29

Problem-Solving on Scalable Parallel Systems Using Application Specification and Reusable Software Components

Karsten M. Decker, Jiri J. Dvorak, and René M. Rehmann
Swiss Center for Scientific Computing (CSCS/SCSC)
Via Cantonale, CH-6928 Manno, Switzerland
E-mail: {decker,dvorak,rehmann}@cscs.ch

Abstract

From the application user's point of view, ease of programming of distributed memory parallel systems has not been achieved yet. It is the purpose of this paper to demonstrate how these limitations can be overcome by our Program Development Environment PDE currently under development. The environment features a problem-oriented specification formalism and is based on a skeleton- and template-oriented application development methodology. The large set of fine-grain algorithmic skeletons and templates used in the system provides the basis for a software reuse mechanism and is managed with a knowledge-based component. Skeletons are completed with computational components by means of automatic program synthesis techniques. We present our current results, summarize the major achievements, such as software reuse and portability, and give an outlook on future research directions and related publications.

1 INTRODUCTION

Despite remarkable progress in hardware and software technology for Distributed Memory Parallel Processor (DMPP) systems over the last few years (for an overview of the state-of-the-art in parallel programming, we refer to (Decker, Dvorak, Rehmann & Rühl 1995)), simple development of scientific and engineering applications has not yet been realized.

To improve the unsatisfactory situation, it is the goal of our research to develop an easy-to-use application engineering environment (problem-solving or programming environment in the following) for the synthesis of new application software in the scientific and engineering sector in a user-centered and application driven way. Four key characteristics describe and position our approach. First, application-oriented problem description formalisms serve to focus on *what* the problem is and which computational methods should

Project funded by the Swiss National Science Foundation (SNF) in the framework of the Swiss Priority Program Informatics (SPPIF), Grant-No. SPPIF-5009-034402

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

be used to solve it. Second, the use of design skeletons and templates provides a software reuse mechanism and hides the difficult parts of programming DMPPs while ensuring good scalability, (efficiency-preserving) portability, and parallel efficiency. Third, interactive guidance supports exploitation of user knowledge as completely as possible. Finally, automatic program synthesis techniques ensure a transparent coding process.

Based on the recent success of parallel systems in the business sector, where familiar application development interfaces are applied and parallelism is offered transparently, we believe that a descriptive formalism is essential. Algorithmic skeletons or similar structures were proposed in the past for software engineering and reuse (Waters 1982) as well as for parallel computations (Cole 1989). The basic idea underlying these approaches is to encode reusable structural characteristics of algorithms in skeletons. A skeleton typically contains open, i.e., *generic*, parts that have to be filled in to adapt the skeleton to the given situation and to get a complete algorithm or algorithm component.

It is the purpose of this paper to outline the achievements of our high-level *Program Development Environment* (PDE) and to report on future research directions. For a detailed description of the design objectives, the underlying application development methodology, and an assessment of the methods we use, the interested reader is referred to (Decker, Dvorak & Rehmann 1994*b*).

2 RESULTS

2.1 A Programming Environment for Stencil-based Problems

In the spirit of a rapid prototyping approach to test system functionality, we started our research with feasibility studies for the very simple class of stencil-based problems (Decker, Dvorak & Rehmann 1994*a*) characterized by the operation of a local computational stencil on n-dimensional grids. Applications in this class include, for instance, the restoration of gray-scale images with different smoothing operators and the solution of partial differential equations according to the finite difference method.

We illustrate the functionality of the PDE for this problem class with the solution of the Poisson equation on a simple two-dimensional rectangular grid with periodic boundary conditions. The Laplace operator is approximated by the nearest-neighbor, symmetric 5-point stencil and we use a Gauss-Seidel algorithm with two colors to accomplish the iterative solution.

The declarative problem description can be done graphically with the *Stencil Modeling Programming Assistant Interface* (SMPAI, Fig. 1) which is then translated into the textual *Stencil Problem Specification Language* (SPSL) description shown in Fig. 2. If this is preferred by the user, the programming task can start directly in SPSL.

As can be seen, the problem description consists of the specification of the problem type, the geometry of the problem domain, the size and dimensionality of the grid, the structure of a grid cell, the boundary conditions for each physical boundary of the grid, the computational stencil, the numerical method, and the domain decomposition scheme. Since SPSL realizes a complete problem description (Roth 1993), there is no further user interaction with the PDE required.

