Prototype Implementation of the Comprehensive Parallel Integration Tool ParInt

Jay Ball
Western Michigan University

Follow this and additional works at: https://scholarworks.wmich.edu/masters_theses

Part of the Computer Sciences Commons

Recommended Citation
https://scholarworks.wmich.edu/masters_theses/4241

This Masters Thesis-Open Access is brought to you for free and open access by the Graduate College at ScholarWorks at WMU. It has been accepted for inclusion in Master's Theses by an authorized administrator of ScholarWorks at WMU. For more information, please contact maira.bundza@wmich.edu.
PROTOTYPE IMPLEMENTATION OF THE COMPREHENSIVE PARALLEL INTEGRATION TOOL PARINT

by

Jay Ball

A Thesis
Submitted to the
Faculty of The Graduate College
in partial fulfillment of the
requirements for the
Degree of Master of Science
Department of Computer Science

Western Michigan University
Kalamazoo, Michigan
April 1997
Copyright © 1997
Jay Ball
ACKNOWLEDGEMENTS

I would like to express my gratitude to my thesis advisors. Dr. Elise de Doncker who started and kept me involved in the parallel world. And to Dr. Ajay Gupta who prodded me a lot when I procrastinated a little too much. Also, Dr. Robert Trenary for just being his cheerful self.

Thanks to Tara, Heidi, Becky, Kristin, and Kelley. To the members of The Computer Club of Western Michigan University. To Jake, Lara, James, Tim, John, and Chris. To Nancy Emmerich. To Ken Thies. And of course, J.R. Bob Dobbs.

More thanks to Darren, Ozz, Valentina, Martin, and Jessie. To Bridget, Josh, Wendi, Cory, and Zach. To Nate, Andrea, Sherry, and Amy. To André, Dan, Andrew, and Ernie. To Sandy, and Amy. To Kerri and Kathy. To Mike and Adrian in far away places. To my roommate Marshall. To Larry, April, and Ben for always being there.

Great thanks goes out to my family, my godparents Margean and Edward, and my parental units: Robert, Lou Ann, and Lon.

Jay Ball
PROTOTYPE IMPLEMENTATION OF THE COMPREHENSIVE PARALLEL INTEGRATION TOOL PARINT

Jay Ball, M.S.

Western Michigan University, 1997

The *ParInt* project allows users to integrate multivariate functions using parallel computers via different methods encompassed in an easy to use interface. This thesis describes the initial version of the *ParInt* package. Various pop up windows of the graphical user interface, program procedures, programming paradigms, integration theory, and future considerations are all described.
# TABLE OF CONTENTS

ACKNOWLEDGEMENTS ................................. ii

LIST OF TABLES .................................. vi

LIST OF FIGURES ................................. vii

CHAPTER

I. INTRODUCTION ................................. 1
   Thesis Version ................................ 2
   A Small Glossary .............................. 2
   Notation .................................... 4
   Thesis Content and Composition Method .... 5

II. PARALLEL PROCESSING ......................... 7
   Parallel Environments ....................... 7
       PVM ..................................... 7
       MPI .................................... 8
       nCUBE2 ................................ 8
       Remarks ................................ 8
   Parallel Numeric Integration .............. 9
   Parallel Implementation Results ......... 11

III. PROGRAMMING OF PARINT ................... 14
Table of Contents – Continued

CHAPTER

Installation .................................................. 14
User Interface .................................................. 15
  Main Window ................................................. 16
  Function Window ............................................. 17
  Compile Window ............................................. 18
  Integration Window ......................................... 19
  Number of Processes Window ............................. 20
  Variable Window ........................................... 21
  Help System .................................................. 22
Execution Platforms ....................................... 23
Parallel Platforms ........................................... 25
Directory Layout .............................................. 26
Internal Procedures ......................................... 27
  PVM Procedures ............................................. 27
  Tcl/Tk Procedures .......................................... 32
IV. THE FUTURE ............................................... 35
  Proverbial “ToDoList” .................................... 35
  Improved Function Loading ............................... 35
<table>
<thead>
<tr>
<th>CHAPTER</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Different Integration Rules</td>
<td>39</td>
<td></td>
</tr>
<tr>
<td>New Number of Processes</td>
<td>39</td>
<td></td>
</tr>
<tr>
<td>Enhancements</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>Report Generation</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>Status Box</td>
<td>42</td>
<td></td>
</tr>
<tr>
<td>Multiple Languages</td>
<td>43</td>
<td></td>
</tr>
<tr>
<td>Rewrite in Java</td>
<td>43</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>APPENDICES</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A. The List File</td>
<td>46</td>
<td></td>
</tr>
<tr>
<td>B. Group Files</td>
<td>49</td>
<td></td>
</tr>
<tr>
<td>C. Makefiles</td>
<td>52</td>
<td></td>
</tr>
<tr>
<td>D. Installation Guide</td>
<td>61</td>
<td></td>
</tr>
<tr>
<td>E. GUI Source Code</td>
<td>63</td>
<td></td>
</tr>
<tr>
<td>F. PVM Source Code</td>
<td>195</td>
<td></td>
</tr>
</tbody>
</table>

BIBLIOGRAPHY | 247 |
LIST OF TABLES

1. Timings on PVM ........................................ 12
LIST OF FIGURES

1. Adaptive Integration Algorithm ............................................. 10
2. Partitioning Loop .............................................................. 10
3. Speedup With Various Processes Quantities & Error Tolerances .... 12
4. The ParInt Main Window ....................................................... 16
5. The Choose Function Window ............................................... 17
6. The Compile Window ............................................................ 19
7. The Integration Window ....................................................... 20
8. The Number of Processes Window ......................................... 21
9. The System Variable Window ............................................... 22
10. The Help System ............................................................... 23
11. User's Directory Layout ..................................................... 27
12. Directory Layout of ParInt .................................................. 28
13. Structure of ParInt's Heap .................................................. 29
14. Multi-Purpose C Header File ............................................... 37
15. Pseudo-Prototype for C File Parser ..................................... 38
16. Template for Auto-Generated HTML File .............................. 41
17. Template for Auto-Generated \LaTeX File ............................... 41
18. Resulting Appearance of Either Template ............................. 42
CHAPTER I

INTRODUCTION

One of the basic foundations of calculus is the integral, the area under a curve (and its multivariate extensions). This area can be obtained with computer programs via symbolic or estimation (numeric) methods. Using computers to compute integrals symbolically is a natural extension of paper and pencil integration, however not every integral can be solved symbolically. So, various numerical methods are used. Since the numerical methods are estimations, the accuracy will be inferior to the exact symbolic methods. The problems that we are concerned with are far too complex for symbolic methods, hence we are forced to use the numeric methods. Yet, even with fast computers, it takes a great deal of time to numerically compute integrals in many cases. To solve the problems in a reasonable amount of time, we use parallel techniques. This brings us to ParInt.

ParInt's major goal is to provide novice as well as experienced users with parallel techniques of integration problem solving. To simplify its use, ParInt has a simple graphical interface that allows end users to have point-and-click integration. ParInt supports major parallel standards, such as PVM and MPI, and runs on many platforms and Unix variants. With added support for modular components, the user can mix and match various integration methods, rules, and
This thesis describes the philosophy, design, and functionality of ParInt, not necessarily how to use it. Examining the help system, user manual, and installation guide will show how to use ParInt. All documentation is presented in the Appendices. Note that most everything is in the form of a hypertext system, so it may not flow very well when read from paper. The installation guide is also presented in the thesis on page 61. Remember that ParInt is an ongoing project; some of the screen shots, features, and faults in any version may not match what is described here within this thesis. The entire ParInt system is available from http://www.cs.umnich.edu/~parint.

Thesis Version

This is version 1.02 (dated May 15, 1997) of this document. The source code described within is the February 28, 1997 compile. Newer versions of both the thesis and source code, either updated by myself or others, will be placed on the Web Address shown above.

A Small Glossary

To prevent misunderstandings, we will define terms which could be unknown and/or ambiguous.

1. GUI An acronym for graphical user interface. Pronounced “goo-ee”.
2. **ISO-LATIN** The standard font used in the majority of the computing universe. This is the “text mode” used by the major European languages with some variation of the Roman character set. This is somewhat related to ASCII.

3. **MPI** Message Passing Interface. This is described in detail on page 8.

4. **Node** refers to the smallest distinct parallel processing element. This could be a CPU as on the nCUBE2 or a process in PVM or MPI.

5. **ParAdmin** The parallel tools administrator. The person who installs ParInt, PVM, MPI, etc. This person is responsible for the upkeep of everything that ParInt requires to operate. The ParAdmin may or may not be the same as the SysAdmin.

6. **ParInt** The overall ideal of the ParInt Project.

7. **ParInt GUI** The graphically interactive portion of the entire ParInt system.

8. **ParInt prototype** The initial version of the package including the GUI, PVM portions, and documentation. Other portions exist and are not necessarily described in this thesis. ParInt is an on-going project and this is only an initial writing.

9. **platform** A type of computer, CPU, and/or operating system.

10. **PVM** Parallel Virtual Machine. This is described in detail on page 7.

11. **user** The everyday end user of ParInt who has no special system privileges.
The following typographical notations will be used throughout this paper:

1. **Dollar** represents a Tcl/Tk or Unix system environment variable named Dollar;

2. *emphasized* text shows new terms, Internet URLs, highly emphasized words, and some specialized or unique program’s names;

3. *MATH*CAL*IC* font text implies some sort of set or special variable;

4. *monospaced* text denotes procedure names, very common programs or shell commands, file type (ie, extensions), or text to type into the computer;

5. *overarrow*’s denote vectors; and

6. *“quotation marks”* surround simple ideas defined near the occurrence, slightly emphasized words, or single buttons like “Cancel”;

7. *simple:colon:lists* denote a nested menu where the user clicks on “simple”, pulls down/over to “colon”, and pulls down/over to “lists”.

Since periods, commas, explanation marks, et. al. are used so often in computer programming, punctuation is placed outside of *emphasized*, “quoted”, or *monospaced* text by default so as to not cause confusion. If punctuation is placed inside the special text, then it is part of the special text.
Thesis Content and Composition Method

The thesis describes the ParInt Prototype, contributed by this author to the larger ParInt Project. It consists of the GUI, documentation, and source code for integrals over hyper-rectangular regions using an adaptive algorithm without load balancing and runs on the PVM platform. The documentation for ParInt is terse at this time, so the thesis will concentrate on the GUI and PVM aspects of the ParInt Prototype.

This paper was composed on a 486 based laptop running Linux, sometimes uploaded to my purple SGI because of its wonderfully huge monitor. Vim was the text editor, \LaTeX{} was the typesetter, and METAFONT was the typeface generator. A style file and accompanying \TeX{} macro files were written by myself and others to meet the WMU formatting requirements\cite{10}. The advantage of using \LaTeX{} over using typical word processors is that my thesis can be examined and used in the future without worry of the file format being incompatible. All screen shots of ParInt are in Postscript format and were captured using John Bradley's \texttt{xv} program. Diagrams were designed in \texttt{xfig}, saved in \LaTeX{} picture format, and simply \texttt{\input{}} into the document. The was auto-generated with \texttt{BibTeX}.

ParInt itself was designed around and implemented on Sun SPARCs with GNU's GCC compiler and Tcl/Tk. At times, it was manipulated, massaged,
and tested on my Linux laptop. Both the Suns and Linux laptop enforce the client/server model. This may will present problems on "unique" platforms, as described in later sections.
CHAPTER II

PARALLEL PROCESSING

Parallel Environments

Both PVM (Parallel Virtual Machine) and MPI (Message Passing Interface) allow programs written in C/C++ and/or Fortran to communicate with each other. Both environments support communications across different processes, CPUs, even computers on a network. However, each has a slightly different method of interfacing with other processes. The nCUBE2 has yet another interface with similarities to both PVM and MPI.

PVM

PVM is a programming environment which simulates virtual machines in the form of processes. These processes can exist on the same CPU, different CPU, different machines, or even different architectures and/or operating systems. Each process communicates with a daemon on the same computer and the daemon transmits the data to other processes where ever they reside. More information on PVM can be found in [9].
**MPI**

MPI is a more universal environment for parallel computing. It runs on more platforms and is more portable than PVM due to being more strict in the way it is used. However, it can only be run on one architecture simultaneously and the same program must be spawned for each process or CPU. MPI is described in [11]. This version of the *ParInt Prototype* does not implement any MPI code.

**nCUBE2**

The nCUBE2 supercomputer uses similar command to MPI's calling interface, but the nCUBE2 is also not as strict as MPI and allows for some PVM-like freedom. The nCUBE2 is limited in its age, especially regarding its pre-ANSI C compiler and lack of C++ compiler. The code written for it is not portable, without a great deal of effort. This version of the *ParInt Prototype* does not implement any nCUBE2 code.

**Remarks**

Even though the *ParInt Prototype* does not implement either MPI or nCUBE2 code, the *ParInt Project* does have an extensive base of code for both of these environments. Both will be merged into the *ParInt Prototype* in the future.
Parallel Numeric Integration

The parallel portions of ParInt are mostly based around adaptive integration techniques. Basically, an integrand $I$ is computed from a function $f(\vec{x})$ over a hyper-rectangular region $\mathcal{D}$ in $\mathbb{R}^n$ via:

$$ I = \int_{\mathcal{D}} f(\vec{x}) d\vec{x}. $$

The numerical approximation $Q$, determined from ParInt's computation, attempts to satisfy ParInt’s error estimate $E_a$ with

$$ |I - Q| \leq E_a \leq \text{Max}\{\varepsilon_a, \varepsilon_r |I|\} $$

where $\varepsilon_a$ and $\varepsilon_r$ are absolute and relative tolerances respectively.

The algorithm starts with an initial estimate of the integral and selects subregion with the highest error estimate for further integration (get_region). The selected subregion is then divided in half along the axis with the highest error and integrated again (process_regions()). A better estimate of the integral and error are now known. These two new subregions now replace the single region on the min-heap (put_regions()). The integration process occurs until some acceptance criteria is met, generally $E_a \leq \text{Max}\{\varepsilon_a, \varepsilon_r |I|\}$. Figures 1 and 2 show the general algorithm.

In a parallel version, our criteria changes. Each node computes a portion of the integral and it need only meet an error proportional to its initial region $R$. So, the node meets a local estimate $e_a \leq \text{Max}\{\varepsilon_a, \varepsilon_r |I|\} \times \text{size}(R)/\text{size}(\mathcal{D})$. The
adaptive_integration_algorithm()
    initialize;
    while(acceptance criterion is not satisfied)
        partition();
        update_globals();

    Figure 1. Adaptive Integration Algorithm.

global error criteria then become $\sum R e_a \leq \varepsilon$. Other acceptance criterion include a maximum number of function evaluations (called *Granularity*), maximum processing time, maximum loop iterations (calls to partition() * NS_Threshold).\(^1\) Each of these have both the global and local versions.

partition()
    process_msg_queue();
    for(i=0; i<NS_Threshold; i++) {
        get_region(); /* select next region */
        process_regions(); /* subdivide and integrate subregions */
        put_regions(); /* add new regions to the pool */
        update_locals(); } /* update local results and flag abnormalities */
    load_balance();

    Figure 2. Partitioning Loop.

During our iteration of the while() loop, partition() will perform a for() loop *NS_Threshold* times. This is the Node Send Threshold which is how often to send data to the global database. This is here to cut down on communication costs which speeds up the algorithm.

\(^1\)Since an integral estimation method may make many function evaluations at a time, Granularity and maximum loop iterations are very different things.
At the end of \texttt{partition()}, \texttt{load_balance()} is called. This procedure determines if the local error is much lower than another node's local error. If so, then the remote node gives the local node some of its regions to integrate. This, in theory, should help the algorithm to concentrate on the regions with the highest error.

At the conclusion of the adaptive algorithm, \( q \approx I \) and \( E_a \to 0 \).

All source code for the parallel portion of the \textit{ParInt Prototype} is shown in Appendix F starting on page 195.

Parallel Implementation Results

As published previously[1, 2, 3, 4, 5, 6, 8], we have shown great results with these adaptive techniques. For example, in [1] the computation of
\[
\int_0^2 \int_0^1 \int_0^1 \int_0^1 \frac{x_3^2 x_4 \exp(x_3 x_4)}{(x_1 + x_2 + 1)^2} \, dx_1 \, dx_2 \, dx_3 \, dx_4
\]
yields excellent speedup results as shown in Table 1. One process per workstation was used. "Spawn" refers to the time it takes PVM to initialize and run the program on all nodes. It is separated out as for a more complex integral, the one or two extra seconds would have no bearing on the longer computation times.

Figure 3 shows the results of a more complex function that comes from the use of logistic regression to estimate the parameters for a birth weight model.[5] As can be seen, spawn time is insignificant in the timings of the function. This also shows that different \( E_r \) will affect the timings.
Table 1

Timings on PVM

<table>
<thead>
<tr>
<th># Workstations</th>
<th>Total Time</th>
<th>Time w/o Spawn</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>9.27</td>
<td>8.70</td>
</tr>
<tr>
<td>2</td>
<td>4.36</td>
<td>3.68</td>
</tr>
<tr>
<td>4</td>
<td>3.26</td>
<td>2.40</td>
</tr>
<tr>
<td>8</td>
<td>2.91</td>
<td>2.15</td>
</tr>
<tr>
<td>16</td>
<td>2.29</td>
<td>0.17</td>
</tr>
</tbody>
</table>

Something interesting was discovered when using PVM or MPI on a small number of high usage, time shared CPUs or machines. The operating system gives each process an equal time slice of the CPU. However, if multiple PVM/MPI processes are spawn per CPU, say 4 to 1, the operating system will give each process an equal share. Thus, the parallel program will get more of the CPU time than other processes. This may seem "obvious", but it was not thought about before the experiments. This simple fact allows parallel programs to actually work faster on heavily loaded systems.
Figure 3. Speedup With Various Processes Quantities & Error Tolerances.
CHAPTER III

PROGRAMMING OF PARINT

ParInt is designed to meet several important criteria: (a) user and ParAdmin installation friendly; (b) easy to learn and use interface; (c) run on as many platforms as possible; and (d) parallel implementation independent.

Installation

ParInt is designed to be easily installed. Since special files in the user’s home directory are required to use ParInt, an installation program called parinstall must be executed before the initial running of ParInt. This program checks to see if previously installed copies of PVM 3.x or ParInt exist (other systems like MPI do not have special user files) and if environment variables, like $PVM_ROOT, are properly set.

parinstall determines if the user’s ~/pvm3 directory was properly configured. Any differences in the ideal setup along with suggested command-line corrections are returned to the user. Automatically performing the corrections is not done because it is generally impolite to change something without asking and the user may have files inside of the directories that would be modified or erased.
parinstall then checks to see if the ~/ParInt directory exists and is configured correctly. After installation, the user need only type ~/ParInt/ParInt to run the program (it can be put on the $PATH of course). Thus, for a user that has never used a parallel system, the installation is one simple step.

The system wide installation is slightly more complicated. It is the responsibility of the ParAdmin to install and configure the required versions of PVM, MPI, etc. The installation method is to untar the archive and run an interactive script which sets up ParInt. The SysAdmin can optionally set the environment variables in the system wide login files. Both installation programs, written in Bourne Shell, are listed on page 61. The problem with “universal” parallel programs is that someplace, somewhere it will not quite work. This will also be the case with ParInt. The current implementation is designed around the client/servers which can immediately execute a program and return the results for each parallel program execution. If a batch processing or job submission system were used, ParInt will need to be modified.

User Interface

The vast majority of today’s user interfaces consist of “intuitive” graphical pointy-clicky windows. So, to fit in, the ParInt GUI was implemented in the same fashion.

The ParInt GUI consists of a main window and many other windows. In
general, each menu selection spawns a new window.

Main Window

![The ParInt Project](image)

Figure 4. The ParInt Main Window.

The main window of *ParInt* (shown in Figure 4) consists of a menu bar, status line, and some ego-enhancing filler. The status line changes to reflect messages that various portions of the program may want to send to the user. As the user selects the pull down menus, other functions and windows are spawned from the main window.

The menu bar of the window is created with a special menu package which simplifies creation and improves readability of the source code. Sample code in [14] was modified for multi-lingual support and added to *ParInt*. The modified
code is shown in on page 188. The remainder of the window was designed with standard Tcl/Tk commands.

Function Window

Figure 5. The Choose Function Window.
The choose function window (Figure 5) appears when the option is picked from the main window. The purpose is to select the file containing the integrand(s). *ParInt* reads a special input file called the *list file* (see Appendix A) and displays the list of groups in the left box. When the user selects a group, its corresponding *group file* (described in Appendix B) is read and all of the possible functions (or more correctly, files) are displayed in the right box. The user can then select a specific integrand function. As the user clicks on a group or function, some information on the selection is displayed in the lower two boxes ridged within the window. The "Update" button will re-read the list file in case it has been changed during the program's run.

An improved method for loading and selecting functions is described on page 35. Reasons for this conclusion are also presented.

**Compile Window**

All output and errors that happen during a compile of the parallel portion of *ParInt* are displayed in the Compile Window (Figure 6). Essentially, the user clicks on "Run" (after selecting a function) and *ParInt* compiles everything for the parallel machines. In a perfect world, this window would not exist, as everything would compile nicely without the need for user intervention. Since this is not the case, the user can watch what happens and notice errors in user created functions or if some components of *ParInt* are non-existent due to a system snafu.
Integration Window

After compiling, the user selects the Integration Window and simply clicks "Run". The results (and any errors) are displayed in the window (Figure 7). Some of the buttons allow the user to save the window's data into a file, clear the data from the window, and produce the data in an ultra-compact, one line format. The later is especially useful when comparing runs of the function with various thresholds, number of processors, etc. It also loads nicely into other programs, spreadsheets, or gnuplot.
Number of Processes Window

In the Number of Processes Window (Figure 8), the user selects the number of parallel CPUs, processes, or nodes (depending on the parallel environment) to run the parallel program on. Just by moving the slider and hitting the Integrate Window’s “Run” button, the integral will be computed with a different number of CPUs. This window will be modified in the next version of ParInt as described on page 39.
Variable Window

This window (Figure 9) shows the value of some internal ParInt variables. Each variable is updated in the window as it changes anywhere inside of ParInt.

The Variable Window was created mainly as an informational and debugging aid for the programmer to track data flow. However, the window can also be used by the ParAdmin to administer remote trouble-shooting with a user. ParInt has the ability to show any of the $Config configuration variables as selected by the ParAdmin. Modification of $Config in gui/System.tcl will reflect the changes.
Help System

*ParInt* has a small internal Help System (shown in Figure 10) which is brought up by clicking Help:HelpSystem. All of the help files are in universal HTML format, the standard of the World Wide Web. Thus, users can also read the files using *Netscape, Lynx*, or other browser of their liking. The help files are easily customizable and need not even be located on-site. ParAdmins can choose to let users access http://www.cs.wmich.edu/~parint/help instead.

The *ParInt* GUI’s internal help viewer does not allow for URL specifications or any off-site links. It also does not support many of the myriad of HTML
ParInt is a package that lets you integrate any function in parallel.

For more help, click on:
- Introduction

Execution Platforms

In deciding which style of user interface to implement, many alternatives were considered. The Windows and Macintosh interfaces were briefly considered, but they were not implemented on machines with ParInt's raison d'être: par-
allelization. Taking this into account, Unix based computers were chosen to be the platform(s) for ParInt. The Unix GUI is the nearly-universal X-windows user interface. Yet getting each variant of Unix to compile an X program without trouble might take a great deal of work. So, instead of the C/C++ X-windows based GUI, the Tcl/Tk language was chosen for the pointy-clicky interface. Tcl/Tk has many advantages over X:

1. Development of the GUI is totally independent of the parallel coding.
2. Tcl/Tk has been ported to nearly all flavors of Unix.
3. Various parallel components can be added with very little or no change in the GUI code.
4. Tcl/Tk even runs on Windows and Macintosh, so future ports of ParInt to these platforms may be possible.

Unfortunately, there exist some disadvantages:

1. Tcl/Tk is interpreted, not compiled; so it is much slower than X.
2. The user's system must have the Tcl/Tk interpreter (i.e., wish) installed on the system, adding another piece of software and the concomitant management problems.
3. Code which a compiler would flag as an error might get into the production release of ParInt. Tcl/Tk's interpreter may miss the invalid code and ParInt would crash during a program run.
4. Different versions of Tcl/Tk have slightly different reactions to the
The source code for the *ParInt GUI* is shown in Appendix E.

ParallelPlatforms

A pure C/C++ version of *ParInt* would require separate *fork()* calls for each variety of parallel program load. Hundreds of *#ifdefs* and *#elifs* would be needed to determine if the user's site has one or more of MPI, PVM, etc installed. Additionally, some systems require users to submit parallel jobs as a batch and/or do not allow logins or compiles on the parallel systems. So, programs must be cross-compiled elsewhere, uploaded, and only then, run on the parallel system.

Here at Western Michigan University, the nCUBE2 is an example of a system which does not allow logins or compiles. Users are not allowed to log into the supercomputer. In addition, *ParInt* will not run on the front end computer which supplies the nCUBE2 with the binary executables. So, a complex method of remote shell calls (*rsh*) and *Makefiles* must be used to compile and execute the program. For special situations as described here, the ParAdmin must modify *ParInt* specifically for these systems.

Hence, *ParInt* requires totally separate programs for each platform: one specifically for the GUI (written in Tcl/Tk) and one for each parallel flavor in (written in C/C++ or Fortran). A separate code base must be kept for MPI and PVM, in addition to the required separate binaries for systems like the nCUBE2.
Whereas MPI and PVM may be able to share binaries for things like integral rules and functions, the nCUBE2 cannot.

ParInt is designed to be modular where components can be cut and paste as we please. Each component is usually located in a different source file. With this, it is only a matter of different linker calls to use the parts that are desired for that program run. The programs basically consist of a main routine, heap and load balancing systems, integrand function definitions, and integration methods. Not all components can be used in conjunction with each other; most components have some sort of optionally definable parameters which can be set at run time, not compile time.

The Makefiles are shown in Appendix C.

