Problem Solving Environmets

Romanowski Karol, Psiuk Marek

Institute of Computer Science, AGH, al. Mickiewicza 30, 30-059, Kraków, Poland
email: [psiuk,kromanow]@student.uci.agh.edu.pl
URL: http://student.agh.edu.pl/ kromanow/index.php?modules=studia

Abstract

There is a wide variety of Problem Solving Environments. In our elaboration we will focus on describing a few of them and on some important PSE issues. PSEs solve many different problems like: partial differential equations; modeling and analysis; elliptic boundary value; large scale optimization problems ; large, sparse, unstructured linear systms of equations. So simply PSEs provides solution for those listed problems (and many others) with the use of Computer System - sounds very useful.

1 Introduction

What are PSEs?

> "A PSE is a computer system that provides all the computational facilities needed to solve a target class of problems. These features include advanced solution methods, automatic and semiautomatic selection of solution methods, and ways to easily incorporate novel solution methods. Moreover, PSEs use the language of the target class of problems, so users can run them without specialized knowledge of the underlying computer hardware or software. By exploiting modern technologies such as interactive color graphics, powerful processors, and networks of specialized services, PSEs can track extended problem solving tasks and allow users to review them easily. Overall, they create a framework that is all things to all people: they solve simple or complex problems, support rapid prototyping or detailed analysis, and can be used in introductory education or at the frontiers of science." [1]

In all PSEs we can point some which are more important. It is triggered by the fact that those PSEs are used to construction of other PSEs. So you can create your own PSE designed for solving your own problems. A good example of such PSE is: CECAAD - Clemson Environment for Computer Aided Application Design. It is the fundamental project, upon which all others are based . CECAAD tries to identify common elements needed for the construction

[1]"Computer as Thinker/Doer: Problem-Solving Environments for Computational Science" by Stratis Gallopoulos, Elias Houstis and John Rice (IEEE Computational Science and Engineering , Summer 1994)

of problem solving environments . Several environment projects are ongoing which use the CECAAD infrastructure, for example:

RCADE - (Reconfigurable Computing Application Design Environment), which seeks to make designing hardware for FPGA-based reconfigurable computing platforms as simple as writing software

CECAAD Parallel Programming Environment

Coven

System like CECAAD were created as an answer for the fact that many scientists and engineers writing code to perform research don't make use of the available tools. This phenomenon appears to be the result of the reluctance of scientists and engineers to undertake the additional task of learning to write parallel programs. The process of learning a new way of structuring a program is too much of a distraction from performing their core research. Computer Scientists have attempted to alleviate the problem in three ways: By creating compilers which can automatically detect parallelism, by creating tools and languages to make the creation of parallel programs simpler, and by creating the application programs themselves in the form of general-purpose solvers or custom codes.

Following sections: 2 to 4, were dedicated to describing PSEs similar to CECAAD (including CECAAD itself), which purpose is to design other PSEs. We have decided that major focus should be placed on those, mentioned above, PSEs. In the another part of our elaboration we tried to discuss major PSEs issues and expose PSEs that deal with them.

2 Characteristics of CECAAD

We will now take a closer look on CECAAD.

2.1 Interaction discussion

We will discuss the interaction between the parts of which CECAAD consist. The parts of the CECAAD system are as follows:

Algorithm Description Format - ADF is the format for all designs in the system

Design Manager - provides access to designs, and facilities for notification of Agents . To provide this notification, the Design Manager also maintains a list of Agents and the designs to which they are attached.

Agents - the tools in CECAAD which manipulate designs. To gain access to designs , Agents contain a reference to the Design Manager in the system. Agents may or may not interact with users. If an Agent interacts with users, it maintains a reference to a Display unto which it is mapped.

Displays - graphical interaction sessions with a programmer. They allow multiple Agents to communicate with a programmer using a unified interface. This is done by each Agent taking over the Display when the Agent wants to communicate with the programmer. Displays maintain references to the Agents which have been mapped unto them, and a list of Frames. Frames

