About this Document

This document describes how to install and use the Visual Performance Analyzer tool. This document will help you install the tool, learn how to collect performance data on your platform and later analyze the data, using the VPA plug-ins.
# Table of Contents

1. INTRODUCTION

   1.1 VPA on Alphaworks ................................................................. 5
   1.2 Release History ........................................................................ 6

2. VPA BASICS

   2.1 Design Objectives ................................................................. 7
   2.2 Deployment ............................................................................ 8
   2.3 Software Stack Information ................................................... 9

3. INSTALLATION

   3.1 Windows............................................................................. 11
      3.1.1 Download from Alphaworks ............................................ 11
      3.1.2 Extract the compressed file ............................................. 11
      3.1.3 Create a Shortcut ........................................................... 15
   3.2 Linux.................................................................................. 18
      3.2.1 Download from Alphaworks ............................................ 18
      3.2.2 Extract the compressed file ............................................. 18
   3.3 AIX.................................................................................... 19
      3.3.1 Download from Alphaworks ............................................ 19
      3.3.2 Extract the compressed file ............................................. 19

4. COLLECTING PERFORMANCE DATA

   4.1 Using Platform Tools ............................................................ 20
   4.2 Setting up Windows to collect Profiling data ............................ 21
      4.2.1 Verify that your Java Runtime is installed on your system .... 21
      4.2.2 Verify that the Windows performance tools are installed ....... 22
      4.2.3 Verify PI Tprof ............................................................... 22
      4.2.4 Copying data files ......................................................... 22
   4.3 Setup up AIX to collect Profiling data ..................................... 23
      4.3.1 Verify that your Java Runtime is installed on your system .... 23
      4.3.2 Verify that the AIX performance tools are installed .......... 23
      4.3.3 Verify AIX Tprof ........................................................... 23
      4.3.4 Copying data files ......................................................... 23
   4.4 Collecting Profiling Data on Linux platform ............................... 23
      4.4.1 Linux Cell BE .............................................................. 24
      4.4.2 Linux PowerPC ............................................................. 25
      4.4.3 Linux X86 .................................................................. 25
   4.5 Collecting Counter Data on Cell BE ....................................... 25
   4.6 Using Remote Connections View ........................................... 25
      4.6.1 Create a remote connection ............................................. 26
      4.6.2 Create a profile configuration ........................................... 27
      4.6.3 Launch remote profiling on remote system ....................... 38
      4.6.4 Create a configuration for remote collecting counter data .... 38
      4.6.5 Launch remote cpc on remote system .............................. 42
   4.7 Collecting Pipeline data on PowerPC ...................................... 42

5. USING THE VPA ANALYSIS TOOLS............................................ 43

   5.1 Profile Analyzer ................................................................... 43
      5.1.1 Load an Existing Profile .................................................. 43
      5.1.2 Profile Navigation ......................................................... 45
APPENDIX A - SAMPLE PROFILING SESSION

5.1.3 Profile Comparison...................................................................................................................... 67
5.1.4 Profile Merge.................................................................................................................................. 70
5.1.5 Symbol Analysis............................................................................................................................ 71
5.1.6 Configure database connections and manage cached database files........................................... 102
5.2 Code Analyzer.................................................................................................................................. 108
  5.2.1 Load an executable for analysis..................................................................................................... 108
  5.2.2 Adding profiling information.......................................................................................................... 111
  5.2.3 Navigate the Executable.................................................................................................................. 112
  5.2.4 Instruction Properties Analysis...................................................................................................... 144
  5.2.5 Statistic Analysis............................................................................................................................ 153
5.3 Pipeline Analyzer................................................................................................................................ 161
  5.3.1 Load an existing pipeline file.......................................................................................................... 161
  5.3.2 Navigating the scroll pipe view....................................................................................................... 165
  5.3.3 Navigating the resource view......................................................................................................... 168
  5.3.4 Show SPU Timing........................................................................................................................... 173
5.4 Counter Analyzer ............................................................................................................................... 177
  5.4.1 Basic concepts for Counter Analyzer............................................................................................. 178
  5.4.2 Load an existing counter data file .................................................................................................. 179
  5.4.3 Navigate the Counter Analyzer Perspective.................................................................................... 184
5.5 Trace Analyzer .................................................................................................................................. 184
  5.5.1 Basic concepts................................................................................................................................ 210
  5.5.2 Load an existing trace file.............................................................................................................. 210
  5.5.3 Navigate the Trace Analyzer Perspective....................................................................................... 212
5.6 Control Flow Analyzer ...................................................................................................................... 216
  5.6.1 Basic Concepts.............................................................................................................................. 216
  5.6.2 Load Call Trace Data File.............................................................................................................. 218
  5.6.3 Control Flow Analyzer Perspective Introduction............................................................................ 218
  5.6.4 View Call Trace Data in Execution Flow Editor Page................................................................. 219
  5.6.5 View Call Trace Data in Call Tree Editor Page............................................................................ 221
  5.6.6 Locate Invocation by Method Name .............................................................................................. 223
  5.6.7 Filter Invocations........................................................................................................................... 225
  5.6.8 Drill Down one Invocation............................................................................................................ 227
6. APPENDIX A - SAMPLE PROFILING SESSION .................................................................................. 229
1. Introduction

What is Visual Performance Analyzer?

Visual Performance Analyzer (VPA) is an Eclipse-based performance visualization toolkit. It consists of six major components: Profile Analyzer, Code Analyzer, Pipeline Analyzer, Counter Analyzer, Trace Analyzer, and Control Flow Analyzer.

Profile Analyzer provides a powerful set of graphical and text-based views that allow users to narrow down performance problems to a particular process, thread, module, symbol, offset, instruction, or source line. Profile Analyzer supports time-based system profiles (Tprofs) collected from a number of IBM platforms and Linux profile tool oprofile 0.9.3.

Code Analyzer examines executable files and displays detailed information about functions, basic blocks, and assembly instructions. It is built on top of FDPR-Pro (Feedback Directed Program Restructuring) technology and allows adding of FDPR-Pro and Tprof profile information. (The Linux® version of FDPR-Pro is available here at AlphaWorks.) Code Analyzer is able to show statistics; navigate disassembled instructions; and display performance comments, instruction grouping information, and map instructions back to source code.

Pipeline Analyzer is a port of the IBM Performance Simulator for Linux on POWER™, another AlphaWorks technology. Pipeline joins the VPA toolkit to provide VPA users with the means of examining how code is executed on various IBM POWER processors. Pipeline Analyzer displays the pipeline execution of instruction traces generated by a POWER series processor. It does so by providing a scroll view and a resource view of the instruction execution.

Counter Analyzer is a common tool to analyze hardware performance counter data among many IBM eServer platforms, which includes systems running AIX, i5/OS, Linux on POWER, Linux on Cell BE. Counter Analyzer accepts hardware performance counter data generated by AIX tools hpmcount and hpmstat in the form of a cross-platform XML file format. The tool uses either build-in hsqldb database engine or external DB2 instance to store the raw performance counter data. The tool provides multiple views to help users identify and eliminate performance bottlenecks by examining the hardware performance counter values, computed performance metrics and also CPI breakdown models.

Trace Analyzer visualizes Cell BE traces containing information such as DMA communication, locking/unlocking activities, mailbox messages, etc. Trace Analyzer shows this data organized by core, along a common timeline. Extra details are available for each kind of events, for example, lock identifier for lock operations, accessed address for DMA transfers, etc.

Control Flow Analyzer is the tool to analyze the call trace data collected by the tool like Performance Inspector JProf. The call trace data contains the information like when one method calls another, how much time is spent in every invocation, and so on. Control Flow Analyzer provides the two major visualization ways to analyze call trace data, which are execution flow graph and call tree table.

How does it work?

Profile Analyzer parses system profiles into an internal profiling data model that supports the profile hierarchy, offset locations, tick counts, CPU counter data, source line information, and disassembly. The plug-in then displays this data model, using various Eclipse views. The system profiles are those produced by Performance Inspector and AIX® Tprof, and Linux oprofile. However, Visual Performance Analyzer can be extended to support almost any platform by converting a system profile to an XML schema that it understands.
Code Analyzer is able to read profiling information generated by AIX Tprof or FDPR-Pro performance tools. It reads in executable files and shared libraries and analyzes them using FDPR-Pro. FDPR-Pro is a post-link analyzer and performance optimization tool that can perform accurate static and dynamic analysis of executable files.

Pipeline Analyzer reads the .pipe and .config input files that are produced by the IBM Performance Simulator for Linux on POWER. An instruction trace is first collected and analyzed by a processor model. The two output files are produced for viewing with either the Performance Simulator or Visual Performance Analyzer.

Counter Analyzer reads the XML output from hardware data collection tools. The XML is parsed and then displayed or graphed for viewing. If a CPI breakdown model is available, the data can be broken down into individual components and viewed in the CPI tab. The CPI breakdown allows the user to view where the workload is spending its processing cycles.

Trace Analyzer reads in traces generated by the Performance Debugging Tool for Cell BE, and displays time-based graphical visualization of the program execution as well as a list of trace contents and the event details for selection.

Control Flow Analyzer reads the call trace data file, and display execution flow graph and call tree to help user analyze when and where one method invocation happens, and how long it runs.

1.1 VPA on Alphaworks

Visual Performance Analyzer was released on Alphaworks to explore the use of Eclipse-based performance tools with IBM customers. VPA is built as an Eclipse Rich Client Platform (RCP) package and there are versions for AIX, Linux, and Windows. An RCP release contains IBM JRE, Eclipse runtime files, all required plug-ins and VPA plug-ins.

1.2 Release History

<table>
<thead>
<tr>
<th>Date</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>09/14/2006</td>
<td>Initial release of VPA to Alphaworks</td>
</tr>
<tr>
<td>06/08/2007</td>
<td>VPA 5.0 Release</td>
</tr>
<tr>
<td>09/28/2007</td>
<td>VPA 6.0 Release</td>
</tr>
</tbody>
</table>
2. VPA Basics

Visual Performance Analyzer is an Eclipse-based tool set that includes: Profile Analyzer, Code Analyzer, Pipeline Analyzer, Counter Analyzer, Trace Analyzer, and Control Flow Analyzer. All of these tools are Eclipse plug-ins.

Figure 1 System Architecture of Visual Performance Analyzer

Profile Analyzer
Profile Analyzer is a system profile analysis tool. This plug-in obtains profile information from various platform specific tools, and provide analysis views for user to identify performance bottlenecks.

Pipeline Analyzer
Pipeline Analyzer gets pipeline information of Power processors from the Sim-GX tool, and provides two analysis views; scroll mode and resource mode.

Code Analyzer
Code Analyzer reads XCOFF (AIX binary file format) files or ELF files running on Linux on Power, and displays program structure with block information. With related profile information, it can provide analysis views on hottest program block as well as some optimization suggestions.

Counter Analyzer
Counter Analyzer reads counter data files generated by AIX hpmcount/hpmstat, and it provides multiple views to help users identify and eliminate performance bottlenecks by examine the hardware performance counter values, computed performance metrics and also CPI breakdown models.

Trace Analyzer

Trace Analyzer reads in traces generated by the Performance Debugging Tool for Cell BE, and displays time-based graphical visualization of the program execution as well as a list of trace contents and the event details for selection.

Control Flow Analyzer

Control Flow Analyzer reads the call trace data file, and display execution flow graph and call tree to help user analyze when and where one method invocation happens, and how long it runs.

2.1 Design Objectives

The base object of Visual Performance Analyzer is to extend the capabilities of Eclipse by adding plug-in support for: system profile, code, pipeline, performance counter, and trace analysis. VPA is a collection of performance data analysis tools that can be used to identify performance bottlenecks. VPA does not supply performance data collection tools. Instead, it relies on platform specific tools, like AIX Tprof, to collect the performance data. When necessary, multi-platform support is provided by converting data into XML. The XML schema is understood by VPA and is parsed and loaded for analysis. The VPA tool is extensible and it achieves this by allowing for additional plug-ins to be added and also by adding integration between plug-ins, e.g. shared internal data models and linked views.

Information about VPA data files:

- The .etm is the XML file for Profile Analyzer
- The .etz is the zipped XML profile data
- The .opm is the oProfile XML file for Profile Analyzer
- The .opz is the zipped oProfile XML file for Profile Analyzer
- Java profile data from IBM JRE Java profiling tools are merged by TProf tools into a single .etm file. No additional post processing is needed.
- The pipeline files are: .pipe data file and .config file is the default configuration file.
- The .pmf is the XML file for Counter Analyzer.
- The .pex is the XML configuration file and .trace is the binary data file for Trace Analyzer
- The .jprof file from Performance Inspector JProf

2.2 Deployment

As a performance analysis tool, Visual Performance Analyzer typically runs on User’s ThinkPad or desktop as a client application. Visual Performance Analyzer can get performance-related data from servers via Remote System Explorer (RSE), Remote Connection Plugin, or by copying the files with FTP or some other means.
Visual Performance Analyzer

Eclipse Rich Client Platform
(With dependency plug-ins)

Java Runtime Environment

Operating System (AIX, Windows, Linux …)

Figure 3 Product Stack of Visual Performance Analyzer

2.3 Software Stack Information

VPA runs on the following Operating Systems:

1. Windows XP with SP2 or later
2. IBM AIX 5.3 in latest Maintenance Level
(3) Linux/x86 – Fedora Core release 7 (Moonshine)

Profile Analyzer, Pipeline Analyzer, Trace Analyzer, Counter Analyzer, and Control Flow Analyzer are Eclipse plugins and are 100% JAVA code. They can run on all above supported platforms.

Code Analyzer is also an Eclipse plug-in, but it depends on FDFR-Pro libraries that are platform-dependent libraries. Code Analyzer can run on Windows, AIX 5.3, and Linux x86 in this release.

Although VPA only runs on the above operating systems, it's important to realize that it can analyze data collected from any platform, providing the data is provided in a format understood by VPA.

VPA supports only IBM J9 JRE 5.

There is an IBM J9 JRE 5 in VPA RCP distribution.

VPA supports the following Eclipse platforms:

1. Eclipse 3.2 with latest fixpack, such as Eclipse 3.2.2
3. Installation

No installer is required for VPA installation. The VPA installation is as simple as:

1. Download a newest VPA RCP release, usually it should be a zip archive or compressed tar archive
2. Extract the archive
3. Run the application by executing vpa binary or vpa.sh script.

The RCP application will not include the following products: Performance Inspector for Windows and DB2 UDB. If you want to use these capabilities, they must install the corresponding product manually.

Configuration

No configuration is required for the VPA application installation.

Advance configuration information is provided in online-help. These configurations address some special requirements, such as setting bigger heap size of JVM for Eclipse when a user analyzes large profile.

Uninstallation

No special uninstallation action is required. If a user wants to uninstall a VPA RCP application, they can simply delete the application directory that VPA was installed to.

3.1 Windows

These steps will walk you through the installation of VPA on your Windows workstation.

3.1.1 Download from Alpha works

Download the latest VPA (Visual Performance Analyzer) from here:

http://www.alphaworks.ibm.com/tech/vpa

Save vpa-rcp-${version}-win32.zip to your favorite download directory.

3.1.2 Extract the compressed file.

Right Click on the file and select Extract All to open the Extraction Wizard.
Extraction Wizard

Welcome to the Compressed (zipped) Folders Extraction Wizard

The extraction wizard helps you copy files from inside a ZIP archive.

To continue, click Next.
As time advances, new versions of VPA will come out. In order to save yourself a lot of headaches with new versions, create the new folder with a name containing the version number and install VPA to that directory. If each version is installed this way, you’ll have multiple working versions. When there is a problem, you can go back to the old version.

Select a root directory and folder to extract files to. You will need to create a folder yourself since it will not create it automatically.
Wait for completion. It's a little slow.

Files have been successfully extracted to the following directory:
C:\WPA

To see your extracted files, check the box below:
- Show extracted files

Press finish to continue.

Click to finish
3.1.3 Create a Shortcut

A window with the folder and its contents will open if you selected “Show extracted files”.

Click on the VPA executable to start VPA.

Otherwise you will see this:
If you see this screen when you start up VPA …

It means that Eclipse is not running one of the VPA tools. You will need to close welcome view and switch to one of the VPA tool perspective by following this procedure.
3.2 Linux

These steps will walk you through the installation of VPA on your Linux workstation.

Supported Linux platforms are: Linux/x86: Fedora Core 7 (Moonshine)

3.2.1 Download from Alphaworks

Download the latest VPA (Visual Performance Analyzer) from here:
http://www.alphaworks.ibm.com/tech/vpa

3.2.2 Extract the compressed file

Go to the directory where gz file is  
```
cd /fadvdir
chmod 755 vpa-rcp-${version}-linux-x86.tgz
```
Decompress the file  
```
tar -xvzf vpa-rcp-${version}-linux-x86.tgz
```
These steps will walk you through the installation of VPA on your AIX workstation.

3.3.1 Download from Alphaworks

Download the latest VPA (Visual Performance Analyzer) from here:
http://www.alphaworks.ibm.com/tech/vpa
Save vpa-rcp-$\{version\}$-aix-ppc.zip to your favorite download directory.

3.3.2 Extract the compressed file

Go to your favorite download directory and follow these steps to extract the VPA tool:

Go to the directory where gz file is ………cd /favdir
Change file attributes ………………………chmod 755 vpa-rcp-$\{version\}$-aix-ppc.tgz
Decompress the file …………………………….gzip –dc vpa-rcp-$\{version\}$-aix-ppc.tgz | tar xvf –
4. Collecting Performance Data

VPA is a collection of performance data analysis tools. It relies on platforms to provide the necessary tools for collecting data and converting the data into a format that is understood by VPA.

4.1 Using Platform Tools

Visual Performance Analyzer works with following tools for collecting profile data.

- AIX Tprof
- Performance Inspector for Windows Tprof
- IBM JRE Java profiling tools
- Linux oProfile on x86, ppc and Cell BE
- AIX hpmcount/hpmstat
- Linux Cell Performance Counter on Cell BE
- Linux Performance Debugging Tools on Cell BE

Profile data from AIX tprof is converted into XML by using the –X flag. The .etm is the XML file for Profile Analyzer; .etz is the zipped XML profile data. Profile data from PI Tprof is in a .out format, which profile analyzer supports directly. Java profile data form IBM JRE Java profiling tools are merged to above tools.