Directory Layout

ParInt has has two separate directory trees: that of the individual user (Figure 11) and that of the shared main system (Figure 12). The main system contains many nested area to separate the proprietary code of each parallel environment from that of the universal code.

The user's area has a directory for the function object files for each platform and a universal area where functions are stored, processed, and compiled. Because of this dual directory structure, more than one user at a time is able to run ParInt.
This section will deal with a few selected procedures of the ParInt program.

PVM Procedures

*ParInt* works by compiling the desired functions into an executable for the target platform. Different integration rules may also be coded (see page 39).
As approximations to integrals are computed, an error estimate is generated. Since one objective of our techniques is to minimize an integral’s estimated error, the most logical method for sorting the errors is a heap. Each node of the heap is structured as in Figure 13.

This code is based on [7] and has been modified for dynamic memory allocation and better modularity. The added procedures RegionNew and RegionFree are used to allocate and free the memory needed to store each region on the heap.
node = {
    node  *p_nptr;  // The parent node
    node  *lc_nptr; // The left child
    node  *rc_nptr; // The right child
    datum_type  *rgn;  // Region to integrate
}

rgn = {
    double  *center;  // Center coordinates of hyper-rectangle
    double  *width;  // Half-widths for each dimension
    double  vol;  // Volume of the hyper-rectangle
    double  res;  // Estimated Result over the hyper-rectangle
    double  err;  // Estimated Error
    int  divax  // The axis with the highest error
    int  dimensions;  // Number of dimensions in the function
}

Figure 13. Structure of ParInt’s Heap.

These procedures and the whole rgn structure need to be moved from the heap.c file into a set of new file possibly called hyper-rect-region.c for example. The current method requires that an entirely different heap module be used for the current hyper-rectangular regions and the future triangle, trapezoidal, and simplex regions. Using this new heap/region procedural coding, other types of heaps, like distributed heaps, can be coded without worry of what the actual region looks like.

The only major change in the heap code would be adding a double error to the node definition and instead of comparing lc_nptr->rgn->err to rc_nptr->rgn->err, the code would use lc_nptr->error to rc_nptr->error.
The added node->error would solve two problems:

1. The new error is a more abstract concept which is set upon creation of the heap node and may not necessarily exist in the region's data structure. This would be useful for vector functions where the desired error is actually the maximum of the errors for the list of functions.

2. The added processing cost of an additional dereference is removed by allocating the extra eight bytes (assuming a double is 64 bits, of course) per node on the heap. For a 100,000 iterations of an integral, only three-quarter megabytes extra memory total from all processes/CPUs would be needed.

Another ideal change would be to combine the add_heap and replace_root functions. Currently, the integration loop will grab the root node, process it, replace it with a new node, and add another leaf to the tree. A better solution is to simply call a new procedure, add_node, which will replace the root or add a new leaf as required.

Finally, better names for the node structure's internal components would help code understanding. lc_nptr just is not too intuitive. A better choice might be left, since we know its a pointer due to a heap's nature.

**Command Line Parsing**

Since the parallel portion is simply a separate program called from the GUI, some sort of interface needs for the programs to communicate. The most
simple method is to pass the data on the command line. A large if-else tree parses options like

1. \texttt{--NS\_Threshold value} Send the controlling process integral estimate, error estimate, quantity of function evaluations, etc every \textit{value} iterations.

2. \texttt{--Percent\_Threshold value} Send above data when the error estimate changes \textit{value} percent (or NS==1000). \textit{value} is an integer.

3. \texttt{--Nodes number} This specifies the \textit{number} of processes, nodes, or CPUs to run the parallel program on.

4. \texttt{--Slaves number} Same as \texttt{--Nodes}

5. \texttt{--Compact} Display the output of the program in a handy, one-line format.

6. \texttt{--Non-Compact} Let the output display in as many lines as it wishes.

7. \texttt{--Limits x1 x2 y1 y2 ...} Limits of integration in order of dim1\_start, dim1\_finish, dim2\_start, etc. All limits that an integral requires must be present or the program may crash.

The \texttt{StringToDouble} procedure emulates the standard C library function \texttt{atoi} with nicer features. Where as \texttt{atoi} only converts strings like "1.69e42" to a pure number, \texttt{StringToDouble} allows for string like "pi/2" to be converted to 1.57. Currently supported are \pi (pi), \infty (inf), and the natural log \e (ex, to not be confused with the computer's 1.2e5 notation), and any of these can be placed with an additional factor, addend, divisor, or minuted. Nice expansions of
the `StringToDouble` procedure's ideas would be a full mathematical parser. The code for this has probably already been written by someone; so scouring the net may give the solution to enhancing the code.

**Tcl/Tk Procedures**

**Compilation Procedures**

The `ParInt GUI` must read and parse user function files so that the rest of `ParInt`'s parallel programs can compile and link the functions into the executable. `CreateSubstitutedFile` starts this by opening the user's function file and calling the appropriate C/C++ or Fortran parser. Within `MakeCFile` or `MakeFortranFile`, `#defines` are added to convert the names that the user chose for the function into a universal `ParInt` name. In addition, the variable `Funct_Calls` is placed in this function in order to tally the number of calls of the user's function. It is better to place the increment within the function so that the integration rules and methods can be written without worry of forgetting to properly count. Using a series of regular expressions, the file is parsed and a new `.c` or `.f` file is produced.

The actual compile is done via the standard Unix `make` command. The `ParInt GUI` calls `make` and displays the results to the user within a window. Pipes are opened between standard in, out, and error to capture the data.
The code for gui/Compile.tcl starts on page 80.

Parallel Execution Procedures

Since the parallel portions run as a separate program, the ParInt GUI has an interface to display the results of the runs. Like the compilation procedures, pipes are used to capture the data. The run procedures also have buttons, like Compact, which cause different output upon a run. Selecting of the buttons causes the procedure CreateRunCommandLine to construct a command line to run the parallel binary. The option for --Nodes numcpus is always included and, for example, selecting Compact adds --compact to the command line.

The parallel execution window does need a Compile and Run button to make the usage of ParInt one step. A batch mode is another needed feature. Source code is shown on page 141.

Some Miscellaneous Procedures

The pathexpand procedure takes a given directory ($dir) and filename and returns a fully qualified path to the file. While this could be as simple as just concatenating the two parameters, pathexpand goes further by allowing for the filename to contain simple variables. So, if a filename begins with a $, then the path to the file would be the variable’s contents, slash, filename. If filename begins with a slash, then a full path from the root directory is assumed. Otherwise, $dir
is used as the starting directory, followed by the filename. $dir is a variable lookup in the table $Dir($dir).
CHAPTER IV

THE FUTURE

This ParInt Prototype is only the start of a much larger project. Presented in this chapter are some future ideas, thoughts, and blather; along with the traditional “to do list”. It should be noted that earlier chapters may contain additional enhancements, especially in the sections on procedures.

Proverbial “ToDoList”

Improved Function Loading

With the current function selection method (shown on page 17, very few files need to be read in by ParInt in order to display the list for the user to select from. This is nice if the user only intends to edit the C function files or only use the sample functions. The user is forced to edit the list file and/or group files if any functions are to be added.

A new function loader should have the list file removed. Instead, the contents of the file will generated on the fly by reading in the group files from a search path containing ParInt’s function directory and the user’s function directory. The new group files would only consist of a function file list and some information on
this entire group of functions. The information on each function would now be stored in the C file within the initial comment header.

I envision the beginning of the function's source file as shown in Figure 14. Notice that everything can be multi-line and that ParInt will ignore the word-wrap when appropriate. A file need not have any of these options. Some of the proposed options (along with suggested names) of the new header file might include:

1. **PARASCII** is an ISO-LATIN representation of the function. Depending on the function, this may be unreadable or untypeable.

2. **PARBEGIN** tells ParInt to start looking for special parse-able lines in the C source file.

3. **PARDESC** is the ISO-LATIN description of the function.

4. **PAREND** indicates the end of the special headers and that parint should ignore the rest of the file (at least for this portion of ParInt's parsing).

5. **PARTEX** specifies a fully qualified \( \text{\LaTeX} \) string assuming that the equation or displaymath environment is in effect.

6. **PARVECTOR** tells ParInt the file contains a vector function and to not allow the user to compile in a non-vector integration rule.

A "groupless" or "other" group selection would be available. This would list out all C files and display the internal information as described above. This way, a user need not edit the group file at all.
Figure 14. Multi-Purpose C Header File.

Other options could include HTML display (using version 3.2's new math mark up tags), additional function information parameters, etc. External TeX or .dvi files could be specified with PARTEXFILE or PARDVIFILE for the display of more complex functions. Spawning xdvii (or even ghostview for PARPOSTSCRIPT) could be another possibility.

Writing a parser in Tcl/Tk for the above comment would not be too hard. Debugging and adding immense amounts of user error checking would be. The Tcl/Tk code for this could be shared between the C/C++ and Fortran parsing procedures with minimal changes. A proposed parser is shown in Figure 15.

The first while loop reads until PARBEGIN is found. If it is never found, the procedure exits upon finding the end of the file. Otherwise, the procedure
Figure 15. Pseudo-Prototype for C File Parser.

enters the second while loop. In here, Buffer is filled with all of the text between command modes. If a special parameter is set, then we take advantage of Tcl/Tk’s semantics and set \$T (dollar T) to the contents of Buffer. The dollar syntax as used above will replace the \$T with contents of T, effectively saying something like “set PARDESC = Buffer” if the value of T is “PARDESC”. This little trick
saves us the trouble of writing unique cases for each command parameter.

The strip_crud procedure would remove all fluff like " * " as generated by pretty-printer programs such as indent. It would also remove any whitespace, "//", "*/", or Fortranisms like "C ". Using the Tcl/Tk regexp and trim commands (along with some error checking ifs) would accomplish this.

Different Integration Rules

Because of the modular programming techniques within the parallel portions of ParInt, future versions will allow for various integration rules. If implemented properly, these integration rules will be as easy to use as functions currently are.

New Number of Processes

The slider used to select the number of nodes to run the parallel program on is limiting. Only selections of $2^0$, $2^1$, ..., $2^{10}$ are allowed. The limitation was imposed due to the abilities of the original region division algorithm. Future division algorithms will not have this limitation, so a new window needs to be devised.

A future window may still have a slider, in order to keep the easy of use, but also have an entry box so that the user may type the desired quantity of nodes. An even better slider to adjust its limits to reflect the division algorithm
used, like $2^0$ to $2^{10}$ for the hyper-rectangular division or a linear division with 1 to $n$. Commonly used values could be saved and check boxes could be clicked to select them. This portion of the ParInt GUI can be enhanced in many ways, none of which require too much work.

Enhancements

Report Generation

An ideal environment for ParInt would be to set up a list of functions to integrate (with all parameters such as number of processors), run the list in a batch mode, and look at the results in an automatically generated $\LaTeX$ or HTML file.

Template files created by the user (or some sample files supplied with ParInt) would be used as the basis for the report. Using Tcl/Tk's `resub` command, certain special phrases would be replaced with the appropriated data. Possible sample files are shown in Figures 16 and 17 and the resulting appearance is shown in Figure 18. All commands described on page 36 are able to be placed in the template file along with new additional commands like PARFUNCTION which displays the ASCII, $\TeX$, or HTML function depending on what was given in the C/C++ file and what type of report is generated. PARCPUS shows the number of processes, PARABSERROR shows the absolute integration error, and
many other obvious options could also exist. Special beginning-of-document and end-of-document files would also be needed; however, these would be independent of the template files. So, the user could mix and match beginning/ending files with the various body templates.

```xml
<center>PARFUNCTION</center>
<table width=100%
<tr>
  <td>Processes</td>
  <td>Time</td>
  <td>Error</td>
</tr>
<tr>
  <td>PARCPUS</td>
  <td>PARTOTALTIME</td>
  <td>PARABSERROR</td>
</tr>
</table>
```

Figure 16. Template for Auto-Generated HTML File.

```latex
\begin{center}PARFUNCTION\end{center}
\begin{tabularx}{\textwidth}{|X|X|X|}
\hline
Processes & Time & Error \\ \hline
PARCPUS & PARTOTALTIME & PARABSERROR \\ \hline
\end{tabularx}
```

Figure 17. Template for Auto-Generated \LaTeX\ File.
\int x_0 + (x_1^2 + 2x_2^3)^{2/4} \, dx

<table>
<thead>
<tr>
<th>Processes</th>
<th>Time</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>42.69666</td>
<td>4.2e-12</td>
</tr>
</tbody>
</table>

Figure 18. Resulting Appearance of Either Template.

**Status Box**

Currently, a single line of text exists at the bottom of the main window which displays information as each procedure desires. A call to the procedure `StatusLine` causes all passed parameters to be displayed on this line. A better method would be to implement a “Status Box”. This box would be a simple scrolling text box, as used in many other parts of the program, that displays a log of all status messages that have occurred. `StatusLine` could be modified to take two parameters, one for the single line in the main window and the other as text to show in the box. Using this new status ideal, the user may get a better idea of what is going on. A “Walk Through” or “Coaching Tutorial” could also be implemented using the Status Box. The entire window might have “Dismiss”, “Clear”, and “Save” buttons, where the later could be used for sending troubleshooting email to the ParAdmin.
Multiple Languages

The current version of the *ParInt GUI* supports different languages. Each menu, title bar, and message can be displayed in something besides English. All that needs to be done is the translation. Even non-ISO-LATIN languages like Cyrillic, Japanese, or even Klingon can be supported. The *ParInt GUI* has built in command line switches for choosing languages, which in turn determines which help directory and `gui/Lang.tcl` file needs to be loaded. Full instructions on translation are in the `gui/Lang.tcl` source code, list on page 117.

Rewrite in Java

The Tcl/Tk GUI is very portable and will run on nearly any platform (including Windows and Macintosh with some work). The C portion is also highly portable, but still may not run on all platforms nicely due to slight compiler, system, and other incompatibilities. The computer language *Java*, on the other hand, has no “dialects” that plague C (and to some degree, Tcl/Tk) and Java runs on most platforms. Unfortunately, Java will probably not be ported to older systems like the nCUBE2. Java also lacks the ability to open files (not true in the next version) and communicate via a PVM or MPI like method. Why then am I professing the glories of Java?

For one, Java’s binary executable files run on all platforms. So, in a mixed
environment with supercomputers, PC’s, and workstations; only one set of files is needed. For a learning environment where our goal is the comparison of speedup via different parallel computation techniques (not necessarily a goal of pure speed), this is a great addition.

Java is also modular in not only the source code, like most all languages, but in its object files. So, you can pick and choose any binary portion of the program without any need for recompiling. If Java ParInt were programmed correctly, no compiling would be necessary at all (except for new functions). This is something that would be impossible under ParInt’s current setup. All multi-platform support would have to be eliminated to accomplish dynamic linking at runtime.

The GUI functions of Java are just as universal as Tcl/Tk. So, no loss of functionality would exist. In fact, I believe the functions are nicer to program and the resulting display is more pleasant to view on the screen.

Alpha versions of PVM support in Java do exist now. A look at http://www.oasis.leo.org/java/classes/parallel/00-index.html will show two experimental libraries: JPVM and JavaPVM. So, a future Java version of ParInt may not be such a far-fetched idea.

There are some disadvantages to Java. Namely, it is slow. But then again, so is Tcl/Tk. Better implementations have come about speeding up Java to much levels faster than Tcl/Tk. Yet, Java is still not to the speed of C/C++. It may
be many years before this level is reached.

For the moment, keeping all of ParInt’s code in C/C++/Tcl/Tk is the most practical action. Decent C++ to Java code converters exist now and will only get better in the future. The same goes for Java compilers and even the soon to be released Java Operating System and Java Microprocessors. Someone might also release a Tcl/Tk to Java converter. The wisest idea is to take a wait and see attitude, and if demand warrants it, ParInt’s eventual conversion to Java could take place.
Appendix A

The List File
This file is \sim user/ParInt/func/Group.

# The Group List File
#
# This file contains each group of integrals that you wish to
# compute. Basically, you just give it the group file, a
# variable name, and a small (optional) description. All of
# these items must occur on a single line for each group. Your
# only requirement is that the variable names must each be unique,
# otherwise, you will not be able to use one that are
# redundantly named. Only the description may contain spaces;
# all other spaces and/or tabs are used to separate the fields.
# A line beginning with a "#" is ignored, so use this to
# temporarily disable group if you desire to.
#
# If a filename begins with a "/", then the path to that file is
# taken as being absolute. Thus,
# "/home/bobdobbs/ParInt/func/my.group" would be literally that
# file. If a filename begins with a "$", then a variable
# substitution occurs for that variable. So,
# "$ParIntHome/func/a.group" would become
# "(path_to_ParInt)/func/a.group". Otherwise, if a filename has
# no "$" or "/", then the filename becomes
# ~/ParInt/func/filename. If a file cannot be found, an error
# will be reported and the group will not show up in the program.
#
# The only valid variable at this time is ParIntHome. If someone
# can think of more or better variables, then add them.
# Error checking is not as good as you would think :)

# Eventually, this file will be a pointy-clicky setup.

# Therefore, the format is:
GROUP = var file desc

# Built-in group files.
#
GROUP = drvd $ParIntHome/func/Drvd.group The DeRidder-Van Dooren Group
group = easy $ParIntHome/func/Easy.group A set of simple integrals

# Note that deleting the drvd group will cause the program to
# produce errors at times. Thus, leave it in. In a future
# version, this requirement will be eliminated.

# A sample user file. Notice that the line may continue beyond
# 80 columns. But, you cannot add an extra line within the group
# definition. Be careful if your text editor auto-wraps.
#
group = user user.group Some functions that the user (i.e., you) can define

# You can add or delete a group as you please. (except drvd)
Appendix B

Group Files
The group file is named after its group, with an extension of `.group`. It can be located in `~user/ParInt/func` or `$ParIntHome/func`.

<table>
<thead>
<tr>
<th>Func = myfunc01</th>
<th>myfunc01.c</th>
<th>My first function</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Func = drvd01</th>
<th>$ParIntHome/func/drvd01.c</th>
<th>Periodic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Func = drvd02</td>
<td>$ParIntHome/func/drvd02.c</td>
<td></td>
</tr>
<tr>
<td>Func = drvd03</td>
<td>$ParIntHome/func/drvd03.c</td>
<td></td>
</tr>
<tr>
<td>Func = drvd04</td>
<td>$ParIntHome/func/drvd04.c</td>
<td></td>
</tr>
<tr>
<td>Func = drvd05</td>
<td>$ParIntHome/func/drvd05.c</td>
<td></td>
</tr>
<tr>
<td>Func = drvd06</td>
<td>$ParIntHome/func/drvd06.c</td>
<td></td>
</tr>
<tr>
<td>Func = drvd07</td>
<td>$ParIntHome/func/drvd07.c</td>
<td>Singular</td>
</tr>
<tr>
<td>Func = drvd08</td>
<td>$ParIntHome/func/drvd08.c</td>
<td></td>
</tr>
</tbody>
</table>
Func = drvd09 $ParIntHome/func/drvd09.c
Func = drvd10 $ParIntHome/func/drvd10.c

$ParIntHome/func/Easy.group

1 func = easy01 $ParIntHome/func/easy01.c Simple Linear
func = easy02 $ParIntHome/func/easy02.c Quadratic
func = easy03 $ParIntHome/func/easy03.c Cubic
Appendix C

Makefiles
# da makefile

# what to print and where to print the source code to
PRINTME = \
    gui/About.tcl \
    gui/Compile.tcl \
    gui/Config.tcl \
    gui/C_Stuff.tcl \
    gui/Debug.tcl \
    gui/DependantVar.tcl \
    gui/Func.tcl \
    gui/Help.tcl \
    gui/Info.tcl \
    gui/Lang.tcl \
    gui/Limits.tcl \
    gui/Main.tcl \
    gui/MessageBox.tcl \
    gui/NumProc.tcl \
    gui/PVM.tcl \
    gui/Run.tcl \
    gui/StatusLine.tcl \
    gui/Text Utils.tcl \
    gui/Var.tcl \
    gui/miscfile.tcl \
    gui/path.tcl \

ParInt \n30 Makefile

# files we don't need
#gui/logo.tcl \n#gui/Fortran_Stuff.tcl \n
PRINTER = dunbar

#ParIntHome = /home/res/parint/ParInt
40 ParIntHome = /home/res/parint/ParInt.EXPERIMENTAL

# PVMPlatform = SUN4SOL2
ECHO = /bin/echo
CC = gcc
AIMK = $(HOME)/pvm3/lib/aimk

# this desperately needs to be changed!!!!
INCLUDE = -I$(ParIntHome)/func -I$(ParIntHome)/pvm

50 CFLAGS = $(INCLUDE)

default:
    $(ECHO) You did not set a distributed system to integrate with.
    return -1

run.pvm:
pvm: $(HOME)/ParInt/func/user.o
   # cp $(HOME)/ParInt/func/user.o $(HOME)/ParInt/pvm/$(PVMPlatform)
   (cd $(ParIntHome)/pvm ; $(AIMK) )
   echo All done. Now you may integrate.

$(HOME)/ParInt/func/func.o: $(HOME)/ParInt/func/func.c
   (cd $(HOME)/ParInt/func ; $(CC) $(CFLAGS) -c func.c)

$(HOME)/ParInt/func/user.o: $(HOME)/ParInt/func/user.c
   echo $(HOME)
   echo before compile of user.c
   (cd $(HOME)/ParInt/func; $(CC) $(CFLAGS) -c user.c )
   echo after compile of user.c

#func/user.o: func/user.h func/userfile
# (cd func; cat user.h 'cat userfile'.c > user.c )
#(cd func; $(CC) -c user.c )

file:
   pr -f $(PRINTME) > PRINTERFILE
print:
   pr -f $(PRINTME) | lpr -P$(PRINTER)
pvm/Makefile.aimk

1

# ParInt
#
# Makefile.aimk for PVM programs.
#
# Set PVM_ROOT to the path where PVM includes and libraries are installed.
# Set PVM_ARCH to your architecture type (SUN4, HP9K, RS6K, SGI, etc.)
# Set ARCHLIB to any special libs needed on PVM_ARCH (-lrpc, -lssocket, etc.)
# otherwise leave ARCHLIB blank

10

# PVM_ARCH and ARCHLIB are set for you if you use "$PVM_ROOT/lib/aimk"
# instead of "make".
#
# ParInt does NOT a $PVM_ARCH directory below this one and will cd to it
# before invoking make like standard aimk does. You CANNOT build
# in parallel on different arches because that would imply all other compents
# of ParInt could be done the same way, which, if you do not have PVM, would
# be handled without aimk. So, we remove this aspect of PVM's features.
#
20

# the aimk call must be invoked from this, the current, directory.
#
# call 'aimk setup' to configure everything if you are the ParAdmin
#

ParIntHome = /home/res/parint/ParInt.EXPERIMENTAL/pvm
SDIR = ..

# The user's binary dir
BDIR = $(HOME)/pvm3/bin

# BDIR = $(SDIR)/../bin

# user's executing dir
XDIR = $(BDIR)/$(PVM_ARCH)

# the user's function dir
FDIR = $(HOME)/ParInt/func

# parint's dir for files that parint compiles (and not user)

PDIR = .

CC = gcc
OPTIONS = -O
CFLAGS = $(OPTIONS) -I$(PVM_ROOT)/include $(ARCHCFLAGS)
LIBS = -lpvm3 $(ARCHLIB)
GLIBS = -lgpvm3

# some included files dirs (used only for user's function) do not use relative paths!!
IDIR = -I$(ParIntHome/pvm)

F77 = f77
FFLAGS = -g
FLIBS = -lfpvm3

LFLAGS = -L$(PVM_ROOT)/lib/$(PVM_ARCH)

ParentObjs = $(PDIR)/pvm_parent.o $(PDIR)/commandline.o
ChildObjs = $(PDIR)/pvm_child.o $(PDIR)/gnzrul.o $(PDIR)/heap.o

FunctionObjs = $(FDIR)/user.o $(PDIR)/function.o $(PDIR)/hyper_rect.o $(PDIR)/heap.o

UserFunction = $(FDIR)/user.o

Parent = $(XDIR)/parent
Child = $(XDIR)/child

all: $(XDIR) $(Parent) $(Child)

setup: $(PDIR)/pvm_parent.o $(PDIR)/pvm_child.o $(PDIR)/heap.o 
 $(PDIR)/gnzrul.o $(PDIR)/commandline.o $(PDIR)/hyper_rect.o 
 $(PDIR)/function.o

$(Parent): $(ParentObjs) $(FunctionObjs)
 $(CC) $(CFLAGS) -o $(Parent) 
 $(ParentObjs) $(FunctionObjs) 
 $(LFLAGS) -lm $(LIBS)

$(Child): $(ChildObjs) $(FunctionObjs)
 $(CC) $(CFLAGS) -o $(Child) 
 $(ChildObjs) $(FunctionObjs) 
 $(LFLAGS) -lm $(LIBS)

$(PDIR)/pvm_parent.o: $(SDIR)/pvm_parent.c $(SDIR)/pvm_slave.h
 $(CC) $(CFLAGS) -DPRINT -c $(SDIR)/pvm_parent.c

$(PDIR)/pvm_child.o: $(SDIR)/pvm_child.c $(SDIR)/pvm_slave.h
 $(CC) $(CFLAGS) -c $(SDIR)/pvm_child.c

$(PDIR)/heap.o: $(SDIR)/heap.c
\begin{verbatim}
$(CC) $(CFLAGS) -c $(SDIR)/heap.c
$(PDIR)/hyper_rect.o: $(SDIR)/hyper_rect.c
  $(CC) $(CFLAGS) -c $(SDIR)/hyper_rect.c
$(PDIR)/commandline.o: $(SDIR)/commandline.c
  $(CC) $(CFLAGS) -c $(SDIR)/commandline.c
$(PDIR)/function.o: $(SDIR)/function.c
  $(CC) $(CFLAGS) -c $(SDIR)/function.c
# Genz's Rule. This really should be elsewhere, so put it there someday.
$(PDIR)/gnzrul.o: $(SDIR)/gnzrul.c
  $(CC) $(CFLAGS) -c $(SDIR)/gnzrul.c
#User's Function(s)
$(UserFunction): $(FDIR)/user.c
  $(CC) $(CFLAGS) $(IDIR) -o $(FDIR)/user.o -c $(FDIR)/user.c

  # make proto is really only needed to be run by me. in general,
  # all the files should be there for everyone else.
proto:
  cd $(SDIR)
  echo "/* This file is autogenerated by 'make proto' */" > _x
  cp _x function.h
  cproto -epqsv function.c >> function.h
  cp _x commandline.h
  cproto -epqsv commandline.c >> commandline.h
  cp _x hyper_rect.h
  cproto -epqsv hyper_rect.c >> hyper_rect.h
\end{verbatim}
# DO NOT GENERATE HEAP.H!

```bash
# cp _x heap.h
# cproto -epqsv heap.c >> heap.h
rm -f _x

clean:
    rm -f $(FDIR)/*.o $(Parent) $(Child)

print:
    pr -f commandline.c hyper_rect.c commandline.c pvm_parent.c pvm_child.c heap.c heap.h\ function.c pvm_slave.h Makefile.aimk ./func/drvd01.c ./func/parint.h ./func/Drvd.group\ ./userfiles/func/Group > PRINTME.pvm.txt
```

```bash
$(XDIR):
    - mkdir $(BDIR)
    - mkdir $(XDIR)
```
Appendix D

Installation Guide
Tcl/TK version 4.0 or better is required for use.

