1997

Parallel ELLPACK 3-D Problem Solving Environment

Vassilios Verykios

Elias N. Houstis
Purdue University, enh@cs.purdue.edu

Report Number:
97-028

PARALLEL ELLPACK 3-D PROBLEM SOLVING ENVIRONMENT

Vassilios Verykios
Elias N. Houstis

Department of Computer Sciences
Purdue University
West Lafayette, IN 47907

CSD-TR 97-028
May 1997

Parallel ELLPACK 3-D Problem Solving Environment

Vassilios Verykios Elias N. Houstis

May 16, 1997

Abstract

Parallel ELLPACK (//ELLPACK) is a problem solving environment (PSE) that supports the solution of field and flow partial differential equation (PDE) problems on sequential and parallel MIMD computational platforms. In this system the parallel processing of steady-state PDE problems is supported by three domain decomposition schemes. These schemes differ with respect to parallelization of the PDE discretization phase adopted and are based on the partitioning of the associated geometric data structures (i.e., meshes and grids) into balanced subdomains. In this report, we review the general issues related to PSEs for scientific computing and discuss the available software technologies for designing and implementing their graphical user interfaces. In particular, we present the functional specifications of the 3-D pre- and post-processing user interface of the //ELLPACK system and its implementation. To support the domain decomposition methodologies of //ELLPACK, we have integrated a number of well known mesh/grid partitioning libraries in this system including CHACO 2.0 and METIS. We review the basic partitioning algorithms involved and attempt a preliminary performance evaluation of their software implementation in the context of an off-line parallel reuse methodology available in //ELLPACK. To model their performance we estimate the execution time and fixed speedup of the underlying PDE computations on two clusters of workstations with different interconnection technologies (e.g., Ethernet and ATM) and an nCUBE/2 parallel machine. The numerical results indicate the cost effectiveness of the parallel ELLPACK methodologies and infrastructure for the solution of steady-state field PDE problems on distributed memory platforms.

1 Introduction

Partial differential equation (PDE) models are used often to model the physical behavior of many artifacts. For this there are many libraries of well written PDE software. Unfortunately, each of this software is based on proprietary

data structures and assumes non-standard I/O specifications. Moreover, these libraries require several pre- and post-processing steps that are usually left to the user. These processing steps involve the solution of very difficult problems and require geometric, symbolic, and numerical computing. In the case of parallel simulation codes additional pre-processing steps are required such as partitioning of geometric and algebraic data structures. In addition the characteristics of parallel execution environments must be integrated in the associated problem solving process. In this report *we address the integration of PDE software at the graphical user interface level over a variety of sequential and parallel computational environments.* Specifically, we extend the design and implementation of the //ELLPACK user interface for 3-D PDE problems and pre- and post-processing tools, their user interfaces, and algorithmic infrastructure needed for 3-D mathematical models associated with the applicability of //ELLPACK PSE [HRW+96]. These tools include the visualization of a polygonal representation of the PDE domain, its finite element discretization, the specification of the PDE boundary conditions, and the mesh/grid decomposition in load balanced subdomains with semi-optimal interface length. The mesh decomposition tool is used to support three different parallelization approaches depending on the parallelization procedure applied to the PDE discretization phase. They are briefly described in Section 5. The 3-D domain decomposition tool is supported by three libraries of mesh partitioning heuristics, the "native" //ELLPACK [WH93] and the integrated "foreign" libraries CHACO [HL95c] and METIS [KK95c]. To assess the effectiveness of these heuristics in solving steady-state field PDE problems on distributed memory machines, we have utilized a parallel reuse methodology referred throughout as MPlus [MH96]. In this framework the PDE discretization is generated sequentially and the system obtained is block partitioned based on a p-way mesh decomposition, where p denotes the machine configuration to be used. The matrix blocks are downloaded on each processor of the targeted machine. The advantage of the method is that we reuse the highly knowledge intensive part of the PDE solver (usually 90% of the code) while solving in parallel the computationally intensive part.

This report is organized as follows. Section 2 reviews the graphical user interface technologies for multi-platforms and summarize the future trends in this area. The //ELLPACK pre- and post-processing tools supporting its GUI interface and the software infrastructures used to implement it, are reported in Section 3. In Section 4, we review a number of basic graph partitioning algorithms. These algorithms are used to implement the "native" and "foreign" mesh/grid decomposition libraries available in //ELLPACK. We have conducted several numerical experiments on two clusters of Sun workstations and an nCUBE/2 parallel machine to test the effectiveness of the data partitioning algorithms and the parallel methodologies supported in //ELLPACK. The performance data obtained and our observations are presented in Section 5.

2 Graphical User Interfaces (GUI) for PSEs

The trend in every application domain today, including scientific computing, is to encapsulate the computing power (i.e., libraries and machine environments) within a software environment that allows the user to exploit it in some "natural" form. This trend led recently to the concept of *problem solving environment* that has become very popular in the area of scientific computing. PSEs are recognized to be systems that provide all the computational facilities necessary to solve a target class of problems by communicating in the user's own terms with the domain of the target area [GHR92]. It has been widely accepted that at a minimum, a PSE consists of a *library of solvers* and a *user interface*. One of the objectives of this report is the design and implementation of the user interface of //ELLPACK PSE and the algorithmic infrastructure supporting it.

Very often user interface software is large, complex and difficult to implement, debug, and modify. A 1992 survey [MR92] found that an average of 48% of the application codes is devoted to the user interface and that about 50% of the implementation time is spent to implement the user interface portion. These numbers are probably much higher today. As interfaces become easier to use, they become harder to create [Mye94]. According to another study [Inc94] almost 97% of all software development on Unix platforms involves the implementation of a GUI. The challenge in interface development is that it requires programmers to deal with elaborate graphics, multiple ways for giving the same command, multiple asynchronous input devices (usually a keyboard and a pointing device such as a mouse), virtual time, and "semantic feedback". It is predicted that tomorrow's user interfaces will provide speech and gesture recognition, intelligent agents, and integrated multi-media. These additional functional specifications undoubtedly will increase significantly the difficulty and effort of such interface developments.

The objective of this section is to review the current software technologies for building user interfaces. Following, we present this review in terms of the three layers of software usually involved in a user interface. These are the windowing system, the toolkit, and higher level tools. The *windowing system* supports the separation of the screen into different (usually rectangular) regions, called *windows*. On top of the windowing system is the *toolkit*, which contains many common *widgets* such as menus, buttons, scroll bars, and text input fields. Above the toolkit one might find *higher-level tools*, which help the designer use the toolkit widgets.

2.1 Windowing System

A windowing system is a software package that helps the user monitor and control different contexts by separating them physically onto different parts of one or more display screens. When the programmer wants to draw application-specific parts of the interface and allow the user to manipulate them, the window

system interface must be used directly.

The first windowing systems were implemented as part of a single program or system. Later systems implemented the windowing system as an integral part of the operating system, such as SunView for Suns, Macintosh and Microsoft Windows systems. In order to allow different windowing systems to operate on the same operating system, some windowing systems, such as X and Sun's NeWS, operate as a separate process, and use the operating system's interprocess communication mechanism to connect to applications.

A windowing system can be logically divided into two layers, each of which has two parts. The *window system* or *base layer*, which implements the basic functionality of the windowing system and the *window manager* or *user interface*. The base layer consists of the *output model* which handle the display of graphics in windows and the *input model* which coordinate the access to various input devices like keyboard and mouse. The primary interface of the base layer is procedural and it is called the windowing system's *Application Programming Interface (API)*. The window manager includes all aspects that are visible to the user. The two parts of the user interface layer are the *presentation* and the *commands*. The first involves the pictures that the window manager displays, while the second supports the manipulation of windows and their contents.

Very often the development of a PSE requires the support for 3-D graphics. All of the standard output models only contain drawing operations for two dimensional objects. Two extensions to support 3-D objects are PEX [Wom92] and OpenGL [SA92]. The current version of the Silicon Graphics, Inc. (SGI) X Server supports both these standards. OpenGL is an Application Programming Interface (API) for 3-D interactive graphics developed by SGI and now administered by the OpenGL Architecture Review Board. OpenGL is the successor to the proprietary IRIS GL graphics interface (the GL stands for graphics library). OpenGL supports the X Window System via the GLX extension and it provides machine independence, for 3-D since it is available for various X platforms (SGI, Sun, etc.) and it is included as a standard part of Microsoft Windows NT. PEX is an X protocol extension developed by the X Consortium to support 3-D graphics for X. PEX's rendering functionality and style are largely influenced by the PHIGS (Programmer's Hierarchical Interactive Graphics System) and PHIGS PLUS 3-D graphics ANSI (American National Standards Institute) standards.

The earlier windowing systems assumed that a graphics package would be implemented using the windowing system. All newer systems, including the Macintosh, X, NeWS, NeXT, and Microsoft Windows, have implemented a sophisticated graphics system as part of the windowing system.

2.2 Toolkits

A *toolkit* is a library of "widgets" that can be called by application programs. Using a toolkit has the advantage that the final UI will look and act similarly to other UIs created using the same toolkit, and each application does not have

to re-write the standard functions, such as menus. A problem with toolkits is that the styles of interaction are limited to those provided. Also, the toolkits themselves are often expensive to create and difficult to use, since they may contain hundreds of procedures. It is often not clear how to use these procedures to create a desired interface.

As with the graphics package, the toolkit can be implemented either using or being used by the windowing system. When the X system was being developed, the developers could not agree on a single toolkit, so they left the toolkit to be on top of the windowing system. In X, programmers can use a variety of toolkits (for example, the Motif, OpenLook, InterViews, or tk toolkits can be used on top of X), but the window manager must usually implement its user interface from scratch.

Because the designers of X could not agree on a single look-and-feel, they created an *intrinsics* layer on which to build different widget sets, which they called xt. This layer provides the common services, such as techniques for object-oriented programming and layout control. The *widget set* layer is the collection of widgets that is implemented using the intrinsics.

Although there are many small differences among the various toolkits, people are still spending a lot of effort to convert software from Motif to the Macintosh and to Microsoft Windows. Therefore, a number of systems have been developed that try to hide the differences among the various toolkits, by providing *virtual* widgets which can be mapped into the widgets of each toolkit. Another name for these tools is *cross-platform development systems*. The programmer writes the code once using the virtual toolkit and the code will run without change on different platforms preserving the original design.

There are two styles of virtual toolkits. In one, the virtual toolkit links to the different actual toolkits on the host machine. The second style of virtual toolkit re-implements the widgets in each style. All of the toolkits that work on multiple platforms can be considered virtual toolkits of the second type. However, these use the same look-and-feel on all platforms and therefore do not look the same as the other applications on that platform.

The AWT (Abstract Windowing Toolkit) that comes with the Java programming language also can be classified as a Virtual Toolkit, since the programmer can write code once independently of the targeted platform. Java programs can be run locally in a conventional fashion, or can be downloaded dynamically over the World Wide Web into a browser such as Netscape.

2.3 Higher Level Tools

Since programming at the toolkit level is quite difficult, there is a significant interest in higher level tools that will make the user interface software production process easier.

High-level user interface tools come in a large variety of forms. One important way that they can be classified is by how the designer specifies the interface

design. Some tools require the programmer to program in a special-purpose language, some provide an application framework to guide the programming, some automatically generate the interface from a high-level model or specification, and others allow the interface to be designed interactively.

In the last category, which is also the easier for the designer to use, the tools allow the user interface to be defined by placing objects on the screen using a pointing device. This is motivated by the observation that the visual presentation of the user interface is of primary importance, and a graphical tool seems to be the most appropriate way to specify the graphical appearance. In the tools that support graphical specification, prototyping tools and interface builders are also included. The goal of *prototyping tools* is to allow the designer to quickly mock up some examples of what the screens in the program will look like. Often, these tools cannot be used to create the real user interface of the program; they just show how some aspects will look. An *interface builder* allows the designer to create dialog boxes, menus and windows that are to be part of a larger user interface. These are called *Interface Development Tools (IDTs).* Interface builders allow the designer to select from a pre-defined library of widgets, and place them on the screen using the mouse. They use the actual widgets from a toolkit, so they can be used to build parts of real applications. Most will generate C code templates that can be compiled along with the application code.

2.4 Interactive GUI Development Tools

Almost fifteen years ago, professional software engineers were the only people capable of developing graphical applications. The toolkits available at that time - the Macintosh toolbox, Digital Research's GEM, Microsoft Windows 1.0, or the VAX-based X window system - were primitive and required mastery of a very complex GUI API. Worse still, applications had to be written in C, a language infamous for being difficult to learn and very unforgiving of programmer mistakes.

A few years later, some bright engineers at Apple Corporation decided that what "the computer for the rest of us" needed was a GUI development environment. The result of this project was the HyperCard, the first full-featured development environment designed for non-programmers (first released in 1989). Since then, many other end user programming tools have been developed including such HyperCard-like tools as SuperCard and ToolBook, and independently involved tools such as Visual Basic (VB) and Visual Basic for Applications (VBA).

Such tools have been developed finally for the Unix/X11 market too. Products like Tcl/Tk, the COSE Desktop KornShell (dtksh) and MetaCard make it possible for non-programmers to develop GUI applications without putting the massive effort required to master C/C++ and the Xt/Motif toolkit.

Fortunately end users are not the only ones to benefit from the development of these tools. By offering a higher-level language for GUI development, these tools are much more productive development tools for almost all developers. In fact these tools will eventually replace conventional GUI development environments for nearly all application development.

Just as C compilers replaced assemblers as the standard development tools in the early 80's, and languages like Perl and the shell languages have replaced C programming for most system administration tasks, it would not be long before developing an application in C or C++ with a low-level GUI API like Xt/Motif will be about as popular as programming in assembly language is today. This transition is already largely complete on the Microsoft Windows platform where the vast majority of applications are developed with tools like VB, PowerBuilder, Borland's Delphi and a wide variety of fourth generation languages 4GL's and other high-level languages.

The primary advantage of a VHLL (Very High Level Language) also commonly called scripting language, over a third generation language like C or C++ is that fewer lines of code must be written to complete a given task. Fewer statements take less time to write, and can be understood and modified more easily. The bottom line is increased developer productivity, better quality, and higher functionality applications.

A second benefit arises from the fact that VHLLs are usually interpreted, which means you do not have to wait for compile-link-run cycles. This also greatly improves developer productivity since more time is spent developing and less time waiting. While one might guess that choosing a high-level language might require sacrificing performance, benchmarking shows that in many cases screen update performance of applications developed with these tools is even faster than comparable applications developed in C with Xt/Motif due to the greater overhead of that toolkit. Furthermore, most VHLLs can be extended by adding commands written in C to improve performance where needed or to take advantage of libraries that are only available for C or C++. Certainly performance concerns aren't a reason to choose a third-generation environment over one based on a VHLL.

