Coven - a Framework for High Performance Problem Solving Environments

Nathan A. DeBardeleben Walter B. Ligon III Sourabh Pandit Dan C. Stanzione Jr.
Parallel Architecture Research Lab
Holcombe Department of Electrical and Computer Engineering
Clemson University
105 Riggs Hall
Clemson, SC 29634-0915
{ndebard, walt, spandit, dstanzi}@parl.clemson.edu

Abstract

This work presents the design of the Coven *framework for construction of Problem Solving Environments (PSEs) for parallel computers. PSEs are an integral part of modern high performance computing (HPC) and Coven attempts to simplify PSE construction.*

Coven targets Beowulf cluster parallel computers but independent of any particular domain for the PSE. Multi-threaded parallel applications are created with Coven that are capable of supporting most of the constructs in a typical parallel programming language. Coven uses an agent-based front-end which allows multiple custom interfaces to be constructed.

Examples of the use of Coven in the construction of prototype PSEs are shown, and the effectiveness of these PSEs is evaluated in terms of the performance of the applications they generate.

1. Introduction

Problem Solving Environments (PSEs) are 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, traditional means of writing simulation software no longer remain practical.

PSEs help to manage the complexity of modern scientific computing. A good PSE provides a user with a comfortable, familiar interface. It hides many of the details of the computer system, the application, or both. A good PSE is flexible enough to allow the user to solve the problem yet powerful enough to provide reasonably high performance though perhaps not as good as a hand-tuned application.

A few PSEs which meet this criteria have been developed in certain domains for certain classes of computer systems. Khoros [7] is a PSE developed initially by the University of New Mexico and later by Khoral Research, Inc. In Khoros, modules (or *glyphs*) are connected to form dataflow graphs. Glyphs are separate, sequential programs which pass data between other glyphs through the host file system. Each glyph reads inputs from a file and writes outputs to be consumed by other glyphs into a file. Cactus [8] is a PSE for solving grid-based problems on parallel computers. With Cactus, modules (or thorns) are placed into the driver (or flesh) and run in parallel. CAMEL (*Cellular Automata environMent for systEms modeLing*) is a cellular automata environment developed by researchers in Italy [1], [9]. CAMEL provides a language called Carpet in which scientists can design cellular automata parallel programs with little understanding of parallel computing. J. Cuny et al. [3] developed a parallel programming environment for studying seismic tomography as well as several visualization tools. The Open Source Software Image Map (OSSIM) [6] is a software tool being developed for solving remote sensing applications on a parallel computer through modular program development.

The problem of constructing these PSEs remains a daunting task. Although there are many good examples of PSEs, little of what has been done to create existing PSEs can be reused to create new ones, and a general model for building PSEs has not yet emerged [10].

Two important characteristics of a good PSE framework are *flexibility* and good use of *abstraction*. Flexibility must exist in the ability to support a wide range of computational models that various domains may require, as well as the flexibility to support a wide range of interfaces. Abstraction must be used to carefully hide the details of both the underlying computer system and the problem domain where appropriate.

In this work, we present the design of the *Coven* framework for construction of PSEs. Coven attempts to address the issues above, and simplify PSE construction. Coven

- has a multi-threaded parallel runtime system that
  - targets Beowulf cluster parallel computers,
  - uses a runtime generated data structure known as a Tagged Partition Handle to manage partitioning data sets among the cluster nodes,
  - executes applications capable of supporting most of the constructs in a typical parallel programming language,
- has an agent-based front-end which
  - allows multiple custom interfaces to be constructed,
  - stores information about the specification, implementation, and performance of the application in an attributed graph format on the front-end, and
  - provides a way for agents to filter the attribute information stored in the graph to provide suitable abstractions for a particular class of user,
- and has a module library with
  - predefined system modules, and
  - reusable application modules.

The rest of this paper is organized as follows. Section 2 gives an overview of the Coven architecture. Detailed description of the runtime support is provided in Section 3 and the graphical user interface is described in Section 4. Section 5 compares Coven with several similar projects and some examples of customization of Coven to particular target environments is given in Section 6. Section 7 shows performance results for applications generated by two of the prototype PSEs created with Coven.

