Arches: An Infrastructure for PSE Development

Nathan DeBardeleben Walter B. Ligon, III Ron Sass

Parallel Architecture Research Lab
Holcombe Department of Electrical
& Computer Engineering
Clemson University
105 Riggs Hall
Clemson, SC 29634-0915
{ndebard,walt,rsass}@parl.clemson.edu

Abstract

The computational problems that scientists (and engineers) desire to solve are escalating to the point that both the programs they write and the computers they use to solve these problems are significantly more complex than the familiar, well-understood sequential model on their desktops. While scientists could be trained to use emerging high-performance computing (HPC) models, it is much more effective to provide them with a higher-level programming environment that has been specialized to their particular domain. By coupling the HPC specialist and the domain scientists, Problem-Solving Environments (PSEs) provide a collaborative environment that allows scientists to focus on expressing their computational problem while the PSE and associated tools support mapping that domain-specific problem to a high-performance computing system.

In this paper, we describe Arches, an object-oriented framework for building domain-specific PSEs. The framework was designed to support a wide range of problem domains and to be extendable in a way that allows it to target very different high-performance computing models. To demonstrate this flexibility we describe two PSEs that have been developed from the same framework yet solve different problems and target very different computing platforms. The Coven PSE supports parallel applications that need the large-scale parallelism that is found in cost-effective Beowulf clusters. In contrast, the RCADE PSE targets reconfigurable computing (FPGA-based) platforms with fine-grain parallelism. RCADE was designed to aid NASA Earth Scientists interested in studying satellite instrument data and who are unlikely to be schooled in low-level hardware design.

1. Introduction

Computational Science — the use of computer simulations to aid and advance our understanding of the physical world — is emerging as an important branch of science and may soon be on par with its siblings experimental and theoretical science. However, in general, scientists are not programmers and their interest and talents are firmly located in a science domain. To address this Problem Solving Environments (PSEs) are emerging as a way to raise the level of abstraction in which scientists program. By utilizing problem-domain-specific information, PSEs allow the scientists to access high performance computing resources, from the GRID to distributed-memory computers to custom-computing platforms. Thus PSEs are becoming an integral part of modern high performance computing (HPC) due to the increasing complexity of the types of simulations being run and the underlying problems being modeled, as well as the increasing complexity of the computer systems employed. PSEs help to manage the complexity of modern scientific computing by hiding many of the details of the computer system, the application, or both behind a comfortable, familiar interface. A good PSE is flexible enough to allow the user to solve the problem yet powerful enough to provide reasonably high performance.

Although there is a growing number of PSEs that have been developed over a wide range of problem domains, there is no standard procedure for their construction. Largely, every PSE is built *ad hoc* and the developers incrementally improve their designs based on their individual experiences. In this paper, we propose an object-oriented framework [11, 9] to make the construction of domain-specific problem-solving environments easier. To demonstrate the flexibility and functionality of our framework, we describe two independent PSEs that have developed from the same code base. Indeed, they still share the same code base — all customizations are achieved through inheritance in the object-oriented hierarchy. The PSEs – Coven and RCADE — target very different hardware platforms. Coven builds systems for large, distributed-parallel computers while RCADE builds hardware designs for reconfigurable computing systems. Likewise, the problem domains that they solve are very different one is for satellite telemetry while the other is a small collection of similar scientific codes.

The remainder of this paper is organized as follows. Below, in section 2, we describe a number of PSEs that have been developed for a variety of problem domains. In section 3 we describe our proposed framework, Arches. To show how two very divergent PSEs can be built from this framework, we provide and overview of Coven and RCADE in section 4 and section 5, respectively. We summarize in section 6.

2. Background

Over the years, a number of widely-different PSEs have been developed. This includes examples such as Khoros [19], ESMF[12], OSSIM[15], Cactus[1], BioPSE and Uintah[3] (based on SCIRun[14]), and CERSe[6] (based on Coven[5]). The Common Component Architecture[2], SCIRun and Coven are problem-solving environment toolkits that can be extended to build additional environments. In Khoros, modules (or *glyphs*) are connected to form data flow graphs. Glyphs are separate, sequential programs which reads input from one or more files and write outputs to one or more files. The Earth System Modeling Framework (ESMF) provides a infrastructure for creating multi-component applications targeting the earth science domain.

SCIRun and BioPSE target shared memory parallel computers which projects like Coven and Uintah target distributed memory machines. The CCA defines an interface for component writers to follow for compatibility with other CCA products. While the CCA is growing wide acceptance for modular software development, there has yet to emerge a larger infrastructure for solving problems from multiple scientific domains.