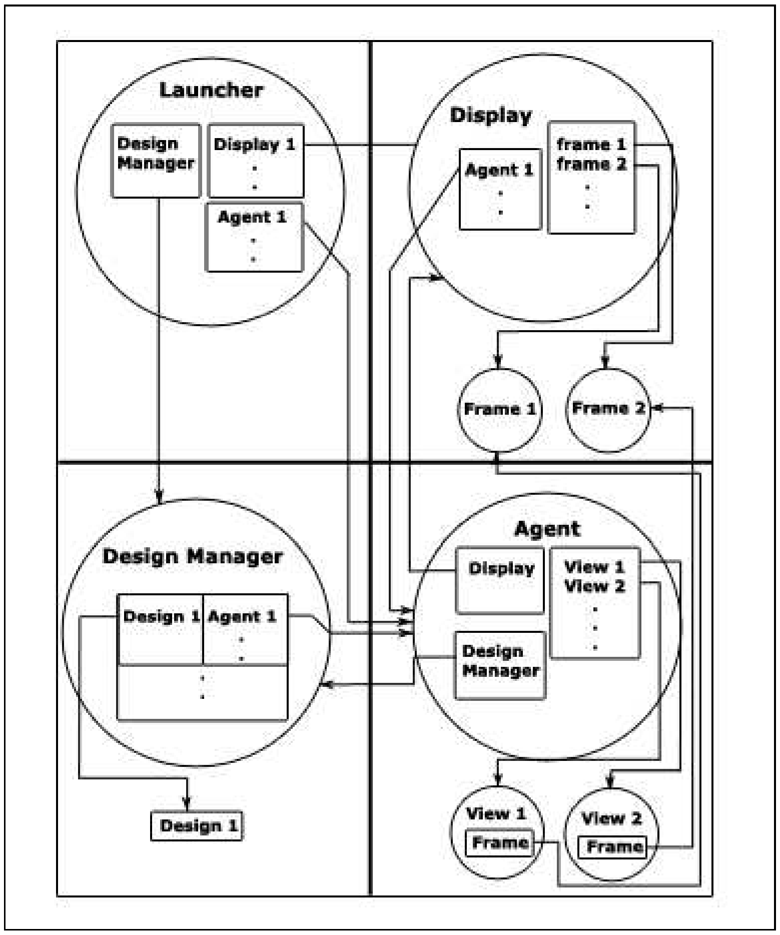

Fig. 1: Brief overview

are established elements in graphical environments. They are boxes on the screen containing a set of graphical elements.

Views - interfaces contained entirely within a Frame. When an Agent takes over the Display, each View gets a Frame from the Display.

Launcher - creates Agents and Displays, and controls the mapping between them. To provide controlled access there must be only one Design Manager accessing designs at a given time. The Launcher makes sure there is only one Design Manager by being the only part of the CECAAD system allowed to create a Design Manager.

2.2 Detailed description

ADF is a generic format for storing an attributed graph. Virtually any algorithm in any application can be stored as an attributed graph one way or another. The attribute format makes ADF extremely flexible – not only can it represent many different problems, it can represent them at many different levels of abstraction. The components of ADF are implemented as a set of Java classes .

Agents are entities which manipulate ADF designs. CECAAD provides a number of agents, and extensive support for creating new agents. The ADF Manager is responsible for coordinating the actions of the agents on the shared design . The primary role of the ADF Manager is to handle all I/O associated with ADF designs. The ADF Manager also manages libraries of ADF designs. Libraries exist either as SQL databases , or as directories on a filesystem. The ADF Manager supports a rich set of queries on the design libraries, but only a subset of the searching functionality works if the library is not in a database. The Manager allows agents (and therefore users) access to both types of libraries simultaneously. The manager also keeps all running agents notified of changes in the currently active designs.

Displays are windows where agents can interact with users. Multiple agents can interact with a user through a single display, and agents can move between displays or use multiple displays. An agent describes to the display how it wishes to appear and interact with the users through its views.

2.3 Summary

The parts of the CECAAD system work together to provide the infrastructure in which to build environments. ADF provides the means of representing designs. The Design Manager stores the designs, provides access to the designs, and allows for the notification of Agents when a design changes. The Agents modify designs through the Design Manager to which they maintain a reference. If an Agent interfaces with a user, the Agent must have a View and a Display . Views provide interfaces to the user. Views are extensible to provide an easy means of creating new interfaces with them. The Displays provide coordination of View's interfaces to the user . The Launcher coordinates the creation of Agents, Displays, and the Design Manager. By providing set of Agents and configuring the Launcher, a development environment is created.

3 Coven

Coven is a framework for creating problem solving environments for parallel computers. This framework encourages collaborative software development through component-based software design.

