture: Kernel PEMPI (KPEMPI) and Outer PEMPI (OPEMPI), as Figure 2 shows. Kernel PEMPI contains the higher abstraction library based on MPI, which can be further divided into three modules: (1) *Process Manage-*

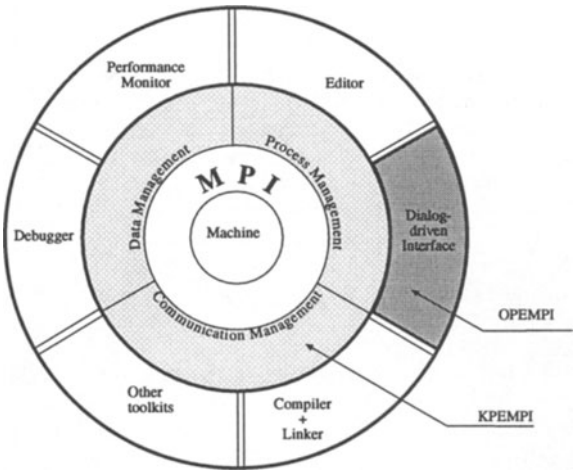


**Figure 2** PEMPI Two-layer Structure

*ment Module*: containing functions to build process topologies such as a tree or hypercube besides the topologies supported in MPI, and functions to access topology information; (2) *Data Management Module*: containing functions to input/output and distribute/collect data according to the layout of the data partition which is specified by the programmer as well as functions structuring commonly-used data components before they are sent/received; (3) *Communication Management Module*: containing some topology-specific communication routines.

Auxiliary functions such as index conversion (global to local and reverse) of distributed data are also supported.

Outer PEMPI consists of a collection of tools: (1) a dialog-driven programming interface: a template generator with GUI is used to generate code related to process topology creation, data layout specification, data I/O and distribution/collection etc; (2) a text editor; (3) target system utilities: including a compiler, linker and debugger; (4) toolkits: e.g. for automatically converting C structs and unions to MPI Datatypes.

## Status and Future Work

KPEMPI has been implemented and has partially been tested and ported to the IBM SP-2 at Argonne National Laboratory and the Fujitsu AP1000 at Imperial College of Science, Technology and Medicine/Fujitsu Parallel Computing Research Center, University of London. The prototype of OPEMPI is still under construction using the NEXTSTEP utilities Project Builder and Interface Builder.

Future work includes: (1) implementing the full OPEMPI prototype using NEXTSTEP; (2) porting KPEMPI to other parallel machines; (3) implementing sample parallel algorithms in PEMPI and also real-world applications such as CFD, image processing, etc; (4) porting OPEMPI to other user interfaces, e.g., OSF/Motif; (5) detailed analysis of performance loss caused by abstraction on different parallel machines and optimization of the code.

*Contact Person* : Niandong Fang (fang@ifi.unibas.ch)
*Home Page* : http://www.ifi.unibas.ch/~pempi
*Reference* : Fang, N. and Burkhart, H. (1994) PEMPI—From MPI Standard to Programming Environment. *Proceedings of Scalable Parallel Libraries Conference II (SPLC'94)*, 31-38, Mississippi.

# 3 ALPSTONE : PREDICTION INCREASES PRODUCTIVITY

Beside correctness, portability, and reusability, efficiency is one of the most important questions to be answered when developing software for a parallel system. However, developing a program, running it and finding out that it is too slow is a bad approach. We need an early estimation of a program's performance to increase the programmer's productivity.

ALPSTONE's methodology guides a skeleton programmer in going from a formal description of an algorithm to a finished program from the point of view of performance studies and supplies an experimental algorithm performance test bed. Techniques included are benchmarking and performance prediction. Fig.3 shows an overview of the architecture.

## Approach and architecture

Early in the software engineering phases, a parallel program is modeled by a macroscopic abstraction (I) containing program properties such as process topology, execution structure, data descriptions, I/O behaviour, and interaction specifications. This information may be refined during the software engineering process where necessary.