2. Coven Overview

Coven is a framework for building problem solving environments for parallel computers. It is composed of three main portions: a runtime driver running on a parallel computer, a front-end running on a workstation, and a module library residing on the workstation. Figure 1 depicts the relationship between these components.

The front-end is used for modular program design where modules are interconnected to form a dataflow graph. In Coven, modules are subroutines in either C or FORTRAN with special directives that identify inputs and outputs placed at the beginning. Modules are compiled and stored

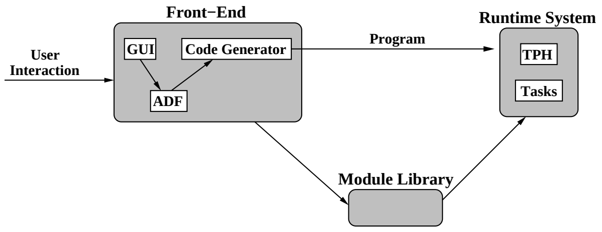


Figure 1. Coven Architecture

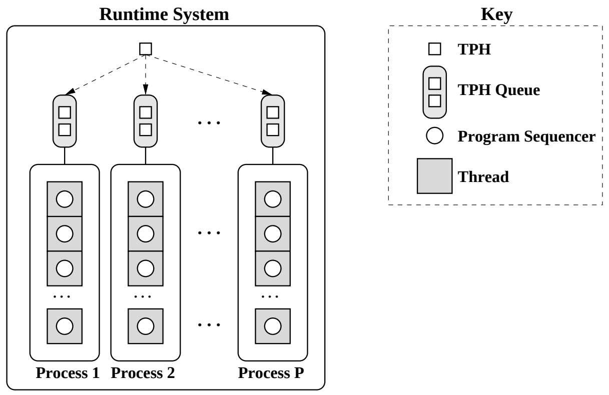


Figure 2. Coven Runtime Driver

in the module library. The code generator transforms the dataflow graph into an internal representation which the runtime driver uses. This representation and the referenced modules are then transferred to the runtime driver (shown in Figure 2) on a parallel computer for execution.

All data within Coven's runtime driver is encapsulated into a data structure called a Tagged Partition Handle (TPH). As TPHs pass through modules, the modules read data from and add new data to the TPHs. At any instance in time, many TPHs are flowing through the system.

The Coven runtime driver runs on each processor and is multi-threaded. The user can define how many threads are used and which modules run under which thread. Coven's runtime driver internally maintains a TPH queue for each thread. With this approach, a great deal of pipelining as well as asynchronous computation, I/O, and communication can be achieved.

The runtime driver and front-end can both be extended for target PSEs. This extensibility makes it possible to customize the Coven framework to an application domain by changing the view of the computation presented in the interface to be more familiar to the application domain user.

The Coven framework builds on prior work in PSE toolkits done in CECAAD [11]. CECAAD provides basic support to allow a group of independent tools to collaborate on a shared design stored as an attributed graph. The CECAAD project sought to provide a toolkit for building PSEs in general, regardless of type of computer architecture employed for the back-end. Coven builds on this model and targets message passing programs for Beowulf clusters specifically in its back-end. From CECAAD, Coven uses the attributed graph format for the shared design (ADF - the algorithm description format) and the GUI ADF editor. Coven also employs the CECAAD agent model to provide an interface to ADF, access to a design database and for synchronization between agents.

Coven takes CECAAD one step further and provides a back-end and runtime environment. Coven is then in turn extended by custom front-ends and to provide a complete PSE.

Coven has evolved from a PSE which had an image processing, macro dataflow input/output relationship model. Most problems within this model fit very easily and nicely. Some computation, such as ones which are iterative or perform data rearranging, do not fit well within this simple flow graph model. The modification of the Coven framework to allow for modules to control flow of TPHs and communicate with other modules (either on the current processor or another parallel processor) broadens the range of possible computations.

