

Parallel Programming Laboratory
University of Illinois at Urbana-Champaign

PROJECTIONS Manual

By Mike DeNardo, Sid Cammeresi, Theckla Louchios, Orion Lawlor, Gengbin Zheng, Chee Wai Lee, Isaac Dooley, and Sindhura Bandhakavi

Version 7.0

University of Illinois
CHARM++/CONVERSE Parallel Programming System Software
Non-Exclusive, Non-Commercial Use License

Upon execution of this Agreement by the party identified below ("Licensee"), The Board of Trustees of the University of Illinois ("Illinois"), on behalf of The Parallel Programming Laboratory ("PPL") in the Department of Computer Science, will provide the CHARM++/CONVERSE Parallel Programming System software ("CHARM++") in Binary Code and/or Source Code form ("Software") to Licensee, subject to the following terms and conditions. For purposes of this Agreement, Binary Code is the compiled code, which is ready to run on Licensee's computer. Source code consists of a set of files which contain the actual program commands that are compiled to form the Binary Code.

1. The Software is intellectual property owned by Illinois, and all right, title and interest, including copyright, remain with Illinois. Illinois grants, and Licensee hereby accepts, a restricted, non-exclusive, non-transferable license to use the Software for academic, research and internal business purposes only, e.g. not for commercial use (see Clause 7 below), without a fee.
2. Licensee may, at its own expense, create and freely distribute complimentary works that interoperate with the Software, directing others to the PPL server (<http://charm.cs.uiuc.edu>) to license and obtain the Software itself. Licensee may, at its own expense, modify the Software to make derivative works. Except as explicitly provided below, this License shall apply to any derivative work as it does to the original Software distributed by Illinois. Any derivative work should be clearly marked and renamed to notify users that it is a modified version and not the original Software distributed by Illinois. Licensee agrees to reproduce the copyright notice and other proprietary markings on any derivative work and to include in the documentation of such work the acknowledgement:

"This software includes code developed by the Parallel Programming Laboratory in the Department of Computer Science at the University of Illinois at Urbana-Champaign."

Licensee may redistribute without restriction works with up to 1/2 of their non-comment source code derived from at most 1/10 of the non-comment source code developed by Illinois and contained in the Software, provided that the above directions for notice and acknowledgement are observed. Any other distribution of the Software or any derivative work requires a separate license with Illinois. Licensee may contact Illinois (kale@cs.uiuc.edu) to negotiate an appropriate license for such distribution.

3. Except as expressly set forth in this Agreement, THIS SOFTWARE IS PROVIDED "AS IS" AND ILLINOIS MAKES NO REPRESENTATIONS AND EXTENDS NO WARRANTIES OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO WARRANTIES OR MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE, OR THAT THE USE OF THE SOFTWARE WILL NOT INFRINGE ANY PATENT, TRADEMARK, OR OTHER RIGHTS. LICENSEE ASSUMES THE ENTIRE RISK AS TO THE RESULTS AND PERFORMANCE OF THE SOFTWARE AND/OR ASSOCIATED MATERIALS. LICENSEE AGREES THAT UNIVERSITY SHALL NOT BE HELD LIABLE FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL, OR INCIDENTAL DAMAGES WITH RESPECT TO ANY CLAIM BY LICENSEE OR ANY THIRD PARTY ON ACCOUNT OF OR ARISING FROM THIS AGREEMENT OR USE OF THE SOFTWARE AND/OR ASSOCIATED MATERIALS.
4. Licensee understands the Software is proprietary to Illinois. Licensee agrees to take all reasonable steps to insure that the Software is protected and secured from unauthorized disclosure, use, or release and will treat it with at least the same level of care as Licensee would use to protect and secure its own proprietary computer programs and/or information, but using no less than a reasonable standard of care. Licensee agrees to provide the Software only to any other person or entity who has registered with Illinois. If licensee is not registering as an individual but as an institution or corporation each member of the institution or corporation who has access to or uses Software must agree to and abide by the terms of this license. If Licensee becomes aware of any unauthorized licensing, copying or use of the Software, Licensee shall promptly notify Illinois in writing. Licensee expressly agrees to use the Software only in the manner and for the specific uses authorized in this Agreement.
5. By using or copying this Software, Licensee agrees to abide by the copyright law and all other applicable laws of the U.S. including, but not limited to, export control laws and the terms of this license. Illinois shall have the right to terminate this license immediately by written notice upon Licensee's breach of, or non-compliance with, any terms of the license. Licensee may be held legally responsible for any copyright infringement that is caused or encouraged by its failure to abide by the terms of this license. Upon termination, Licensee agrees to destroy all copies of the Software in its possession and to verify such destruction in writing.
6. The user agrees that any reports or published results obtained with the Software will acknowledge its use by the appropriate citation as follows:

"CHARM++/CONVERSE was developed by the Parallel Programming Laboratory in the Department of Computer Science at the University of Illinois at Urbana-Champaign."

Any published work which utilizes CHARM++ shall include the following reference:

"L. V. Kale and S. Krishnan. CHARM++: Parallel Programming with Message-Driven Objects. In 'Parallel Programming using C++' (Eds. Gregory V. Wilson and Paul Lu), pp 175-213, MIT Press, 1996."

Any published work which utilizes CONVERSE shall include the following reference:

"L. V. Kale, Milind Bhandarkar, Narain Jagathesan, Sanjeev Krishnan and Joshua Yelon. CONVERSE: An Interoperable Framework for Parallel Programming. Proceedings of the 10th International Parallel Processing Symposium, pp 212-217, April 1996."

Electronic documents will include a direct link to the official CHARM++ page at <http://charm.cs.uiuc.edu/>

7. Commercial use of the Software, or derivative works based thereon, REQUIRES A COMMERCIAL LICENSE. Should Licensee wish to make commercial use of the Software, Licensee will contact Illinois (kale@cs.uiuc.edu) to negotiate an appropriate license for such use. Commercial use includes:
 - (a) integration of all or part of the Software into a product for sale, lease or license by or on behalf of Licensee to third parties, or
 - (b) distribution of the Software to third parties that need it to commercialize product sold or licensed by or on behalf of Licensee.
8. Government Rights. Because substantial governmental funds have been used in the development of CHARM++/CONVERSE, any possession, use or sublicense of the Software by or to the United States government shall be subject to such required restrictions.
9. CHARM++/CONVERSE is being distributed as a research and teaching tool and as such, PPL encourages contributions from users of the code that might, at Illinois' sole discretion, be used or incorporated to make the basic operating framework of the Software a more stable, flexible, and/or useful product. Licensees who contribute their code to become an internal portion of the Software agree that such code may be distributed by Illinois under the terms of this License and may be required to sign an "Agreement Regarding Contributory Code for CHARM++/CONVERSE Software" before Illinois can accept it (contact kale@cs.uiuc.edu for a copy).

UNDERSTOOD AND AGREED.

Contact Information:

The best contact path for licensing issues is by e-mail to kale@cs.uiuc.edu or send correspondence to:

Prof. L. V. Kale
Dept. of Computer Science
University of Illinois
201 N. Goodwin Ave
Urbana, Illinois 61801 USA
FAX: (217) 333-3501

Contents

1	Introduction	3
2	Preparing the Charm++ Application	3
2.1	PROJECTIONS Tracing Modules at Application Link Time	3
2.1.1	Tracemode <code>projections</code>	3
2.1.2	Tracemode <code>summary</code>	4
2.2	General Runtime Options	4
2.3	PROJECTIONS API for CHARM++ Applications	5
2.3.1	Selective Tracing	5
2.3.2	User Events	5
2.3.3	Function-level Tracing for Adaptive MPI Applications	6
3	Advanced Tracing Features	7
3.1	End-of-run Analysis for Data Reduction	7
4	The Projections Performance Visualization Tool	8
4.1	Building PROJECTIONS	8
4.2	Visualization and Analysis using PROJECTIONS	8
4.2.1	Starting Up	8
4.2.2	Available Tools	9
4.3	Performance Views	10
4.3.1	Graphs	10
4.3.2	Timelines	12
4.3.3	Usage Profile	14
4.3.4	Communication	15
4.3.5	Communication vs Time	16
4.3.6	View Log Files	17
4.3.7	Histograms	18
4.3.8	Overview	20
4.3.9	Animations	21
4.3.10	Time Profile Graph	21
4.3.11	User Events Profile	22
4.3.12	Outlier Analysis	22
4.3.13	Multirun Analysis	24
4.3.14	Function Tool	24
4.3.15	AMPI Usage Profile	25
4.3.16	NoiseMiner View	25
4.4	Miscellaneous features	26
4.4.1	Standard Graph Display Interface	26
4.4.2	Standard Dialog Features	27
4.4.3	Data Fields	28
5	Known Issues	28

1 Introduction

PROJECTIONS is a performance analysis/visualization framework that helps you understand and investigate performance-related problems in your parallel (CHARM++) application. It is a framework with an event tracing component with features that allow you to control the amount of information generated and to a lesser degree the amount of perturbation the tracing activities introduce into the application. It also has a Java-based visualization and analysis component with various views that will help present the performance information in a visually useful manner.

Performance analysis with PROJECTIONS typically involves 2 simple steps:

1. Prepare your application code by linking with the appropriate trace generation modules and executing it to generate trace data. (see section 2)
2. Using the Java-based tool to visually study various aspects of the performance information to locate application execution performance problems. (see section 4)

2 Preparing the Charm++ Application

The CHARM++ runtime automatically records pertinent performance data at performance-related events encountered by the runtime. These events include the start and end of entry method execution, message sends from entry methods and scheduler idle time. This means *most* users will not need to manually insert code into their applications in order to generate trace data. In scenarios where special performance information not captured by the runtime is required, an API (see section 2.3) is available for user-specific events with some support for visualization by the Java-based tool. If greater control over tracing activities (e.g. dynamically turning instrumentation on and off) is desired, the API also allows users to insert code into their applications for such purposes.

The automatic recording of events by the PROJECTIONS framework introduces the overhead of an if-statement for each runtime event, even if no performance analysis traces are desired. Developers of CHARM++ applications who consider such an overhead to be unacceptable (e.g. for a production application which requires the absolute best performance) may recompile the CHARM++ runtime with the `--with-production` flag which removes the instrumentation stubs.

To enable performance tracing of your application, users simply need to link the appropriate trace data generation module(s) (also referred to as *tracemode(s)*). (see section 2.1)

2.1 Projections Tracing Modules at Application Link Time

PROJECTIONS tracing modules dictate the type of performance data, data detail and data format each processor will record. They are also referred to as “tracemodes”. There are currently 2 tracemodes available. Zero or more tracemodes may be specified at link-time. When no tracemodes are specified, no trace data is generated.