PVM version 3.3.7 or better is required for PVM portions of the program.

MPI specification 1.0 is required for MPI portions of the program. For us, we used MPICH version 1.0.12.

As designed, ParInt can only work on one architecture at a time. However, manual compiles of the program components may all multi-architecture runs of programs. This is highly not-recommended.

The SysAdmin must do a ‘make setup’ to compile all of ParInt’s files
Appendix E

GUI Source Code
# The ParInt Package

# This is the main ParInt program.  ie, "Run Me!!!"

# You can invoke this with with 'ParInt' or 'wish ParInt'.

# The Computer Science Department

# Western Michigan University

# Kalamazoo, Michigan  49008

# USA

# Thesis person:

# Jay Ball

# Advisors:

# Dr. Elise de Doncker

# Dr. Ajay Gupta

# Parallel Contributors:

# Patricia Ealy

# Gui Contributors:

# Aman Sureka

# Find wish on the path and run it.  If it's not there, have your SysAdmin
# install it or tell you where it is located.
# These two lines are a tcltk comment, but a sh command!
exec 'which wish' "$0" ${1+"$0"}

if { $tk_version < 4.0 } {
    puts "You must run wish version 4.0 or higher.\n"
    exit -1
}

# set up the source directory
set Dir(src) gui

# now, for some stupid reason, the var $dir must be set to the source area,
# even though we may wish to use the var elsewhere.
set dir $Dir(src)

# load the files that set up the variables
source $Dir(src)/System.tcl

source $Dir(src)/Var.tcl

source $Dir(src)/Lang.tcl

# source 'miscfile.tcl' since we need it now to for the uid stuff
source $Dir(src)/miscfile.tcl
GetUserInfo

# load the library manager
# the library auto-update needs to be removed in the production version.
#load the library file
source $Dir(src)/tclIndex
lappend auto_path $Dir(src)

# ie, remove this if we release w/o desiring to get feedback or new procs.
# we must remove it. if a user tries to run it, the do not have write
# access and wish will crash out.
if { $Config(username) == $Config(paradmin) } {
    puts $StartMsg(generatelibrary)
    Library_UpdateIndex $Dir(src)
}

if { [string length [array names env PVM_ROOT]] == 0} {
    TextPrint $StartMsg(nopvmroot)
    exit -1
}

SetPVMArch

source $Dir(src)/DependantVar.tcl

ParseCommandLine

if [TestPVMRun] {
    TextPrint $StartMsg(pvmdaemon)
} else {
    TextPrint $StartMsg(nopvmdaemon)
}
# call the Main proc which builds the windows and starts the program
Main

gui/parinstall

#!/bin/sh
#
# The ParInt user install script.
#
# The point of this is to set up a user's ability to run ParInt
#
# The person that installs ParInt really really should modify these
# variables to match your system configuration.
#
# Someone should add some error checking and user interaction. Write
# in C or Perl?
#
#96???? jay: wrote it
#970220 jay: some fixes
#

#PVMhome=/usr/local/pvm3
PVMhome=/local/odin2/jaystuff/pvm-solaris/pvm3
#ParIntHome=/home/res/parint/ParInt
ParIntHome=/home/res/parint/ParInt.EXPERIMENTAL

# some system dependant defines.
CAT=cat
#CAT=/bin/cat
# this must support the -n option.
ECHO=/usr/ucb/echo
#ECHO=echo

# define some neat functions to make the user think we're alive

tic () { $ECHO -n "." ; }
endtic () { $ECHO "." ; }

# to my my typing less, and removing of statements easier
db () { $ECHO "(debug) $*" ; }

# functions for determining if a directory symbolic link is correct
#
# returns the value of a dir link in the EndDir variable.
linkend () { old='pwd' ; cd $1 ; EndDir='pwd' ; cd $old ; export EndDir ; }

# this tests if the file exists (ie, readable) then if it is a symlink,
# then attempts to correctly set it.
linkset () {
  if [ -r $1 ] ; then
    if [ -h $1 ] ; then
      linkend $1
      var="$EndDir"
    fi
  fi
  if [ "$var" = "$PVMhome/$1" ] ; then
    # Good, symlink properly set
    tic
  else
    $ECHO WARNING - `/pvm3/$1 points to $EndDir, but should point to $PVMhome/$1
  fi
else
  $ECHO WARNING - ~/pvm3/$1 should be a link to $PVMhome/$1
fi
else
  ln -s $PVMhome/$1 .
tic
fi
}

# begin main script....

$CAT «- EOL

Welcome to the ParInt user setup program!

This will attempt to install the user portions of PVM and ParInt into your home directory. This is required if you wish to use the ParInt package.

Installation will begin in 15 seconds unless you press ctrl-C now.

( note that 15 second delay is disabled for my patience threshold -j )

EOL
#sleep 15

# first, install a user version of pvm...
cd $HOME
tmppath=bin/"$PVMhome/lib/pvmgetarch"
if [ -d pvm3 ]; then
  $CAT <<- EOM
    Good, a user edition of PVM appears to be already installed. Be
    cautioned that if PVM is not installed properly, ParInt will not work.

    Now, an attempt will be made to verify a correct install.
  EOM

cd pvm3
linkset conf
linkset lib
linkset man
linkset include
else
  $ECHO -n "Installing PVM stuff."
mkdir pvm3
tic
cd pvm3
tic
ln -s $PVMhome/conf .
tic
ln -s $PVMhome/lib .
tic
ln -s $PVMhome/man .
tic
ln -s $PVMhome/include .
tic
mkdir bin
tic
mkdir $tmppath
endtic

120 fi

spacepath='$ECHO $PATH | sed 's/:/ /g'`

RequiredPath="$HOME/pvm3/$tmppath $HOME/pvm3/lib"

notperfect=0;
for requiredpath in $RequiredPath ; do {
    # $ECHO Searching for $requiredpath
    bad=0
    130 for userpath in $spacepath ; do {
        if [ $requiredpath = $userpath ] ; then
            bad=1;
            notperfect=1;
            fi
    } ; done
    if [ $bad = 0 ] ; then
        # $ECHO Found $requiredpath on your \$PATH
tic
    else
        $ECHO WARNING - Could not find $requiredpath on your \$PATH
        fi
    } ; done
tic

if [ $notperfect = 1 ] ; then
One or more warnings occurred, so some directories were not found on your \$PATH. For ParInt to work properly, you must add these to your path as defined in your login files (.cshrc, .profile, .bashrc, etc).

EOM

if [ $(PVM_ROOT:='not_set') != "$HOME/pvm3" ] then
  if [ $PVM_ROOT = "not_set" ] ; then
    $CAT «- EOM

Could not detect if the \$PVM_ROOT variable is set. For PVM (and consequently, ParInt) to work, you must set this variable in your login files (.cshrc, .profile, .bashrc, etc). Set the variable to '/pvm3' (or '\$HOME/pvm3' if using sh).

EOM

else
  $CAT «- EOM

The PVM_ROOT variable is set to '$PVM_ROOT' but, the expected value is '$HOME/pvm3' (ie, '/pvm3'). Thus, PVM may not run properly unless this variable is set correctly.

EOM

fi
the PARINT user install part
#
#

$ECHO "(debug) starting the parint install."
cd $HOME
if [ -d Parint ] ; then
    $ECHO "(debug) skipping parint."
else
    $CAT <<- EOM

    ParInt appears already installed. None of the ParInt file will be modified. If you are attempting to update or refresh your personal ParInt setup, you must first remove the old ParInt files. Do this by typing 'rm -rf $HOME/ParInt'. Then, re-run this program.

    Note that this will erase any work that you might have saved in the ParInt directories, including but not limited to personalized function group files, functions, methods, etc.

EOM

else
    $ECHO -n "Installing ParInt."
    mkdir ParInt
tic
# set up main parint dir
cd $ParIntHome/userfiles
tar -cf - * | (cd $HOME/ParInt ; tar -xf -)
tic

# create ParInt executable
cd $HOME/ParInt
$CAT > ParInt <<-EOM
#!/bin/sh
cd $ParIntHome
wish ./ParInt
EOM
tic
chmod 755 ParInt
tic

tic
# set up the pvm part of parint
cd $HOME/pvm3/$tmppath
if [ -r child ] ; then
  if [ -h child ] ; then
tic
  else
    $CAT <<-EOM
    WARNING - $HOME/pvm3/$tmppath/child exists and is not a symbolic link to $HOME/ParInt/pvm/child like it should be. Please correct this or ParInt will not run.
    EOM
    fi
  else
  fi
ln -s $HOME/ParInt/pvm/child child
fi
tic

if [ -r parent ]; then
    if [ -h parent ]; then
tic
    else
        $CAT <<-EOM
        WARNING - $HOME/pvm3/$tmppath/parent exists and is not
        a symbolic link to $HOME/ParInt/pvm/parent like it should
        be. Please correct this or ParInt will not run.
        EOM
    fi
else
    ln -s $HOME/ParInt/pvm/parent parent
fi
tic

# now, set up parint.h (ie, symlink to ParInt home)
db "now at the ln -s parint.h"
cd $HOME/ParInt/func
if [ -r parint.h ]; then
    if [ -h parint.h ]; then
        tic
    else
        db doing cat
        $CAT <<-EOM
        WARNING - $HOME/ParInt/func/parint.h exists and is not
        a symbolic link to $ParIntHome/func/parint.h like it should
be. Please correct this or ParInt will not run.
EOM
fi
else
270 db linking parint.h
    ln -s $ParIntHome/func/parint.h parint.h
fi

fi
# end of if ParInt is installed

endtic
280
cd $HOME
$CAT <<EOM

ParInt is now installed in your home area. To run, just execute 'ParInt/ParInt'. You can set up some sort of alias or add $HOME/ParInt to your \$PATH if desired.

If you received any warnings during installation, please correct them and re-run this installation program.

EOM
gui/About.tcl

1  
#  
# This file contains all necessary procs for the About box.  
#  
# RCS:  File: $RCSfile$
# Revision: $Revision$
# Last Changed by $Author$ on $Date$
# Currently Edited by: $Locker$
# ToDo: make a nice listbox with everyone's name and fun stuff like
#       ray traced graphics and feedback button so you can send us
#       donations to let us eat tommorrow.
# PseudoBugs: need to truely put the box at the center of the screen, not
# just put the nw corner there.
# Bugs:
# Log:  
#960127 Jay: creation
#970118 Jay: wm title created  

#  
# This is the very simple "About ParInt" box  
20  
#  
# ToDo:  
# PseudoBugs:  
# Bugs:  
# Log:  
#960127 Jay: creation  
proc AboutParInt {} {  
  global B ScreenCenterX ScreenCenterY WinTitle
if { [ winfo exists .about ] } {
    raise .about
} else {
    toplevel .about
    # place .about -relx 0.5 -rely 0.5 -anchor center
    wm geometry .about +$ScreenCenterX+$ScreenCenterY
    wm title .about $WinTitle(about)

    label .about.title -textvariable TextMsg(about,title)

    button .about.dismiss -textvariable B(dismiss) -command { destroy .about }

    pack .about.title -side top -fill x
    pack .about.dismiss
}

gui/C_Stuff.tcl

#
#
#
# RCS: File: $RCSfile$
#    Revision: $Revision$
#    Last Changed by $Author$ on $Date$
#    Currently Edited by: $Locker$
# ToDo:
# PseudoBugs:
# Bugs:
# Log:
#970210 Jay: created
#

proc MakeCFile { userfuncname infpvar outfpvar } {
    global Output
    upvar 1 $infpvar infp
    upvar 1 $outfpvar outfp

    puts $outfp $Output(header,c)
    #puts $outfp "#define start_${userfuncname} Start\n"
    #puts $outfp "#define finish_${userfuncname} Finish\n"
    #puts $outfp "#define dim_${userfuncname} FunctionDimensions\n"
    puts $outfp "#define initialize_${userfuncname} InitializeFunction\n"
    puts $outfp "#define ${userfuncname} functn_\n"
    puts $outfp "extern long Funct_Calls;\n"

    while { [gets $infp line] >= 0 } {
        # find "func(" or "func (" etc.
        set t [regexp -- "${userfuncname}(\s|\t|\n)*\([\]()\)"
                 $line match junk1 ]
        if { $t == 1} {
            # we found the func declaration line, now find for the starting brace"
            set t [ regexp -- "\{" $line match ]
            if { $t == 1} {
                # we have the "\{" on the same line as the func declaration
                puts $outfp $line
                # output the declaration line, then the F_C++, then rest of file
            }
        }
    }
puts $outfp "\n\n\t\tFunct_Calls++;\n\n"
while { [gets $infp line] >= 0} {
puts $outfp $line
}

} else {
#the "\{" appears somewhere else
#output declaration line, read and print lines until find "\{"
puts $outfp $line
while { [gets $infp line] >= 0} {
set t [ regexp -- "\{" $line ]
if { $t == 1} {
puts $outfp $line
puts $outfp "\n\n\t\tFunct_Calls++;\n\n"
while { [gets $infp line] >= 0} {
puts $outfp $line

50

}

}

} ;#end while
} ;#end if t==1
} else {
puts $outfp $line

60

}

}
}

gui/Compile.tel
1

#

#This file compiles the program
00
0


# RCS: File: $RCSfile$

# Revision: $Revision$

# Last Changed by $Author$ on $Date$

# Currently Edited by: $Locker$

# ToDo: everything.

# PseudoBugs:

# Bugs:

# Log:

#960320 Jay: creation
#960328 Jay: many changes. ugh.
#960522 Jay: added start, finish, and new integration dir structure.
#960731 Jay: more changes. more days. ugh.
#960831 Jay: additions, lot's of them.
#970108 Jay: i should remember to update the dammed comments once and a while.

# some files and name changes and many other things.

#970210 Jay: Ripped out C stuff and put into a new file. Added a

# switch call for C/Fortran check in Run.tcl, the integration

# window file. A button to "compile and run" now exists. Remember

# to check out the code you muck with the procedure names and button

# definitions here!!!!

#970211 Added "clear" button and function.

#

# description: The CompileStuff variable

#

# (arr index) what it does....

# log the text in the window's text box

# input the input buffer for the compiling process which is the output
buffer of the make program

# but the button which performs "run" or "stop"

# text the actual text of the run/stop button

# command the spawned command, ie 'make pvm'

# CreateSubstitutedFile
#
# this "attempts" to hack a user's .c (or .f etc) program function
# to a version that can be compiled and linked into the main program.
#
# Returns 0 upon success.
#
# ToDo:
# Somewhere in here, the code for a fortran program must be added.

# PseudoBugs:
# Bugs:
# This does not properly handle user C files that have commands
# immediatly starting after the "{" of the function. Ie,
# "funct( ... ) { code; ..." Code also implies comment blocks.

# Log:
#960522 Jay: made a real .c file that works.
#960820 Jay: added hooks for .f files. someone else can actually add the code.
#960825 Jay: make it so it actually reads the user's file.
#960831 Jay: message boxes. woo woo.

proc CreateSubstitutedFile { intfuncname userfuncname inputfilename } {
# check to see if they bothered to select a function
# actually, we should never get to this point....., i hope
if {{string length $userfuncname} < 1} {
    MessageBox $TextMsg(nofunc)
    puts "we got to some point that i didn't want to get to. search: GGGLPO"
    $CompileStuff(log) insert end $TextMsg(nofunc)\n
    return -1
}

StatusLine $StatusMsg(main,func)

set file [pathexpand func $inputfilename]

if [catch {open $file r 0644} infp] {
    $CompileStuff(log) insert end "$B(error): $infp. $ErrorMsg(compile,cantopeninfp)"
    return -1
}

if [catch {open $Dir(func)/user.c w 0644} outfp] {
    $CompileStuff(log) insert end "$B(error): $outfp. $ErrorMsg(compile,cantopenoutfp)"
    return -1
}

switch $Config(proglang) {
    c { MakeCFile $userfuncname infp outfp }
    f { MakeFortranFile $userfuncname infp outfp }
    default { puts "we shouldn’t get here Config(proglang) set to something funny..." ; exit }
}
close $infp
close $outfp
return 0
} ;# end CreateSubstitutedFile

#
#
# ToDo:
# PseudoBugs:
# Bugs:
# Log:
#
proc CompileRun {} {
    global CompileStuff B Config Group

    set err [CreateSubstitutedFile $Config(intfuncname) $Config(func) $Config(funcfile)]
    if {$err == 0} {
        if [catch {open "$CompileStuff(command) I& cat"} CompileStuff(input)] {
            $CompileStuff(log) insert end $CompileStuff(input)
            $CompileStuff(log) insert end "$B(errorc)$ErrorMsg(externalcommand,A)
            CompileStuff(command)$ErrorMsg(externalcommand,B)"
        } else {
            fileevent $CompileStuff(input) readable CompileLog
            $CompileStuff(log) insert end $CompileStuff(command)\n
        set CompileStuff(text) $B(stop)
        $CompileStuff(but) config -command CompileStop
    }
else {
    puts "an error in comprun - createsubfile occurred"
}
$CompileStuff(log) see end
}; # end CompileRun

# 130 # Read and log output from the program
#
#
# ToDo:
# PseudoBugs:
# Bugs:
# Log:
#
proc CompileLog {} {
    global CompileStuff
    140 if [eof $CompileStuff(input)] {
        CompileStop
    } else {
        gets $CompileStuff(input) line
        $CompileStuff(log) insert end $line\n        $CompileStuff(log) see end
    }
}

# 150 # Stop the program and fix up the button
#
#
# ToDo:
# PseudoBugs:
# Bugs:
# Log:
#
proc CompileStop {} {
    global CompileStuff B
    catch {close $CompileStuff(input)}
    set CompileStuff(text) $B(run)
    $CompileStuff(but) config -command CompileRun
}

#
# Clear the compile window
#
# ToDo:
# PseudoBugs:
# Bugs:
# Log:
#
proc CompileLogClear {} {
    global CompileStuff
    $CompileStuff(log) delete 1.0 end
}

#
proc CompileProgram {} {
    global Config B CompileStuff TextMsg WinTitle
    StatusLine ""
    if {[string length $Config(func)] < 1 } {
        MessageBox $TextMsg(nofunc)
        return -1
    }

    set w .compilelog ;# set a var to save typing
    if { [wininfo exists $w] } {
        raise $w
    } else {
        toplevel $w
        wm title $w $WinTitle(compile)

        # Create a frame for buttons and entry.
        frame $w.but -borderwidth 10
        pack $w.but -side bottom -fill x

        # Create the command buttons.
        button $w.but.dis -textvariable B(dismiss) -command {destroy .compilelog}
        button $w.but.clear -textvariable B(clear) -command CompileLogClear

        set CompileStuff(but) [button $w.but.run -width $B(runstopwidth)\
-textvariable CompileStuff(text) -command CompileRun]
pack $w.but.dis $w.but.run $w.but.clear -side right

# Create a labeled entry for the command
#label $w.t.l -text Command: -padx 0
#entry $w.t.cmd -width 20 -relief sunken -textvariable command
#label $w.t.cmd -width 20 -relief sunken -textvariable command

pack $w.t.l -side left
pack $w.t.cmd -side left -fill x -expand true

# Set up keybinding equivalents to the buttons
#bind $w.t.cmd <Return> CompileRun
#bind $w.but.cmd <Control-c> CompileStop
bind $w <Control-c> CompileStop
#focus $w.t.cmd

# Create a text widget to log the output
frame $w.text
set CompileStuff(log) [text $w.text.log -width 40 -height 10 
   -borderwidth 2 -relief raised -setgrid true -wrap word 
   -yscrollcommand "$w.text.scroll set"]
scrollbar $w.text.scroll -command "$w.text.log yview"
pack $w.text.scroll -side right -fill y
pack $w.text.log -side left -fill both -expand true
pack $w.text -side top -fill both -expand true
}

#puts "The cwd dir is [cwd]"
set CompileStuff(command) "make $Config(system)"
set CompileStuff(text) $B(run)
gui/Config.tcl

# Information on different internal variables can be displayed with the procs
# in here
#
# RCS: File: $RCSfile$
# Revision: $Revision$
# Last Changed by $Author$ on $Date$
# Currently Edited by: $Locker$
# ToDo:

# PseudoBugs:
# Bugs:
# Log:
#960212 Jay: creation
#970211 Jay: change wm title to a var. added nice "text mode show all"
# to the procedure.

#
#
#
# ToDo:
# PseudoBugs:
# Bugs:
# Log:
#960212 Jay: creation
#

proc UpdateConfigDisplay {} {
    global B Config ConfigOptions ConfigLongDescName WinTitle
    set w .config
    if { [ winfo exists .config] } {
        raise $w
    } else {
        toplevel $w
        #wm minsize $w 250 40
        wm title $w $WinTitle(config)
        frame $w.var
        frame $w.des
        frame $w.val -width 100
        frame $w.but
        pack $w.but -side bottom
        pack $w.var $w.des $w.val -side left
        label $w.var._head -anchor w -relief raised -text "Name"
        label $w.des._head -anchor w -relief raised -text "Description"
        label $w.val._head -anchor w -relief raised -text "Value"
        pack $w.var._head -side top -fill x
        pack $w.val._head -side top -fill x
        foreach o $ConfigOptions {
            label $w.var.$o -anchor w -text $o
            label $w.val.$o -anchor w -textvariable Config($o)
            label $w.des.$o -anchor w -textvariable ConfigLongDescName($o)
            pack $w.var.$o -side top -fill x
            pack $w.val.$o -side top -fill x
        }
pack $w.des.$o -side top -fill x
}

button $w.but.dismiss -textvariable B(dismiss) -command {destroy .config}
pack $w.but.dismiss -side bottom
}

# display all possible values of a $Config for giggles
if {$Config(debugmessagelevel) <= 1000 } {
    foreach i [array names Config] {
        format "%30s %s" $i $Config($i)
    }
} ;#endif

} ;# end UpdateConfigDisplay

gui/Debug.tcl

# Debugging crap
#
# RCS: File: $RCSfile$
#    Revision: $Revision$
#    Last Changed by $Author$ on $Date$
#    Currently Edited by: $Locker$
# ToDo:
# PseudoBugs:
# Bugs:
# Log:
#
# simply prints out debugging messages if Config <= passed level
#
# ToDo:

# PseudoBugs:
# Bugs:
# Log:
#
proc db {level message} {
    global Config

    if { $Config(debugmessagelevel) <= $level } {
        puts $message
    }
}

# DependantVar.tcl
#
# These variable depend on others, so they must be set later.
#
# RCS: File: $RCSfile$
# Revision: $Revision$
# Last Changed by $Author$ on $Date$
# Currently Edited by: $Locker$
# ToDo:
# PseudoBugs:
# Bugs:
# Log:
#

# Configuration options is what will be displayed in the 'show variables' box
set ConfigOptions { system numproc dim proglang group func funcfile}

#set Config(system)                  pvm/$Config(pvmarch)
set Config(system)                  pvm

set GroupListFile                    Group
set Config(numproc)                 4
set Config(dim)                     2
set Config(proglang)                c
#set Config(group)                   drvd
set Config(group)                   ""
#set Config(func)                    drvd01
set Config(func)                    ""
set Config(intfuncname)             "functn_"
#set Config(funcfile)                func.c
set Config(funcfile)                ""

# this is the width of Run and Stop. This is the maximum size and is used when
# packing the windows that have the button that changes text. this is put in
# $B for ease.
# the only problem is that it makes it much wider than i want it to be.
set B(runstopwidth) 0
foreach XXval { $B(run) $B(stop) } {
    if {[string length $XXval] > $B(runstopwidth)} {
        set B(runstopwidth) [string length $XXval]
    }
}
unset XXval

gui/Fortran_Stuff.tcl

#
#
# RCS: File: $RCSfile$
# Revision: $Revision$
# Last Changed by $Author$ on $Date$
# Currently Edited by: $Locker$
# ToDo:
# PseudoBugs:

THIS DOES NOT WORK!
I WILL NOT MAKE IT WORK!
FORTRAN SHOULD DIE LIKE A SQUEALING PIG!

