Problem Solving Environment Infrastructure for High Performance Computer Systems

Daniel C. Stanzione, Jr. and Walter B. Ligon III

Parallel Architecture Research Lab
Clemson University
dstanzi@clemson.edu
http://www.parl.clemson.edu/

Abstract. This paper presents the status of an ongoing project in constructing a framework to create problem solving environments (PSEs). The framework is independent of any particular architecture, programming model, or problem domain. The framework makes use of compiler technology, but identifies and addresses several key differences between compilers and PSE. The validity of this model is being tested through the creation of several prototype PSEs, which apply to significantly different domains, and target both parallel computers and reconfigurable computers.

1 Introduction

A number of approaches have been taken to make it simpler to create applications for High Performance Computing (HPC) systems. Existing programming languages have been extended with parallel constructs, notably PVM and implementations of the Message Passing Interface(MPI). The resulting systems remained more difficult to program than sequential computers, and have been referred to as the "assembly language" of parallel computing. New languages have been created in which parallelism is inherent, but to date none of these languages have seen widespread adoption.

In recent years, more and more interest has been paid to the idea of creating problem solving environments (PSEs) for high performance computers. While a number of prototype PSEs have been created, the construction of these PSEs remains largely an ad hoc procedure. The PSE community has repeatedly made calls for infrastructure to be developed which supports the creation of PSEs.

This paper describes ongoing research in creating just such an infrastructure. Presented here is an architecture for the creation of PSEs. The architecture uses a layered model which provides abstractions for the hardware, programming model, mathematical model, and whatever science models are used by a particular problem solving environment. The layered architecture employs a model somewhat similar to an open compiler. The goal is to produce PSEs that provide abstraction bridges, i.e. the PSEs can be used by both computer scientists and users in the domain of the PSE, but each sees the environment through a

different level of abstraction. Another goal is to decouple the specification of the application from the target architecture in order to make applications at least somewhat portable between widely varying types of high performance computing systems.

The following sections describe the proposed PSE architecture in more detail. An implementation of the infrastructure proposed by the model is described, followed by a brief look at two PSEs under construction employing this infrastructure which are used to validate the model.

2 The Proposed Model for Problem Solving Environments

This section proposes a model for PSE construction which draws heavily upon compiler technology and concepts from software engineering and object-oriented programming to create a framework from which effective PSEs can be constructed. The fundamental structure of this model is a number of independent tools which interact through a shared open representation of the design in question. A simple block diagram of this model is shown in fig. 1. The tools which access and act on the design are known as *agents*. The design itself is stored in the *Algorithm Description Format* (ADF). The role of the *manager* is to coordinate the actions of the agents, and load and store ADF designs from the *library*.

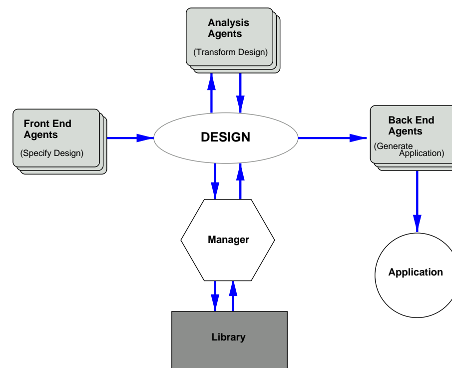

Fig. 1. Basic Structure of an Environment

The basic idea is similar to that of a traditional compiler. A compiler is in essence a set of tools that make a series of transformations on a user's design to transform it into a running application, typically using one or more internal formats to store the design. Agents in this new environment perform many of the same functions. They are involved with specification, analysis, and transformation of designs, and for generating application code from the design. A

shortcoming of the compiler model is that once constructed, compilers are difficult to extend. A primary cause if this is that the compiler's internal formats are almost always mysterious. This model attempts to remedy this problem by making the internal format used by all the agents to store, analyze, and transform the design a well-documented open format, ADF. In this sense this architecture can be considered an open compiler for PSEs.

