Projections – A Performance Tool for Charm++ Applications

Chee Wai Lee
PPL, UIUC

Tutorial Outline

- General Introduction
- Instrumentation
- Trace Generation
- Performance Analysis using Projections
- Using Projections effectively

General Introduction

- Introduction to Projections
- The Basic Charm++ Model
- NAMD as a case study application

Projections

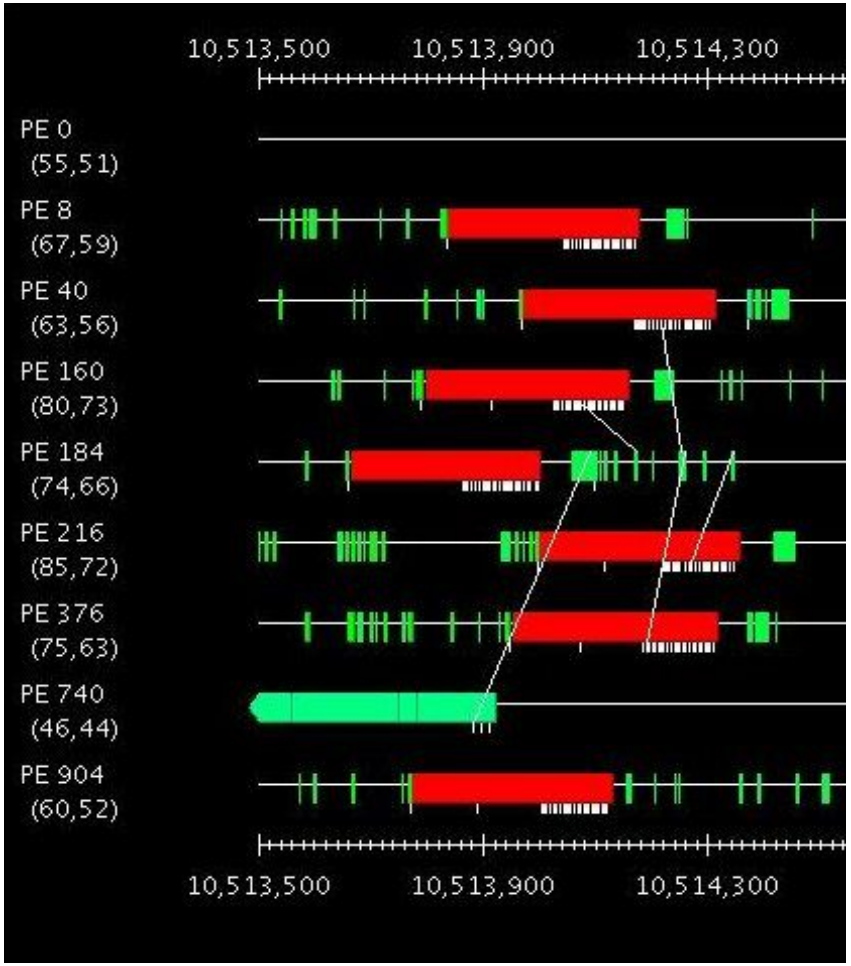
- Projections is a performance tool designed for use with Charm++.
- Trace-based, post-mortem human-centric analysis.
- Supports highly detailed traces, summary formats and a flexible user-level API.
- Java-based visualization tool for presenting performance information

What you will need

- A version of Charm++ built *without* the *CMK_OPTIMIZE* flag. (Note: This tends to be the default build)
- Java Runtime 1.3.1 or higher
- Projections Visualization binary.
  - Distributed with the Charm++ source. (in *tools/projections*)
  - Acquire a standalone Java archive (*projections.jar*)

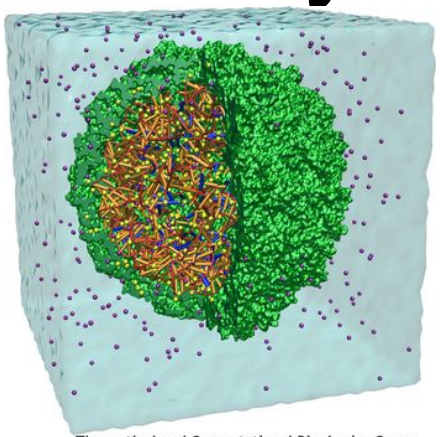
The Basic Charm++ Model

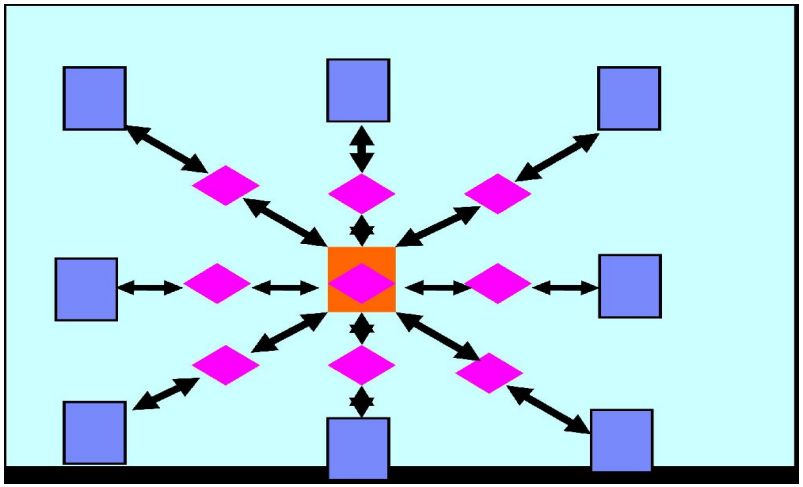
- Object-Oriented. ***Chares*** encapsulate data and ***entry methods***.
- Message-driven. An ***entry method*** is scheduled for execution on a processor when a ***message*** arrives.