2.1.1 Tracemode projections

Link time option: `-tracemode projections`

This tracemode generates detailed event log files that contain information about all CHARM++ events like entry method calls and message packing during the execution of the program. The data will be used by PROJECTIONS in visualization and analysis.

This tracemode will generate a single symbol table file and *p* ASCII log files for *p* processors. The names of the log files will be `NAME.#.log` where `NAME` is the name of your executable and `#` is the processor `#`. The name of the symbol table file is `NAME.sts` where `NAME` is the name of your executable.

This is the main source of data expected by the performance visualizer. Certain tools like timeline will not work without the detailed data from this tracemode.

The following is a list of runtime options available under this tracemode:

- **+logsize NUM**: keep only NUM log entries in the memory of each processor. The logs are emptied and flushed to disk when filled.
- **+binary-trace**: generate projections log in binary form.
- **+gz-trace**: generate gzip (if available) compressed log files.
- **+checknested**: a debug option. Checks if events are improperly nested while recorded and issue a warning immediately.
- **+trace-subdirs NUM**: divide the generated log files among NUM subdirectories of the trace root, each named `PROGNAME.projdir.K`

2.1.2 Tracemode summary

Compile option: `-tracemode summary`

In this tracemode, execution time across all entry points for each processor is partitioned into a fixed number of equally sized time-interval bins. These bins are globally resized whenever they are all filled in order to accomodate longer execution times while keeping the amount of space used constant.

Additional data like the total number of calls made to each entry point is summarised within each processor.

This tracemode will generate a single symbol table file and p ASCII summary files for p processors. The names of the summary files will be `NAME.#.sum` where `NAME` is the name of your executable and `#` is the processor `#`. The name of the symbol table file is `NAME.sum.sts` where `NAME` is the name of your executable.

This tracemode can be used to control the amount of output generated in a run. It is typically used in scenarios where a quick look at the overall utilization graph of the application is desired to identify smaller regions of time for more detailed study. Attempting to generate the same graph using the detailed logs of the prior tracemode may be unnecessarily time consuming or impossible.

The following is a list of runtime options available under this tracemode:

- **+bincount NUM**: use NUM time-interval bins. The bins are resized and compacted when filled.
- **+binsize TIME**: sets the initial time quantum each bin represents.
- **+version**: set summary version to generate.
- **+sumDetail**: Generates a additional set of files, one per processor, that stores the time spent by each entry method associated with each time-bin. The names of “summary detail” files will be `NAME.#.sumd` where `NAME` is the name of your executable and `#` is the processor `#`.
- **+sumOnly**: Generates an additional file that stores a single utilization value per time-bin, averaged across all processors. This file bears the name `NAME.sum` where `NAME` is the name of your executable. This runtime option currently overrides the `+sumDetail` option.

2.2 General Runtime Options

The following is a list of runtime options available with the same semantics for all tracemodes:

- **+traceroot DIR**: place all generated files in `DIR`.
- **+traceoff**: trace generation is turned off when the application is started. The user is expected to insert code to turn tracing on at some point in the run.
- **+traceWarn**: By default, warning messages from the framework are not displayed. This option enables warning messages to be printed to screen. However, on large numbers of processors, they can overwhelm the terminal I/O system of the machine and result in unacceptable perturbation of the application.

2.3 Projections API for Charm++ Applications

2.3.1 Selective Tracing

CHARM++ allows user to start/stop tracing the execution at certain points in time on the local processor. Users are advised to make these calls on all processors and at well-defined points in the application.

Users may choose to have instrumentation turned off at first (by command line option `+traceoff` - see section 2.2) if some period of time in middle of the application's execution is of interest to the user.

Alternatively, users may start the application with instrumentation turned on (default) and turn off tracing for specific sections of the application.

Again, users are advised to be consistent as the `+traceoff` runtime option applies to all processors in the application.

- `void traceBegin()`

Enables the runtime to trace events (including all user events) on the local processor where `traceBegin` is called.

- `void traceEnd()`

Prevents the runtime from tracing events (including all user events) on the local processor where `traceEnd` is called.

2.3.2 User Events

PROJECTIONS has the ability to visualize traceable user specified events. User events are usually displayed in the Timeline view as vertical bars above the entry methods. Alternatively the user event can be displayed as a vertical bar that vertically spans the timelines for all processors. Follow these following basic steps for creating user events in a charm++ program:

1. Register an event with an identifying string and either specify or acquire a globally unique event identifier. All user events that are not registered will be displayed in white.
2. Use the event identifier to specify trace points in your code of interest to you.

The functions available are as follows:

- `int traceRegisterUserEvent(char* EventDesc, int EventNum=-1)`

This function registers a user event by associating `EventNum` to `EventDesc`. If `EventNum` is not specified, a globally unique event identifier is obtained from the runtime and returned. The string `EventDesc` must either be a constant string, or it can be a dynamically allocated string that is **NOT** freed by the program. If the `EventDesc` contains a substring `“***”` then the Projections Timeline tool will draw the event vertically spanning all PE timelines.

`EventNum` has to be the same on all processors. Therefore use one of the following methods to ensure the same value for any PEs generating the user events:

1. Call `traceRegisterUserEvent` on PE 0 in `main::main` without specifying an event number, and store returned event number into a readonly variable.
2. Call `traceRegisterUserEvent` and specify the event number on processor 0. Doing this on other processors would have no effect. Afterwards, the event number can be used in the following user event calls.

Eg. `traceRegisterUserEvent("Time Step Begin", 10);`

Eg. `eventID = traceRegisterUserEvent('“Time Step Begin”');`

There are two main types of user events, bracketed and non bracketed. Non-bracketed user events mark a specific point in time. Bracketed user events span an arbitrary contiguous time range. Additionally, the user can supply a short user supplied text string that is recorded with the event in the log file. These strings should not contain newline characters, but they may contain simple html formatting tags such as
, , <i>, , etc.

The calls for recording user events are the following:

- `void traceUserEvent(int EventNum)`

This function creates a user event that marks a specific point in time.

Eg. `traceUserEvent(10);`

- `void traceUserBracketEvent(int EventNum, double StartTime, double EndTime)`

This function records a user event spanning a time interval from `StartTime` to `EndTime`. Both `StartTime` and `EndTime` should be obtained from a call to `CmiWallTimer()` at the appropriate point in the program.

Eg.

```
traceRegisterUserEvent("Critical Code", 20); // on PE 0
double critStart = CmiWallTimer(); // start time
// do the critical code
traceUserBracketEvent(20, critStart, CmiWallTimer());
```

- `void traceUserSuppliedNote(char * note)`

This function records a user specified text string at the current time.

- `void traceUserSuppliedBracketedNote(char *note, int EventNum, double StartTime, double EndTime)`

This function records a user event spanning a time interval from `StartTime` to `EndTime`. Both `StartTime` and `EndTime` should be obtained from a call to `CmiWallTimer()` at the appropriate point in the program.

Additionally, a user supplied text string is recorded, and the `EventNum` is recorded. These events are therefore displayed with colors determined by the `EventNum`, just as those generated with `traceUserBracketEvent` are.

2.3.3 Function-level Tracing for Adaptive MPI Applications

Adaptive MPI (AMPI) is an implementation of the MPI interface on top of CHARM++. As with standard MPI programs, the appropriate semantic context for performance analysis is captured through the observation of MPI calls within C/C++/Fortran functions. Unfortunately, AMPI's implementation does not grant the runtime access to information about user function calls. As a result, the tracing framework must provide an explicit API for capturing this piece of performance information in addition to MPI calls (which are known to the runtime).

The functions, similar to those used to capture user events, are as follows:

- `int _TRACE_REGISTER_FUNCTION_NAME(const char *name);`

This function registers an AMPI function `name`. The tracing framework assigns to function `name` a unique id and returns it. It is the user's responsibility to ensure that `name` is unique and consistent.

This registration function should be called near the start of the application, just after `MPI.Init`.

- `int _TRACE_REGISTER_FUNCTION_ID(const char *name, int idx);`

This function registers an AMPI function `name` to be associated explicitly to the id `idx`. It is the user's responsibility to ensure that `name` as well as `idx` are unique and consistent. The tracing framework returns `idx`.

This is an alternative registration function and should be called near the start of the application just after `MPI_Init`.

- `void _TRACE_BEGIN_FUNCTION_NAME(const char *name);`

This function tells the tracing framework to record a begin event associated with the registered function `name`. If this were called in a C/C++ environment with pre-processor support, the line number of the function call will also be recorded.

- `void _TRACE_BEGIN_FUNCTION_ID(int idx);`

This function tells the tracing framework to record a begin event associated with the registered function indexed by `idx`. If this were called in a C/C++ environment with pre-processor support, the line number of the function call will also be recorded.

- `void _TRACE_END_FUNCTION_NAME(const char *name);`

This function tells the tracing framework to record a end event associated with the registered function `name`.

- `void _TRACE_END_FUNCTION_ID(int idx);`

This function tells the tracing framework to record a end event associated with the registered function indexed by `idx`.

AMPI function events captured by the use of this API are recognized by the visualization system and used for special AMPI-specific views in addition to standard CHARM++ entry methods.

3 Advanced Tracing Features

3.1 End-of-run Analysis for Data Reduction

As applications are scaled to thousands or hundreds of thousands of processors, the amount of data generated becomes extremely large and potentially unmanagable by the visualization tool. At the time of this documentation, PROJECTIONS is capable of handling data from 8000+ processors but with somewhat severe tool responsiveness issues. We have developed an approach to mitigate this data size problem with options to trim-off “uninteresting” processors’ data by not writing such data at the end of an application’s execution.

This is currently done through heuristics to pick out interesting extremal (i.e. poorly behaved) processors and at the same time using a k-means clustering to pick out exemplar processors from equivalence classes to form a representative subset of processor data. The analyst is advised to also link in the summary module via `+tracemode summary` and enable the `+sumDetail` option in order to retain some profile data for processors whose data were dropped.

- `+extrema`: enables extremal processor identification analysis at the end of the application’s execution.
- `+numClusters`: determines the number of clusters (equivalence classes) to be used by the k-means clustering algorithm for determining exemplar processors. Analysts should take advantage of their knowledge of natural application decomposition to guess at a good value for this.

This feature is still being developed and refined as part of our research. It would be appreciated if users of this feature could contact the developers if you have input or suggestions.

4 The Projections Performance Visualization Tool

The PROJECTIONS Java-based visualization tool (henceforth referred to as simply PROJECTIONS) comes pre-built with the CHARM++ source release. It can be located at `CHARM_LOCATION/tools/projections` which will henceforth be referred to as `PROJECTIONS_LOCATION`.