The PDE now successively transforms the SPSL problem description into a compil-

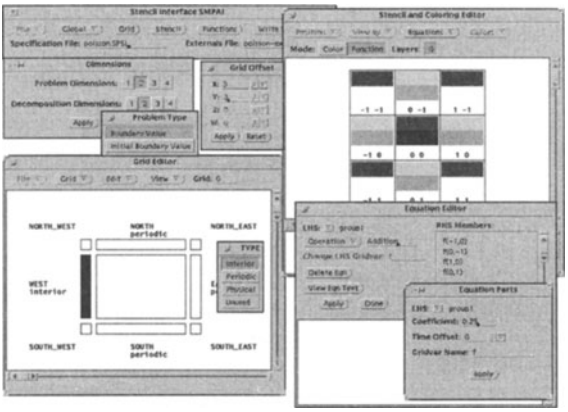


Figure 1 The graphical user interface of the SMPAI.

```
grid_spec {
  grid_offset = [0,0];
  grid_size = {512,512};
  boundary_grid {
    boundary_grid_type = PERIODIC_BND;
  } SOUTH;
  boundary_grid {
    boundary_grid_type = PERIODIC_BND;
  } NORTH;
  .......
};
stencil_spec {
  stencil_action = {
    f<[0,0]> = 0.25*(f<[-1,0]> + f<[0,-1]> + f<[1,0]> + f<[0,1]>)
               - f_rho<[0,0]>
             };
};
```

Figure 2 SPSL description of the programming example.

able program. The *Programming Assistant* (PA) first reads the problem representation and starts the reasoning process with the goal to find the most appropriate algorithmic skeleton for the given problem. The reasoning process is a rule-driven descent in the skeleton hierarchy, based on the information given in the problem specification and various rule bases maintained by the PA. Expertise about the application domain, parallel programming, and software engineering are encoded in the rules that control this skeleton selection. Knowledge about characteristics of different hardware platforms will be added

in a future prototype. Together with the specification of the problem at an abstract level, the hardware knowledge integrated in the skeleton selection will ensure portability at the algorithmic level, above the level of general-purpose programming languages.

The chosen skeleton contains all information needed for the generation of a parallel framework suitable for the problem under consideration. In particular, it defines the data distribution scheme and the communication and synchronization structure. Based on the chosen skeleton and the problem representation, the PA produces two different outputs for the two-component structure of the *Program Synthesizer* (PS).

One part of the final code generation step is done by the TINA skeleton generator (Gutzwiller 1993), which generates the C-code for setting up the process topology, data distribution and communication calls for different message-passing interfaces, and calls the computational components.

The computational components are generated by the other component of the program synthesizer PS (Rehmann 1994). This component starts from an abstract definition of the computational units of the application and the definition of the grid structure. Using this input, it generates code for the function which calculates the user-defined computational stencil, the functions for filling and scattering the communication buffers from and to the grid, and the function calls for the various types of boundary conditions.

2.2 Towards Programming of General Data-parallel Problems

From the application user's point of view, the problem domain of stencil-based applications discussed in Sect. 2.1 is of rather low importance. To qualitatively increase the usefulness and attractivity of the PDE, our most recent and current research is concerned with a major step towards supporting the programming of general data-parallel problems which are of real practical interest to the scientific user community.

To achieve this goal, we follow a step-wise approach. Analyzing user requirements and identifying the important components of real applications, we first focus on problems resulting in the formulation of linear algebra operations. Within this problem class, we concentrated on developing a programming environment for iterative solvers for general linear systems.

The problem class of iterative solvers for general linear systems has very different characteristics than the class of stencil-based problems: the problem description may be incomplete, problem realization may require more than one algorithmic skeleton or template with a fixed parallel structure, and consequently, the careful design of data structures becomes crucial.

To realize a programming environment for this problem class, all three functional components of the PDE, i.e., the PAI, the PA, and the PS, need to be reconsidered and enhanced. In this paper, we report on three activities: the development of the specification language for the problem class of iterative solvers, a first prototype for the corresponding *Data Modeling Programming Assistant Interface* (DMPAI), and a related prototype of the PA.

An essential part of the DMPAI is the underlying declarative problem specification language. The *Basic Language for Iterative, Parallel Solvers* (BLIPS) (Toupin 1994) has a Pascal-like syntax, provides support for intrinsic and user libraries, supports abstract property specification to describe problem characteristics, and has dynamic language support. The latter characteristic of BLIPS allows the user to define and adapt the language

```
template solve(A : matrix; x, y : colvector);
  A is symmetric and pos_definite;
  for solve A * x = b;
    do
      { implementation of the solver algorithm (e.g., CG)}
    end do;

template solve(A : matrix; x, y : colvector);
  A is non_symmetric and transp_not_avail and storage_limited;
  for solve A * x = b;
    do
      { implementation of the solver algorithm (e.g., BiCGSTAB)}
    end do;

procedure main;
  var   A: matrix;
      x,b: colvector;
  where A is symmetric and pos_definite;
  do
    read A from "pde.mat";
    read b from "pde-b.vec";
    solve A*x=b;
    write x to "pde_soln.vec";
  end do;
```

Figure 3 Definition of a linear solver in BLIPS.

according to his specific needs. An example showing how to define a linear solver for a specific type of matrix is given in Fig. 3.