Log:
#970210 Jay: created. Fortran sucks!
#
#
# ToDo:
# PseudoBugs:
# Bugs:
# THIS DOES NOT WORK!
# I WILL NOT MAKE IT WORK!
# FORTRAN SHOULD DIE LIKE A SQUEALING PIG!
# Log:
#970210 Jay: created. Fortran sucks!

proc MakeFortranFile { userfuncname infpvar outfpvar } {
    global Output
    upvar 1 $infpvar infp
    upvar 1 $outfpvar outfp
    puts $outfp $Output(header,f)
    #puts $outfp "#define start_${userfuncname} Start\n"
    #puts $outfp "#define finish_${userfuncname} Finish\n"
    #puts $outfp "#define dim_${userfuncname} FunctionDimensions\n"
    puts $outfp "#define initialize_${userfuncname} InitializeFunction\n"
    puts $outfp "#define ${userfuncname} func\n"
    puts $outfp "extern long Funct_Calls;\n"
    while { [gets $infp line] >= 0 } {
        # find "func(" or "func (" etc.
        set t [regexp -- "$\{userfuncname\}\((\[ |\t|\n\])*\[\]"
            $line match junk1 ]
        if { $t == 1} {


# we found the func declaration line, now find for the starting brace
\{
set t [ regexp -- "\{" $line match ]
if { $t == 1 } {
    # we have the "\{" on the same line as the func declaration
    puts $outfp $line
    # output the declaration line, then the F_C++, then rest of file
    puts $outfp "\n
    Funct_Calls++;\n"
    while { [gets $infp line] >= 0 } {
        puts $outfp $line
    }
}
else {  
    # the "\{" appears somewhere else
    # output declaration line, read and print lines until find "\{
    puts $outfp $line
    while { [gets $infp line] >= 0 } {
        set t [ regexp -- "\{" $line ]
        if { $t == 1 } {
            puts $outfp $line
            puts $outfp "\n
            Funct_Calls++;\n"
            while { [gets $infp line] >= 0 } {
                puts $outfp $line
            }
        } ;# end while
    } ;# end if t==1
} else { 
    puts $outfp $line
}
proc choosefunc {} {
    global B GenMsg TextMsg Config GroupList WinTitle ErrorMsg
}
set w .choosefunc ;# set a var to save typing
if { [winfo exists $w] } {
    raise $w
} else {
    toplevel $w
    wm title $w $WinTitle(func)
    frame $w.f
    pack $w.f -expand true -padx 5 -pady 5 -fill both

    message $w.f.howto -relief ridge -justify center -aspect 750 
    -textvariable TextMsg(choosefunc,use)

    frame $w.f.curfun
    label $w.f.curfun.l -textvariable ConfigLongDescName(func)
    label $w.f.curfun.r -textvariable Config(func)
    pack $w.f.curfun.l $w.f.curfun.r -side left -fill x

    frame $w.f.curser
    label $w.f.curser.l -textvariable ConfigLongDescName(group)
    label $w.f.curser.r -textvariable Config(group)
    pack $w.f.curser.l $w.f.curser.r -side left -fill x

    frame $w.f.but
    button $w.f.but.dis -textvariable B(dismiss) -command {destroy .choosefunc}
    button $w.f.but.upd -textvariable B(update) -command {UpdateGroupBox}
    #pack $w.f.but.dis $w.f.but.upd -side left
    pack $w.f.but.dis -side left -padx 5 -pady 5
    pack $w.f.but.upd -side left -padx 5 -pady 5

    frame $w.f.box
ScrolledListbox2 $w.f.box.group -width 15 -height 4 -selectmode single
ScrolledListbox2 $w.f.box.func -width 15 -height 4 -selectmode single
pack $w.f.box.group $w.f.box.func -side left -fill both -expand true

message $w.f.serinfo -relief ridge -justify center -aspect 750 \ 
-textvariable GenMsg(serinfo)

message $w.f.funcinfo -relief ridge -justify center -aspect 750 \ 
-textvariable GenMsg(funcinfo)

pack $w.f.howto -side top -fill x
pack $w.f.but -side bottom
pack $w.f.funcinfo -side bottom -fill x
pack $w.f.curfun -side bottom -fill x
pack $w.f.serinfo -side bottom -fill x
pack $w.f.curser -side bottom -fill x
pack $w.f.box -side bottom -expand true -fill both

#bind $w.f.box.group.list <ButtonRelease-1> \ 
[list UpdateSecondList %W %y $parent.subchoice.list $choices]
bind $w.f.box.group.list <ButtonRelease-1> \ 
{ Clicked_on_Group %W %y }

bind $w.f.box.func.list <ButtonRelease-1> \ 
{ Clicked_on_Function %W %y }

# Selecting in subchoice deletes items
#bind $parent.subchoice.list <ButtonPress-1> \ 
{SetTelnetHost %W %y }
UpdateGroupBox

} ;# end of raise window/create window
90 } ;# end of proc choose function

#
#
#
# ToDo:
# PseudoBugs:
# Bugs:
# Log:
#960901 Jay: cleared out Config(func & funcfile) if group is clicked
100 
#
proc Clicked_on_Group { w y } {
    global Group GenMsg Config TextMsg
    $w select set [$w nearest $y]
    set group [$w get [$w nearest $y]]
    set Config(group) $group
    set Config(func) ""
    set Config(funcfile) ""

    .choosefunc.f.box.func.list delete 0 end
    eval {.choosefunc.f.box.func.list insert end} $Group($group)
    set GenMsg(funcinfo) ""

    set GenMsg(serinfo) $Group($group,desc)
    if {([string length $GenMsg(serinfo)] == 0) {
        set GenMsg(serinfo) $TextMsg(noinfo)
proc Clicked_on_Function { w y } {
    global Group GenMsg Config TextMsg
    $w select set [$w nearest $y]
    set func [$w get [$w nearest $y]]
    set Config(func) $func
    set Config(funcfile) $Group($Config(group),$func,file)
    set GenMsg(funcinfo) $Group($Config(group),$func,desc)
    if {[string length $GenMsg(funcinfo)] == 0} {
        set GenMsg(funcinfo) $TextMsg(noinfo)
    }
} ;# end of Clicked_on_Function
# Bugs:
# Log:

150 proc UpdateGroupBox {} {
    global GroupList GroupListFile GenMsg Config
    LoadGroupList $GroupListFile

    #foreach ser $GroupList(var) {
        #eval {.choosefunc.f.box.group.list insert end} $ser
    #}
    .choosefunc.f.box.group.list delete 0 end
    eval {.choosefunc.f.box.group.list insert end} $GroupList(var)
    .choosefunc.f.box.func.list delete 0 end
    set GenMsg(serinfo) ""
    set GenMsg(funcinfo) ""
    set Config(group) ""
    set Config(func) ""

} ;# end of proc UpdateGroupBox

#
# LoadGroupList

170 #
# in theory, this will load a file containing the group list. the format is:
#
# '##' at the beginning of line is a comment
# blank lines are ignored, indentation is ignored
# TYPE = val1 val2 ...
#
# the only supported type is GROUP, so
# GROUP = variable filename [description]
# variable is the internal tcl name to use, and each one in the file
# must be unique. Only A-Z, a-z, 0-9, and _ may be used in the
# var name.
# filename is the file which contains the function list. The filename
# must be a valid unix name, except no whitespace is allowed.
# description is a nice phrase to describe the group. Optional.
#
# NEXT, the group itself is loaded. The file format is similar.
#
# (place here when not so bored)
#
# ToDo:
# need to add more error checking around the regexp commands.
# make the regexp error out better with extra words at start of line
#
# PseudoBugs:
# Log:
# 950331 jay created and modified this so many times that i didn’t feel like
# recording it here.
#
proc LoadGroupList { ListFileName } { ,
    global Dir GroupList Group
    set hostlist x
    set f [pathexpand func $ListFileName]
    # if [catch {open $Dir(func)/$ListFileName r} flfp]
    if [catch {open $f r} flfp] {
        # this needs to be more user friendly
        puts stderr "$B(error): ‘$flfp’. $ErrorMsg(func,nolistfile)"
    }
exit -1

} else {
    #unset GroupList(file) GroupList(var) GroupList(count)
    210 if [info exists GroupList] {
        unset GroupList Group ;# if one exists, then the other must too
    }
    set GroupList(count) 0
    # while { [set line [getvalidline $flfp]] } >= 0
    #while { [getvalidline flfp line] } >= 0
    while { [gets $flfp line] } {
        # get rid of whitespace on the ends (^ and $)
        set line [string trim $line]
        # how do you combine these two ifs into one line?
        220 if { [string index $line 0] == "#" } {
            continue
        }
        if { [string length $line] == 0 } {
            continue
        }
        set t [regexp -- "^([a-zA-Z0-9_]*)([ l	l
]+)=( [ l	l
]+)(.*)" $line match word junk1 junk2 predicate ]
        if { $t != 1 } {
            puts stderr "The group input file '$ListFileName' contains an invalid line '$line'"
            continue
        }
        #puts "$match='\$match' word='\$word' ($junk1) = ($junk2) predicate='\$predicate'"
        set word [string tolower $word]
        #puts stdout "line is '$line'"
        #set e [string first {=} $line ] ;# the location of the =
        #puts stdout " e is $e"
#set word [string trimright [string range [string tolower $line] 0 [expr $e - 1]]]
#puts stdout "word is '$word'
#set predicate [string trimleft [string range $line [expr $e + 1] end]]
#puts stdout "predicate is '$predicate'
switch -exact -- $word {
  group {
    # this regexp command saves like 6 lines of string&set commands
    # the \[ is needed to escape out the [ command.
    regexp "([a-zA-Z0-9_]*)(\[ |\t|\n\])*([\t|\n\]*)(\[ |\t|\n\]*)(.*)" \\
    $predicate match var junk1 file junk2 desc
    #puts "var='"$var' ("$junk1) filename='"$file' ("$junk2) desc='"$desc'"
    # end of 'group'
  }
  default {
    # puts stderr "The group input file '$ListFileName' has an invalid\n    command\n    puts stderr "$ErrorMsg(func,badline,A) '$ListFileName'"
    puts stderr "$ErrorMsg(func,badline,B) '$word' $ErrorMsg(func,badline,C)"$line'\n    continue
    # end of 'default'
  }
} ;# end switch

# now we load each file
set file [pathexpand func $file]
if [catch {open $file r} fp] {
  puts stderr "Cannot open '$file'.\nReason given: $fp."
} else {
  # if the file can be open, then assume that it is valid.
# add it to the group list.
lappend GroupList(var) $var
incr GroupList(count)

set Group($var,file) $file
set Group($var,desc) $desc
set func ""
while { [gets $fp line] >= 0 } {
    # get rid of whitespace on the ends (~ and $)
    set line [string trim $line]

    # how do you combine these two ifs into one line?
    if { [string index $line 0] == "#" } {
        continue
    }
    if { [string length $line] == 0 } {
        continue
    }
    puts stdout "line is 'line'
    set e [string first {=} $line ] ;# the location of the =
    puts stdout " e is $e"
    set word [string trimright [string range [string tolower $line] 0 [expr $e - 1] ]] ;# puts stdout "word is 'word"
    set predicate [string trimleft {[string range $line [expr $e + 1] end ]] ;# puts stdout "predicate is 'predicate"

    switch -exact -- $word {
    func {
        regexp "([^a-zA-Z0-9_]*)([^t\n]*[^a-zA-Z0-9/\$.*])([^t\n]*[^.]*)([^t\n]*)\$predicate match func junk1 file junk2 desc
        lappend Group($var) $func
    }
set Group($var,$func,var) $func
set Group($var,$func,file) $file
set Group($var,$func,desc) $desc

# puts "func='"$func' ("junk1) file='"$file' ("junk2) desc='"$desc'"
# end of 'func'
}
ascii {

# end of 'ascii'
}
tex {

# end of 'tex'
}
html {

# end of 'html'
}
default {
    puts stderr "The group input file '$file' has an invalid line '$line'"
    continue
    # end of 'default'
}
}
};# end switch

};# while getline for fp
close $fp
};# end of else for fp open
};# end while flfp
close $flfp
gui/Help.tcl

# The Help File
#
# RCS: File: $RCSfile$
#   Revision: $Revision$
#   Last Changed by $Author$ on $Date$
#   Currently Edited by: $Locker$
# Loads: The HTML Library
# ToDo: Redo this whole file into something nicer.
# PseudoBugs:
# Bugs: This is not fully integrated. Namely, the style is different and all of
#   the same vars for Config are not read.
#   Has it's own gloabals that may config (but don't).
# Log:
# #960127 Jay: Initially created
# #960901 Jay: removed the options menu (commented out)
#   set default font size down to 1
# *****MAKE SURE THAT IF html_library_0.1 IS UPGRADED, THEN HMLink_callback
#   MUST BE DELETED FROM THERE AS IT WILL CONFLICT WITH THE AUTO-LIBRARY
# 20 # LOADER AND PREVENT THE CORRECT VERSION FROM WORKING!!!*****
#
# The actual help command and library.
#
# Parts of this code were taken from the html_library-0.1 demos. The code has
# been modified for toplevel windows and other fun stuff.
#
# Copyright (c) 1995 by Sun Microsystems
# See the file "license.terms" for information on usage and redistribution
# of this file, and for a DISCLAIMER OF ALL WARRANTIES.
#
# ToDo: Redo this whole file into something nicer.
#
# PseudoBugs:
# Bugs:
# Log:
#960128 Jay: create and port.
#960901 Jay: added $B variable
#970109 Jay: added $B(busy & ready)
#970118 Jay: more stuff
#970220 Jay: nicer top of window
#

proc HMsetup {} {
    global B WinTitle

    toplevel .helpwindow
    wm geometry .helpwindow 500x400
    wm title .helpwindow $WinTitle(help)

    frame .helpwindow.top
    frame .helpwindow.top.f
frame .helpwindow.bottom

# menubutton .helpwindow.menu -relief raised -bd 2 -text options... -menu .helpwindow.menu.m
button .helpwindow.bottom.quit -command {destroy .helpwindow} -textvariable B(dismiss)
entry .helpwindow.top.f.entry -textvariable Url -width 55
label .helpwindow.top.f.file -textvariable B(filec)
label .helpwindow.bottom.status -textvariable Running -width 6 -relief ridge -bd 2 -padx 9 -pady 3
label .helpwindow.top.msg -textvariable message

scrollbar .helpwindow.scrollvert -command ".helpwindow.text yview" -orient v
scrollbar .helpwindow.scrollhorz -command ".helpwindow.text xview" -orient h
option add *Text.height 40 startup
option add *Text.width 80 startup
text .helpwindow.text -xscrollcommand ".helpwindow.scrollhorz set" \
-yscrollcommand ".helpwindow.scrollvert set" -padx 3 -pady 3

pack .helpwindow.top.f.file -side left -anchor w
pack .helpwindow.top.f.entry -side left -anchor w -fill x
pack .helpwindow.top.f .helpwindow.top.msg -side top -fill x
pack .helpwindow.bottom.status -side left -anchor w
pack .helpwindow.bottom.quit -side right -anchor e

pack .helpwindow.top -side top -fill x
pack .helpwindow.bottom -side bottom -fill x
pack .helpwindow.scrollvert -side left -expand 0 -fill y
pack .helpwindow.scrollhorz -side bottom -expand 0 -fill y
pack .helpwindow.text -side left -fill both -expand 1

# set up some bindings
bind .helpwindow.top.f.entry <Return> {render $Url}
bind all <End> {.helpwindow.text yview end}
bind all <Home> {.helpwindow.text yview 0.0}
bind all <Next> {.helpwindow.text yview scroll 1 page}
bind all <Prior> {.helpwindow.text yview scroll -1 page}

# Menus are not used.

# menu .helpwindow.menu.m
# .helpwindow.menu.m add command -label "option menu"
# .helpwindow.menu.m add separator
# .helpwindow.menu.m add command -label "font size" -foreground red
# .helpwindow.menu.m add radiobutton -label small -value 0 -variable HtmlSize \
#   -command {HMset_state .helpwindow.text -size $HtmlSize; render $Url}
# .helpwindow.menu.m add radiobutton -label medium -value 4 -variable HtmlSize \
#   -command {HMset_state .helpwindow.text -size $HtmlSize; render $Url}
# .helpwindow.menu.m add radiobutton -label large -value 12 -variable HtmlSize \
#   -command {HMset_state .helpwindow.text -size $HtmlSize; render $Url}
# .helpwindow.menu.m add separator
# .helpwindow.menu.m add command -label "indent level" -foreground red
# .helpwindow.menu.m add radiobutton -label small -value 0.6 -variable Htmllndent \
#   -command {HMset_indent .helpwindow.text $Htmllndent}
# .helpwindow.menu.m add radiobutton -label medium -value 1.2 -variable Htmllndent \
#   -command {HMset_indent .helpwindow.text $Htmllndent}
# .helpwindow.menu.m add radiobutton -label large -value 2.4 -variable Htmllndent \
#   -command {HMset_indent .helpwindow.text $Htmllndent}
}

# go render a page. We have to make sure we don't render one page while
# still rendering the previous one.
proc render {file} {
    global HM.text B Running message

    HMreset_win .helpwindow.text
    set Running $B(busy)
    set message "Displaying $file"
    update idletasks
    HMparse_html [get_html $file] {HMrender .helpwindow.text}
    set Running $B(ready)
    HMset_state .helpwindow.text -stop 1 ;# stop rendering previous page if busy
    set message ""
}

# given a file name, return its html

proc get_html {file} {
    global Home
    if {[catch {set fd [open $file]} msg]} {
        return "
        <title>Bad file $file</title>
        <h1>Error reading $file</h1><p>
        $msg<hr>
        <a href=$Home>Go home</a>
     "
    } else {
        set result [read $fd]
        close $fd
        return $result
    }
}
# Override the library link-callback routine for the sample app.
# It only handles the simple cases

proc HMlink_callback {win href} {
    global Url
    if {[string match f* $href]} {
        set Url $href
    } else {
        set Url [file dirname $Url]/$href
    }
    update
    render $Url
}

# supply an image callback function
# Read in an image if we don't already have one
# callback to library for display

proc HMset_image {handle src} {
    global Url message
    if {[string match f* $src]} {
        set image $src
    } else {
        set image [file dirname $Url]/$src
    }
    set message "fetching image $image"
    update
    if {[string first " $image " [image names] "] >= 0} {
        HMgot_image $handle $image
    }
} else {
    
    catch {image create photo $image -file $image} image
    HMgot_image $handle $image
}
}

# Lets invent a new HTML tag, just for fun.
# Change the color of the text. Use html tags of the form:
# <color value=blue> ... </color>
# We can invent a new tag for the display stack. If it starts with "T"
# it will automatically get mapped directly to a text widget tag.

proc HMtag_color {win param text} {
    upvar #0 HM$win var
    set value bad_color
    HMextract_param $param value
    $win tag configure $value -foreground $value
    HMstack $win "" "Tcolor $value"
}

proc HMtag_/color {win param text} {
    upvar #0 HM$win var
    set value bad_color
    HMstack $win / "Tcolor {}"
}

#
#
# These are loaded when the user selects HELP
#
# set initial values
set HtmlSize 1 ;# font size adjustment
set HtmlIndent 1.2 ;# tab spacing (cm)
set Home $Dir(help)/help.html ;# home document
set Url $Home ;# current file
set Running $B(busy) ;# page status
set message "" ;# message line

proc HelpSystem {} {
    global HtmlSize HtmlIndent Home

    if { [ winfo exists .helpwindow ] } {
        raise .helpwindow
    } else {

        HMsetup
        HMinit_win .helpwindow.text
        HMset_state .helpwindow.text -size $HtmlSize
        HMset_indent .helpwindow.text $HtmlIndent
        render $Home
proc PrintDirs { } {
    global Dir
foreach d $Dir {
    puts "$d=$Dir($d)"
}

gui/Lang.english.tcl

# This is the visual language file. It contains all phrases, words, menus,
# dialogues, etc for the entire ParInt program. We could do somesort of
# special system to load the text, but all the information about the text
# would still have to be present. Thus, a different file format is almost
# futile. The only savings would be the removal of the 'set' in each line.
# Hence, just edit the source. :)
#
# RCS: File: $RCSfile$
# Revision: $Revision$
# Last Changed by $Author$ on $Date$
# Currently Edited by: $Locker$
# ToDo: Make a "real" setup file that doesn't look like tcltk.
# PseudoBugs:
# Bugs:
# Log:
# 960XXX Jay: created and modified way to many times.
# 960114 Jay: changed alot and took out most non-language stuff
#
#
# -------------------------
# def ConfigLongDescName Array This contains single line descriptions of
# Config options. These are displayed when people select "Show Variables"

set ConfigLongDescName(system) "Current Integration System"
set ConfigLongDescName(numproc) "Number of Processes"
set ConfigLongDescName(dim) "Processor Dimensions"
set ConfigLongDescName(proglang) "Programming Language"
set ConfigLongDescName(group) "Current Group"
set ConfigLongDescName(func) "Current Function"
set ConfigLongDescName(intfuncname) "Integration Function Name (in source)"
set ConfigLongDescName(funcfile) "Function File Name"

# -------------------------
# def TextMsg in general, these are longer messages that are displayed
# in windows. The arrays are two dimensional (hopefully) by (windowish,
# usage)

set TextMsg(dim, warn) "Note: Most of the procedures for area division only support process quantities in power of 2. Thus, for ease, we impose this limit on everything for this version of ParInt."
set TextMsg(dim, use) "Please select the number of dimensions of processes to use"

# the text that appears in the about box.
set TextMsg(about, title) "The ParInt Package"
set TextMsg(init,nopvmgetarch) "Cannot find PVM's get arch command"

set TextMsg(choosefunc,use) "First, choose a group to integrate, then choose an individual function."

set TextMsg(noinfo) "No information is available"

# on the main menu, some features are not yet supported.
set TextMsg(main,fortran) "As of this time, the use of Fortran as a programming language is not supported."
set TextMsg(main,ncube) "Use of the nCUBE is not yet supported."
set TextMsg(main,mpi) "Use of MPI is not yet supported."
set TextMsg(main,seq) "Use of Sequential processing is not yet supported."

# if you try to compile or integrate w/o a selected f'n.
set TextMsg(nofunc) "You must select a function first."

#-------------------------
# def Status Line messages should be short. These are changed when the user does something in the program. Usage is: (windowish, usage).

# main,system is the starting phrase to when the user selects pvm, mpi, etc.
set StatusMsg(main,system) "Distributed System set to " ;# followed by pvm, mpi
set StatusMsg(main,language) "Programming Language set to " ;# followed by C, etc
set StatusMsg(main,pvmconsole) "Started the PVM console"
set StatusMsg(main,func) "Select a group, then function"
#-------------------------
# what did i mean by this?
# def GenMsg in general are items that the user inputs into files which may be
# in a single language. These are used in various parts of the program

#
90 #-------------------------
100
90 #-------------------------
100 # Basic windowing buttons is short named array
100 set B(ok) OK
100 set B(cancel) Cancel
100 set B(dismiss) Dismiss
100 set B(update) Update
100 set B(stop) Stop
100 set B(save) Save
100 set B(run) Run
100 set B(ready) Ready
100 set B(busy) Busy
100 set B(clear) Clear
100 set B(compact) Compact
100 set B(wrap) "Word Wrap"
100 set B(compileandrun) "Compile+Run"
100
100 # not usually a button, yet we put it here for ease:
100 set B(error) Error
100 set B(errorc) "Error:"
100 set B(filec) "File:"
100
110 # these are used in the limit setting window
110 set B(limits) Limits
set B(defaultlimits) "Default Limits"
set B(userlimits) "Above Limits"

#
#-------------------------
# The status line at the bottom is just a simple variable. However, use the
# special command to set it, only it set it here for the first time.
# The status line at the bottom is just a simple variable. However, use the

set Status "Welcome to ParInt"

#
#-------------------------
# some error messages that appear in the program.
#
# preceeded by "error: fileerror. "

setErrorMsg(compile,cantopeninfp) "ParInt needs to read in the file that
contains the function to be integrated. If this is a user created file, make
sure that it exists in the proper directory."

# preceeded by "error: fileerror. "
setErrorMsg(compile,cantopenoutfp) "ParInt needs be able to create this file
for you to use the program. Check the directory permissions and try to compile
again."

# format: "$B(errorc)$partA$filename_in_proc$partB" notice that no spaces
# can be in the string that uses these commands

setErrorMsg(externalcommand,A) "For some reason, ParInt could not run the external command ."
setErrorMsg(externalcommand,B) ". Please check your PATH variable to en sure
ParInt can find this required program, then re-load ParInt.

"$B(error): '$flfp'. $ErrorMsg"

set ErrorMsg(func, nolistfile) "Since this file cannot be loaded, you cannot integrate, hence, the running of this program will be terminated."

# this deals with bad lines in the list file
# it goes: "partA '$filename' partB '$word' partC 'line'"
set ErrorMsg(func, badline, A) "The group input file"
set ErrorMsg(func, badline, B) "has an invalid command"
set ErrorMsg(func, badline, C) "on line"

set ErrorMsg(pvmarch) "Unknown or not installed"

# the start up messages. these are only used before the main window
# init. in general, once the program starts you can delete the array.
set StartMsg(unknownargv) "Unknown command line argument:"

set StartMsg(unknownuid) "Cannot find your user ID. This may present problems depending on your system configuration."

set StartMsg(unknownlang) "Unknown visual language:"

set StartMsg(cantfindid) "An error occured trying to parse the
'id' command. Please inform the ParInt administrator to fix this error.

# this occurs when the sysadmin sets up parint wrong.
set StartMsg(uidnotsetup) "Please tell the ParInt SysAdmin to set up the GetUid function and Config(uidsearch) vars properly."

# if PVM_ROOT is not set, then display this:
set StartMsg(nopvmroot) "This version of ParInt only works with PVM, and the setting of the environment variable 'PVM_ROOT' is required. You usually set this in your .cshrc or .bashrc file to '~/pvm3"

# if you are running the PVM daemon....
set StartMsg(pvmdaemon) "It appears that you are already running the PVM daemon, good."

# if you are NOT running the PVM daemon now....
set StartMsg(nopvmdaemon) "PVM daemon not running. You may wish to select PVM console to run the daemon. Choose Systems:Options:PVMConsole."

# if the ParAdmin is running ParInt, then update the Library files in case of modifications
set StartMsg(generatelibrary) "Generating the library files..."