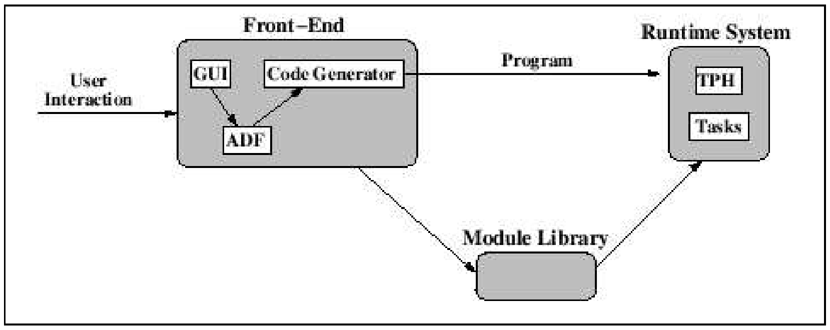

Fig. 2: Coven architecture

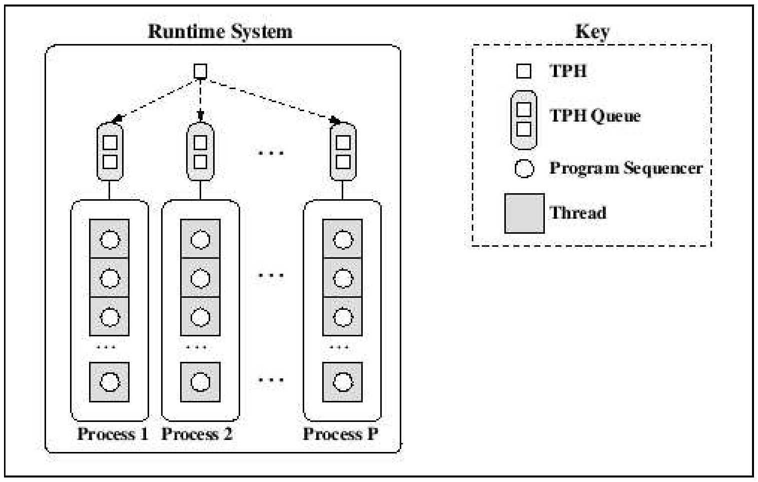

Fig. 3: Coven runtime driver

3.1 Architecture

Coven is comprised of a Java GUI front-end which hooks to a parallel back-end written in C using MPI. The hooks are accomplished through a Coven programming language which is compiled . A custom module parser to ease module programming is available as well. At Coven's core is a custom data structure - TPH [2] for keeping related data together as well as easing data flow between modules. Coven 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. It is described on figure number 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

[2]Tagged Partition Handle

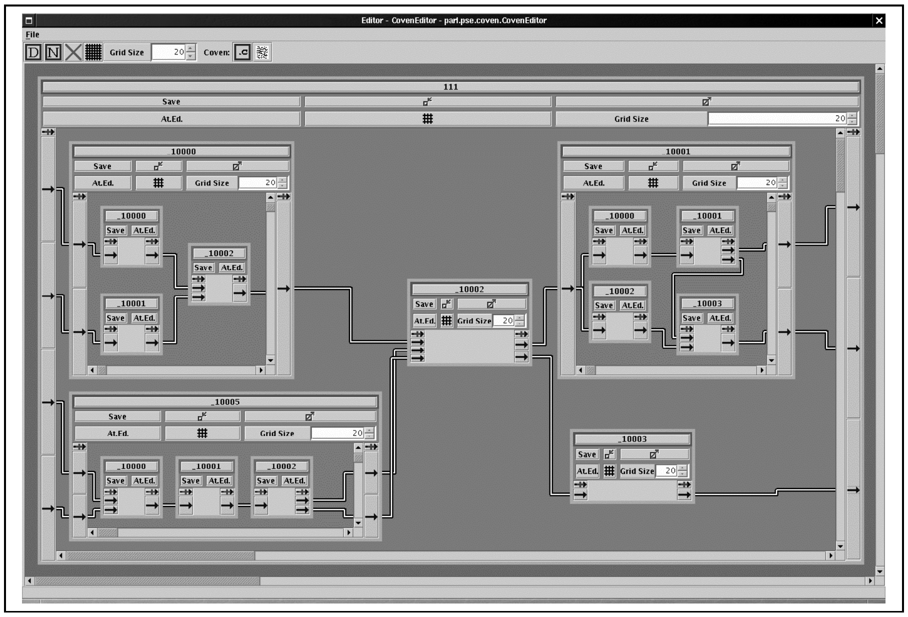

Fig. 4: Coven's graphical editor agent

the runtime driver (described in Figure 3) on a parallel computer for execution. All data within Coven's runtime driver is encapsulated into 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.

3.2 Gui frontend

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 handcoded or a tool such as the code generator can assist in this process.

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. Please examine it at figure 4

3.3 Coven versus Cecaad

The Coven framework is built on prior work in PSE toolkits done in CECAAD. 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 is built 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 to provide a complete PSE.

3.4 Summary

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.

4 PSEware

The PSEware project targets research on archetypes, software tools, and computer algebra systems for problem solving environments (PSEs). The main aim of this project is to create a toolkit for building problem-solving environments.

4.1 Overview

PSEware is a multi-institutional, multidisciplinary research project on Problem Solving Environments (PSEs) focused on symbolic computation, user interfaces and collaborative technologies for parallel object-oriented programming. The PSEware (pronounced SEAware) project is driven by a several applications, including:

numerical tokamak

cosmology modelling

environmental airshed models - Airshed Modeler (University of California - Irvine / Caltech) [3]

The Airshed Modeler is a large scale system for modeling the fluid flow and chemical interactions of air over a geographic region, particularly for smog models. Its GUI is TCL-based, and the computational components are

[3] http://www.eng.uci.edu/ maeadmin/Faculty/dabdub/dd1.html

scalable parallel codes. The Airshed Modeler has been successfully used for both research and teaching.

sparse linear system analysis - Linear System Analyzer (Indiana University/LANL) [4]

The Linear System Analyzer (LSA) is a modular system for exploring and analyzing solution strategies for large, sparse linear systems of equations. A visual programming GUI is provided, which allows a user to create and connect modules performing a variety of sparse matrix manipulations . The modules can be started on any machine on the network, and an extensive information subsystem is part of the LSA design.