Some certain features of a scripting language include:

- interpreted execution
- simple syntax
- untyped variables
- no pointers or memory allocation

Many of the above features are also common to the fourth-generation languages used with many database front-end toolkits. The various shell languages and UNIX pattern-matching languages like awk and Perl also have many of

these characteristics. Even some older languages like Lisp could be classified as scripting languages based on these criteria, though perhaps the simple syntax requirement cannot be met. New-generation 3GLs like Java have some of these characteristics, but the complex syntax, requirement to compile as a separate step, and the need to type all variables means that Java should not be grouped with true scripting languages.

While it might be tempting to equate these interactive GUI tools with C and Xt/Motif based User Interface Management Systems (UIMSs), there are several important distinctions that should be made. First, the interpreted languages used in these systems are designed to supplement the third generation (3GL) code used to build the bulk of the application code, not to replace it. These tools are also not complete environments since they must use compilers and other tools to build applications. This increases development costs and results in applications that are not as easily portable as those built with the interactive GUI tools. Finally, these tools are targeted at professional software developers, so they are much more expensive and usually much more difficult to learn and use.

2.5 Future Trends in User Interfaces

The rapid development of 3-D hardware and software has brought revolutionary changes in 3-D technology. Expensive supercomputers and state-of-the-art workstations, which were absolutety required in the early days of 3-D computing, have been substituted by low-cost 3-D workstations and powerful personal computers. Tremendous progress in stereoscopic output devices initiated the transfer of Virtual Reality from research laboratories to industry, arcades, and the home. All these have significantly expanded the user base of 3-D computing from limited groups of technological "outfits" – elite groups of scientists and engineers – to an enormous audience of researchers and practitioners in such areas as CAD/CAM, medicine, engineering, and others.

Software vendors have responded to the new opportunities in this market by developing interactive 3-D applications oriented towards end-users. However, this new orientation towards a broad and rather heterogeneous user base imposes extremely high requirements on the user interface. The user should be able to start using applications with a minimal background in 3-D computing, without qualified assistance, optimally even without reading the manuals. Therefore, the usability of 3-D applications and the development of effective 3-D interfaces has become one of the most important directions of research and development in 3-D user interfaces.

It appears, the present development of 3-D computer graphics software is suffering from the lack of well accepted standards. Most of the developments in this area follow artistic guidelines.

Another serious problem is the lack of software tools and programming environments for developing 3-D user interfaces across platoforms. Most of the

present 3-D applications were developed for particular GUI environments, such as X Windows and Microsoft Windows. The reason being the availability of powerful interface developing tools for X Windows. The usage of such tools increases the performance of developers a great deal and provides a guaranteed quality of the interface. However, these toolkits do not provide any specialized 3-D widgets for interaction techniques specific for 3-D interfaces, such as selecting in 3-D, moving in 3-D, etc. Each developer implements his or her own techniques, which violates one of the most attractive advantages of windowing environments - consistency between interfaces of different applications developed for the same windowing system. Generally speaking, the applicability of these tools, and more generally the whole WIMP paradigm (Windows, Icons, Menu and Pointers), for 3-D interfaces is still an open research problem.

3 Parallel ELLPACK Graphical User Interface

The //ELLPACK GUI consists of a main session editor that allows the user to specify the PDE problem, the PDE solver and its parameters, and the output specifications at very high level language or utilizing a number of graphical interactive tools. The tools that are made available to the user within a session are dependent on the dimensionality of the PDE problem to be solved (i.e., 1-D, 2-D, 3-D) and the discretization methodology applied (i.e., finite difference/finite element). Different tools support a different part of the problem specification or the solution framework specification.

While the graphical tools are active, the current //ELLPACK program is internally represented by a set of parsed data structures. In addition, it is represented as text within the session editor as the user's session log. Each tool manipulates one or more pieces of this data structure and is responsible for leaving them in a consistent state. In some cases, a tool is actually a separate process. Then, the appropriate data structures are communicated to the other process via inter-process communication and made consistent when the changes are "committed". The tools also have a dependency relationship; for example, the mesh tool cannot be invoked until a domain has been specified. This is supported by having the tools themselves be aware of their position in the chain of operation and having them do the appropriate tests to ensure that the proper order is maintained.

As the PDE problem and solution framework are being defined, the session editor reflects the current status of specification by displaying the interactivity log in the //ELLPACK language. The user may choose to edit the //ELLPACK program directly as well, but in order to maintain consistency, the user must not be running any of the graphical tools at the same time. For solution and performance visualization and analysis, the user specifies where to save the appropriate data at problem specification time. Then, the visualization environment loads this data at post-processing time to visualize the results.

In this section we present the implementation of the 3-D //ELLPACK user interface and the functional/software specification of the //ELLPACK tools.

3.1 Interface Software Infrastructure Utilized in //ELLPACK

The OpenGL graphics system is a powerful software interface for graphics hardware that allows graphics programmers to produce high-quality color images of 2-D and 3-D objects. The technology was developed by Silicon Graphics Inc. (SGI) and is the result of ten years of experience designing production software interfaces for a full spectrum of graphics hardware.

OpenGL is now controlled by an industry consortium known as the OpenGL Architectural Review Board (ARB) currently composed of Digital Equipment, IBM, Intel, Microsoft, and SGI. The interface is licensed to a large number of computer software and hardware vendors and OpenGL implementations are now appearing on the market.

OpenGL provides a layer of abstraction between graphics hardware and an application program. It is visible to the programmer as a set of routines consisting of about 120 distinct commands. Together these routines make up the OpenGL application programming interface (API). The routines allow graphics primitives (points, lines, polygons, bitmaps, and images) to be rendered to a frame buffer. Using the available primitives and the operations that control their rendering, high-quality color graphics images of 3-D objects can be rendered.

The model used for integration of OpenGL commands is *client-server*. This is an abstract model and does not demand OpenGL be implemented as distinct client and server processes. A client-server approach means the boundary between a program and the OpenGL implementation is well-defined to clearly specify how data is passed between the program and the OpenGL. This allows OpenGL to operate over a *wire protocol* much as the X protocol operates but does not mandate such an approach.

The OpenGL specification is *window system independent* meaning it provides rendering functionality but does not specify how to manipulate windows or receive events from the window system. This allows the OpenGL interface to be implemented for distinct window systems. For example, OpenGL has been implemented for both the X Window System and Windows NT.

The specification which describes how OpenGL integrates with the X Window System is known as GLX [Kil93, Kil94a]. It is an extension to the core X protocol for communicating OpenGL commands to the X server. It also supports window system specific operations such as creating rendering context, binding those contexts to windows, and other window system specific operations.

GLX does not demand OpenGL commands be executed by the X server. The GLX specification explicitly allows OpenGL to render directly to the hardware if supported by the implementation. This is possible when the program is running on the same machine as the graphics hardware. This potentially allows

extremely high performance rendering because OpenGL commands do not need to be sent through the X server to get to the graphics software.

Graphics systems are often classified as one of two types: *procedural or descriptive*. Procedural means the programmer is determining what to draw by issuing a specific sequence of commands. Descriptive means the programmer sets up a model of the scene to be rendered and leaves how to draw the scene up to the graphics system. OpenGL is procedural. In a descriptive system, the programmer gives up control of exactly how the scene is to rendered. Being procedural allows the programmer a high degree of control to achieve the best performance. It is expected that descriptive graphics systems will be implemented using OpenGL as a low level interface. SGI's Inventor toolkit is one example of such a descriptive graphics system.

OSF/Motif is the X Window System's industry-standard programming interface for user interface construction. Motif programmers writing 3-D applications have to understand how to integrate Motif with the OpenGL graphics system. Most 3-D applications end up using 3-D graphics primarily in one or more "viewing" windows. For the most part, the graphical user interface aspects of such programs use standard 2-D user interface objects like pulldown menus, sliders, and dialog boxes. Creating and managing such common user interface objects is what Motif does well. The "viewing" windows used for 3-D are where OpenGL rendering happens. These windows for OpenGL rendering can be constructed with standard Motif drawing area widgets or OpenGL-specific drawing area widgets. After binding an OpenGL rendering context to the window of a drawing area widget the environment is ready for 3-D rendering [Kil94b].

This is the approach which has been adopted in the case of the development of the 3-D mesh tools in the latest version of //ELLPACK. In particular, OpenGL-specific drawing area widgets have been used for 3-D rendering.

Programming OpenGL with Motif has numerous advantages over using "Xlib only". First and most important, Motif provides a well-documented, standard widget set that gives in the application a consistent look and feel. Second, Motif and the X Toolkit take care of routine but complicated issues such as *cut and paste* and window manager conventions. Third, the X Toolkit's work procedure and time-out mechanisms make it easy to animate a 3-D window without blocking out user interaction with the application.

Furthermore, a Tk OpenGL widget called *Togl* [PB96] has been developed for OpenGL rendering. Togl allows one to create and manage a special Tk/OpenGL widget with Tcl and render into it with a C program. That is, a typical Togl program will have Tcl code for managing the user interface and a C program for computations and OpenGL rendering.

Mesa [Pau96a] is a free 3-D graphics library which uses the OpenGL API and semantics. It works on most modern computers allowing people without OpenGL to write and use OpenGL-style applications.

Mesa began as an experiment in writing a 3-D graphics library. After about a year of "spare time" development it was released on the Internet. It has since

evolved with the help of many contributors to the point where it is a viable and popular alternative to OpenGL.

While Mesa uses the OpenGL API and follows the OpenGL specification very closely, it is important to note that Mesa is not true implementation of OpenGL. Official OpenGL products are licensed and must completely implement the OpenGL specification and pass a suite of conformance tests. Mesa meets none of these requirements.

At first, Mesa may seem to be a competitor to official OpenGL products. Actually, Mesa has helped to promote the OpenGL API by expanding the range of computers which may execute OpenGL programs. There are many systems which are not supported by OpenGL vendors but can run Mesa instead. People who are curious about OpenGL may try Mesa at no cost and later purchase a commercial OpenGL implementation which utilizes graphics hardware. Students may learn 3-D programming using Mesa and later develop OpenGL applications on the job.

The Mesa distribution includes implementations of the core OpenGL library functions, the aux and tk toolkits, Xt/Motif widgets, drivers for X11, Microsoft Windows '95/NT, NeXTStep, AmigaDOS, and many demonstration programs. A Macintosh driver is distributed separately. Mesa compiles easily, requiring only an ANSI C compiler and the development resources (header files and libraries) for the target platform.

Mesa does not implement the full OpenGL specification. Also does not typically perform as well as commercial OpenGL implementations for several reasons. First, portability to a wide range of computers is considered more important than optimizing for a particular architecture. Second, the features of the underlying hardware can't be directly accessed since Mesa exists as a software library above the operating system and window system programming interfaces. And finally, Mesa's development is not supported by any sort of development team. Only so much can be accomplished by people working in their spare time.

Some of the applications that already use the Mesa library [Pau96b] are:

- General Mesh Viewer (GMV), an easy to use 3-D scientific visualization tool designed to view simulation data from any type of structured or unstructured mesh.
- gOpenMol, is the graphical interface into the OpenMol set of computational chemistry programs. OpenMol is an integrated program for electronic structure and property calculations of molecules.
- Hydra, a distributed Multi-User Computer Aided Physics and Engineering Package that can be used to setup physics simulations of any size with any of the simulation codes developed in X Division.
- LinkWinds, is a visual data exploration system resulting from a program of research into the application of computer graphics to rapidly and inter-

actively accessing, displaying, exploring and analyzing large multivariate multidisciplinary data sets.

- Vis5D is a system for interactive visualization of large 5D gridded data such as those made by numeric weather models. One can make isosurfaces, contour line slices, colored slices, volume renderings, etc. of data in 3-D grid then rotate and animate the image in real time.

- The Visualization ToolKit (vtk) which is a software system for 3-D Computer Graphics and Visualization.

3.2 //ELLPACK Graphical User Interface

//ELLPACK [HRW+96, Wee94] allows the solution of single linear and nonlinear elliptic and parabolic PDE equations defined on 1-D, 2-D, and 3-D dimensional domains. In this PSE, the user can specify a solution method by naming selected library modules corresponding to the phases of the PDE solution process.

The process of specifying, solving, and analyzing a PDE problem occurs within a //ELLPACK session. In the //ELLPACK PSE, a PDE problem is defined by the components (or objects) that specify:

- the PDE equations,

- the domain of definitions,

- the boundary and initial conditions,

- the solution strategy,

- the output requirements.

Each session consists of a text window and an attached toolbox of editors. The editors are used to create or modify the PDE objects which specify the PDE problem and describe how to solve it. These include equation editors, geometry editors, mesh generators, and algorithm selection menus. Each editor is a graphical, interactive tool which creates or edits a specific PDE object, and then writes a textual representation of the object to the text window using the //ELLPACK language.

In this context, we shall concentrate on the tools used by the //ELLPACK for visualization purposes and more specifically in 3 dimensions. Namely, we shall discuss the 3-D Mesh Editor, the 3-D Boundary Condition Editor, the 3-D Decomposition Editor and the Visual3D tool which is used to visualize 2-D and 3-D meshes, 2-D and 3-D structured (non-orthogonal) grids, 2-D and 3-D poly files, and 2-D and 3-D solution files.

Figure 1: //ELLPACK System Icon

3.2.1 3-D Mesh Editor

The 3-D Mesh Editor is available in the //ELLPACK (Figure 1) from the toolbox of the 3-D Finite Element Session Figure 2.

To access the 3-D Mesh Editor, the icon in Figure 3 in the session toolbox must be clicked. The editor is used to display and edit 3-D meshes which have been saved in the //ELLPACK format or in the neutral format.

3.2.2 3-D Mesh Editor Areas

The 3-D Mesh Editor consists of three windows: a Command Panel, a Display Window, and a Palette. The Command Panel contains buttons and scrollbars for loading, saving, and manipulating the mesh. The Display Window is used as a canvas for viewing the mesh, and the Palette displays the colors associated with the surface patches which are currently defined for the mesh.

Surface patches are groups of nodes on the surface where boundary conditions may be defined. These nodes will be colored according to their surface patch grouping, and there are as many colors as there are surface patches. The grouping of surface nodes into patches has already been specified in the mesh file, but this specification may be changed within the 3-D Mesh Editor.