4.1 Building Projections

To rebuild PROJECTIONS (optional) from the source:

- 1) Make sure the JDK commands “java”, “javac”, “ant”, and “jar” are in your path
- 2) Make sure that your versions of java and javac are at least 1.5. Do this by running “java -version” and “javac -version”. Also, make sure the environment variable `JAVA_HOME` is not pointing at an old version of java.
- 3) From `PROJECTIONS_LOCATION/`, type “ant clean” then “ant”
- 4) The following files are placed in ‘bin’:
 - `projections` : Starts projections, for UNIX machines
 - `projections.bat` : Starts projections, for Windows machines
 - `projections.jar` : archive of all the java and image files

4.2 Visualization and Analysis using Projections

4.2.1 Starting Up

From any location, type:

```
> PROJECTIONS_LOCATION/bin/projections [NAME.sts]
```

where `PROJECTIONS_LOCATION` is the path to the main projections directory.

Available options to the visualization component of PROJECTIONS include:

- `-h` or `--help`: displays help information about available options.
- `-V` or `--version`: displays current PROJECTIONS version number.
- `-u` or `--use-version < ver >`: overrides the data interpretation behavior of PROJECTIONS to explicitly use *ver* instead of the current version. This is useful in scenarios where the latest version of PROJECTIONS is not backward-compatible with older log formats.
- `-no-idle`: tells PROJECTIONS to ignore idle time information contained in the logs.
- `-bgsz < x >< y >< z >`: tells PROJECTIONS to compute additional derived information by assuming a logical 3-D Torus topology with the specified dimensionality and a processor-to-torus placement policy that matches CHARM++’s placement policy on the BG/L class of machines. The presence of this option enables additional communication visualization options (see later). Note that this option is really meant to be used for logs generated from virtual-node mode BG/L executions. The additional information derived from any other logs would probably be misleading.
- `-print_usage`: tells PROJECTIONS to also write to standard output the detailed graph numbers when viewing Usage Profiles (see later). This is useful for generating visuals that are better expressed by tools such as gnuPlot than through screen captures of PROJECTIONS plots.

Supplying the optional `NAME.sts` file in the command line will cause PROJECTIONS to load data from the file at startup. This shortcut saves time selecting the desired dataset via the GUI’s file dialog.

When PROJECTIONS is started, it will display a main window as shown in figure 1. If summary (`.sum`) files are available in the set of data, a low-resolution utilization graph (Summary Display) will be displayed as shown. If summary files are not available, or if PROJECTIONS was started without supplying the optional `NAME.sts` file, the main window will show a blank screen.

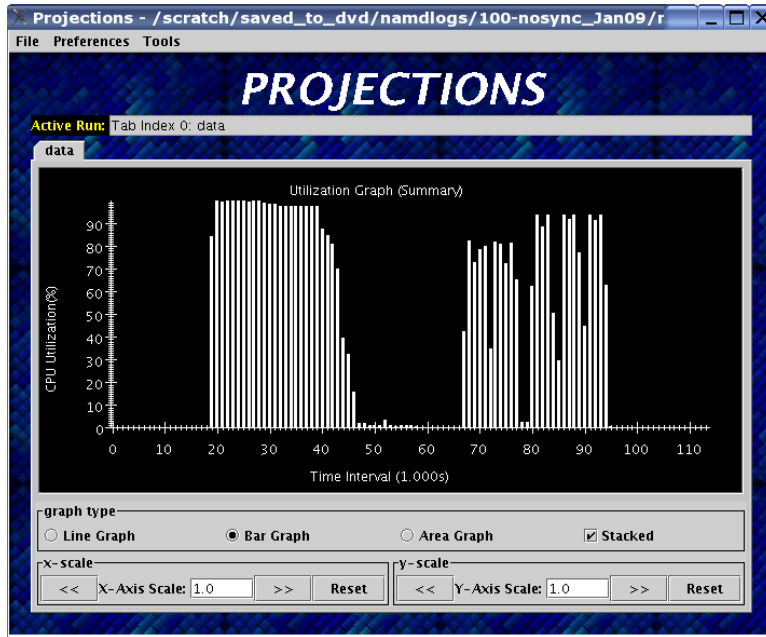


Figure 1: PROJECTIONS main window

- **File** contains 3 options. *Open File(s)* allows you to explicitly load a data set. This happens if you had not specified a `NAME.sts` file in the command line when starting PROJECTIONS or if you wish to explicitly load a new dataset. It brings up a dialog box that allows you to select the location of the dataset you wish to study. Navigate to the directory containing your data and select the `.sts` file. Click on 'Open'. If you have selected a valid file, PROJECTIONS will load in some preliminary data from the files and then activate the rest of the options under the menu item **Tools**. *Close current data* currently works the same way as *Close all data*. They unload all current PROJECTIONS data so one can load in a new set of data. They will also deactivate the individual items provided in the **Tools** menu option.
- **Preferences** generally allows you to set foreground or background colors and entry method color schemes. This is useful for configuring the color schemes of PROJECTIONS windows to be print-friendly.
- **Tools** lists the set of available tools for analysis of generated trace data. It will be described in great detail under section 4.2.2.

The Summary Display loaded on the Main Window displays basic processor utilization data (averaged across all processors) over time intervals. This is provided by the data generated by the summary tracemode. This view offers no special features over and above the **Standard Graph Display** described in section 4.4. Please refer the appropriate section on information for using its available features.

There should not be any serious performance issues involved in the loading of summary data on the main window.

4.2.2 Available Tools

The following tools and views become available to you after a dataset has been loaded (with the exception of Multirun Analysis) and may be accessed via the menu item **Tools**:

- The **Graphs** view is where you can analyze your data by breaking it into any number of intervals and look at what goes on in each of those intervals.
- The **Timelines** view lets you look at what a specific processor is doing at each moment of the program. It is the most detailed view of a parallel application PROJECTIONS offers (and correspondingly, the most resource-hungry).

- The **Usage Profile** view lets you see percentage-wise what entry methods each processor spends its time on during a specified time range. It is particularly useful for identifying load imbalance and the probable offending entry method.
- The **Communication** view is a general tool that presents communication properties contributed by each entry point across the processors.
- The **Log File Viewer** provides a human-readable, verbose interpretation of a log file's entries.
- The **Histograms** view presents entry point or communication histogram information (ie. the frequency of occurrence of events given an activity property like time bin size or message size on the x-axis).
- The **Overview** view gives user an overview of the utilization of all processors during the execution. It is an extremely useful initial tool to begin your performance analysis efforts with as it provides an overall picture of application performance while being very light-weight at the same time.
- The **Animation** view animates the processor usage over a specified range of time and a specified interval size.
- The **Time Profile Graph** view is a more detailed presentation of the **Graphs** utilization view in that it presents the time contribution by each entry point across the desired time interval. While the **Graphs** view can show the same data, it is unable to stack the entry points, which proves useful in some cases.

4.3 Performance Views

4.3.1 Graphs

The Graphs window (see figure 2) is where you can analyze your data by breaking it into any number of intervals and look at what goes on in each of those intervals.

When the Graph Window first appears, a dialog box will also appear. It will ask for the following information (Please refer to 4.4 for information on special features you can use involving the various fields)::

- Processor(s): Choose which processor(s) you wish to visualize graph information for.
- Start Time : Choose the starting time of interest. A time-based field.
- End Time : Choose the ending time of interest. A time-based field.
- Interval Size : Determine the size of an interval. The number of intervals will also be determined by this value (End Time - Start Time divided by Interval Size). A time-based field.

Standard PROJECTIONS dialog options and buttons are also available (see 4.4 for details).

The following menu items are available:

- **File** contains 2 options: *Print Graph* uses Java's built-in print manager feature to render the tool's displays (usually to a printer or a file depending on the platform on which Java is supported). Note that the current implementation of the renderer does not behave in exactly the same fashion as the screen renderer, so you should expect the output to look somewhat different. *Close* simply closes the Graph window.
- **Tools** contains 2 options: *Set Interval Size* reloads the dialog box so you could select a new time range over which to view Graph data. *Timeline* is currently not implemented. Its intended as a convenient way to load Timeline data (see section 4.3.2) over the same parameters as the current Graph view.

The amount of time to analyze your data depends on several factors, including the number of processors, number of entries, and number of intervals you have selected. A progress meter will show the amount of data loaded so far. The meter will not, however, report rendering progress which is determined mainly by the number of intervals selected. As a rule of thumb, limit the number of intervals to 1,000 or less.

The Graph Window has 3 components in its display:

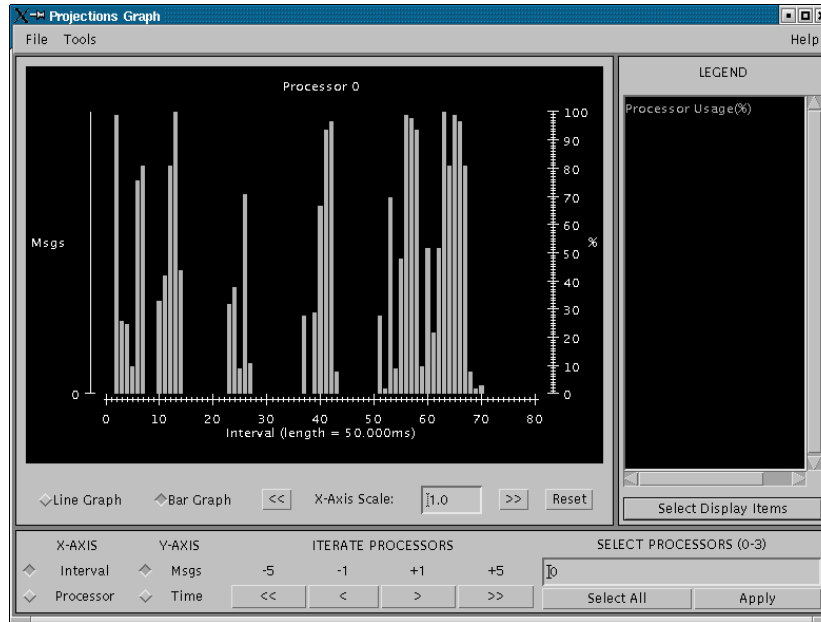


Figure 2: Graph tool

1) **Display Panel** (located : top-left area)

- Displays title, graph, and axes. To the left is a y-axis bar for detailed information involving the number of messages sent or time executed depending on the **Control Panel** toggle selected (see below). To the right is a y-axis bar for average processor-utilization information. The x-axis may be based on time-interval or per-processor information depending on the appropriate **Control Panel** toggle.
- Allows you to toggle display between a line graph and a bar graph.
- Allows you to scale the graph along the X-axis. You can either enter a scale value ≥ 1.0 in the text box, or you can use the << and >> buttons to increment/decrement the scale by .25. Clicking on Reset sets the scale back to 1.0. When the scale is greater than 1.0, a scrollbar will appear along the bottom of the graph to let you scroll back and forth.