Products at different levels of the application life cycle exist and are often added on to existing PSEs. One popular low level trace analysis toolkit is TAU[20]. TAU is a performance monitoring system which allows for instrumentation of code during compilation and execution to monitor trace and performance statistics of programs. TAU interfaces with VAMPIR[18], a commercial performance and trace visualizer. The PSE Uintah uses TAU and its own visualization system XPARE[4]. Popular tools are often brought together with little support infrastructure. This lack of infrastructure makes it difficult to create tools that coalesce information from multiple sources. Using runtime performance analysis and detailed information about the underlying computer system coupled with the application and module design is cumbersome without an underlying infrastructure.

In contrast the the significant number of PSEs designed to assist with high-end parallel computing, very little has been done for hardware design or reconfigurable computing. Recently, many commercial tools have emerged with graphical and high-level interfaces that target an application's programmer. However, in our albeit limited but insightful conversations with users of these tools, we believe most users employ largely generic modules. This disables a great deal of specialization and suffers from inefficiency concerns, which motivate the use of PSEs. Two examples of these systems include CoreFire [17] and Viva [22].

Other high-level application development environments are emerging as well. For example, Forge [13], which converts Java source to Verilog HDL, shows great promise, and by avoiding pragmas and other directives, it is considerably more accessible to scientists and application developers. However, a drawback is that a scientist can easily make a small change that produces dramatically different hardware. Short of studying the application or pouring through lengthy computer-generated Verilog, it can be challenging to isolate problems. Nonetheless, we think Forge is an excellent tool, and we aim to integrate it as an alternative to manually generating modules.

Many of the ideas for Arches originated in the CECAAD project [21]. In addition to rewriting the implementation, there are several key differences. Arches supports concurrent access to the data flow graph and directly implements hierarchical designs. The Arches metaphor is more strongly centered as a collaboration tool whereas PSEs created for CECAAD were intended to support individuals. The concept of abstraction levels that was major feature of CECAAD has not been implemented in Arches, although it may appear in a future version. CECAAD also supports the use of SQL databases for non-volatile storage, whereas Arches used XML exclusively.

3. Arches Infrastructure Overview

In itself, Arches is not a PSE. It is an object-oriented framework [10] for building PSEs. As such, one has to understand the general model that Arches intends to support and the underlying data structures. In this section we provide an overview of both and then conclude with a description of how new users might build their own PSE.

3.1. Model

There are (at least) two primary stakeholders in an Arches-based PSE. The first are the scientists with domain-specific applications; the others are computer engineers with detailed knowledge of the target platform. (In some situations, there are additional stakeholders such as technology managers and agent-writers but their role is beyond the scope of this paper.) An Arches-based PSE is intended to be a multi-disciplinary collaboration tool that enables each stakeholder to focus on the problems that they do best. That is, it is designed to prevent the computer engineer from having to learn the domain-specific science to help the scientist map their application to the HPC system. Likewise, we want to protect the scientist from having to learn the latest HPC technology in order to get its performance advantage.

The main idea behind Arches-based PSEs is that there is persistent data structure (described below) that is acted on by several independent *agents*. The agents — or *tools* — are expertise-adding programs that annotate the data structure. The agents are coded by computer engineers that have a detailed knowledge target. A PSE-specific system generator is responsible for emitting an application (and, possibly, a run-time system) suitable for deploying on the target HPC system.

The agents are critical to extendable design. For example, the spatial nature of hardware designs is one of the primary advantages of reconfigurable computing. However, with it comes the requirement that the application know how to relatively place components on a 2D surface. In an Arches-based PSE, the placement expertise can be automated by developing an agent to add placement information to the data structure. In another example, profile information from a prior execution can be used by an agent to analyze a particular computation's organization. The agent can suggest or effect the reorganization. While agents are superficially similar to compiler 'passes', this example points out a striking difference. Our model assumes that agents are semi-automatic; that is, agents have the option of interacting with the scientists.

3.2. Data Structure

The underlying data structure is a persistent, attributed data flow graph, which represents a hardware design. There is a single manager object which is used to create, save, and load data flow graphs. Each agent runs in its own thread, and the manager arbitrates access to the designs that are stored in shared memory. It is common, for example, to have an editor (implemented as an agent) running with an open design while another agent is transforming the same design. The manager ensures that the agents remain consistent. The graph itself consists of three primary entities:

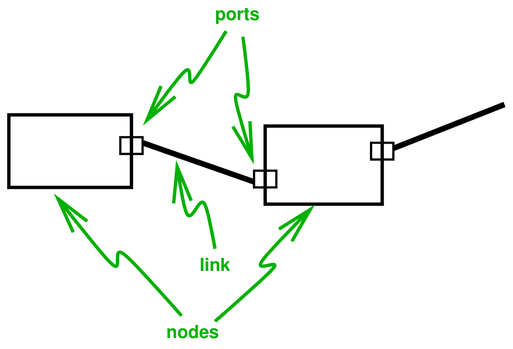


Figure 1. Arches entities in persistent data flow graph

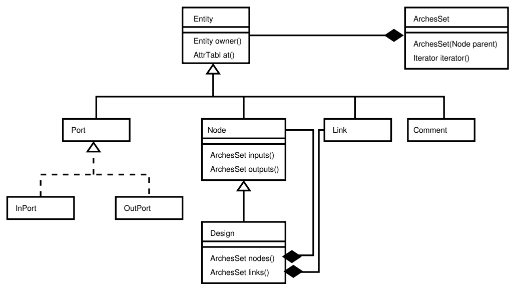


Figure 2. Arches entities in class diagram

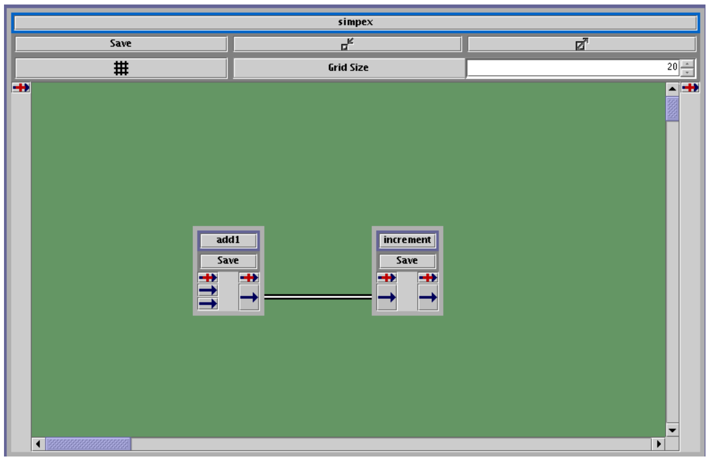
nodes, ports, and links, as shown in Figure 1. All entities have attribute tables that store persistent information. This includes information that may have been determined by previous agent invocations or from prior executions of the application. Attribute tables are themselves valid attributes and nested attribute tables are used extensively to organize agent-specific attributes. Similarly, there is a fourth entity called a design which extends node and is the algorithmic unit for holding a composition of nodes. Specifically, a design entity is a collection of nodes and links. Since it is an extension of node, it inherits the input and output ports of node. A more complete description of the relationships between the Arches classes, in UML notation, is shown Figure 2. Furthermore, because designs descends from node, it can be used anywhere node can be used. Thus, complex designs can be built hierarchically.

To get a flavor of how these classes interact, we describe a simple example. In Figure 3(a), we show the creation of a chain of two nodes. This might represent, depending on the problem domain, a pipelined computation or three concurrent tasks. In Figure 3(b), we open a design and invoke an Editor agent.

```
Design design ;
Node add1, inc ;
Port add1in0, add1in1, add1out, incin, incout

design = new Design("Calculate x+y+1") ;
add1 = new Node("add1") ;
add1.put("implementation","adder") ;
add1in0 = add1.add(new InPort()) ;
add1in1 = add1.add(new InPort()) ;
add1out = add1.add(new OutPort()) ;
inc = new Node("increment") ;
incin = inc.add(new InPort()) ;
incout = inc.add(new OutPort()) ;
design.add(add1) ;
design.add(inc) ;
design.link(add1out,incin) ;
```

(a)



(b)

Figure 3. (a) constructing a simple design (b) screen shot of editor agent

3.3. Creating a PSE

The main idea behind the Arches framework is to assist a programmer in developing a new PSE. If the PSE to be developed follows the same general model as previously described, then much of the required infrastructure can be supplied by the Arches package. To help the reader understand the framework, we describe the general steps required to create a PSE.

The first step in creating a PSE with Arches is determining if a data flow graph is a suitable representation for algorithms in the target problem domain and platform. If the algorithms can be componentized (into Arches nodes) and exchange data between components using a communication path (Arches links) then the Arches framework may be able to supply a great deal of the base PSE infrastructure.

The PSE designer should next determine what agents are appropriate for their problem domain. If there are any common tools that can add expertise or assist users in solving their problems, then these should be implemented as (or ported to) Arches agents. The relevant attributes of these agents should be identified and supplied to Arches so that the agents can interact with the data flow graph and other agents.