NAMD as a case study

- Spatial decomposition into Patches (blue).
- Force decomposition into "Computes" (magenta diamonds)
- Different force types (bonded, non-bonded, ...)
- Long-range electrostatics computed using Particle Mesh Ewald (PME) method.





Instrumentation

- Basics
- Application Programmer Interface (API)
  - User specific events
  - Turning tracing on/off

Instrumentation: Basics

- Nothing to do!
- The Charm++ runtime automatically instruments entry method execution and communication events. (as described in the Basic Charm++ Model)
- In a majority of cases, this generates very useful data for analysis yet introduces minimal overhead/perturbation.

Instrumentation: User Events

- If user-defined events are required, these can be inserted into application code:

Register:

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

Track a Point-Event:

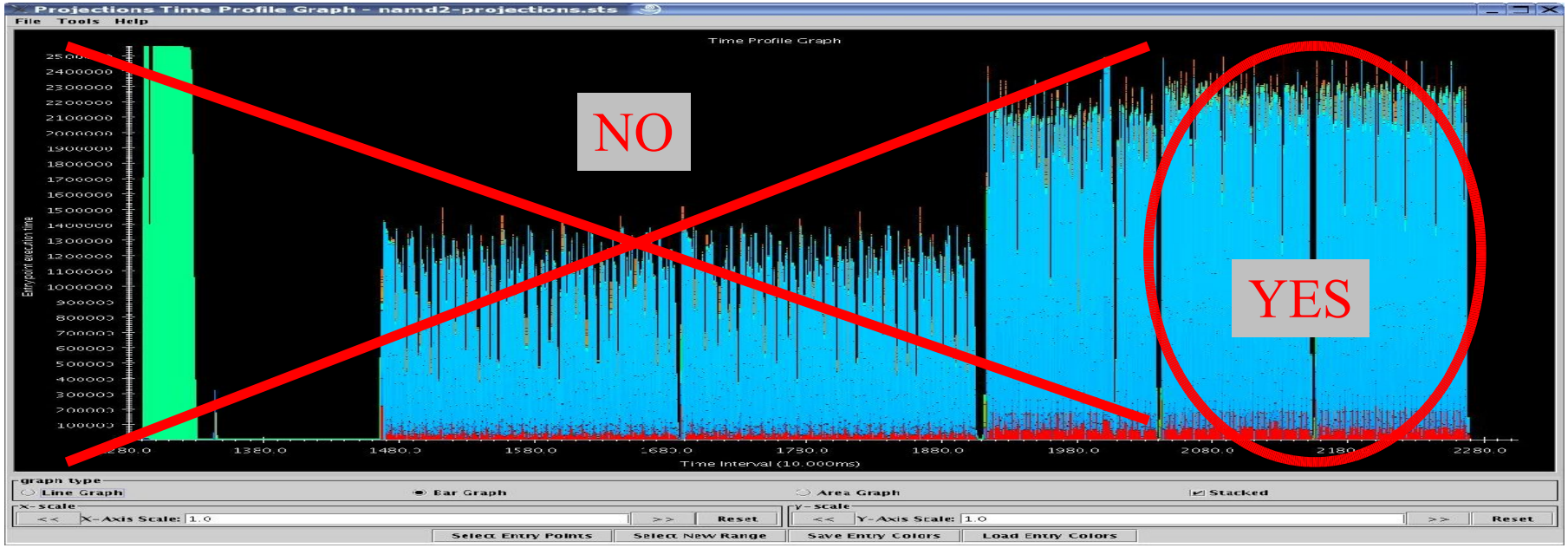
void traceUserEvent(int EventNum)

Track a Bracketed-Event:

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

Instrumentation: Selective Tracing (1)

- API also exists for selective tracing of application code. Why do we want this?



Selective Tracing (2)

- Simple Interface, not so easy to use:

```
void traceBegin()

void traceEnd()
```

- Calls have per-processor effects, but is called from entry methods encapsulated in objects.
- Best place to make these calls are at/around synchronization points.

Selective Tracing Example

```
  // in the case when trace is off at the beginning,
  // only turn trace of from after the first LB to the firstLdbStep after
  // the second LB.
  // 1    2   3     4     5          6   7
  // off on Alg7 refine refine ...  on
#if CHARM_VERSION >= 050606
  if (traceAvailable()) {
    static int specialTracing = 0;
    if (ldbCycleNum == 1 && traceIsOn() == 0)  specialTracing = 1;
    if (specialTracing) {
      if (ldbCycleNum == 4) traceBegin();
      if (ldbCycleNum == 6) traceEnd();
    }
  }
#endif
```

Trace Generation

- Build time options (tracing modules)
  - *summary* (aggregated data)
  - *trace logs* (event traces)
- Runtime options
  - summary resolution control
  - buffer control
  - output control
  - tracing control

Build Options

- Link into application one or more modules for tracing at various levels of detail:
  - "-tracemode summary" for aggregated data.
  - "-tracemode projections" for event traces.