In addition to the open format making the PSE extensible, this model differs from a compiler in three significant ways. A traditional compiler represents an individual program at one level of abstraction; the source language. While some compilers support more than one language for the same back end, this model differs substantially in that it can support multiple *simultaneous* abstractions to represent the same problem. Each agent can present a different abstraction of the same design, and allow the user to interact with only the aspects of the design important to that user or to the task being performed by that particular agent. A second significant difference between PSEs and traditional compilers is that the internal format of compilers represents only information about the computation itself. In the PSE model, the internal format can be used to store many other types of additional information other than what is considered the traditional source program. ADF can be used to represent the data flow within the computation, information about the architecture the application can or has been run on, performance information and results from past runs, monitoring information, higher level functional descriptions of the problem, etc. This flexibility allows the internal format to be used by many of the tools expected in a PSE that are not traditionally part of a compiler, such as experimental and runtime management, steering and monitoring tools.

The third significant difference with the traditional compiler model is that ADF is dynamic. In a traditional compiler, once compilation takes place, the executable is independent of the source specification. In PSEs constructed with this model, the job of the environment is not necessarily complete once the application is generated. Agents can monitor the running design, and use the design description provided by the internal format to back annotate information about the execution of the application. The dynamic nature of the design and the way agents can interact and iterate with it give the PSE the ability to manage the application throughout it's entire life cycle.

2.1 The Layered Architecture

If the model above is accepted as the way in which PSEs are to be constructed, what remains is to provide the integration method through which the various agents in the PSE can interact. This problem is addressed through the creation of a layered architecture, outlined in fig. 2. Agents that exist within a layer share a set of common attributes which exist on the designs on which they operate. Definition of a set of attributes for a particular layer defines that layer of the environment. These agreed upon attributes allow the agents within a layer to interact. The clear definition of attribute sets at each layer also provides the facility for reuse, as agents written for a particular environment can be reused

in another as long as that agent's subset of the design attributes exist in each environment.

There are 5 different layers which provide 5 levels of abstraction. In multi-physics environments, or environments targeted at multiple architectures, there may be multiple implementations of particular layers existing concurrently within a single environment. The abstraction hierarchy provides a two-way abstraction, with details about the computation and the computer hidden as you move up the hierarchy, and details about the problem domain and mathematics hidden as you move down.

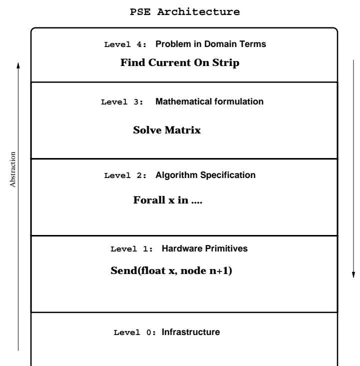

Fig. 2. PSE Architecture

2.2 Level 0 - Infrastructure

Level 0 is the common or kernel layer, and provides the infrastructure necessary for the creation of any PSE targeted at any domain. Level 0 should provide the following:

- An open, shared, attributed graph format
- A database system for managing and maintaining libraries of designs
- A model for agents (both interactive and autonomous) to interact with the designs and with each other
- A mechanism for creating attribute sets and components

The internal format represents a directed attributed graph. Each graph is composed of an attribute list, and a set of nodes connected by links. Each node consists of an attribute list, and lists of input and output ports. Each of the ports has its own attribute list as well. There are no restrictions placed on attributes in any of the lists; there can be any number of attributes, and they can contain any kind of information. This provides tremendous flexibility in what ADF can

be used for. Support for hierarchy need not be explicitly provided, as it can be done by specific environments via the attribute mechanism.

In addition to the design format, an agent API is provided. The API provides mechanisms to create and modify all portions of the graph, and to locate, store and sort the graphs in an environment. The API provides a method for searching designs via any attribute or attribute/value pair within the design. Furthermore, the API allows agents to share an open design, and provide a method for agents synchronize their changes to designs.

A core set of attributes are assigned in level 0 that make it possible to create named attribute sets for use in higher levels, and to check designs and pieces of designs for conformance with particular sets. The attributes on the ports function as an interface description for the design. If two designs comply to the same attribute set, and the port specifications match, then those two designs may be connected. The attribute set mechanism in conjunction with ports and the layered architecture form an extremely versatile component system. When two components are connected using this layered approach, the semantic correctness, mathematical correctness, and if the connection makes sense in terms of the problem domain are all checked by examining the attribute set at each layer.