Pipeline data is generated from tools found in the IBM Performance Simulator for Linux on POWER™ project on Alphaworks. The .pipe file is pipeline data file and .config file is the default configuration file.
4.2 Setting up Windows to collect Profiling data

4.2.1 Verify that your Java Runtime is installed on your system

Run the following command:

```
java --version
```

You should see something similar to the following:

```
java version "1.4.2"
Java(TM) 2 Runtime Environment, Standard Edition (build 1.4.2)
Classic VM (build 1.4.2, J2RE 1.4.2 IBM Windows 32 build cn142-20050609 (JIT enabled: jtc))
```

**Note:** You need version 1.4.1 or higher
4.2.2 Verify that the Windows performance tools are installed

VPA runs with the Performance Inspector for Windows performance tools. Run the following command:

Swtrace -?

You should see something similar to the following:

D:\>swtrace -?

SWTRACE Version: 7.1.1

Valid SWTRACE commands: …

The Performance Inspector for Windows package can be downloaded from here:

http://www.alphaworks.ibm.com/tech/pi

4.2.3 Verify PI Tprof

Using the PI tools themselves, you can verify their operation by capturing a system trace using these steps:

Swtrace init
Swtrace enable Tprof
Swtrace on
Swtrace off
Swtrace get
Swtrace post
Post

At this point you should have a PI profile (.out file) in your working directory that you can look at. Refer to PI documentation for details on PI tools.

You can capture traces yourself or you can configure VPA to collect traces. Refer to the Profile Analyzer plug-in section in this document.

4.2.4 Copying data files

Running Performance Inspector for Windows Tprof, produces an ascii profile (.out) file. You can simply use FTP to transfer the file to your system running VPA or open the profile locally if you have VPA installed on the same system. See section 4.5 about using Remote Connections View.
4.3 Setup up AIX to collect Profiling data

4.3.1 Verify that your Java Runtime is installed on your system

Run the following command:

```bash
java -version
```

You should see something similar to the following:

```text
java version "1.4.1"
Java(TM) 2 Runtime Environment, Standard Edition (build 1.4.1)
Classic VM (build 1.4.1, J2RE 1.4.1 IBM build cxppc321411-20040301 (JIT enabled: jtic))
```

**Note:** You need version 1.4.1 or higher

4.3.2 Verify that the AIX performance tools are installed

Recent versions of AIX Tprof can generate XML profiles. AIX 5.3.TL5 or higher is required. The utility that produces a VPA profile from the Tprof output is bundled with the bos.perf.tools package. It includes an updated versions of Tprof, Symlib and the added tprof2xml utility.

Verify installation of bos.perf.tools package:

```bash
lspp -L bos.perf.tools | grep "bos.perf"
```

If not installed, you can use smitty or installp

4.3.3 Verify AIX Tprof

Using the AIX tools, you can verify their operation by capturing a system trace using these steps:

```bash
tprof -eukj -X -A -F -r vpa_test -x sleep 5
```

At this point you should have a Tprof profile (vpa_test.etm file) in your working directory that you can look at.

You can capture traces yourself or you can configure VPA to collect traces. Refer to the Profile Analyzer plug-in section in this document.

4.3.4 Copying data files

All versions of AIX support FTP. So, once AIX Tprof has produced the XML profile (.etm) file, you can simply use FTP to transfer the file to your system running VPA. If VPA has been installed on the same AIX system you can open the profile locally. See section 4.5 about using Remote Connections View.

4.4 Collecting Profiling Data on Linux platform
4.4.1 Linux Cell BE

- Hardware: Cell BE blade
- Software Requirement: fedora core 7 and Cell BE SDK 3.0 installation
- Verify oprofile 0.9.3: opcontrol/opreport –X
- Tool Usage:

After verifying that oprofile has been installed successfully, users should first use "opcontrol --init" to initialize oprofile module; and then, use "opcontrol --event=event:count" to add an event to measure for the hardware performance counters (users can refer to event names and minimal counters by using "opcontrol –l").

Next, use "opcontrol --separate=all" to separate samples based on the given separator. If intending to import the OProfile report into the VPA (Visual Performance Analyzer) tool, then this is a required step.

Users can use "opcontrol --start" to start collecting profiling data, and start one user application: then, use "opcontrol --stop" to stop collecting profiling data; and use "opcontrol --dump" to force a flush of the collected profiling data to the daemon; (NOTE: 'opcontrol --stop' already performs an implicit dump so the latter command is not required on Cell).

Finally, use "opreport -X -g -l -d -o xxx.opm" to generate a specified XML output, which can be imported to Profile Analyzer plugin of VPA. The xml output file must be suffixed with the extension '.opm', which identifies an acceptable file.

Users must further issue "opcontrol --reset" to clear out data, and choose "opcontrol --deinit" to unload the oprofile module. Failing to do so between a PPU profile and an SPU profile will lead to unpredictable results.

Some important command usage:

- **opcontrol --init**:
  - loads the oprofile module and oprofilefs

- **opcontrol --event=event_name:count:unit_mask:kernel-space_count:user-space_count**:
  - Choose an event with specified event_name, count, unit_mask, kernel-space counting, user-space counting.
  - Here, the unit_mask, kernel-space counting, user-space counting are optional.
  - A default event can be specified with the command "opcontrol --event="default"" .
  - Generally, the default event is some arch-specific predefined number of processor cycles.
  - Currently, for SPU, only average instruction cycles profiling is supported using the command:
    - opcontrol --event=SPU_CYCLES:100000
  - PPU and SPU profiling need to be collected separately and before each profile collection the "opcontrol --deinit" is recommended.

- **opcontrol -l**:
  - List event types and unit masks

- **opcontrol --start/--stop/--reset/--deinit**:
  - Start running the oprofile, stop oprofile, reset (i.e., "clear") the profile data in default session. Unload oprofile module.

- **opreport -X -g -l xxx.opm**:
  - Generate a specified XML output
    - -X: specifies the output file in XML format.
    - -g: show source file and line for each symbol.
    - -l: list per-symbol information instead of a binary image summary.
    - -d: show per-instruction details for all selected symbols.
4.4.2 Linux PowerPC

- Hardware: System p servers or POWER blade
- Software Requirement: Linux, oprofile (oprofile Download Link: http://oprofile.sourceforge.net)
- Verify oprofile: the same as Cell BE
- Tool Usage: the same as Cell BE

4.4.3 Linux X86

- Hardware: X86 based machine
- Software Requirement: Linux, oprofile (oprofile Download Link: http://oprofile.sourceforge.net)
- Verify oprofile: the same as Cell BE
- Tool Usage: the same as Cell BE

4.5 Collecting Counter Data on Cell BE

- Hardware: Cell BE blade
- Software Requirement: fedora core 7 and Cell BE SDK 3.0 installation
- Verify CPC exist.
- Tool Usage:
  After verifying that cpc has been installed successfully, you can use cpc to collect counter data directly.
  User can use “cpc -e EVENT,EVENT,... -i INTERVAL –X xml based report name -t TIME” to collect counter data in system wide
  And user also can use “cpc -e EVENT,EVENT,... -i INTERVAL –X xml based report name Workload” to collect counter data for special workload.
  Here is an example:
  ```
cpc -e 2100, 2101, 2103, 2104 -X cpc4event.pmf ls
  ```
  User collect counter data for ls and output the xml report to cpc4event.pmf.

Some important command usages:

```
cpc -l
```
Display all available events.

```
cpc -e EVENT,EVENT,... --sampling-mode -c CPUS -i 100000000 -X xml based report name -t TIME
```
Monitor the specified physical CPUs (CBE nodes) in system wide mode instead of monitoring a single workload. ‘CPUS’ can be 'all' to monitor all available CPUs, or it can be a comma-separated list of physical CPUs. This option only applies if a workload is not specified.

4.6 Using Remote Connections View
User can configure VPA to use Remote Connections View to remotely collect profile data and transfer files. Through the Remote Connections View, firstly, user can create a connection to the remote system, and then create configurations on launching profile data collection remotely towards different systems and profiling tools.

4.6.1 Create a remote connection

The below steps illustrate how to configure a remote connection.

➢ Open Remote Connections View

Launch VPA, open Profile Analyzer or Counter Analyzer perspective, Remote Connections View appears by default.

Or choose Window -> Show view -> Other -> Visual Performance Analyzer -> Remote Connections to open it.

➢ Configure connection parameters

1. Press "Create Connection" button on upper-right Remote Connections view. A "Connections Properties" dialog is started up for fill in.

**Host:** generally specify the IP address of the remote system here

**Port:** since Remote Connections view connects to remote system with SSH protocol, "22" is the default port value here.

**User:** specify a user name on the remote system. *Note: here prefers to a user with "root"(AIX, Linux) or "administrator"(Windows) access right, because such kind of user are required by running most system performance counters or tools.*

**System:** specify the type of the remote system. There are three options: AIX, Linux, Windows

3. Specify a public key file if authenticating with remote SSH server on public-key authentication.

This field is optional, and is required in case of that user wants to set up public-key authentication between remote OpenSSH server and local SSH client.

4. Press **OK** button to finish creating a connection.

### 4.6.2 Create a profile configuration

- Create a profile configuration for remote AIX system

Right click the "Profile Analyzer" over remote AIX connection node, and choose the "Create Configuration... " menu. Then a wizard will lead user step by step to create a remote profiling configuration:

1. Profiling tool and its working directory (profiling tool location) are set by default. Pick up a CPU type, click **Next**.
2. Define related parameters if a application needs to be launched along with the profiling. If the application is a Java application running an IBM Virtual Machine for Java, select the **Enable Java profiling** checkbox, which will define the IBM_JAVA_OPTIONS environment variable for the Java process being started, so that JIT-compiled Java methods are profiled. Note: the fields on this wizard page are optional, leaving them as blank will run profiling on system wide mode.
Pressing "..." button to launch a SSH session with the remote system, and browse the file system on remote site.
3. Configure the start/sop qualifier options for profiling

- **Start qualifier options:**
  
  Choose to profile manually or with the application. (Only manual profiling is available if an application is selected to launch.) Fully automatic profiling is the most straight forward: it involves a single click of the Run icon, which will launch the profiler and the application, run the application to completion, stop the profiler, and load the profile into Profile Analyzer. Fully manual profiling requires you to start the profiler, start and stop the application (if one was entered on the second page), and stop the profiler.

  **For fully automatic profiling**, choose **With application** and leave the entry fields to their default values of 0. If you want to run your application automatically but give it a predetermined time to “warm up” before profiling begins, choose **With application** and enter the warm-up time in the **Delay profiling start by (seconds)** entry field.

  **For fully manual profiling**, choose Manual. You can define the time for profiling in Profile for entry field.

- **Stop qualifier options:**
  
  You can input number of successive run according to your needs and choose to terminate profiling or kill application when application exits and profiling ends.

  **Important note:** If choose running profiling with a application, user has to ensure tprof be aware of where is the executable and binary of the application. This could be achieved by setting the PATH environment variable on remote system.

4. Configure time or event based sampling options. Choose supported CPU events to profile, or leave System timer as chosen. The number of cycles can be defined for the processor is neither halted nor in sleep.
In addition, there are also some options for AIX tpriof XML converter could be configured. These options affect the collected profiling data which is output as XML format.

5. Press Finish, a new configuration will appears under the "Profile Analyzer" product node of the remote connection to AIX system.

➢ Create a profile configuration for remote Linux system

Right click the "Profile Analyzer" over remote Linux connection node, and choose the "Create Configuration..." menu. A wizard will leads user step by step to create a remote profiling configuration:

1. Profiling tool and its working directory (profiling tool location) are set by default. Note: Linux system use OProfiles as profiling tool. Pick up a CPU type, click Next.
2. Define related parameters if a application needs to be launched along with the profiling. **Note**: the fields on this wizard page are optional, leaving them as blank will run profiling on system wide mode.
Pressing "..." button to launch a SSH session with the remote system, and browse the file system on remote site.

3. Configure the start/sop qualifier options for profiling

![Create Remote Connection Configuration](image)

- **Start qualifier options:**
  Choose to profile manually or with the application. (Only manual profiling is available if an application is selected to launch.) Fully automatic profiling is the most straightforward: it involves a single click of the Run icon, which will launch the profiler and the application, run the application to completion, stop the profiler, and load the profile into Profile Analyzer. Fully manual profiling requires you to start the profiler, start and stop the application (if one was entered on the second page), and stop the profiler.

  For fully automatic profiling, choose **With application** and leave the entry fields to their default values of 0. If you want to run your application automatically but give it a predetermined time to "warm up" before profiling begins, choose **With application** and enter the warm-up time in the **Delay profiling start by (seconds) entry field**.

  For fully manual profiling, choose **Manual**. You can define the time for profiling in the **Profile for (seconds, 0 for indefinite):** entry field.

- **Stop qualifier options:**
  You can input number of successive run according to your needs and choose to terminate profiling or kill application when application exits and profiling ends.
4. Configure time or event based sampling options. Choose supported CPU events to profile, or leave System timer chosen as default. In addition, there are also some options for Linux OProfiles XML converter could be configured. These options affect the collected profiling data which is output as XML format.

- x86 and AMD64 CPUs

- Cell BE
5. Press **Finish**, a new configuration will appears under the "Profile Analyzer" product node of the remote connection to Linux system.

- Create a profile configuration for remote Windows system

Right click the "Profile Analyzer" over remote Windows connection node, and choose the "Create Configuration..." menu. A wizard will leads user step by step to create a remote profiling configuration:

1. "Performance Inspector tprof" is defined as the default profiling tool for Windows system. Press button 📂 to browse remote Windows file system and choose the installed location of Performance Inspector tprof. Then pick up a CPU type. Click **Next**.
2. Define related parameters if an application needs to be launched along with the profiling. If the application is a Java application running an IBM Virtual Machine for Java, select the **Enable Java profiling** checkbox, which will define the IBM_JAVA_OPTIONS environment variable for the Java process being started, so that JIT-compiled Java methods are profiled. Note: the fields on this wizard page are optional, leaving them as blank will run profiling on system wide mode.
3. Configure the start/sop qualifier options for profiling

![Configuration Options](image)

- **Start qualifier options:**
  
  Choose to profile manually or with the application. (Only manual profiling is available if an application is selected to launch.) Fully automatic profiling is the most straightforward: it involves a single click of the Run icon, which will launch the profiler and the application, run the application to completion, stop the profiler, and load the profile into Profile Analyzer. Fully manual profiling requires you to start the profiler, start and stop the application (if one was entered on the second page), and stop the profiler.

  **For fully automatic profiling,** choose **With application** and leave the entry fields to their default values of 0. If you want to run your application automatically but give it a predetermined time to “warm up” before profiling begins, choose **With application** and enter the warm-up time in the **Delay profiling start by (seconds) entry field**.

  **For fully manual profiling,** choose **Manual**. You can define the time for profiling in Profile for entry field.

- **Stop qualifier options:**
  
  You can input number of successive run according to your needs and choose to terminate profiling or kill application when application exits and profiling ends.

4. Configure time or event based sampling options. Choose supported CPU events to profile, or leave System timer chosen as default. The number of cycles can be defined for the processor is neither halted nor in sleep.
5. Press **Finish**, a new configuration will appears under the "Profile Analyzer" product node of the remote connection to Windows system.

### 4.6.3 Launch remote profiling on remote system

1. Choose a configuration of Profile Analyzer in Remote Connections View. Then **start tool button** is enabled and the **progress bar** is ready. If the configuration defines an application to run with the profiling tool, the **start application** button is also enabled.

2. Start profiling
   
   - Press **start tool** button to launch remote profiling. According to the configuration, the profiling may stop after running for specified time duration, user also can press **stop tool button** to halt the remote profiling manually.
   
   - Press **start application** button to start application manually if an application is defined with the profiling. According to the configuration, the profiling tool may start after the application running for a predefined time.

3. Finish profiling, the result data files are download to local VPA automatically. Double click the result data file under a configuration in Remote Connections view to launch Profile Analyzer to open it.

### 4.6.4 Create a configuration for remote collecting counter data

Right click the "Counter Analyzer" over remote Linux connection node, and choose the "Create Configuration..." "Pop-up menu. A wizard will leads user step by step to create a configuration for remote performance counter data collection:
1. Currently, for Counter Analyzer configuration on remote Linux system, only **Cell BE performance counter (CPC)** for Cell BE is supported. By default, /tmp is set as the directory to run performance counting. File Prefix is optional parameter, user can define custom prefix for the performance collecting data file.

![Create Remote Configuration](image)

**Remote Configuration of Counter Analyzer**

Select the tool used to profile, select the CPU type.

- **Configuration Name**: NetConfiguration
- **Operation System**: Linux
- **Profiling Tool**: CellPerfCount
- **Tool Location**: /tmp
- **Processor Type**: Cell BE
- **File Prefix**: 

2. Choose monitor mode for CPC. If **Monitor CPUs with application** is selected, then it needs define an application and its arguments to be executed; if **Monitor CPUs on system wide mode**, it needs to specify the time duration to monitor, and which Cell BE nodes to monitor (monitor all Cell BE nodes by default).
3. Configure other more options:
Event list

There are 4 counters available in CPC. So 4 events are group as a event set. Multiple event sets can be defined.

Switch timeout

After defining multiple event sets, all event sets are loaded into the kernel. The kernel will run each one for a specific amount of time defined by **Switch timeout**.

Count interval

Specify the sampling interval time for CPC. Note: if the interval is not very large, it will causes collecting a great number of performance counter data.
• Sampling mode
Indicates the type of data stored no the sampling buffer. If Count interval is specified, the sampling mode will default to "count".

• Count mode
Define CPC count in specified mode.

• Start/stop qualifier options
Define CPC starting or stopping once a qualifier is happen.

4.6.5 Launch remote cpc on remote system

1. Choose a configuration of Counter Analyzer in Remote Connections View. Then start tool button is enabled and the progress bar is ready.