Runtime Options (1)

- General options:
  - *+traceoff* tells the tracing framework not to record events until it encounters a traceBegin() API call.
  - *+traceroot <dir>* tells the tracing framework which folder to write output to.
  - +gz-trace tells the tracing framework to output compressed data (default is text). This is useful on extremely large machine configurations where the attempt to write the logs for *p* processors would overwhelm the IO subsystem.

Summary runtime options

- +sumdetail – tells the framework to aggregate data by entry method as well as time-intervals. (normal summary data is aggregated only by time-interval)
- +numbins <*k*> – tells the framework to reserve enough memory to hold information for <*k*> time-intervals. (default is 10,000 bins)
- +binsize <*duration*> - tells the framework to aggregate data such that each time-interval represents <*duration*> *seconds* of execution time. (default is 1ms)

Event Trace Options

- +logsize <*k*> – tells the framework to reserve enough buffer memory to hold <*k*> events. (default is 1,000,000 events)

Memory Usage

- What happens when we run out of reserved memory?
  - summary: doubles time-interval represented by each bin, aggregates data into the first half and continues.
  - event traces: asynchronously flushes event log to disk and continues.

Memory Usage (2)

- Trade-offs to consider:
  - Summary
    - Frequency of data compaction.
    - Performance data resolution.
    - Memory usage (usually not a problem).
  - Event Traces
    - Frequency of data flushing (this usually perturbs the application badly).
    - Memory usage.

Trace Generation: How to?

- Run your application normally (batch or interactive).
- When run ends, you will find any number of “.log”, “.sum”, “.sts” files generated in the directory specified with *+traceroot* or where your binary is located.

Visualization: Basic Steps

- Run the script:

  charm/tools/projections/bin/projections

- You may choose to supply the location for a ".sts" file as an argument to tell the tool to load performance data associated with it.

Visualization/Analysis Reference Guide

Analysis Techniques

- Zoom in/out to find potential problem spots.
- Loading sufficient but not overwhelming data.
- Using effective Colors.
- Make use of the history feature in dialog boxes to keep track of time-ranges explored.

Visualization Limitations

- Timeline
  - workable time ranges depend heavily on the event-density of the selected range. (typical ranges are 10ms-10s worth of time for 10-20 processors)
- Interval-based views (eg. Time Profile)
  - keep to 2000 or fewer intervals.

Look Closely!

Do not be fooled! Projections does not handle some visualization artifacts well:

- Fine grain details can sometimes look like one big solid block on timeline.
- It is hard to mouse-over items that represent fine-grained events.
- Other times, tiny slivers of activity become too small to be drawn.

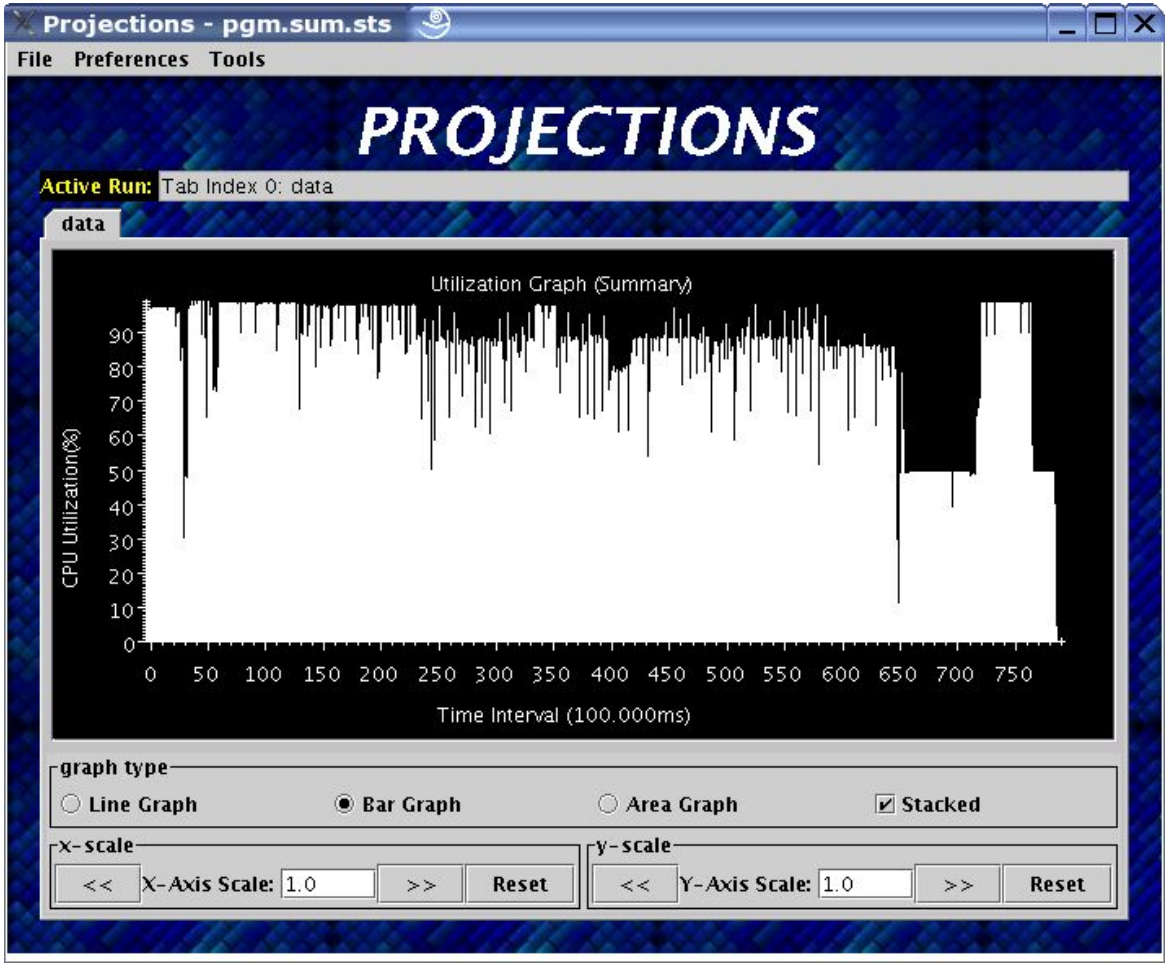
Sample Screenshots

Starting Screen/Summary

Starting screen.