#-------------------------
# the title bar of the toplevel windows will read....
# WARNING, this is highly window manager dependant. Thus, if you wish
# use a a different font than the window manager will allow, this will
# not work.
set WinTitle(compile) "Compile Log"
set WinTitle(main) "The ParInt Project"
set WinTitle(func) "Choose Function"
set WinTitle(message) "Message"
set WinTitle(numproc) "Number of Processes"
set WinTitle(help) "Help System"
set WinTitle(config) "System Variable Configuration"
set WinTitle(about) "About ParInt"
set WinTitle(integrate) "Integration Log"
set WinTitle(limits) "Limits of Integration"

#----------------------------------------
# Output is an array of text that is somehow associated with the production
# of varies files that need to be created. Note, these are generally ISO
# or Latin-1 based files, so different character sets will not work, like
# with WinTitle. So, use the ISO-Latin equivalents.
#----------------------------------------

# the header on the auto-gernated .c file.
set Output(header,c) /*
 This file is auto-generated. Do
 not modify. */

# for a fortran file, use "header,f". however, someone else write the bloody
# code. I have no clue what this should even be.
set Output(header,f) "A fortran comment."
# The Main Menu system
#
#
# these are done like: MenuTxt(menu,option). I suppose you really should
# do (menu,option,option,...), but we really don't have that many menus,
# so do it the simple way.
#
240 set MenuTxt(file) "File"
set MenuTxt(file,vars) "Show Variables"
set MenuTxt(file,compile) "Compile"
set MenuTxt(file,integrate) "Integrate"
set MenuTxt(file,exit) "Exit ParInt"

set MenuTxt(method) "Method"
set MenuTxt(method,determin) "Deterministic"
set MenuTxt(method,montecarlo) "Monte-Carlo"
set MenuTxt(method,adap) "Adaptive"
set MenuTxt(method,nadap) "Non-Adaptive"
set MenuTxt(method,prodint) "Product Integration"
set MenuTxt(method,locpq) "Local PQ -- No LB"
set MenuTxt(method,locpqlb) "Local PQ With LB"
set MenuTxt(method,globpq) "Global PQ"
set MenuTxt(method,seq) "Seq. of Rules"
set MenuTxt(method,pair) "Simple/Pair Rule"
set MenuTxt(method,user) "User's Rule"

set MenuTxt(system) "System"
set MenuTxt(system,pvm) "PVM"
set MenuTxt(system,ncube) "nCUBE"
set MenuTxt(system,mpi) "MPI"
set MenuTxt(system,seq)  "Sequential"
set MenuTxt(system,numproc)  "Number Processes"
set MenuTxt(system,distopts)  "Options"
set MenuTxt(system,pvmcon)  "Run PVM console"

set MenuTxt(lang)  "Language"
set MenuTxt(lang,c)  "C"
set MenuTxt(lang,f)  "Fortran"

set MenuTxt(func)  "Function"
set MenuTxt(func,choose)  "Choose Function"
set MenuTxt(func,limits)  "Limits of Integration"

set MenuTxt(help)  "Help"
set MenuTxt(help,helpsys)  "Help System"
set MenuTxt(help,about)  "About..."

gui/Limits.tcl

1  
#
#
#
# RCS:  File: $RCSfile$
# Revision: $Revision$
# Last Changed by $Author$ on $Date$
# Currently Edited by: $Locker$
# ToDo:
# PseudoBugs:
10  # Bugs:
proc LimitsWindow {} {
    global B Config WinTitle

    set w .limits ;# set a var to save typing
    if { ![winfo exists $w] } {
        raise $w
    } else {
        toplevel $w
        wm title $w $WinTitle(limits)
    }

    frame $w.bottom
    frame $w.bottom.but
    frame $w.bottom.but.limits

    # general buttons
    button $w.bottom.but.dis -textvariable B(dismiss) -command {destroy .limits}
    pack $w.bottom.but.dis -side right -anchor e

    # limits area
    label $w.bottom.but.limits.txt -textvariable B(limits)
    radiobutton $w.bottom.but.limits.def -textvariable B(defaultlimits) -variable Config(limits) -value default
    radiobutton $w.bottom.but.limits.use -textvariable B(userlimits) -variable Config(limits) -value user
    pack $w.bottom.but.limits.txt $w.bottom.but.limits.def $w.bottom.but.limits.use -side top -anchor w
    pack $w.bottom.but.limits -side left -fill y -anchor w
pack $w.bottom.but -side bottom -fill x
pack $w.bottom -side bottom -fill x

#pack $w.f -expand true -padx 5 -pady 5 -fill both
#pack $w -expand true -padx 5 -pady 5 -fill both
}

};# end Limits Window

gui/Main.tcl

# Program for GUI
#
#
# RCS: File: $RCSfile$
# Revision: $Revision$
# Last Changed by $Author$ on $Date$
# Currently Edited by: $Locker$
# ToDo:

# PseudoBugs:
# Bugs:
# Log:
# 9508XX Aman: wrote and did many things.
# 9601XX Jay: re-wrote and deleted many things. :)
# 960501 Jay: added the new menuing toolset and delete the old ones.
# 960118 Jay: removed last remnants of old menu system (loads much faster now)
# and put in old_main.tcl with the old stuff. All of Aman’s original
# code has now been removed.

#-------------------------------
# Main program starts
#-------------------------------
#
# ToDo:
# PseudoBugs:
# Bugs:
# Log:
#
proc Main {} {
    global WinTitle MenuTxt
    wm title . $WinTitle(main)
    #wm geometry . +200+200 ;# To set the position of window
    wm minsize . 100 50 ;# To make window resizeable

    #-----------------------------
    # Code for main window
    #-----------------------------

    frame .main -bg black -relief raised -borderwidth 1 ;# -width 25c -height 25c
    pack .main -fill both

    # |----------------------.main [frame]----------------------|

    #-----------------------------
    # End of code
    #-----------------------------
[frame .main.menu -bg gray50 -relief raised -borderwidth 1 ;# -width 100
frame .main.project -bg gray50
frame .main.query -bg gray50 -relief raised -borderwidth 1 ;# -width 80
pack .main.menu .main.project .main.query -expand 1 -fill both

# The entire MenuXXXX set of commands is described in Welsh's
# tcl/tk book near page 180.
# MenuSetup .main.menu.m

# Creating options headings
#-----------------------------------
# File
---

# Menu file $MenuTxt(file)
MenuCommand file $MenuTxt(file,vars) UpdateConfigDisplay
MenuCommand file $MenuTxt(file,compile) CompileProgram
MenuCommand file $MenuTxt(file,integrate) RunParallel
MenuCommand file $MenuTxt(file,exit) {destroy .}

# Method
#---

Menu method $MenuTxt(method)
MenuCascade method determin $MenuTxt(method,determin)
MenuRadio method $MenuTxt(method,montecarlo) method 3

MenuCascade determin adap $MenuTxt(method,adap)
MenuCascade determin nadap $MenuTxt(method,nadap)
MenuRadio determin $MenuTxt(method,prodint) method 31

#MenuCascade adap qlb "Local PQ"
#MenuRadio qlb "No LB" method 21
#MenuRadio qlb "LB" method 22

MenuRadio adap $MenuTxt(method,locpq) method 21
MenuRadio adap $MenuTxt(method,locpqlb) method 22
MenuRadio adap $MenuTxt(method,globpq) method 23

MenuRadio nadap $MenuTxt(method,seq) method 21
MenuRadio nadap $MenuTxt(method,pair) method 23
MenuRadio nadap $MenuTxt(method,user) method 24
#-------------------
# Distributed System
#-------------------

Menu system $MenuTxt(system)
MenuRadio system $MenuTxt(system,pvm) Config(system)
pvm {StatusLine $StatusMsg(main,system) "PVM" }
MenuRadio system $MenuTxt(system,ncube) Config(system)
ncube {MessageBox $TextMsg(main,ncube); set Config(system) pvm ; StatusClear }
MenuRadio system $MenuTxt(system,mpi) Config(system)
mpi {MessageBox $TextMsg(main,mpi); set Config(system) pvm ; StatusClear }
MenuRadio system $MenuTxt(system,seq) Config(system)
seq {MessageBox $TextMsg(main,seq); set Config(system) pvm ; StatusClear }
MenuCommand system $MenuTxt(system,numproc) SetNumProc
MenuCascade system distopts $MenuTxt(system,distopts)

MenuCommand distopts $MenuTxt(system,pvmcon) { StatusLine
$StatusMsg(main,pvmconsole); exec $Config(xterm) -e
$Config(pvmconsole) & }

#-----------------------
# Language Selection
#-----------------------

Menu lang $MenuTxt(lang)
MenuRadio lang $MenuTxt(lang,c) Config(proglang)
c {StatusLine $StatusMsg(main,language) "C" }
MenuRadio lang $MenuTxt(lang,f) Config(proglang)
f {MessageBox $TextMsg(main,fortran); set Config(proglang) c;
StatusClear }

140  #-----------------------
    # Function
    #-----------------------
Menu  func  $MenuTxt(func)
MenuCommand  func  $MenuTxt(func,choose) choosefunc
MenuCommand  func  $MenuTxt(func,limits) LimitsWindow

150  #-----------------------
    # Help
    #-----------------------
Menu  help  $MenuTxt(help)
MenuCommand  help  $MenuTxt(help,helpsys) HelpSystem
MenuSeparator  help
MenuCommand  help  $MenuTxt(help,about) AboutParInt

160  label .main.status -justify left -textvariable Status
    pack .main.status -side bottom -fill x

#-----------------------
# Set Font Credits
#-----------------------
set font -adobe-helvetica-bold-r-***-190-***-***-**
set small -adobe-helvetica-medium-r-***-100-***-***-**
set middlebg Papayawhip

frame .main.project.title -bg $middlebg
#frame .main.project.developed -bg $middlebg

label .main.project.title.name -bg $middlebg \
- text "The ParInt Project" -font $font \
# label .main.project.title.name -bg $middlebg \
- text "Parallel and Distributed Adaptive Integration" -font $font \
#label .main.project.title.alias -bg $middlebg -text "a.k.a. PARINT"
# label .main.project.title.p1 -bg $middlebg -font $small -text "presented by"
label .main.project.title.p1 -bg $middlebg -font $small -text "created by"
# label .main.project.title.p2 -bg $middlebg -text "Jay Ball, Elise deDoncker"
# label .main.project.title.p3 -bg $middlebg -text "Patricia Ealy, Alan Genz & Ajay Gupta"
# label .main.project.title.p4 -bg $middlebg -text "Western Michigan University"

label .main.project.title.p2 -bg $middlebg -text "Elise deDoncker,"
label .main.project.title.p3 -bg $middlebg -text "Alan Genz & Ajay Gupta"

# pack .main.project.title.name .main.project.title.p1 .main.project.title.p2 \
# .main.project.title.p3 .main.project.title.p4 -fill both -anchor center
pack .main.project.title.name .main.project.title.p1 .main.project.title.p2 \
# .main.project.title.p3 -fill both -anchor center

# pack .main.project.title.name .main.project.title.alias -fill both -anchor center
#label .main.project.developed.announce -bg $middlebg -text "Developed by:"
#label .main.project.developed.elise -bg $middlebg -text "Elise de Doncker" -font $font
#label .main.project.developed.alan -bg $middlebg -text "Alan Genz" -font $font
#label .main.project.developed.ajay -bg $middlebg -text "Ajay Gupta" -font $font

frame .main.project.logoleft -bg $middlebg
frame .main.project.logoright -bg $middlebg
LogoFrame .main.project.logoleft
LogoFrame .main.project.logoright
pack .main.project.logoleft -side left -fill y -anchor center
pack .main.project.logoright -side right -fill y -anchor center

pack .main.project.title -anchor center -fill both
#pack .main.project.title .main.project.developed -pady 1 -fill both
#pack .main.project.developed.announce .main.project.developed.elise
    .main.project.developed.alan .main.project.developed.ajay -fill both

} ;# end of main

gui/MessageBox.tcl

# A simple message box
#
# RCS: File: $RCSfile$
#     Revision: $Revision$
#     Last Changed by $Author$ on $Date$
#     Currently Edited by: $Locker$
# ToDo:
# PseudoBugs:
 proc MessageBox { m } {
   global B ScreenCenterX ScreenCenterY WinTitle

   # you should just add a new box and kill the old one. Maybe do a while
   # loop and create .message1,2,3... but, someone else can add that.
   if { [winfo exists .message] } {
      destroy .message
   }
   toplevel .message
   wm geometry .message +$ScreenCenterX+$ScreenCenterY
   wm title .message $WinTitle(message)

   # for some reason, using a textvariable does not work when passed in the args....
   # so fuck it. just use text and it won't change with the language.
   #label .message.title -textvariable m
message .message.title -justify center -relief ridge -aspect 800 -text $m

button .message.dismiss -textvariable B(dismiss) -command { destroy .message }

pack .message.title -side top -fill x
pack .message.dismiss

} ;# end MessageBox

gui/NumProc.tcl

#
# This sets the number of processors in a fancy gui way.
#
# RCS: File: $RCSfile$
#     Revision: $Revision$
#     Last Changed by $Author$ on $Date$
#     Currently Edited by: $Locker$
# # ToDo:
# # PseudoBugs:
# # Bugs:
# # Log:
# #
# #
# #
# #
# #
# #
# #
# #
# #
# # ToDo:
# # PseudoBugs:
proc SetDim {val} {
    global Config
    set Config(numproc) [expr int( pow(2,$val) )]
}

proc SetNumProc {} {
    global TextMsg B Config WinTitle
    if { [ winfo exists .numproc] } {
        raise .numproc
    } else {
        toplevel .numproc
        wm title .numproc $WinTitle(numproc)
        frame .numproc.f
        pack .numproc.f -padx 5 -pady 5
        label .numproc.f.l -textvariable ConfigLongDescName(numproc)
        label .numproc.f.r -textvariable Config(numproc)
scale .numproc.f.s -from 0 -to 12 -length 300 -variable Config(dim) \
   -orient horizontal -tickinterval 2 -showvalue true -command SetDim \
   -label $TextMsg(dim,use)
message .numproc.f.m -relief ridge -justify center -aspect 750  \
   -textvariable TextMsg(dim,warn)
button .numproc.f.b -textvariable B(dismiss) -command {destroy .numproc}
pack .numproc.f.b -side bottom
pack .numproc.f.m .numproc.f.s -side bottom -fill x
pack .numproc.f.l .numproc.f.r -side left -fill x
}

}
# Tests to see if pvmd is running (hopefully). returns 1 if it is, 0 otherwise
#
# ToDo:
# PseudoBugs:
# Bugs:
20 # Log:
#
proc TestPVMRun { } {
    global Config

    if [file exists "/tmp/pvmd.$Config(uid)"] {
        return 1
    } else {
        return 0
    }
}

#
# Sets the pvm arch var. returns 1 on error or 0 for ok.
#
# ToDo:
# PseudoBugs:
# Bugs:
# Log:
#
40 proc SetPVMArch { } {
    global Config

    if [file exists $Config(pvmdir)/lib/pvmgetarch] {
        set Config(pvmarch) [exec $Config(pvmdir)/lib/pvmgetarch]
puts $TextMsg(init,nopvmgetarch)
    set Config(pvmarch) $ErrorMsg(pvmarch)
}
}

gui/Run.tcl

1 #
   # The File that actually runs the program
#
# RCS:  File: $RCSfile$
#    Revision: $Revision$
#    Last Changed by $Author$ on $Date$
#    Currently Edited by: $Locker$
# ToDo:
# PseudoBugs:
10 # Bugs:
# Log:
#950524 Jay: creation
#

# # #
# #
# #
# ToDo:
20 # PseudoBugs:
# Bugs:
# Log:
# proc RunLogClear {} {
#   global Runlog

#   $Runlog(log) delete 1.0 end
#
#}

# # # #
# # ToDo:
# # PseudoBugs:
# # Bugs:
# # Log:
# #
# proc RunLogSave {} {
#   global Runlog Config

# set filename [pathexpand func $Config(runlogfilename)]
set filename [pathexpand func $Config(runlogfilename)]
if [catch {open $filename w 0644} outfp] {
    puts "error, cannot open file"
}
puts $outfp [$Runlog(log) get 1.0 end],
close $outfp
}

# # # #
# # ToDo:
# PseudoBugs:
# Bugs:
# Log:

proc RunlogWrap {} {
    global Runlog

    if { $Runlog(wrap) == "word" } {
        pack forget .runlog.text.horizscoll
    } else {
        pack .runlog.text.horizscroll -fill x
    }
}

# Todo:
# PseudoBugs:
# Bugs:
# Log:

proc CompileAndRun {} {
    global CompileStuff Runlog

    if { [winfo exists .compilelog] } {
        wm deiconify .compilelog
    }
    CompileProgram
$CompileStuff(but) invoke
wm iconify .compilelog
$Runlog(but) invoke
}

# #

# ToDo:
# PseudoBugs:
# Bugs:
# Log:
#
proc CreateRunCommandLine {} {
    global Runlog Dir Config

    # you cannot use a makefile. how do you pass parameters?
    #set Runlog(command) "make run.$Config(system)"
    set Runlog(command) "$Dir(home)/$Config(system)/parent --nodes $Config(numproc)"

    if { [string compare $Config(compact) yes ] == 0 } {
        append Runlog(command) " --compact"
    }
    if { [string compare $Config(compact) no ] == 0 } {
        append Runlog(command) " --non-compact"
    }
}
# ToDo:
# PseudoBugs:
# Bugs:
# Log:

proc ParallelRun {} {
    global Runlog B Config Dir ErrorMsg

    #CreateSubstitutedFile $Config(intfuncname) $Config(func) $Config(funcfile)

    CreateRunCommandLine

    if [catch {open "$Runlog(command) & cat" Runlog(input)}] {
        #$Runlog(log) insert end $Runlog(command)
        $Runlog(log) insert end $Runlog(input)
        # $Runlog(log) insert end
        # "$B(errorc)$ErrorMsg(externalcommand,A) 'Runlog(command)'.
        # $ErrorMsg(externalcommand,B)"
        # $Runlog(log) insert end
        "$ErrorMsg(externalcommand,A)$Runlog(command)$ErrorMsg(externalcommand,B)"
        $Runlog(log) see end
    } else {
        fileevent $Runlog(input) readable ParallelLog

        # we don't want to print the command line if the format is for "compact"
        if { [string compare $Config(compact) yes ] != 0 } {
            $Runlog(log) insert end $Runlog(command)
        }
    }
set Runlog(text) $B(stop)
$Runlog(but) config -command ParallelStop
}
} ;# end ParallelRun

150  # Read and log output from the program
    #
    #
    #
    # ToDo:
    # PseudoBugs:
    # Bugs:
    # Log:
    #
    proc ParallelLog {} {
160    global Runlog
        if [eof $Runlog(input)] {
            ParallelStop
        } else {
            gets $Runlog(input) line
            $Runlog(log) insert end $line
            $Runlog(log) see end
        }
    }
}

170  # Stop the program and fix up the button
    #
    #
# ToDo:
# PseudoBugs:
# Bugs:
# Log:
#
proc ParallelStop {} {
    global Runlog B
    catch {close $Runlog(input)}
    set Runlog(text) $B(run)
    $Runlog(but) config -command ParallelRun
}

# RunParallel
#
# ToDo:
# PseudoBugs:
# Bugs:
# Log:
#950524 Jay: creation
#9701?? jay: added compact, wrap, clear, save, compile+run
proc RunParallel {} {
    global Config B Runlog WinTitle

    # place check for if it was compiled here!
set w .runlog

if { [winfo exists $w] } {
    raise $w
} else {
    toplevel $w
    wm title $w $WinTitle(integrate)
    # if it is not set, default to a word wrap mode, otherwise, keep previous value
    if { ! [info exists Runlog(wrap)] } {
        set Runlog(wrap) word
    }

    # Create a frame for buttons and entry.
    frame $w.but -borderwidth 5
    pack $w.but -side bottom -fill x
    frame $w.but.check

    # Create the command buttons.
    button $w.but.dis -textvariable B(dismiss) -command {destroy .runlog}
    set Runlog(but) [button $w.but.run -width $B(runstopwidth)
    -textvariable Runlog(text) -command ParallelRun]
    button $w.but.save -textvariable B(save) -command RunLogSave
    button $w.but.clear -textvariable B(clear) -command RunLogClear
    button $w.but.compileandrun -textvariable B(compileandrun) -command CompileAndRun

    pack $w.but.dis $w.but.clear $w.but.save $w.but.run $w.but.compileandrun -side right
# the "compact format" check button
checkbutton $w.but.check.compact -textvariable B(compact)
-variable Config(compact) -onvalue yes -offvalue no

# the "word wrap"
checkbutton $w.but.check.wrap -textvariable B(wrap) -variable Runlog(wrap) -onvalue word -offvalue none

pack $w.but.check.compact $w.but.check.wrap -anchor w -side top
pack $w.but.check -side left

# Set up keybinding equivalents to the buttons
#bind $w.t.cmd <Return> ParallelRun
bind $w <Control-c> ParallelStop
#focus $w.t.cmd

# Create a text widget to log the output
frame $w.text
set Runlog(log) [text $w.text.log -width 40 -height 10 \
-borderwidth 2 -relief raised -setgrid true -wrap $Runlog(wrap) \
-yscrollcommand "$w.text.vertscroll set"
scrollbar $w.text.vertscroll -command "$w.text.log yview"
#scrollbar $w.text.horizscroll -command "$w.text.log xview"
#pack $w.text.horizscroll -side bottom -fill x
pack $w.text.vertscroll -side right -fill y
pack $w.text.log -side left -fill both -expand true
pack $w.text -side top -fill both -expand true

set Runlog(text) $B(run)
In theory, we use a function rather than just to set the Status variable so that in the future, we can make the status line a list box or anything else that we desire. Who says that I don’t think ahead.
proc StatusLine { args } {
    global Status
    # we do the join so that people can do multiple args
    set m [join $args ""]
    set Status $m
}

proc StatusClear {} {
    global Status
    set Status ""
}

# Log:
#
# RCS: File: $RCSfile$
# Revision: $Revision$
# Last Changed by $Author$ on $Date$
# Currently Edited by: $Locker$
# ToDo:
# PseudoBugs:
# Bugs:
# Log:
# 

# TextPrint
# 
# nicely prints text to the current screen width

# ToDo:
#   * this does not handle tabs or newlines etc. this needs to
#   account for them by replacing them with spaces. this
#   would make for a more desirable behaviour. it does
#   condense multiple tabs and spaces into one space, via the
#   multiple args in the parameters. if you quota in \t or \n, etc,
#   it will not handle nicely (it'll work, but look shitty).
#   * use termcap or terminfo to get width
#   * reset the width variable via SIGRESIZE when changing

# PseudoBugs:
# Bugs:
# Log:
# 

proc TextPrint { args } { 
    global Config

    # set the width to format with. This could change if we could
    # detect any SIGRESIZE messages, etc...
    # we use "-2" since we don't want it to be == and because tcl
    # defaults to 0-79 and not 1-80 like terminals report
set w [expr $Config(textwidth) - 1]

# make the args into one variable.
set m [string trim [join $args " "]]

while { [string length $m] > $w } {
    set t $w
    while { [string index $m $t] != " " } {
        incr t -1
        if { $t == -1 } {
            break
        }
    }
    if { $t != -1 } {
        set start [string wordstart $m $t]
        incr start -1
        puts [string range $m 0 $start]
        incr start
        set m [string trim [string range $m $start end]]
    } else {
        ;# account for no spaces
        puts [string range $m 0 [expr $w - 1]]
        set m [string trim [string range $m $w end]]
    }
} ;# end of while.

# if anything is left after the loops, print it
if { [string length $m] > 0 } {
    puts $m
}
}
gui/Var.tcl

# All general variable information should be set here. This is loaded before
# any of the procedure library files.
#
# RCS: File: $RCSfile$
# Revision: $Revision$
# Last Changed by $Author$ on $Date$
# Currently Edited by: $Locker$
# ToDo:
# PseudoBugs:
# Bugs:
# Log:
#
# set the help lib directory
#set hlib [pwd]/help
#set Dir(ParIntHome) /local/odin2/jaystuff/ParInt
#set Dir(ParIntHome) /home/jay/ParInt/ParInt
set Dir(ParIntHome) /home/res/parint/ParInt.EXPERIMENTAL
set Dir(help) $Dir(ParIntHome)/help
set Dir(home) $env(HOME)/ParInt
set Dir(func) $Dir(home)/func

# user name of the person/account that owns parint
set Config(paradmin) parint

set Make make
# see the GetUid command for all options.
set Config(uidsearch) sysv_id

# the level of debugging to show the user. lower=more info higher=less
set Config(debugmessagelevel) 0

set Config(xterm) xterm
set Config(pvmdir) $env(HOME)/pvm3
set Config(pvmconsole) $Config(pvmdir)/lib/pvm

set Config(runlogfile) "ParInt.runlog"
set Config(compact) "yes"

set ScreenCenterX [expr int([winfo screenwidth .] / 2) - 100]
set ScreenCenterY [expr int([winfo screenheight .] / 2) - 100]

# the width of the text screen. This is the default value, which should be obtained by
# more civil ways. see TextPrint procedure.
set Config(textwidth) 60

gui/html_library.tcl

# 960901 Jay: You must disable the HMLink_callback function in here, because
# the auto-library load will use this version and not the one that
# renders the document!!!!
# Note this if you upgrade this file to version 0.3!!!!
#
# Simple HTML display library by Stephen Uhler (stephen.uhler@sun.com)
# Copyright (c) 1995 by Sun Microsystems
# Version 0.1 Thu Jul 20 09:06:28 PDT 1995

See the file "license.terms" for information on usage and redistribution
of this file, and for a DISCLAIMER OF ALL WARRANTIES.

To use this package, create a text widget (say, .text)
and set a variable full of html, (say $html), and issue:

\begin{verbatim}
HMinit_win .text
HMparse_html $html "HMrender .text"
\end{verbatim}

You also need to supply the routine:

\begin{verbatim}
proc HMlink_callback {win href} { ... }
\end{verbatim}

\begin{itemize}
\item \texttt{win}: The name of the text widget
\item \texttt{href}: The name of the link
\end{itemize}

which will be called anytime the user "clicks" on a link.
The supplied version just prints the link to stdout.