Next, the PSE designer must determine how algorithms will be put into the PSE and how the resulting implementation is generated. If a graphical PSE is needed, then extending the graphical editor (described above) may provide an easy solution. However, some PSEs might simply use a parser and a custom agent to get the algorithms into the Arches framework. Additionally, once the algorithm has been implemented in Arches, the designer must decide what output they expect the PSE to provide. For example, this might require a code generator to translate the Arches data flow graph into the target-specific code.

Finally, runtime considerations are required. It may be important for the runtime system implemented for the PSE to send information back to the PSE so that it can be placed into the data flow graph. This is common for post-analysis agents that need to visualize data, performance, or trace statistics. With the Arches framework, some of the tedious tasks related to creating a PSE have been centralized into a reusable, extensible infrastructure. Therefore, more emphasis can be placed on the parts of a PSE that are specific to a particular problem domain.

4. Coven

Even with embarrassingly parallel computing applications, managing parallel codes is time consuming. The Coven PSE is targeted to scientists with the need to run applications on distributed memory parallel computers. The goal is to make these architectures more approachable.

4.1. Problem

With the growing availability of parallel computing systems, scientists are expanding their models and simulations to levels of complexity which far surpass those that are feasible to run on a conventional computer. Automatic parallelizing compilers have proven useful for some types of applications, but many are left with suboptimal performance. The conventional alternative lies in explicit parallelization, often through the use of MPI[16]. Mastering the details of explicit parallel programming is complex and often requires a great deal of understanding of both the application and the underlying computer system.

4.2. Coding With Parallel Modules

Coven provides an environment where the application specialist and parallel computing specialist can collaborate to solve the problem in parallel. This is accomplished through a data flow model where user code is encapsulated into modules with a well defined interface. Modules are implemented in either C or Fortran and are connected together to form a data flow graph.

Through intercommunicating components and a pluggable, modular, data flow model, the code pertaining to performing the science is kept separate from the code pertaining to parallelism. Thus, the scientist only needs to program their algorithm in a collection of modules while a parallel computing specialist programs the parallel communication modules. Coven handles putting the modules together into a complete runtime application. With this modular approach, the code is not only more manageable and maintainable but also allows for portions to be replaced very easily to test new ideas.

4.3. Implementation

Coven is composed of two main components: a front-end running on a user's workstation and a back-end running on a parallel computer. The front-end is implemented with the Arches framework where nodes represent user supplied modular code. The interface to the modules are depicted as Arches ports and data flow between modules is represented as links. Once a Coven program has been constructed using the PSE tools, the job is compiled and submitted to the back-end for execution.

A runtime engine executes the Coven application in parallel, profiling trace information and reporting it back to the front-end upon completion. The runtime engine is multi-threaded as well as running on multiple parallel processors. This allows modules to be grouped together on the same thread to improve performance from asynchronous operations. Shared memory is provided behind the scenes to quickly move data between threads running on the same processor. Clearly, there is not enough space to discuss Coven in detail. However we will highlight a few agents; an interested reader can refer to [7, 8].

Coven Graphical Editor Programs are assembled using the Coven graphical editor. Coven module source files are imported into the design which automatically constructs a node for the module and the appropriate ports. Module inputs and outputs are represented as ports and attribute tables are filled with type information which is used in type checking. Using the Coven editor, the programmer can drag and drop module code, interconnect the pieces, specify runtime parameters, and submit the job to be run.

Code Generator and Compiler Coven uses an intermediate language to represent programs. This language specifies the flow of data between modules in a textual form, as well as containing any parameter values. Users can code Coven applications in this language, or use the code generator agent to automatically generate a Coven language source file from the Arches data flow graph. The code generator agent is initiated by the Coven editor at the start of runtime. The generated language source file is converted by the Coven language compiler into a form which the back-end uses to determine what modules to execute, with what parameters, and how the modules should be threaded.

Data Visualization Agents have been created which visualize application data both in real time and offline. Several provided real time visualization agents have corresponding modules which in parallel transmit data to be visualized back to the agent. This is done in parallel and can provide runtime updates of the progress of a simulation. Agents have been created to visualize molecular interactions, heat transfer CFD problems, n-body simulations, and satellite remote sensing applications. These agents utilize OpenGL, Java3D, and Java2D APIs and additional agents can be easily written for different problem domains.

Profiler Trace information about the running application is transferred after execution to the front-end where the profiler agent processes it. Many different visualizations help to depict the runtime characteristics of the application, including memory consumption, CPU utilization, module runtime, and data flow between threads and processes. The visualizations can be tailored to display information in a helpful manner for either an application scientist or a parallel computing specialist.