Summary graph only shows up if ".sum" files are available.

Shows average CPU utilization across all processors as a function of time.



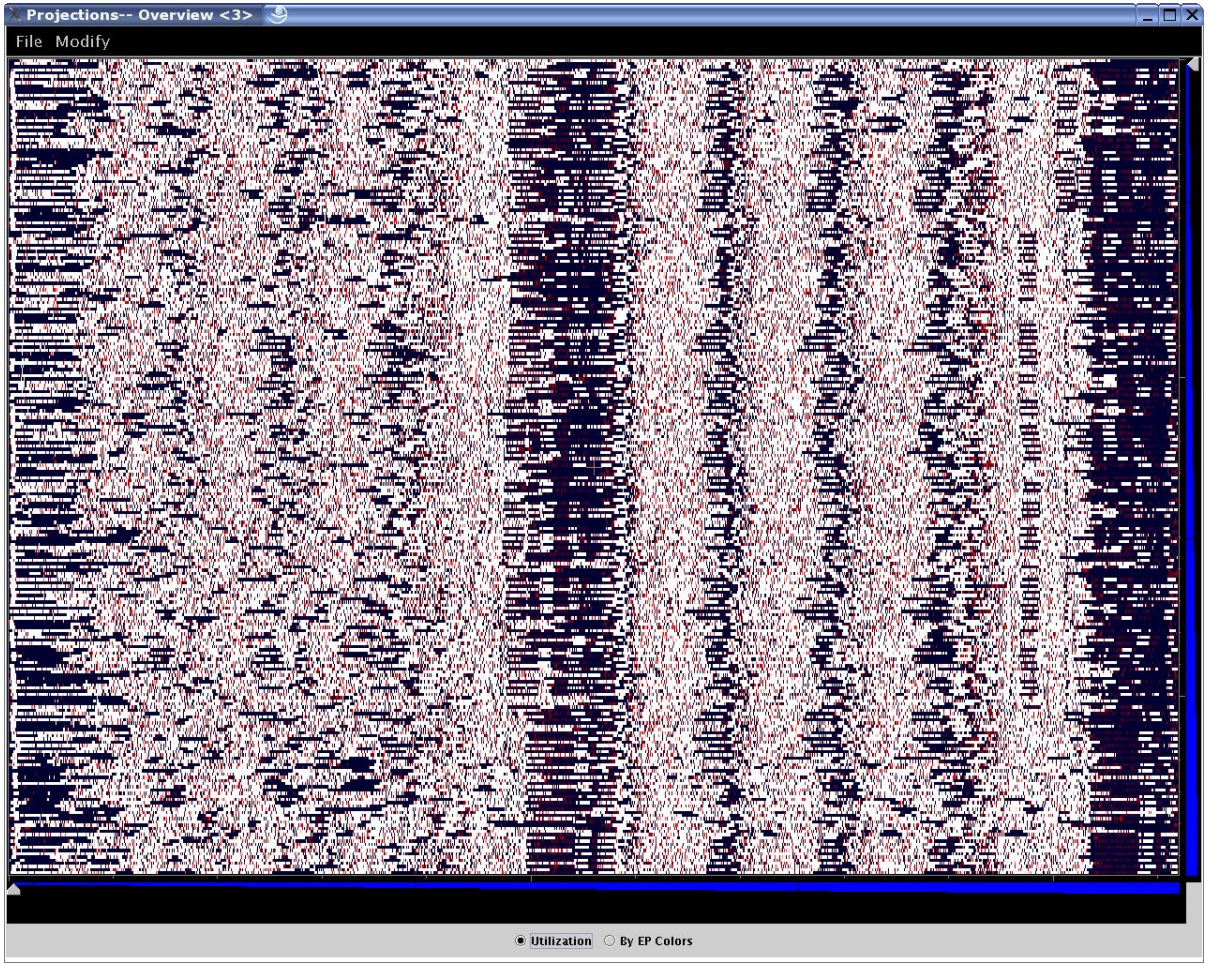
Overview

Time progresses from left to right.

Each row represents a selected processor.

Color intensity shows processor utilization. Black = 0%, shades of Red 1-99%, White = 100%.

Mouse-over support for details.