The *Extractor* (II) now derives a time model and may (later) generate a rudimentary, instrumented skeleton program (e.g. ALWAN or PEMPI). The user has to supply additional information, e.g. code for a local computation or refinements of the skeleton.

The generated time model predicts the program performance in terms of the abstraction. The *Estimator* (III) supports the performance prediction steps through the whole software engineering cycle by calculating the model's runtime and supporting its refinement. The estimator detects idle times or allows the overlapping of computation and interaction. Other information is available as well, such as the amount of certain operations or transferred data. Statistical models for runtimes and the modelling of an inhomogeneous system are supported. In order to be as accurate as possible, a library of performance data representing topology creation, data distribution times, etc is mandatory (VI).

If the prediction promises good performance, program implementation and translation language will follow (IV). Otherwise, modifications will be done. This early detection of performance faults is important as this is where programmer productivity can be increased.

After having run an instrumented program, an *Analyzer* (V) stores measurements (for reuse in other estimations) in the performance database. Of course, comparing the predicted and measured runtime is of interest.

The ALPSTONE *benchmark suite* (VI) supplies the estimator with data from selected benchmarks. We include some well known codes, but define some new benchmarks as well.

The approach is hierarchical because the building blocks of a parallel program will often be found in complex routines composed of actions from the lower levels, in that way detailed measurements of building "blocks" may refine first estimations based on the "base" level. The suite is vertically structured using the basic properties modeled in the performance skeletons.

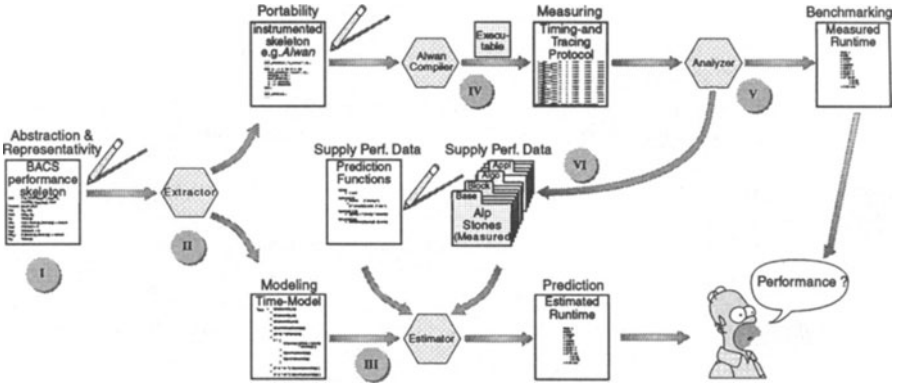


**Figure 3** An overview of the ALPSTONE environment

## Status and future work

First case studies showed that it is worth integrating performance studies in the software engineering cycle while formulating an outline of a program. Today, most of the macroscopic specification and test cases are defined. The ALPSTONE benchmark suite is specified in three layers and most benchmarks are implemented on several machines. The *Extractor* is currently realized using a compiler toolbox. The *Estimator*, defined as a layered system, and the *Analyzer* are running in the Mathematica environment on a subset of the generated models. Finally, an instrumentation library has been written in C.

We will finish the implementation, port our benchmarks to different systems and predict more skeletons in the next steps. Directions where ALPSTONE can be used or extended are: a suitability study of algorithms on different architectures, a support for load balancing strategies, the improvement of the estimation for network resources, or studying the behaviour of implementation strategies without the need of accessing a parallel system.

*Contact Person* : Walter Kuhn (kuhn@ifi.unibas.ch)
*Home Page* : http://www.ifi.unibas.ch/~alpstone
*Reference* : Kuhn, W. and Burkhart,H. (1995) The ALPSTONE project: An Overview of a Performance Modeling Environment. Proceedings of the Conference on High Performance Computing (HiPC'95), New Delhi, 1995.