The attribute set and layered architecture also serve to provide a bridge between abstractions. Users working on the design at any level can work independently of those on other layers. Level 0 provides services similar to many of the distributed object frameworks such as CORBA [2], the Common Component Architecture [3], or DCOM [4]. Level 0 attempts to provide the same support for PSEs equivalent to that which SUIF [1] provides for compilers

2.3 Level 1 - Hardware Abstractions

Level 1 is the "back-end" layer. At this level, abstractions are provided to describe the hardware in a particular HPC target system, and the basic constructs for implementing designs on that system. Level 1 seeks to provide abstractions at the level of an "assembly language" for the architecture in question. Level 1 provides the following services and abstractions:

– A representation of any distinctive characteristics of the compute platform
– Attributes suitable to describe applications on the target system
– A component model for composing code fragments
– Basic data types
– Low-level optimization, analysis, debugging, and code generation agents
– Resource management services
– A library of design constructs

A level 1 implementation for a cluster computer would provide the ability to describe and generate message-passing programs (or, at the very least, collections of processes capable of communication through the cluster's network), the ability to schedule the nodes on the cluster and dispatch jobs, program debugging services, and performance visualization (Data visualization would not

be included here, as that would entail knowledge of the context of the problem; however, visualization of processor or memory utilization would be appropriate). A level 1 implementation for a configurable computing target might include a set of hardware macros for basic operations, a set of attributes which described the interconnection of these modules, and agents to perform code generation ,optimization, placement, and routing.

Many existing pieces of software provide services at an abstraction layer suitable for level 1, including message passing or shared memory services such as those provided by PVM [5], or implementations of MPI or OpenMP. The components at this level operate with a level of abstraction similar to that presented in the Interface Description Language [6]. Level 1 would also be the appropriate place to employ metacomputing software.

2.4 Level 2 - Programming Model

Level 2 is the programming model layer. At this level, abstractions are provided which operate at approximately the level of most parallel programming languages. This is accomplished by enforcing some programming style in a somewhat machine-neutral way, for example data parallelism, task parallelism, or object parallelism. An attribute set is required to represent more sophisticated data structures and concepts such as parallel tasks, communication patterns, *Forall* loops, and data distributions.

Agents should be created at this level which provide an interface suitable for use by a good parallel programmer. This level is also the appropriate place for source level optimizations performed by many source-to-source compilers to take place.

The abstraction presented at this level is roughly equivalent to that presented by many parallel languages and programming environments, such as HPF [13], Mentat [7], Data Parallel C [13], Enterprise [10],or Hence [11].

2.5 Level 3 - Mathematics

Level 3 is the layer at which mathematical abstractions are provided. It moves an abstraction level above that of dealing with programs to the level of dealing with a particular class of problems. Typical level 3 abstractions would include geometry, discretization, data range information, and linear algebra constructs such as vectors and matrices.

At level 3, the environments begin to become more customized to a particular problem domain, so not every environment will necessarily include all the abstractions listed above. Services provided at level 3 via agents might include an equation-based interface to the user, or an agent which presents a choice of linear system solvers to apply to the problem. The separation of the mathematical specification of the problem from the science specification also makes it simpler to apply techniques like adaptive mesh refinement to the problem without having to deal with the specifics of the science and unfamiliar terminology. It also

allows for some simple sanity checking (i.e. matrix conformity) to be done in a straightforward way.

Level 3 provides a level of abstraction roughly equivalent to that provided by some of the ultra high level languages or general purpose PSEs currently in existence, such as Matlab, SciVis, or Scientific IDL [3]. At this level, existing libraries that are appropriate for the domain can also be integrated with the environment.

2.6 Level 4 - Domain Specific Interface

Level 4 turns the PSE into a domain specific solver. Level 4 provides domain specific abstractions, and a user interface that operates in domain terms. The user interface (simply a collection of agents) translates the user's requirements into a specification usable by the underlying layers. It is at this level that many of the artifacts of creating programs are hidden in favor of solving problems. For instance, an unknown could be represented as an electric current in level 3, a column vector in level 2, and an array in levels 1 and 0. The concept of data flow can be hidden beneath graph templates that present only the steps necessary to solve the problem, in the order that the user would normally solve the problem.