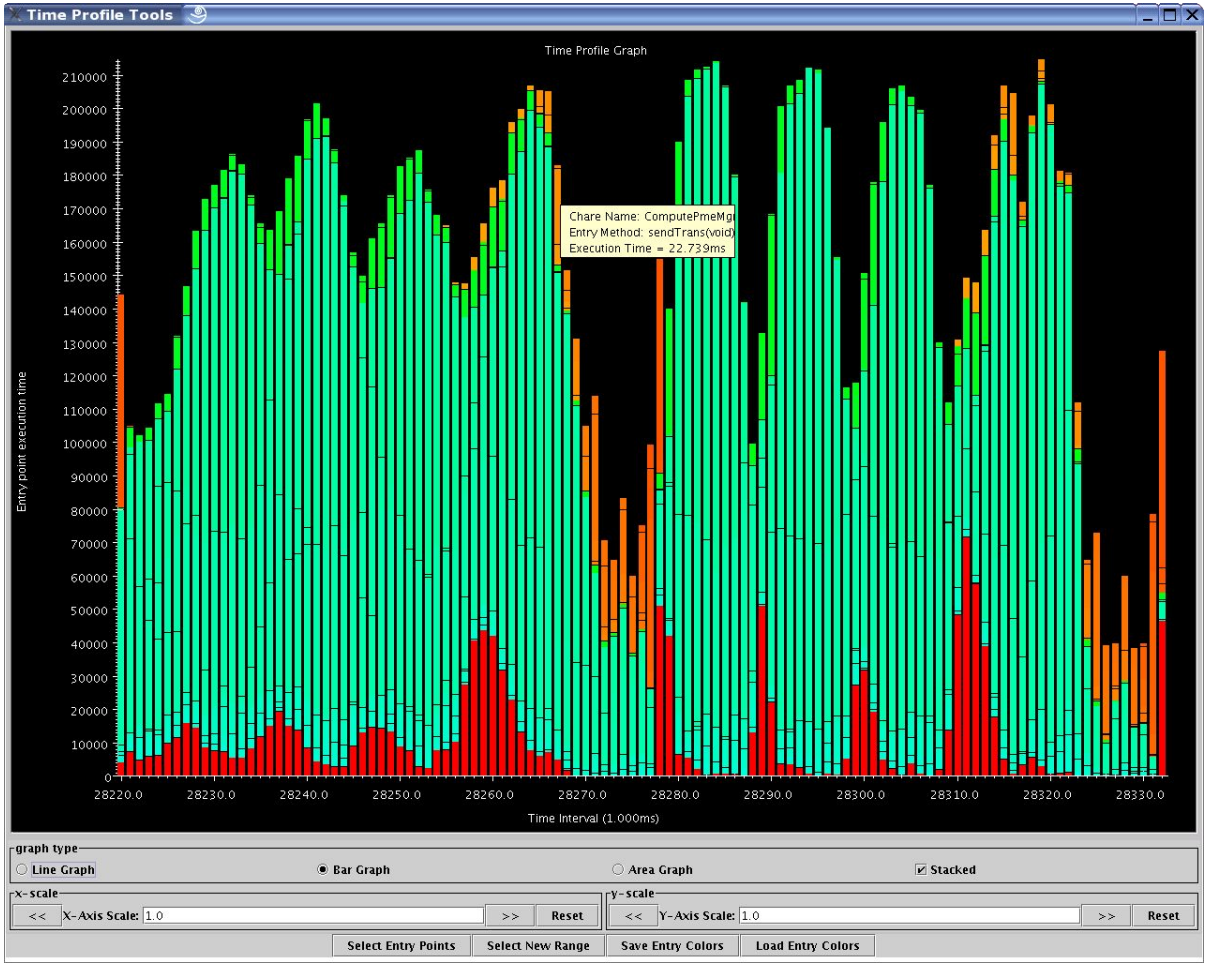


Time Profile

Time progresses from left to right.

Each bar shows the amount of time spent by each colored EP summed across all processors during the associated time-interval.

Mouse-over support for more details.