In addition, if you wish to use embedded images, you will need to write

\begin{verbatim}
proc HMsset_image {handle src}
\end{verbatim}

\begin{itemize}
\item \texttt{handle}: an arbitrary handle (not really)
\item \texttt{src}: The name of the image
\end{itemize}

which calls

\begin{verbatim}
HMgot_image $handle $image
\end{verbatim}

with the TK image.

To return a "used" text widget to its initialized state, call:

\begin{verbatim}
HMreset_win .text
\end{verbatim}

See "sample.tcl" for sample usage
# mapping of html tags to text tag properties
# properties beginning with "T" map directly to text tags

array set HMtag_map {
  a {Tlink link}
  b {weight bold}
  blockquote {style i indent 1 Trindent rindent}
  bq {style i indent 1 Trindent rindent}
  cite {style i}
  code {family courier}
  dfn {style i}
  dir {indent 1}
  dl {indent 1}
  em {style i}
  h1 {size 24 weight bold}
  h2 {size 22}
  h3 {size 20}
  h4 {size 18}
  h5 {size 16}
  h6 {style i}
  i {style i}
  kbd {family courier weight bold}
  menu {indent 1}
  ol {indent 1}
  pre {fill 0 family courier Tnowrap nowrap}
  samp {family courier}
  strong {weight bold}
  tt {family courier}
u {Tunderline underline}
ul {indent 1}
var {style i}

# These are in common(?) use, but not defined in html2.0

array set HMtag_map {
center {Tcenter center}
strike {Tstrike strike}
}

80 # initial values

set HMtag_map(hmstart) {
    family times weight medium style r size 14
    Tcenter "" Tlink "" Tnowrap "" Tunderline ""
    list list
    fill 1 indent "" counter 0 adjust 0
}

# html tags that insert white space

90 array set HMinsert_map {
    blockquote "\n\n" /blockquote "\n"
    br "\n"
    dd "\n" /dd "\n"
    dl "\n" /dl "\n"
    dt "\n"
    form "\n" /form "\n"
# tags that are list elements, that support "compact" rendering

array set HMlist_elements {
    ol 1  ul 1  menu 1  dl 1  dir 1
}

# initialize the window and stack state

proc HMinit_win {win} {
    upvar #0 HM$win var
    HMinit_state $win
    $win tag configure underline -underline 1
    $win tag configure center -justify center
    $win tag configure nowrap -wrap none
$win$ tag configure rindent -rmargin $var(S\_tab)c$
$win$ tag configure strike -overstrike 1
$win$ tag configure mark -foreground red ;# list markers
$win$ tag configure list -spacing1 3p -spacing3 3p ;# regular lists
$win$ tag configure compact -spacing1 0p ;# compact lists
$win$ tag configure link -foreground blue ;# hypertext links
HMset_indent $win$ $var(S\_tab)$
$win$ configure -wrap word

# for horizontal rules
$win$ tag configure thin -font [HM\_font times 2 medium r]
$win$ tag configure hr -relief sunken -borderwidth 2 -wrap none \
    -tabs [winfo width $win$]
bind $win$ <Configure> {%W tag configure hr -tabs %w}

# generic link enter callback
$win$ tag bind link <1> "HMlink\_hit $win$ %x %y"

# set the indent spacing (in cm) for lists
# TK uses a "weird" tabbing model that causes \t to insert a single
# space if the current line position is past the tab setting
proc HMset_indent {win cm} {
    set tabs [expr $cm / 2.0]
    $win$ configure -tabs ${tabs}c
    foreach i {1 2 3 4 5 6 7 8 9} {
        set tab [expr $i * $cm]
        $win$ tag configure indent$i -lmargin1 ${tab}c -lmargin2 ${tab}c \
-tabs "{expr $tab + $tabs}c {expr $tab + 2*$tabs}c"

}

# reset the state of window - get ready for the next page
# remove all but the font tags

proc HMreset_win {win} {
    regsub -all { +[-L ][- ]*} " ["$win tag names] " {} tags
    catch "$win tag delete $tags"
    eval $win mark unset [$win mark names]
    $win delete 0.0 end
    $win tag configure hr -tabs [winfo width $win]
    HMinit_state $win
}

# initialize the window's state array
# Parameters beginning with S_ are NOT reset
# adjust_size: global font size adjuster
# unknown: character to use for unknown entities
# tab: tab stop (in cm)
# stop: enabled to stop processing
# update: how many tags between update calls
# tags: number of tags processed so far
# symbols: Symbols to use on un-ordered lists

proc HMinit_state {win} {
    upvar #0 HMS$win var
    array set tmp [array get var S_*]
    catch {unset var}
array set var {
    stop 0
    tags 0
    fill 0
    list list
    S_adjust_size 0
    S_tab 1.0
    S_unknown \xb7
    S_update 10
    S_symbols 0*=+-o\xb7\xb0>:\xb7
}
array set var [array get tmp]
array set HMparam_map {
    -update S_update
    -tab S_tab
    -unknown S_unknown
    -stop S_stop
    -size S_adjust_size
    -symbols S_symbols
}
proc HMset_state {win args} {
    upvar #0 HM$win var
    global HMparam_map
set bad 0
if {([catch {array set params $args}]} {return 0}
foreach i [array names params] {
    incr bad [catch {set var($HMparam_map($i)) $params($i)}]
}
return [expr $bad == 0]

# manage the display of html
# this gets called for every html tag
# win: The name of the text widget to render into
# tag: The html tag (in arbitrary case)
# not: a "/" or the empty string
# param: The un-interpreted parameter list
# text: The plain text until the next html tag

proc HMrender {win tag not param text} {
    upvar #0 HM$win var
    if {$var(stop)} return
    global HMtag_map HMinsert_map HMlist_elements
    set tag [string tolower $tag]
    set text [HMmap_esc $text]
    # manage compact rendering of lists
    if {[info exists HMlist_elements($tag)]} {
        set list "list [expr {[HMextract_param $param compact] ? "compact" : "list"]}"
    } else {
        set list ""
# adjust tag state
catch {HMstack $win $not "$HMtag_map($tag) $list"}

# insert white space (with current font)
# adding white space can get a bit tricky. This isn't quite right
set bad [catch {[$win insert end $HMinsert_map($not$tag) "space $var(font)"]}]
if {!$bad && [lindex $var(fill) end]} {
    set text [string trimleft $text]
}

# to fill or not to fill
if {[lindex $var(fill) end]} {
    set text [HMzap_white $text]
}

# do any special tag processing
catch {HMtag_$not$tag $win $param text} msg

# debugging only (code will not be here if no debugging was set)

# add the text with proper tags
set tags [HMcurrent_tags $win]
$win insert end $text $tags

# We need to do an update every so often to insure interactive response.
# This can cause us to re-enter the event loop, and cause recursive
# invocations of HMrender, so we need to be careful.
if {!([incr var(tags)] % $var(S_update))} {
html tags requiring special processing
_procs of the form HMtag_<tag> or HMtag_</tag> get called just before
_the text for this tag is displayed.
win: The name of the text widget to render into
param: The un-interpreted parameter list
text: A pass-by-reference name of the plain text until the next html tag
Tag commands may change this to affect what text will be inserted
next.

proc HMtag_title {win param text} {
    upvar $text data
    wm title [winfo toplevel $win] $data
    set data ""
}

proc HMtag_hr {win param text} {
    $win insert end "\n" space "\n" thin "\t" "thin hr" "\n" thin
}

# list element tags

proc HMtag_ol {win param text} {
    upvar #0 HM$win var
    set var(count$var(level)) 0
}
proc HMtag_ul {win param text} {
    upvar #0 HM$win var
    catch {unset var(count$var(level))}
}

proc HMtag_menu {win param text} {
    upvar #0 HM$win var
    set var(menu) ->
    set var(compact) 1
}

proc HMtag_/menu {win param text} {
    upvar #0 HM$win var
    catch {unset var(menu)}
    catch {unset var(compact)}
}

proc HMtag_dt {win param text} {
    upvar #0 HM$win var
    upvar $text data
    set level $var(level)
    incr level -1
    $win insert end "$data" "hi [lindex $var(list) end] indent$level $var(font)"
    set data {}
set x [string index $var(S_symbols)+++-+-+"$level]
catch {set x [incr var(count$level)]}
catch {set x $var(menu)}
$win insert end \t$x\t "mark [lindex $var(list) end] indent$level $var(font)"
}

# hypertext links.

proc HMtag_a {win param text} {
  upvar #0 HM$win var
  if {![HMextract_param $param href]} {
    set var(Tref) L:$href
    HMlink_setup $win $href
  }
}

proc HMtag_/a {win param text} {
  upvar #0 HM$win var
  catch {unset var(Tref)}
}

# This interface is subject to change

# Most of the work is getting around a limitation of TK that prevents
# setting the size of a label to a widthxheight in pixels
#
# Images have the following parameters:
#   align: top,middle,bottom
#   alt:  alternate text
#   ismap: A clickable image map
# src: The URL link
# Netscape supports
# width: A width hint (in pixels)
# height: A height hint (in pixels)
# border: The size of the window border

proc HMtag_img {win param text} {
    upvar #0 HM$win var

    # get alignment
    array set align_map {top top middle center bottom bottom} ;# The spec isn't clear what the default should be
    HMextract_param $param align
    catch {set align $align_map([string tolower $align])}

    # get alternate text
    set alt "<image>"
    HMextract_param $param alt
    set alt [HMmap_esc $alt]

    # get the border width
    set border 1
    HMextract_param $param border

    # see if we have an image size hint
    # If so, make a frame the "hint" size to put the label in
    # otherwise just make the label
    set item $win.$var(tags)
    catch {destroy $item}
    if {([HMextract_param $param width] && [HMextract_param $param height])} {
frame $item -width $width -height $height
pack propagate $item 0
set label $item.label
label $label
pack $label -expand 1 -fill both
} else {
    set label $item
    label $label
}

$label configure -relief ridge -fg orange -text $alt
catch {$label configure -bd $border}
$win window create end -align $align -window $item -pady 2

# add in all the current tags (this is overkill)
set tags [HMcurrent_tags $win]
foreach tag $tags {
    $win tag add $tag $item
}

# set imagemap callbacks
if {[HMextract_param $param ismap]} {
    set link ""
    regsub -all {\^L\*L:([^ ]*)\*} $tags {\1} link
    global HMevents
    foreach i [array names HMevents] {
        bind $label <$i> "%W configure $HMevents($i)"
    }
    bind ismap <ButtonRelease-1> "HMlink_callback $win $link?x,y"
    bindtags $label "ismap [bindtags $label]"
# now callback to the application
set src ""
HMextract_param $param src
HMset_image $label $src

# The app needs to supply one of these
proc HMset_image {handle src} {
    puts "Found an image <$src> to put in $handle"
    HMgot_image $handle "can't get
$src"
}

# When the image is available, the application should call back here.
# If we have the image, put it in the label, otherwise display the error
# message. If we don't get a callback, the "alt" text remains.
# if we have a clickable image, arrange for a callback

proc HMgot_image {win image_error} {
    # if we're in a frame turn on geometry propagation
    if {[info exists $win]} {
        pack propagate [winfo parent $win] 1
    }
    if {[catch {$win configure -image $image_error}]} {
        $win configure -image {}
        $win configure -text $image_error
    }
}
# Sample hypertext link callback routine - should be replaced by app
# This proc is called once for each <A> tag.
# Applications can overwrite this procedure, as required, or
# replace the HMevents array

# win: The name of the text widget to render into
# href: The HREF link for this <a> tag.

array set HMevents {
  Enter   {-borderwidth 2 -relief raised }
  Leave   {-borderwidth 0 -relief flat }
  1       {-borderwidth 2 -relief sunken}
  ButtonRelease-1 {-relief raised}
}

proc HMlink_setup {win href} {
  global HMevents
  foreach i [array names HMevents] {
    eval $win tag bind L:$href <$i> \n      {$win tag configure L:$href $HMevents($i)}

  }
}

# generic link-hit callback
# This gets called upon button hits on hypertext links
# Applications are expected to supply their own HMlink_callback routine
# win: The name of the text widget to render into
# x,y: The cursor position at the "click"

proc HMlink_hit {win x y} {
  set tags [$win tag names @$x,$y]
```tcl
regsub -all {\([\^L]*L:(\[\^ ]\)*\).\} $tags \{\1\} link
HMlink_callback $win $link

# replace this!
# win: The name of the text widget to render into
# href: The HREF link for this <a> tag.

#proc HMlink_callback {win href} {
#    puts "Got hit on $win, link $href"
#}

# extract a value from parameter list (this needs a re-do)
# returns "1" if the keyword is found, "0" otherwise
# param: A parameter list. It should already have been processed to
#        remove any entity references
# key:   The parameter name
# val:   The variable to put the value into (use key as default)

proc HMextract_param {param key {val ""}} {
    if {$val == ""} {
        upvar $key result
    } else {
        upvar $val result
    }

    # look for name=value combinations. Either (') or (") are valid delimiters
    if {
        [regsub -nocase [format {.*%s *= *"([\^"]\)*.*)} $key] $param \{\1\} value] ||
```
520 [regsub -nocase \[format \{.*s *= *'(\'\'\')*\} \$key\] \$param \{\1\} value\} ||
    [regsub -nocase \[format \{.*s *= *\([\-\+\ ]\)\}\} \$key\] \$param \{(\1)\} value\} ] {
    set result \$value
    return 1
}

# now look for valueless names
# I should strip out name=value pairs, so we don’t end up with "name"
# inside the "value" part of some other key word - some day

set bad \[-a-zA-Z\]+
if {\[regexp -nocase "$bad$key$bad" \$param\]} {
    return 1
} else {
    return 0
}

# These next two routines manage the display state of the page.

# push or pop tags to/from stack
# the current tag is the last item on the list

540 proc HMstack \{win push list\} {
    upvar #0 HMS\$win var
    array set tags $list
    if {$push == ""} {
        foreach tag [array names tags] {
            lappend var($tag) $tags($tag)
        }
    }
} else {
    foreach tag [array names tags] {
        set cnt [regsub { *-[ ]+$} $var($tag) {} var($tag)]
    }
}

# extract set of current text tags
# tags starting with T map directly to text tags

proc HMcurrent_tags {win} {
    upvar #0 HM$win var
    set font font
    foreach i {family size weight style} {
        set $i [lindex $var($i) end]
        append font : [set $i]
    }
    set xfont [HMx_font $family $size $weight $style $var(S_adjust_size)]
    catch {set $win tag configure $font -font $xfont} msg
    set indent [llength $var(indent)]
    incr indent -1
    lappend tags $font indent$indent
    foreach tag [array names var T*] {
        append tags " [lindex $var($tag) end]"
    }
    set var(font) $font
    set var(level) $indent
    return $tags
}
# generate an X font name
proc HMx_font {family size weight style {adjust_size 0}} {
    catch {incr size $adjust_size}
    return "-*-family-$family-$weight-$style-normal-*-{-size}{$size}0-*-*-*-*"
}

# Turn HTML into TCL commands
# html A string containing an html document
# cmd A command to run for each html tag found

proc HMparse_html {html {cmd HMtest_parse}} {
    regsub -all \{ $html \{&ob;\} html
    regsub -all \} $html \{&cb;\} html
    set w "\t\r\n" ;# white space
    proc HMcl x {return "\[$x\]"}
    set exp <(/?)([HMcl -$w]*\[HMcl -$w\]*(\[HMcl ->]*)?>
    set sub "\}\n\$cmd \{"\} \{"\} \{"\} \"\n    regsub -all $exp $html $sub html
    eval "$cmd hmstart {} {} \ $html \"
    eval "$cmd hmstart / {} {} \"
}

proc HMtest_parse {command tag slash text_after_tag} {
    puts "==> $command $tag $slash $text_after_tag"
}

# Convert multiple white space into a single space
proc HMzap_white {data} {
    regsub -all "\[ \t\r\n\]+" data " " data
return $data
}

# find HTML escape characters of the form &xxx;
proc HMmap_esc {text} {
    if {! [regexp & $text]} {return $text}
    regsub -all {([][\$\)])} $text {\\1} new
    regsub -all {&(\[0-9][0-9]?);?} $new {[format %c \1]} new
    regsub -all {&([- ;]+);?} $new {[HMdo_map \1]} new
    return [subst $new]
}

# convert an HTML escape sequence into character
proc HMdo_map {text {unknown ?}} {
    global HMesc_map
    set result $unknown
    catch {set result [HMesc_map($text)]}
    return $result
}

# table of escape characters (ISO latin-1 esc's are in a different table)
array set HMesc_map {
    lt <   gt >   amp &   quot "   copy \xa9
    reg \xae  ob \x7b  cb \x7d  nbsp \xa0
}

#############################################################
# ISO Latin-1 escape codes

array set HMesc_map {
  nbsp \xa0 iexcl \xa1 cent \xa2 pound \xa3 curren \xa4 yen \xa5 brvbar \xa6 sect \xa7 uml \xa8 copy \xa9 ordf \xa9 laquo \xab not \xac shy \xad reg \xae hibar \xaf deg \xb0 plusmn \xb1 sup2 \xb2 sup3 \xb3 acute \xb4 micro \xb5 para \xb6 middot \xb7 cedil \xb8 sup1 \xb9 ordm \xba raquo \xbb frac14 \xbc frac12 \xbd frac34 \xbe iquest \xbf Agrave \xc0 Aacute \xc1 Acirc \xc2 Atilde \xc3 Auml \xc4 Aring \xc5 AElig \xc6 Ccedil \xc7 Egrave \xc8 Eacute \xc9 Ecirc \xca Euml \xcb Iacute \xcd Icirc \xce Iuml \xcf ETH \xd0 Ntilde \xd1 Ograve \xd2 Oacute \xd3 Ocirc \xd4 Otilde \xd5 Ouml \xd6 times \xd7 Oslash \xd8 Ugrave \xd9 Uacute \xda Ucirc \xdb Uuml \xdc Yacute \xdd THORN \xde szlig \xdf agrave \xe0 aacute \xe1 acirc \xe2 atilde \xe3 auml \xe4 aring \xe5 aelig \xe6 ccedil \xe7 egrave \xe8 eacute \xe9 ecirc \xea euml \xeb igrave \xec icirc \xed iuml \xff eth \xf0 ntilde \xf1 ograve \xf2 oacute \xf3 ocirc \xf4 otilde \xf5 ouml \xf6 divide \xf7 oslash \xf8 ugrave \xf9 uacute \xfa ucirc \xfb uuml \xfc yacute \xfd thorn \xfe yuml \xff
}
gui/library.tcl

1  #
# The procedure for doing the libraryizing of the procs.
#
# this was stolen Welsh's book in the library chapter.
#
# RCS: File: $RCSfile$
# Revision: $Revision$
# Last Changed by $Author$ on $Date$
# Currently Edited by: $Locker$

# ToDo:
# PseudoBugs:
# You must actually own the files to run this proc. Thus, the
# average user will not be able to runs thses, and an error
# will occur and the program will harshly exit.

# Bugs:
# Log:
#

proc Library_UpdateIndex { libdir } {
    if ![file exists $libdir/tclIndex] {
        set doit 1
    } else {
        set age [file mtime $libdir/tclIndex]
        set doit 0
        foreach file [glob $libdir/*/tcl] {
            if {[file mtime $file] > $age} {
                set doit 1
                break
            }
        }
    }
    if { $doit } {
        
    }
}
auto_mkindex $libdir *.tcl

proc ScrolledListbox2 { parent args } { 
    frame $parent
    # Create listbox attached to scrollbars, pass thru $args
    eval {listbox $parent.list \
        -yscrollcommand [list $parent.sy set] \
        -xscrollcommand [list $parent.sx set]} $args
    scrollbar $parent.sy -orient vertical \
        -command [list $parent.list yview]
    # Create extra frame to hold pad and horizontal scrollbar
    frame $parent.bottom
    scrollbar $parent.sx -orient horizontal \

# Create padding based on the scrollbar width and border
set pad [expr [$parent.sy cget -width] + 2* \n    ([$parent.sy cget -bd] + \n    [$parent.sy cget -highlightthickness])]
frame $parent.pad -width $pad -height $pad

# Arrange everything in the parent frame
pack $parent.bottom -side bottom -fill x
pack $parent.pad -in $parent.bottom -side right
pack $parent.sx -in $parent.bottom -side bottom -fill x
pack $parent.sy -side right -fill y
pack $parent.list -side left -fill both -expand true
return $parent.list

# this is all commented out for some reason. do we need it anymore?
if 0 {
    frame $parent
    eval {listbox $parent.list \n        -yscrollcommand [list $parent.sy set] \n        -xscrollcommand [list $parent.pad.sx set]} $args
    scrollbar $parent.sy -orient vertical \n        -command [list $parent.list yview]
    # Create extra frame to hold pad and horizontal scrollbar
    frame $parent.pad
    scrollbar $parent.pad.sx -orient horizontal \n        -command [list $parent.list xview]
    # Create padding based on the scrollbar’s width
    set pad [expr [$parent.sy cget -width] + \n        2*([$parent.sy cget -bd] + [$parent.sy cget -highlightthickness])]
}
frame $parent.pad.it -width $pad -height $pad
# Arrange everything in the parent frame
pack $parent.pad -side bottom -fill x
pack $parent.pad.it -side right
pack $parent.pad.sx -side bottom -fill x
pack $parent.sy -side right -fill y
pack $parent.list -side left -fill both -expand true
}

$parent.sy

gui/logo.tcl

# Creates a canvas in the passed frame f with a bad attempt at the ParInt logo
# RCS: File: $RCSfile$
#    Revision: $Revision$
#    Last Changed by $Author$ on $Date$
#    Currently Edited by: $Locker$
# ToDo:
# PseudoBugs:
# Bugs:
# Log:
# 960321 Jay: initial creation
#

# Creates a canvas in the passed frame f that is a bad attempt at the ParInt logo
# ToDo:
# PseudoBugs:
# Bugs:
# Log:
# 960321 Jay: initial creation
# 960501 Jay: played with the color so it would stand out
# 960820 Jay: played with expand y to get it centered vertically, no luck..
#
proc LogoFrame { f {scale 1}} {
    #catch [frame $f]
    canvas $f.c -width 60 -height 100 -bg blue
    $f.c create line 10 25 10 75 -width 2 -capstyle round
    $f.c create line 20 25 20 75 -width 2 -capstyle round
    #$f.c create line 20 80 30 90 40 10 70 20 -width 3 -smooth true -splinesteps 20 -capstyle round
    $f.c create line 15 80 30 100 40 0 55 20 -width 2 -smooth true -splinesteps 30 -capstyle round
    pack $f.c -fill y -expand 1
} ;# end of LogoFrame

# ToDo:
# PseudoBugs:
# Bugs:
# Log:

guimiscfile.tcl
#
# getvalidline
# get the next line of a file that does not begin with a "#" and is not blank. It also trims spaces off both ends and returns the resulting line.
#
20
# use: getvalidline filepointer
#
# ToDo:
# PseudoBugs:
# Bugs:
# Bugs:
# Log:
#96xxxx Jay: finished this so it emulates gets near exactly
#
30 proc getvalidline { fpvar linevar } {
    puts here
    upvar 1 $linevar line ;# be able to return line
    upvar 1 $fpvar fp ;# be able to return line
    while { [gets $fp line] >= 0 } {

        # get rid of whitespace on the ends (^ and $)
        set line [string trim $line]

        # how do you combine these two ifs into one line?
        if { [string index $line 0] == "#" } {
            continue
        }
if { [string length $line] == 0 } {
    continue
}
break
} ;# end of while
#puts stdout "line is \"$line\"

return line ;# return line to emulate gets's features

#
# GetUserlnfo
#
# sets the user id.

# ToDo:
#    we really need to have an autoconfig here.
# PseudoBugs:
# Bugs:
#    you must find a proper solution for your system.
# Log:
#970118 Jay:  you now use a switch command with a var that is set in Var.tcl
#
# Usage:  You must set the $Config(uidsearch) var, found in Var.tcl, to the
# properly method of finding the user id. The options, in the preferred order
# of usage, are:
# environment: if you systems sets a 'UID' environment var, use this.
# gnu_id: use if your system has a GNU comparable 'id' command.
# sysv_id: for System V versions of 'id', use this
# read_passwd: if you just prefer to read the password file, this
#               might work. Assumes you have 'whoami' installed.