3.2.3 3-D Mesh Palette Window

The Palette Window in Figure 4 is used to identify the surface patches of the 3-D mesh. All surface nodes are assigned to a specific patch where boundary conditions can later be assigned. The patch identifications in the mesh file are shown on the canvas via color assignments. The number and color of the patches are also shown in the Palette. The number of patches can be changed through the Command Panel.

3.2.4 3-D Mesh Display Window

The Display Window is the canvas (Figure 5) for viewing the mesh. Meshes that are loaded in are displayed in the canvas, and the effect of all transformations

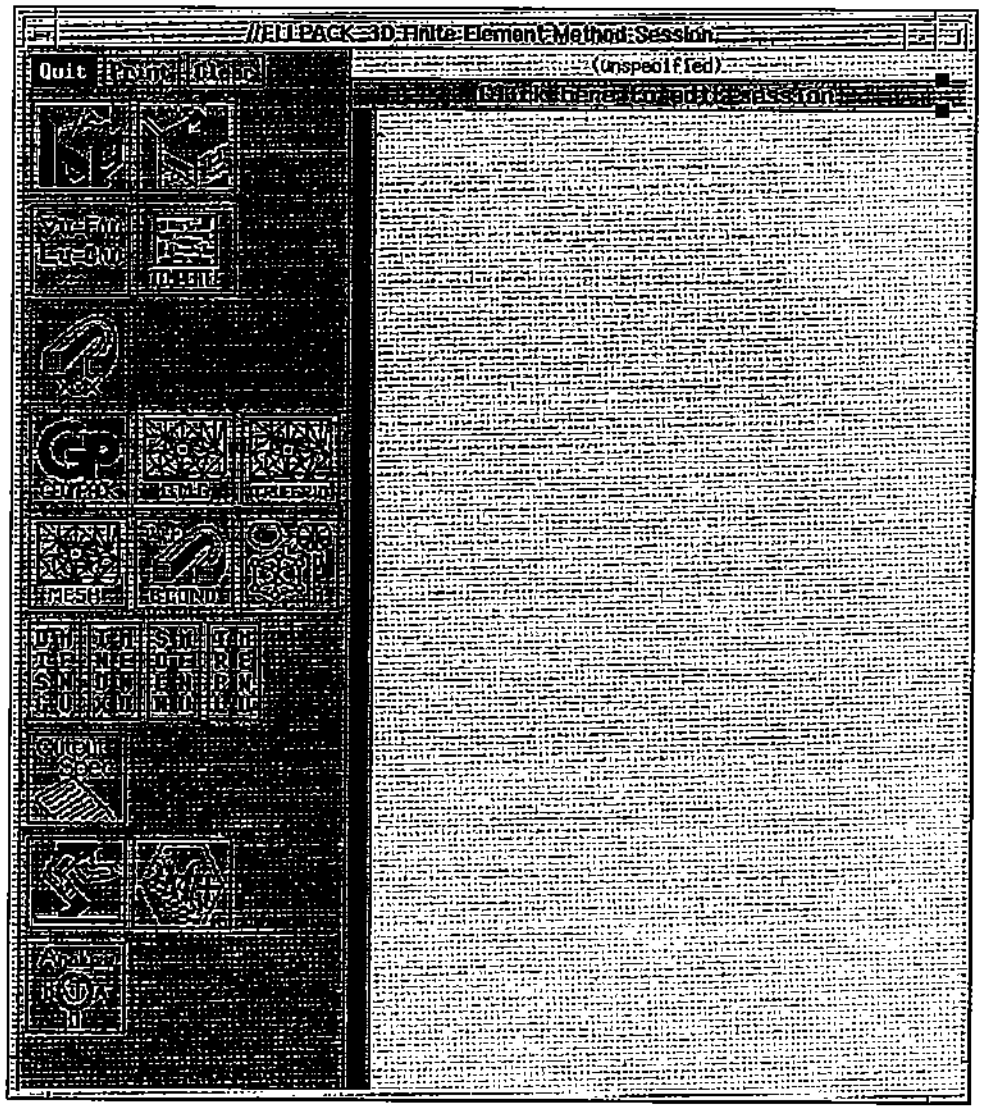

Figure 2: 3-D FEM Session Toolbox

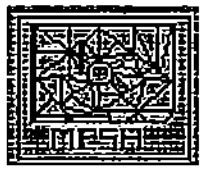

Figure 3: 3-D Mesh Tool Icon

Figure 4: 3-D Mesh Palette Window

specified through the Command Panel can be viewed here.

3.2.5 3-D Mesh Command Panel

The Command Panel (Figure 6) consists of two sections: the top level buttons and the transformation scrollbars. The top level button panel contains buttons to load, edit, save, and display the 3-D mesh. The scrollbars control the view of the mesh in the Display Window. The Command Panel Buttons are the following:

Quit Exits the 3-D Mesh Editor without saving

Load Loads a mesh data file. Selecting Load causes the File selection dialog (Figure 7) to appear so that users may enter a file type and filename for loading. Two file formats are supported by the 3-D Mesh Editor: //ELLPACK's internal format and the neutral file format. Conversions between these two types are also supported by the 3-D Mesh Editor, since the selection of a discretization module for the solution scheme may require that the mesh file has a specific format.

Save Saves the generated mesh to the session using the //ELLPACK language. A File dialog appears so that users may specify a filename for the saved mesh. This dialog operates exactly as the Load File dialog. Mesh files that have been loaded as //ELLPACK format files may be saved in either format; neutral format files loaded into this editor may also be saved in either format.

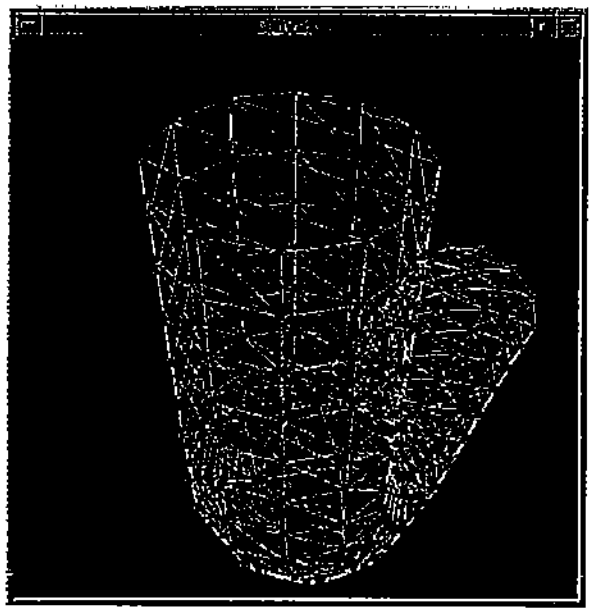

Figure 5: 3-D Mesh Canvas Window

Reset Resets to the original mesh.

Redisplay Redisplays the generated mesh.

Axis Displays the coordinate axes in the Display Window in the correct orientation.

Edges

- *All edges*: displays the entire mesh, including the edges of all elements, both interior and on the surface.
- *Boundary edges*: shows edges of only the boundary edges of the elements on the surface of the mesh.

Vertices Displays only the vertices of the mesh.

Node/Patch Allows the user to redefine the patch definition of the surface node. Selecting Node allows users to assign a patch (color) to that specific node; selecting Patch allows users to assign a new patch (color) to an entire patch.

Undo Undoes the previous operation.

Apply Applies the selected patch operation, specified by node/patch assignments.

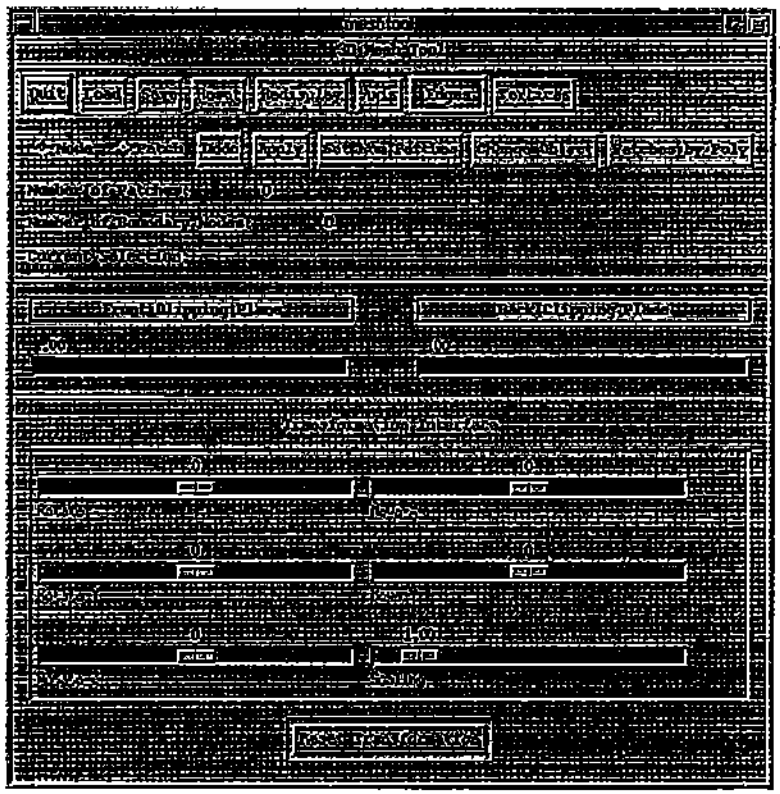

Figure 6: 3-D Mesh Command Panel Window

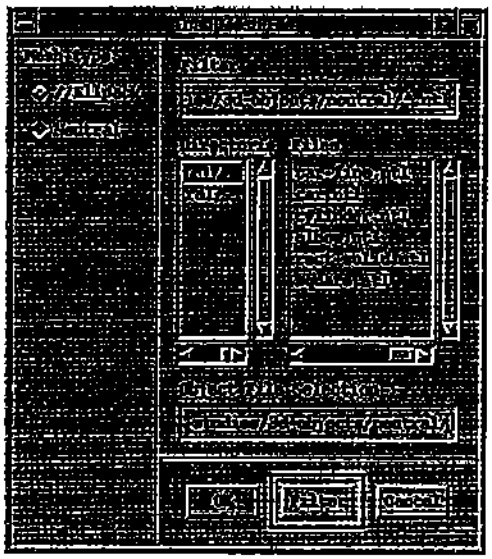

Figure 7: 3-D Mesh File Selection Dialog Window

Set Num Patches Sets the number of patches. When this button is selected, the user is requested to enter the number of desired patches. The current number of patches is listed below the button panel. Increasing the number of patches allows users to define more surface patches by assigning nodes to the new patches. This allows users to assign more boundary conditions on the surface patches of the mesh.

Change Object Changes the object into a mesh without showing points on the surface.

Patches by Poly Allows users to load the poly format file that was used to generate the mesh (if the mesh was generated from a poly definition of the 3-D object). This poly file is then used to define the surface node patches of the mesh for the future assignment of boundary conditions. This assists users in grouping the surface nodes by setting up an initial configuration of patches.

Information about the mesh is contained in the Command Panel, and is updated when new meshes are loaded, new patches are selected, or the total number of patches is changed. The Command Panel provides the following information:

Number of Patches Lists the current number of patches defined for this mesh.

Number of Boundary Nodes Lists the total number of surface nodes for this mesh.

Current Selection shows the current patch (color) selection. The currently selected patch affects the assignment of nodes to patches.

The transformation scrollbars allow users to enlarge or shrink, rotate or translate the mesh object in the canvas area as is depicted in Figure 8.

New values in the scrollbar are set by holding the left mouse button down on the scrollbar indicator and moving it to the left or to the right.

Front Clipping Plane, Back Clipping Plane Changes the location of the clipping plane in the foreground/background of the mesh displayed in the canvas.

Rotate X, Rotate Y, Rotate Z Rotates the object in the X/Y/Z direction by the amount specified in the scroll bar. The initial rotation value is zero. Rotational amounts are in degrees.

Move X, Move Y Translates the object in the X/Y direction by the amount specified in the scroll bar. The initial translation value is zero.

Scaling Resizes the object in the canvas. The initial scale value is 1. Values greater than one enlarge the object; values less than one shrink the object.

Reset Transformation Reinitializes all transformation parameters to their original values and redisplays the object in the canvas.

3.2.6 3-D Boundary Condition Editor

The 3-D Boundary Condition Editor is available also from the toolbox of the 3-D Finite Element Session. To access the 3-D Boundary Condition Editor, the icon in Figure 9 in the Session Toolbox must be clicked. This editor is used to display 3-D meshes which has been saved in the //ELLPACK format or in the neutral format and to assign boundary conditions to the surface nodes which have been grouped together into surface patches. Modification of the surface patches is handled by the 3-D Mesh Editor. When the mesh is loaded into the Boundary Condition Editor, the surface patches are fixed, and only the boundary conditions may be set or modified.

The 3-D Boundary Condition Editor consists of three windows: a Command Panel, a Display Window, and a Palette. The Command Panel contains buttons and scrollbars for loading, saving and manipulating the mesh and its associated boundary conditions. The Display and the Palette window has the same functionality as in the 3-D Mesh Editor.

If a mesh file exists in the current //ELLPACK session, this mesh will automatically be displayed. Otherwise, the editor will display the file selection dialogue (Figure 7) so that users may select a mesh file to load. //ELLPACK and neutral file formats are supported by the Boundary Condition Editor.