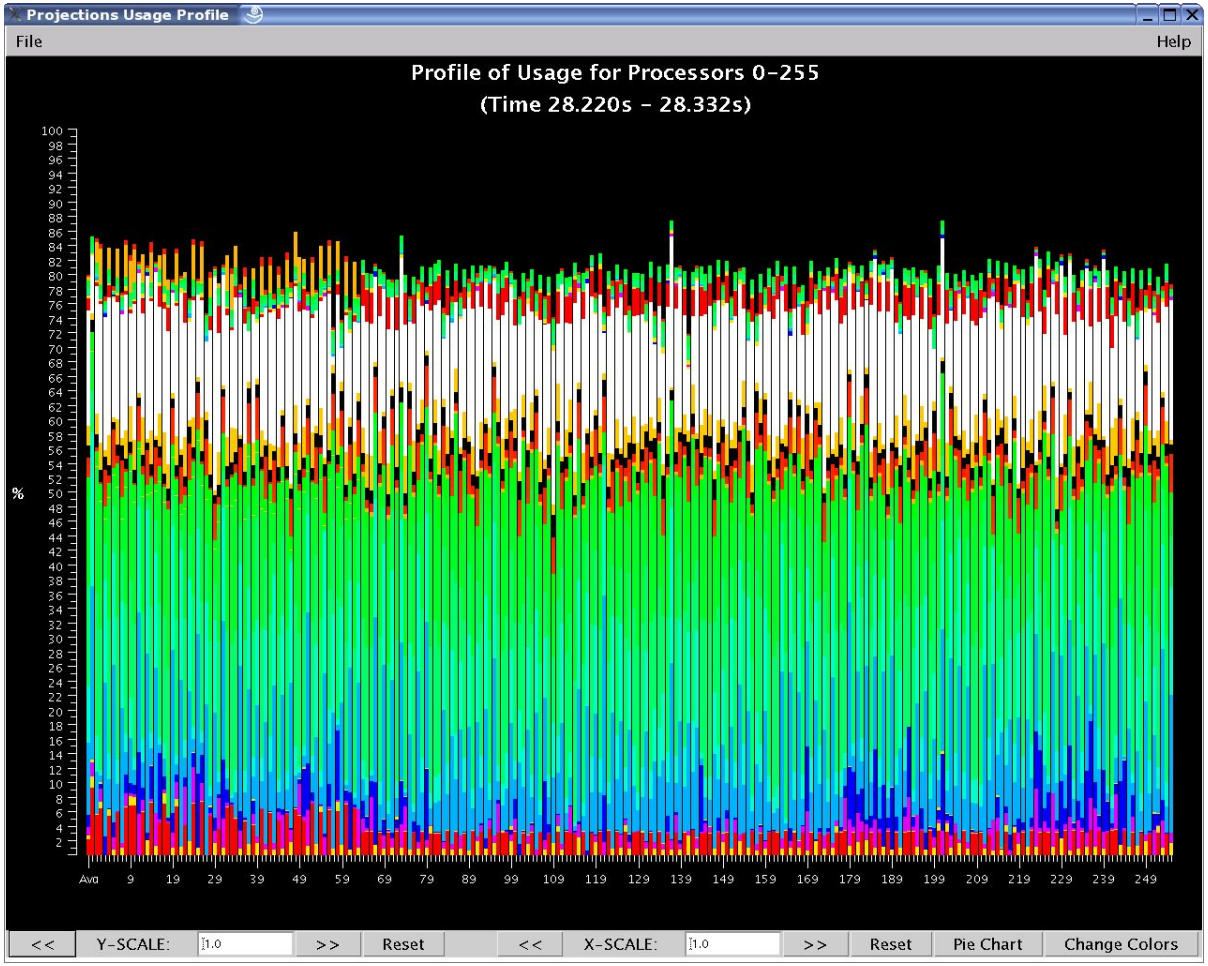


Usage Profile

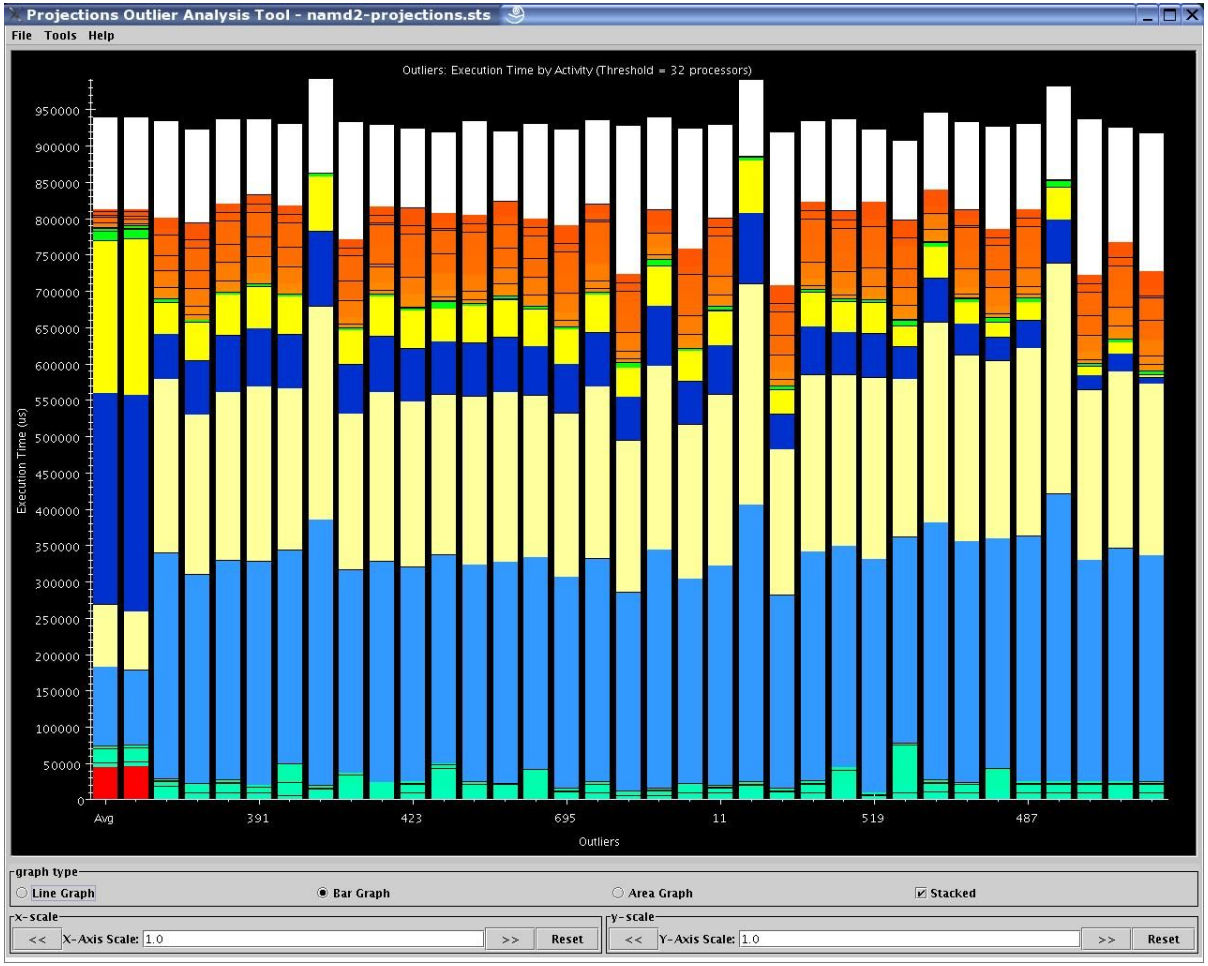
Shows % CPU utilization by colored Eps over selected time-range.