2. Press start tool button to launch remote performance counter

3. Finish counting performance data, the result data files are downloaded to local VPA automatically. Double click the result data file in Remote Connections view to launch Counter Analyzer to open it.

4.7 Collecting Pipeline data on PowerPC

Pipeline Analyzer is a port of the IBM Performance Simulator for Linux on POWER™, another alphaWorks technology. Please refer to the directions given by this project for collecting pipeline data. While VPA provides the Pipeline data analysis tool, the project provides the tools necessary for collecting and generating Pipeline data files.
5. Using the VPA analysis tools

This section describes the use of each plug-in, structure of the system by first focusing on some typical usage scenarios where various tasks are performed, then outlining the major components of the system and their interactions. You can find this information by selecting Help - Help Contents within VPA. To get context sensitive help, press F1 for Windows and AIX or press Ctrl+F1 for Linux.

5.1 Profile Analyzer

Profile Analyzer is a tool that allows you to navigate through a system profile, looking for performance bottlenecks. It provides a powerful set of graphical and text-based views to allow users to narrow down performance problems to a particular process, thread, module, symbol, offset, instruction or source line. It supports profiles generated by Performance Inspector (tprof) and AIX tprof. It also merges IBM JRE Java profile data when it is merged into the above profiles. To load huge profile data files and reduce memory footprint, Profile Analyzer now uses database to cache profile files. The current version supports DB2 and an embedded database.

You can also find the Profile Analyzer User Guide from within VPA. Select Help - Help Contents within VPA. To get context sensitive help, press F1 for Windows and AIX or press Ctrl+F1 for Linux.

5.1.1 Load an Existing Profile

When you first start Visual Performance Analyzer, close Welcome View and then select Profile Analyzer to load the plug in.
You can also load Profile Analyzer perspective by choosing **Window - Open Perspective - Other - Profile Analyzer**.

If you already have profiles generated by TPROF or a Profile Analyzer compatible XML-based profile generator, you can select **File – Open File** to open the profile that Profile Analyzer supports. Profile Analyzer profiles must have one of the following extensions:

- `.out`, `.etn`, or `.etm`
- `.opm` and `.opz`

In VPA a profile data file loading process is able to run as a background runnable job. When VPA is loading a file, you can click a button to put the loading job to run in the background. While the loading job is running in the background, you can use Profile Analyzer to view already loaded profile data files, or even start another loading job at the same time.

As already stated, profile data files are loaded into database tables and kept in database tables until the user deletes them. Once a profile data file is successfully loaded into a database, further attempts to load the same data
file will result in the data being reloaded directly from the database tables. Profile Analyzer does not need to read and parse the original file again. This allows for much faster loading of profile data into VPA after the initial database caching.

Note: although further use of a profile data file results in loading from the database, the original file is still required for Profile Analyzer to work properly. This is because not all of the content of the original file is loaded into database tables. For example, time data is kept in original file and we only store the offset and length information in database tables. When needed, this data is read from their original file on-demand.

5.1.2 Profile Navigation

The following are tasks that you can perform to navigate around profiles within Profile Analyzer.

5.1.2.1 Navigate process hierarchy

The Process hierarchy view appears by default in the top center pane. It shows an expandable list of all processes within the current profile. You can expand a process to view its module, later thread and etc. You can also view the profile in the form of thread or module and etc. Actually, you can define the hierarchy view by right-click profile and choose Hierarchy Management. Thread data is not available in merged profiles (.etm extension).

For more information about Hierarchy Management, you can refer to Navigate Generic Hierarchy Model

The following screen capture shows a process hierarchy in its unexpanded state:
As in most Profile Analyzer views, objects are sorted from most to fewest ticks. In this view you can see that the **IdleProcess** was the process with the most ticks, indicating either I/O delays or actual idle time during the process (for example, if the application being profiled ran on one CPU and the system had a second, mostly idle CPU).

You can expand a process to view the threads or modules beneath it. As you select a process, thread, or module, the Symbols view updates to display the list of symbols that belong to that process, thread, or module. The Samples Distribution Chart also changes, as you select different processes or threads, to display the proportion of ticks used by the most important modules within the selected process or thread.

The following two views show part of the above process in **Process>Thread>Module** hierarchy:

Navigate generic hierarchy model

A process may have some threads, and each thread can visit some modules (for instance, DLLs) and call procedures/methods (symbols) in these modules. In default condition, you can observe systems in the hierarchy of **Process > Core > Thread > Module, Process > Thread > Module, Process-> Module and Module**. With the function of generic hierarchy model, you can create your own hierarchy view. For example, if you want to group threads which use a common module, you can display the hierarchy Process > Module > Thread by creating it in the **Hierarchy Management**.

Attach hierarchy file to profile file, please do the following steps:

1. Right-click in the process hierarchy view and choose **Hierarchy Management**
2. Click the ... button to open a new hierarchy file for attaching to the profile file
3. select the check-box to attach the new hierarchy file to the profile file

Here are some pictures to show how to attach a new hierarchy file - "testing.xml":

---

DLM Alphaworks Page 46 of 244
To create your own hierarchy view, follow these steps:

1. Right-click in the process hierarchy view and choose **Hierarchy Management**
2. In the Hierarchy Management Wizard, click **New**
3. Give your hierarchy a specific name if you like, or the system will generate a name for you.
4. Select the element you want to have in your view. You may reorder your hierarchy by choosing **up** or **down**
5. Click **Apply** or **Ok**

Here is a new hierarchy view we created to see the threads under each module:

Click the **New** button to create a new hierarchy view:
When you click **Apply** or **OK**, you can see the change in **Process Hierarchy View**

You may add other hierarchy views in the Process Hierarchy View as you like.
To lines of editor symbol table, user has to set the symbol threshold of the hierarchy. Symbol table contains symbols of the selected hierarchy node, but it does not list all the symbols in default. It often lists no more than a number of them. This is called threshold. After user sets the threshold to another value, the symbol table is refreshed and the listed symbols are no more than the new threshold. The default threshold of editor symbol table is 100, that is, no more than 100 symbols is listed in the table whatever hierarchy node is selected.

If user selects the Change Profile Symbol Threshold item in the context menu, then it pops up the threshold box.
The default symbol threshold is 100. In the symbol table, there are no more than 100 rows listed.

If user sets the threshold to new value 200, the editor symbol table is refreshed and the number of the rows is no more than 200.
If user set the threshold to ‘All’, symbol table is refreshed and it lists all symbols of the threshold.

5.1.2.2 Bucket Management

You can management your buckets settings by doing the followings:

1. Right-click in the process hierarchy view and choose **Buckets-> Bucket Management**
2. Attach a new bucket management file to the profile file
3. Create a new bucket or edit/remove an existing bucket by clicking the corresponding buttons
If you have changed bucket definition, the current opened profile will be automatically reloaded. If there are any other opened profiles that are also affected by the new bucket definition, they will not be automatically reloaded. User must reload them manually.

5.1.2.2.1 Add new bucket to existing one

If there are existing buckets to group some components, you can add one to another so as to further filter those components and get the views of those you really want.
If clicked, the bucket selection dialog will open.

After selecting a bucket, the bucket property dialog will open for users to modify the filters as follows:
You can add the filter requirement in this wizard and give this bucket a new name. In order to view this bucket in Process Hierarchy View, you may move it up in the Bucket Management. You can also edit, delete, enable, disable bucket or create new bucket group in Bucket Management.

5.1.2.2 Create new bucket to group common components

In a system, certain groups of modules, threads, or symbols share common components. For example, in a WebSphere java process, threads can be logically grouped by name, with one group containing threads with names like tid_WebContainer*, another with names like tid_Gc_Slave_Thread_*, and another with names like tid_Alarm*. Buckets function in Profile Analyzer provides mechanisms for users to group different components, such as objects, together into buckets. It acts as a new layer in the Hierarchy Process View.

To create a new bucket, please follow these steps:

1. Right-click in the process hierarchy view and choose Buckets-> Create new bucket
2. Give you bucket Name and choose the type of components you want to group from thread, process, module
3. Give the components' name you hope to filter in the corresponding blank, such as tid* thread in thread filter
4. Choose the bucket group you want to put your new bucket into.

The following is a example of creating a bucket filtering tid* thread
Now you can define multiple filters for a bucket.

You can see a new layer called test2 appeared above the thread hierarchy view. All the threads with the title beginning `tid` remain as follows:

5.1.2.3 Navigate Java package hierarchy

You can view the Java package hierarchy for a process, thread, or module that contains JITted methods using the Java/Hierarchy view. The process or thread must contain a JITCODE module, and the module must be the JITCODE module.
Note: The Java/Classes hierarchy view is normally displayed at the bottom right, along with the Disassembly/Offsets view, the Temporal Profiling view, and the Profiling Configurations Console view. If you cannot see it displayed, you can open it with Windows -> Show View -> Other -> Profile Analyzer-> Java/Classes hierarchy.

To view the Java package hierarchy for a process, thread, or JITCODE module, click on the Java/Classes hierarchy tab, then navigate the Process hierarchy view. As you select different processes, threads, or modules, the Java/classes hierarchy is updated to show you the Java package hierarchy for any active methods. The following screen shot shows an initial view of the Java package hierarchy for a JITCODE module on an AIX profile:

You can select a top-level name to view all methods in all packages that match that name (for example java). Or you can expand a top-level name to display the packages and classes beneath it. By selecting a package or class in the hierarchy, you can limit the list of displayed methods to those in that class or hierarchy. The following shows the active methods for the BufferedWriter class in the java.io package:
When you double-click on a method in the table at the right, the following views are updated to display information for that method:

- The OffsetAsm Information view
- The Disassembly Resolved Call Information view

### 5.1.2.3.1 Notes on appropriate use of this view

The Java/Classes Hierarchy view is useful if you are working on tuning the code for specific classes under your control and are trying to determine what bottlenecks exist in those classes. However, you should avoid the pitfall of focusing simply on the classes you have control over (classes that you can make source code changes to). For example, trying to tune the hottest method in a class you own may provide some benefit, but if that method takes only 1% of total ticks, while java/io/BufferedWrite.write takes 20%, you should look at what methods are calling java/io/BufferedWrite.write (using the Disassembly Resolved Call Information view on Windows, Linux-IA32, and Linux-x86-64, or using a call graph profiling tool such as ITRACE on other platforms).

Conversely, just because the methods in your packages do not show significant CPU usage does not mean they are efficiently written or have no impact on performance. For instance, if a significant percentage of profile time is spent in the JVM garbage collection library (e.g. libj9gc23.so or j9gc23.dll), this may indicate that you are making inefficient use of memory by allocating too many objects or failing to make them available to garbage collection when they are no longer needed.

### 5.1.2.4 View counters

In Counters view, you can view the ticks of process, thread, module or bucket you selected in the Process Hierarchy View. To open this view, choose Window -> Show View -> Others->Profile Analyzer -> Counters or just find it in left pane.
For example, if you select a process in the **Process Hierarchy View**, you can see the total ticks of this process in the **Counters** view.

5.1.2.5 **Select default counter**

Whenever you have a profile that contains more than one counter, Profile Analyzer will allow you to choose which is the default counter used for sorting. Profile Analyzer supports the “sort by counter” feature. A user can select any available counter as the default “sort” counter. Once the default counter is selected, all editors and views understand this selection and will sort/format their outputs according to the current active “sort” counter. Following picture shows this feature:
5.1.2.6 View module sample distribution

The Samples distribution chart shows a tick distribution for modules in the currently selected process or thread in the process hierarchy.

**Note:** If you cannot see this view from within the Profile Analyzer perspective, select Windows -> Open view -> Other -> Profile Analyzer -> Samples Distribution Chart.

This view provides a starting point for determining where you should focus your attention. For example, the following screenshot shows the graph for a java process using 23.4% of total profile time:

![Sample Distribution Chart](image)

This graph shows that the JITCODE module (the module containing JIT-compiled Java methods) was the busiest module for this process, which suggests that some tuning of Java methods may be advisable.

The following graph (for a different Java program on a different system) shows heavy activity both in j9gc22 and in JITCODE. j9gc22 is the Garbage Collection library of the IBM Virtual Machine for Java (J9) indicating that the application may be memory constrained, or may be creating new objects too frequently. The third column (j9jit22) is the JIT compiler library, indicating that the profile may not have run for very long, since long-running applications typically have a small percentage of time used by the JIT. (A high percentage of time in the JIT may also indicate excessive use of invoke_interface calls, which require JIT library runtime support even when executing JIT-ted methods).
5.1.2.7 View profile details

You can see the detail information of a profile file when you select it in the Process Hierarchy View. We can see it from the following example:
An opm file is opened, and its corresponding information shows up in the Profile Details View.

### 5.1.2.8 View resolved call information

When you open an IA32 profile, Profile Analyzer can analyze the disassembly in it to identify all call sites that have an immediate address as a target, and can attempt to connect those call sites to target symbols. This is done automatically for you if the Resolved Call Information view is visible. The following shows the Resolved Call Information view for a method selected from the jvm.dll module:
5.1.2.9 View basic blocks

A basic block is a block of instructions that contain a single entry point and at most two exit points. Basic blocks are a concept used by compilers to perform dataflow analysis and to perform effective optimizations. Profile Analyzer attempts to detect basic blocks by analyzing the targets of all branch instructions within the disassembly for a symbol. Note that the basic blocks detected by Profile Analyzer may not match the basic blocks indicated in a compiler listing, as the compiler may use a higher-level basic block structure that includes internal branches. For example, a single source or intermediate-language instruction would likely not span multiple basic blocks from a compiler perspective. However, some source or intermediate-language instructions may result in multiple basic blocks at the disassembly level. An array assignment operation in Java is one such instance: the assignment is a single source statement, but may require both a null check and an array bounds check, each of which are intermediate-language instructions that may result in multiple conditional branches in the resulting disassembly.

You can see basic block information by choosing Windows-> Show View -> Other... -> Profile Analyzer -> Basic Block or just find it in the left panes.

When you open a process, you can see its basic block information as follows:
Each basic block has a number (BB1, BB2 etc.), a tick count, zero or more incoming edges, and one or two outgoing edges (a terminating basic block does not have any outgoing edges). Each block with ticks is colored red, magenta or blue according to the same rules used to determine symbol tick color, and shaded according to the relative tick count of the basic block as compared to the symbol as a whole.

You can click on a basic block to highlight its incoming and outgoing edges in red:
In this view, BB2 was selected, and its outgoing edges to BB3 (the "fallthrough" basic block) and BB4 (the target basic block) are highlighted.

### 5.1.2.10 Basic Block Table View

After Profile Analyzer collects and analyzes disassembly information of instructions, basic block table view shows its multiple basic block at the disassembly level. The view shows a table containing basic block numbers (BB1, BB2 etc.), address offsets for the first instruction in each basic block, and tick information for each basic block if disassembly can be generated for a symbol. There is a hotness bar alongside the view.

You can open basic block table view by choosing **Windows -> Show View -> Other... -> Profile Analyzer -> Basic Block Table** or just find it in the left bottom panes.

![Basic Blocks Table](image)

#### 5.1.3 Profile Comparison

The following are tasks you can do to compare profiles within Profile Analyzer:

#### 5.1.3.1 Compare two profiles
1. Click the **Compare two profiles** button in the **Profile Comparison** view toolbar. Alternatively, for any open profile, right-click in the process hierarchy view and select **Compare this profile with another**.

2. In the Profile Comparison Wizard, select two profiles you want to compare. The wizard supports compressed (.etz) profiles. Comparison of tprof profiles is still limited in that it does not expose compilation levels.

3. Click **Next**. Enter the following values for each profile:
   - **Transaction rate**: The transaction rate is the number of transactions completed per elapsed second. What we call a transaction varies from workload to workload, but it is a consistent indicator of the amount of work we are doing per second. If we are running background jobs, the number of transactions might be the number of jobs. If we are running an HTTP server, the number of transactions might be the number of HTTP requests we have served. For each type of workload, the transaction is clearly defined.
   - **CPU utilization**: CPU utilization is a percentage that describes how busy servers are. It is defined as the average utilization of all CPUs in a server.
   - **Number of CPUs**: This is the number of CPUs available to the system in each of the profile runs being compared. It is very common that we compare runs with different number of CPUs. For example, we may compare a 1-CPU run against a 4-CPU run to determine how well a workload scales up as we add CPUs to the configuration.

4. Click **Finish**. Two files are loaded and opened.

5. Right-click the process, thread or module of the first file in the process hierarchy view which you want to compare and select **Mark for comparison**.

6. Go to the other profile, right-click the process, thread of module of the second file which you want to be compared with and select **Compare with <filename>**.
7. In the **Profile Comparison** view, the detailed information of modules you have selected to compare is listed as follows:

![Profile Comparison View](image)

### 5.1.3.2 Understand the calculations

The comparison tool uses the normalization factors entered in the Profile Comparison Wizard to calculate the microseconds of CPU consumed per transaction (us/Tx). Since the us/Tx values are computed on a per transaction basis, they can be compared directly from profile to profile.

The us/Tx values are calculated as follows:
1. **Calculate percentage of total ticks in the specific symbol:**
   \[ \text{CPU\%} = \frac{\text{Ticks in the specific symbol}}{\text{Total ticks in the profile run}} \]

2. **Calculate transactions per busy second:**
   \[ \text{ITR} = \frac{\text{Transaction rate}}{\text{CPU utilization}} \]

3. **Calculate total CPU microseconds per transaction:**
   \[ \text{Total microseconds per transaction} = \frac{1,000,000}{\text{ITR} \cdot \text{Number of CPUs}} \]

4. **Calculate average CPU microseconds per transaction in the specific symbol:**
   \[ \text{us/Tx} = \frac{\text{Total microseconds per transaction}}{\text{CPU\%}} \]

### 5.1.3.3 Save a profile comparison

1. Click the **Save Comparison** button in the Comparison view toolbar. Alternatively, right-click anywhere in the view and select **Save Comparison** from the pop-up menu.

2. The **Save As** dialog opens. Browse to the desired directory and enter a file name.

3. Click **Save**. The comparison will be saved as a Profile Analyzer comparison (.etc) file. This file contains both compared profiles (zipped) and the normalization factors used in the comparison.