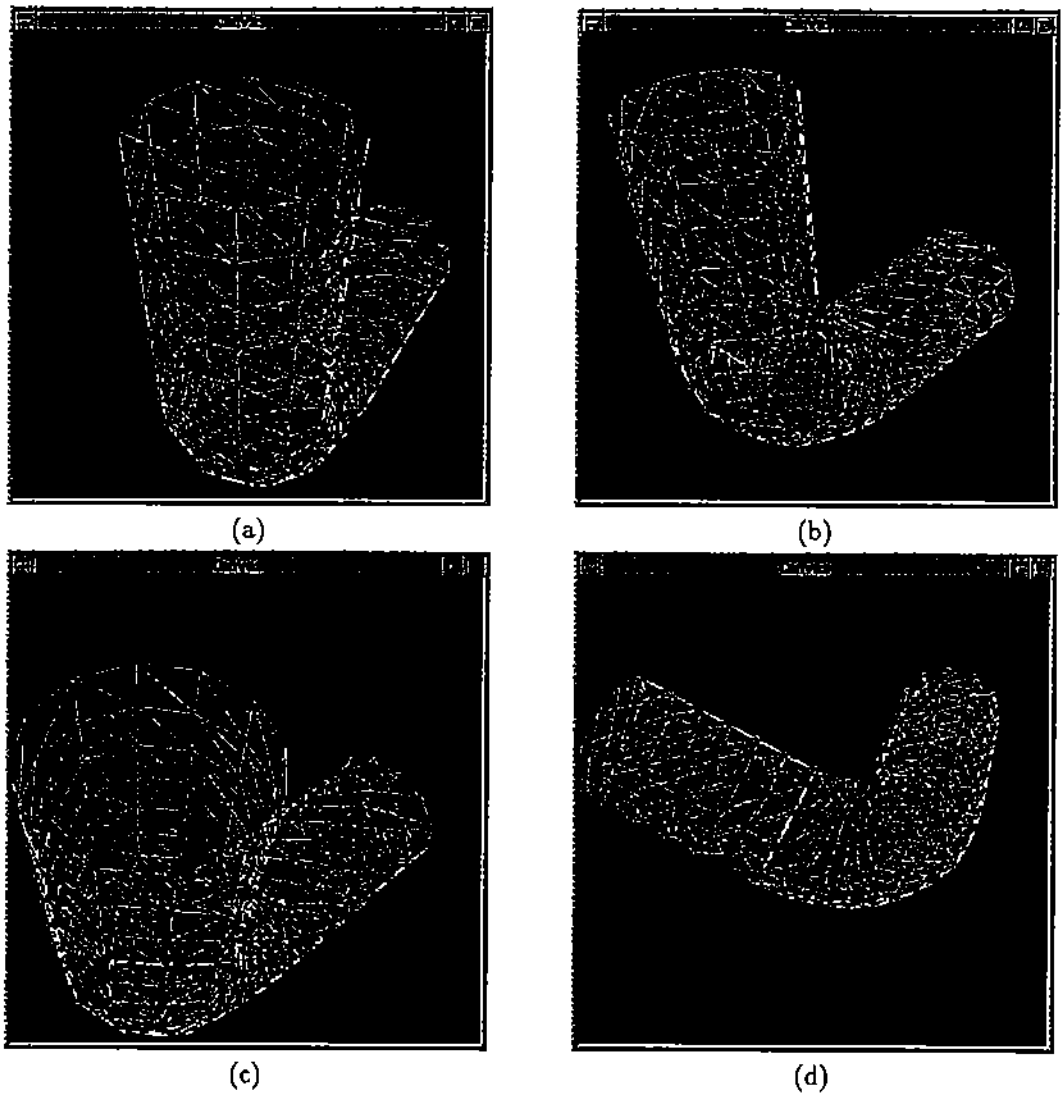

Figure 8: (a), (b), (c), (d): Various Transformations of the Original Object

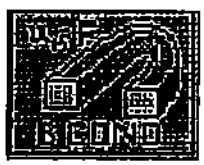

Figure 9: 3-D Boundary Condition Tool Icon

3.2.7 3-D Boundary Condition Display Window

The Display Window is the canvas (Figure 5) for viewing the mesh. Meshes that are loaded in are displayed in the canvas, and the effect of all transformations specified through the Command Panel can be viewed here.

3.2.8 3-D Boundary Condition Palette Window

The Palette Window in Figure 4 is used to identify the surface patches of the 3-D mesh. All surface nodes are assigned to a specific patch where boundary conditions can be assigned. The patch identifications in the mesh file are shown on the canvas via color assignments. The number and color of the patches are also shown in the Palette. The number of patches cannot be changed with the Boundary Condition Editor.

3.2.9 3-D Boundary Condition Command Panel

The Control Panel (Figure 10) consists of two sections: the top level buttons and the transformations scrollbars. The top level button panel contains buttons to load, edit, save, and display the 3-D mesh. Activate a button by clicking on it with the left mouse button. Boundary conditions can be set or modified. The scrollbars control the view of the mesh in the Display Window.

Quit Exits the 3-D Boundary Condition Editor.

Load Load a mesh file. This button causes the file selection dialogue to appear which allows the users to specify the path, the filetype and the filename of the file to be loaded.

Save Saves the boundary conditions to the session using the //ELLPACK language.

Reset Resets to the original mesh.

Redisplay Redisplays the generated mesh.

Axis Displays the coordinates axes in the Display Window in the appropriate orientation, according to the objects position in the display window.

Edges

- *All edges*: displays the entire mesh, including the edges of all elements, both interior and on the surface.
- *Boundary edges*: shows only the boundary edges of the elements on the surface of the mesh.

Vertices Displays only the vertices of the mesh.

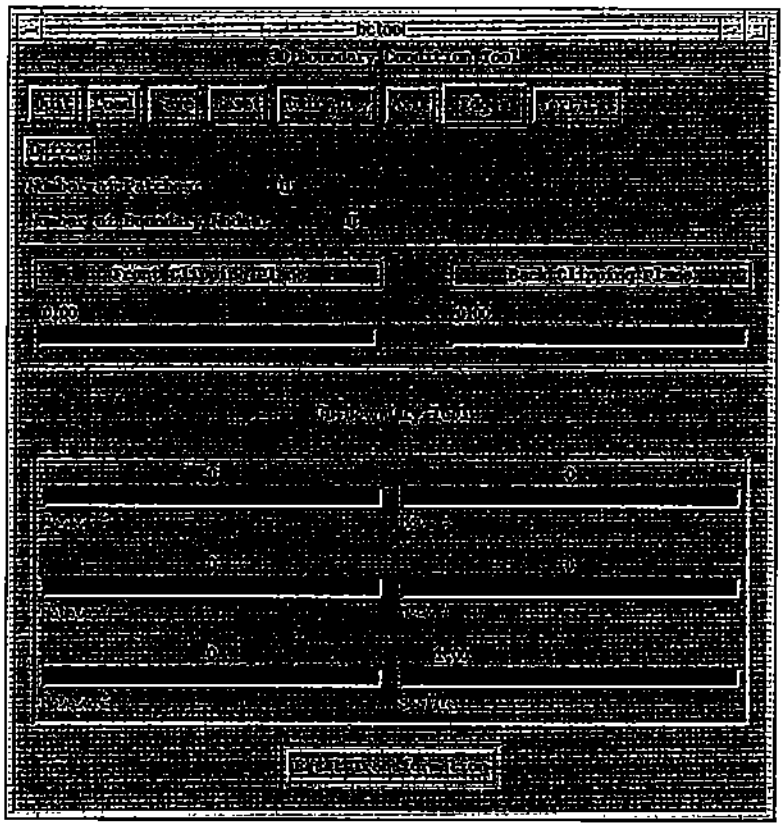

Figure 10: 3-D Boundary Condition Command Panel Window

Define By pressing the *Define* button, the window in Figure 11 will be appeared for defining or editing the boundary conditions for each patch. The patches are numbered, and the color code in the palette defines the mapping between the colors of the surface nodes and the patch number of the associated boundary condition.

The rest of the information contained in the Command Panel has already been described previously, in the corresponding area of the 3-D Mesh Editor.

3.2.10 3-D Decomposition Editor

The 3-D Decomposition Editor (FEM session only) is used to define a decomposition on the currently defined discrete 3-D domain. The 3-D Decomposition Editor loads in a mesh file and produces a file containing a description of the domain decomposition, and generates a //ELLPACK language description of

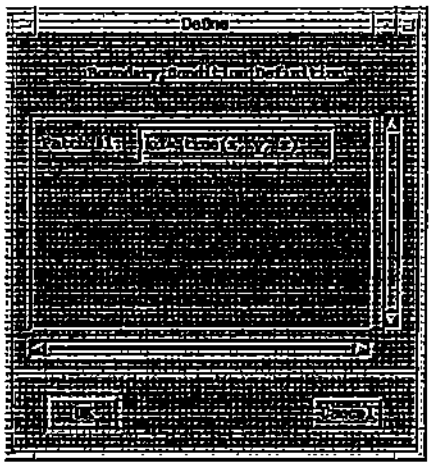

Figure 11: 3-D Boundary Condition Definition Window

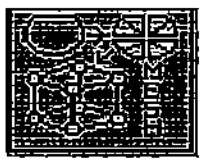

Figure 12: 3-D Decomposition Tool Icon

the decomposition that is saved in the //ELLPACK session. The 3-D Decomposition Editor can also be used to read in the decomposed file and display it.

The 3-D Decomposition Editor is available from the toolbox of the 3-D Finite Element Session in the //ELLPACK (Figure 12).

It consists of three windows: a Display Window, a Palette and a Command Panel.

3.2.11 3-D Decomposition Display Window

The Display Window is the canvas (Figure 13) for viewing the mesh or decomposition. Meshes and decompositions that are loaded are displayed in the canvas, and the effect of all transformations specified through the Command Panel can be viewed here.

3.2.12 3-D Decomposition Palette Window

The Palette Window (Figure 14) is used to identify the subdomains for the 3-D mesh. Each node is assigned to a specific subdomain and the correspondence is

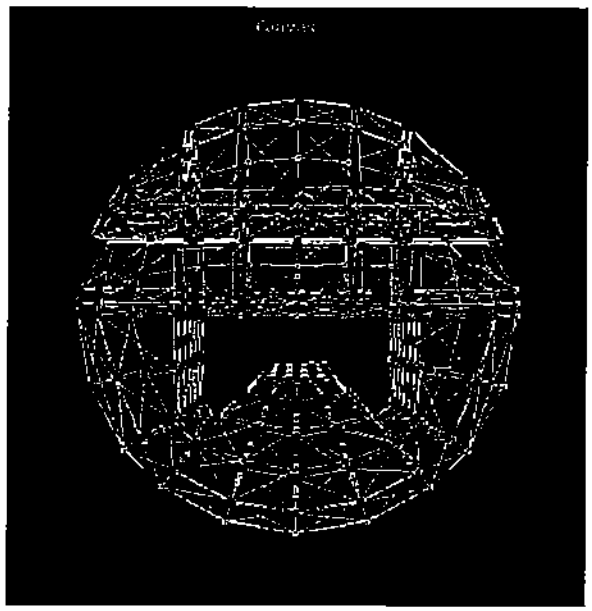

Figure 13: 3-D Decomposition Canvas Window

shown on the canvas via color assignments. The number of subdomains can be changed through the Command Panel.

3.2.13 3-D Decomposition Command Panel

The Command Panel (Figure 15) consists of two parts: the top level buttons and the transformation scrollbars. The top level button panel contains buttons to load, edit, save, and display the 3-D mesh or decomposition. The scrollbars control the view of the mesh and its decomposition in the Display Window. The Command Panel Buttons are the following:

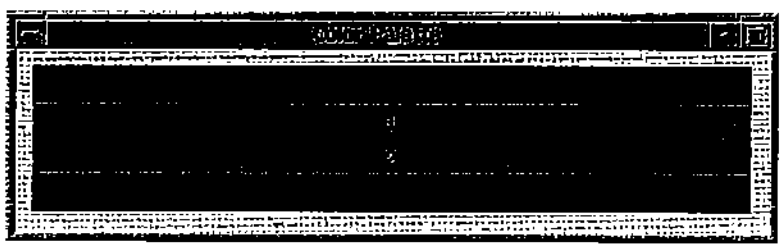

Figure 14: 3-D Decomposition Palette Window

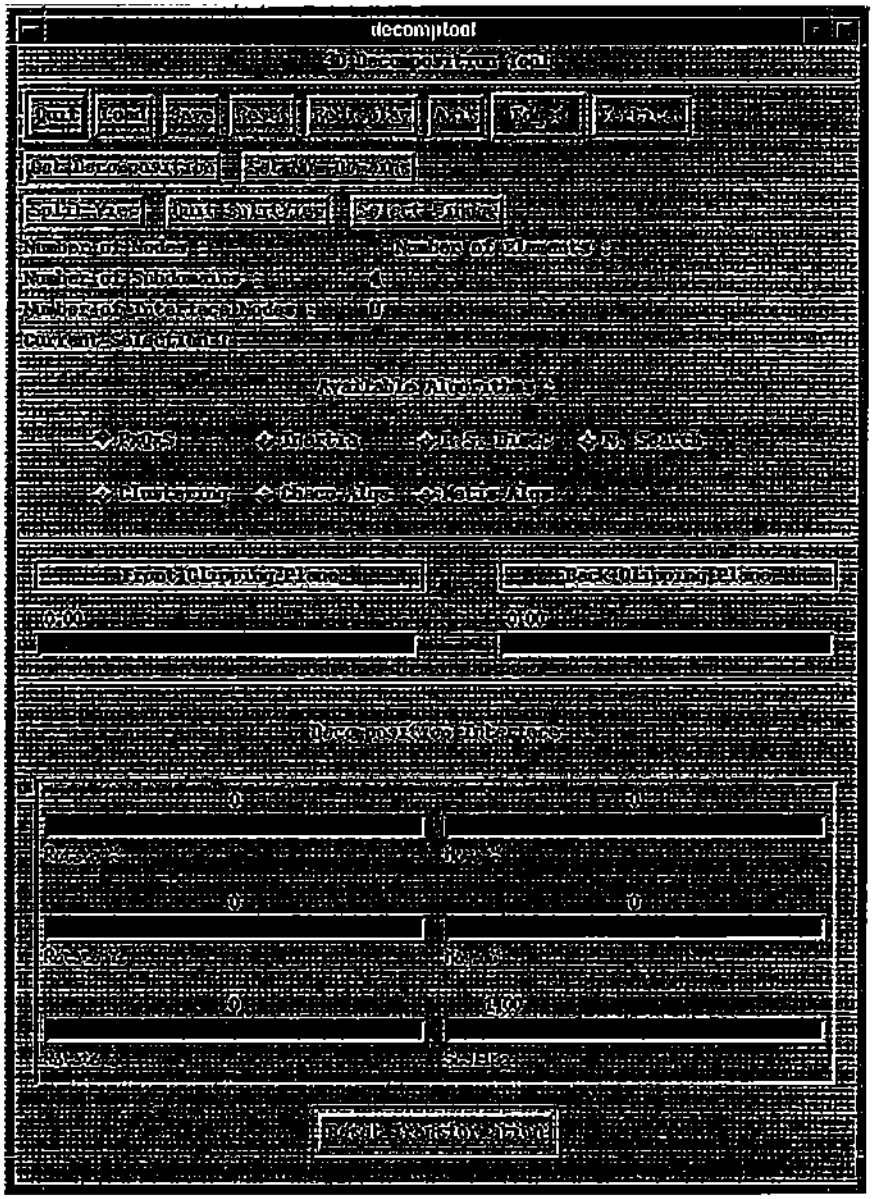

Figure 15: 3-D Decomposition Command Panel Window

Quit Exits the 3-D Decomposition Editor. The current decomposition will not be saved to the session unless the Save button is clicked.