collaboration system for mixed symbolic-numeric computing - Virtual Collaboratorium (LANL/IU) [5]

The Virtual Collaboratorium is a set of tools for distributed code development across the Web . Its most familiar part is a distributed CVS (code control system), with a drag and drop GUI front end.

navigation and generation of FFT programs - FFT Navigator (Drexel) [6]

The FFT Navigator is a high level Maple system which allows a user to explore the combinatorially large space of possible FFT transformations, to find one which is optimal for a particular application.

high performance object-oriented code generation from symbolic representation

soliton exploration - Soliton Explorer (Drexel) [7]

The Soliton Explorer is an Iris Explorer based PSE for examining solitons mathematically.

4.2 Major Issues

- PSEs that support the transformation of symbolic problem definitions to parallel object-oriented programs that can be executed efficiently on a variety of sequential and parallel architectures.
- Object-oriented libraries of parallel program templates or archetypes that can be refined to obtain specific applications by using ideas such as inheritance.
- User-interface archetypes for scientific and engineering PSEs that can be refined to construct a PSE for specific problem domain.
- Technologies for collaboration and ubiquitous distributed computing focused on the internet, the Web, and Java, with the goal of applying these technologies to distributed collaborative PSEs.
- Generic PSE components and tools to give non-experts in computer science the ability to rapidly and easily construct their own PSEs.

[4] http://www.extreme.indiana.edu/pseware/lsa/index.html

[5] http://www.extreme.indiana.edu/pseware/vc/index.html

[6] http ://www.mcs.drexel.edu/bchar/pse_public/pse_home.html

[7] http://www.mcs.drexel.edu/ bchar/pse_public/pse_home.html

4.3 Summary

As we already know problem-solving environment is an integrated collection of software tools that facilitates problem-solving in some domain. A significant amount of work has been carried out in the area of PSEs for problems specified by differential equations. This project has a different emphasis from much of the work in PSEs by focusing on symbolic specifications, methods of reuse of object structures for user interfaces and parallel execution, component technologies for PSEs and collaboration technologies for problem solving. A scientist or engineer should be able to specify a problem symbolically with the notation that they use in communication with each other. PSEs should help them refine their symbolic specification to efficient parallel object-oriented programs. Parallel program archetypes should facilitate this refinement. The symbolic and graphical user interfaces should also be object-oriented to facilitate reuse of the user interface for large classes of scientific applications.

Most modern scientific computing research is done by groups of people. A PSE should facilitate collaboration among a distributed group as well as empower a single person. This project researchs adapting internet-based collaboration technologies to help collections of people use shared PSEs to solve problems together. A collaborative problem-solving environment is also an ideal teaching tool.

The PSEware project targets the generic underpinnings of problem solving environments, and one application of that research is the identification and creation of components and glue technologies which will allow application scientists to quickly and easily build their own PSEs.

5 Overview of the main PSEs' issues

5.1 Partial Differential Equations

Partial differential equations are at the foundation of much of computational science. Most physical phenomena depend in complex ways on space and time. Examples include fluid flow, heat transfer, nuclear and chemical and reactions and population dynamics. Computational scientists often seek to gain understanding of such phenomena by casting fundamental principles, such as conservation of mass, momentum and energy in the form of mathematical models of the underlying physical phenomena. Usually a mathematical model requires more than one independent variable to characterize the state of the physical system. For example, to describe a general fluid flow usually requires that the physical variables of interest, say pressure, density and velocity , be dependent on time and three space variables. If a mathematical model involves more than one independent and if at least one of the physical variables of interest is nonconstant with respect to space or time, then the mathematical model will involve a partial differential equation (PDE). As we can see the class of problems which involve PDE is very wide.

Here are some PDE packages, wich are of course PSEs:

5.1.1 Diffpack

- Software for Finite Element Analysis and Partial Differential Equations

Diffpack is an object oriented development framework for the solution of partial differential equations. Diffpack is based on the latest developments in Object-Oriented Numerics, what gives unsurpassed modeling flexibility while satisfying the strictest demand for computational efficiency. It supports human abstraction and increases problem solving abilities. Currently the newest version of Diffpack is The Diffpack 4.0 Product Line which consists of the following units:

– Diffpack Kernel 4.0
– Diffpack Adaptivity Toolbox 1.2
– Diffpack Datafilter Toolbox 1.2
– Diffpack Multilevel Toolbox 1.2
– Diffpack Parallel Toolbox I 1.2

These products gives a complete set of C++ tools to create advanced numerical applications . Diffpack allows easy modification and combination of all numerical building blocks, resulting in few restrictions on the types of PDEs user can solve. Diffpack's layered design ensures flexible APIs and computational efficiency competing with tailored FORTRAN codes. Diffpack is maintained with same source on all platforms. Technology leaders are using Diffpack in areas like geology, geophysics, chemistry, finance, medicine, physics, electronics, mechanics and mathematics.
Here are some examples of application areas:

- The Laplace, Poisson, Helmholtz, heat, and wave equations in general 1D/2D/3D geometries
- Structural analysis described by 2D/3D linear elasticity theory
- Compressible and incompressible 2D/3D Newtonian fluid flow
- Optimal control and optimization problems in forming processes
- Electrical activity in the heart
- Deformation of tissue during surgery
- Shallow water waves and tsunami propagation, also with weakly dispersive and nonlinear effects
- Fully nonlinear 3D water waves
- Continuous Markov processes, modeling e.g. random vibrations of simple structures

but there are many many more.