### 5.1.3.4 Open a profile comparison

1. In the Navigator view, double-click the Profile Analyzer comparison (.etc) file.

2. Both compared profiles will open automatically in the Profile Analyzer editor as temporary files, and the Comparison view populated.

### 5.1.4 Profile Merge

If user has profiled a benchmark multiple times using TPROF, he can merge the .out files for these runs using the Merge Wizard in Profile Analyzer. This can be useful for several types of situations:

- If user is going to measure a short-run application (or a short-run phase such as startup of a JVM during a benchmark), each individual profile may have too few ticks to draw meaningful results, but a with a merged profile patterns may begin to emerge

- If user is going to measure different CPU events (ticks, data cache misses, branch mispredictions) on different runs of a benchmark, he can merge these runs and see the data for all counters in a single profile

- If user wants to compare two runs, he can use profile merging to see which processes, modules, symbols, and symbol offsets were active in both runs.

To merge several profiles, at least one of them must be opened. Then follow these steps:

1. Right-click in the process hierarchy view and choose **Merge this profile with another**.

2. In the **Merge Profiles Wizard**, select one or more profiles from the current project and click **Add >**, or use the **Browse** button to open a file dialog to select profiles from other locations. Note that Profile Analyzer will only let user to add profiles whose platform matches the profile already added to the list. Click **Next**.

3. On the **Select processor type from platform family** page, select a CPU type (this option is only available on platforms where different CPU types support different counters). Click **Next**.
4. On the **Counter options** page, user can select each profile in turn and chooses what counter to attribute its events to. One of the profiles must be the "primary" profile; this is the profile used to merge other profiles into. Click **Next**.

5. Select a file name and click **Finish**.

   At any time the **Finish** button is not grayed out user can click it to merge immediately.

When user merges profiles, Profile Analyzer creates a new file with an .etm extension (ETM=e-Tune Merged). This file is in a Profile Analyzer-supported XML format. Profile Analyzer also opens the file immediately after the merge. This profile looks much like ordinary TPROF profiles when viewed inside Profile Analyzer, with three differences:

- Processes, modules, symbols, and offsets that had data from more than one source profile are colored in green
- Multiple counter columns may appear in the Offsets view as well as ticks, if user chooses different counters for each source profile.
- No threads data is available, as it does not make sense to merge thread data from separate runs.

### 5.1.5 Symbol Analysis

The following are tasks that you can perform to analyze symbols within Profile Analyzer:

#### 5.1.5.1 Code Miner support

The Code Miner support in Visual Performance Analyzer enables you to populate an SQL database with information from Profile Analyzer profiles, and then perform SQL queries on the database to detect performance patterns that are not easily detected by traditional profilers. The database tables allow you to associate profile counter information, symbols, and disassembly instructions so that you can find inefficient or highly active patterns of instructions, instruction sequences, symbols, register usage, and so on. For example, a Code Miner query can be used to find the hottest pairs of sequential instructions, or all symbols that contain a particular instruction sequence, or all symbols that are hotter than a certain threshold that have a certain pattern in their name. Code Miner is ideal for analyzing flat profiles, namely profiles where no single symbol uses more than a fractional percentage of total ticks. In flat profiles, the objective is to find patterns of disassembly code, or usage patterns of certain types of symbols, that are inefficient. Without Code Miner it is extremely difficult to determine which patterns are worth investigating. Because Code Miner lets you determine the overall cost of a particular pattern within an entire module or an entire profile, you can use it to detect the patterns that will yield the maximum benefit when optimized, replaced, or eliminated.

Code Miner user interface support within Profile Analyzer includes a Code Miner wizard for populating data from a profile, and two views: a **Code Miner Query** view, that lets you query the Code Miner tables for a particular profile to find patterns of interest, and that displays the results in a sortable column-based table; and a **Query Tree** view that saves queries and database configurations so that you can easily locate, edit, re-run past queries or queries imported from another user, such as Compiler listings.

#### 5.1.5.1.1 Populate Code Miner Database

In Profile Analyzer, you can choose to keep profile or trace file data into DB2 database. This can be realized via **Populate Code Miner Database Wizard**. Every time when you open Populate Code Miner Database Wizard, you can choose to create new table, append to existing table or clean tables. If you decide to keep data into database, the prefix of table which is designed to store data should be defined at first. In the next page of
wizard, given name, host, port and admin password, Profile Analyzer can get access to the database and populate data automatically. Please follow the steps below to populate profile file into new table in existing database:

1. Open a profile file and right click.

2. Choose **Populate CodeMiner Database**.

3. Defines table prefix, create new tables on database and include proper fields as you need.

4. Input db2 connection information into next wizard page. Be sure to pass firewall so as to connect database.
5. Click Finish

5.1.5.1.2 Code Miner Database Queries

To check data in database, run query in Code Miner Queries. Please follow the steps below:

1. Create a new connection to the db2 database above through Code Miner Queries or Query Tree view
   - Press Edit button in Code Miner Queries view
   - Right-click and choose Define Database Connection
2. Input sort in Prefix box and press to list fields and tables with input prefix

![VFA: SORT](image)

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Table</th>
<th>Field name</th>
<th>Type</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>SORT</td>
<td>DISASSM</td>
<td>SID</td>
<td>BIGINT</td>
<td>8</td>
</tr>
<tr>
<td>SORT</td>
<td>DISASSM</td>
<td>ADDRESS</td>
<td>BIGINT</td>
<td>8</td>
</tr>
<tr>
<td>SCRT</td>
<td>DISASSM</td>
<td>OFFSET</td>
<td>BIGINT</td>
<td>8</td>
</tr>
<tr>
<td>SCRT</td>
<td>DISASSM</td>
<td>BYTES</td>
<td>VARCHAR</td>
<td>255</td>
</tr>
<tr>
<td>SCRT</td>
<td>DISASSM</td>
<td>DISASSEMBLY</td>
<td>VARCHAR</td>
<td>1024</td>
</tr>
<tr>
<td>SCRT</td>
<td>DISASSM</td>
<td>SEQ</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>SCRT</td>
<td>DISASSM</td>
<td>OPCODE</td>
<td>VARCHAR</td>
<td>255</td>
</tr>
<tr>
<td>SCRT</td>
<td>DISASSM</td>
<td>OPERANDS</td>
<td>VARCHAR</td>
<td>1024</td>
</tr>
</tbody>
</table>

3. Choose lines and right-click to add select statement to query
4. Press to run this statement in database.
5. Double-click instruction item in new pop-up tab to view the profile this instruction belongs to

![Image of Visual Performance Analyzer interface]

5.1.5.2 Couple with Code Analyzer

Profile Analyzer can integrate with Code Analyzer for better navigation and comparison of module information between profiling file and executable file. This function can be initiated in generic hierarchy view. Then you can scroll both kind of information in Disassembly/Offsets view of Profile Analyzer and Instruction Table view of Code Analyzer at the same time. To couple with Code Analyzer, be sure to have both profile and binary file containing at least one same module. Please follow the steps below:

- Open a profile file.
Navigate generic hierarchy view, click a module and view its disassembly and offset information by pressing ENTER button.
Right-click this module symbol in generic hierarchy view. In popup menu, choose "Open in CodeAnalyzer".
Choose the corresponding binary file of this module.
Later, Code Analyzer Perspective opens automatically with this binary file. To scroll Profile Analyzer view with Code Analyzer view at the same time, be sure to open Disassembly/Offsets view. Now, when you select a table row in Disassembly/Offsets view, the instructions of this address will be highlighted in Instructions Table accordingly.
You may also trigger highlighting from Instructions Table of Code Analyzer.

5.1.5.3 View disassembly comparison

In this release, you can compare the offset, ticks and disassembly of two profiles in disassembly comparison view. You can open Disassembly Comparison view by choosing Windows -> Show View -> Other... -> Profile Analyzer -> Disassembly Comparison or just find it in the right bottom panes.

To view the disassembly comparison, you should follow these steps:

1. Compare two profiles through Profile Comparison Wizard or just open an .etc file.
2. Open Profile Comparison view by choosing Windows -> Show View -> Other... -> Profile Analyzer -> Profile Comparison
3. Right-click the line in Profile Comparison View and choose **Show disassembly comparison**

4. The corresponding disassembly comparison view opens.
There are two rows of hottest bars referring to two profiles which are compared. You can navigate disassembly comparison view by clicking them as follows:

To right-click and select menu item, you can sort columns by source line number or offset.
5.1.5.4 View offsets and disassembly

Profile Analyzer can disassemble the instruction stream for any symbol for which such a stream is available. Disassemble support is available for the following platforms:

- Intel IA32
- AMD-64 or EM64T (same instruction set)
- PowerPC
- Cell BE (both PPE and SPE)

Whether the profile contains an instruction stream is dependent on the profiling tools used to create it.

For JITCODE (JIT-compiled Java methods), instruction streams are available if the JPROF library was loaded with the JVM (using the -Xrunjprof option), the jints sub-option was specified as part of this option, and the log-jita2n* files produced were available at the time that mergetprof was run. Only the IBM Virtual Machine for Java supports the JPROF library.

When disassembly can be generated for a symbol, Profile Analyzer displays a table containing instruction addresses, the bytes for each instruction, the instruction sequence, and tick information. The following view shows the disassembly for a Java method on an Intel IA32 system:
If no disassembly is available, Profile Analyzer displays an Offsets view containing ticks for each offset. The following view shows the offsets for the NTOSKRNL.EXE module of the same profile; this module has a single symbol referred as NoSymbols for that symbol data (and by extension, instruction stream of a symbol) could not be obtained currently:
If you are expecting to see disassembly data for a symbol and instead see only offset data, check the following:

There should be a tprof.micro section (for static-compiled methods) or a log-jita2n section (for JIT-compiled methods) in the profile you have loaded.

- The appropriate section should contain instruction data. In the tprof.micro section, the symbol must have a sequence of lines beginning with C:; if no such lines exist in the tprof.micro section, the -off option may not have been specified in the POST options (if you were manually profiling). In the JITA2N section, after each symbol there should be a sequence of binary bytes or hex data.

- You may be able to view the disassembly by switching to the Disassembly view. Click on the pulldown menu at the top right of the view and ensure that the Show disassembly item is checked.

5.1.5.4.1 Navigating the Disassembly/Offsets View

You can quickly navigate to areas of high activity in this view using either the Hotness bar or a combination of sort and selection actions:

- **Navigate to hot areas by sorting and selecting**

  You can click on any column in the offsetAsm Information view to sort by that column. Repeated clicks on the same column reverse the previous sorting order. To navigate to hot areas in a symbol you can follow these steps:

  1. Click on a column heading that relates to CPU activity, to sort the view by that column (e.g. Ticks, %CPU activity, or a CPU counter if the profile contains CPU counter counts). The lines with the most events are sorted to the top.

  2. Select a line of interest; the top line should be the one with the most events in the column you selected.

  3. Click on the Offset column heading to sort by offsets again. The busy line you had selected in the previous step remains selected and remains in the viewable area.

If your platform supports symbol call resolution (currently only the x86 and x86-64 platforms, as these are the only platforms in which direct relative or absolute branch instructions are used to make calls to other symbols), you can also quickly find calls to resolved targets by sorting by the Remarks column. The following shows disassembly for a JIT-compiled Java method, sorted by the Remarks column so that lines containing call targets are displayed at the top:
You can use the same three-step sorting technique as for offsets, to find calls to a particular symbol:

4. Sort by the Remarks column. You may need to select the column header twice, if no call targets are visible the first time you select the column.

5. Select the line containing a call target of interest.

6. Sort by the offsets column. The line you had previously selected is now in the viewable area and instructions are displayed in offset order.

You can double-click on a line containing a call target to switch the current symbol in the offsetAsm Information view to the target symbol.

**Loop nest detection and branch target detection**

When Profile Analyzer loads the disassembly for a symbol it analyzes internal direct branches in the disassembly to determine loop patterns. Any backward branch may be considered the end of a loop, provided certain other parameters are met. Any block of code detected to be within a loop is indented by one space; if multiple nested loops are detected, sections of code may appear more deeply indented. In some cases the level of indentation may be extreme, as in the following example:
Here the indentation shows at least 15 levels of nesting. While it is unlikely that a programmer would have written a loop nest 15 layers deep, this level of nesting may occur where a compiler has inlined calls that occur within loops, and the inlined calls themselves contain other nested loops or further inlined calls.

You can remove loop nest indenting by clicking on the icon at the top of the view. If the icon is displayed, clicking on it will display loop nest indenting for a symbol whose disassembly was not indented.

**Reordering columns**

You can reorder the columns of the Disassembly/Offsets view to hide or show particular columns or change the order in which columns are displayed. This is one of the features of eclipse 3.1. You can reorder the column just by clicking on the column name and dragging it to the place where you want it to be. It shows as follows:

<table>
<thead>
<tr>
<th>Offset</th>
<th>Disassembly</th>
<th>Bytes</th>
<th>Ticks</th>
</tr>
</thead>
<tbody>
<tr>
<td>0108002FD0</td>
<td>ADD EDX, EBX</td>
<td>031F</td>
<td>3.37 11</td>
</tr>
<tr>
<td>0108002FD6</td>
<td>JEE LF8$4000000</td>
<td>0F86H</td>
<td>6.64 22</td>
</tr>
<tr>
<td>0108002FD0</td>
<td>SUB EDX, EBX</td>
<td>2F0A</td>
<td>24.7 02</td>
</tr>
<tr>
<td>0108002FD4</td>
<td>XCH EDX, EBX</td>
<td>22DB</td>
<td></td>
</tr>
<tr>
<td>0108002FD3</td>
<td>CMW ESP, DWORD PTR [ESP+18H]</td>
<td>E606</td>
<td></td>
</tr>
<tr>
<td>0108002FD9</td>
<td>PUSH EAX</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>0108002FEC</td>
<td>PUSH EAX</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>0108002FDD</td>
<td>PUSH EDX</td>
<td>53</td>
<td></td>
</tr>
<tr>
<td>0108002FDF</td>
<td>POP EDX</td>
<td>6B</td>
<td></td>
</tr>
<tr>
<td>0108002FE2</td>
<td>IMUL EDX, EBX, 3RGB</td>
<td>6D288030000</td>
<td></td>
</tr>
<tr>
<td>0108002FE3</td>
<td>PUSH EAH</td>
<td>6A28</td>
<td></td>
</tr>
<tr>
<td>0108002FE4</td>
<td>PUSH EAH</td>
<td>6A28</td>
<td></td>
</tr>
<tr>
<td>0108002FEC</td>
<td>JE LF8$1B00H</td>
<td>7407</td>
<td></td>
</tr>
</tbody>
</table>

**Branch and call navigation**

Any disassembly line that contains a branch to a known target within the current symbol, or a call to another symbol, is indicated by an arrow in the left margin.

- \( \Rightarrow \) denotes a forward branch, one whose target is a subsequent instruction.
- \( \Rightarrow \) denotes a backward branch, one whose target is a previous instruction.
- \( \rightarrow \) denotes a call or branch to another profiled symbol. This is only available for x86 and x86-64 platforms.

When you double-click on a line containing one of these icons, the view changes to show the target of the branch (a different location in the current symbol, or the target symbol of a call).

To navigate back to the last in-symbol branch you selected, after you have followed that branch, press the button on the offsetAsm Information view toolbar.

**The hotness bar**

By clicking on an area in the hotness bar, you will be taken to the corresponding disassembly instructions, or offsets. For lengthy disassembled methods, you may need to page up or down to find the hot area in question, as a line in the hotness bar that is one pixel high may relate to several pages of disassembly.

When you select a line in the Disassembly/Offset Information view, a yellow square appears in the hotness bar to show the currently selected area of the symbol.
5.1.5.5 View source code

When an executable or library has been compiled with line number information (for example the -g option on some compilers), the platform profiler, like Tprof on AIX, may be able to obtain line number information for profiled symbols in such an executable or library. You can then view source code for these symbols within Profile Analyzer.

Line number support is available when the TPROF post-processing command includes the -off option. This option is enabled by default when you use the run.tprof_e script or run the profiling session from the Profiling Configurations view.

When you first select a symbol for which line numbers are available from the Symbols view, a dialog is displayed to ask whether you want to view source code for the symbol:

If you choose Yes, a File dialog is displayed that lets you navigate to the path containing the file. The name of the file you choose from this dialog does not have to match the name in the TPROF output, but if the line numbers do not match those of the file from which the code was compiled (for example, if the file has been edited since it was compiled), the tick information may not map to the correct source line numbers.

If you choose No, you are not prompted to enter source for any other symbols in the current profile, but when you load a different profile containing line number information, you may again be prompted to locate source files. If you choose Don't ask me again, you will not be asked to open a source file until you exit and restart Profile Analyzer.

The following view shows source code for a symbol:
If you choose No or Don't ask me again, no source code will be shown. The source code view shows as follows:

You can again associate source file by pressing the button in the center of the view, or click Associate Source File icon on the toolbar of the view.