3. Coven Runtime Driver

Coven's runtime driver (Figure 2) is composed of several pieces. An internal data structure, the Tagged Partition Handle, is used by the system for transfer of data between modules and is described in Subsection 3.1. Different classes of modules exist within Coven and are outlined in Subsection 3.2. Finally, the many components which make up Coven are outlined in Subsection 3.3.

3.1. The Tagged Partition Handle

The *Tagged Partition Handle* (TPH) is an internal data structure within Coven. TPHs are structures which hold buffers of data and provide means for creating, accessing, modifying, and destroying these buffers. All inputs to and outputs from modules are managed through TPHs. Helper functions are provided so module programmers need not interact directly with the data structure. Access to buffers is through a tag which provides a way for a module to describe the data it wants to process (or create). This is especially important in a parallel system where the data passes from machine to machine and likely will not reside within the same address space.

Most sequential programs in a conventional programming language have "the data space." This exists as a piece of memory where data can be created, accessed, destroyed, etc. This is sometimes static (FORTRAN) or dynamic (Java, or C). Mechanisms for accessing this data are provided by the language.

Parallel programs, however, break the data space in two different ways: spatially and temporally. An example of spacial partitioning of data is where a matrix is divided into submatrices which are processed by different tasks. An example of temporal partitioning of data is when a piece of data exists on one processor at some time and then exists solely on another processor at another time.

Consider matrix multiplication algorithms like Cannon's algorithm or Fox's algorithm. In these, each processor holds a subarray of each of the two input matrices. Then each processor exchanges submatrices of one matrix vertically, and the other horizontally. Part of writing a parallel program is knowing how to get the right data together in the same place at the same time.

In a general model of parallel computation a task is a sequential set of computations performed on a piece of data. Within Coven, a task is created by the combination of a module (a piece of code which operates on some data) and a TPH (a data structure containing related data). The objective is to schedule the execution of modules and TPHs to perform tasks in parallel. The Coven runtime driver provides means for this which allows for overlapping of I/O, computation, and communication.

3.2. Application and System Modules

There are two classes of modules within Coven: *application modules* and *system modules*. The two classes differ only in the types of operations they perform. There is no distinction to the runtime engine between the two classes.

Application modules are modules which are written by an application designer. Examples of this type of module would be those which:

- compute the resultant of multiplication of two vectors,
- compute the partial force between two bodies,
- calculate the latitude and longitude of a buffer of gridpoints, or
- update a temperature matrix based on values of neighboring cells.

System modules are modules which are probably written by someone familiar with parallel computing, load balancing techniques, etc. Examples of this type of module would be those which:

- perform parallel communication (such as with MPI),
- partition data,
- create TPHs, or
- consume TPHs.

Through system modules, the distribution of TPHs (and therefore the scheduling of tasks) can be steered. This allows users to write (or import) modules to customize these tasks to individual problems. For example, when partitioning data modules decide where TPHs go and the runtime driver actually transports the TPHs. As another example, results from parallel computation are stored in TPHs residing in modules in the parallel tasks. These results often have to be combined into a single result (placing submatrices into a larger matrix, compositing image partitions into a larger image, etc). This task can be performed by a module which consumes TPHs - it accepts many TPHs as input and produces a single TPH upon completion.

Common system modules (such as parallel data distribution modules) would typically be available in the module library for easy use in many applications. If an appropriate module does not exist in the library, a new one can be written.

3.3. Runtime Driver Components

Coven's runtime driver executes on a parallel computer. Specifically, the target machine is a Beowulf-class supercomputer. Beowulf clusters [12] are large computers comprised of commodity off-the-shelf (COTS) hardware components such as inexpensive memory, EIDE disks, commodity networks, and conventional CPUs. The software (including operating system, parallel programming libraries and tools) are open source and are available for development of user code.

Coven executes on a dataflow graph of interconnected modules (subroutines). Modules are dynamically loaded from a library of modules by the *Module Loader*. All data is managed by and accessed through TPHs. Each module is assigned a thread and any number of modules can exist within the same thread. Data is passed sequentially between modules in the same thread and is then enqueued on the next thread (on the same process or possibly another process).