3 Implementation

In this section a quick look is provided at an implementation of the level 0 toolkit, and at two PSEs currently being constructed to use this toolkit, one which supports electromagnetics application on clusters of workstations, and a second which supports image processing applications on reconfigurable computers. Due to space constraints, details of the attribute sets used at each level in each environment have been omitted. However, more detailed information about the toolkit and each of the environments, including examples and other environments not mentioned here, can be found at [15] and [16].

3.1 CECAAD

The Clemson Environment for Computer Aided Application Design (CECAAD) is a prototype implementation of level 0 as described in the previous section. CECAAD is an environment toolkit, and is the basis for the construction of the PSEs described at the end of this paper.

CECAAD consists of the ADF, the manager, the launcher, and a set of core agents. At the core of CECAAD is the ADF. ADF is the internal format for representing a directed attributed graph, as described in the previous chapter. The ADF manager provides synchronization of the actions of the agents on ADF designs and all I/O functions related to the library of designs. A small set of attributes are included in level 0 that implement the attribute set mechanism previously described.

Level 0 also contains several agents which are either useful for all environments or serve as a basis for the creation of more complicated domain specific agents. The ADF editor is a level 0 agent which provides a graphical view of ADF designs and allows the creation or changing of any ADF construct. The ADF text translator is another level 0 agent that translates ADF designs to and from a simple text-based language. The translator provides a basis for the creation of agents which could integrate existing languages for expressing parallel or scientific computation into CECAAD based environments. The Partition Agent attaches a "task" attribute to each node, and provides a graphical interface to group nodes into particular tasks.

3.2 An Electromagnetics Environment for Cluster Computers

A CECAAD-based PSE is currently being created to allow for parallel solution of integral equation method of moment problems in electromagnetics using Beowulf-class cluster computers. Problems of this type typically have roughly the same parallel structure, though numerically they can be very different.

Level 1 abstracts the cluster computer itself, and provides an abstraction for message passing between the nodes in the cluster. The hardware abstraction includes attributes for representing a heterogeneous cluster with various network topologies. A code generation agent uses the level 1 specification of the application along with the specification of the hardware to generate a PVM or MPI-based application. Another agent gathers information from the running program and adds performance information to the specification of both the hardware and the application, which is used by static and dynamic load balancing agents. The level 2 abstraction is of a data parallel programming model. Since the target problems have a fairly static data flow structure, an editor agent using the level 2 abstraction is employed by a user with computer science expertise to create ADF templates of the target problems.

Level 3 provides mathematical abstractions. At this level, a discretization agent allows the user to select basis functions to represent the geometry of the problem to be solved, and a linear system solver agent allows the user to select an appropriate solution method and convergence criteria. Level 4 provides the domain specific interface. At this level, the user employs a geometry editor to graphically define the problem's geometry in terms of insulators, conductors and dielectric materials. The user uses another agent to choose the quantity to solve for, and either select the equation to be used from a library, or to provide custom code (usually Fortran) that is to be used in conjunction with the geometry to fill the matrix (The user supplied code is wrapped in an ADF node that is then bound to the template supplied by the computer scientist).

This environment allows the parallelism and the details of the cluster to be hidden from the electromagnetics user, and allows the computer science user to collaborate without ever examining any of the electromagnetics involved. Applications could be ported to new architectures, or underlying models (such as shared memory) without any changes to the electromagnetics users specification of the problem.

3.3 An Image Processing Environment for Reconfigurable Computers

Reconfigurable Computers based on Field Programmable Gate Array (FPGA) technology are an emerging class of HPC system with the potential for providing enormous performance. Unfortunately, the problems associated with generating applications for this type of platform are even more daunting than those in parallel computing, as "programming" a reconfigurable system using conventional methods is akin to ASIC design in complexity. RCADE [14](the Reconfigurable Computing Application Development Environment) is a CECAAD based environment for FPGA based computing systems, which allows the user to work at roughly the level of a visual high level language to generate image processing applications. The user connects components which represent arithmetic and image filtering functions and basic loop constructs together in a data flow graph. Each component represents a pre-placed logic macro for the target platform.

At level 1, RCADE abstracts the computing platform, and uses attributes to represent the performance and interconnections between the FPGAs, as well as the routing resources between the logic blocks within the devices. At level 2, a data flow programming model is imposed which introduces the components and basic data types. Level 3 views the components from an equation based perspective, showing the mathematical transformations on the data, and level 4 adds an image processing layer by representing components in terms of the filtering operation they perform on the data.