When you click on different areas of the hotness bar in the right side of the source code view, the corresponding line in the source file you select will be highlighted. You can see it in the following view:
You can export these source codes to file by selecting Export to Files... in the menu of the toolbar as follows:

5.1.5.6 View temporal profiling

When a Profile Analyzer profile contains a trace buffer section, Profile Analyzer attributes buffer events to appropriate symbol offsets, symbols, modules, threads, and processes. When you select a profile object (click on a process, thread, or module in the process tree, double-click on a symbol in the symbol list, or double-click on a disassembly or offset line with tick information in the offsetAsm Information view), Profile Analyzer can display a temporal graph showing when during the profile run the ticks for that object occurred. This version of the Temporal
profiling view is called **Tick intensity over time**. The following screen capture shows the Temporal Profiling view for a java.exe process in a profile that ran for about 7.5 seconds:

You can change the number of intervals of a temporal profile by sliding the intervals slider (on the left) to the left or right. This changes the number of intervals used to display the temporal graph. Changing the granularity of a 25-second trace run to 50 intervals will result in each bar showing the events for a particular half-second. For the same trace run, a granularity of 10 would have 2.5 seconds attributed to each bar. The following screen capture shows the same information as above, but with a granularity of 8 (that is, 8 equal-time intervals):

When the selected object is a Profile, Process, Thread, or Module, up to six of the "natural" children of that object are shown in a line graph superimposed on the bar graph, as can be seen in the above images. A child is only shown if it is a significant contributor to the parent tick count. You can hide the tick information for a child by deselecting the check box beside its name below the bar graph. The following screen capture shows the same profile, with only the top two children selected:
The "natural child" of a profile object is as follows:

<table>
<thead>
<tr>
<th>Parent</th>
<th>Natural children</th>
</tr>
</thead>
<tbody>
<tr>
<td>Profile</td>
<td>Processes</td>
</tr>
</tbody>
</table>
| Process | • Modules, if "Ignore thread data" in the process tree checkbox is selected  
          • Threads, if "Ignore thread data" checkbox is not selected |
| Thread | Modules |
| Module | Symbols |
| Symbol | No children |
| Offset tick | No children |

5.1.5.6.1 Changing the tick scale for children

In some profiles, it may be hard to distinguish the lines of the "natural children" of a particular profile object. You can slide the Zoom children vertically slide bar to exaggerate or diminish the scale for the line graphs that are superimposed on the bar graph. The following shows the view immediately above, with all children selected and their vertical scale exaggerated by a factor to 400% of the parent object scale. The lines for the highest-contributing children extend off the top edge of the chart, but the difference in relative contribution of the three lesser modules is easier to distinguish because their lines are further apart.
5.1.5.6.2 Zooming in on a time range

To zoom on a particular time range, press and hold mouse button 1 at one end of the time range, and drag left or right. A rectangle shows the time range selected (the vertical dimensions of the rectangle are not relevant to the result of the selection). The following shows the initial selection of a time range within a profile:

When you release mouse button 1 the selected area is zoomed:
You can drag a zoomed view forward or backward in time by holding down mouse button 3 (the right mouse button for left-handed users) and dragging. The following animated graphic shows this effect:

Note that the chart bars and lines move up and down as you drag: this is because, each time Profile Analyzer handles an increment of the drag event, it redistributes the trace events according to the current co-ordinates.

You can further zoom in an already zoomed image by selecting a new zoom area:
This zooms to the time range 5.76 to 9.10 seconds:

To restore the time view to the full duration of the profile, right-click over the bar chart.

5.1.5.6.3 Time cursor

From VPA 5.0 on, temporal profiling view is decorated with two time cursors, one for start time, and the other for end time. Temporal profiling view displays a column chart of time intervals of all the profile data. Time is labeled along below the column chart, once every some time intervals. The time cursor helps user accurately position the time line of the column chart. When user selects the cursor and drags it along the column line, cursor time is displayed as it is moved. At first, the start cursor is at the bottom left and the end cursor is at the top right.
It is the temporal profiling view for an .etm profile data file with time data. The two cursors are displayed at the initial positions in the column chart.

When user drags the cursors, time line is displayed and time data is changed as the cursor is moved. Here the start cursor is labeled 11.26, and the end cursor is dragged to 23.78, in second.
Time cursor helps user focus on a range within a time interval, and helps user position the mouse to select a concerned range of time interval. In the above graph, the cursors having displayed a range of time interval, if user is interested in the interval of profile data, he can drag a rectangle aligned with the cursors. It helps user ranged area of time intervals more accurately.

When user selects a time interval range with mouse, temporal profiling view refreshes the graph, and displays new column chart of profile data inside selected time interval. The above chart is resulted from selecting time intervals range from 8.91 second and 32.75 second. The start cursor is at the left most and the end cursor is at the right most. In this graph time cursor is able to move its position too.

5.1.5.6.4 Time/memory profiling

The temporal profiling view also lets you view how tick events for a selected object are distributed within a matrix of memory and time. This is mainly of interest to compiler writers, or other specialists concerned with how well busy sections of a symbol or module are distributed within a processor’s instruction cache. To switch to the
Time/memory profiling view select the **Show memory usage over time** icon. Select a profile object for which this view makes sense - typically a symbol or module. (It is not normally productive to view the entire profile or a particular process in this view, because individual libraries within the profile or even a single process may occupy widely different memory ranges.) The following shows a time/memory view of the JITCODE module of a java process (the module containing JIT-compiled Java methods):

Individual colored rectangles in this view represent profile intensity, for a given time and a given address range. To produce this view, Profile Analyzer divides the profile ticks for the selected object (profile, process, module or symbol) into equal time intervals (determined by the left-hand or time interval slider, as for the **Tick intensity over time** view) and divides the memory range of the selected object into equal memory intervals (determined by the right-hand or memory interval slider). Individual rectangles that contain ticks represent a region of memory that was busy at a particular time. If there is sufficient space, Profile Analyzer displays the tick count for each active intersection within the rectangle. If you increase the number of intervals the ticks may disappear but the color scheme still gives an indication of which memory ranges are busy at what times, with darker shades denoting higher tick counts:
The table at the bottom of the view displays information about the active objects in a particular rectangle in the view. To use this capability, make sure the Temporal Profiling view is the active view (click on the tab), then **hold down the Shift key** while moving the mouse. As the pointer moves over different areas of the graph the objects that occupy the memory range for the current area are shown in the table. Two tick counts are shown for each object: **In range** identifies the number of ticks the indicated object contributes to the current time/space interval, while **Total** represents the number of ticks the indicated object contributes to the profile as a whole. In the following view, the Shift key is being held down and the pointer is over the rectangle with a tick count of 426. The busiest symbols for that time/memory range are displayed at the top of the table:
Once you release the Shift key, the table contents do not change. This allows you to use the mouse to scroll through the table after you have chosen a particular rectangle, without mouse movements changing what is displayed in the table.

### 5.1.6.5 How to use the time/memory view

The time/memory view below is for a static compiled function within the garbage collection module of the J9 virtual machine for Java:
In this view, each row represents 69 bytes (as shown by the legend beside the right-hand slider bar). You can see that there are two fairly busy ranges: the first range consisting of the top two rows, and the second range consisting of the fifth row (the row whose first displayed value is 21 ticks).

One use of the time/memory view is to show whether code is properly ordered within busy symbols. For instance, the above function might provide better performance if the areas of code that are busy were closer together in memory, as they would likely use fewer l-cache lines if grouped together. Note that you should only attempt code reordering based on the time/memory view after analyzing the same symbol in several profiles, and there are no guarantees that your reordering will yield improvements. For example, compilers may completely reorder sections of your code when they generate the machine code for a symbol. However, this view may help you identify symbols or modules with time/memory usage patterns that warrant further investigation.

5.1.6 Configure database connections and manage cached database files

To improve the performance, Profile Analyzer will load its files into a database, either hsqldb or DB2, instead of keeping them in the memory, sometimes in page files. Then, each action just executes a query to get the data needed without any useless data.

To open this view, click Window - Show View - Other, then, select Database Connections under Visual Performance Analyzer category, as below.
When you start VPA for the first time, a default connection of hsqlbd will be created, as well as product supports under this connection node, as below.

Default Connection and Product Supports

The only Profile Analyzer support is set as active, so that user can open files without any setting actions. However, you can create other connection or edit this default connection as you like, such as modifying its path.
Choose Database
Choose database to create connection.

- HsqlDB
- IBM DB2

New Hsql Connection
Set the properties for HsqlDB connection.

Connection Name: My HsqlDB
Database Path: c:\hsqldb

Create HsqlDB Connection
After these two connections have been created, the view looks as below.
User can create Profile Analyzer support under each connection, for common use, hsqldb is enough, for performance consideration, db2 is the better choice.

Hsqldb, as we known, is an embedded database system, so it must have some limitations. To prevent too much disk space occupied as more and more files being loaded in, you should set a size limitation. This num of Size Limitation stands for the upper limit of the hsqldb's data file. If the database file was larger than the num, system will delete oldest files, until the size of the database file is smaller than the size limitation. During the auto delete process, some file whose size is larger than the size limitation will be deleted first.

Set Hsqldb's size limitation for Profile Analyzer Support

You can also delete files manually to release disk space. Multi-selecting is allowed. See the picture below.
To set one Profile Analyzer support as active, you should switch the perspective to Profile Analyzer Perspective, so that the related menu named “Set Active” will occur, click it to set the connection you like as active.

Active Profile Analyzer Support cannot be deleted. To delete it, you should set other support as active first. Connection cannot be deleted, until you delete all the product supports under this connection.

This view support sorting operation. You can sort the files by name, size or date. To sort, select the sort mode on the action bar, or click the title directly.
5.2 Code Analyzer

You can also find the Code Analyzer User Guide from within VPA. Select Help - Help Contents within VPA. To get context sensitive help, press F1 for Windows and AIX or press Ctrl+F1 for Linux.

5.2.1 Load an executable for analysis

When you first start Visual Performance Analyzer after installation, what you see is the Welcome View. To open CodeAnalyzer, choose Windows -> Show Perspective -> Other -> CodeAnalyzer.

The following screen capture shows the default workbench window of CodeAnalyzer.

Choose File -> CodeAnalyzer -> Analyze Executable or the toolbar icon.
In the pop-up wizard, select the executable you want to analyze.

If you press **Open**, the executable will be loaded in CodeAnalyzer. The advanced section allows passing a configuration file through which the CSects/functions that will be analyzed can be filtered.
When an executable is loaded, a wizard for further action appears as follows that allows you to load profile information.

To enhance the views with profile information for the loaded executable, you can either an instrumentation profile file or a sampling profile file in the above wizard. After pressing **OK**, the profile information will be added into CodeAnalyzer workbench window.
5.2.2 Adding profiling information

If you have been working with an executable, without profiling information, you can add the profiling information by choosing File -> CodeAnalyzer -> Add Profile Information, or from the toolbar icon. After pressing OK, the profile information will be added into CodeAnalyzer workbench window.

A profile can also be removed. To do this, choose File -> CodeAnalyzer -> Remove Profile Information or the toolbar icon.

The following screen capture shows the workbench window after profile information is loaded.
5.2.3 Navigate the Executable

5.2.3.1 Navigate the Program Tree

The Program tree displays a hierarchical view of the loaded executable. It is automatically opened in the left side of the workbench window when you first use CodeAnalyzer. You can open program tree view by choosing Windows -> Show View -> Program Tree. First, you need to load an executable. Then navigate program tree by expanding all, collapsing all, sorting, drilling into items, opening source code and viewing control flow.

5.2.3.1.1 Expand All

To see all functions under each file in the loaded executable, press the button in title bar. The resulting view of program tree expands as follows:
5.2.3.1.2 Collapse All

To close all sub-items of each file in the executable, press the button in title bar. The result view of program tree shows as follows:
5.2.1.3 Sort

There are three kinds of sorting in program tree view: lexicographical order, ascending order and descending order. You can press their corresponding buttons in the view’s title bar or right-click any object in the view and choose these items. If you want to exit a special sort, you need to click the selected button one more time.

5.2.1.4 Drill Into

If you want to get detailed information of the items which are under certain object, you can select this object and press . The program tree will display all its sub-items and only them. If you want to go to upper level, just press . Pressing can lead you to root view.

For example, if you want to see all the functions in file os.c, choose os.c file in program tree as follows:
The result view displays all the functions within this file.
You can also navigate forward or backward as you like.

5.2.3.1.5 Open source code

To open source code, right-click any object in program tree and select **Open Source Code**. Source code can be opened from the instructions view/editor as well. Opening from the tree will just locate the method, whereas opening from a selected assembly instruction will map to its exact source line.

If selected element has source file and line number information, the user will be prompted for a source location. After specifying the source directory appropriate source file will be opened in the source view. Source view has the title on top of it, saying which source file is currently opened, as an executable may be compiled from a few source files. In order to make source view synchronized with the rest of the Code Analyzer views, i.e. to display the appropriate source file and mark the source range for the current selection automatically, Link with Table button from the view tool bar can be used. As long as it pushed, source view is listening to the current selection and its content changed appropriately. If there is no source code for current selection (for instance linker code is selected), the view will display info message.
5.2.3.1.6 Get Control Flow

To get calling functions and called functions, select a function in the program tree, right-click and choose Control Flow.
The called functions of the selected function in the above screen capture show as follows:

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Address</th>
<th>File Name</th>
<th>Executes</th>
</tr>
</thead>
<tbody>
<tr>
<td>.xjumpon</td>
<td>0x100000e8</td>
<td>xjumpon.c</td>
<td>0</td>
</tr>
<tr>
<td>.xlabprint</td>
<td>0x100000ed</td>
<td>xldbug.c</td>
<td>0</td>
</tr>
</tbody>
</table>

The calling functions of the selected function in the above screen capture show as follows:
5.2.3.2 View Static Color Bar

Static color bar gives an overview of frequency distribution of basic blocks in the loaded executable. You can open this view by choosing Windows -> Show View -> Static Color Bar.

To obtain this information, do following steps:

1. Load an executable
2. Add profile information

After profile information is added, there is a yellow pointer in the static color bar. It indicates the position of basic block you have selected. The yellow pointer scrolls in accordance with the basic block you select in the Instructions Table or Program Tree.

The following screen capture shows the selection of a specific color bar.
5.2.3.3 Navigate Instructions View

Instructions view is the default view of Instructions Table. It shows the contents of an executable or shared object as a table of assembly instructions, with its control flow graph drawn vertically at its side. To open this view, select Switch to instructions in the Instructions Table's title bar. The explanation of functions of some buttons in title bar can be found in Instructions Table.

In instructions view, when a new file begins, there is a line of table beginning with icon and followed by its name. When a new function starts, there is also declaration in a separate line. Instructions which belong to the same basic block are organized together with blank lines between them.
After adding FDPR-Pro profile information, you can get the frequency of each instruction in color. The color legend can be found in the lower part of workbench window. Instruction groups are indicated in the front of each instruction. In **comment** column, there are red triangles which contain performance comments to specific instructions.
If you right-click any line item in Instruction View, a menu list appears. Typically, there are four sections of this menu. In the first section, you can get PPC (Power PC) assembly reference help, branch profile information, dispatch information, and set points to collect the value of important resources during profiling. In the second section, you can choose to find specific instruction or open source code of the selected instruction. In the third section, the menu will show the target of the basic block (Fall thru means the next basic block) to which the selected instruction belongs. The last section shows the callers of the basic block to which the selected instruction belongs. The first and the last two sections may vary according to the selected instruction.

5.2.3.3.1 Overlay information for Cell SPE binaries

Cell has a very limited memory size. Therefore, the code in SPE binaries may need to be split into a number of overlays. This code should conceptually be displayed in parallel as only one of the overlays in each region may be in memory at a given time.

To facilitate this in CodeAnalyzer, a dialog window is displayed to the user in which he may choose which overlay to display for each region.

Below are screenshots showing such an overlay dialog and how the displayed code changes after the user chooses a different overlay.
Figure 4- The overlay of function sc in region 1
Figure 5 - The overlay of function sg in region 1
5.2.3.3.2 Get PPC help

To get the assembly reference of certain instruction

If a PPC file is open, right-click on the instruction and choose Show PPC Help. You can also select the instruction and press `PPT` button in title bar.

5.2.3.3.3 Show Dispatch Information
To get the dispatch information of certain instruction, right-click this instruction and choose **Show Dispatch Info**. You can also select this instruction and press the button in title bar. To obtain more detailed description, please refer to View dispatch information.

### 5.2.3.3.4 Show Branch Information

To get the branch information of certain basic block, right-click the last instruction of this block and choose **Show Branch Info**. You can also select the last instruction of this basic block and press the button in title bar. To obtain more detailed description, please refer to View branch profile.

### 5.2.3.3.5 Collect Value Profiling

Before profile information of the loaded executable is added, you can collect value of certain resources of specific instructions. To get this information, you need to first select these instructions by right-clicking an instruction and choosing **Collect Valuing Profiling**. A wizard will appear for you to choose resources value you try to get. You can also do this by pressing the button in title bar. To obtain more detailed information, please refer to View value profile.

### 5.2.3.3.6 Find Instructions

To find specific instructions in this view, press button in title bar. You can also open it by right-clicking any instruction and choosing **Find**.

The following screen capture shows the initial activating of Find wizard.

A Find Instruction wizard appears.
The default instruction to be sought is that of the selected instruction. The initial scope of this wizard is set between the start address and the end address of the loaded executable or shared object. You may change the scope as you like, but it should not exceed the boundary. To choose the direction of searching, select Forward or Backward. Error and information messages are displayed in the bottom of the dialogue.

5.2.3.3.7 Open Source Code

To get the source code of the selected instruction, right-click this instruction and choose Open Source Code. You can also use the button in title bar. For more information, please refer to View source code.

5.2.3.3.8 Navigate control flow graph

Right click the first instruction in this basic block. The menu list shows its callers, their addresses and functions name which they belong to. The caller is defined to be the first instruction of basic block which calls the selected instruction.
To get target basic blocks of the selected instruction, right-click the last instruction of a basic block. The menu list will show its target basic blocks, with addresses of their first instructions and functions name which they belong to. If its target is next basic block, it shows Fall thru only.
The screen capture above shows that the selected basic block has two target basic blocks. One is the one below, and the other is in the function `main`. You can jump to these basic blocks simply by clicking it. You can verify this relationship by referring to the graph beside it.

5.2.3.3.9 Highlight a control flow graph edge

It is possible to highlight an edge in the control flow graph to make it easier to follow its course. To do this, right-click on the graph column (edge) you want to follow and press highlight. See the screenshot below:
After pressing the highlight action the selected edge will be colored in green and placed on the leftmost column.
See the screenshot below:

```
0x1004734  0x2c030000  cmpi cr0, 0x0, r3, 0  973566
0x1004738  0x41b20044  beq cr0, 0x1000477c  973566

<table>
<thead>
<tr>
<th>BASIC BLOCK</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x100473c</td>
</tr>
<tr>
<td>0x1004740</td>
</tr>
<tr>
<td>0x1004744</td>
</tr>
<tr>
<td>0x1004748</td>
</tr>
<tr>
<td>0x100474c</td>
</tr>
<tr>
<td>0x1004750</td>
</tr>
<tr>
<td>0x1004754</td>
</tr>
<tr>
<td>0x1004758</td>
</tr>
<tr>
<td>0x100475c</td>
</tr>
<tr>
<td>0x1004760</td>
</tr>
<tr>
<td>0x1004764</td>
</tr>
</tbody>
</table>
```

To remove the highlight, right-click on the selected column and press the ‘Remove highlight’ action.