2) **Legend Panel** (located : top-right area)

- Shows what information is currently being displayed on the graph and what color represents that information.
- Click on the 'Select Display Items' button to bring up a window to add/remove items from the graph and to change the colors of the items:
 - * The **Select Display Items** window shows a list of items that you can display on the graph. There are 3 main sections: System Usage, System Msgs, and User Entries. The System Usage and System Msgs are the same for all programs. The User Entries section has program-specific items in it.
 - * Click on the checkbox next to an item to have it displayed on the graph.
 - * Click on the colorbox next to an item to modify its color.
 - * Click on 'Select All' to choose all of the items
 - * Click on 'Clear All' to remove all of the items
 - * Click on 'Apply' to apply you choices/changes to the graph
 - * Click on 'Close' to exit

3) Control Panel (located : bottom area)

- Allows you to toggle what is displayed on the X-axis. You can either have the x-axis display the data by interval or by processor.
- Allows you to toggle what is displayed on the Y-axis. You can either have the y-axis display the data by the number of msgs sent or by the amount of time taken.
- Allows you to change what data is being displayed by iterating through the selections. If you have selected an x-axis type of 'interval', that means you are looking at what goes on in each interval for a specific processor. Clicking on the <<, <, >, >> buttons will change the processor you are looking at by either -5, -1, +1, or +5. Conversely, if you have an x-axis of 'processor', then the iterate buttons will change the value of the interval that you are looking at for each processor.
- Allows you to indicate which intervals/processors you want to examine. Instead of just looking at one processor or one interval, the box and buttons on the right side of this panel let you choose any number or processors/intervals to look at. This field behaves like a processor field. Please refer to section 4.4 for more information about the special features on using processor fields. Clicking on 'Apply' updates the graph with your choices. Clicking on 'Select All' chooses the entire processor range. When you select more than one processor's worth of data to display, the graph will show the desired information summed across all selected processors. The exception to this is processor utilization data which is always displayed as data averaged across all selected processors.

4.3.2 Timelines

The Timeline window (see figure 3) lets you look at what a specific processor is doing at each moment of the program.

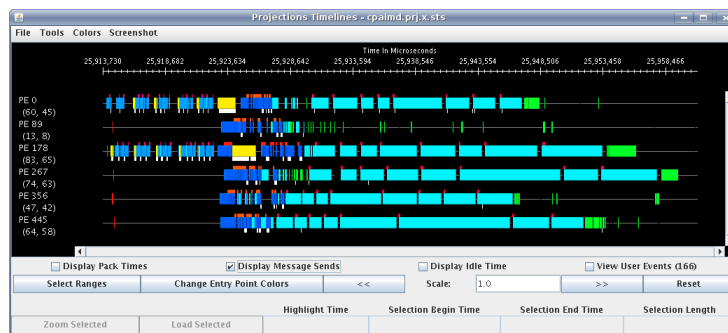


Figure 3: Timeline Tool

When opening a Timeline view, a dialog box appears. The box asks for the following information (Please refer to 4.4 for information on special features you can use involving the various fields):

- Processor(s): Choose which processor(s) you want to see in the timeline.
- Start Time : Choose what time you want your timeline to start at. A time-based field.
- End Time : Choose what time you want your timeline to end at. A time-based field.

Standard PROJECTIONS dialog options and buttons are also available (see 4.4 for details). The following menu options are available:

- **File** contains 1 enabled option: *Close* simply closes the Timeline Window.
- **Tools** contains 1 option: *Modify Ranges* opens the initial dialog box thereby allowing you to select new set of processors or time duration parameters.

- **Colors** contains options for loading, using, and modifying color schemes. *Change Colors* functions in a manner similar to the button of the same name described under control panel information below. *Save Colors* allows you to save the current color set to a file named “color.map” into the same directory where your data logs are stored. Note that the directory must have write permissions for you before this can work. We are currently working on a more flexible scheme for storing saved color sets. *Restore Colors* allows you to load any previously saved color sets described above. *Default Colors* resets the current color set to the default set that PROJECTIONS assigns without user intervention.

Other color schemes are provided that can be used for some applications. The colors set as described above are the default coloring scheme. Other options for coloring the events are by event id (chare array index), user supplied parameter, or memory usage. In order to color by a user supplied parameter such as timestep, the C function `traceUserSuppliedData(int value);` should be called within some entry methods. If such a method is called in an entry method, the entry method invocation can be colored by the parameter. The user supplied data can also be viewed in the tooltip that appears when the cursor hovers over an entry method invocation in the window. To color by memory usage, the C function `traceMemoryUsage();` should be called in all entry methods. The call records the current memory usage. Red indicates high memory usage, and green indicates low memory usage. The actual memory usage can also be viewed in the tooltips that appear when the cursor is over an event. The memory usage is only available in when using a Charm++ version that uses gnu memory.

- **Screenshot** contains 1 option: *Save as JPG or PNG* save the visible portion of the visualization to an image file. You must choose a filename ending with a `.png` or `.jpg` extension when choosing the location to save the image. The appropriate filetype is chosen based on the chosen filename extension. If the view is zoomed in, only the portion currently shown on screen is saved.

The Timeline Window consists of two parts:

1) **Display Panel** (located: top area)

This is where the timelines are displayed and is the largest portion of the window. The time axis is displayed at the top of the panel. The left side of the panel shows the processor labels, each containing a processor number and two strange numbers. These two numbers represent the percentage of the loaded timeline during which work occurs. The first of the two numbers is the “non-idle” time, i.e. the portion of the time in the timeline not spent in idle regions. This contains both time for entry methods as well as other uninstrumented time spent likely in the Charm++ runtime. The second number is the percentage of the time used by the entry methods for the selected range.

The timeline itself consists of colored bars for each event. Placing the cursor over any of these bars will display information about the event including: the name, the begin time, the end time, the total time, the time spent packing, the number of messages it created, and which processor created the event.

Left clicking on an event bar will cause a window to popup. This window contains detailed information about the messages sent by the clicked upon event.

Right clicking on an event bar will cause a line to be drawn to the beginning of the event bar from the point where the message causing the event originated. This option may not be applicable for threaded events. If the message originated on a processor not currently included in the visualization, the other processor will be loaded, and then the message line will be drawn. A warning message will appear if the message origination point is outside the time duration, and hence no line will be drawn.

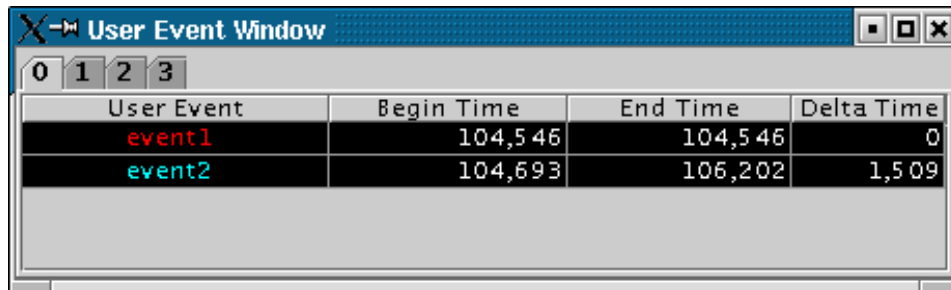
User events are displayed as bars above the ordinary event bars in the display area. If the name of the user event contains a substring “***” then the bar will vertically span the whole screen.

Message pack times and send points can be displayed below the event bars. The message sends are small white tick marks, while the message pack times are small pink bars usually occurring immediately after the message send point. If zoomed in to a point where each microsecond takes more than one pixel, the message send point and the following packing time may appear disconnected. This is an inherent problem with the granularity used for the logfiles.

2) Control Panel (located: bottom area)

The controls in this panel are obvious, but we mention one here anyway.

View User Event - Checking this box will bring up a new window showing the string description, begin time, end time and duration of all user events on each processor. You can access information on user events on different processors by accessing the numbered tabs near the top of the display.



The screenshot shows a window titled "User Event Window" with a tabbed interface. The active tab is labeled "0". Below the tabs is a table with the following data:

User Event	Begin Time	End Time	Delta Time
event1	104,546	104,546	0
event2	104,693	106,202	1,509

Figure 4: User Event Window

Various features appear when the user moves the mouse cursor over the top axis. A vertical line will appear to highlight a specific time. The exact time will be displayed at the bottom of the window. Additionally a user can select a range by clicking while a time is highlighted and dragging to the left or right of that point. As a selection is being made, a vertical white line will mark the beginning and end of the range. Between these lines, the background color for the display will change to gray to better distinguish the selection from the surrounding areas. After a selection is made, its duration is displayed at the bottom. A user can zoom into the selection by clicking the "Zoom Selected" button. To release a selection, single-click anywhere along the axis. Clicking "Load Selected" when a selection is active will cause the timeline range to be reloaded. To zoom out, the "||" or "Reset" button can be used.

To then zoom into the selected area via this interface, click on either the "Zoom Selected" or the "Load Selected" buttons. The difference between these two buttons is that the "Load Selected" zooms into the selected area and discards any events that are outside the time range. This is more efficient than "Zoom Selected" as the latter draws all the events on a virtual canvas and then zooms into the canvas. The disadvantage of using "Load Selected" is that it becomes impossible to zoom back out without having to re-specify the time range via the "Select Ranges" button.

Performance-wise, this is the most memory-intensive part of the visualization tool. The load and zoom times are proportional to the number of events displayed. The user should be aware of how event-intensive the application is over the desired time-period before proceeding to use this view. If PROJECTIONS takes too long to load a timeline, cancel the load and choose a smaller time range or fewer processors. We expect to add features to alleviate this problem in future releases.

4.3.3 Usage Profile

The Usage Profile window (see figure 5) lets you see percentage-wise what each processor spends its time on during a specified period.

When the window first comes up, a dialog box appears asking for the processor(s) you want to look at as well as the time range you want to look at. This dialog functions in exactly the same way as for the Timeline tool (see section 4.3.2).

The following menu options are available in this view:

- **File** has 2 options: *Select Processors* reloads the dialog box for the view and allows you to select a new processor and time range for this view. *Print Profile* currently does nothing. This will be addressed in a later release of PROJECTIONS.