Load Loads a mesh or decomposition data file. Selecting Load causes the File Dialog (Figure 16) to appear so that users may enter a file type and filename for loading. Users may either load a mesh file for decomposing it, or load a decomposition file for viewing or modifying.

Save Saves the generated decomposition to the session using the //ELLPACK language. A File dialog appears so that users may specify a filename for the saved decomposition. This dialog operates exactly as the Load File Dialog.

Reset Resets to the original decomposition.

Redisplay Redisplays the generated decomposition on the current mesh.

Axis Displays the coordinate axes in the Display Window in the correct orientation.

Edges

- *All edges*: displays the entire mesh, including the edges of all elements, both interior and on the surface.
- *Boundary edges*: shows only the boundary edges of the elements on the surface of the mesh.

Vertices Displays only the vertices of the mesh.

Get Decomposition Applies the selected decomposition algorithm with its current parameter settings and currently specified number of subdomains to the mesh. The result is shown in the Display Window.

Set Num Domains Brings up a dialog box so that a new number of subdomains can be specified. The next three buttons apply to the subdomain display windows which can be used to view each subdomain separately. These "views" are useful for assessing the results of the decomposition algorithm.

Split View Displays each subdomain in a separate window.

Quit Split View Closes the subdomain display windows.

Select Window Allows the user to select a subdomain display window so that subsequent object transformation input applies to that window.

The particular algorithms offered through //ELLPACK will be described in the next section where further information will be provided for this subject matter.

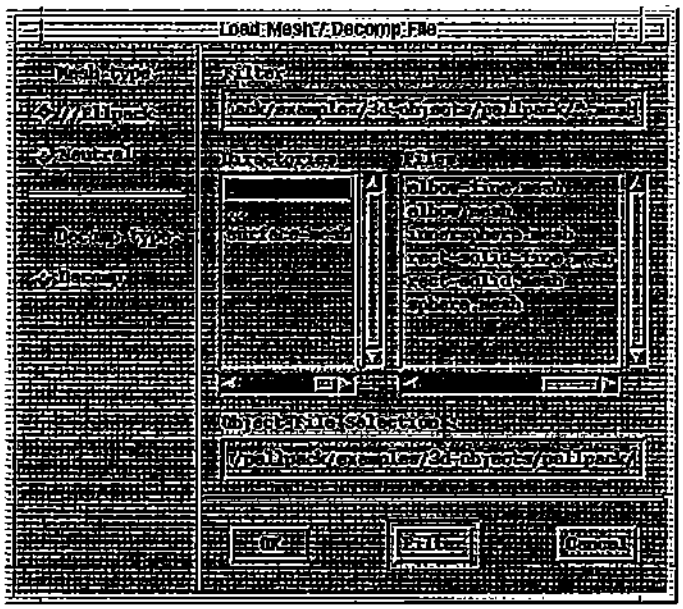

Figure 16: 3-D Decomposition File Selection Dialog Window

3.2.14 Visual3D

As mentioned before, Visual3D is used to visualize 2-D/3-D meshes, structured grids, poly files, and solution data. Data files are loaded directly into Visual3D (Figure 17) from its own menu, so files should not be loaded into the OutputTool before selecting this tool. When Visual3D has been invoked, the window in Figure 17 is displayed.

Use "load mesh" under the File Menu option to load 2-D or 3-D //ELLPACK format mesh files, 2-D/3-D poly files, or neutral format files by using the dialog window depicted in Figure 18.

Use "load solution" under the File Menu option to load //ELLPACK format solution files or component files by using the dialog window depicted in Figure 19. For each of the above File dialogs, the file type must be selected before the file name is specified. In addition, the mesh file must be specified before the solution file can be loaded. When the mesh file is loaded, it is immediately shown in the Display Window. After the solution is loaded, the Rendering menu option should be used for plotting the solution.

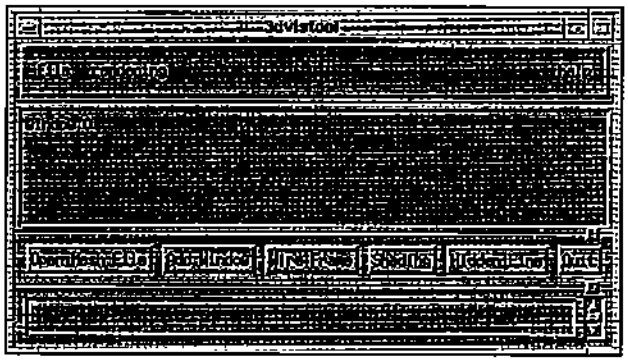

Figure 17: Visual3D Tool

4 //ELLPACK Domain Decomposition Library

Identifying the parallelism in a problem by partitioning its data and tasks among the processors of a parallel computer is a fundamental issue in parallel computing. This problem very often is reduced to a graph partitioning (GP) problem [CHR94] in which the vertices of a graph are divided into a specified number of subsets such that a minimum number of edges join two vertices in different subsets. A partition of the graph into subgraphs leads to a decomposition of the data and/or tasks associated with a computational problem and the subgraphs can then be mapped to the processors of the target multiprocessor.

Graph partitioning has an important role to play in the design of many parallel algorithms by means of the divide and conquer paradigm. In the case of //ELLPACK three parallel paradigms are supported based on the so-called non-overlapping domain decomposition approach.

Following, we survey several classes of graph partitioning algorithms whose various software implementations are included in //ELLPACK for the partitioning of meshes and grids. These mesh/grid partitions are used to decompose the underlying computations and map them to target parallel machines.

4.1 Terminology – Background

A graph G will be denoted by means of its set of vertices V, and the set of edges E. An edge in E is a pair of vertices (u, v); the vertices u and v are the endpoints of the edge. The number of elements in a set S will be indicated by $|S|$. Often, the number of vertices in a graph will be equal to n, and so the equation $|V| = n$ holds.

A *partition* of a connected graph $G = (V, E)$ is a division of its vertices into two sets A and B. The set of edges joining vertices in A to vertices in B is an *edge separator*, that will be denoted by $\delta(A, B)$; the removal of these edges

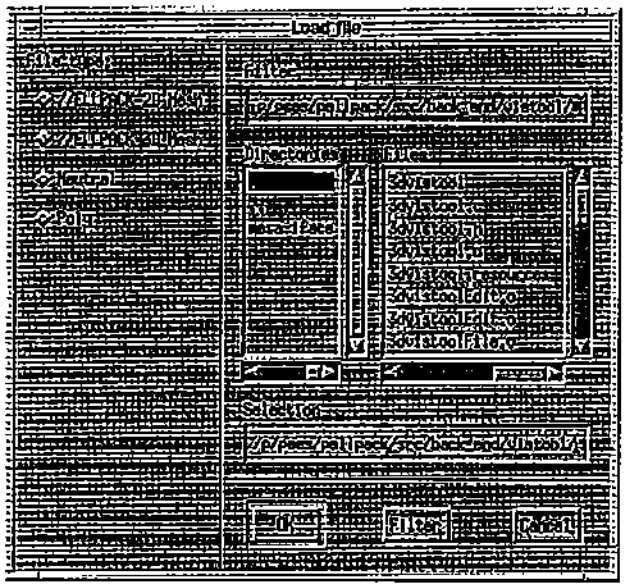

Figure 18: Loading a Mesh/Poly or Neutral file in Visual3D Tool

disconnects the graph into two or more connected components. In applications such as domain decomposition, the set of vertices A would be mapped to one set of processors and the set of vertices B to another, and $|\delta(A, B)|$ is a measure of the amount of communication necessary between the two groups of processors. Hence one goal in partitioning a graph in these applications is to minimize the number of edges cut by the partition so as to keep communication costs in the algorithm minimal.

A second goal is to balance the computational work (load) between the two sets of processors. This is achieved by prescribing the number of vertices in A and B to within a tolerance. If A and B are equal in size, then the partition is called a *bisection* and $|\delta(A, B)|$ is the bisection width.

In other applications, such as nested dissection, a *vertex separator* is desired; this is a set of vertices S whose removal disconnects the graph into two parts with no edge joining a vertex in one part to a vertex in the other. Here the goals are that the separator should have a small number of vertices, and as above, the two parts should not differ by too many vertices.

4.2 A survey of early and recent partitioning algorithms

In this section a brief description of the most important algorithms for partitioning graphs is given. The software implementation of these algorithms is available in //ELLPACK.

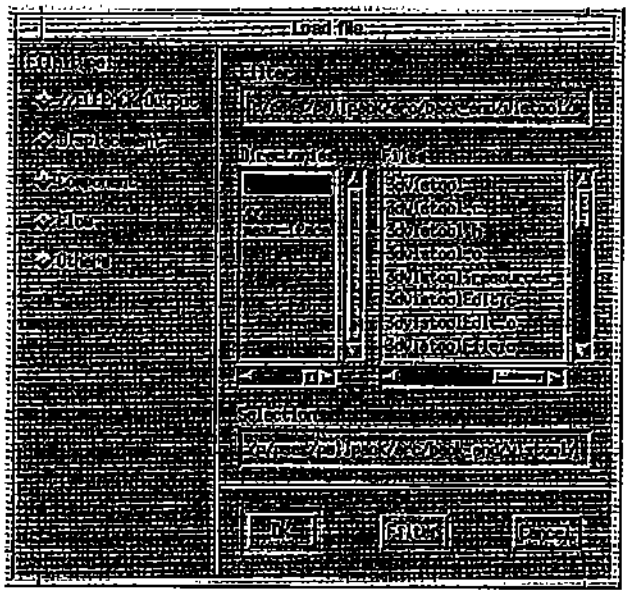

Figure 19: Loading Solution Files in Visual3D Tool

4.2.1 The Kernighan-Lin algorithm

The Kernighan-Lin algorithm is one of the earliest algorithms for partitioning graphs [KL70]. A graph is to be partitioned into k subsets so as to minimize the number of edges joining vertices in different subsets. The algorithm begins with an initial partition into two sets, obtained by a random partition or from some initial information available about the problem. It is an iterative algorithm, in which the basic operation consists of swapping subsets of vertices of equal number between the two sets to reduce the number of edges joining the two sets. The algorithm terminates when it is no longer possible to reduce the number of edges by swapping subsets, or when a specified number of swaps have been made. The algorithm chooses subsets of vertices to swap based on the fundamental concept of the *gain* associated with moving a vertex u belonging to one set A to the other set B. This gain is the net reduction in the number of edges cut by the partition. An important idea in the Kernighan-Lin (KL) algorithm is to continue to move vertices with negative gains for a preset number of steps, in the hope that such a sequence of moves might create vertices with positive gains later on, and thus reduce the number of edges cut overall. This enables the algorithm to climb out of local minima in partition space. Also we can partition a graph into more than two subsets by recursively applying the bipartition algorithm.

The quality of a partition generated by this algorithm depends strongly on

the initial partition. There are poor initial partitions for the regular grid graph that cannot be improved by the KL algorithm. This is the reason, that KL is the most widely used *local refinement* algorithm for GP. That is, KL can be applied in a post-processing phase, to further reduce the number of edges cut, after another "global" algorithm is used to compute a good initial partition.

4.2.2 Level-structure Partitioning

Another early algorithm [CGLN84] for computing vertex separators is the one that finds a pseudo-peripheral vertex v in the graph (one of a pair of vertices that are approximately at the greatest distance from each other in the graph) and then a breadth-first search from v is used to partition the vertices into levels: the vertex v belongs to the zeroth level, and all neighbors of vertices in the ith level belong to the $(i+1)$th level, for $i = 0, 1, \ldots$. Some algorithms choose the vertices in the median level as the vertex separator. A slight variant chooses the separator to be the vertices in the smallest level k such that the levels $0, 1, \ldots, k$ together contain more than half the vertices. This variant partitions the vertices into roughly equal sets. Another improvement is to remove from the separator those vertices in level k that are adjacent to vertices in level $(k-1)$ but not to vertices in level $(k+1)$. The algorithms can easily be modified to compute edge separators, and are quite fast, requiring only $\mathcal{O}(|E|)$ time, since they employ only a few breadth-first searches to compute the pseudo-peripheral vertex and the level structures. Unfortunately, the quality of the separators is quite poor, relative to the other algorithms described here.

4.2.3 Inertial Algorithm

The inertial algorithm [NORL86] employs the geometrical coordinates of the vertices of a graph embedded in two or three dimensions to compute a partition. We view the set of vertices of the mesh as a discrete point set, and compute the center of gravity of this set (x_c, y_c, z_c). Then we compute the 3×3 inertia matrix

$$I = \begin{pmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{yx} & I_{yy} & I_{yz} \\ I_{zx} & I_{zy} & I_{zz} \end{pmatrix}$$

where

$$\begin{aligned}
I_{xx} &= \textstyle\sum_i (y_i - y_c)^2 + (z_i - z_c)^2, \\
I_{yy} &= \textstyle\sum_i (x_i - x_c)^2 + (z_i - z_c)^2, \\
I_{zz} &= \textstyle\sum_i (x_i - x_c)^2 + (y_i - y_c)^2, \\
I_{xy} = I_{yx} &= \textstyle\sum_i (x_i - x_c)(y_i - y_c), \\
I_{yz} = I_{zy} &= \textstyle\sum_i (y_i - y_c)(z_i - z_c), \\
I_{xz} = I_{zx} &= \textstyle\sum_i (x_i - x_c)(z_i - z_c).
\end{aligned}$$

The eigenvector $\underline{v}_1$ associated with the smallest eigenvalue of I represents the axis of minimum angular momentum. Following this, the orthogonal projection of the coordinates of the vertices onto this eigenvector $\underline{v}_1$ is computed and the median value of these projections can be used to partition the vertices into two sets. The intuitive idea behind this algorithm is that the rotational angular momentum is minimum about the principal axis of inertia. If the domain is nearly convex, then this axis will align itself with the overall shape of the grid, and the grid will have a small spatial extent in a direction orthogonal to this axis. The virtue of the inertial algorithm is that it is fast, though the quality of the partitions may be poor. The quality can be improved by the use of a Kernighan-Lin post-processing phase. The algorithm also requires a geometric embedding of the graph.

4.2.4 The Spectral Partitioning Algorithm

The bisection problem requires the partitioning of the vertices of a graph $G = (V, E)$ into two sets A and B such that the number of "cut-edges" is minimized. The sizes of the two parts should be the same after partitioning if the graph has an even number of vertices.