4.4. Example

Coven has been used to create applications for satellite remote sensing, complex fluid dynamics, molecular dynamics, and n-body simulations. In previous work[7, 8] we have outlined the performance benefits from multithreading and shared memory utilization. Due to space limitations, a complete example cannot be given. However, we will discuss a couple of screenshots from different types of Coven applications.

Figure 4 is a screenshot of a complex fluid dynamics heat transfer OpenGL 3D graphical agent which depicts how a steel plate heats up due to an external source. Figure 5 is a screenshot of a 2D Fast Fourier Transform (FFT) application being analyzed in the Coven profiler agent. This particular visualization is looking at the load placed on the CPU by different modules.

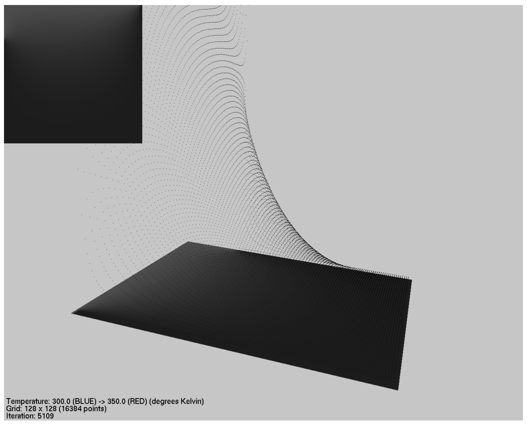

Temperature: 300.0 (BLUE) -> 350.0 (RED) (degrees Kelvin)
Grid: 128 x 128 (16384 points)
Iteration: 5109

Figure 4. 3D Visualization Agent

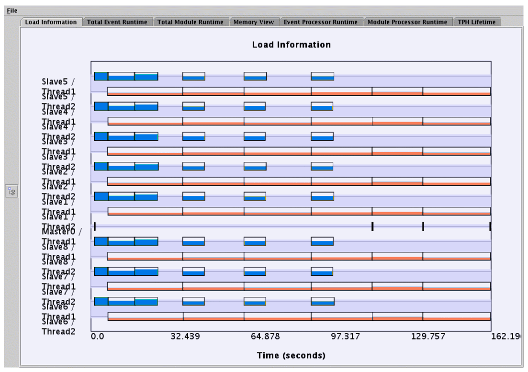


Figure 5. CPU Load Profiler Agent

5. RCADE

The RCADE PSE is targeted to NASA Earth Scientists that would like to take advantage of FPGA-based Reconfigurable Computing (RC). An FPGA is programmable device that can be configured into any arbitrary digital circuit. This allows the user to develop application-specific hardware designs that can be downloaded to accelerate their application. Although the technology is rapidly advancing, programming an FPGA requires an understanding of hardware design principles.

5.1. Design Quality

The fundamental problem is that fully automatic hardware designs from scientific applications is not yet feasible for RC. While a number of automatic tools have been discussed in the research community, they all generally suffer from the fact that they make inefficient use of the resources. Even though newer devices are significantly larger, a simple, direct mapping of the computation to the FPGA does not achieve the performance required by the NASA scientists.

To get the desired performance, the mapping of the design needs to pay attention to three quality issues: scheduling, latency, and specialization. In hardware design, the computation is arranged spatially and the designer needs to ensure that the timing of values propagated through the computation are correct. While handshaking can be introduced to guarantee the correctness, the design takes a performance hit if the timing is incorrect. Another factor of hardware design is that clock speed is determined by the length of longest wire. Without attention to placement, the latency of propagating a signal through the device can destroy the performance. Finally, one of the most valuable features of reconfigurable computing is that it allows for custom designs. Replacing a general multiplier with a constant multiplier can have an order-of-magnitude performance improvement with a significant reduction in space. Unfortunately, automated tools cannot make all of these decisions because some require user interaction.

5.2. Typical Scenario

To understand how a PSE addresses these concerns specifically, consider the following scenario that shows how a scientist gets from algorithm to HPC platform.

Suppose a scientist has an algorithm that they believe would map well to a RC board. The data set may be coming from a satellite receiver or it may be an archived data set. In either case, there is usually some pre-processing that generates a look-up table that is referenced while processing the data set. Since this calculation is usually a tiny fraction of the compute time, it is easily calculated in the usual way with either a C or Fortran program. Next there is fairly regular structure that processes the data set and performs the calculations on the data set.