The following components are supported in this view:

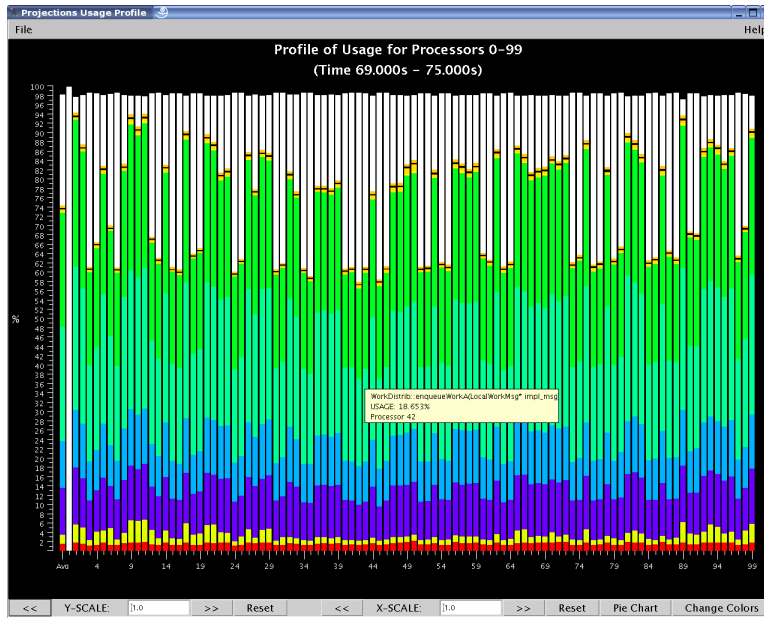


Figure 5: Usage Profile

- 1) **Main Display** (located: top area) The left axis of the display shows a scale from 0% to 100%. The main part of the display shows the statistics. Each processor is represented by a vertical bar with the leftmost bar representing the statistics averaged across all processors. The bottom of the bar always shows the time spent in each entry method (distinguished by the entry method's assigned color) . Above that is always reported the message pack time (in black), message unpack time (in orange) and idle time (in white). Above this, if the information exists, are colored bars representing communication CPU overheads contributed by each entry method (again, distinguished by the same set of colors representing entry methods). Finally the black area on top represents time overheads that the CHARM++ runtime cannot account for.

If you mouse-over a portion of the bar (with the exception of the black area on top), a pop-up window will appear telling you the name of the item, what percent of the usage it has, and the processor it is on.

- 2) **Control Panel** (located: bottom area) The panellets you adjust the scales in both the X and Y directions. The X direction is useful if you are looking at a large number of processors. The Y direction is useful if there are small-percentage items for a processor. The “Reset” button allows you to reset the X and Y scales.

The “Pie Chart” button generates a pie chart representation (see figure 6) of the same information using averaged statistics but without idle time and communication CPU overheads.

The “Change Colors” button lists all entry methods displayed on the main display and their assigned colors. It allows you to change those assigned colors to aid in highlighting entry methods.

The resource consumption of this view is moderate. Load times and visualization times should be relatively fast, but dismissing the tool may result in a very slight delay while PROJECTIONS reclaims memory through Java's garbage collection system.

4.3.4 Communication

The communication tool (see figure 7) visualizes communication properties on each processor over a user-specified time range.

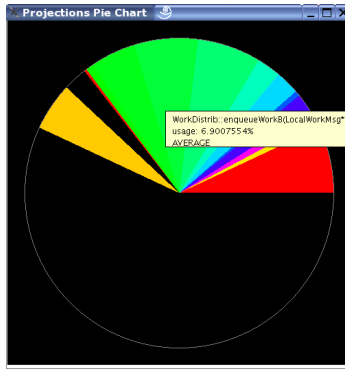


Figure 6: Pie Chart representation of average usage

The dialog box of the tool allows you to specify the time period within which to load communication characteristics information. This dialog box is exactly the same as that of the Timeline tool (see section 4.3.2).

The main component employs the standard capabilities provided by PROJECTIONS' standard graph (see 4.4).

The control panel allows you to switch between the following communication characteristics:

- Number of Messages Sent by entry methods (initial default view);
- Number of Bytes Sent by entry methods;
- Number of Messages Received by entry methods;
- Number of Bytes Received by entry methods;
- Number of Messages Sent externally (physically) by entry methods;
- Number of Bytes Sent externally (physically) by entry methods;
- and Number of hops messages travelled before being received by an entry methods. This is available when the runtime option `-bgsiz` (See section 4.2.1) is supplied.

This view uses memory proportional to the number of processors selected.

4.3.5 Communication vs Time

The communication over time tool (see figure 8) visualizes communication properties over all processors and displayed over a user-specified time range on the x-axis.

The dialog box of the tool allows you to specify the time period within which to load communication characteristics information. This dialog box is exactly the same as that of the Communication tool (see section 4.3.4).

The main component employs the standard capabilities provided by PROJECTIONS' standard graph (see 4.4).

The control panel allows you to switch between the following communication characteristics:

- Number of Messages Sent by entry methods (initial default view);
- Number of Bytes Sent by entry methods;
- Number of Messages Received by entry methods;
- Number of Bytes Received by entry methods;

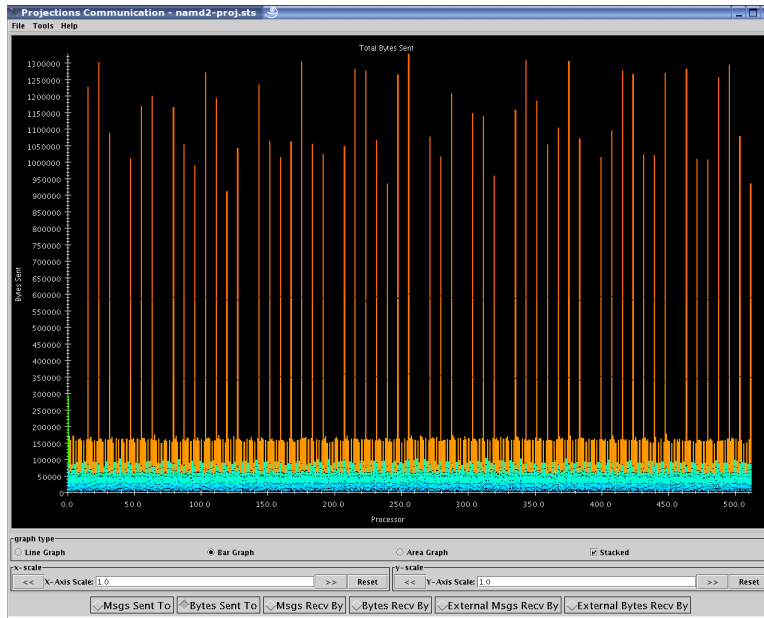


Figure 7: Communication View

- Number of Messages Sent externally (physically) by entry methods;
- Number of Bytes Sent externally (physically) by entry methods;
- and Number of hops messages travelled before being received by an entry methods (available only on trace logs generated on the Bluegene machine).

This view has no known problems loading any range or volume of data.

4.3.6 View Log Files

This window (see figure 9) lets you see a translation of a log file from a bunch of numbers to a verbose version. A dialog box asks which processor you want to look at. After choosing and pressing OK, the translated version appears. Note that this is *not* a standard processor field. This tool will only load *exactly* one processor's data.

Each line has:

- a line number (starting at 0)
- the time the event occurred at
- a description of what happened.

This tool has the following menu options:

- **File** has 2 options: *Open File* reloads the dialog box and allows the user to select a new processor's data to be loaded. *Close* closes the current window.
- **Help** has 2 options: *Index* currently does not do anything. This will be addressed in a later release of PROJECTIONS. *About* currently does not do anything. This will also be addressed in a later release of PROJECTIONS.

The tool has 2 buttons. "Open File" reloads the dialog box (described above) and allows the user to select a new processor's data to be loaded. "Close Window" closes the current window.

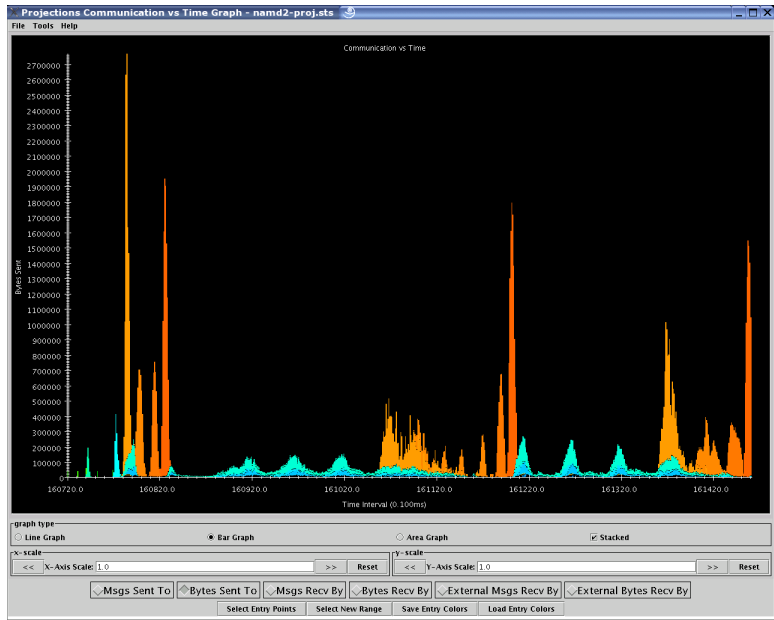


Figure 8: Communication View over Time

4.3.7 Histograms

This module (see figure 10) allows you to examine the performance property distribution of all your entry points (EP). It gives a histogram of different number of EP's that have the following properties falling in different property bins:

The dialog box for this view asks the following information from the user. (Please refer to 4.4 for information on special features you can use involving the various fields):

- Processor(s): Choose which processor(s) you wish to visualize histogram information for.
- Start Time: Choose the starting time of interest. A time-based field.
- End Time: Choose the ending time of interest. A time-based field.
- Number of Bins: Select the number of property bins to fit frequency data under. A simple numeric field.

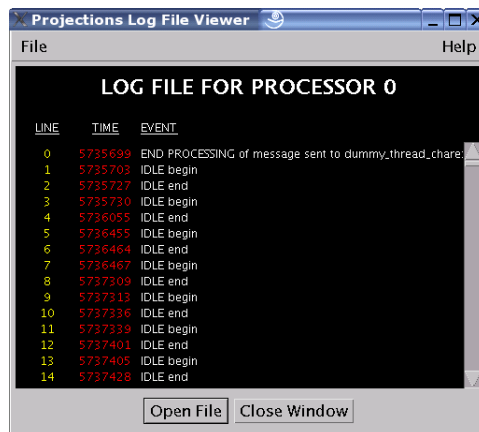


Figure 9: Log File View

- Size of Bin: Determine (in units - microseconds or bytes) how large each bin should be. A simple numeric field.
- Starting Bin Size: Determine (in units - microseconds or bytes) how far to offset the data. This is useful for ignoring data that is too small to be considered significant, but could overwhelm other data because of the sheer numbers of occurrences. A simple numeric field.