# What Systems:
# env: ???
# gnu: Linux
# sysv: Linux, Solaris 2.\{3,4,5\}
# pass: (most non-shadow password systems)

proc GetUserlnfo {} {
    global Config StartMsg

    set Config(uid)   -1
    set name unknownuser
    switch -exact -- $Config(uidsearch) {

        gnu_id {
            set Config(uid)     [exec id -u]
            set Config(username) [exec id -un]
        }

        read_passwd {
            set Config(uid)     [exec egrep "\^[exec whoami]"
            set Config(username) [exec whoami]
        }

    }
}

# /etc/passwd | awk -F : "\{print $3\}"
set Config(username) [exec whoami]
sysv_id {
    set err [regexp --
    "uid=(\[0-9]*)(.)(\[0-9,A-Z,a-z,_,]*)(.*)" [exec id] match
    Config(uid) junk Config(username) junk]
    if { $err == 0 } {
        puts $StartMsg(cantfindid) ;
        set Config(uid) -1
        set Config(username) -1
    }
    #set Config(username) [exec whoami]
}

environment {
    set Config(uid) $env(UID)
    set Config(username) [exec whoami]
}

default {
    puts $StartMsg(uidnotsetup)
}

}; #end of switch

if { $Config(uid) == -1 } {
    puts $StartMsg(unknownuid)
}

puts "username is \'\$Config(username)\' and uid is \'\$Config(uid)\'"
} ;# end of GetUid

#
# Parse the Command Line
#
# ToDo:
# PseudoBugs:

# Bugs:
# Log:
#
proc ParseCommandLine {} {
    global Config StartMsg argv argc

    set i 0
    while {$i < $argc} {
        set t [lindex $argv $i]
        db 100 "at arg $i == $t"
        switch -exact -- $t {
            -l
            -language
            -lang { incr i
                set t [lindex argv $i],
                SetVisualLanguage $t
            }
            default { puts StartMsg(unknownargv) $t }
        } ;# end switch
    } ;# end while
} ;# end of parse command line
# the visual language setter
#
# ToDo:
# PseudoBugs:
# Bugs:
# Log:
#
proc SetVisualLanguage { $lang } { 
    global Config Dir StartMsg

    # improve this to up/lo case, etc
    set t $lang

    switch --exact -- $t {
        English   -
        english   -
        en        -
        us        -
        eng       { source $Dir(src)/lang/Lang.english.tcl }
        default   { puts "$StartMsg(unknownlang)\$t\n" }
    }

} ;# end switch

} ;# end of set visual language

gui/mymenutool.tcl
# My Menu Tools, partially stolen from Welsh's Menu chapter.
#
# many changes from his stuff thought, the main of which is the names of the
# menu do not have to be same as the menu var.
#
# RCS: File: $RCSfile$
# Revision: $Revision$
# Last Changed by $Author$ on $Date$
# Currently Edited by: $Locker$
#
# ToDo:
# PseudoBugs:
# Bugs:
# Log:
#
proc MenuSetup { {frame .menubar} } {
    global Menu
    frame $frame
    pack $frame -side top -fill x
    set Menu(menubar) $frame
    set Menu(uid) 0
}

proc Menu { label textvar } {
    global Menu
    if [info exists Menu(menu,$label)] {
        error "Menu $label already defined"
    }

    # Create the menubutton and its menu
    set name $Menu(menubar).mb$Menu(uid)
set menuName $name.menu
incr Menu(uid)
set mb [menubutton $name -text $textvar -menu $menuName]
pack $mb -side left
set menu [menu $menuName -tearoff 1]

# Remember the widget name under a variable derived from the label.
# This allows mxMenuBind to be passed the label instead of the widget.
set Menu(menu,$label) $menu
return $menu

proc MenuCommand { menuName textvar command } {
    global Menu
    if [catch {set Menu(menu,$menuName)} menu] {
        error "No such menu: $menuName"
    }
    $menu add command -label $textvar -command $command
}

proc MenuCheck { menuName textvar var {command {}} } {
    global Menu
    if [catch {set Menu(menu,$menuName)} menu] {
        error "No such menu: $menuName"
    }
    $menu add check -label $textvar -command $command \ -variable $var
}

proc MenuRadio { menuName textvar var {value{}} {command {}} } {
global Menu
if [catch {set Menu(menu,$menuName)} menu] {
    error "No such menu: $menuName"
}
if {[string length $value] == 0} {
    set value $textvar
}
$menu add radio -label $textvar -command $command \
    -value $value -variable $var

proc MenuSeparator { menuName } {
    global Menu
    if [catch {set Menu(menu,$menuName)} menu] {
        error "No such menu: $menuName"
    }
    $menu add separator
}

proc MenuCascade { menuName label textvar } {
    global Menu
    if [catch {set Menu(menu,$menuName)} menu] {
        error "No such menu: $menuName"
    }
    if [info exists Menu(menu,$label)] {
        error "Menu $label already defined"
    }
    set sub $menu.sub$Menu(uid)
    incr Menu(uid)
    menu $sub -tearoff 0
$menu add cascade -label $textvar -menu $sub
set Menu(menu,$label) $sub
}

proc MenuBind { what sequence menuName label } {
    global Menu
    if [catch {set Menu(menu,$menuName)} menu] {
        error "No such menu: $menuName"
    }
    if [catch {$menu index $label} index] {
        error "$label not in menu $menuName"
    }
    set command [$menu entrycget $index -command]
    bind $what $sequence $command
    $menu entryconfigure $index -accelerator $sequence
}

proc MenuEntryConfigure { menuName label args } {
    global Menu
    if [catch {set Menu(menu,$menuName)} menu] {
        error "No such menu: $menuName"
    }
    eval {$menu entryconfigure $label $args}
}

gui/path.tcl
# ToDo:
# PseudoBugs:
# Bugs:
# Log:
#960722 Jay: created.
#
proc pathexpand { dir filename } {
    global Dir

    db 1000 "dir: \'$dir\' filename: \'$filename\'

    if { [string index $filename 0] == "$" } {
        set t [regexp -- "(\[\$]\)([a-zA-Z0-9_]*)(\[/\])(.*)" \$filename match dollar var slash rest ]
        # puts " t: \'$t\'
        # puts " variable: \'$var\' rest: \'$rest\'
        if { $t != 1 } {
            puts stderr "Invalid variable name within filename \'$filename\'.\n            set ret $filename
        } else {
            set ret $Dir($var)/$rest
        }
    } elseif { [string index $filename 0] != "/" } {
        set ret $Dir($dir)/$filename
    } elseif { [string index $filename 0] != "/" } {
        set ret $Dir($dir)/$filename
    }
} else {
    set ret $filename
}
# puts " returning '$ret'"
return $ret
}; #end pathexpand
Appendix F

PVM Source Code
All header files are auto-generated with the `make proto` command. The only exceptions are `pvm/heap.h`, `pvm/pvm_slave.h`, and `func/parint.h`.

```
#include <stdio.h>
#include "function.h"
#include <math.h>
#include "commandline.h"
```

```
int Compact_Output = 1; /* the oneline or the verbose results */
double dInfinity = 9999999.99;
double pi = 3.14159;
```

/*
   This parses:

   double:  NUMBER, -NUMBER
   special: pi, ex (e^1), inf (infinity), -inf (neg inf)
   mixes:   piXNUMBER, exXNUMBER (where X means +,-,* or /)
   thus:    it does PI*1.5 not 3*PI/2 or PI*3/2

There is NO error checking!!!

This assumes everything is already lowercase.

TODO:
   it would be nice if an entire string parser that could handle
expressions like "(1+PI)/2" were implemented!

*/

double StringToDouble(char *s) {
    double ret;
    char *p;

    p=s;
    ret=0.0;
    if (strncmp(p,"pi",2)==0) {
        ret=pi;
        p+=2;
    } else if (strncmp(p,"ex",2)==0) {
        ret=exp(1.0);
        p+=2;
    } else if (strncmp(p,"inf",3)==0) {
        ret=dInfinity;
        p+=3;
        return ret; /* return it now since inf*num is meaningless */
    } else if (strncmp(p,"-inf",4)==0) {
        ret=-dInfinity;
        p+=4;
        return ret; /* return it now since -inf*num is meaningless */
    }

    if (p!=NULL) {
        if (*p=='+') {
            p++;
            ret+=atof(p);
        } else if (*p=='-') {
            p++;
        }
    }
return ret;}
} /* end of String To Double */

void parse_command_line( int argc, char **argv) {
    int x,p;
    char *arg, *ch;
    for (x=1; x<argc; x++) {
        arg = (argv[x]);
        /*printf("'¼s' '¼s'\n", argv[x], arg);*/
        if (*arg=='-') arg++;
        if (*arg=='-') arg++;
        ch = arg;

        while (*ch!='\0') { *ch = tolower(*ch); ch++; } /* make it lowercase */
        if ((strncpy(arg,"nodes",strlen(arg))==0)
|| (strncmp(arg,"slaves",strlen(arg))==0)) {
if (x++ == argc) error_args(argv[0],NULL);
Total_Slaves = atoi(argv[x]);
if (Total_Slaves < 1) error_args(argv[0],argv[x]);
} else if (strncmp(arg,"ns_threshold",strlen(arg))==0) {
if (x++ == argc) error_args(argv[0],NULL);
NS_Threshold = atoi(argv[x]);
if ( NS_Threshold < 0 ) error_args(argv[0],argv[x]);
} else if (strncmp(arg,"percent_threshold",strlen(arg))==0) {
if (x++ == argc) error_args(argv[0],NULL);
Percent_Threshold = (double)(atoi(argv[x])) / 100.0;
NS_Threshold = 1000;
if ( Percent_Threshold < 0 ) error_args(argv[0],argv[x]);
} else if (strncmp(arg,"method",strlen(arg))==0) {
if (x++ == argc) error_args(argv[0],NULL);
} else if (strncmp(arg,"limits",strlen(arg))==0) {
if (x++ == argc) error_args(argv[0],NULL);
for (p=0; p<FunctionDimensions; p++) {
if (x+1 >= argc) error_args(argv[0],NULL);
Start[p]=StringToDouble(argv[x]);
x++;
Finish[p]=StringToDouble(argv[x]);
}
} else if (strncmp(arg,"help",strlen(arg))==0) {
CommandLineHelp(argv[0]);
} else if (strncmp(arg, "non-compact", strlen(arg)) == 0) {
    Compact_Output = 0;
}
}
} else if (strncmp(arg, "compact", strlen(arg)) == 0) {
    Compact_Output = 1;
}
} /* end for x=1 to argc */
if ((Function < 0) || (Total_Slaves < 1)) error_args(argv[0], NULL);

#ifdef Cubes
    Proc_Dim = -2;
    /* Test to see if total_slaves is power of 2 */
#endif
dddd
    for (x = 0; x < MaxFunctionDimensions; x++)
        if (1 << x == Total_Slaves) {
            Proc_Dim = x;
            Total_Slaves = 1 << x;
            break;
        }
#endif
    p = x = 0;
    while (p <= Total_Slaves) {
        p = 1 << x;
        if (p == Total_Slaves) {
            Proc_Dim = x;
            Total_Slaves = p;
            break;
        }
        x++;
    }
if (Proc_Dim<0) {
    printf("When compiled with the 'Cubes' option, <num Slaves> must be a power of 2.\n");
    exit(1);
}
#endif

/* end parse command line */

void CommandLineHelp(char *progname) {

    printf("The proper usage is \"s [options] \" where options is\none of the following: \n", progname);
    printf("\"
--NS_Threshold <ns val> Change the data send times. \n
--Percent_Threshold <val> Send data upon error changes of X percent (or \n
NS==1000). X is an integer.\n
--Slaves <number> or \n
--Nodes <number> How many nodes to use.\n
--Method <formula> Use gnzrul or basrul (not yet working)\n
--Compact One line of output\n
--Non-Compact Many lines of output with explanations\n
--Limits x1 x2 y1 y2 .... Limits of integration in order of \n
dim1_start, dim1_finish, 'dim2_start, etc. All limits must\n
be present\n");

    printf("\nNodes must be given on command line\n");
#endif

    printf("and the number of nodes must be a power of 2\n");
#endif
    exit(-1);
void error_args(char *progname, char *text)
{
    if (text)
        printf("%s: error at or '%s'\n", progname, text);
    else
        printf("%s: error at or near end of command line arguments\n", progname);  
        printf("Use '%s --help' for a list of command line options\n", progname);
        exit(-1);
} /* end error args */
long    Granularity   = 100000; /* max func calls per node */
int     NS_Threshold = 20;    /* number child splits between updates */
double  Percent_Threshold= 0.0; /* local_err change required before send */
int     Total_Slaves   = 0;
int     Proc_Dim      = -1;   /* ifdef Cubes, proc dim needed */
double  Total_Result  = 0.0;
double  Total_Error   = 0.0;
long    Total_Funct_Calls= 0;
double  eps_a        = 0.000001;
double  eps_r        = 0.000001;
double  eps          = 0.0;
int     Initial_Split_Dir= 0;

int     Function     = 0;
long    Funct_Calls  = 0;

double  Error_Threshold = 0.000006;
int     FunctionDimensions= -1;

double  *Start;
double  *Finish;

int     parent        = -1;   /* mainly used for pvm */
int     whoami        = -1;

/* used only for pvm, but put here for ease */
int     PVM_mytid;
int     *PVM_tids;     /* the child won’t use it */
pvm/gnzrul.c

   You must link the resulting object file with the libraries:
   -lF77 -ll77 -lm -lc (in that order)
 */

/* code modified in jan-feb-97 by jay!

code adds the divaxn-=1; to the end results value to work
with c.
also, funct_ modified to accept "vector compatible syntax" function.
*/

#include "f2c.h"
#include "math.h"

/* double functn_ (int N, double *x); */

20

/* Subroutine */ int gnzrul_(ndim, center, width, rgnvol, rgnval, rgnerr, 
divaxn)
integer *ndim;
integer *divaxn;
{
   /* System generated locals */
   integer i_1, i_2, i_3;
doubleal d__1, d__2;

/* Builtin functions */
double pow_di(), sqrt();

/* Local variables */
static doubleal half, zero, weit1, weit2, weit3, weit4, weit5;
static integer i, k, l, m, n;
static doubleal z[100], ratio, f1, f2, f3, f4, lamda2, lamda4, lamda5,
weitp1, weitp2, weitp3, weitp4, difmax, rlndim, rgncmp, df1, df2;
extern doubleal functn_();

static doubleal widthl[100];
static integer divaxo;
static doubleal twondm;
static integer rgnstr;
static doubleal dif, one, two, sum1, sum2, sum3, sum4, sum5;
static doubleal tmpres[1];

/* ***** INITIALISATION OF SUBROUTINE */

/* Parameter adjustments */
--width;
--center;

/* Function Body */
zero = 0.;
one = 1.;
two = 2.;
half = one / two;
*twondm = pow_di(&two, ndim);*/
twondm = pow(two, (double) (*ndim));
	n= (*ndim << 1) + 3;

divaxo = 0;

/* ***** END SUBROUTINE INITIALISATION */
/* ***** BASIC RULE INITIALISATION */

/* ***** CHANGE THE NEXT THIRTY STATEMENTS FOR DOUBLE PRECISION */
lamda5 = .47368421052631576;
rlndim = (doubleral) (*ndim - 1);
if (*ndim > 15) {

goto L10;
}
lamda4 = .90000000000000002;
lamda2 = .12857142857142856;
/* Computing 3rd power */
d__1 = lamda5 * 3., d__2 = d__1;
weit5 = one / (d__2 * (d__1 * d__1)) / twondm;
goto L20;
L10:
	ratio = (rlndim - one) / 9.;
lamda4 = (one / 5. - ratio) / (one / 3. - ratio / lamda5);
ratio = (one - lamda4 / lamda5) * rlndim * ratio / 6.;
lamda2 = (one / 7. - lamda4 / 5. - ratio) / (.2 - lamda4 / 3. - ratio / lamda5);
/* Computing 3rd power */
d__1 = lamda5 * 6., d__2 = d__1;
weit5 = one / (d__2 * (d__1 * d__1));
L20:
/* Computing 2nd power */
d_2 = lamda4;
weit4 = (one / 15. - lamda5 / 9.) / ((lamda4 - lamda5) * 4. * (d_2 * d_2));
weit3 = (one / 7. - (lamda5 + lamda2) / 5. + lamda5 * lamda2 / 3.) / (two * lamda4 * (lamda4 - lamda5) * (lamda4 - lamda2)) - two * rlndim * weit4;
weit2 = (one / 7. - (lamda5 + lamda4) / 5. + lamda5 * lamda4 / 3.) / (two * lamda2 * (lamda2 - lamda5) * (lamda2 - lamda4));
if (*ndim > 15) {
weit1 = one - two * (rlndim + one) * (weit2 + weit3 + rlndim * (weit4 + two * (rlndim - one) * weit5 / 3.));
}
if (*ndim < 16) {
weit1 = one - two * (rlndim + one) * (weit2 + weit3 + rlndim * weit4) - twondm * weit5;
}
/* Computing 2nd power */
d_2 = lamda4 * 6.;
weitp4 = one / (d_2 * d_2);
weitp3 = (one / 5. - lamda2 / 3.) / (two * lamda4 * (lamda4 - lamda2)) - two * rlndim * weitp4;
weitp2 = (one / 5. - lamda4 / 3.) / (two * lamda2 * (lamda2 - lamda4));
weitp1 = one - two * (rlndim + one) * (weitp2 + weitp3 + rlndim * weitp4);

ratio = lamda2 / lamda4;
lamda5 = sqrt(lamda5);
lamda4 = sqrt(lamda4);
lamda2 = sqrt(lamda2);

/* ***** END BASIC RULE INITIALISATION */
/ * ***** BEGIN BASIC RULE */
  i__1 = *ndim;
  for (j = 1; j <= i__1; ++j) {
  /* L50: */
    z[j - 1] = center[j];
  }
  functn_(ndim, z, tmpres);
  sum1 = tmpres[0];
  /* ***** COMPUTE SYMMETRIC SUMS OF FUNCTN(LAMDA2,0,0,...,0) AND */
  /* FUNCTN(LAMDA4,0,0,...,0), AND MAXIMUM FOURTH DIFFERENCE */
  difmax = -one;
  sum2 = zero;
  sum3 = zero;
  i__1 = *ndim;
  for (j = 1; j <= i__1; ++j) {
    z[j - 1] = center[j] - lamda2 * width[j];
    functn_(ndim, z, tmpres);
    f1 = tmpres[0];
    z[j - 1] = center[j] + lamda2 * width[j];
    functn_(ndim, z, &f2);
    widthl[j - 1] = lamda4 * width[j];
    z[j - 1] = center[j] - widthl[j - 1];
    functn_(ndim, z, &f3);
    z[j - 1] = center[j] + widthl[j - 1];
    functn_(ndim, z, &f4);
    sum2 = sum2 + f1 + f2;
    sum3 = sum3 + f3 + f4;
    df1 = f1 + f2 - two * sum1;
    df2 = f3 + f4 - two * sum1;
/* ***** FOR DOUBLE PRECISION CHANGE ABS TO DABS IN THE NEXT STATEMENT */

150 nt */

dif = (d_1 = df1 - ratio * df2, fabs(d_1));
if (dif <= difmax) {
    goto L60;
}

difmax = dif;
*divaxn = j;

L60:
    z[j - 1] = center[j];
}

if (sum1 == sum1 + difmax * half) {
    *divaxn = divaxo % *ndim + 1;
}

/* ***** COMPUTE SYMMETRIC SUM OF FUNCTN(LAMDA4, LAMDA4, 0, 0, ... , 0) */

sum4 = zero;
i_1 = *ndim;
for (j = 2; j <= i_1; ++j) {
    i_2 = *ndim;
    for (k = j; k <= i_2; ++k) {
        for (l = 1; l <= 2; ++l) {
            widthl[j - 2] = -widthl[j - 2];
            z[j - 2] = center[j - 1] + widthl[j - 2];
        }
    }
}

functn_(ndim, z, tmpres);
sum4 += tmpres[0];
180 /* L80: */
    z[k - 1] = center[k];
/* L90: */
    z[j - 2] = center[j - 1];
/* ***** IF NDIM < 16 COMPUTE SYMMETRIC SUM OF */
/* ***** FUNCTN(LAMDA5,LAMDA5,...,LAMDA5) */
    sum5 = zero;
    if (*ndim > 15) {
        goto L130;
    }
    i__1 = *ndim;
    for (j = 1; j <= i__1; ++j) {
        widthl[j - 1] = -lamda5 * width[j];
/* L100: */
        z[j - 1] = center[j] + widthl[j - 1];
    }
L110:
    functn_(ndim, z, tmpres);
    sum5 += tmpres[0];
    i__1 = *ndim;
    for (j = 1; j <= i__1; ++j) {
        widthl[j - 1] = -widthl[j - 1];
        z[j - 1] = center[j] + widthl[j - 1];
        if (widthl[j - 1] > zero) {
            goto L110;
        }
/* L120: */
210 goto L190;

/* ***** IF NDIM > 15 COMPUTE SYMMETRIC SUM OF */
/* ***** FUNCTN(LAMDA5,LAMDA5,LAMDA5,0,0,...,0) */
L130:
    i__1 = *ndim;
    for (j = 1; j <= i__1; ++j) {
        /* L140: */
        widthl[j - 1] = lamda5 * width[j];
    }
    i__1 = *ndim;
    for (i = 3; i <= i__1; ++i) {
        i__2 = *ndim;
        for (j = i; j <= i__2; ++j) {
            i__3 = *ndim;
            for (k = j; k <= i__3; ++k) {
                for (l = 1; l <= 2; ++l) {
                    widthl[i - 3] = -widthl[i - 3];
                    z[i - 3] = center[i - 2] + widthl[i - 3];
                    for (m = 1; m <= 2; ++m) {
                        widthl[j - 2] = -widthl[j - 2];
                        z[j - 2] = center[j - 1] + widthl[j - 2];
                        for (n = 1; n <= 2; ++n) {
                            widthl[k - 1] = -widthl[k - 1];
                            z[k - 1] = center[k] + widthl[k - 1];
                            /* L150: */
                            functn_(ndim, z, tmpres);
                            sum5 += tmpres[0];
                        }
                    }
                }
            }
        }
    }
}
} /* L160: */
    z[k - 1] = center[k];
}
/* L170: */
    z[j - 2] = center[j - 1];
}
/* L180: */
    z[i - 3] = center[i - 2];
}

/* ***** COMPUTE FIFTH AND SEVENTH DEGREE RULES AND ERROR */
250 L190:
    rgncmp = *rgnvol * (weitp1 * sum1 + weitp2 * sum2 + weitp3 * sum3 + weitp4 * sum4);
    *rgnval = *rgnvol * (weit1 * sum1 + weit2 * sum2 + weit3 * sum3 + weit4 * sum4 + weit5 * sum5);

/* ***** FOR DOUBLE PRECISION CHANGE ABS TO DABS IN THE NEXT STATEMENT */
*/
    *rgnerr = (d__1 = *rgnval - rgncmp, fabs(d__1));

/* ***** END BASIC RULE */

260 /* this is the dammed fortran BUG FIX!!! */
    divaxn =-1;

    return 0;
} /* gznru1_ */
The heap module.

written by Patricia Ealy for the nCUBE and modified heavily by
Jay Ball for (hopefully) really really universal use.

#include <stdio.h>
#include <stdlib.h>
#include "heap.h"
#include <math.h>

#ifndef fadsf
typedef struct datum_type_d {
    double *center;
    double *width;
    double *vol;
    double *res;
    double *err;
    int divax;
    int *dimensions;
} datum_type;

typedef struct node {
    datum_type *rgn;
    struct node *p_nptr; /* pointer to parent node */
    struct node *lc_nptr; /* ptr to left child node */
} node;

struct node *rc_nptr; /* ptr to right child node */

) hnode;

#endif

int size_hnode = sizeof(hnode);

datum_type *RegionNew(int dimensions) {
    datum_type *p;
    p=(datum_type *)malloc(sizeof(datum_type));
    p->dimensions=dimensions;
    p->center=(double *)malloc(dimensions * sizeof(double));
    p->width=(double *)malloc(dimensions * sizeof(double));
    return p;
}

void RegionFree(datum_type *p) {
    free(p->center);
    free(p->width);
    free(p);
}

/******************* Heap Maintenance Functions - Definitions *******************/

/**************************** Heaps - Definitions *****************************/

/* create_heap */
/* 60 */ Creates the heap, setting the root's region data to newrgn and setting */ */  *ptr_avail_parent to h_root.  *ptr_avail_parent always points to the */ */  node to which a new leaf can be added to the tree so that the new */ */  tree will still be a complete, balanced binary tree. */ */  void create_heap(hnode **ptr_h_root, hnode **ptr_avail_parent, datum_type *newrgn) { */ */  /* Allocate space for the root node */ */  *ptr_h_root = malloc( size_hnode ); */  if ( !( *ptr_h_root ) ) { /* if space allocation not successful */  printf("\nMemory allocation error in create_heap.\n"); */  exit(1); */  } */  /* Set fields of **ptr_h_root */ */  (*ptr_h_root)->rgn = newrgn; */  (*ptr_h_root)->p_nptr = NULL; */  (*ptr_h_root)->lc_nptr = NULL; */  (*ptr_h_root)->rc_nptr = NULL; */  *ptr_avail_parent = *ptr_h_root; */  } /* End of function 'create_heap' */ /****************************************************************************/ /* *replace_root */
Given the heap with root pointed to by h_root, this function sets the root's rgn (region) data to newrgn and then readjusts the complete, balanced tree so that it again satisfies the "heap condition". So, the algorithm starts at the 'top' of the tree and shifts data 'down' as necessary until the newly added region data is in the correct position to give a heap.

```c
void replace_root( hnode *h_root, datum_type *newrgn )
{

    /* Adjust tree from top down */
    double abserr = fabs(newrgn->err);
    parent = h_root;
    l_child = h_root->lc_nptr;
    r_child = h_root->rc_nptr;

    while (! ( l_child==NULL && r_child==NULL ) ) {
        if ( r_child == NULL ) child = l_child;
        else if (fabs(l_child->rgn->err) > fabs(r_child->rgn->err))
            child = l_child;
        else child = r_child;

        if ( fabs(child->rgn->err) > abserr ) {
            parent->rgn = child->rgn;
            parent = child;
            l_child = parent->lc_nptr;
            r_child = parent->rc_nptr;
        }
    }
```
else break;
}
} /* end of while still more levels to process */

parent->rgn = newrgn;

} /* End of function 'replace_root' */

/***************************************************************************/
/*

130 /* add_heap */
/*
/* * Attaches a new leaf to the tree with root *h_root at the position re-
actual* quired in order to maintain a complete, balanced bin. tree and then */
/* re-heapifies the tree. "Re-heapifies" means that the algorithm starts */
/* at the new leaf and shifts data 'up' the tree as necessary until the */
/* newly added region data is in the correct position to give a heap. */
/*
/* *ptr_avail_parent points to the node to which the new leaf should be */
/* attached. If the new leaf is added as a right-child of the node, then */
140 /* a new available parent node must be found (to be used in the next call */
/* to add_heap) by a call to get_new_avail_parent. */

void add_heap( hnode **ptr_avail_parent, datum_type *newrgn )
{
    hnode *p, *add_to, *parent;
    double abserr;

    /* Allocate space for the new leaf */
if (!(p = malloc(size_hnode))) { /* if space allocation not */
    printf("\nMemory allocation error in add_heap."); /* successful */
    exit(1);
}

add_to = *ptr_avail_parent;

/* Set the new node’s links */
p->p_nptr = add_to;
p->lc_nptr = p->rc_nptr = NULL;

/* Attach new node as correct child of *ptr_avail_parent */
if (add_to->lc_nptr == NULL) add_to->lc_nptr = p;
else {
    add_to->rc_nptr = p;
    get_new_avail_parent(ptr_avail_parent);
}