Each bar represents a selected processor. Leftmost bar shows the average processor profile.

White represents scheduler idle time.



Outlier/Extrema Discovery (1)



Outlier/Extrema Discovery (2)

- The leftmost bar shows average processor profile.
- The 2nd bar shows the non-outlier averge processor profile.
- The 3rd bar shows the outlier average processor profile.
- Subsequent bars show outlier processors ranked from least (left) to most (right) significant.

Outlier/Extrema Discovery (3)

- Clicking on a processor bar loads the processor onto an existing Timeline view using the specified time-range.
- Mouse-overs provide more information about the time spent by each EP.

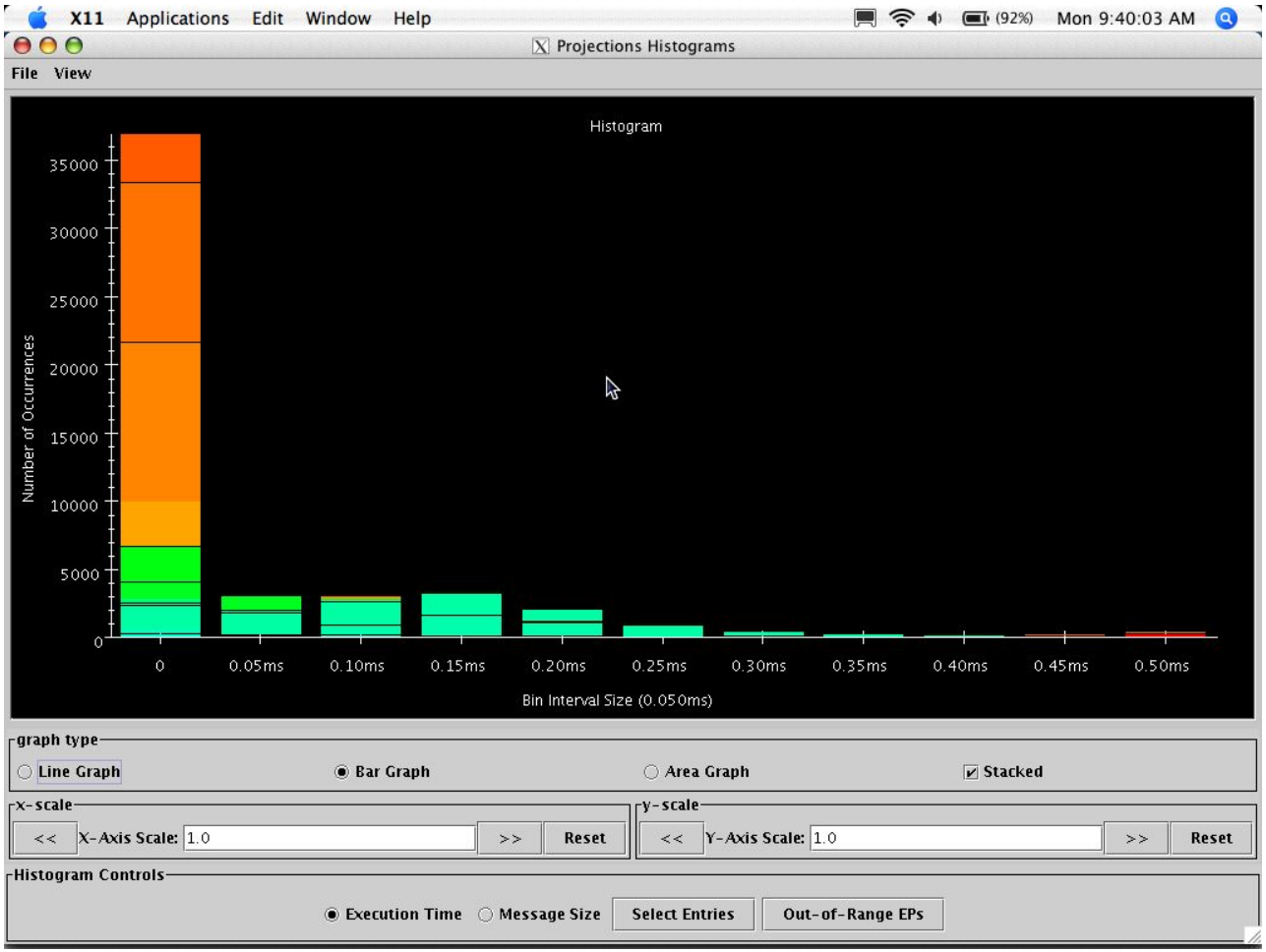
Histograms

Time histograms show EP grainsize distributions.

Each bar represents a bin for EPs that executed for a certain duration.

Values are freq counts.

Mouse-over supported.