The dialog box reports the selection of bins as specified by the user by displaying the minimum bin size (in units - microseconds or bytes) to the maximum bin size. “units” refer to microseconds for time-based histograms or bytes for histograms representing message sizes.

Standard graph features can be employed for the main display of this view (see section 4.4). The following menu items are available in this tool:

- **File** offers 3 options: *Select Entry Points* currently does nothing. It is intended to behave similarly to the button “Select Entries” described below. This will be fixed in a later release of PROJECTIONS. *Set Range* reloads the dialog box and allows the user to load data based on new parameters. *Close* closes the current tool window.
- **View** provides 1 option: *Show Longest EPs* currently does nothing. It is intended to behave similarly to the button “Out-of-Range EPs” and will be fixed in a later release of PROJECTIONS.

The following options are available in the control panel in the form of toggle buttons:

- Entry method execution time (How long did that entry method ran for?)
- Entry method creation message size (How large was the message that caused the entry method’s execution?)

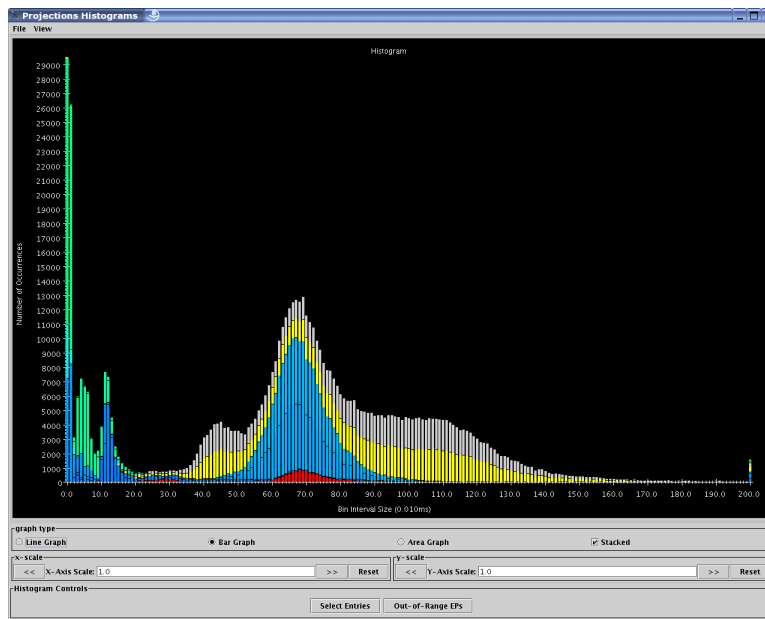


Figure 10: Histogram view

The use of the tool is somewhat counterintuitive. The dialog box is created immediately and when the tool window is created, it is defaulted to a time-based histogram. You may change this histogram to a message-size-based histogram by selecting the “Message Size” radio button which would then update the graph using the same parameters provided in the dialog box. This issue will be fixed in upcoming editions of PROJECTIONS.

The following features are, as of this writing, not implemented. They will be ready in a later release of PROJECTIONS.

The “Select Entries” button is intended to bring up a color selection and filtering window that allows you to filter away entry methods from the count. This offers more control over the analysis (e.g. when you already know EP 5 takes 20-30ms and you want to know if there are other entry points also takes 20-30ms).

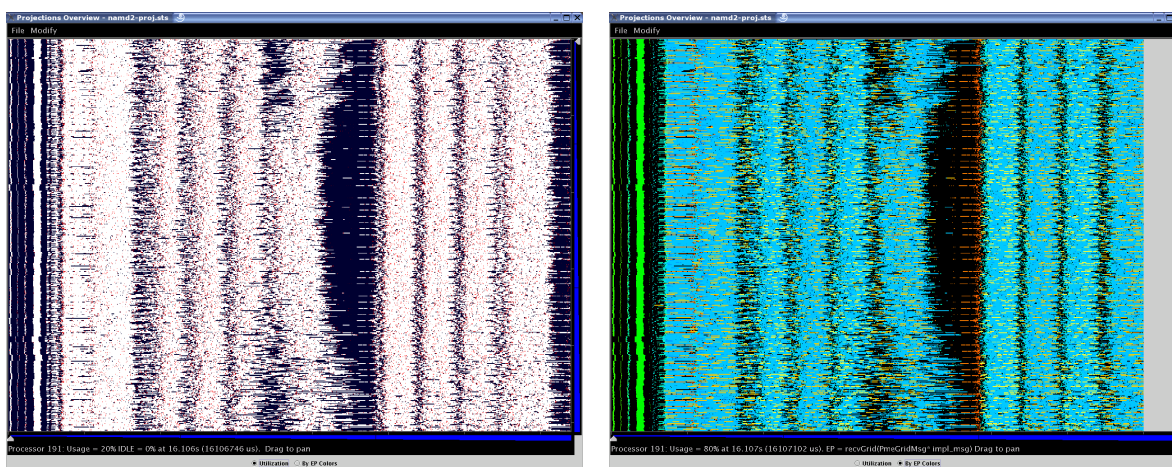
The “Out-of-Range EPs” button is intended to bring up a table detailing all the entry methods that fall into the overflow (last) bin. This list will, by default, be listed in descending order of time taken by the entry methods.

The performance of this view is affected by the number of bins the user wishes to analyze. We recommend the user limits the analysis to 1,000 bins or less.

4.3.8 Overview

Overview (see figure 11(a)) gives users an overview of the utilization of all processors during the execution over a user-specified time range.

The dialog box of the tool allows you to specify the time period within which to load overview information. This dialog box is exactly the same as that of the Timeline tool (see section 4.3.2).



(a) Overview

(b) Overview with dominant Entry Method colors

Figure 11: Different Overview presentation forms.

This tool provides support for the following menu options:

- **File** provides 1 option: *Close* closes the current tool.
- **Modify** provides 1 option: *Set Range* reloads the dialog box and allows the user to specify new parameters for rendering new overview information.

The view currently hard codes the number of intervals to 7,000 independent of the time-range desired.

Each processor has a row of colored bars in the display, different colors indicating different utilization at that time (White representing 100utilization (100representing 0a display of the processor usage of the specific processor at the specific time in the status bar below the graph. Vertical and horizontal zoom is enabled by two zooming bars to the right and lower of the graph. Panning is possible by clicking on any part of the display and dragging the mouse.

The “by EP colors” radio button provides more detail by replacing the utilization colors with the colors of the most significant entry method execution time in that time-interval on that processor represented by the cells (as illustrated in figure 11(b)).

The Overview tool uses memory proportional to the number of processors selected. If an out-of-memory error is encountered, try again by skipping processors (e.g. 0-8191:2 instead of 0-8191). This should show the general application structure almost as well as using the full processor range.

4.3.9 Animations

This window (see figure 12) animates the processor usage over a specified range of time and a specified interval size.

The dialog box to load animation information is exactly the same as that of the Graph tool (see section 4.3.1).

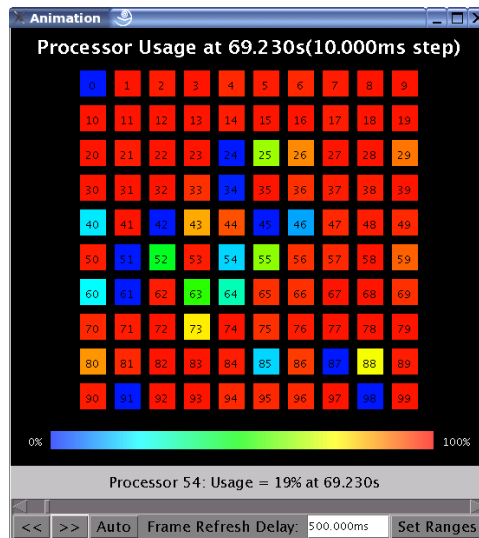


Figure 12: Animation View

A color temperature bar serves as a legend for displaying different processor utilizations as the animation progresses. Each time interval will have its data rendered as a frame. A frame displays in text on the top of the display the currently represented execution time of the application and what the size of an interval is.

Each selected processor is laid out in a 2-D plot as close to a square as possible. The view employs a color temperature ranging from blue (cool - low utilization) to bright red (hot - high utilization) to represent utilization.

You may manually update the frames by using the “<<” or “>>” buttons to visualize the preceding or next frames respectively. The “Auto” button toggles automatic animation given the desired refresh rate.

The “Frame Refresh Delay” field allows you to select the real time delay between frames. It is a time-based field (see section 4.4 for special features in using time-based fields).

The “Set Ranges” button allows you to set new parameters for this view via the dialog box.

This view has no known performance issues.

4.3.10 Time Profile Graph

The Time Profile view (see figure 13) is a visualization of the amount of time contributed by each entry method summed across all processors and displayed by user-adjustable time intervals.

Time Profile’s dialog box is exactly the same as that of the Graph tool (see section 4.3.1).

Standard graph features can be employed for the main display of this view (see section 4.4).

Under the tool options, one may:

- Filter the set of entry methods to be displayed on the graph via the “Select Entry Points” button. One may also modify the color set used for the entry methods via this option.
- use the “Select New Range” button to reload the dialog box for the tool and set new parameters for visualization (eg. different time range, different set of processors or different interval sizes).
- store the current set of entry method colors to disk (to the same directory where the trace logs are stored). This is done via the “Save Entry Colors” button.

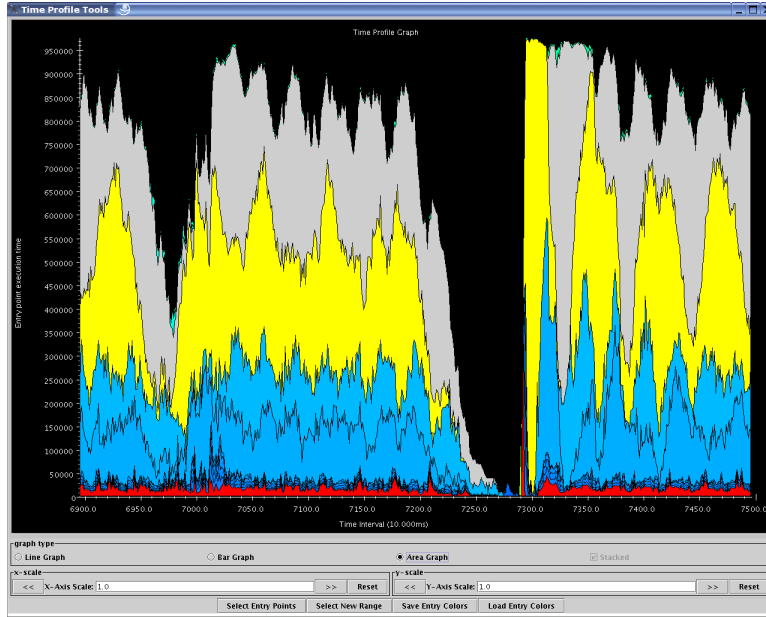


Figure 13: Time Profile Graph View

- load the stored set of entry method colors (if it exists) from disk (from the same directory where the trace logs are stored). This is done via the “Load Entry Colors” button.