Each thread runs a component named the *Program Sequencer* which handles the tasks of dequeuing TPHs, running modules, and enqueuing the TPHs on other threads. Queues of TPHs are maintained before every thread. The use of queues, as well as a user-tunable maximum number of TPHs in any queue, allows Coven to get a large pipeline of TPHs flowing through the system. Additionally, the runtime driver can take full effect of asynchronous computation, I/O, and communication as the operating system will handle scheduling threads as they have data ready to process.

Coven supports profiling. TPHs are profiled as they enter and leave queues, modules, threads, and processes. Profiling is also performed of interprocess communication and file I/O. Large profiling reports are generated during and after a parallel job is run. Front-end agents can make use of this profiling information to produce a graphical view of program performance. A prototype agent that provides a simple profile has been constructed. Tools that can take full advantage of Coven's advanced profiling are being developed. It is expected that these tools will allow visualization of the life cycle of TPHs. Queuing analysis will be possible as will visualization of which modules and threads spent the most time executing.

4. Coven Front-End

Coven's front-end runs on a workstation separate from the runtime environment running on a Beowulf cluster. The front-end is used to generate the Coven program which will run under the runtime driver. This program can be hand-coded or a tool such as the code generator (Figure 1) can assist in this process.

A key part of the front-end is to assist programmers in writing their application. This can be done through different kinds of interfaces such as language-based or graphical. CECAAD has agents for both language-based and graphical specification.

The graphical front-end which was adapted from CECAAD provides (through ADF) a framework for graphical agents to act on the design (or dataflow graph). A graph editor agent is used for building the dataflow graph. This agent provides a drag-and-drop interface where modules are placed into either sequential or parallel regions and then interconnected. Each module is assigned a thread under which it will execute in the runtime driver. Figure 3 depicts the graphical editor agent with a sequential region for problem decomposition, a parallel region with many interconnected modules, and a sequential region following for combination of parallel results.

All modules referenced in the dataflow graph reside within the module library. Each modules contain special directives which retrieve inputs from TPHs and create new outputs. These directives contain the type of the data as well as user-specified text descriptions. Whenever a user places a module in the dataflow graph or modifies an existing one, a parsing agent is run on the resulting C or FORTRAN code

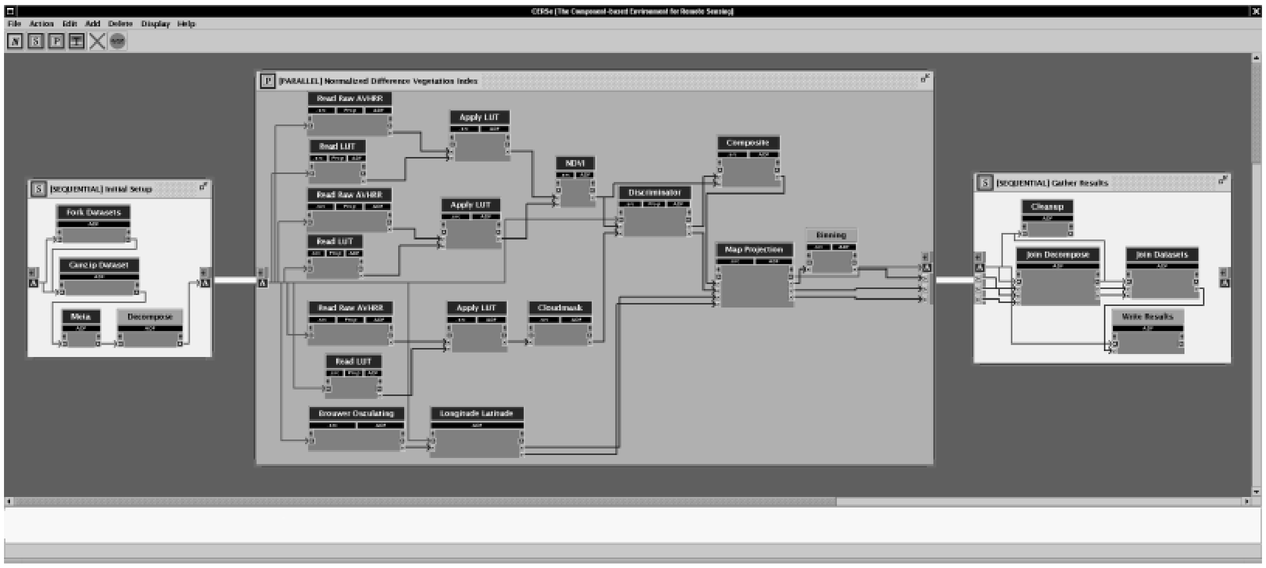


Figure 3. Graphical Editor Agent

which extracts this information. Coven then has a form of introspection, which is the ability to query a module and determine what its interface is. This information is then stored in the attributed graph.