There are several ways of entering a design into RCADE. A scientist can drag-and-drop nodes and then connect ports. However, it is much easier to use a translator agent. Some sample translator agents exist that translate expressions from familiar languages such as a FORTRAN or C into RCADE data flow graphs. In any case, part of the application is done on the processor and part has been translated into Arches. There are set of routine steps the scientist does to generate hardware. The scientist would analyze the computation for precision, run a pipeline balancer, and a placement agent. Next the part selector would go through and pick implementations for every part. Changes might cause the pipeline balancer to run again. Eventually, the design is ready for synthesis.

After synthesis the scientist might discover that the design doesn't fit or doesn't meet the performance goals. A hardware engineer's solution might result in the development of a highly customized module for the application or, perhaps, the recommendation to the technical manager that a new agent be commissioned to handle this circumstance in future designs. In the middle of this collaborative effort is RCADE, which provides the communication tool as well as development environment. To achieve this the RCADE environment must not only support the conventional development elements (user interfaces, graph transformations, *et al.*) but also the ability to add agents dynamically and a design generator that allows the discovery of modules as described below.

5.3. Implementation

Once the design is entered, a number of agents are invoked. A precision tool is a semi-automatic tool that propagates precision information throughout the design. A placement tool can locate modules on an FPGA to accelerate the commercial place&route times and improve the system-level data flow paths. Other agents, such as a partitioning tool, a reconfigurable cache tool, *et al.* may be invoked if desired. All of these tools update the attributes associated with entities throughout the design. The final step is for the design generator to build a hardware representation.

Precision Tool Traditional hardware design using HDLs relies on the programmer to evaluate data paths and determine the appropriate bit-widths to minimize logic, while allowing enough bits for correct data transfer. As discussed previously, designs that have undergone analysis to optimize data path bit precision can be shown to have decreased latency, as well as provide a resulting design that utilizes less logic (smaller footprint), when compared to the original, generic design. By providing an agent to perform data path analysis, RCADE allows those with little hardware design experience (scientists) to generate better performance solutions to problems and better utilize available chip space, while freeing those familiar with hardware (hardware engineers) to concentrate more on the structure of their designs.

The RCADE precision tool utilizes user-editable attributes to calculate data path bit-widths. These attributes represent the largest and smallest possible values for each port within a design. (i.e. the range of values for the port). Calculations are done at the node level and may be performed in a forward (progressing from node inputs to outputs) or backward (from outputs to inputs) manner. For example, during a forward calculation, the range information for each input port of a node will be used in conjunction with the node's behavior (addition, multiplication, etc.) to correctly determine the range of the output port(s). Once a node's calculation is complete, the resulting output port's range values are propagated to all input ports to which it has a link. Calculations begin from all external design input ports or output ports, depending on the desired direction of calculation, and progress to all internal nodes. This progression follows the link structure of the design from port to port until there are no more links.

The forward/backward functionality of the precision tool is implemented in the RCADE Tile interface. This interface provides a bridge between the hardware implementation of a part and its software representation in RCADE. For example, RCADE provides a node representation of an adder. This RCADE adder node coexists with an implementation of the Tile interface that specifies how to properly calculate the input and output port bit-widths for an adder hardware part. Each RCADE node, whether an adder, multiplier, or user designed node, must also have a tile specific to its behavior. Inside that tile, forward and backward methods should access the attributes associated with bit-precision, perform calculations to minimize the data path bit-widths, and reset the bit-precision values on ports accordingly. It is these bit-precision values that the code generator will use to construct hardware implementations of the final design.

Placement Tool A placement package has been created to represent the chip and determine locations for modules to be placed on the chip. An RCADE agent that utilizes the package is launched from the Editor window. This agent updates attributes of each Node to specify its location.

The Placer creates a module for each Node to be placed. Each module's connection information is read and a list of its inputs and outputs are recorded. A tree is created based on the connectivity information and the roots are determined. A root is defined as a module that outputs directly to the pins of the FPGA. For each tree, the modules are placed starting with the root in a breadth-first manner.

When all modules are placed, an implementation is created. An implementation is a group of modules with fixed locations. The best implementation of a set group of modules is determined by the routing cost. The routing cost between any two modules is a function of the Manhattan distance and number of turns. An implementation's routing cost is the sum of the routing costs of all its modules. If the routing cost is above a given threshold, the placer can run again with variations in the placement order. The variations occur by placing either the trees or module siblings in a different order. Changing the default placement location within an empty rectangle also produces new implementations. One or several permutations of these modifications, depending on time considerations, can me made. In all cases, the best implementation is chosen and the module locations are set permanently.