Time Profile also reacts to the presence of data about MPI functions (See section 2.3.3). When such data is detected, an extra tabbed window displays a graph similar to entry method profiles, but for MPI functions only.

This tool’s performance is tied to the number of intervals desired by the user. We recommend that the user stick to visualizing 1,000 intervals or less.

4.3.11 User Events Profile

The User Events view is essentially a usage profile (See section 4.3.3) of bracketed user events (if any) that were recorded over a specified time range. The x-axis holds bars of data associated with each processor while the y-axis represents the time spent by each user event. Each user event is assigned a color.

It is important to note that user-events can be arbitrarily nested. The view currently displays information based on raw data without regard to the way the events are nested. Memory usage is proportional to the number of processors to be displayed.

4.3.12 Outlier Analysis

For performance logs generated from large numbers of processors, it is often difficult to view in detail the behavior of poorly behaved processors. This view attempts to present information similar to usage profile but only for processors whose behavior is “extreme”.

“Extreme” processors are identified through the application of heuristics specific to the attribute that analysts wish to study applied to a specific activity type. You can specify the number of “extreme” processors are to be picked out by PROJECTIONS by filling the appropriate number in the field “Outlier Threshold”. The default is to pick 10% of the total number of processors up to a cap of 20. As an example, an analyst may wish to find “extreme” processors with respect to the idle time of normal CHARM++ trace events.

Figure 15 shows the choices available to this tool. Specific to this view are two pull-down menus: *Attribute* and *Activity*.

There are four *Activity* options:

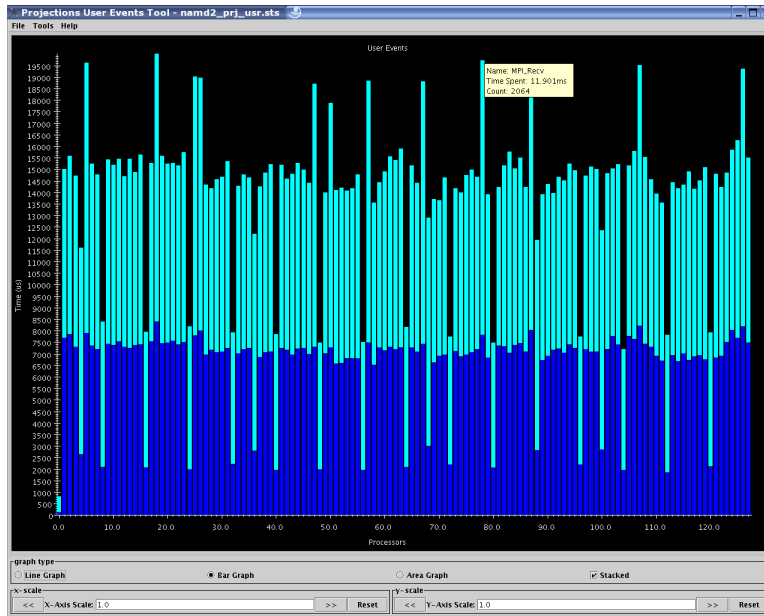


Figure 14: User Events Profile View

1. The *Projections* activity type refer to the entry methods executed by the CHARM++ runtime system.
2. The *User Events* activity type refer to records of events as captured through `traceUserEvent`-type calls described in section 2.3.2.
3. The *Functions* activity type refer to the events captured for AMPI functions through the functions described in section 2.3.3.

There are four *Attribute* options:

1. *Execution time by Activity* tells the tool to apply heuristics based on the execution time of each instance of an activity occurring within the specified time range.
2. *Idle time* tells the tool to apply a simple sort over all processors on the least total idle time recorded. This will work only for the *Projections* activity type.
3. *Msgs sent by Activity* tells the tool to apply heuristics based on the number of messages sent over each instance of an activity occurring within the specified time range. This option is currently not implemented but is expected to work over all activity types.
4. *Bytes sent by Activity* tells the tool to apply heuristics based on the size (in bytes) of messages sent over each instance of an activity occurring within the specified time range. This option is currently not implemented but is expected to work over all activity types.

At the same time, a k-means clustering algorithm is applied to the data to help identify processors with exemplar behavior that is representative of each cluster (or equivalence class) identified by the algorithm. You can control the value of k by filling in the appropriate number in the field “Number of Clusters”. The default value is 5.

The result of applying the required heuristics to the appropriate *attribute* and *activity* types results in a chart similar to figure 16. This is essentially a usage profile that shows, over the user’s selected time range, from left to right:

- A bar representing the global average of execution time of each activity over all processors.

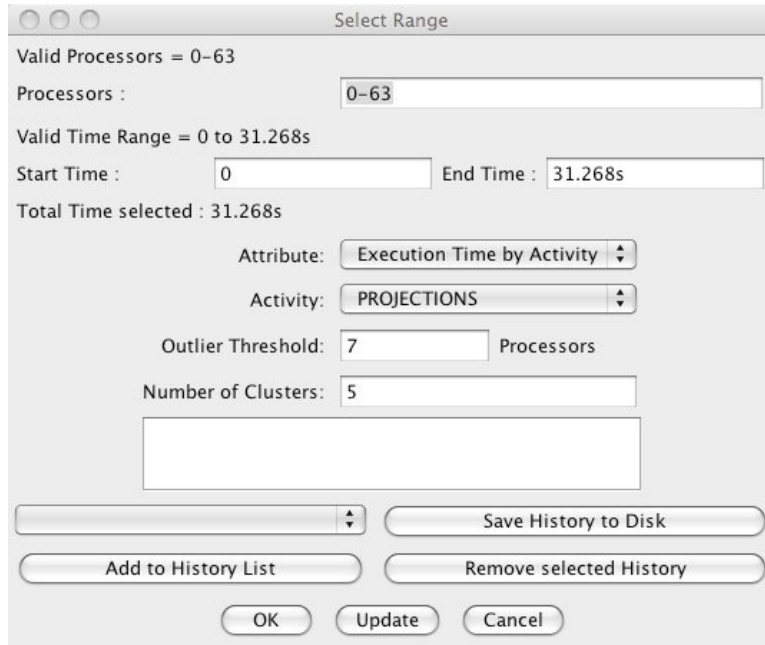


Figure 15: Outlier Analysis Selection Dialog

- A bar representing the average activity profile of all non-outlier (or non-extreme) processors.
- A bar representing the average activity profile of all outlier (or extreme) processors identified by the heuristics.
- One bar representing the activity profile of the representative processor from each cluster of processors identified by the application of the k-means clustering algorithm.
- One bar representing the activity profile of each identified outlier processor, sorted in order of significance (rightmost processor bar is the most significant).

The tool helps the user reduce the number of processor bars that must be visually examined in order to identify candidates for more detailed study. To further the cause of this goal, if the analyst has the *timeline* view (see section 4.3.2) open, a mouse-click on any of the processor activity profile bars (except for group-averaged bars) will load that processor's detailed timeline (over the time range specified in the timeline view) into the timeline view itself.

4.3.13 Multirun Analysis

4.3.14 Function Tool

The Function Tool view presents a graph that is a usage profile (See section 4.3.3) of AMPI function information. This view allows the analyst to choose to display the time spent by each function or the number of calls made over the selected time range.

In the case of AMPI functions, the events are properly nested. The information displayed is currently that of *inclusive time* (i.e. if function B's calls are nested within function A's, the time spent in function B contribute to both function B's and function A's displayed performance information). There are plans to implement the presentation of AMPI function information based on *exclusive time* (i.e. time within functions are computed by subtracting the measured time spent minus the time spent by any calls to nested functions).

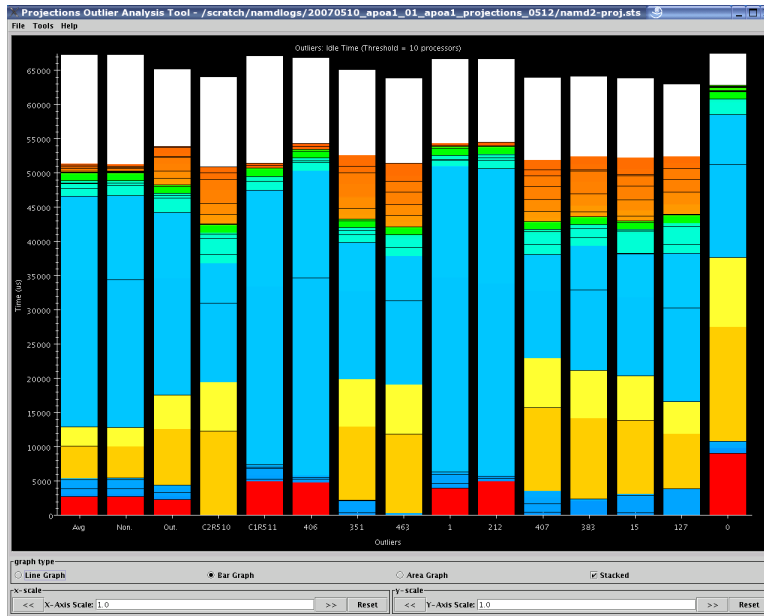


Figure 16: Outlier Analysis View

4.3.15 AMPI Usage Profile

The AMPI Usage Profile view presents a graph similar to Function Tool's (See section 4.3.14) with several modifications:

1. In it's per-processor mode, displayed via the tabbed window "Per Processor", the information displayed includes the time spent by other events outside of AMPI. This is displayed as a white bar marked "Others" when moused-over. This allows the analyst to compare the time spent by events within AMPI functions along with other recorded events. In contrast, Function Tool shows only AMPI function events.
2. In it's per-function mode, displayed via the tabbed window "Per Function", the information is displayed with each bar on the x-axis showing the percentage utilization for a different AMPI function.