Agents exist which take this information and can alter the way in which the user visualizes the dataflow graph. For instance, textual descriptions of each data line can be shown which can assist in visualizing the process as well as interconnecting modules.

Another agent is the code generator. This agent transforms the user-specified dataflow graph into a sequential listing of modules to run. Those modules marked as parallel are scheduled to run on each node of the Beowulf cluster. Sequential modules are run only on the master node of the cluster. The Code Generator also labels which threads each module will run under and creates directives that the driver uses at runtime.

5. Related Work Comparison

Coven provides an input / output relationship between interconnect modules very similar to Khoros. Unlike Khoros, in which each module is a separate sequential program, modules in Coven are dynamically loaded and run within a single parallel program. Data passes between modules using the TPH interface and is maintained completely within main memory instead of using files as in Khoros.

OSSIM provides a parallel runtime environment for remote sensing applications. An internal data structure called a *tile* is used for partitioning of data and distribution of data between modules. This approach is similar to the TPH approach in that it provides a data partitioning mechanism but not as flexible in that it does not provide a way to keep related data together or provide a tagged access mechanism. While a GUI for dataflow graph generation is planned, OSSIM does not yet have a tool such as the graphical editor agent of Coven.

Cactus is a PSE for grid-based applications only. As with OSSIM, Cactus does not yet provide a GUI for dataflow graph creation. Coven is based on a PSE designed for non-cyclic dataflow (such as remote sensing) applications. Coven attempts to generalize the PSE for parallel programs to work in radically different domains such as grid-based, iterative, and data rearranging computations.

6. Prototype PSEs

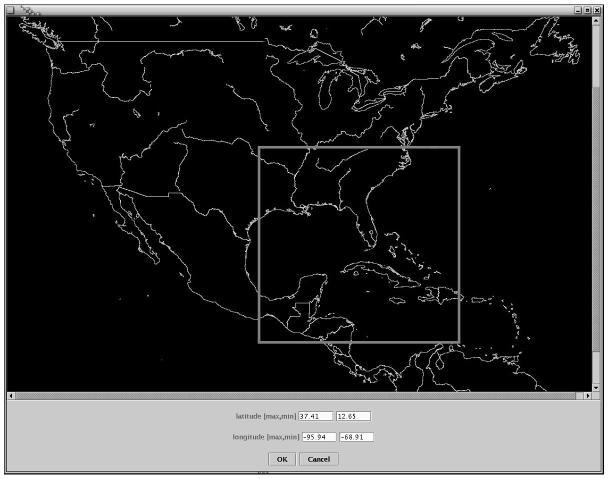
Two prototype environments have been created from Coven at this time. The first is *The Component-based Environment for Remote Sensing (CERSe)* [5, 4].

The second is *Medea*, a PSE for n-body simulations.

6.1. CERSe

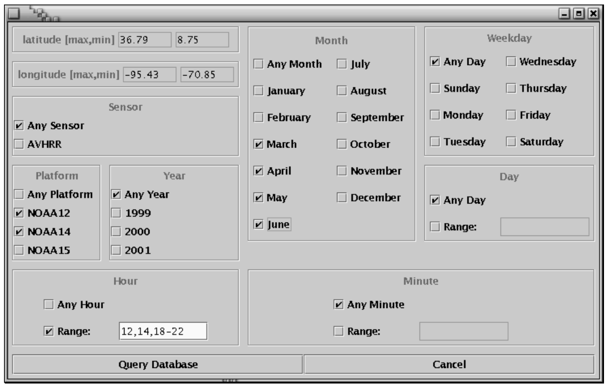
CERSe targets processing of satellite data and common remote sensing algorithms such as Normalized Difference Vegetation Index (NDVI), cloud masking, and Sea Surface Temperature (SST) [2]. Individual datasets are recombined and fused together with other datasets to create a composite result. This composite is projected into a standard coordinate system for visualization.