5.1.2 VECFEM

VECtorised Finite Element Method solves nonlinear boundary value problems. It has several components , including the graphical user interface xvem as well as the VECFEM kernel. It was created by the research group for numerical algorithms on supercomputers at the Computing Center of the University of Karlsruhe. VECFEM includes interfaces to well-known CAD and visualization tools. VECFEM was originally commercial software but has now been released to the public domain.

It is used in the analysis of many types of problems, including those in heat transfer, structural analysis, fluid dynamics and electromagnetism.

5.2 Mesh generation

An important step for the numerical simulation of physical phenomena is the transformation of the underlying differential equation into a finite dimensional space. In the considered domain the resulting partial differential equation is approximated using numerical methods on finite intervals. Determining the finite intervals requires a discretization of the domain . This discretization is in most cases obtained by generating a net. The nodes of the net consist of nodes on the domain boundary as well as nodes in the interior of the domain (Steiner points). The exact type of discretization is determined by the numerical solution method. While, especially in complex domains, finite element methods often use unstructured meshes (i.e. triangulations), difference methods require in any case block-structured nets, i.e. grids. Structured meshes are typically easier to compute with (saving a constant factor in runtime) but may require more elements or worse-shaped elements. Unstructured meshes are often computed using quadtrees, or by Delaunay triangulation of point sets; however there are quite varied approaches for selecting the points to be triangulated.

There has now been considerable theoretical work in the geometry community on these problems , complementing and building on practical work in the numerical community. There is also beginning to be a convergence of these communities, in which theoretical work is fed back into practical mesh generation applications. Theoretically, the preferred type of mesh is the triangulation or simplicial complex, but in mesh generation practice quadrilaterals or higher dimensional cubical element shapes are preferred (both because fewer are typically needed and because they have better numerical properties). Remaining problem areas in the theory of meshing include triangulations in dimensions higher than two, meshes with cubical elements, mesh smoothing , mesh decimation and multigrid methods, and data structures for efficient implementation of meshing algorithms.

There are many research groups working on theoretical problems which provide commercial or public domain packages.

Now we will introduce some examples of mesh generators:

5.2.1 LaGriT

LaGriT is a software tool for generating, editing and optimizing multi-material unstructured finite element grids (triangles and tetrahedra). LaGriT maintains the geometric integrity of complex input volumes, surfaces, and geologic data and produces an optimal grid (Delaunay , Voronoi) elements. The LaGriT grid generation system has many special features tailored to geological applications.

Geologic applications for grids produced with LaGriT are modeling subsurface porous flow and reactive chemical transport by finite element and finite difference methods. LaGriT is also used as the first step in quality analysis of geometric data.

Projects in which LaGriT is utilized include:

- Yucca Mountain Site Characterization Project (YMP)
- Environmental Restoration at Los Alamos and Savannah River
- Oil and Gas Reservoir Modeling
- Semiconductor Design Modeling
- High Speed Hydrodynamics

There is a wide variety of geological applications (PSEs) where accurate representation of complex engineering systems and geologic structure and stratigraphy is critical to producing accurate numerical models of fluid flow and mass transport. Oil and gas reservoir production, groundwater resource development and waste disposal in a geologic repository are examples of the areas where modeling is used to predict the long term behavior of a system. In all the systems, grid generation is a key link between the geoscientific information systems and numerical models. Grids must capture complex geometry and insure the computationals are optimized to produce accurate and stable solutions. LaGriT is a toolbox library with functions to produce 2D and 3D grids of elements that are tetrahedral, triangular, hexahedral and quadrilaterals .

5.2.2 GTS

GTS stands for the GNU Triangulated Surface Library. It is an Open Source Free Software Library intended to provide a set of useful functions to deal with 3D surfaces meshed with interconnected triangles. The source code is available free of charge under the Free Software LGPL license.

The code is written entirely in C with an object-oriented approach based mostly on the design of GTK+. Careful attention is paid to performance related issues as the initial goal of GTS is to provide a simple and efficient library to scientists dealing with 3D computational surface meshes.