The most recent prototype contains the static knowledge base for the PA, i.e., templates implementing different algorithms for parallel iterative solvers. These templates are wrappers for a library of parallel iterative solvers developed in another project at CSCS. Additionally, we have extended the dynamic knowledge, i.e., the rules to make use of the correct templates for a specified application and we have developed a user interface that allows the specification of an application and the editing of new or existing templates. As the templates only contain calls to library functions, no sophisticated program synthesizer is needed.

3 FUTURE DIRECTIONS

Our future research will concentrate on improved programming environments for the class of data-parallel programs. We intend to successively relax the constraints on the supported application spectrum currently imposed. Formalization of the large amount of application knowledge to enhance the rule base of the PA will be crucial to ensure the long-term success of these systems.

An important topic which will be investigated is which requirements the user dialog with the PDE must satisfy to guarantee successful interaction, respecting (and taking

advantage of) the conceptual models, knowledge structures, and working processes of our target user community, i.e., application users.

Another subject of research is the development of a hardware knowledge base with corresponding hardware-dependent algorithmic skeletons and templates. Together with appropriate rules to choose the correct skeleton or template for a given hardware, it will be possible to generate program code which runs optimally on specific hardware. This mechanism ensures true, i.e., efficiency-preserving portability across hardware platforms as the application-oriented problem description needs not be changed to optimally run an application on different hardware platforms.

A further important topic which we believe should be addressed in the near future is teaching effective usage of DMPPs. Here we envisage machine-assisted learning which could be realized more or less easily by suitable enhancements of our PDE.

Finally, the supported application spectrum should be broadened further, in particular with support for non-numerical problems.

REFERENCES

Cole, M. (1989), *Algorithmic Skeletons: Structured Management of Parallel Computation*, Research Monographs in Parallel and Distributed Computation, The MIT Press, Cambridge, MA, USA.

Decker, K. M., Dvorak, J. J. & Rehmann, R. M. (1994*a*), A Knowledge-based Scientific Parallel Programming Environment, *in* K. M. Decker & R. M. Rehmann, eds, 'Working Conference on Programming Environments for Massively Parallel Distributed Systems', Birkhäuser Verlag, Basel, pp. 127–138.

Decker, K. M., Dvorak, J. J. & Rehmann, R. M. (1994*b*), User-driven development of a novel programming environment for distributed memory parallel processor systems, *in* 'Priority Program Informatics Research Information Conference Module 3 Massively Parallel Systems', Swiss National Science Foundation, pp. 40–47.

Decker, K. M., Dvorak, J. J., Rehmann, R. M. & Rühl, R. (1995), 'Matching User Requirements in Parallel Programming', *Future Generations Computer Systems.* accepted for publication.