### 5.2.3.10 Navigate along with program tree

In the program tree on the left side of the workbench window, a hierarchical organization of loaded executable or shared object is displayed. You can navigate along Instructions Table by selecting objects in program tree, or vice versa.

For example, if you select the first file in the program tree, the first instruction of this file will be highlighted in instructions view, the first basic block of this file will be highlighted in blocks view and the first function of this file will be highlighted in functions view.

The following screen capture shows the selection of the first file in Program Tree.
The selected instructions in instructions view changes correspondingly.

### 5.2.3.4 Navigate the Blocks View

Blocks View shows the detailed information of loaded executable or shared object in form of basic blocks. To open this view, select **Switch to blocks** in the Instructions Table's title bar. The explanation of functions of some buttons in title bar can be found in Instructions Table. In blocks view, there is an icon in the front of the first basic block of a file in loaded executable or shared object. When a new function begins, an icon appears in the front of its first basic block.

The following screen capture shows blocks view of a loaded executable.
After adding FDPR-Pro profile information, you can get the frequency of each basic block in color. The denotation of each kind of color can be found in the lower part of the window.
If you right-click a basic block, a menu list will appear. Typically, there are three sections in this menu. In the first section, you can search for specific basic block or open source code of the selected basic block. In the second section, the menu shows the target of this basic block (Fall thru means the next basic block). The last section gives information of all the callers of this basic block. The last two sections may be varied according to the selected basic block.

**5.2.3.4.1 Find basic blocks**

To find specific basic blocks in this view, press button in title bar. You can also open it by right-clicking any basic block and choosing Find.

The following screen capture shows the initial activating of Find wizard.
A Find BB wizard appears.
The default basic block to be sought is that of the selected one. The initial scope of this wizard is set between the start address and the end address of the loaded executable or shared object. You may change the scope as you like, but it should not exceed the boundary. To choose the direction of searching, select **Forward** or **Backward**. Error and information messages are displayed in the bottom of the dialogue.

### 5.2.3.4.2 Open Source Code

To get the source code of the selected basic block, right-click this basic block and choose **Open Source Code**. You can also use the **button in title bar. For more information, please refer to View source code.

### 5.2.3.4.3 Go to callers and callees

Right click any basic block in this view. The menu list shows all its callers and their addresses. The address of caller is defined to be that of the first instruction in the basic block which calls the selected basic block. The menu list also displays the information of its target basic blocks, along with addresses of their first instructions and functions name which they belong to. If its target is next basic block, it shows **Fall thru** only. By choosing its callees or callers, you can jump directly to these basic blocks.

In the screen capture below, the selected basic block has two callees and one caller.
You can verify these relationships by referring to graph column on the right side.

5.2.3.5 Navigate Source Code View

Source code view displays source code of the selected object in loaded executable.

The following screen capture shows a typical source code view.
Source View features:

- **Code annotations**
  If some hazard comments were collected, each comment will be mapped to specific line of a source file if possible, using line number information FDPR can maintain. All the source lines having comments will be marked by hazard sign, which will appear on the most left ruler of the source viewer (vertical bar along the source page). Pressing on one of these image annotations on the annotations ruler opens list of the addresses that have comments and are mapped to the current source line. The user can choose one of the addresses from the list and clicking on it will lead him to this address in the instructions table. The same list for the current line can be obtained by pressing Ctrl+1 (similarly to the way Java Quick Assist works).
The other ruler from the right (called overview ruler) also displays comments annotations for current source file, but in compact form. I.e. if annotations ruler has hazard image near each line that has comment annotation, and therefore contains only annotations for the currently visible part of the code, overview ruler makes viewing all the annotations for the current file possible. Pressing on one of the annotations in overview ruler will select the source line containing this annotation.

If the number of comments of specific type is above the threshold, such comment annotations will appear in yellow △, but will not be displayed by default, in order to prevent source view to become too loaded with details. The user is able to control what kind of annotations will be displayed by Filter Annotations button on the source view tool bar.

• **Profile colors**
  If profile was loaded for the current exe file, it's possible to view the profile color each source line gets (calculating by the hottest instruction that mapped to this line). Profile ruler serves this purpose. This is the second ruler from the left and it can be shown or hidden by pressing the Show Profile Colors button on the source view tool bar.

• **Line numbers**
  Another action from the tool bar, marked by "1...n" is for showing source line numbers. If this button pressed, another ruler will appear next to profile ruler, showing near each source line its number.

• **Folding code segments**
  The most inner ruler from the left may have .expand and .fold images along the code lines. It enables collapsing functions body and multi-line comments to one line and expanding it back for more convenient code viewing.
The following buttons in toolbar can change the appearance of source code view.

- Choose displaying profiling color bar or not.

- Choose displaying line number of not.

User also can click the filter button to select comment annotation to display in the following dialog.
In order to go to the instruction of the collected comments, click the comment ▲ and select one.

```c
74    75    /* x1getline - get the value of a symbol (checked) */
76    NODE *x1getline(NODE *sym)
77    {
78    register NODE *val;
79    while ((val = x1getline(sym)) == s_unbound) {
```

5.2.3.6 Navigate Functions View

Functions View shows the detailed information of loaded executable or shared object in form of functions. To open this view, select Switch to functions in the Instructions Table's title bar. The explanation of functions of some buttons in title bar can be found in Instructions Table. In functions view, there is an icon ▲ in the front of the first function of a file in loaded executable or shared object.

The following screen capture shows functions view of a loaded executable.
After adding FDPR-Pro profile information, you can get the frequency of each function in color. The denotation of each kind of color can be found in the lower part of workbench window.
If you right-click a function, a menu list will appear. Typically, there are three sections of this menu. In the first section, you can choose to find functions or open source code of the selected function. In the second section, the menu shows the target functions of this one. The third section displays the callers of this function. The last two sections may be varied according to the selected function.

### 5.2.3.6.1 Find functions

To find specific functions in this view, press button 🕵️ in title bar. You can also open it by right-clicking any function and choosing **Find**.

The following screen capture shows the initial activating of Find wizard.
A Find Function wizard appears.
The default function to be sought is that of the selected one. The initial scope of this wizard is set between the start address and the end address of the loaded executable or shared object. You may change the scope as you like, but it should not exceed the boundary. To choose the direction of searching, select Forward or Backward. Error and information messages are displayed in the bottom of the dialogue.

5.2.3.6.2 Open Source Code

To get the source code of the selected function, right-click this function and choose Open Source Code. You can also use the button in title bar. For more information, please refer to View source code.

5.2.3.6.3 Go to callers and callees

Right click any function in this view. The menu list shows all its callers and their addresses. The address of caller is defined to be that of the first instruction in the basic block which calls the selected function. The menu list also displays the information of its target basic blocks, along with addresses of their first instructions and functions name which they belong to. If its target is next basic block, it shows Fall thru only. By choosing its callees or callers, you can jump directly to these functions.

The following screen capture below, the selected function has one callee and six callers.

You can verify these relationships by referring to graph column on the right side of view.

5.2.4 Instruction Properties Analysis

The following tasks allow you to analyze instruction properties within Code Analyzer:
5.2.4.1 View Branch Profile

Branch Profile table shows the detailed information of targets of the instruction in the end of an instruction group. This information is available only after loading a profile file.

To get the information, please follow these steps: load an executable, add profiling information, be sure to open the Instructions Table, set instructions view in the Instructions Table, select the last instruction of an instruction group and right-click it and then choose Show Branch Profile.

You can also press the button in the Instructions Table's title bar.
Then branch profile tab of instruction properties will appear in the right bottom of the workbench window. It displays the addresses (including function's name) and counts of the target basic blocks of the selected instruction group.

To simultaneously display branch profile information while scrolling along Instructions Table, press in Instruction Properties view's title bar. If you select an instruction within an instruction group, branch profile will show following information:
5.2.4.2 View Dispatch Information

In Power5 or Power6 architecture, instructions are tracked in groups of one to five instructions rather than as individual instructions. Groups are formed that contain up to five internal instructions, each occupying an internal instruction slot (numbered 0 through 4). Each internal instruction slot in a group feeds separate issue queues for the floating-point units, the branch execution unit, the CR execution unit, the logical CR execution unit, the fixed-point execution units and the load/store execution units. With profile information, CodeAnalyzer can display this information in Dispatch information tab in Instruction Properties view.

In Cell, both for PPE and SPE code the instructions are tracked in groups of up to two instructions. There are two pipes; the odd and even pipes. Each instruction can be placed in either the odd or the even pipe. There are no instructions that may be placed in both pipes. Each instruction uses one or more resources from its pipe (called slot in PPC).

To get dispatch information of an executable, please follow these steps: Load an executable, Add the profiling information, open the Instruction Table, Set the instructions view and press in the Code Analyzer toolbar.

Select the kind of Power architecture your executable is run on in the following wizard.

In the Instruction Table view, select an instruction, right-click and choose Show Dispatch Info. The architecture you have chosen in the previous step will display alongside the menu item.
5.2.4.3 View Value Profile

Value Profile is used to show the resources, their values and counts of specific instruction in the loaded executable. To collect these values, please follow these steps: load an executable, open the Instructions Table view and set instructions view and then set to collect resources of some instruction by right-clicking each of them and choose Collect Value Profiling.
Choose resources and their types in the following wizard.

Press **Ok**. The following screen capture shows an icon may appear on the selected instructions after the above steps.
Press 📹 in the CodeAnalyzer toolbar to run instrumentation.

Press 📥 to write the instrumented file to disk. If you are running on a windows, please make sure to set the output of profile-file value to ./<filename>

Upload the instrumented executable and the profile file you have created to an AIX machine. Make sure to put them under the same directory.

Run the instrumented executable with some training data. It will write information of collected value to your original profile file. You can get relevant training data from SPEC2000.

Copy the profile file created back to the windows.

Load the original executable in CodeAnalyzer again.

Add profile file you've created in --profile--file.

The following screen capture shows that the original grey icon may change to green icon in the front of the selected instruction after doing the above steps.
To view the value of collected resources, right-click the marked instruction and choose **Show Value Profile**. The resources value will be displayed in Value Profile table. You can also press in Instructions Table view's title bar.

To simultaneously display value profile information while scrolling along Instructions Table, press in Instruction Properties view's title bar. If you select an instruction without green icon, value profile will show as follows:
5.2.4.4 Collect Comments

CodeAnalyzer can display comments generated by FDPR-Pro engine. Different type of files have different comments for collection. So far there are three sources of comments: Power 5, Power 6 and general comments. All the general comments are dependent on profile information. To view those comments, you need to collect comments first. Please follow the steps below:

Load an executable

Add profile information of this executable

Be sure to open Instructions Table

Select File - > Code Analyzer - > Collect Hazard Info or press button to choose type of comments to collect

Press button to open Comments View

Press button or in tool bar to navigate each comment in Instruction table

Press button to display the currently collected comments statistics

Please notice that if you have restricted grouping to some architecture, the comments view may remove inappropriate comments from it.
5.2.5 Statistic Analysis

Code Analyzer provides users with a graphical display of statistics gathered on the loaded executable based on different degrees of granularity. There are three perspectives of analysis: file, function and instruction mix. They are shown in the form of tab in Statistics view. All the graphs are drawn to scale in cylinder. Each column is colored according to its frequency heat. The Statistics view normally displays top (hottest) files, functions or instructions. In the upper level of each tab, there is filter button. Only items that pass the average threshold you set in filter value will be displayed. Therefore, if you enter 0% and press refresh, the view will show all the files. And if you enter 100% and press refresh, a single column will be displayed, representing the hottest execution unit.

To open these views, you need to load an executable and add profile information first. Then press the corresponding buttons in Code Analyzer’s toolbar. You can also open it from File -> Code Analyzer -> Statistics.

When your cursor stops in a column in the graph for a while, there is reference information like this:

![File Heat Graph](image)

5.2.5.1 View File Heat Graph

To open file heat graph, press button in CodeAnalyzer toolbar or choose File -> CodeAnalyzer -> Statistics -> Files Heat. You can also select the tab within Statistics view.

After that, load an executable and add necessary profile information.

The following screen capture shows a typical file heat graph of loaded executable.
The x-ordinate shows the names of files in the loaded executable. The y-ordinate denotes execute count distribution. Each column of the graph refers to a file in the loaded executable. Their color shows how frequently they are called. You can set graph options to customize the view. There are three methods in the average method box: simple average, weighted average and highest function value. By inputting the minimum percentage in the latter box and press Refresh, you can get the count distribution of those files whose percentages calculated by average method are above this value.

For example, if you use the highest function value as average method and try to filter 25% files, the graph options should be set as follows:

```
Graph Options
Average Method: Highest Function Value
Filter files under (%): 25
```

Press Refresh. Then the files whose highest function value percentages are at least 25% of the maximum value will be listed in the graph as follows:
5.2.5.2 View Function Heat Graph

To get function heat graph, you can press button in CodeAnalyzer toolbar or choose File -> CodeAnalyzer -> Statistics -> Functions Heat. You can also select the tab in Statistics view.

After that, load an executable and add necessary profile information.

The following screen capture shows a typical function heat graph of loaded executable.
The x-ordinate shows the names of functions in the loaded executable. The y-ordinate denotes execute count distribution. Each column of the graph refers to a function in the loaded executable. Their color shows how frequently they are called. You can set graph options to customize the view. There are four average methods: simple average, weighted average, highest BB value and prolog value. By inputting the minimum percentage in the latter box and press Refresh, you can get the count distribution of those functions whose percentages calculated by average method are above this value.

For example, if you use the highest BB value as average method and try to filter 25% files, the graph options should be set as follows:

```
Average Method: Highest BB Value
Filter functions under (%): 10.0
```

Press Refresh. Then the functions whose highest BB value percentages are at least 25% of the maximum value will be listed in the graph as follows:
5.2.5.3 View Instruction Graph

To open instruction graph, press button in CodeAnalyzer toolbar or choose File -> CodeAnalyzer -> Statistics -> Instruction Mix. You can also select the tab in Statistics view. There are two modes of instruction mix graph: count and percentage. You can press the button Show Executions or Show Count to switch between them.

The following screen capture shows a typical instruction count graph of loaded executable.
The x-ordinate shows the names of instructions in the loaded executable. The y-ordinate denotes count distribution of instructions. Each column of the graph refers to an instruction. Their color shows how frequently these instructions appear in the loaded executable.

The following screen capture shows a typical instruction executions graph of loaded executable.
The x-ordinate shows the names of instructions in the loaded executable. The y-ordinate denotes executions distribution. Each column of the graph refers to an instruction. Their color shows how frequently these instructions are executed in the loaded executable.

5.2.5.4 View Comments Graph

To get comments graph, be sure to collect comments first.

Then press button in toolbar or choose File -> CodeAnalyzer -> Statistics -> Comments. You can also select its tab in Statistics view directly. There are two options of graph: show graph for comment and show graph for function. In graph for comments, you can select type of comments to display.
In the above graph, the x-ordinate shows the names of functions that have comments of **Load After Store** in power 5. The y-ordinate denotes number of this comment for each function. Each column of the graph refers to a function.
In graph for function, you can select any of functions which have comments and display the number of different comments it contains. In the above graph, the x-ordinate shows the name of comments we try to collect in the first step. The y-ordinate denotes number of these comments in .checkfneg function. You can filter the value of columns by pressing Refresh.

5.3 Pipeline Analyzer

Pipeline Analyzer is a port of the IBM Performance Simulator for Linux on POWER™, another alphaWorks technology. Pipeline joins the VPA toolkit to provide VPA users with the means of examining how code is executed on various IBM POWER processors. Pipeline Analyzer displays the pipeline execution of instruction traces generated by a POWER series processor. It does so by providing a scroll view and a resource view of the instruction execution.

You can also find the Pipeline Analyzer User Guide from within VPA. Select Help - Help Contents within VPA. To get context sensitive help, press F1 for Windows and AIX or press Ctrl+F1 for Linux.

5.3.1 Load an existing pipeline file

The IBM Performance Simulator for Linux on POWER™ project has directions for capturing an instruction trace and generating Pipeline data files.
Once you have made a run and generated a .pipe and .config file you can use the Pipeline Analyzer to look at them. When you start Visual Performance Analyzer, what you see is Welcome View. To open Pipeline Analyzer, choose Window -> Open Perspective -> Other -> Pipeline Analyzer.

The following screen capture shows the initial layout of Pipeline Analyzer.
Choose **File -> Open File**, and in Open File dialog select .pipe file with scroll mode information for inspection.

Please note if your .config file has different name as .pipe file, a second dialog for corresponding .config file will turn up for you to choose.
Next, the Pipeline Analyzer Perspective loads the data of .pipe file. A scroll editor is opened automatically. The general information of this .pipe file is displayed in the panel of Pipe View. The following screen capture shows the data loading of this file.
To open .pipe file with resource mode information, choose **File -> Open File**. A resource editor is opened as follows:

![Image of resource editor](image.png)

### 5.3.2 Navigating the scroll pipe view

Each time Pipeline Analyzer Perspective is opened, a pipe view turns up in the left part of perspective. It shows the detailed information of the currently active editor.

To open Pipe View manually, choose **Window -> Show View -> Pipeline Category -> Pipe View**. Note every perspective contains only one pipe view.

To view pipeline file containing scroll mode information, select **File -> Open File** and choose corresponding file. A scroll editor opens in the Pipeline Analyzer Editor and Pipe View display its information at the same time.

### 5.3.2.1 Using the overview graph
In the overview graph, the green box indicates the boundary of data displayed in the currently active scroll editor. To display data elsewhere, click its location in the graph. You will see that the green box moves to where you just click and scroll editor displays the data in detail.