CERSe customizes the Coven framework by providing tools for data selection and visualization. Agents were created which prompt the user for a region of interest (Figure 4)


latitude [max,min] 37.41 12.65

longitude [max,min] -95.94 -68.91

Figure 4. Latitude / Longitude Selector


latitude [max,min] 36.79 8.75

longitude [max,min] -95.43 -70.85

Sensor: [✓ Any Sensor] ☐ AVHRR

Platform: ☐ Any Platform [✓ NOAA12] [✓ NOAA14] ☐ NOAA15

Year: [✓ Any Year] ☐ 1999 ☐ 2000 ☐ 2001

Month: ☐ Any Month ☐ January ☐ February [✓ March] [✓ April] [✓ May] [✓ June] ☐ July ☐ August ☐ September ☐ October ☐ November ☐ December

Weekday: [✓ Any Day] ☐ Sunday ☐ Monday ☐ Tuesday ☐ Wednesday ☐ Thursday ☐ Friday ☐ Saturday

Day: [✓ Any Day] ☐ Range:

Hour: ☐ Any Hour [✓ Range:] 12,14,18-22

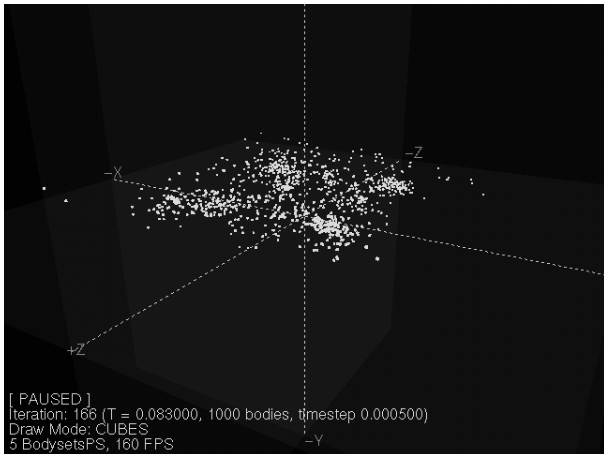
Minute: [✓ Any Minute] ☐ Range:

Figure 5. Database Dataset Selector

and then query a relational database for available datasets which meet the requirements (Figure 5). Users are also able to constrain the search to meet temporal criteria such as time of day, week, month or year as well as which satellites are to be used. An agent then estimates the overlap of each selected dataset with the chosen bounding box and presents this to the user so that they may select those datasets which they feel best fit the region. During runtime, results are sent back from the Coven runtime driver to an agent which visualizes intermediate results. This has the effect of being able to watch as the composite projected image evolves over time.

6.2. Medea

The Medea prototype environment based on the Coven framework is for n-body simulations. An interface for creating applications which is very similar to CERSe is provided. The user can specify the number of bodies to simulate and an Open-GL 3-D agent is available for visualizing



[PAUSED]
Iteration: 166 (T = 0.083000, 1000 bodies, timestep 0.000500)
Draw Mode: CUBES
5 BodysetsPS, 160 FPS

Figure 6. Medea Open-GL 3-D Visualization Agent

results in real-time (Figure 6). This agent provides a fully controllable 3-D world in which the user can watch the simulation evolve over time. Custom modules were built which write checkpoint information to disk for off line visualization. Modules have been built and tested which alter the governing physical equations, the level of gravity, and the interaction between colliding bodies.

7. Performance Results

A sequential NASA legacy remote sensing code was converted into a parallel Coven application within the CERSe prototype PSE for use as a performance test. This application computes the Normalized Difference Vegetation Index (NDVI) [2], masks clouds, and projects the output to a common coordinate system.