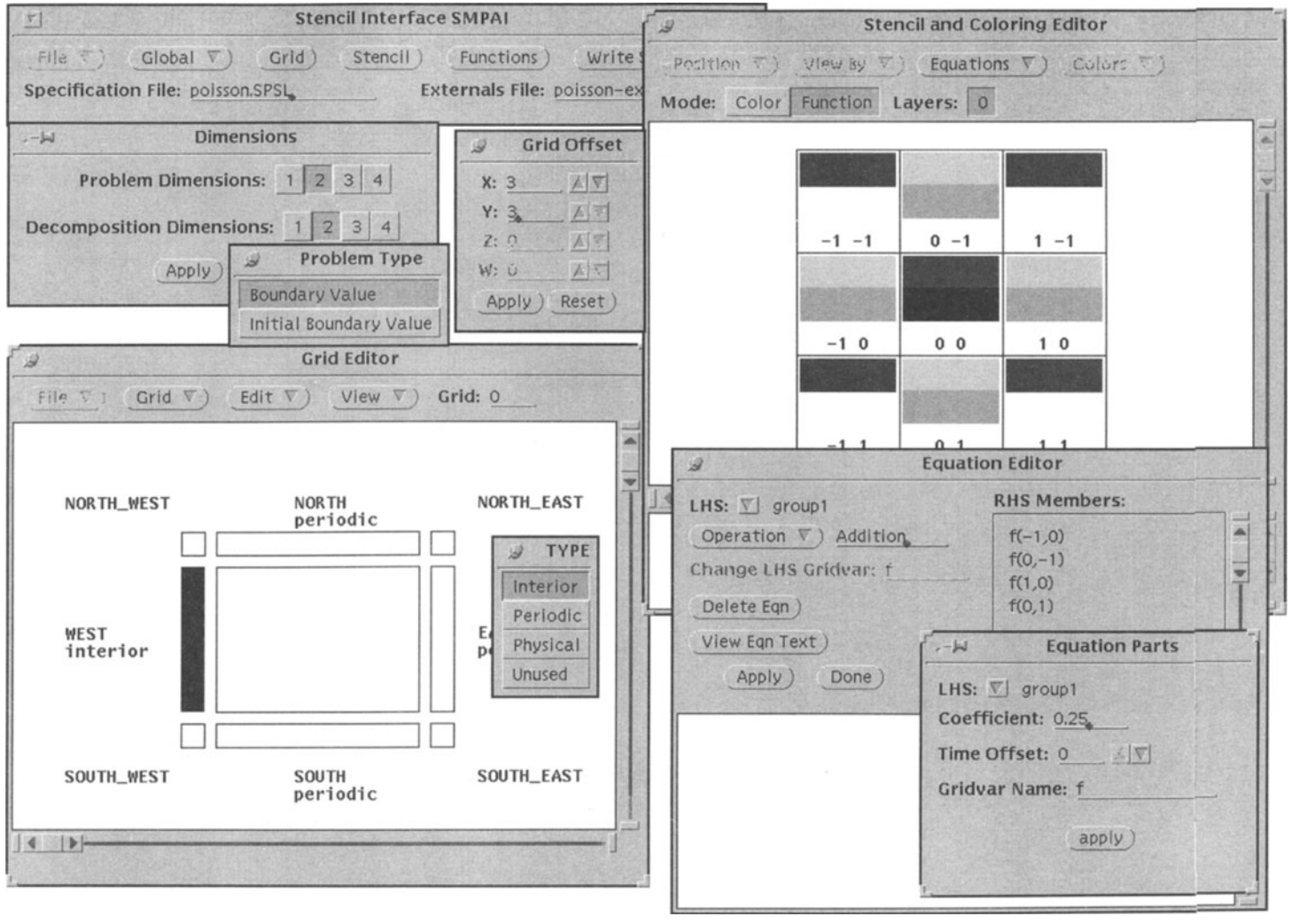
Gutzwiller, S. (1993), Werkzeuge und Methoden des skelettorientierten Programmierens von Parallelrechnern, PhD thesis, University of Basel. In German.

Rehmann, R. (1994), Automatic Generation of Programs for a Scientific Parallel Programming Environment, Technical Report CSCS-TR-94-02, Centro Svizzero di Calcolo Scientifico, CH-6928 Manno, Switzerland.

Roth, M. (1993), Generation of Algorithmic Skeletons from Stencil Specifications, Master's thesis, IAM, University of Bern. In German.

Toupin, T. (1994), **BLIPS** Language Specification Proposal, Technical Note SeRD-CSCS-TN-94-09, Swiss Scientific Computing Center, CH-6928 Manno, Switzerland.

Waters, R. C. (1982), 'The Programmer's Apprentice: Knowledge Based Program Editing', *IEEE Trans. on Software Eng.* **SE-8**(1), 1–12.


Stencil Interface SMPAI
File
Global
Grid
Stencil
Functions
Write
Specification File: poisson.SPSL
Externals File: poisson-ex
Dimensions
Problem Dimensions: 1 2 3 4
Decomposition Dimensions: 1 2 3 4
Apply
Problem Type
Boundary Value
Initial Boundary Value
Grid Offset
X: 3
Y: 3
Z: 0
W: 0
Apply
Reset
Grid Editor
File
Grid
Edit
View
Grid: 0
NORTH_WEST
NORTH
periodic
NORTH_EAST
WEST
interior
TYPE
Interior
Periodic
Physical
Unused
SOUTH_WEST
SOUTH
periodic
SOUTH_EAST
Stencil and Coloring Editor
Position
View By
Equations
Colors
Mode: Color Function
Layers: 0
-1 -1
0 -1
1 -1
-1 0
0 0
1 0
-1 1
0 1
1 1
Equation Editor
LHS: group1
Operation
Addition
Change LHS Gridvar: f
Delete Eqn
View Eqn Text
Apply
Done
RHS Members:
f(-1,0)
f(0,-1)
f(1,0)
f(0,1)
Equation Parts
LHS: group1
Coefficient: 0.25
Time Offset: 0
Gridvar Name: f
apply