5.3.2.2 Zoom in or out

To zoom in or out this graph, press buttons in tool bar. To fit both width and height of the graph to overview panel, choose in tool bar. Note that no scroll bars appear in overview panel after this operation.

5.3.2.3 The event message and offset panel

The Event Message and Offset panel displays the denotation of instruction event which your mouse points at in the Scroll Editor. The following screen shot shows the change of this information when the mouse moves from symbol D to symbol M in the same line of table.
If two more events in an instruction execution occur at the same time cycle, their corresponding symbol will be highlighted and all the events be listed in Event Message panel. The following screen shot shows the event message of a highlighted symbol "F" in the above picture.

![Event Message]

If there are too many event messages to display, you can resize this panel.
To scroll simultaneously with the currently active editor, press \( \text{in tool bar}. \) The following screen shot shows this effect.

Please note that this function ensures that the green box is always within the overview graph of Pipe View.

### 5.3.3 Navigating the resource view

To view pipeline file containing resource mode information, select \textit{File -> Open File} and choose corresponding file. A resource editor opens in the Pipeline Analyzer Editor and Pipe View display its information at the same time.
The following screen shot shows the appearance of resource editor after a pipeline file is loaded.

In resource editor, a side bar named **Resource Name** in the right lists all the resources recorded in pipeline file. Each line of table in the left shows the usage distribution of this resource during time period. Each symbol in the table means an instruction event. Its denotation is shown in Event Message panel of Pipe View. If more than two events struggle for one resource at the same time, its symbol in the table turns red automatically. In this case, all the event messages are listed in Event Message panel of Pipe View. From General panel of Pipe View, you can see that this file has a total of 507 cycles and 11648 lines. The current time divider is 1, which means each symbol in the table indicates one time cycle.

The two red sliders in the table are named slider bar. It focuses on the cell which your mouse points at. Its ordinate is shown as Cursor value of Offset panel in Pipe View. The grey sliders in the table are named base axes. It focuses on the latest click of your left mouse. Its ordinate is shown as Base value of Offset panel in Pipe View. The distance between slider bar and base axes is calculated and shown as Offset value of Offset panel in Pipe View.

To navigate the editor, you can press left, ri0067ht, up, down keys or 'H', 'J', 'K', 'L' keys.

### 5.3.3.1 Zoom in or out

To zoom in or out the table in scroll editor, press buttons in tool bar or select it in Pipeline Menu.

### 5.3.3.2 Show Dots
To show dots in the table, press in tool bar or select it in Pipeline Menu. The following picture shows a table with dots.

5.3.3.3 Show Hover

To show the denotation of each symbol in the table, press in tool bar or select it in Pipeline Menu. While your mouse points at a line in the table for a while, a label which explains this line of resource usage turns up. The following screen shot shows the symbol message for a confile point in resource editor.

5.3.3.4 Show Slider

To show slider bar while scrolling around the table, press in tool bar or select it in Pipeline Menu.

5.3.3.5 Hide Unvisited Resources
When the usage distribution of pipeline file is distributed loosely, you can hide those unvisited resources. To do this function, press in tool bar or select it in Pipeline Menu.

The following screen capture shows the condensed resource table for the above opened file.

5.3.3.6 Change Color for symbol

To change color for each symbol in the table, choose Pipeline -> Settings... -> Trace.
For example, if we set Dprefetch event in the opened resource file in the first screen shot to be yellow, the resource editor changes as follows:

Please note, the red lines in this view mean conflict for resource use. This is a system-defined color.

5.3.3.7 Change Time Divider

To change the number of time cycles for each symbol in the table, choose Pipeline -> Change Time Divider.
The default value of this box is the current time divider. The following screen capture shows the resource editor after setting time divider to be three.

5.3.4 Show SPU Timing

5.3.4.1 Compile SPU Binary

Code Analyzer analyzes the compiled binary with relocation information and sends its disassembly result to pipeline analyzer to show the spu timing info.
User need to compile the binary with 
\texttt{-Wl,-q} compiler option to keep the relocation infomation. For example, user can use this command "\texttt{spu-gcc ackermann.c -Wl,-q -o ackermannWlq}" to generate the executable(\texttt{ackermannWlq}) with relocation infomation.

5.3.4.2 Open SPU Binary

First, open the spu binary file with code analyzer.

Then, the symbols in the binary file are listing in the program tree view.
Right click on one symbol, and click the Show SPU Timing menu item.

SPU Timing Editor is shown there.
Click Pipeline Analyzer icon to show the spu timing in the pipeline perspective.
5.4 Counter Analyzer

The Counter Analyzer tool is a common tool to analyze hardware performance counter data among many IBM eServer platforms, which includes systems running on AIX, i5OS, zOS, Linux on POWER, Linux on Cell BE.

The Counter Analyzer tool accepts hardware performance counter data in the form of a cross-platform XML file format. The tool uses either build-in hsqldb database engine or external DB2 instance to store the raw performance counter data. The tool provides multiple views to help user identify the data. The views can be divided into two categories: one category is the “table” views, which are basically two-dimension tables displaying data. The data could be raw performance counter values, derived metrics, counter comparison results and so on. Another category is the “plot” views. In these views data are represented by different kind of plots. The data could also be raw performance counter values, derived metrics, and comparison results and so on. Besides these “table” views and “plot” views, there are also some “utility” views to help user configure and customize the tool.

You can also find the Pipeline Analyzer User Guides from the VPA. Select Help - Help Contents within VPA. To get context sensitive help, press F1 for Windows and AIX or press Ctrl+F1 for Linux.
5.4.1 Basic concepts for Counter Analyzer

- **Performance Monitoring Counter**

Performance monitor counter provides comprehensive reports of events that are critical to performance on IBM systems. It is able to gather critical hardware events, such as the number of misses on all cache levels, the number of floating point instructions executed, the number of instruction loads that cause TLB misses.

- **Metrics**

Metric is calculated with user-defined formula and event count from performance monitor counter. It's used to provide performance information like CPU utilization rate, million instructions per second. This helps the algorithm designer or programmer identify and eliminate performance bottlenecks.

- **CPI Breakdown Model**

Cycles per instruction (CPI) is the measurement for analyzing the performance of a workload. CPI is simply defined as the number of processor clocked cycles needed to complete an instruction. It is calculated as \( CPI = \frac{\text{Total Cycles}}{\text{Number of Instructions Completed}} \). A high CPI value usually implies underutilization of machine resources.

On a POWER5 system, you can break down your workload CPI into individual components, as the POWER5 has several programmable counters available to count events that can calculate the components of CPI and allow you to determine how to improve performance on a given workload.

The following is an instance of CPI breakdown model:
<table>
<thead>
<tr>
<th>Total cycle &lt;# cycles&gt;</th>
<th>Completion cycles &lt;A:group complete cycles&gt;</th>
<th>PowerPC Base completion cycles &lt;A1: One or more PowerPC instructions completed this cycle&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Completion Table empty (GCT empty) cycles &lt;B&gt;</td>
<td>overhead of cracking/microcoding and grouping restriction <a href="">A2:(A)-(A1)</a></td>
</tr>
<tr>
<td></td>
<td>Completion Stall cycles &lt;C: total-(A)-(B)&gt;</td>
<td>Stall by LSU inst &lt;C1&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Stall by reject &lt;C1A&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Stall by translation (rejected by ERAT miss) &lt;C1A1&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Other reject &lt;C1A2: (C1A)-(C1A1)&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Stall by D-cache miss &lt;C1B&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Stall by LSU basic latency, LSU Flush penalty &lt;C1C: (C1)-(C1A)-(C1B)&gt;</td>
</tr>
<tr>
<td></td>
<td>Stall by FXU inst &lt;C2&gt;</td>
<td>Stall by any form of DIV/MTSPR/MFSPR inst &lt;C2A&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Stall by FXU basic latency &lt;C2C: (C2)-(C2A)&gt;</td>
</tr>
<tr>
<td></td>
<td>Stall by FPU inst &lt;C3&gt;</td>
<td>Stall by any form of FDIV/FSQRT inst &lt;C3A&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Stall by FPU basic latency &lt;C3B: (C3)-(C3A)&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>others (Stall by BRU/CRU inst, flush penalty (except LSU flush), etc.) &lt;C4: (Completion Stall cycles)-(C1)-(C2)-(C3) &gt;</td>
</tr>
</tbody>
</table>

The above table represents a CPI breakdown model where the total cycles of a workload is divided into three components: Completion cycles, Completion Table empty or GCT empty cycles, and Completion Stall cycles. The base completion cycles are the number of cycles that would be needed if grouping was perfect. Otherwise, stalls happen, and they can be attributed to either Completion Table empty or Completion Stall cycles. A Completion Table empty condition occurs when no groups are completing on a given cycle because of either icache miss or branch misdirection. Meanwhile, the Completion Stall cycles are those stalls caused by any of the following instructions: LSU, FXU, FXU long (all forms of div, mtsp, mfspr), FPU, and FPU long (all forms of fsqrt, fdiv); or by other events such as Dcache miss, Reject, and Reject by translating (ERAT miss).

5.4.2 Load an existing counter data file

5.4.2.1 Open Counter Analyzer Perspective
1. When you first start Visual Performance Analyzer after installation, what you see is Welcome View. To start Counter Analyzer, choose Window -> Open Perspective -> Other.

2. In the Select Perspective dialog choose Counter Analyzer and click OK.
3. The following screen capture shows the initial layout of Counter Analyzer.
4. You can also open Counter Analyzer Perspective simply by clicking in the toolbar.

5.4.2.2 Load in counter data file

Choose File -> Open File, and in Open File dialog select one counter data file with suffix ".pmf".
You can also load in counter data from repositories, which will not be covered here.

5.4.2.3 Brief Introduction to Counter Analyzer Perspective

After loading in the counter data of `.pmf` file, the Counter Analyzer Perspective displays the data in its views and editors. Primary information of details, metrics and CPI breakdown is displayed in Counter Editor. Resource statistics information of the file (if available) will be showed in tabular view Resource Statistics. View Graph illustrates the details, metrics and CPI breakdown in a graphic way.
5.4.3 Navigate the Counter Analyzer Perspective

The following are tasks that you can perform to navigate around counter data within Counter Analyzer.

5.4.3.1 Open Counter Data

There are two ways to open counter data using Counter Analyzer, either from counter data files or from database repository.

5.4.3.1.1 Open Counter Data from PMF Files

1. Choose **File -> Open File**, and in Open File dialog select one counter data file with suffix ".pmf".
If one counter file is selected, it will be opened in editor, and its resource statistics information (if available) will be showed in tabular view **Resource Statistics**.

By default, a local repository is provided by VPA tool. You can also connect to remote DB2 repositories. Remote repositories can be created, configured, refreshed, and discarded. Files in these repositories can be opened in
Counter Editor and deleted. Besides, you can import counter data files into and query counter data from these repositories.

5.4.3.1.2 Open Counter Data from Connection View
By default, a local repository is provided by VPA tool. You can also connect to remote DB2 repositories. The local connection is a local connection, which you can add Counter Analyzer Support under it, as well as other supports. Remote connection is a DB2 connection. It can be created, configured, refreshed, and deleted.

- Create/Configure/Refresh/Discard a Connection

Create Connection
In Database Connection view, select New Connection in the context menu, you can create a new repository. Local Repository is created by default and can only be configured and refreshed. Description view lists basic information of the repository.

Configure Connection
The following is the Configure Dialog of Connection.
Add Product Support

Right-click on the connection node, choose the product support you want to add to the connection node, a confirm dialog will occur.

You can create a new repository, or select an existing repository from the Existing List.
**Refresh Connection**

Double click a Connection or click on the + icon to expand the tree.

A **Password Dialog** will pop up when opening the repository for the first time. If the password is saved, you no longer have to re-enter it when Counter Analyzer is closed and restarted later. If the password is not saved, you are required for it every time they start Counter Analyzer.

A **Password Dialog** will pop up when opening the repository for the first time. If the password is saved, you no longer have to re-enter it when Counter Analyzer is closed and restarted later. If the password is not saved, you are required for it every time they start Counter Analyzer.

You can also choose to clear the password on the context menu.

**Discard Connection**

You can choose to discard the Connection on the context menu. Be sure that delete all its children first.
**Discard Support**

You can choose to discard the Product Support on the context menu.

- **Import File**
  
  Right-click on the Support node, select Import File action, then choose the file you want to import from the file dialog.
5.4.3.2 View Counter Data

You can see raw counter data, metrics data and CPI breakdown data in different pages of editor. All these data are retrieved from one counter file.

5.4.3.2.1 View Raw Counter Data

Row counter data are shown in Tab Details. On the context menu, you can switch display mode here.
In all tabs, **Filter** is used to filter processors and events.
Row counter data can be displayed in three modes: **Event**, **Group/Event/Time Slice**, and **Group/Time Slice/Event**. If the counter data has time slice information, all these three modes can be supported. If not, only **Event** mode is enabled, and the other two modes are disabled.

**“Event” Mode**

In this mode, all events and their counter data are listed in the editor. Data here are normalized event count instead of actual data in counter data file.
"Group/Event/Time Slice" Mode

If the counter data has time slice information, there are two modes to display more. In **Group/Event/Time Slice** mode, the data can be grouped first by group, and then event. Data here are actual event count in counter data file.

<table>
<thead>
<tr>
<th>Group/Event/TimeSlice</th>
<th>All Processors</th>
<th>Average</th>
<th>Processor 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>pm_L1_tlbmiss (No=43)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 PM_DATA_TABLEWALK_CYC</td>
<td>34,604</td>
<td>8,851</td>
<td>14</td>
</tr>
<tr>
<td>Time Slice 0 (1.0406)</td>
<td>9,796</td>
<td>2,449</td>
<td>9</td>
</tr>
<tr>
<td>Time Slice 9 (1.0399)</td>
<td>13,419</td>
<td>3,355</td>
<td></td>
</tr>
<tr>
<td>Time Slice 10 (1.0406)</td>
<td>5,462</td>
<td>1,365</td>
<td>2</td>
</tr>
<tr>
<td>Time Slice 27 (1.070)</td>
<td>5,927</td>
<td>1,482</td>
<td>2</td>
</tr>
<tr>
<td>2 PM_DTLB_MISS</td>
<td>108</td>
<td>27</td>
<td></td>
</tr>
<tr>
<td>3 PM_LD_MISS_L1</td>
<td>010</td>
<td>204</td>
<td></td>
</tr>
<tr>
<td>4 PM_LD_REF_L1</td>
<td>407,838,799</td>
<td>101,959,700</td>
<td>89,636,893</td>
</tr>
<tr>
<td>5 PM_INST_CMPL</td>
<td>1,273,066,580</td>
<td>318,271,645</td>
<td>296,306,297</td>
</tr>
<tr>
<td>6 PM_RUN_CYC</td>
<td>6,794,592,143</td>
<td>1,686,546,036</td>
<td>1,566,506,566</td>
</tr>
</tbody>
</table>

"Group/Time Slice/Event" Mode

If the counter data has time slice information, the other mode to display more information is **Group/Time Slice/Event mode**, in which counter data is grouped first by group, and then time slice. Data here are also actual event count in counter data file.

<table>
<thead>
<tr>
<th>Group/TimeSlice/Event</th>
<th>All Processors</th>
<th>Average</th>
<th>Processor 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>pm_L1_tlbmiss (No=43)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Time Slice 0 (1.0406)</td>
<td>9,796</td>
<td>2,449</td>
<td>9,555</td>
</tr>
<tr>
<td>1 PM_DATA_TABLEWALK_CYC</td>
<td>44</td>
<td>11</td>
<td>44</td>
</tr>
<tr>
<td>2 PM_DTLB_MISS</td>
<td>342</td>
<td>86</td>
<td>337</td>
</tr>
<tr>
<td>3 PM_LD_MISS_L1</td>
<td>95,163,255</td>
<td>23,73</td>
<td>89,635,808</td>
</tr>
<tr>
<td>4 PM_LD_REF_L1</td>
<td>314,592,394</td>
<td>78,64</td>
<td>296,304,297</td>
</tr>
<tr>
<td>5 PM_INST_CMPL</td>
<td>1,006,501,209</td>
<td>421,5</td>
<td>1,500,609</td>
</tr>
<tr>
<td>6 PM_RUN_CYC</td>
<td>6,284,592,143</td>
<td>1,686</td>
<td></td>
</tr>
</tbody>
</table>

5.4.3.2.2 View Metrics Data

Metrics data are shown in Tab **Metrics**.

**Edit/Load/Save Metrics Variables**

In the right part of the editor, all variables associated with the current metrics are listed in a table with their names and values. Left click the **Value** column of each variable, and you can modify its value. On the context menu, users can further choose **Load Variables** to load one variables file, and choose **Save Variables** to save current variables in the editor to one variables file. If one variable file is loaded, all variables’ values are updated, which makes all metrics be calculated again. (Only Variable “total_time” is read only and cannot be overwritten.)
Change Metrics

You can choose to change metrics file on the context menu, and applies it to the active counter data.

The metrics file to be selected can be either external files or built-in files. The change derived metrics dialog is as follows:
5.4.3.2.3 View CPI Breakdown Data

CPI breakdown data are shown in Tab CPI Breakdown.

Change CPI Breakdown Model

You can change CPI breakdown model on the context menu, and apply it to the active counter data.
The CPI breakdown model to be selected can be either external files or built-in files. The Change CPI breakdown model dialog is as follows:

**Export as HTML**

Choose **Export as HTML** on the context menu, you can export CPI breakdown data into an HTML file.
The CPI breakdown HTML file is as follows:

### CPI Breakdown Model

<table>
<thead>
<tr>
<th>Model</th>
<th>CPI</th>
<th>WCET</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOTAL CPI</td>
<td>5.332</td>
<td>1.981</td>
</tr>
<tr>
<td>CPI_CMU_CRC</td>
<td>0.775</td>
<td>0.165</td>
</tr>
<tr>
<td>CPI_GET_EMPTY</td>
<td>0.115</td>
<td>0.265</td>
</tr>
<tr>
<td>CPI_CMU_CRC</td>
<td>0.003</td>
<td>0.001</td>
</tr>
<tr>
<td>Show as Double</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Show as Science Double</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Show as Integer</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Filter</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Change CPI Breakdown Model</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Export as HTML</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

5.4.3.3 View Temporal Chart

Temporal chart displays the counter data by samples. Each sample has one value represented in chart. You can aggregate several samples as one using Aggregation Scale.
Aggregation Scale is used to scale the aggregation rate of samples. You may feel overwhelmed in front of too many samples in Graph View in temporal mode sometimes. And with the aggregation scale, it is easy for you to choose the number of samples to display. For example, you may choose to display 20 samples in Graph View:

And you may also choose to display 10 samples here:

Filter is used to filter processors.
In temporal mode, when you select one event in Tab Details, one metric in Tab Metrics, or one CPI component in Tab CPI Breakdown, the temporal information of this item is displayed in Graph view. The chart type can be switched between Multiple Bar Chart and Line Chart.