Timeline (1)


Projections Timeline
File Tools Colors
Help
75,000,000
75,500,000
76,000,000
76,500,000
77,000,000
77,500,000
78,000,000
78,500,000
79,000,000
79,500,000
80,000,000
PE 0 (33.5%)
PE 10 (59.7%)
PE 20 (57.4%)
PE 30 (33.6%)
PE 40 (32.6%)
PE 50 (49.2%)
PE 60 (33.5%)
PE 70 (52.8%)
PE 80 (34.5%)
PE 90 (38.6%)
Display Pack Times
Display Message Sends
Display Idle Time
View User Events (2121)
Select Ranges
Change Colors
<<
SCALE:
1.0
>>
Reset
Highlight Time
Slection Begin Time
Selection End Time
Selection Length
Zoom Selected
Load Selected
Jump to graph:
Usage Profile
Graph
Histogram
Communication
Overview

Timeline (2)

- Each row represents a processor.
- Shows in detail each colored event.
- White ticks below an event block represents out-going communication.
- Colored ticks above an event block visualizes user-events.
- Scheduler Idle time represented by white “event” blocks.

Timeline (3)

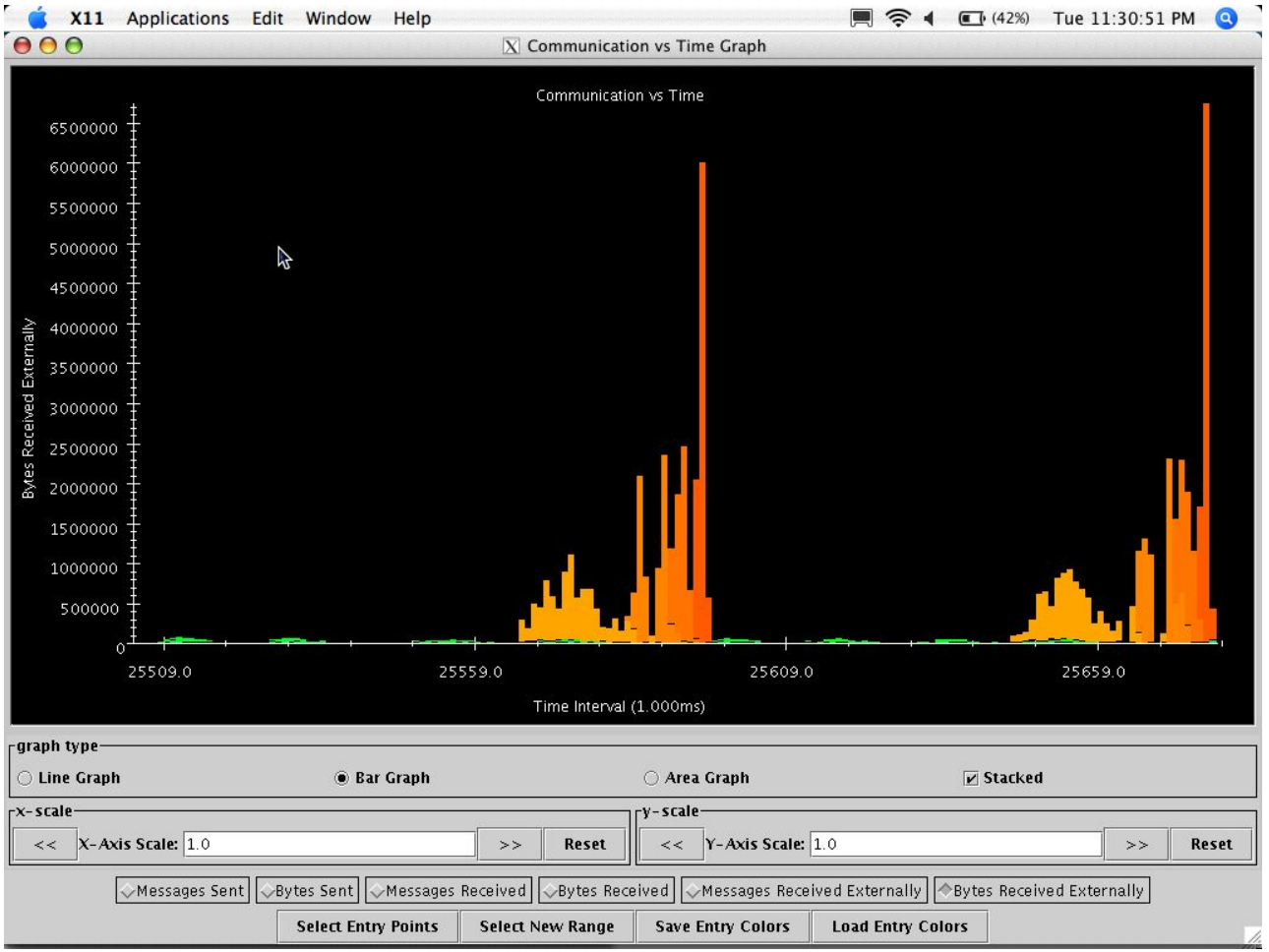
- Lines showing the source of an incoming message may be drawn by right-clicking on an event block.
- Left-clicking on an event block shows all out-going message information for that event block.
- Mouse-over event blocks to acquire highly detailed information about each event block.

Communication Tools

Time progresses from left to right.

Each bar shows some communication attribute associated to a colored EP.

Mouse-over support for more detail.



Dialog Boxes

General Dialog Features:

Select Processors.
(e.g. *"3-7,15,100-1000:5" will load data for processors 3,4,5,6,7,15,100,105,...,995,1000*)

Select time-ranges.

Time-range history feature.