The Beowulf cluster used as a testbed consisted of 32 nodes with dual 1GHz Pentium III processors and 1GB memory. The nodes were connected by Fast Ethernet and running Scyld Beowulf with a Scyld modified 2.2.19 Linux kernel. Each node had 60GB of disk space. For these tests remotely sensed Advanced Very High Resolution Radiometer (AVHRR) satellite data was replicated and placed on the local disks of each processor. This allows fair, local access to all data files without concern for overhead due to network access of data. A parallel MPI version (without use of Coven) was not implemented for this problem. The Coven-based version of this code running on a single processor imposed a 12% overhead in execution time over the sequential version. This overhead is due to the use of components (queues, threads, TPHs, etc.) which do not begin to show benefit without parallelism. Figure 7 is a comparison of the normalized execution time of the CERSe NDVI application

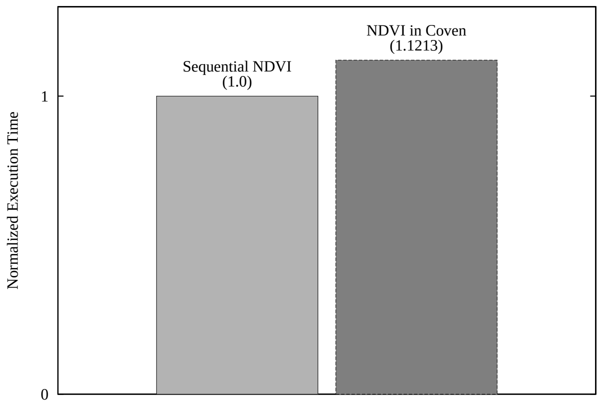


Figure 7. Overhead Incurred by Uniprocessor NDVI Application

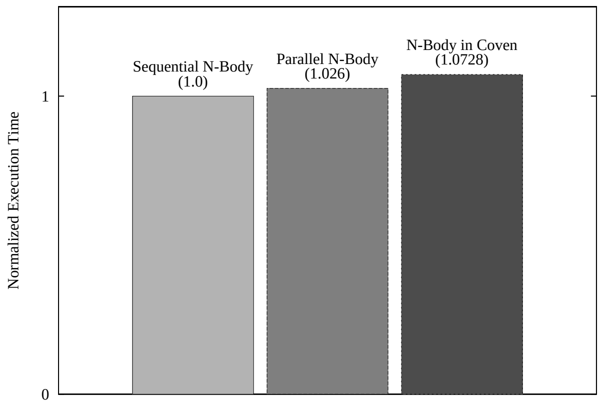


Figure 9. Overheads Incurred by Uniprocessor N-Body Application

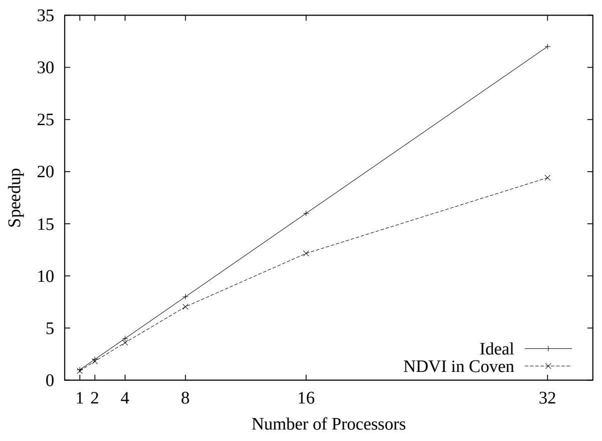


Figure 8. Speedup for NDVI Application in Coven

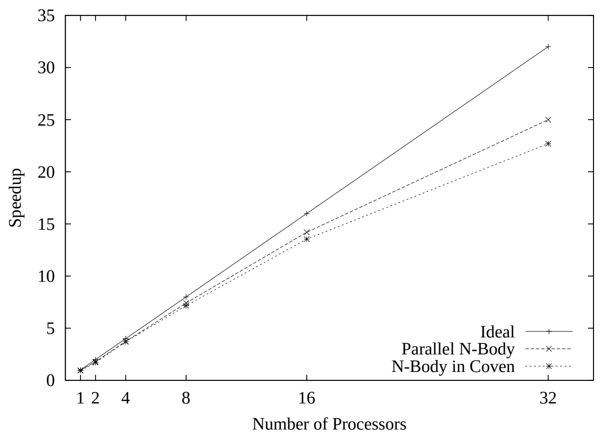


Figure 10. Speedup for N-Body Application in Coven

running on one processor when compared to the sequential version. Figure 8 is the speedup graph of this application running on up to 32 nodes of the Beowulf cluster. For this graph the execution time of the sequential program was used for speedup calculations. Figure 8 shows good scalability to around 16 compute nodes. The tapering off effect seen when 32 nodes are used is due to a sequential portion of the NDVI code. This code segment (module) reassembles results of dataset partitions into a final result. When more compute nodes are employed the node responsible for this module becomes overloaded with dataset partitions to process. Modules which assemble the results in parallel are being designed to address this drop off in scalability.

The second set of tests were performed using an n-body application. A sequential application was used for speedup calculations and a simple parallel version was implemented. Medea was used to implement a Coven-based n-body program and tests were performed on the same cluster as the NDVI application to compare the three codes. Figure 9 is a comparison of the normalized execution time of the parallel n-body and Medea n-body applications when compared to the sequential version. Figure 10 is the speedup graph of this application running on up to 32 nodes of the Beowulf cluster. The Coven-based parallel n-body code runs slightly slower than the hand-coded MPI parallel program. This example indicates the level of overhead incurred by the modularity and abstraction provided by the Coven environment.