4.3.16 NoiseMiner View

Projections Computational Noise Miner : /expand/home/idooley2/NoiseMi					
Results		Text Summary			
Noise Duration	Seen on Processors	Occurrences	Periodicity	Likely Source of Noise	Exemplar Timelines
5.70(ms)	0	1425	21.34(ms)	internal	view
610.75(us)	0, 1	3268	2.17(ms)	internal	view
487.23(us)	0, 1	5144	1.83(ms)	internal	view

[Select New Range](#)

Figure 17: NoiseMiner View showing a 5.7 ms noise component that occurred 1425 times during a run. In this case, MPI calls to a faulty MPI implementation took an extra 5.7 ms to return.

The NoiseMiner view (see figure 17 and 18) displays statistics about abnormally long entry methods. Its purpose is to detect symptoms consistent with *Operating System Interference* or *Computational Noise* or

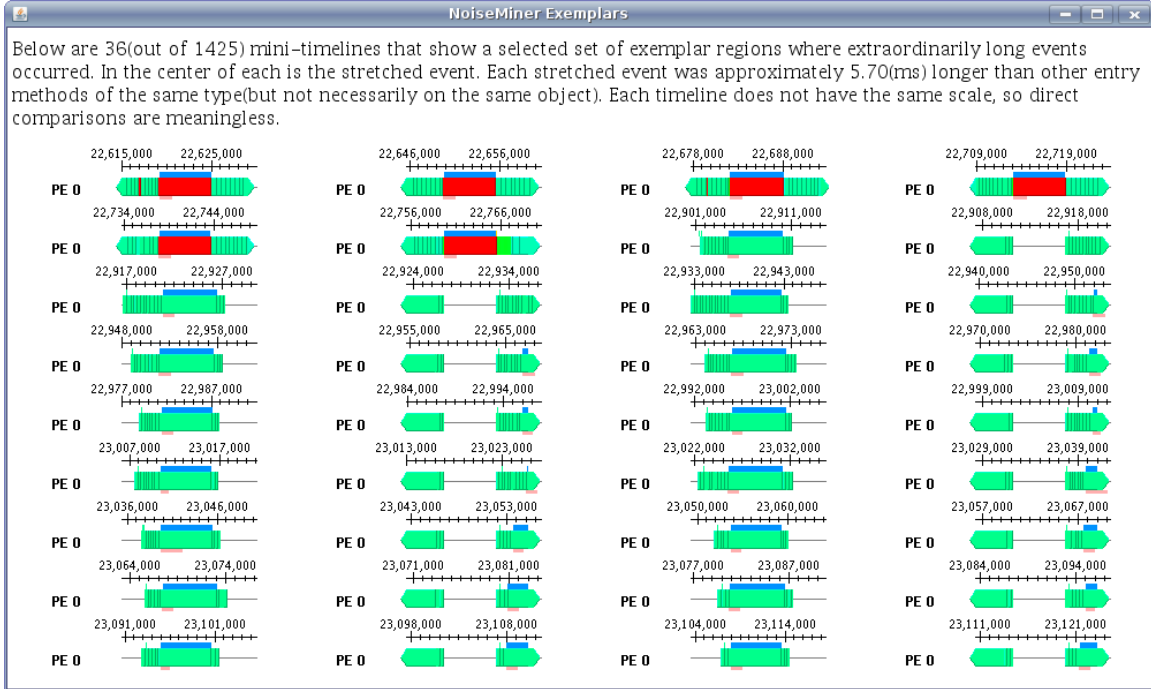


Figure 18: NoiseMiner noise component view showing miniature timelines for one of the noise components.

Software Interference. The abnormally long events are filtered and clustered across multiple dimensions to produce a concise summary. The view displays both the duration of the events as well as the rate at which they occur. The initial dialog box allows a selection of processors and a time range. The user should select a time range that ignores any startup phase where events have chaotic durations. The tool makes only a single pass through the log files using a small bounded amount of memory, so the user should select as large time range as possible.

The tool uses stream mining techniques to produce its results by making only one pass through the input data while using a limited amount of memory. This allows NoiseMiner to be very fast and scalable.

The initial result window shows a list of zero or more noise components. Each noise component is a cluster of events whose durations are abnormally long. The noise duration for each event is computed by comparing the actual duration of the event with an expected duration of the event. Each noise component contains events of different types across one or more processors, but all the events within the noise component have similar noise durations.

Clicking on the “view” button for a noise component opens a window similar to figure 18. This second window displays up to 36 miniature timelines, each for a different event associated with the noise component.

NoiseMiner works by storing histograms of each entry method’s duration. The histogram bins contain a window of recent occurrences as well as an average duration and count. After data stream has been parsed into the histogram bins, the histogram bins are clustered to determine the expected entry method duration. The histograms are then normalized by the expected duration so that they represent the abnormally stretched amounts for the entry methods. Then the histogram bins are clustered by duration and across processors. Any clusters that do not contribute much to the overall runtime are dropped.

4.4 Miscellaneous features

4.4.1 Standard Graph Display Interface

A standard graph display (an example of which can be found with the Main Summary Graph - figure 1) has the following features:

- **Graph types** can be selected between “Line Graph” which connects each data point with a colored line representing the appropriate data entry. This information may be “stacked” or “unstacked” (controlled by the checkbox to the right). A “stacked” graph places one data point set (Y values) on top of another. An “unstacked” graph simply uses the data point’s Y value to directly determine the point’s position; “Bar Graph” (the default) which draws a colored bar for each data entry and the value of the data point determines its height or starting position (depending on whether the bar graph is “stacked” or “unstacked”). A “Bar Graph” displayed in “unstacked” mode draws its bars in a tallest to shortest order so that the large Y values do not cover over the small Y values; “Area Graph” is similar to a “Line Graph” except that the area under the lines for a particular Y data point set is also colored by the data’s appropriate color. “Area Graph”s are always stacked.
- **x-scale** allows the user to scale the X-Axis. This can be done by directly entering a scaling factor in the text field (simple numeric field - see below) or by using the “<<” or “>>” buttons to increase or decrease the scale by 0.25 each time. The “Reset” button changes the scale factor back to 1.0. A scrollbar automatically appears if the scale factor causes the canvas to be larger than the window.
- **y-scale** allows the user to scale the Y-Axis. This functions similarly to the **x-scale** feature where the buttons and fields are concerned.

4.4.2 Standard Dialog Features

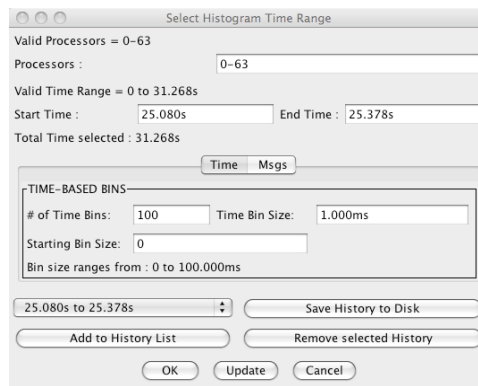


Figure 19: An example Dialog with standard fields

Figure 19 shows a sample dialog box with standard features. The following are standard features that can be employed in such a dialog box:

- **Moving from field to field** via the tab key causes the dialog box update the last field input by the user. It also performs a consistency check. Whenever it finds an inconsistency, it will move mouse focus onto the offending field, disabling the “OK” button so as to force the user to fix the inconsistency. Examples of inconsistency includes: input that violates a field’s format; input whose value violates constraints (eg. start time larger than end time); or out-of-range stand-alone values.
- **Available buttons** include “OK” which confirms the user’s choice of parameters. This button is only activated if the dialog box considers the parameters’ input to be consistent. “Update” causes the dialog box to update the last field input by the user and perform a consistency check. This is similar in behavior to the user tabbing between fields. “Cancel” closes the dialog box without modifying any parameters if the tool has already been loaded or aborts the tool’s load attempt otherwise.
- **Parameter History** allows the user to quickly access information for all tools for a set of frequently needed time periods. An example of such a use is the desire by the analyst to view a particular phase or timestep of a computation without having to memorize or write on a piece of paper when exactly the phase or timestep occurred.

It consists of a pull-down text box and 2 buttons. “Add to History List” adds the current time range to the pull-down list to the left of the button. The dialog box maintains up to 5 entries, replacing older entries with newer ones. “Remove Selected History” removes the currently selected entry in the history list. “Save History to Disk” stores current history information to the file “ranges.hst” in the same directory where your logs are stored. Note that you will need write access to that directory to successfully store history information. A more flexible scheme is currently being developed and will be released in a later version of PROJECTIONS. Clicking on the pull-down list allows the user to select one out of up to 5 possible time ranges. You can do so by moving the mouse up or down the list. Clicking on any one item changes the start and end times on the dialog box.

4.4.3 Data Fields

Throughout PROJECTIONS tools and dialog boxes (see sample figure 19), data entry fields are provided. Unless otherwise specified, these can be of the following standard field with some format requirements:

- **Simple numeric fields:** An example can be found in figure 19 for “Number of Bins:”. This field expects a single number.
- **Time-Based Field:** An example can be found in figure 19 for “Start Time:”. This field expects a single simple or floating point number followed by a time-scale modifier. The following modifiers are supported: *none* - this is the default and means the input number represents time in microseconds. A whole number is expected; *The characters “us”* - the input number represents time in microseconds. A whole number is expected; *The characters “ms”* - the input number represents time in milliseconds. This can be a whole number or floating point number; or *The character “s”* - the input number represents time in seconds. This can be a whole number or floating point number.
- **Processor-Based Field:** An example can be found in figure 19 for “Processors:”. This field expects a single whole number; a list of whole numbers; a range; or a mixed list of whole numbers and ranges. Here are some examples which makes the format clearer:

eg: Want to see processors 1,3,5,7: Enter 1,3,5,7

eg: Want to see processors 1,2,3,4: Enter 1-4

eg: Want to see processors 1,2,3,7: Enter 1-3,7

eg: Want to see processors 1,3,4,5,7,8: Enter 1,3-5,7-8

Ranges also allow skip-factors. Here are some examples:

eg: Want to see processors 3,6,9,12,15: Enter 3-15:3

eg: Want to see processors 1,3,6,9,11,14: Enter 1,3-9:3,11,14

This feature is extremely flexible. It will normalize your input to a canonical form, tolerating duplication of entries as well as out-of-order entries (ie. 4,6,3 is the same as 3-4,6).

5 Known Issues

This section lists known issues and bugs with the PROJECTIONS framework that we have not resolved at this time.

- CHARM++ scheduler idle time is known to be incorrectly recorded on the BG/L machine at IBM TJ Watson.
- End-of-Run analysis techniques (while tracing applications) are currently known to hang for applications that make multiple calls to `traceBegin()` and `traceEnd()` on the same processor through multiple CHARM++ objects.