/* Now re-heapify working bottom to top */
abserr = fabs(newrgn->err);
parent = add_to;
while ((parent != NULL) && (abserr > fabs(parent->rgn->err))) {
    p->rgn = parent->rgn;
    p = parent;
    parent = p->p_nptr;
} /* End of while another level up needs processing */

p->rgn = newrgn;
void get_new_avail_parent( hnode **ptr_avail_parent )
{
    hnode *child, *parent;

    /* If the current available parent is the heap’s root, then make */
    /* the available parent the root’s left child */
    if ( ((*ptr_avail_parent)->p_nptr == NULL) 
        (*ptr_avail_parent) = ((*ptr_avail_parent)->lc_nptr);

    child = *ptr_avail_parent;
    parent = child->p_nptr;

    /* Go up the tree until a change from child is right child to child is */
    /* left child or until root node. */
    while ( (parent->p_nptr != NULL) && (child == parent->rc_nptr) ) {
        child = parent;
parent = child->p_nptr;

} /* Now go back down the tree to the left from appropriate parent to find available parent node. */
if ( (parent->p_nptr != NULL) || (child != parent->rc_nptr) ) /* Have not reached root from the right: change parent for descent */
    parent = parent->rc_nptr;
/* else at root node and arrived from right: don't change parent */
while ( parent->lc_nptr != NULL ) /*ptr_avail_parent = parent;*/
    parent = parent->lc_nptr;
} /* End of function 'get_new_avail_parent' */

***************************************************************************
/* delete_heap */
/* Frees all memory allocation for the binary tree with root p */
/* using postorder traversal of the tree */

void delete_heap(hnode *p)
{
    if (p != NULL) {
        delete_heap(p->lc_nptr);
        delete_heap(p->rc_nptr);
free(p);
}

} /* End of function 'delete_heap' */

int size_hnode;

extern datum_type *RegionNew(int dimensions);
extern void RegionFree(datum_type *p);
extern void create_heap(hnode **ptr_h_root, hnode **ptr_avail_parent, datum_type *newrgn);
extern void replace_root(hnode *h_root, datum_type *newrgn);
extern void add_heap(hnode **ptr_avail_parent, datum_type *newrgn);
extern void get_new_avail_parent(hnode **ptr_avail_parent);
extern void delete_heap(hnode *p);

pvm/hyper_rect.c

#include "stdio.h"
#include "pvm3.h"
#include "function.h"
#include "pvm_slave.h"
#include "heap.h"

void Parent_Initial_Single_Axix_Divide() {
    int l;

    /* this next line MUST be fixed. :) */
    /* Initial_Split_Start = Start[Initial_Split_Dir]; */
    for (l=0; l<Total_Slaves ; l++) {
        /*printf("Processor %d:%x is getting:\n",l,PVM_tids[l]);*/
        pvm_initsend(PvmDataDefault);
        pvm_pkint( &l,1,1);           /* user-readable "whoami" */
        pvm_pkint( &Total_Slaves,1,1);
        pvm_pkint( &Function, 1,1);
        pvm_pkint( &Initial_Split_Dir,1,1);
        pvm_pklong( &Granularity,1,1);
        pvm_pkint( &NS_Threshold,1,1);
        pvm_pkdouble( &Percent_Threshold,1,1);
        pvm_send( PVM_tids[l], Initial_Msg );         /* send Initial Message */
    }
}
datum_type *Child_Initial_Single_Axix_Divide() {
    datum_type *node;
    int 12;
    double interval_size;

    pvm_recv(parent,Initial_Msg); /* get the initial start message */
    pvm_upkint(&whoami,1,1);
    pvm_upkint(&Total_Slaves,1,1);
    pvm_upkint(&Function,1,1);

    InitializeFunction();
    node = RegionNew(FunctionDimensions);

    pvm_upkint(&node->divax,1,1);
    interval_size = (Finish[node->divax]-Start[node->divax]) / (double)Total_Slaves;
    Start[node->divax] = interval_size * (double)whoami;
    Finish[node->divax] = interval_size + Start[node->divax];
    for (12=0; 12<FunctionDimensions; 12++) {
        node->width[12] = (Finish[12] - Start[12]) / 2.0;
    }
    pvm_upklong(&Granularity,1,1);
    pvm_upkint(&NS_Threshold,1,1);
    pvm_upkdouble(&Percent_Threshold,1,1);

    return(node);
}

void Parent_Initial_Hyper_Rectangular_Divide() {
int l;

printf("------>Entering Parent Initial Hyper Rectangular Divide\n");
for (l=0; l<Total_Slaves; l++) {
    printf("Processor %d:%x is getting: \n", l, PVM_tids[l]);
    pvm_initsend(PvmDataDefault);
    pvm_pkint(&l, 1, 1); /* "whoami" for cube number */
    pvm_pkint(&Total_Slaves, 1, 1);
    pvm_pkint(&Proc_Dim, 1, 1);
    pvm_pkint(&Function, 1, 1);
    pvm_pklong(&Granularity, 1, 1);
    pvm_pkint(&NS_Threshold, 1, 1);
    pvm_pkdouble(&Percent_Threshold, 1, 1);
    pvm_send(PVM_tids[l], Initial_Msg); /* send Initial Message */
}
printf("<---Leaving Parent Initial Hyper Rectangular Divide\n");
}

datum_type *Child Initial Hyper Rectangular Divide() {
    int nd_1, nd_2, *nd, j, d;
    double *full_int, *sd;
    datum_type *node;

    /*printf("------>Entering Child Initial Hyper Rectangular Divide\n");*/
    /*fflush(stdout);*/
    pvm_recv(parent, Initial_Msg); /* get the initial start message */
    pvm_upkint(&whoami, 1, 1);
    pvm_upkint(&Total_Slaves, 1, 1);
pvm_upkint(&Proc_Dim, 1, 1);
pvm_upkint(&Function, 1, 1);
pvm_upklong(&Granularity, 1, 1);
pvm_upkint(&NS_Threshold, 1, 1);
pvm_upkdouble(&Percent_Threshold, 1, 1);

/*printf("before InitializeFunction\n");*/
/*fflush(stdout);*/
InitializeFunction();

node = RegionNew(FunctionDimensions);

/*printf("allocating nd with funcdim=%d\n", FunctionDimensions);*/
/*fflush(stdout);*/
nd = (int *)malloc(sizeof(int) * FunctionDimensions);

/*printf("allocating sd\n");*/
/*fflush(stdout);*/
sd = (double *)malloc(sizeof(double) * FunctionDimensions);

/*printf("allocating full_int\n");*/
/*fflush(stdout);*/
full_int = (double *)malloc(sizeof(double) * FunctionDimensions);

/*printf("first loop....\n");*/
/*fflush(stdout);*/
nd_1 = 1 << (Proc_Dim/FunctionDimensions);

nd_2 = 2 * nd_1;
for (j=0; j<Proc_Dim%FunctionDimensions; j++) nd[j] = nd_2;
for (j=Proc_Dim%FunctionDimensions; j<FunctionDimensions; j++) nd[j] = nd_1;

/*printf("second loop....\n");*/
void Rectangular_Divide(datum_type *old, datum_type *new1, datum_type *new2)
{
    register int x, axis;

    /* with luck, axis is properly set from the acutaly divaxn stuff */
    /* axis = old->divax - 1; */ /* damned fortran stuff */
new1->divax = new2->divax = old->divax;
new1->dimensions = new2->dimensions = old->dimensions;
for (x=0; x<old->dimensions; x++) {
    new1->width[x] = new2->width[x] = old->width[x];
    new1->center[x] = new2->center[x] = old->center[x];
}
new1->width[axis] = new2->width[axis] = old->width[axis] / (double)2.0;
new1->center[axis] = old->center[axis] - new1->width[axis];
new2->center[axis] = old->center[axis] + new2->width[axis];
new1->vol = new2->vol = old->vol / (double)2.0;
}

pvm/pvm_child.c

/*
ParInt
The PVM Child Program
*/

#include <stdio.h>
#include "pvm3.h"
#include "f2c.h"
#include "function.h"
#include "pvm_slave.h"
#include <time.h>
#include "heap.h"
#include "hyper_rect.h"
#include <math.h>
/**
#include "methods.h"*/
/*double  full_int[MaxFunctionDimensions];
   double  sd[MaxFunctionDimensions]; */

int dim;
int whoami;
int parent;

int gznzrul_(int *ndim, double *center, double *width, double *
*rgnvol, double *rgnval, double *rgnerr, int *divaxn);

int main(int argc, char**argv)
{

double ns_error, ns_result, result;
long  ns_send,  ns_count;
int  12, mytid, go_flag, got, divax;
long  count0, count_inc;
double interval_size;
double eps, upd_r, upd_e, local_error, local_result, local_eps;
datum_type *node, *n1, *n2;
hnode *Root, *Parent;

/* this needs to be put in a different file */
/* int nd_1, nd_2, *nd, j, d;
   double *sd, *full_int; */

   mytid = pvm_mytid();
   parent = pvm_parent();

   printf("\n----------------------\nI'm alive pvm id= %d\nand my

parent is=\%d\n------------------\n" ,mytid,parent); 

    node = Child_Initial_Hyper_Rectangular_Divide();

    printf("\n------------------------\nI'm alive:\n\%d:\%d\n------------------------\n" ,whoami,mytid);

    printf("computing volume...\n");
    fflush(stdout);
    node->vol = 1;
    for (l2=0; l2<FunctionDimensions ; l2++) {
        node->vol*=\%d(\%d)2.0*(\%d)node->width[l2];
    }

#elif Extra_Crap
    pvm_init_send(PvmDataDefault); // tell parent that i am alive */
    pvm_pkint( &whoami ,1,1);
    pvm_pkdouble( &node->vol ,1,1);
    pvm_pkdouble( node->center , FunctionDimensions ,1);
    pvm_pkdouble( node->width , FunctionDimensions ,1);
    pvm_send(parent, I_am_Alive );
#endif

/* node->divax+=1; /* damned fortran bug fix. */
printf("\%d\tPre-gnzrul\n",whoami);
fflush(stdout);
gnzrul_(&FunctionDimensions, node->center, node->width, &node->vol),
    &(node->res), &(node->err), &(node->divax));
printf("\%d\tPost-gnzrul\n",whoami);
flush(stdout);

pvm_initsend(PvmDataDefault);     /* send initial estimate */
pvm_pkint( &whoami ,1,1);
pvm_pkdouble( &node->res ,1,1);
pvm_pkdouble( &node->err ,1,1);
pvm_pklong( &Funct_Calls ,1,1);
pvm_send(parent, First_Est);
printf("%d\tdone sending first estimate\n",whoami);
flush(stdout);

#define Slow_Method

if (go_flag==0) {
   pvm_exit();
   exit(0);
}
#endif

local_error = node->err;
local_result = node->res;
local_eps = -10000.0;
go_flag = 1;
/*HeapInsert(node);*/
printf("About to create heap....\n");
fflush(stdout);
create_heap(&Root,&Parent,node);
printf("Heap created....\n");
fflush(stdout);

#ifdef Extra_Crap
  pvm_initsend(PvmDataDefault);
  /* send doing it message*/
  pvm_pkint( &whoami ,1,1);
  pvm_pkint( &go_flag,1,1);
  pvm_pkdouble( &local_error ,1,1);
  pvm_pkdouble( &local_eps ,1,1);
  pvm_send( parent, Doing_It);
#endif

printf("Entering main loop....%d\n", ((go_flag)&&(local_error
> local_eps)&&(Funct_Calls < Granularity)));
fflush(stdout);
ns_error = ns_result = ns_send = ns_count = count0 = 0;
while (((go_flag)&&(local_error > local_eps)&&(Funct_Calls < Granularity))){
  printf("Iteration %d\n",Funct_Calls-count0);
  fflush(stdout);
  node = Root->rgn ;
  n1 = RegionNew(FunctionDimensions);
  n2 = RegionNew(FunctionDimensions);
  Rectangular_Divide(node, n1, n2);
  count0 = Funct_Calls;
  gnrzu1_(&FunctionDimensions, n1->center, n1->width,
  &(n1->vol), &(n1->res), &(n1->err), &(n1->divax));
}
replace_root(Root,n1);

gnzu1_(&FunctionDimensions, n2->center, n2->width,
     &(n2->vol), &(n2->res), &(n2->err), &(n2->divax));

add_heap(&Parent,n2);

upd_r = n1->res + n2->res - node->res;
upd_e = n1->err + n2->err - node->err;
local_result += upd_r;
local_error += upd_e;

if ( ((++ns_send % NS_Threshold)==0) || (upd_e < Error_Threshold) )
{
    printf("%03d: Sending loop est\n",whoami);
    fflush(stdout);*/
    pvm_initsend(PvmDataDefault); /* send loop estimate */
    pvm_pkint( &whoami ,1,1);
    pvm_pkdouble( &ns_result ,1,1);
    pvm_pkdouble( &ns_error ,1,1);
    pvm_pklong( &ns_count ,1,1);
    pvm_send(parent, Loop_Est );
    ns_error = ns_result = ns_send = ns_count = 0;
    Error_Threshold = upd_e * Percent_Threshold;
    got = pvm_nrecv( parent, Flag); /* Do we need to quit? */
    if (got)
        pvm_upkint(&go_flag ,1,1);

    got = pvm_nrecv(parent, Start_Computation );
}
if (got) {
    /* parent send eps? */
    pvm_upkdouble( &eps ,1,1);
    pvm_upkint( &go_flag ,1,1);
    local_eps = (double) (eps / (double)Total_Slaves);
}
#endif

if (ns_send==0) {
    /* if data left, then send it */
    /*printf("%03d: Sending final loop est\n",whoami);
    fflush(stdout);*/
    pvm_initsend(PvmDataDefault); /* send loop estimate */

    pvm_pkint( &whoami ,1,1);
    pvm_pkdouble( &ns_result ,1,1);
    pvm_pkdouble( &ns_error ,1,1);
    pvm_pklong( &ns_count ,1,1);
    pvm_send(parent, Loop_Est);
}
#else
    pvm_initsend(PvmDataDefault); /* send Quit verify */
    pvm_pkint( &whoami ,1,1);
    pvm_send(parent, I_Quit);
#endif

    printf("Just send the I_Quit code\n");
    fflush(stdout);
    pvm_exit();
    exit(0);
}
/*
ParInt

PVM Parent Program

Jay Ball
*/

#include "pvm3.h"
#include <stdio.h>
#include "function.h"
#include "pvm_slave.h"
#include "commandline.h"
#include <math.h>
#include <time.h>
#include <unistd.h>
#include "heap.h"
#include "hyper_rect.h"

void error_pvm( char *s);
void process_terminate(int *PVM_tids);

int main (int argc, char**argv )
{
    int error=0, x, 1, l2, go_flag, temp, running, morestuff;
double Initial_Split_Start, tempd;
int child, got;
long count_inc;
double upd_r, upd_e;

/* various vars for timing */
unsigned long tvdiff1, tvdiff2, us1, us2, rt1, rt2;
struct timeval tv1, tv2, tvfin;
struct timezone tz;

(tv1).tv_sec = (tv1).tv_usec = 0;  /* zero out timer */
(tv2).tv_sec = (tv2).tv_usec = 0;
(tvfin).tv_sec = (tvfin).tv_usec = 0;

Total_Slaves = -1;  /* force user to enter these values */

/* can we just delete this Function Variable? */
/* Function = 0;  /* becuase we now no longer have the full set of functions built in,
we now just set Function to an arbitary value */
InitializeFunction();

parse_command_line(argc, argv);

if (Compact_Output) {
    /* this is the "one per line" format output. see bottom of main.c for other part */
    printf("%03d (%02d) %02d %lf %lf %ld %04d ", Total_Slaves,
            Proc_Dim, Function, eps_a, eps_r, Granularity, NS_Threshold);
} else {
/* the pretty gui version .... */

printf("Number processes: %d\n", Total_Slaves);
}

/* for (l=0; l<FunctionDimensions; l++) { */
    if (l == Initial_Split_Dir) {
        int_size[l] = (Finish[l] - Start[l]) / (double)Total_Slaves;
    } else {
        int_size[l] = (Finish[l] - Start[l]);
    }
    printf("\t%02d %lf\n", l, int_size[l]);
/* }

gettimeofday( &tv1 , &tz); /* start the timer */

PVM_tids = (int *)malloc(sizeof(int) * Total_Slaves);

PVM_mytid = pvm_mytid();

/* i suppose i could put argv instead of 0 here.... */
error = pvm_spawn(Slave_Name, (char **)0, PvmTaskDefault, (char *)0, Total_Slaves, PVM_tids);

printf("allocated array at %x, calling %s, with slaves of %d\n", PVM_tids, Slave_Name, error);
if (error!=Total_Slaves) {
    error_pvm("screwed up upon spawning.");
}

/* for added verbosity.... */
/* printf("Slave PVM_tids are: "); */
/* for (l=0; l<Total_Slaves ; l++) { */
/*     printf("%d:%x ",l,PVM_tids[l]); */

} */

/* printf("\nSending Initial Message\n"); */

Parent_Initial_Hyper_Rectangular_Divide();

printf("Initial Message has been sent\n");

/* lots of extra verbosity */
#ifdef Extra_Crap
  for (l=0; l<Total_Slaves; l++) {

    int child;
    double vol, Center[MaxFunctionDimensions], HalfWidth[MaxFunctionDimensions];

    pvm_recv(-1, I_am_Alive); /* get alive message */
    pvm_upkint( &child, 1, 1);
    pvm_upkdouble( &vol, 1, 1);
    pvm_upkdouble( Center, FunctionDimensions, 1);
    pvm_upkdouble( HalfWidth, FunctionDimensions, 1);
    printf("child %d in alive with volume %lf\n", child, vol);
    for (12=0; 12<FunctionDimensions; 12++) {
      printf("\t%02d Center: %lf\tHalfWidth:%lf\n", 12, Center[12], HalfWidth[12]);
    }
  }
  printf("Got all Alive Messages\n");
#endif

/* now, get the first error est */
for (l=0; l<Total_Slaves; l++) {

double res, err;
long func;
int got=-1, child;

printf("waiting for first estimate \\
" , 1);
got=pvm_recv(-1 , First_Est ); /* get first estimate */
pvm_upkint( &child , 1, 1);
pvm_upkddouble( &res , 1, 1);
pvm_upkddouble( &err , 1, 1);
pvm_upklong( &func , 1, 1);
Total_Result+=res;
Total_Error+=fabs(err);
Total_Funct_Calls+=func;
#endif PRINT

printf("%d\tResult: %.16lf\tError: %.16lf\tFunct Calls: %ld\n",
    child, res, err, func);
#endif
#endif PRINT

printf("Est Result: %.16lf\tError: %.16lf\tFunct Calls: %ld \n",
    Total_Result, Total_Error, Total_Funct_Calls);

/* eps = max( eps_1 , eps_r * abs(result) ) */
tempd = eps_r * fabs(Total_Result);
eps = (eps_a > tempd) ? eps_a : tempd;

pvm_initsend(PvmDataDefault); /* send START command */
pvm_pkdouble( &eps , 1, 1);
go_flag = Total_Error > eps ; /* 1=work 0=done */
pvm_pkint( &go_flag , 1, 1);
pvm_mcast( PVM_tids, Total_Slaves , Start_Computation );

#define PRINT

printf("First Eps computation: %lf and go_flag is %d\n",eps,go_flag);
#endif

if (go_flag==0) {
    gettimeofday( &tvfin , &tz);
    tvdiff1 = tvfin.tv_sec - tv1.tv_sec;
    us1 = tvfin.tv_usec - tv1.tv_usec;
    rt1 = tvfin.tv_sec*1000000 - tv1.tv_sec*1000000 + tvfin.tv_usec - tv1.tv_usec;

    if (Compact_Output==1) {
        printf("%16.16lf %16.16lf %lf
", Total_Result, Total_Error,
               (double)rt1/1000000.0,(double)rt2/1000000.0);
    } else {
        printf("Result: %16.16lf\nError: %16.16lf\nTotal Time: %lf\nComputation Time: %lf\n",
               Total_Result, Total_Error,(double)rt1/1000000.0,(double)rt2/1000000.0);
    }

    pvm_exit();
    exit(0);
}

#define Extra_Crap

for (l=0; l<Total_Slaves ; l++) {
    int child, go_flag;
    double local_err, local_eps;

    pvm_recv( -1 , Doing_It);
    pvm_upkint( &child ,1,1);
    pvm_upkint( &go_flag,1,1);
pvm_upkdouble( &local_err ,1,1);
pvm_upkdouble( &local_eps  ,1,1);
printf("%03d\tgo\t%03d\terr: %1f\teps: %1f\n",
   child, go_flag, local_err, local_eps);
}
#endif

#ifdef PRINT
    printf("Gathering Data\nCPU incr upd_r\ttot_r\t\tupd_e\ttot_e\n");
#endif
    running = Total_Slaves;

gettimeofday( &tv2 , &tz); /* get second timing for the main loop */
while ( (Total_Error > eps) && (running>0)) {
    got = pvm_nrecv(-1, Loop_Est);
    if (got > 0) {
        pvm_upkint( &child,1,1);
pvm_upkdouble( &upd_r ,1,1);
pvm_upkdouble( &upd_e ,1,1);
pvm_upklong( &count_inc ,1,1);
       Total_Result += count_inc;
       Total_Error  += upd_e;
    
    #ifdef PRINT
        printf("%03d %06ld %1f\tt%1f\tt%1f\tt%1f\n",child,
            /*count_inc*/ Total_Funct_Calls,
            upd_r,Total_Result,upd_e, Total_Error);
    #endif
    } else if (got < 0) {
        printf("Error getting Loop_est\n");
got = pvm_nrecv(-1, I_Quit);
if (got > 0) {
    pvm_upkint( &child, 1, 1);
    running--;
    #ifdef PRINT
    printf("%03d\thas quit running. nodes still left: %d\n", child, running);
    #endif
} else if (got < 0) {
    printf("Error getting I_Quit\n");
}

/* Since the quit message can be read before all data is in, it is possible that we will have data waiting to in a buffer after all children have died. Thus, we check for that here */
if (!running)
    running = pvm_probe(-1, Loop_Est);

/* sleep(5); printf("La la\n"); */

/* get finish time and compute the difference */
gettimeofday( &tvfin, &tz);
tvdiff1 = tvfin.tv_sec - tv1.tv_sec;
tvdiff2 = tvfin.tv_sec - tv2.tv_sec;
us1 = tvfin.tv_usec - tv1.tv_usec;
us2 = tvfin.tv_usec - tv2.tv_usec;
rt1 = tvfin.tv_sec*1000000 - tv1.tv_sec*1000000 + tvfin.tv_usec - tv1.tv_usec;
rt2 = tvfin.tv_sec*1000000 - tv2.tv_sec*1000000 + tvfin.tv_usec - tv2.tv_usec;
/* printf("%16.16lf %16.16lf %lf %lf\n",Total_Result,Total_Error,
 (double)rt1/1000000.0,(double)rt2/1000000.0); */

printf("Result: %16.16lf\nError: %16.16lf\nTotal Time: %lf\nComputation Time: %lf\n",
 Total_Result,Total_Error,(double)rt1/1000000.0,(double)rt2/1000000.0);

/*printf("Final total: %lg error: %lg\n",Total_Result,Total_Error);*/

process_terminate(PVM_tids);
} /* end of Main */

void error_pvm( char *s)
{
    printf("PVM %s\n" ,s);
    exit(-1);
} /* end error pvm */

void process_terminate(int *PVM_tids)
{
    int kill=0;
    pvm_initsend(PvmDataDefault);                /* send Flag (ie quit) command*/
    pvm_pkint( &kill ,1,1);
    pvm_mcast( PVM_tids, Total_Slaves , Flag );

    pvm_exit();
    exit(0);
}

I* end of file for parent gnzrul integral */

pvm/pvm_slave.h

1 #ifndef SLAVEH
#define SLAVEH

/* ! ! !ONLY ONE OF THE FOLLOWING CAN BE DEFINED!! ! ! */
/* New is a method of sending data via the command line. (not yet working)
 Old sends data via normal sends and is a linear split of the area
 Cubes lets the child figure the data by cubing the area up
 */
/**#define New*/
10 /**#define Old*/
/**#define Cubes*/
#define Cubes

/* Print out data as we go? (screws up timings) */
/**#define PRINT*/

/* send extra data to make sure children are getting all information */
/**#define Extra_Crap */

20 /* tells the slave to wait until it gets eps and go_flag instead of continuing
 on it merry way and finding the solution and waiting for the info until
 later */
/* #define Slow_Method */
/**#define Max_Slaves 1024        */ this MUST be a power of 2 ifndef Cubes */
#define Slave_Name "child"

#include "func/drvd01.c"

/* different message IDs */
#define Initial_Msg 1
#define I_am_Alive 2
#define First_Est 3
#define Start_Computation 4
#define Loop_Est 5
#define Flag 6
#define I.Quit 7
#define Doing_It 8
#define Div 200
#define TopLoop 201

/*
   This is an example of what a basic function file looks like.

   Jay Ball
   The DeRidder-Van Dooren Group

PI_DISCRIPTION A simple Periodic Function
*/
#include "parint.h"

#include "drvd.h"
#include <math.h>
#include "function.h"

#define dims 6

double start_drvd01[dims] = { 0.0, 0.0, 0.0, -1.0, -1.0, -1.0 };
double finish_drvd01[dims] = { 2.0, 1.0, PI_2, 1.0, 1.0, 1.0 };

int dim_drvd01 = dims;

int result_vector_size=1;

double drvd01(int N, double *x, double *r)
{
    *r = ( x[0]*x[1]*x[1]*sin(x[2]) / (4.0+x[3]+x[4]+x[5]) );
    return *r;
}

int initialize_drvd01()
{
    FunctionDimensions = dim_drvd01;
    Start = start_drvd01;
    Finish = finish_drvd01;
    /*
    ResultVectorSize=result_vector_size;
    */
    return 0;
}
In general, all user function files should \#include this file.

```
#define MaxFunctionDimensions 10 /* 0 .. 9 */
```
BIBLIOGRAPHY