The bisection problem can be formulated as the minimization of a quadratic objective function by means of the Laplacian matrix $Q = Q(G)$ of the graph G. Let $d(i)$ denote the degree of the vertex i, that is the number of vertices adjacent to i. The Laplacian matrix Q has as elements

$$q_{ij} = \begin{cases} -1 & \text{if } i \neq j \text{ and } (i,j) \in E, \\ 0 & \text{if } i \neq j \text{ and } (i,j) \notin E, \\ d(i) & \text{if } i = j \ . \end{cases}$$

The Laplacian matrix Q is symmetric and has row and column sums equal to zero. It can also be expressed in terms of two other matrices associated with a graph, as $Q = D - A$, where A is the adjacency matrix of a graph, and the D is the $n \times n$ diagonal matrix of the degrees of the vertices of G.

Let $\underline{x}$ be an n-vector with component $x_i = 1$ if $i \in A$ and $x_i = -1$ if $i \in B$. Then

$$\sum_{(i,j)\in E} (x_i - x_j)^2 = \sum_{i\in A, j\in B, (i,j)\in E} (x_i - x_j)^2 = 4\,|\delta(A,B)|\,.$$

On the other hand,

$$\underline{x}^T Q \underline{x} = \underline{x}^T D \underline{x} - \underline{x}^T A \underline{x} = \sum_{i=1}^{n} d_i x_i^2 - 2 \sum_{(i,j)\in E} x_i x_j = \sum_{(i,j)\in E} (x_i - x_j)^2 \,.$$

Thus the bisection problem is equivalent to the problem of minimizing the quadratic form $\underline{x}^T D \underline{x}$ over n-vectors $\underline{x}$ with components $x_i = \pm 1$ and $\sum_{i=1}^{n} x_i = 0$. Formally,

$$|\delta_{min}(A,B)| = \min_{x_i = \pm 1, \sum_{i=1}^{n} x_i = 0} \underline{x}^T D \underline{x}.$$

Since the bisection problem is NP-complete, it cannot be solved exactly, except if the constraint that $x_i = \pm 1$ is relaxed and each component can vary continuously in value between $+\sqrt{n}$ and $-\sqrt{n}$.

The minimizer of the relaxed problem is the second eigenvector of the Laplacian matrix. Thus the algorithm is constructed as follows: compute a second eigenvector of the Laplacian of the graph, and then partition the vertices into two sets by the median eigenvector component. It should be noted that if a partition into subsets of size k and $n - k$ are desired, then the kth most positive component (or kth most negative component) could be used to obtain a partition. In [PSL90] the spectral partitioning algorithm is used to compute separators for parallel computing, to prove additional lower bounds on separators, and to describe computational results.

4.2.5 The Geometric Algorithm

Finite element or finite difference meshes embedded in space contain geometric information about the coordinates of the mesh points. Algorithms for partitioning meshes by bisecting along coordinate axes have been considered by many authors (see for example [Wu95]). They are fast and easy to implement in parallel, but the quality of the separators obtained by such straight-line cuts are not good relative to other algorithms, especially for adapted meshes.

Another family of algorithms used for geometric partitioning, compute a separator by using a circle rather than a straight-line to cut the mesh. Given a graph embedded in d-dimensional space, the edges of the graph are disregarded and the graph is viewed as a collection of vertices. Since the graph is embedded in d-dimensional space, each vertex has a set of geometric coordinates attached to it.

A *centerpoint*, which is of major importance for such kind of algorithms, is a point such that every hyperplane through it divides the given set of points approximately evenly in two subsets. "Approximately evenly" in this case means that the worst-case ratio of the sizes of the two subsets is $d \div 1$. It can be proved that every finite point set in $\mathcal{R}_d$ has a centerpoint, and the proof yields a polynomial time algorithm that employs linear programming to compute the centerpoint. However, this algorithm is too slow to be practical, and heuristics to compute approximate centerpoints are used instead.

The geometric algorithm has several advantages. It examines only the vertices of the graph, and makes no use of the edges except to compute the quality of the generated separators. The computations involved, are simple operations on the points. However, the feature that the algorithm makes no use of the edge information in the graph is also a weakness in the partitioning of edge-weighted meshes where the weight of the cut edges needs to be minimized. Another disadvantage is that graphs arising in many areas, such as econometric modeling, do not have coordinate information since they are not embedded in space.

4.2.6 A Multilevel Algorithm

Multilevel algorithms (see for example [KK95b]) for graph partitioning are similar in spirit to multigrid algorithms for solving linear systems of equations. In this case, the given graph can be viewed as the finest graph in a sequence of coarse graphs to be computed. Given a "fine" graph, we obtain a "coarse" graph with fewer vertices by a suitable shrinking procedure. We construct a sequence of "coarse" graphs until the coarsest graph computed thus far is small enough. A high-quality partitioning algorithm such as the spectral algorithm or Kernighan-Lin algorithm is used to partition the coarsest graph. A partition of a coarse graph is then used to partition the fine graph immediately preceding it in the sequence of graphs by reversing the shrinking step used to coarsen (this is an "uncoarsening" step). Next, this "rough" partition of the fine graph is refined by means of a vertex-swapping algorithm that moves vertices between the parts to reduce the number of edges cut by the partitioning algorithm (this is a "refinement" step). The uncoarsening and refinement steps are repeated for each successive pair of fine and coarse graphs in the sequence until a partition of the given graph is computed.

After the description of the various ideas of the partitioning algorithms proposed, we give an extended description and some comparisons of the implemented algorithms in the //ELLPACK environment. Specifically, we list three groups of partitioning libraries: the //ELLPACK [HRW+96], CHACO [HL95c] and METIS [KK95c].

4.3 Native //ELLPACK Decomposition Software

The native //ELLPACK decomposition algorithms are described extensively in [Wu95, WH96]. They include the following:

PxQxS The PxQxS algorithm splits the domain along the main axis (Cartesian or Polar) after sorting the coordinates of the nodes (FD mode) or the center of mass of the elements (FEM mode).

Inertia Axis The Inertia Axis algorithm keeps splitting the domain into subdomains along the symmetry axis defined by the coordinates of the nodes (FD) or the center of mass of the elements (FEM), until the specified number of subdomains is reached.

RSB The Recursive Spectral Bisection algorithm uses eigenvector spectral search. Nodes are visited in order of increasing eigenvector values of the Laplacian matrix of the graph.

Neighborhood Search The Neighborhood Search algorithm splits the initial mesh based on the neighborhood traversal scheme. That is, the subdomains are gathered on the basis of the searching order defined.

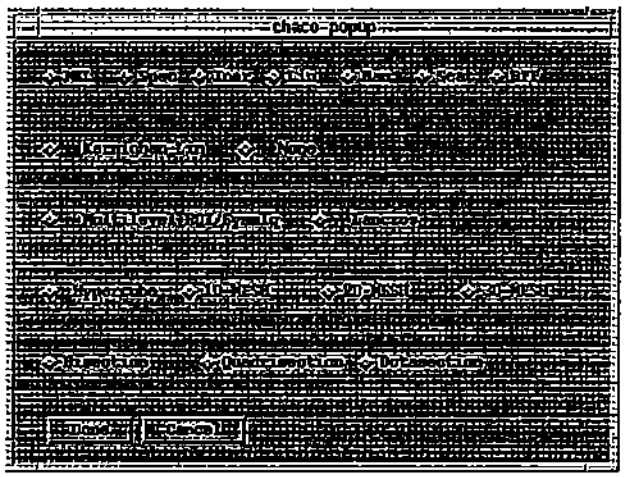

Figure 20: Graphical Interface for the CHACO Package

Clustering The objective of a data clustering algorithm is to group the mesh points into clusters such that the points within a cluster have a high degree of "natural association" among themselves, while the clusters are "relatively distinct" from each other.

The interface controlling these algorithms is depicted in Figure 15.

4.4 CHACO Graph Partitioning Software

We have incorporated the CHACO 2.0 version of CHACO [HL95c, HL93, HL95a, HL95b] into //ELLPACK.

Broadly speaking, CHACO addresses three classes of problems. First and foremost, it partitions graphs using a variety of approaches with different properties. Second, it intelligently embeds the partitions it generates into several different topologies. The topologies the code knows about are those matching the common architectures of parallel machines, namely hypercubes and meshes. Third, it can use spectral methods to sequence graphs in a manner that preserves locality. This capability has been used, for example, in data base organization, sparse matrix envelope reduction and DNA sequencing. The graphical user interface to the CHACO software package is displayed in Figure 20.

The five classes of partitioning algorithms currently implemented in CHACO are simple, spectral, inertial, Kernighan-Lin(KL) and multilevel-KL. In the first row of the user interface, the global partitioning algorithms can be selected. CHACO includes three very simple global partitioning schemes. The *linear* scheme, the *random* scheme and the *scattered* scheme. One global method for

large problems in which high quality partitions are sought is the multilevel-KL. Also a spectral method is available, where eigenvectors of the matrix constructed by the graph, are used to decide how to partition the graph. The inertial method is a relatively simple and fast partitioner that uses geometric information. CHACO also allows a partition to be read from a file (RFF) and be refined with a local method or with one of the various post-processing methods. In the second row, the user can select a local partitioning scheme, where only Kernighan-Lin is considered as such in this version. Since KL does not find very good partitions of large graphs unless it is given a good initial partition, in the context of CHACO, it is used in conjunction with one of the global partitioners. In the sequel the user can select an eigen solver, between the two offered by CHACO: a Lanczos based solver and a multilevel RQI/Symmlq (this algorithm combines a graph coarsening strategy with Rayleigh Quotient Iteration using the linear solver Symmlq to refine approximate eigenvectors projected from a coarse graph onto a finer graph). In the next line the user can specify the size of the parallel machine for which the partitioning is used. CHACO knows the topology of hypercube and mesh parallel machines. Finally, the user will choose whether to apply the partitioning method, in bisection, quadrisection, or octasection form.

4.5 METIS Unstructured Graph Partitioning

Another system for graph partitioning that has been integrated into //ELLPACK is the Version 2.0 of the METIS [KK95c] system. METIS is a set of programs that implement the various algorithms described in [KK95a, KK95b]. The basic idea behind the multilevel graph partitioning algorithms implemented in METIS is that the graph G is first coarsened down to a few hundred vertices, a bisection of this much smaller graph is computed, and then this partition is projected back towards the original graph (finer graph), by periodically refining the partition. The coarsening phase is accomplished by finding a maximal matching and collapsing together the vertices that are incident on each edge of the matching. The next phase of the multilevel algorithm is to compute a minimum edge-cut bisection of the coarse graph, such that each part contains roughly half of the vertices of the original graph. Since during coarsening, the weights of the vertices and edges of the coarser graph were set to reflect the weights of the vertices and edges of the finer graph, the graph resulted from the coarsening process contains sufficient information to intelligently enforce the balanced partition and the minimum edge-cut requirements. During the last phase, the partition of the coarsest graph is projected back to the original graph by going through the sequence of graphs created during the coarsening process. Furthermore, even if the partition of a graph is at a local minimum, the projected partition may not be at a local minimum. Since the projected graph is finer, it has more degrees of freedom that can be used to further improve the partition and thus decrease the edge-cut. Hence, it may still be possible to

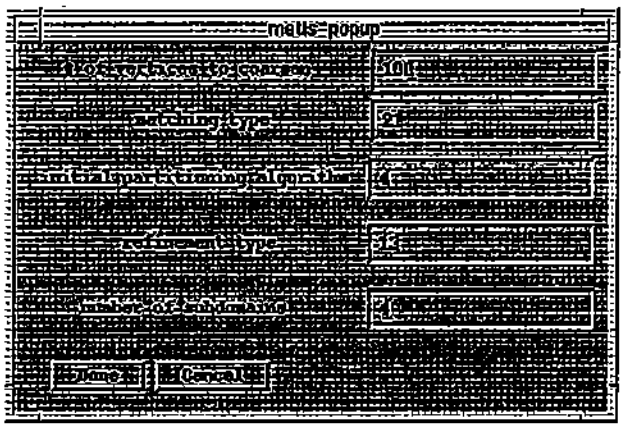

Figure 21: Graphical Interface for the METIS Package

improve the projected partition by local refinement heuristics. The graphical user interface to the METIS software package is displayed in Figure 21. The default values displayed in the window are the ones used for the algorithm which is compared with the other algorithms (from //ELLPACK and CHACO) in the various experiments.

5 Performance Evaluation of Domain Decomposition Software

In //ELLPACK there are three parallel methodologies for solving steady-state field PDE models on distributed machines. All are based on the decomposition of the associated finite element mesh or grid. In the first approach, a parallel PDE discretization module is assumed based on a decomposition of the geometric data structure involved. The second approach is based on a decomposition of the continuous PDE domain obtained from a semi-optimal mesh partitioning. Then in each subdomain/processor the sequential PDE discretizer is invoked to generate the algebraic equations. In some instances, the coefficients of the unknowns at the interface might need to be communicated among the neighbor subdomains. Finally, the decomposed system of PDE discretization equations is stored on each processor of the target machine. The third and last approach, assumes that the PDE model is discretized sequentially. Then, using the mesh decomposition data, the matrix is partitioned into blocks which are downloaded on each processor of the parallel platform selected with the execution tool. In all three methodologies, the decomposed system is solved in parallel using any of the //ELLPACK linear solvers. For the performance evaluation of the partitioning heuristics listed in Table 1, we have applied the last approach, referred

Acronym	Description	Reference
CHACO linear	Linear scheme from CHACO	[HL95c]
CHACO inertial	Inertial method from CHACO	[HL95c]
CHACO spectral	Spectral partitioning from CHACO	[HL95c]
CHACO MKL	Multilevel KL from CHACO	[HL95c]
//ELLPACK pxqxs	Partitioning of the mesh into a specified number of rows and columns	[WH93]
//ELLPACK inertial	Partitioning of the mesh by Inertia Axis Splitting	[WH93]
METIS default	Multilevel graph partitioning (PMETIS) with default values chosen by implementors from METIS	[KK95c]

Table 1: The selected mesh partitioning algorithms for their performance evaluation in the context of MPlus parallel reuse methodology

throughout as MPlus, for two elliptic model PDE problems and several domains. Following, we give a detailed description of the MPlus approach, the hardware and PDE solvers used, the numerical experiments carried out to evaluate the partitioning algorithms listed in Table 5.1, and present the numerical results obtained with an analysis.