A brief summary of its main features:

- Simple object-oriented structure giving easy access to topological properties.
- 2D dynamic Delaunay and constrained Delaunay triangulations.
- Robust geometric predicates (orientation, in circle) using fast adaptive floating point arithmetic (adapted from the fine work of Jonathan R. Shewchuk).
- Robust set operations on surfaces (union, intersection, difference).
- Surface refinement and coarsening (multiresolution models).
- Dynamic view-independent continuous level-of-detail.
- Preliminary support for view-dependent level-of-detail.
- Bounding-boxes trees and Kd-trees for efficient point location and collision/intersection detection.
- Graph operations: traversal, graph partitioning.
- Metric operations (area, volume, curvature ...).
- Triangle strips generation for fast rendering.

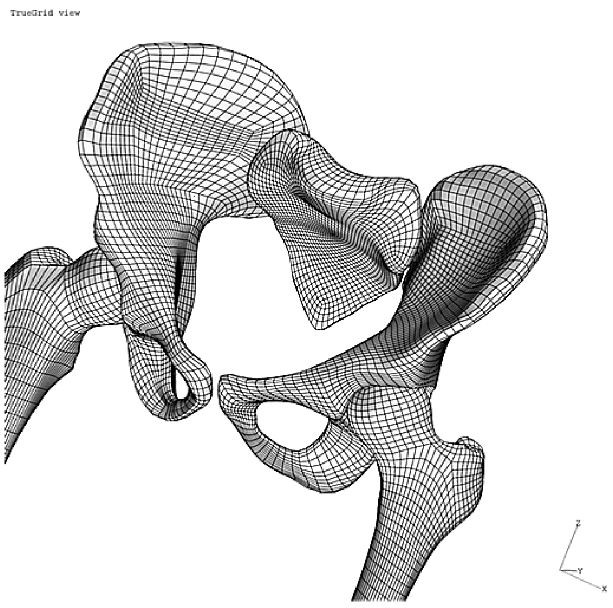

Fig. 5: A sample of trueGrid's biomechanical meshes - quality hex mesh of human hip bones

5.2.3 TrueGrid

TrueGrid provides high quality hexahedral grid and mesh generation for fluids and structures . TrueGrid is a general purpose mesh generation program with sophisticated relaxation and parameterization capabilities. It has been optimized to produce high quality, structured, multi-block hex meshes or grids and serves as a preprocessor to over 30 of today's most popular analysis codes.

5.3 Graph Partitioning

Graph partitioning is a technique for executing a set of tasks in parallel so as to balance the load and minimize communications among processors.

Here is a simple model of a discrete event system we can use to describe partitioning. We model the system as a graph G=(N,E), with nodes N and edges E connecting them. Each node represents a physical component (like a subcircuit) to be assigned to a processor, and each edge represents a direct dependence between components (like a wire). Furthermore, each node n has a weight W_n representing the work required to simulate it, and each edge e has a weight W_e representing how much information must be passed along it. A natural set of goals for partitioning nodes to processors are the following.

Each processor should have an approximately equal amount of work to do, i.e. the sum of the weights W_n of the nodes assigned to each processor should be

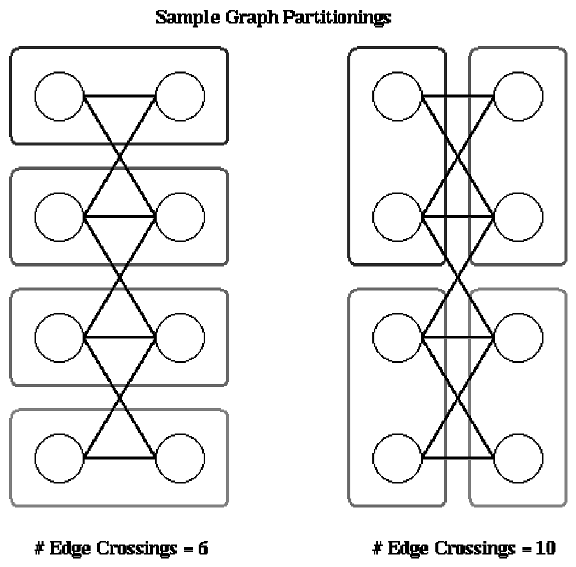

Fig. 6: examples of simple graph partitioning

approximately the same for each processor. Dividing up work equally this way is also called load balancing. The amount of communication should be minimized, i.e. the sum of the weights W_e of edges crossing processor boundaries should be minimized.

For example, in the figure 6 ovals are drawn around nodes to be assigned to the same processor. Each partitioning shown assigns two nodes each to four processors. The partition on the left only has 6 edges crossing processor boundaries, whereas the partition on the right crosses ten, so assuming each edge has the same weight, the left partitioning requires less communication and so is superior.

This graph partitioning problem is a central one in graph theory and parallel computing. It is NP-complete, and so we can only expect approximate solutions in general. This problem arises in a variety of parallel computing problems, including the following: Sparse matrix-vector multiplication, Solving PDEs, VLSI layout (In this case the nodes are logical units on the chip, and the edges are wires connecting them. The goal is to place the units on the chip so as to minimize the numbers and lengths of the wires connecting them, which speeds up the chip), Telephone network design (This was an original motivation for these algorithms: deciding how to minimize the cost of building telephone lines connecting switches), Sparse Gaussian elimination (Graph partitioning can be used to reorder the rows and columns of the matrix resulting in dramatically decrease the number of nonzero entries created during elimination and the number of floating point operations required), Physical mapping of DNA.