A small library of RCADE components has been implemented on Xilinx 4000 series FPGAs. Currently, existing agents include a VHDL code generator and a partitioning tool based on level 1 specifications, a throughput analysis agent and pipeline balancing agent based on level 2 with latency information drawn from level 1, a simple component selection agent to match level 1 macros with level 2 specifications, and a data range analysis tool which examines level 3 information in order to allow the user to adjust the precision throughout the application in order to generate chip area savings.

Among the agents currently being developed for RCADE include automatic spatial and temporal partitioning, a macro generator which will use level 1 information to resize the hardware macros for components to specific geometries and bit precisions, and a graphical user interface to the level 4 specification. Eventually, the RCADE back end will be fused with a web-based interface for doing remote-sensing/image processing applications on parallel computers which currently exists at Clemson.

4 Conclusion

As PSEs become more prevalent to perform steadily larger and more complex scientific computation on steadily more advanced computing platforms, the need to better understand the construction of these PSEs will also increase. This paper has presented a proposal for a PSE architecture and infrastructure,

which leverages compiler technology and current approaches to creating high-performance applications. This infrastructure provides multiple abstractions to multiple groups of users, and allows these users to collaborate via these abstractions. Several different environments using different types of computers and different problem domains are being created to show the utility and versatility of this infrastructure.

Future work includes the completion of the prototype environments to more thoroughly test the infrastructure, and a public release of the CECAAD implementation with a more robust agent collaboration model.

References

1. Robert P. Wilson, Monica S. Lam, and John L. Hennessy et al. Suif: An infrastructure for research on parallelizing and optimizing compilers. Technical report, Computer Systems Laboratory, Stanford University, 1996.
2. OMG et al. Corba components: Joint revised submission. Technical report, Dept. of Computer Science,Rice University, December 21 1998. ftp://ftp.omg.org/pub/docs/orbos.98-12-02.pdf.
3. Rob Armstrong, Dennis Gannon, Al Geist, and et al. Toward a common component architecture for high-performance scientific computing. In *Proceedings of the 8th IEEE Int'l Symposium on HPDC*, pages pp.115–132. IEEE Computer Society, IEEE Computer Society Press, Nov. 1999.
4. R. Sessions. *COM and DCOM: Microsoft's Vision for Distributed Objects*. John Wiley & Sons, 1997.
5. A. Beguelin, JJ Dongarra, G.A. Geist, R.Manchek, and V.S. Sundaram. A users' guide to the pvm parallel virtual machine. Technical Report ORNL/TM-11826, Oak Ridge National Laboratory, July 1991.
6. Richard Snodgrass. *The Interface Description Language:Definition and Use*. Computer Science Press, 1989.
7. A. S. Grimshaw. Easy to use object-oriented parallel programming with mentat. *IEEE Computer*, pages 39–51, May 1993.
8. HyperParallel Technologies. Hyper c parallel programming language. http://www.meridian-marketing.com/HYPER_C/index.html, June 1999.
9. Jagannathan Dodd Agi. Glu: A high level system for granular data-parallel programming. *Concurrency: Practice and Experience*, 1995.
10. Schaeffer, Szafron, and Duane Lobe an Ian Parsons. The enterprise model for developing distributed applications. *Parallel and Distributed Technology*, 1995.
11. A. Beguelin, J.J. Dongarra, G.A. Geist, R. Manchek, and V. S. Sunderam. Visualization and debugging in a heterogeneous environment. *Computer*, 26(6):88–95, June 1993.
12. P. Bellows and B. Hutchings, "JHDL - An HDL for Reconfigurable Systems", *Proceedings of FCCM '98*, April, 1998.
13. Francois Bodin, Thierry Priol, Piyush Mehotra, and Dennis Gannon, "Directions in Parallel Programming: HPF, Shared Virtual Memory, and Object Parallelism in pC++", *Journal of Scientific Computing*, Vol. 2, no. 3, pp 7-22, June, 1993.
14. Ligon, Stanzione, et al, "Developing Applications in RCADE", *Proc of the IEEE Aerospace Conf*, March 1999.
15. The Clemson PSE web site,URL: *http://www.parl.clemson.edu/pse/*, 2000.
16. The RCADE web site, URL: *http://www.parl.clemson.edu/pse/rcade*, 2000.