5.1 MPlus (M+): An Off-line Parallel Reuse Methodology for Solving PDEs

M+ is a parallel framework that allows the "reuse" of the discretization part of sequential general elliptic PDE solvers. This reuse methodology is based on the "divide and conquer" computational paradigm and uses an off-line approach. It assumes that the discretization of the PDE model is realized by an existing sequential PDE solver, and it goes off-line to a parallel machine to solve the resulting system of discrete equations. For the parallel solution of the discrete equations, a partitioning of the linear system is required. This is obtained implicitly through a decomposition of the corresponding discrete PDE problem domain. The partitioned linear system is then downloaded onto the parallel machine.

The M+ [MH96] software package has a self-contained, interactive, graphical interface which can be used to obtain a domain decomposition based partition of the discrete algebraic equations generated by PDE software. Input to the tool consists of the linear system and a corresponding non-overlapping domain decomposition. In addition to the linear system partitioning module, M+ also provides modules for linear system visualization, parallel solver specification, and template-based parallel driver program generation.

The linear system input is represented in a tagged, self-identifying format which is generated by the *save linear system* output specification in //ELLPACK. The domain decomposition input is accepted in the current //ELLPACK decomposition file format. The visualization module supports the display of the original and partitioned linear systems and provides a facility to permute a par-

titioned system and visualize it in arrowhead format. Based on the parallel solver specification, the parallel solution module generates the parallel solver driver program from a template and the requisite input data files for the node processors of the MIMD platform. The current version of M+ supports the entire library of parallel ITPACK linear solvers.

5.2 Parallel Linear Solvers and Hardware Platforms

The experiments for this study were performed on two different hardware platforms: an nCUBE/2 and a network of workstations. The nCUBE/2, is a 64-node system with 4MB memory per node. The network of Sun workstations consists of 8 Sparc Station 20s with 32MB memory running Solaris 2.4. The SS20's (Model 61), are connected to both a 10Mbps Ethernet, and an ATM switch running at speeds up to 155 Mbps. The //ELLPACK PSE is supported by a parallel library of PDE modules for the numerical simulation of stationary and time dependent PDE models on two and three dimensional regions. The parallel PDE solver libraries are based on the "divide and conquer" computational paradigm and utilize the discrete domain decomposition approach for problem partitioning and load balancing. The parallel implementation of the ITPACK [KRG82] linear solver library is integrated in the //ELLPACK PSE and is applicable to any linear system stored in //ELLPACK's distributed storage scheme. It consists of seven modules implementing Successive Over-Relaxation (SOR) with Red/Black (RB) ordering, Jacobi Conjugate Gradient (CG) with Natural and RB ordering, Jacobi with Chebyshev Accelaration (SI) and N/RB ordering, Reduced System CG (RSCG) with RB ordering, Reduced System SI (RSSI) with RB ordering, Symmetric SOR CG (SSOR-CG) with RB ordering, and Symmetric SOR SI (SSOR-SI) with RB ordering [Kim93]. The code is based on the sequential version of ITPACK which was parallelized by utilizing a subset of level two sparse BLAS routines. The communication modules of the parallel ITPACK library have been implemented for several MIMD platforms using different native and portable communication libraries. The implementations utilize standard send/receive, reduction, barrier synchronization and broadcast communication primitives from these message passing communication libraries. No particular machine configuration topology is assumed in the implementation. In the context of our measurements we are using the MPI [GLS94] portable communication library of the parallel ITPACK implementation.

5.3 Experimental Results

For the evaluation of the integrated mesh decomposition libraries, we have selected two simple 3-D elliptic PDE models:

$$u_{xx} + u_{yy} + u_{zz} = 0 \qquad (1)$$

procs	fem solve	speedup	vecfem solve	speedup	decomposition
1	30.65	1.00	330.95	1.00	-
2	16.63	1.84	138.36	2.39	CHACO linear
	16.11	1.90	140.12	2.36	CHACO inertial
	16.02	1.91	143.04	2.31	CHACO spectral
	15.99	1.92	133.34	2.48	CHACO MKL
	15.78	1.94	128.33	2.57	//ELLPACK pxqxs
	15.90	1.93	133.81	2.47	METIS default
4	8.51	3.60	66.99	4.94	CHACO linear
	8.12	3.77	66.51	4.97	CHACO inertial
	8.17	3.75	64.52	5.12	CHACO spectral
	8.41	3.64	66.88	4.94	CHACO MKL
	8.23	3.72	63.39	5.22	//ELLPACK pxqxs
	8.41	3.64	67.08	4.93	METIS default
8	5.37	5.70	36.03	9.18	CHACO linear
	4.93	6.21	33.025	10.02	CHACO inertial
	5.00	6.13	35.65	9.28	CHACO spectral
	5.00	6.13	31.26	10.58	CHACO MKL
	5.43	5.64	33.98	9.73	//ELLPACK pxqxs
	4.76	6.43	33.24	9.95	METIS default

Table 2: The parallel ITPACK Jacobi CG time (seconds) and fixed speedups for //ELLPACK FEM and VECFEM discretization systems with different partitioning schemes on an ethernet based cluster of SS20 workstations. The system of equations was generated using FEM and VECFEM modules applied to PDE problem (5.1) defined on the union of a cube and a cylinder. The mesh used had 16198 nodes and 72028 elements.

$$u_{xx} + u_{yy} + u_{zz} = 3e^{x+y+z} \tag{2}$$

with Dirichlet boundary conditions defined on a cylinder, a union of a cube with a cylinder, a cube with a hole, and an airplane (Figure 22).

For the discretization of (5.1) and (5.2), we are applying two different //ELLPACK software modules implementing the linear finite element method, known as VECFEM [GS91] and FEM. The generated PDE discretization systems are partitioned and downloaded on the selected hardware platforms, using the MPlus approach. The corresponding algebraic systems obtained are solved in parallel, using the //ITPACK Jacobi CG solver. The mesh partitioning required were obtained using the software modules listed in Table 1. The parallel solution times and the sequential mesh partitioning times are measured and listed in the following tables.

In Table 2, we observe that for the linear FEM discretization module, the fixed speedup is smaller than the VECFEM module. This is due to the difference of the number of equations per processor. In the case of FEM module the boundary degrees of freedom are eliminated during the discretization process. The super linear speedup observed for the VECFEM discretization, is due to paging since the large system cannot be stored in core memory.

The data of Table 2 and Table 3 suggest that the optimality of all partitioning algorithms of Table 1, with respect to parallel execution of the linear system or fixed speedup, is almost equivalent on the two clusters of workstations con-

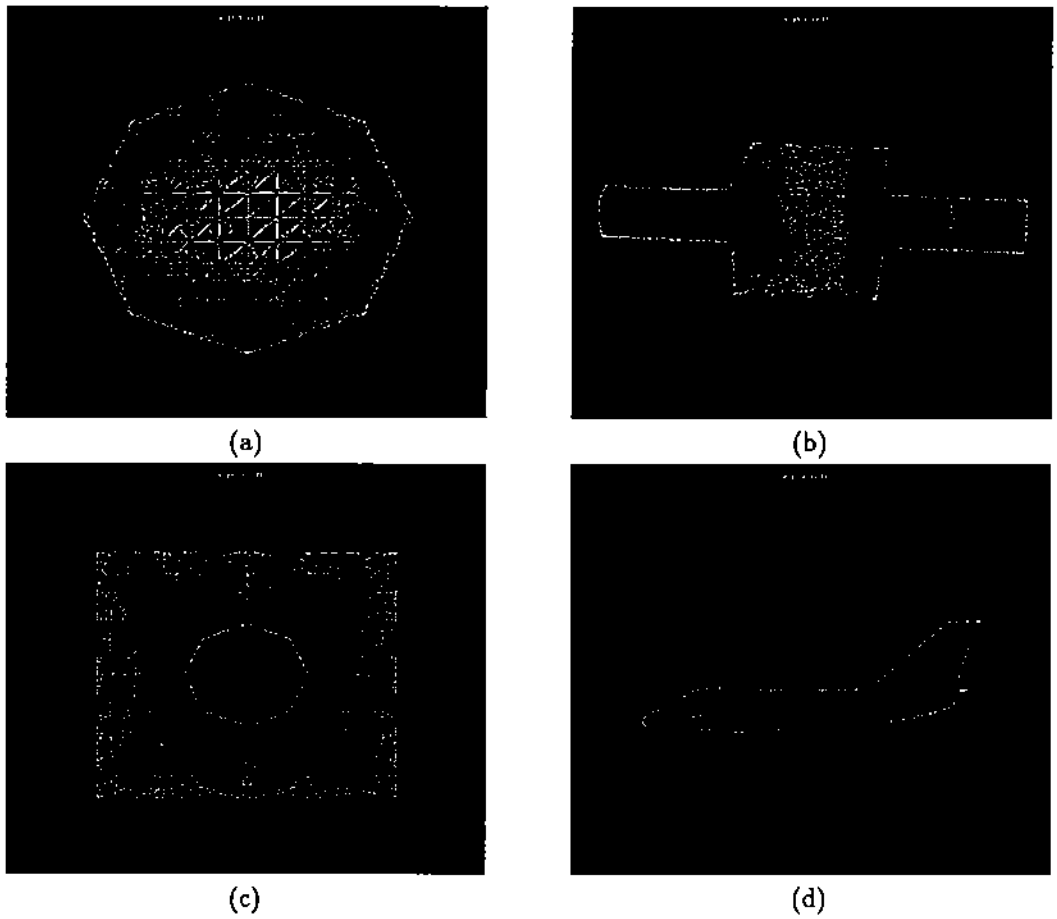

Figure 22: (a) Cylinder, (b) Cube with a Cylinder, (c) Cube with a Hole (d) Airplane

procs	fem solve	speedup	vecfem solve	speedup	decomposition
1	31.096	1.00	330.95	1.00	-
2	16.49	1.88	136.93	2.41	CHACO linear
	15.93	1.95	139.96	2.41	CHACO inertial
	18.88	1.64	143.45	2.30	CHACO spectral
	16.04	1.93	133.07	2.48	CHACO MKL
	15.67	1.98	126.91	2.60	//ELLPACK pxqxs
	15.96	1.94	134.68	2.45	METIS default
4	12.25	2.53	66.83	4.95	CHACO linear
	7.98	3.89	66.63	4.96	CHACO inertial
	8.00	3.88	64.77	5.10	CHACO spectral
	9.10	3.41	66.91	4.94	CHACO MKL
	8.24	3.77	63.24	5.23	//ELLPACK pxqxs
	10.18	3.05	67.95	4.87	METIS default
8	7.63	4.07	35.88	9.22	CHACO linear
	6.11	5.08	32.70	10.12	CHACO inertial
	5.79	5.37	35.26	9.38	CHACO spectral
	5.58	5.57	31.53	10.49	CHACO MKL
	6.09	5.10	33.92	9.75	//ELLPACK pxqxs
	6.17	5.03	33.30	9.93	METIS default

Table 3: The parallel ITPACK Jacobi CG time (seconds) and fixed speedups for //ELLPACK FEM and VECFEM discretization systems with different partitioning schemes on an ATM based cluster of SS20 workstations. The system of equations was generated using FEM and VECFEM modules applied to PDE problem (5.1) defined on the union of a cube and a cylinder. The mesh used had 16198 nodes and 72028 elements.

sidered. If we add the time to generate the decomposition (see Table 5), then we observe that PxQxS algorithm performs the best with METIS default and CHACO MKL and Inertial following very closely (see Figure 23 for the FEM case). CHACO spectral is not competitive due to its large computational cost.

Table 4 lists the nCUBE/2 times in seconds and fixed speedups of parallel ITPACK Jacobi CG solver for the FEM module applied to the Poisson problem (5.1) defined on the union of a cube and a cylinder. The mesh used has 16198 nodes and 72028 elements. Table 4 and Table 5, suggest that METIS default, is performing the best for large number of processors ($n > 8$) with //ELLPACK PxQxS and CHACO MKL and Inertial sufficiently close. For $n < 16$, PxQxS performs the best.

The data in Table 6 include timings of parallel ITPACK Jacobi CG on the nCUBE/2 for the FEM discretizer, fixed speedups, and sequential decomposition time on a SS20. The FEM module is applied to the model PDE problem defined on the cube with a hole (Figure 22 (b)) using a mesh with 16198 nodes and 72028 elements. In this case METIS performs the best for $n > 16$, while CHACO outperforms the rest for $n < 32$. It's worth noticing that the fixed speedup does not improve in the case of 64 processors. Table 7 lists the sequential times of a part of the partitioning algorithms listed in Table 1 for the domain depicted in Figure 22 (d). The data suggest, the expected behavior, that the simpler algorithms are the less costly.

procs	fem solve	speedup	decomposition
1	169.82	1.00	-
2	97.07	1.74	CHACO linear
	90.04	1.88	CHACO inertial
	106.98	1.58	CHACO spectral
	91.06	1.86	CHACO MKL
	85.05	1.99	//ELLPACK pxqxs
	91.02	1.86	METIS default
4	74.12	2.29	CHACO linear
	47.22	3.59	CHACO inertial
	46.51	3.65	CHACO spectral
	52.20	3.25	CHACO MKL
	44.70	3.80	//ELLPACK pxqxs
	58.91	2.88	METIS default
8	38.58	4.40	CHACO linear
	31.71	5.35	CHACO inertial
	28.94	5.86	CHACO spectral
	28.97	5.86	CHACO MKL
	30.82	5.51	//ELLPACK pxqxs
	32.50	5.22	METIS default
16	18.91	8.98	CHACO MKL
	26.14	6.49	//ELLPACK pxqxs
	18.27	9.29	METIS default
32	12.07	14.06	CHACO MKL
	20.46	8.30	//ELLPACK pxqxs
	11.76	14.44	METIS default
64	11.80	14.39	CHACO MKL
	19.50	8.70	//ELLPACK pxqxs
	12.70	13.37	METIS default

Table 4: The parallel ITPACK Jacobi CG time (seconds) and fixed speedups for //ELLPACK FEM discretization system with different partitioning schemes on the nCUBE/2. The system of equations was generated using FEM module applied to PDE problem (5.1) defined on the union of a cube with a cylinder. The mesh used had 16198 nodes and 72028 elements.

procs	time	decomposition
2	2.85	CHACO linear
	2.04	CHACO inertial
	407.93	CHACO spectral
	2.51	CHACO MKL
	0.69	//ELLPACK pxqxs
	2.09	METIS default
4	4.31	CHACO linear
	2.76	CHACO inertial
	-	CHACO spectral
	3.94	CHACO MKL
	1.11	//ELLPACK pxqxs
	2.73	METIS default
8	5.60	CHACO linear
	3.55	CHACO inertial
	-	CHACO spectral
	4.62	CHACO MKL
	1.58	//ELLPACK pxqxs
	3.33	METIS default
16	6.75	CHACO MKL
	1.80	//ELLPACK pxqxs
	4.01	METIS default
32	8.49	CHACO MKL
	2.11	//ELLPACK pxqxs
	4.81	METIS default
64	10.57	CHACO MKL
	2.42	//ELLPACK pxqxs
	5.65	METIS default

Table 5: Sequential time of the various decomposition algorithms and number of subdomains on a SS20 workstation for the decomposition of the finite element mesh of domain depicted in Figure 5.1b with 16198 nodes and 72028 elements.