5.3.1 CHACO

Before a calculation can be performed on a parallel computer, it must first be decomposed into tasks which are assigned to different processors. Efficient use of the machine requires that each processor have about the same amount of work to do and that the quantity of interprocessor communication is kept small. Finding an optimal decomposition is provably hard, but due to its practical importance, a great deal of effort has been devoted to developing heuristics for this problem.

Researchers at Sandia Labs have developed a variety of innovative algorithms for this decomposition problem and implemented them in a software package called Chaco. This code is being used at most of the major parallel computing centers in the country to simplify the development of parallel applications, and to ensure that optimal performance is obtained. Chaco has contributed to a wide variety of computational studies including investigation of the molecular structure of liquid crystals, evaluating the design of a chemical vapor deposition reactor and modeling automobile collisions.

The algorithms developed for Chaco have also been successfully applied to a number of problems which have nothing to do with parallel computing. These include the determination of genomic sequences (a critical part of the Human Genome Project), the design of space-efficient circuit placements, organization of databases for efficient retrieval, and ordering of sparse matrices for efficient factorization.

5.3.2 METIS

METIS(Family of multilevel partitioning algorithms) is a family of programs for partitioning unstructured graphs and hypergraphs and computing fill-reducing orderings of sparse matrices. The underlying algorithms used by METIS are based on the state-of-the-art multilevel paradigm that has been shown to produce high quality results and scale to very large problems.

The METIS family consists of three different packages:

METIS - Serial graph partitioning and sparse matrix ordering. Provides the following key components: Graph partitioning, Mesh partitioning, Fill-reducing reordering. It is available both as a set of stand-alone programs and as a library.

ParMETIS - Parallel graph partitioning and sparse matrix ordering. Provides the following key components: Static graph partitioning, Static mesh partitioning, Dynamic graph partitioning , Fill-reducing reordering. It is available as an MPI-based (described later) library.

hMETIS - Serial hypergraph partitioning. It has been specifically optimized for partitioning large VLSI circuits. It is available both as a set of stand-alone programs and as a library.

5.4 Linear Solvers

Many PSEs deal with linear algebra problems. Therefore optimization of linear algebra operations is very important so that our PSE using them is fast enough to fulfill our needs. Many research groups worked on theoretical problems and created packages which consist of optimized basic linear algebra subprograms. There are many such libaries like BLAS or ScaLAPACK which are used to ensure that optimal speed and stability is obtained. We would like to present here SuperLU package.

5.4.1 SuperLU

SuperLU is a high performance sparse linear system solver. SuperLU package is a set of subroutines to solve a sparse linear system $A * X = B$. It implements Gaussian elimination with partial pivoting , using supernodes and BLAS to optimize performance. It has achieved over 120 Mflops on realistic examples on an IBM RS 6000/590. It is currently the fastest available algorithm for a variety of problems.

SuperLU is implemented in ANSI C, and must be compiled with standard ANSI C compilers. It supplies the BLAS in C, but for highest performance optimized BLAS should be used. Currently only the real single-precision and double-precision versions are provided. The complex versions are still under construction. The calling sequence is modeled on LAPACK.

SuperLU package comes in three different flavors:

- SuperLU for sequential machines
- SuperLU_MT for shared memory parallel machines
- SuperLU_DIST for distributed memory

SuperLU is a general purpose library for the direct solution of large, sparse, nonsymmetric systems of linear equations on high performance machines.

5.5 Message Passing Libraries

Message passing is a paradigm used widely on certain classes of parallel machines, especially those with distributed memory. Although there are many variations, the basic concept of processes communicating through messages is well understood. First we will describe the MPI - Message Passing Interface which established a well designed standard for MPL. Main advantages of MPI are portability and ease-of-use.

5.5.1 Message Passing Interface

In a distributed memory communication environment in which the higher level routines and/or abstractions are build upon lower level message passing routines the benefits of standardization are particularly apparent. Furthermore, the definition of a message passing standard, provides vendors with a clearly defined base set of routines that they can implement efficiently, or in some cases provide hardware support for, thereby enhancing scalability.

The goal of the Message Passing Interface simply stated is to develop a widely used standard for writing message-passing programs. As such the interface should establish a practical, portable, efficient, and flexible standard for message passing.

Here's a list of goals that MPI tries to achieve:

- Design an application programming interface (not necessarily for compilers or a system implementation library).
- Allow efficient communication: Avoid memory-to-memory copying and allow overlap of computation and communication and offload to communication co-processor, where available.
- Allow for implementations that can be used in a heterogeneous environment.
- Allow convenient C and Fortran 77 bindings for the interface.
- Assume a reliable communication interface: the user need not cope with communication failures. Such failures are dealt with by the underlying communication subsystem.
- Define an interface that is not too different from current practice, such as PVM, NX , Express, p4, etc., and provides extensions that allow greater flexibility.
- Define an interface that can be implemented on many vendor's platforms, with no significant changes in the underlying communication and system software.
- Semantics of the interface should be language independent.
- The interface should be designed to allow for thread-safety.