Pipeline Balancer Just prior to generating a hardware description of the user's design, a pipeline balancer is invoked. In our system, the role of the pipeline balancer is two-fold. First, it is responsible for inserting all passive FERP (variable depth FIFO) parts between the active computational parts. The FERP signalling maintains correctness by ensuring that computation only proceeds if the data has arrived. However, as noted earlier, parallel paths in the data flow graph with an unequal number of pipeline stages can destroy performance. The second responsibility of the pipeline balancer is to traverse the graph and for each FERP set the appropriate depth such that all converging paths have the same latency.

This is accomplished by calculating the number of stages needed in the FIFO. The number of additional buffers needed, N_i along any give link i, can be determined by calculating the maximum difference in latency across all of the parallel paths converging, as shown below.

$$N_i = \max_k \{E_k - E_i\} \tag{1}$$

For each one of the parallel paths, that path's latency E_m is governed by

$$E_m = \sum_j C_m \frac{l_j}{t_j} + L_m \tag{2}$$

where the summation is over all each component j on that path, the latency of component j is l_j, and the throughput of component j is t_j. The terms L_m is the number of stages in FERPs between the components in the path and C_m is the number of components in the path.

These calculations are easily automated and this agent is in process; however, the number of buffers needed for the results reported in this paper were manually calculated.

5.4. Example

Although space limitations prevent us from completely explaining an example, we believe a screenshot is helpful. In Figure 6, a simple calculation is shown. The format is such that both the scientist and the hardware engineer can understand what is happening.

6. Conclusion

In this paper we have introduced Arches, an object-oriented framework for building Problem-Solving Environments. The framework was designed with extensibility in mind. It is generic enough to be used in a wide variety of situations while still providing several concrete services and functionalities needed when developing problem-solving environments. To demonstrate the aptitude of this design, we describe two complete problem-solving environments; namely Coven and RCADE. While both PSEs have the same goals of helping scientists map their applications to high-performance computing systems, they are radically different in their customizations and the support structures. Despite their different problem domains and different implementations, they both effectively reuse the Arches infrastructure.

References

[1] G. Allen, T. Goodale, J. Massó, and E. Seidel. The cactus computational toolkit and using distributed computing to collide neutron stars. In *Proceedings of the Eighth IEEE International Symposium on High Performance Distributed Computing*, pages 57–61, Redondo Beach, CA, August 1999. IEEE Computer Society Press.

[2] R. Armstrong, D. Gannon, A. Geist, K. Keahey, S. Kohn, L. McInnes, S. Parker, and B. Smolinski. Toward a Common Component Architecture for High-Performance Scientific Computing. In *Proceedings of the 1999 Conference on High Performance Distributed Computing*, pages 115–124, 1999.

[3] J. de St. Germain, J. McCorquodale, S. Parker, and C. Johnson. Uintah: A massively parallel problem solving environment. In *Ninth IEEE International Symposium on High Performance and Distributed Computing*, pages 33–41. IEEE, Piscataway, NJ, Nov 2000.

[4] J. D. de St. Germain, A. Morris, S. G. Parker, A. D. Malony, and S. Shende. Integrating performance analysis in the uintah software development cycle. In *International Symposium on High Performance Computing (ISHPC-IV)*, pages 190–206, May 2002.

[5] N. A. DeBardeleben, W. B. Ligon III, S. Pandit, and D. C. Stanzione Jr. Coven - a framework for high performance problem solving environments. In *Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing (HPDC-11 2002)*, pages 291–298, Edinburgh, Scotland, UK, July 2002. IEEE Computer Society.

[6] N. A. DeBardeleben, W. B. Ligon III, and D. C. Stanzione Jr. The Component-based Environment for Remote Sensing. In *Proceedings of the 2002 IEEE Aerospace Conference*, pages 6–2661–6–2670, March 2002.