The design of the n-body program with Medea did require creation of new system modules to do parallel communication with other parallel modules. These modules now exist in the module library which is anticipated in time to become an extensive library.

8. Conclusions and Future Work

In this work, we have presented a customizable framework for the creation of problem solving environments. The use of this framework to build domain-specific problems solving environments capable of taking advantage of Beowulf clusters has been demonstrated. Applications built from the prototype environments were found to have reasonably high performance. The use of the framework has also shown the possibility of identifying a common core for problem solving environments which can be reused from one PSE to the next, taking advantage of the framework to speed the PSE construction process. In the future we plan to continue to investigate the flexibility of the framework by creating additional prototype PSEs from Coven in new domains, and to continue to study the effectiveness of the PSEs themselves as well as the Coven framework.

Acknowledgments

This work was supported in part by NASA grant NAG5-8605 and by the ERC Program of the National Science Foundation under Award Number EEC-9731680.

References

[1] M. Cannataro, S. D. Gregorio, R. Rongo, W. Spataro, G. Spezzano, and D. Talia. A Parallel Cellular Automata Environment on Multicomputers for Computational Science. *Parallel Computing*, 21(5):803–823, May 1995.

[2] A. P. Cracknell. *The Advanced Very High Resolution Radiometer*. Taylor and Francis, 1997.

[3] J. E. Cuny, R. A. Dunn, S. T. Hackstadt, C. W. Harrop, H. H. Hersey, A. D. Malony, and D. R. Toomey. Building Domain-Specific Environments for Computational Science: A Case Study in Seismic Tomography. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(3):179–196, Fall 1997.

[4] N. A. DeBardeleben, W. B. Ligon III, S. Pandit, and D. C. Stanzione Jr. CERSe - a Tool for High Performance Remote Sensing Application Development. In *Science and Data Processing Workshop 2002*, February 2002.

[5] 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*, March 2002.

[6] K. Melero. Open Source Software Image Map Documentation. http://www.ossim.org, 2001.

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

[8] E. Seidel. Cactus. *IEEE Computational Science & Engineering*, 1999.

[9] G. Spezzano, D. Talia, S. D. Gregorio, R. Ronga, and W. Spataro. A Parallel Cellular Tool for Interactive Modeling and Simulation. *Computational Science and Engineering*, 3(3):33–43, Fall 1996.

[10] D. C. Stanzione Jr. *Problem Solving Environment Infrastructure for High Performance Computer Systems*. PhD thesis, Clemson University, 12 2000.

[11] 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.

[12] T. Sterling. *Beowulf Cluster Computing with Linux*. The MIT Press, 2001.