5.5.2 Parallel Virtual Machine

PVM is a software package that permits a heterogeneous collection of Unix and/or Windows computers hooked together by a network to be used as a single large parallel computer. Thus large computational problems can be solved more cost effectively by using the aggregate power and memory of many computers. The software is very portable. The source, which is available free thru netlib , has been compiled on everything from laptops to CRAYs.

PVM enables users to exploit their existing computer hardware to solve much larger problems at minimal additional cost. Hundreds of sites around the world are using PVM to solve important scientific, industrial, and medical problems in addition to PVM's use as an educational tool to teach parallel programming. With tens of thousands of users, PVM has become the de facto standard for distributed computing world-wide.

5.5.3 PVM versus MPI

The MPI Forum stated very specific goals and make each approach solve as many of those goals possible. For example, datatypes solve both heterogeneity and noncontiguous data layouts, both for messages and for files. Similarly, communicators combine both process groups with communications contexts. The MPI standard has been widely implemented and is used nearly everywhere , attesting to the extent to which these goals were achieved.

PVM had, with the exception of support for heterogeneous computing and a different approach to extensibility, different goals. In particular, PVM was aimed at providing a portable, heterogeneous environment for using clusters of machines using socket communications over TCP/IP as a parallel computer. Because of PVM's focus on socket-based communication between loosely-coupled systems , PVM places a greater emphasis on providing a distributed computing environment and on handling communication failures.

Despite their differences, PVM and MPI certainly have features in common:

- Both MPI and PVM are portable
- They provide support for heterogeneity - two processes on different machine architectures communicate with one another despite differences in byte ordering in memory or even word length.
- Both MPI and PVM permit different processes of a parallel program to execute different executable binary files.
- interoperability - possibility of communicating among processes linked with two completely different implementations.

In summary, both MPI and PVM are systems designed to provide users with libraries for writing portable, heterogeneous, MIMD [8] programs. In comparing issues, one must not confuse the MPI specification with a particular implementation subcase, such as the p4 device of MPICH [9] , which is widely used on clusters but does not define MPI.

6 Conclusion

A problem-solving environment (PSE) is a complete, integrated software environment for the computational solution of a particular problem, or class of related problems. Currently there are many PSEs. For example, PELLPACK for the parallel solution of elliptic partial differential equations, CAMEL for modelling cellular automata, and CACTUS which is being used in gravitational wave simulations, to name but a few. In addition to applications in computational science and engineering, PSEs are also being developed for use in education, financial modelling, and bioinformatics. Typically, a PSE provides support for composing, compiling, and running distributed applications in a specific problem area which is often multi-disciplinary in nature . In many cases a PSE also incorporates a certain degree of intelligence in assisting users in formulating problems, running the problem on an appropriate platform, and viewing and analysing results. A PSE may also have access to virtual libraries, sophisticated execution control systems, and advanced visualisation environments. For many applications interactivity is a key feature.

The concept of a PSE has been around for many years. Systems such as Matlab and Mathematica may be viewed as mathematical PSEs, however, they are designed for uniprocessor machines, rather than for multiprocessor servers,

[8] Multiple Instruction Multiple Data - type of parallel computing architecture where many functional units perform different operations on different data.

[9] Portable Implementation of MPI

massively parallel computers, or networks of workstations . Utilising and managing distributed high performance computing resources is important for a PSE to meet the requirements of large-scale simulations. System integration software, such as CORBA, and network software technologies, such as Java and XML, coupled with the rapid increase in network bandwidths, are now making it possible to develop sophisticated distributed PSEs. These types of PSE have the potential for profoundly changing the way high performance computing resources are used to solve problems. In the future PSEs may be the primary way in which high performance computing resources are accessed. But to achieve this supercomputing centres have to change their mode of operation since the batch queueing system currently in common use is not suitable for interactive applications involving, for example, visual steering .

References

1. http://www.cs.purdue.edu/research/cse/pses/ - Problem Solving Environments Home Page
2. http://www.math.unm.edu/ wester/aca96/Steinberg_abstract.html
3. http://www.cs.cf.ac.uk/euresco99/
4. http://www.rmcs.cranfield.ac.uk/esd/amor/intelligentProblemSolving/view
5. http://research.cs.vt.edu/pse/
6. http://www.it-innovation.soton.ac.uk/research/eng_solve.shtml
7. http://www.diffpack.com
8. http://www-unix.mcs.anl.gov/mpi/
9. http://www.nas.nasa.gov/Research/Tasks/probsolvingtasks.html
10. http://www.cs.cornell.edu/Info/Projects/NuPrl/Intro/PSE/pse.html