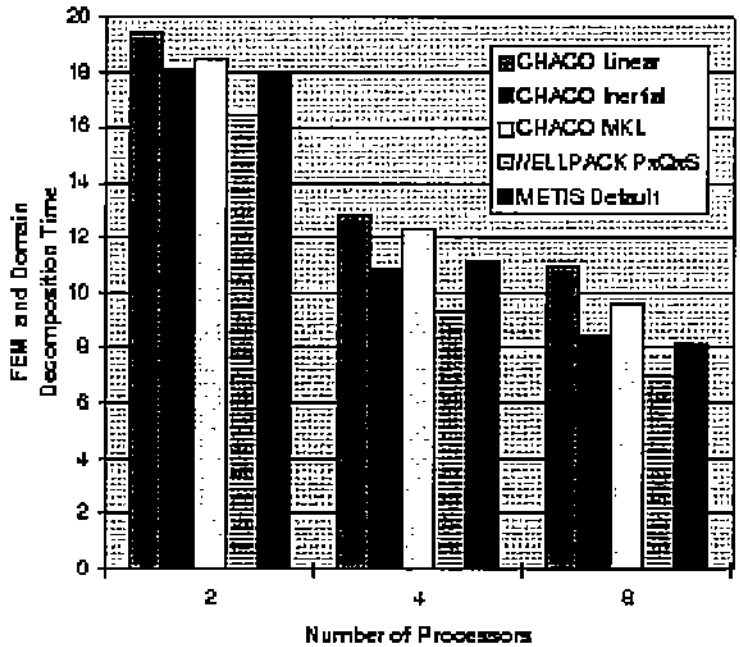

Figure 23: The FEM data of Table 5.2 combined with the data from Table 5.5 presented in a bar graph.

6 Conclusions

We have presented a detailed description of the 3-D pre-processing and post-processing user interface of the //ELLPACK PSE and the infrastructure used to implement it. In addition we have started an extensive evaluation of finite element partitioning algorithms within the //ELLPACK environment. The results suggest that the coordinate based heuristics (PxQxS and Inertial), the CHACO MKL, and METIS default algorithms, can produce partitionings with very small cost that lead to significant speedups of the underlying computations even for moderate size PDE discretization models.

References

[BDG+92] Adam Beguelin, Jack Dongarra, Al Geist, Robert Manchek, and Vaidy Sunderam. A User's Guide to PVM: Parallel Virtual Machine. Technical Report ORNL/TM-11826, Oak Ridge National Laboratory, Engineering Physics and Mathematics Division, Mathematical Sciences Section, 1992.

[BHK85] R. F. Boisvert, S. E. Howe, and D. K. Kahaner. GAMS - A Framework for the Management of Scientific Software. *ACM Trans. Math. Soft.*, 11:313–355, 1985.

[BLCL+94] Tim Berners-Lee, Robert Cailliau, Ari Luotonen, Henrik Frystyk Nielsen, and Arthur Secret. The World-Wide Web. *Communications of the ACM*, 37(8):76–82, August 1994.

[CGLN84] C. Chu, J. A. George, J. W. Liu, and E. G. Ng. User's Guide for SPARSPAK-A: Waterloo Sparse Linear Equations Package. Technical Report CS-84-36, Computer Science, University of Waterloo, Ontario, Canada, 1984.

[CHR94] N. P. Chrisochoides, E. N. Houstis, and J. R. Rice. Mapping Algorithms and Software Environments for Data Parallel PDE Solvers. *Special Issue of the Journal of Parallel and Distributed Computing on Data-Parallel Algorithms and Programming*, 21(1):75–95, April 1994.

[GHR92] E. Gallopoulos, E. Houstis, and J. R. Rice. Future Research Directions in Problem Solving Environments, for Computational Science. Technical Report CSD-TR-92-0032, Department of Computer Sciences, Purdue University, 1992.

[GLS94] W. Gropp, E. Lusk, and A. Skjellum. *Using MPI: Portable Parallel Programming with the Message Passing Interface.* MIT Press, October 1994.

[Gro77] The MATHLAB Group. *MACSYMA Reference Manual, Version 9.* Laboratory for Computer Science, M.I.T., Cambridge, 1977.

[Gro87] Symbolic Computation Group. *Maple User's Guide.* University of Waterloo, Department of Computer Science, Waterloo, Canada, 1987.

[GS91] L. Gross and P. Sternecker. *The Finite Element Tool Package VECFEM.* University of Karlsruhe, 1991.

[HL93] B. Hendrickson and R. Leland. An Improved Spectral Load Balancing Method. In *Proc. 6th SIAM Conf. Parallel Processing for Scientific Computing*, pages 953–961, March 1993.

[HL95a] B. Hendrickson and R. Leland. An Improved Spectral Graph Partitioning Algorithm for Mapping Parallel Computations. *SIAM J. Sci. Computing*, 16, 1995.

[HL95b] B. Hendrickson and R. Leland. A Multilevel Algorithm for Partitioning Graphs. In *Proc. Supercomputing '95*. ACM, December 1995.

[HL95c] B. Hendrickson and R. Leland. *The Chaco User's Guide, Version 2.0*. Sandia National Laboratories, July 1995.

[HRW+96] E. N. Houstis, J. R. Rice, S. Weerawarana, A. C. Catlin, P. Papachiou, K. Y. Wang, and M. Gaitatzes. Parallel (///) ELLPACK: A Problem Solving Environment for PDE Based Applications on Multicomputer Platforms. Technical Report CSD-TR-96-070, Department of Computer Sciences, Purdue University, 1996.

[Inc94] X Business Group Inc. *Interface Development Technology*. 3155 Kearney Street, Suite 160, Fremont, CA, 1994.

[Kil93] Mark Kilgard. OpenGL and X, Part 1: An Introduction. *The X Journal, SIGS Publications*, November/December 1993.

[Kil94a] Mark Kilgard. OpenGL and X, Part 2: Using OpenGL with Xlib. *The X Journal, SIGS Publications*, January/February 1994.

[Kil94b] Mark Kilgard. OpenGL and X, Part 3: Using OpenGL with Motif. *The X Journal, SIGS Publications*, July/August 1994.

[Kim93] S. B. Kim. *Parallel Numerical Methods for Partial Differential Equations*. PhD thesis, Department of Computer Sciences, Purdue University, 1993.

[KK95a] G. Karypis and V. Kumar. Analysis of Multilevel Graph Partitioning. Technical Report TR 95-037, Department of Computer Science, University of Minnesota, 1995.

[KK95b] G. Karypis and V. Kumar. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. Technical Report TR 95-035, Department of Computer Science, University of Minnesota, 1995.

[KK95c] George Karypis and Vipin Kumar. *METIS Unstructured Graph Partitioning and Sparse Matrix Ordering*. University of Minnesota, Department of Computer Science, August 26, 1995.

[KL70] B. Kernighan and S. Lin. An Efficient Heuristic Procedure for Partitioning Graphs. *Bell System Technical Journal*, 29:291–307, 1970.

[KRG82] D. Kinkaid, J. Respess, and R. Grimes. Algorithm 586: Itpack 2c: A Fortran Package for Solving Large Linear Systems by Adaptive Accelerated Iterative Methods. *ACM Trans. Math. Soft.*, 8(0):302–322, 1982.

[MH96] S. Markus and E. N. Houstis. Parallel Reuse Methodologies for Elliptic Boundary Value Problems. Technical Report CSD-TR-96-056, Department of Computer Sciences, Purdue University, 1996.

[MLB90] C. Moler, J. Little, and S. Bangert. *PRO MATLAB for Sun Workstations: User's Guide.* The MathWorks, Inc., Sherborn, 1990.

[MR92] Brad A. Myers and Mary Beth Rosson. Survey on User Interface Programming. In *Proc. of SIGCHI'92, Human Factors in Computing Systems*, pages 195–202, Monterey, May 1992.

[Mye94] Brad A. Myers. Challenges of HCI Design and Implementation. *ACM Interactions*, 1(1):73–83, January 1994.

[NORL86] B. Nour-Omid, A. Raefsky, and G. Lyzenga. Solving Finite Element Equations on Concurrent Computers. *Parallel Computations and their Impact on Mechanics, A. K. Noor, ed.*, pages 209–227, New York, American Soc. of Mech. Eng. 1986.

[Ous90] J. Ousterhout. An Embeddable Command Language. In *Proc. of the USENIX Winter Conference*, pages 133–146, January 1990.

[Pau96a] Brian Paul. The Mesa 3-D Graphics Library. University of Winsconsin-Space Science and Engineering Center, May 1996.

[Pau96b] Brian Paul. *The Mesa 3-D Graphics Library.* http://www.ssec.wisc.edu/~brianp/Mesa.html, 1996.

[PB96] Brian Paul and Ben Bederson. *Togl – a Tk OpenGL Widget.* http://www.ssec.wisc.edu/~brianp/Togl.html, 1996.

[PSL90] A. Pothen, H. D. Simon, and K. P. Liou. Partitioning Sparse Matrices with Eigenvectors of Graphs. *SIAM J. Matrix Anal. Appl.*, 11:430–452, 1990.

[Ric89] J.R. Rice. Libraries, Software Parts and Problem Solving Systems. In Cai, Fosdick and Huang (Eds.) Symposium on Scientific Software, 191-203, Tsinghua Univ. Press, 1989.

[Ric96] John R. Rice. Scalable Scientific Software Libraries and Problem Solving Environments. Technical Report CSD-TR-96-001, Department of Computer Sciences, Purdue University, 1996.

[SA92] Mark Segal and Kurt Akeley. The OpenGL Graphics System: A Specification. Technical report, Silicon Graphics Computer Systems, Mountain View, 1992.

[Wee94] Sanjiva Weerawarana. *Problem Solving Environment for Parallel Differential Equation based Applications.* PhD thesis, Department of Computer Sciences, Purdue University, August 1994.

[WH93] Poting Wu and E. N. Houstis. Parallel Dynamic Mesh Generation and Domain Decomposition. Technical Report CSD-TR-93-075, Department of Computer Sciences, Purdue University, 1993.

[WH96] P. Wu and E. N. Houstis. Parallel Adaptive Mesh Generation and Decomposition. *Engineering with Computers*, 1(12):155–167, 1996.

[Wol96] Stephen Wolfram. *The MATHEMATICA Book.* Wolfram Media, Inc., February 1996.

[Wom92] Paula Womack. PEX Protocol Specification and Encoding, Version 5.1p. *The X Resource, Special Issue A*, May 1992.

[Wu95] Poting Wu. *Parallel Electronic Prototyping of Physical Objects.* PhD thesis, Department of Computer Sciences, Purdue University, 1995.

procs	fem solve	speedup	decomposition time	decomposition type
1	168.91	1.00	-	-
2	86.45	1.95	0.61	//ELLPACK pxqxs
	84.89	1.99	0.44	//ELLPACK inertial
	85.20	1.98	1.46	CHACO MKL
	84.88	1.99	0.95	CHACO inertial
	87.81	1.92	1.40	METIS default
4	57.53	2.93	0.76	//ELLPACK pxqxs
	45.52	3.71	0.36	//ELLPACK inertial
	44.72	3.77	2.42	CHACO MKL
	44.19	3.82	1.62	CHACO inertial
	45.57	3.70	1.70	METIS default
8	37.75	4.47	0.92	//ELLPACK pxqxs
	29.10	5.80	0.36	//ELLPACK inertial
	25.63	6.59	3.11	CHACO MKL
	28.81	7.09	2.03	CHACO inertial
	24.96	6.76	2.43	METIS default
16	26.60	6.35	0.98	//ELLPACK pxqxs
	18.58	9.09	0.38	//ELLPACK inertial
	16.85	10.02	4.50	CHACO MKL
	15.87	10.64	2.78	CHACO inertial
	16.54	10.21	2.78	METIS default
32	19.20	8.79	2.41	//ELLPACK pxqxs
	17.23	9.80	0.38	//ELLPACK inertial
	14.60	11.56	5.73	CHACO MKL
	12.72	13.29	3.63	CHACO inertial
	12.24	13.80	3.41	METIS default
64	18.15	9.30	1.54	//ELLPACK pxqxs
	17.73	9.52	0.36	//ELLPACK inertial
	15.43	10.94	6.83	CHACO MKL
	12.70	13.30	4.50	CHACO inertial
	12.37	13.65	3.58	METIS default

Table 6: The parallel ITPACK Jacobi CG time (seconds), fixed speedups, and sequential time of the various decomposition algorithms for //ELLPACK FEM discretization system with different partitioning schemes on the nCUBE/2 for the finite element mesh of the domain Ω depicted in Figure 5.1c with 10169 nodes and 45873 elements.

procs	time	decomposition
2	0.67	//ELLPACK pxqxs
	0.81	//ELLPACK inertial
	2.19	CHACO MKL
	1.58	CHACO inertial
	2.85	METIS default
4	1.06	//ELLPACK pxqxs
	0.75	//ELLPACK inertial
	3.49	CHACO MKL
	2.65	CHACO inertial
	3.13	METIS default
8	1.63	//ELLPACK pxqxs
	0.73	//ELLPACK inertial
	5.20	CHACO MKL
	3.81	CHACO inertial
	3.85	METIS default
16	1.91	//ELLPACK pxqxs
	0.68	//ELLPACK inertial
	6.58	CHACO MKL
	5.12	CHACO inertial
	4.67	METIS default
32	2.10	//ELLPACK pxqxs
	0.73	//ELLPACK inertial
	8.53	CHACO MKL
	6.39	CHACO inertial
	5.35	METIS default
64	2.38	//ELLPACK pxqxs
	0.74	//ELLPACK inertial
	11.02	CHACO MKL
	8.23	CHACO inertial
	6.29	METIS default

Table 7: Sequential time of the various decomposition algorithms and number of subdomains on a SS20 workstation for the decomposition of a finite element mesh domain Ω depicted in Figure 5.1d with 18924 nodes and 86448 elements.