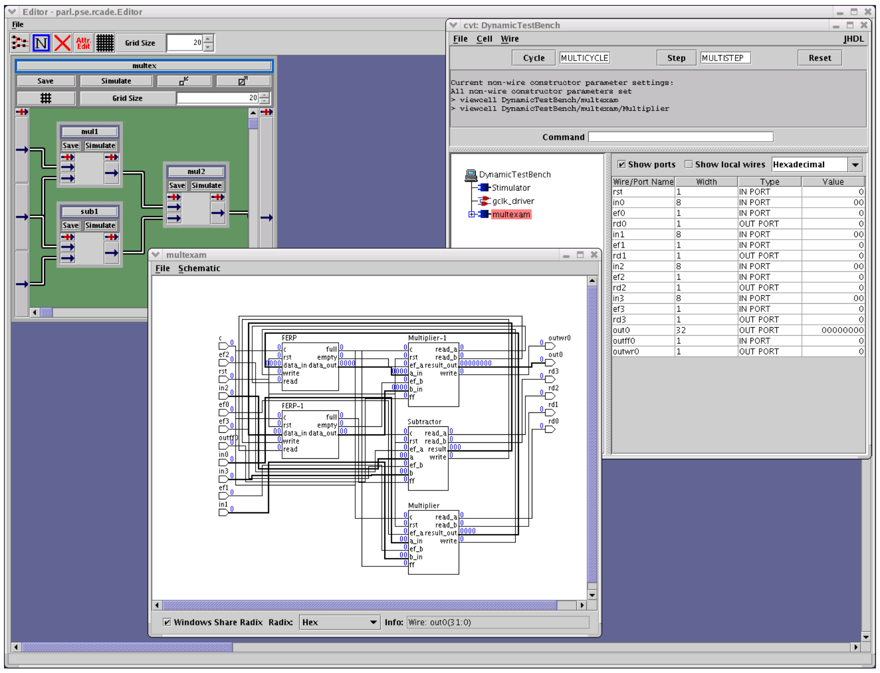
[7] N. A. DeBardeleben, V. Patil, and W. B. Ligon III. Performance enhancements to coven through multithreading. Technical Report PARL-2004-001, Parallel Architecture Research Laboratory, Clemson University, 2004.

[8] N. A. DeBardeleben, V. Patil, and W. B. Ligon III. Using coven to profile and tune parallel programs. Technical Report PARL-2004-002, Parallel Architecture Research Laboratory, Clemson University, 2004.

[9] M. E. Fayad. Introduction to the computing surveys' electronic symposium on object-oriented application frameworks. *ACM Computing Surveys*, 32(1):1–9, Mar. 2000.

[10] M. E. Fayad, D. S. Hamu, and D. Brugali. Editorial: Enterprise frameworks. *Software Practice & Experience*, 32(8):735–736, July 2002.

[11] M. E. Fayad and D. C. Schmidt. Object-oriented application frameworks. *Communications of the ACM*, 40(10):32–38, Oct. 1997.



```
Current non-wire constructor parameter settings:
All non-wire constructor parameters set
> viewcell DynamicTestBench/multexam
> viewcell DynamicTestBench/multexam/Multiplier
```

Show ports [✓] Show local wires [ ] Hexadecimal

Wire/Port Name	Width	Type	Value
rst	1	IN PORT	0
in0	8	IN PORT	00
ef0	1	IN PORT	0
rd0	1	OUT PORT	0
in1	8	IN PORT	00
ef1	1	IN PORT	0
rd1	1	OUT PORT	0
in2	8	IN PORT	00
ef2	1	IN PORT	0
rd2	1	OUT PORT	0
in3	8	IN PORT	00
ef3	1	IN PORT	0
rd3	1	OUT PORT	0
out0	32	OUT PORT	00000000
outff0	1	IN PORT	0
outwr0	1	OUT PORT	0

Figure 6. an example of hardware design generated

[12] R. Ferraro, T. Sato, G. Brasseur, C. DeLuca, and E. Guilyardi. Modeling the earth system. In *International Geoscience and Remote Sensing Symposium*, Sep 2003.

[13] X. Inc. Forge. `http://www.xilinx.com/`.

[14] C. Johnson, S. Parker, D. Weinstein, and S. Heffernan. Component-based problem solving environments for large-scale scientific computing. *Journal on Concurrency and Computation: Practice and Experience*, (14):1337–1349, 2002.

[15] K. Melero. Open Source Software Image Map Documentation. http://www.ossim.org.

[16] Message Passing Interface Forum. Document for a standard message-passing interface. Technical Report CS-93-213 (revised), University of Tennessee, April 1994.

[17] A. Microsystems. Corefire. `http://www.annapmicro.com/`.

[18] Pallas GmbH. VAMPIR: Visualization and Analysis of MPI Resources. http://www.pallas.de/pages/vampir.htm.

[19] J. Rasure and S. Kubica. *The Khoros Application Development Environment*. Khoral Research Inc., Albuquerque, New Mexico, 1992.

[20] S. Shende and A. D. Malony. Integration and application of the tau performance system in parallel java environments. In *Proceedings of the 2001 joint ACM-ISCOPE conference on Java Grande*, pages 87–96, University of Oregon, Eugene, Oregon, June 2001.

[21] D. C. Stanzione Jr. and W. B. Ligon III. Infrastructure for High Performance Computer Systems. In e. a. Jose Rolim, editor, *IPDPS 2000 Workshops*, *LNCS 1800*, pages 314–323. ACM/IEEE, Springer-Verlag, May 2000.

[22] S. Systems. Viva. `http://www.starbridgesystems.com/`.