- Line chart of event:

- Multiple bar chart of one event:
5.4.3.4 Create Chart Graph

You can use chart wizard to customize some complicated chart. Let us see some examples.

5.4.3.4.1 Example one: Compare some events data in average.

1. Open a counter file. Choose "Create Chart Graph..." in Graph View context menu to enter chart wizard.

![Chart Graph Example](image)

2. For the first step of wizard, please choose what kind of data you want to display, Events/Metrics or CPI. Here we choose Events/Metrics for this demonstration.

![Chart Graph Wizard](image)

3. Choose some Events and/or Metrics and click Next.
4. Decide which processor to display.

5. Last step, Choose a chart type you like.
6. Finally, we get a chart which compare some CPI data of different files.

5.4.3.4.2 Example two: Compare some CPI data.

1. Open a comparison editor. Choose "Create Chart Graph..." in Graph View context menu to enter chart wizard.

2. Choose what kind of data you want to display, Events/Metrics or CPI. This time we choose CPI for this demonstration.

3. Choose some CPI and click Next.
4. Then we will enter "Group By" page. Group by CPI means that each multiple bar group shows one specified CPI value of all files. The form is just like "FileA.CPIA FileB.CPIA FileA.CPIB FileB.CPIB". While group by file means that each multiple bar group shows all of the selected CPIS of one file. The form is just like "FileA.CPIA FileA.CPIB FileB.CPIA FileB.CPIB".

5. Last step is Choose Compare Mode. In one word, "Compare Side By Side" will show all files while "Compare Against Baseline" will ommit baseline file.
6. Because the only form of this chart is Multiple Bar Chart, so "Choose Chart Type" page is hidden. Finally, we get a chart which compare some CPI data of different files.

5.4.3.5 Compare Counter Data

You can click on the tool bar to select several files to compare. In the pop-up dialog, you must select 2~4 target files to compare. You can either select files from the explorer, or select files from repositories (you have to refresh repositories in Counter Repository View in advance). Select one file, and click the Add Button to add it. The first file you add is regarded as the base file.

You can further specify the metrics and CPI breakdown Model to apply to the counter data, or do it later.

The following screen snapshot displays the Select To Compare Dialog:
5.4.3.5.1 View Comparison Data

Comparison Editor is opened to display comparison result side by side. The following items can be compared:

- The total counts of one event are compared.
- The derived metrics values are compared.
- The CPI breakdown values are compared.

The following screen snapshot displays comparison data of each event from different data sources. △ refers to difference between the target file and the base file, while % refers to the target files’ proportion of the base file.
The comparison is based on the sum of all processors, the average of all processors, or both. You can specify it in Filter.
5.4.3.6 Edit Metrics/CPI Breakdown Model

5.4.3.6.1 Edit Derived Metrics

1. You can select on the tool bar to open one derived metrics file to edit.
2. You can view and edit derived metric formula in Metrics Editing dialog. The supported operation includes:
   o Add new derived metric
   o Edit derived metric
   o Delete derived metric
3. After modification, you can save the file or save it as another derived metrics file.

5.4.3.6.2 Edit CPI Breakdown Model

1. You can select on the tool bar to open one CPI breakdown model definition file.
2. You can view and edit CPI breakdown model in Edit CPI Breakdown Model dialog. The supported operation includes:
   o Add new component
   o Edit component
   o Delete component
3. After modification and validation, you can save the file or save it as another CPI breakdown model definition file.

5.4.3.7 Import Counter Data File into Repository

To create Counter Analyzer Support, you should create a database connection first. Please refer to section 5.1.7 for details on how to create and manage database connections.

Then, right-click on the connection to create Counter Analyzer Support,

If you are using hsqldb connection, all existing database under this path will be listed, you can create a new one, or attach an existing one.

If you are using db2 connection, all existing schema under this database will be listed, you can create a new one, or attach an existing one.
After Counter Analyzer Support has been created, then, you can import file into this support, open it in editor, or delete it.

You can delete files manually to release disk space. Multi-selecting is allowed. See the picture below.
This view supports sorting operation. You can sort the files by name, size, or date.

To sort files, select the sort mode on the action bar, or click the title directly.

5.5 Trace Analyzer

Trace Analyzer visualizes Cell BE traces containing information such as DMA communication, locking/unlocking activities, mailbox messages, etc.

Trace Analyzer provides several views that help the user make sense of the trace data. The trace can be plotted in a graphical view, organized by core, along a common timeline. Alternatively, the user can traverse the trace records in a textual table. Another view provides the detailed data for each kind of records, for example, lock identifier for lock operations, accessed address for DMA transfers, etc.

You can also find the Trace Analyzer User Guides from the VPA. Select Help - Help Contents within VPA. To get context sensitive help, press F1 for Windows and AIX or press Ctrl+F1 for Linux.

5.5.1 Basic concepts

- **Events**

  Events are records that have no duration, for example, records describing non-stalling operations, such as releasing a lock. Events’ input on performance is normally insignificant, but they may be important for understanding the application and tracking down sources of performance problems.

- **Intervals**

  Intervals are records that may have non-zero duration. They normally come from stalling operations, such as acquiring a lock. Intervals are often a very significant performance factor, and identifying long stalls and their sources is an important task in performance debugging. A special case of an interval is *live interval*, that starts when an SPE thread begins to execute and ends when the thread exits.

5.5.2 Load an existing trace file

5.5.2.1 Open Trace Analyzer Perspective
When you first start Visual Performance Analyzer after installation, what you see is Welcome View. To start Trace Analyzer, choose Window -> Open Perspective -> Other. In the Select Perspective dialog choose Trace Analyzer and click OK.

The following screen capture shows the initial layout of Trace Analyzer.

5.5.2.2 Load in trace file

Choose File -> Open File ..., and in Open File dialog select one trace file with suffix ".pex"..

5.5.2.3 Brief introduction to Trace Analyzer Perspective

After loading in the trace data, the Trace Analyzer Perspective displays the data in its views and editors. Going from the top left clockwise, we see:

- **Navigator View**
- **Trace Editor** showing the trace visualization by core
• **Details View** showing the details of the selected record (if any)
• **Color Map View**, allowing the user to view and modify color mapping for different kinds of events
• **Trace Table View**, which shows all the events on the trace in the order of their occurrence

### 5.5.3 Navigate the Trace Analyzer Perspective

The following are tasks that you can perform to navigate around trace data within Trace Analyzer.

#### 5.5.3.1 View Trace Data Graph

A graphical presentation of the trace is shown in the editor window.

![Graphical View of Trace Data](image)

Data from each core is displayed in a separate row, and each trace record is represented by a rectangle. Time is represented on the horizontal axis, so that the location and size of a rectangle on the horizontal axis represent the corresponding event's time and duration. The color of the rectangle represents the type of event, as defined by the Color Map View.

In the rows corresponding to the SPEs, note the full-height green rectangles. They show the live intervals starting with the context switch that takes the thread on CPU and ending with a context switch that takes it off CPU. On top of them are painted representations of the events that occurred during the thread execution.

**Trace Editor Components**

The following figure shows the different components of the Graphical View.
Going from top to bottom, we have:

- The **marker ruler** shows where the selected record (if any) is located on the trace's timeline (look for the orange-and-white selection marker). Clicking on the marker ruler scrolls the view to make the selected event visible. Note that there are also vertical marker rulers, located in each row between the core id and the graph. These rulers show on which core the selected event occurred.

- The **scrollbar** can be used to scroll back and forth in time.

- The **ruler bar** shows the time values, in the same units as those used in the trace.

**Trace Editor Tools**

When a trace is open in the Graphical View, the following toolbar is added to the standard Eclipse toolbars:

This toolbar is only active when the focus is on the Trace Editor. The following tools are available:

- ![Selection tool](image) : **Selection tool**. Pick this tool and click with it a record on the Graphical View to select the record. This will scroll the Record List View to the selected record and display its details in the Record Details View.

- ![Zoom-in point tool](image) : **Zoom-in point tool**. Pick this tool and click one of the graphs to zoom in while keeping the time value at the click point at the same location.
• **Zoom-out point tool.** Pick this tool and click one of the graphs to zoom out while keeping the time value at the click point at the same location.

• **Zoom-all tool.** Pick this tool and click anywhere in the graph to fit all the trace into the view.

• **Zoom-in area tool.** To fit a specific region into the view, pick this tool and in one of the rows mark the area you want to fit into the view.

• **Drag tool.** To scroll the view back and forth along the time axis, pick this tool, and hold the right mouse button pressed while dragging the graph

**Trace Editor Coloring Conventions**

When analyzing a trace, it is often important to distinguish between a large number of short intervals and a single long interval, which may be a good target for optimization. In order to aid in this analysis, Trace Analyzer emphasizes intervals with a border whose color is a darker hue of the event's color. Please refer to the Color Map View for the exact color mapping of any particular editor instance.

![Long interval and multiple short intervals](image)

**5.5.3.2 View List of Trace Records**

You can view the list of the records in the trace in the **Trace Table View**. Click on a row to select a record and see its details in the **Record Details View**. **Trace Table View** selection is also synchronized with the selection in the **Trace Editor**, so that each scrolls to and highlights the selection done in the other. Controls at the top of the view allow to change the start and size of the trace chunk visible in the table, or to scroll to the next/previous chunk. To save the trace in a text file, click the **Save as Text File** in the view menu. The file will be placed next its corresponding .pex file, under extension .txt.
5.5.3.3 View Record Details

The **Record Details View** shows the names and values for all the fields defined for the selected record.

![Record Details View](image)

5.5.3.4 Change Colors used in Trace Editor

The **Color Map View** holds a hierarchical list of record types and their color mapping.

![Color Map View](image)
To change the color assigned to a particular type, double-clicking the corresponding row in the color map to open the color chooser dialog. Changing a color for a category changes the color for all record types in this category. To change the color of an individual type within a category, click on the plus sign at the category row's left to expand it, then double-click the line that corresponds to the desired type to change its color mapping.

5.6 Control Flow Analyzer

Control Flow Analyzer is the tool to analyze the call trace data collected by the tool like Performance Inspector JProf. The call trace data contains the information like when one method calls another, how much time is spent in every invocation, and so on. Control Flow Analyzer provides the two major visualization ways to analyze call trace data, which are execution flow graph and call tree table.

You can also find the Control Flow Analyzer User Guides from the VPA. Select Help - Help Contents within VPA. To get context sensitive help, press F1 for Windows and AIX or press Ctrl+F1 for Linux.

5.6.1 Basic Concepts

5.6.1.1 Call Tree and Call Context Tree

The two common representations of call trace data are Call Tree and Call Context Tree. In a Call Tree, each method invocation is represented as one tree node. The caller method invocation is parent node, while the callee method invocation is the node itself. In a Call Context Tree, all child nodes of the same method are merged to one node, which also attached under the node of caller method invocation.

If we have the call sequence as follow (“A -> B” means A calls B, and “<-” means B returns back to A.) :

A -> B, B -> C, C -> D, <-, C -> E, <-, <-, B -> C, C -> E, <-, <-, A -> B, B -> F, <-, <-

The Call Tree looks like:
The Call Context Tree looks like:

```
      A
  /   \   
 B     C
  |     |
 D     E
```

The Call Context Tree looks like:

```
      A
  /   \   
 B     C
  |     |
 D     E
```

Among the tools Control Flow Analyzer supports, Call Tree can be generated from z/OS VICOM trace data, AIX ctrace trace data, and WBI JProf generic trace data, while only Call Context Tree can be generated from WBI JProf runtime trace data.

### 5.6.1.2 Base Time and Cumulative Time

There are two kind of time measured, wall clock time and per-thread/per-process time. The wall clock time is the time elapsed in wall clock, while per-thread/per-process time is the time only measured when this thread/process runs.

Different call trace collection tools also have different metric to measure time taken by one method invocation. The possible metrics are second, millisecond, microsecond, nanosecond, cycle, and event instruction.
In order to help call trace analysis, some terms are defined as follow:

- **Starting Time** is when this method is invoked relative to the first method invocation of the whole call tree.

- **Base Time** is the time spent on this method itself. It doesn't include the time its child method invocations take.

- **Cumulative time** is the time from this method's entry to its exit. It include the time its child method invocations take.

So for one invocation, we have the equation.

\[
\text{cumulative time} = \text{base time} + \text{children's cumulative time}
\]

### 5.6.2 Load Call Trace Data File

When you first start Visual Performance Analyzer after installation, what you see is Welcome View. To start Control Flow Analyzer, choose **Window -> Open Perspective -> Other**. In the Select Perspective dialog choose **Control Flow Analyzer** and click **OK**.

The following screen capture shows the initial layout of Control Flow Analyzer.

Choose **File -> Open File**, and in Open File dialog select one call trace data file with suffix ".jprof".

### 5.6.3 Control Flow Analyzer Perspective Introduction
Control Flow Analyzer perspective contains one editor and some views. Going from the top left clockwise, we see:

- Navigator View
- Control Flow Analyzer Editor containing Execution Flow Editor Page and Call Tree Editor Page, which display execution flow graph and call tree.
- Call Stack View showing the call stack of the selected invocation.
- Information View showing the overall information of the opened call trace data file.
- Invocation View showing the parent invocation and children invocations of the selected invocation.

### 5.6.4 View Call Trace Data in Execution Flow Editor Page

Execution Flow Editor Page provides one powerful way to visualize how one application executes. It consists of Execution Flow Graph and Call Tree Table.

![Execution Flow Graph example](image)

Execution Flow Graph visualizes the application execution thread by thread or process by process depending on different call trace data. Each thread/process has one rectangle area, in which all method invocations are visualized. At the rightmost side, there is one time axis. The unit of time axis can be cycle, instruction, or second depending on different call trace data.
The color bar represents the life cycle of one method invocation, and its height represents how long this invocation lasts from entry to exit. The red line between color bar represents that the left method initiates invocation to the right one. So from this graph, you can know how long one invocation lasts, and when one invocation initiates another one.

There are many tool buttons to zoom in/out, navigate, and save image of execution flow graph.

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="icon.png" alt="Zoom In" /></td>
<td>Zoom in.</td>
</tr>
<tr>
<td><img src="icon.png" alt="Zoom Out" /></td>
<td>Zoom out.</td>
</tr>
</tbody>
</table>
The Call Tree Table has the following attributes for each method invocation.

<table>
<thead>
<tr>
<th>Invocation (with instruction)</th>
<th>Method Set</th>
<th>Starting Time</th>
<th>Base Time</th>
<th>Cumulative Time</th>
<th>Cumulative %</th>
</tr>
</thead>
<tbody>
<tr>
<td>method_name()</td>
<td>Class name</td>
<td>1234567.0</td>
<td>123456.0</td>
<td>1234567.0</td>
<td>0.0%</td>
</tr>
<tr>
<td>method_name()</td>
<td>Module name</td>
<td>1234567.0</td>
<td>123456.0</td>
<td>1234567.0</td>
<td>0.0%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invocation</td>
<td>Method name.</td>
</tr>
<tr>
<td>Method Set</td>
<td>Method Set name. This can be class name for Java method, or module name for</td>
</tr>
<tr>
<td></td>
<td>C/C++ method.</td>
</tr>
<tr>
<td>Starting Time</td>
<td>The timestamp when the current invocation is initiated.</td>
</tr>
<tr>
<td>Base Time</td>
<td>The time spent on the current invocation itself. It doesn't include the time</td>
</tr>
<tr>
<td></td>
<td>its child invocations spend.</td>
</tr>
<tr>
<td>Cumulative Time</td>
<td>The time spent on the current invocation itself and its child invocations.</td>
</tr>
<tr>
<td>Cumulative %</td>
<td>The percentage of the cumulative time relative to the time period of the whole</td>
</tr>
<tr>
<td></td>
<td>trace.</td>
</tr>
</tbody>
</table>

5.6.5 View Call Trace Data in Call Tree Editor Page

Call Tree Editor Page is used to analyze the relationship between caller and callee.

It consists of one call tree table and a set of invocation relationship tables. Call Tree Table displays how and when one method calls another method, and is as same as the one in Execution Flow Editor Page. When you double click one method invocation in Call Tree Table, the invocation relationship tables display the parent invocation and child invocations of the selected one.
The Call Tree Table has the following attributes for each method invocation.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invocation</td>
<td>Method name.</td>
</tr>
<tr>
<td>Method Set</td>
<td>Method Set name. This can be class name for Java method, or module name for C/C++ method.</td>
</tr>
<tr>
<td>Starting Time</td>
<td>The timestamp when the current invocation is initiated.</td>
</tr>
<tr>
<td>Base Time</td>
<td>The time spent on the current invocation itself. It doesn't include the time its child invocations spend.</td>
</tr>
<tr>
<td>Cumulative Time</td>
<td>The time spent on the current invocation itself and its child invocations.</td>
</tr>
<tr>
<td>Cumulative %</td>
<td>The percentage of the cumulative time relative to the time period of the whole trace.</td>
</tr>
</tbody>
</table>

Invocation Relationship Tables consists of three tables to display parent invocation, selected invocation, and child invocations. Double click on parent invocation or child invocations will select it, and display parent invocation and child invocations. It's possible to navigate among selected invocation using in invocation selection history.
5.6.6 Locate Invocation by Method Name

Sometimes, we know the method name, and want to locate it in the enormous call trace data. You can follow the following two steps to locate it.

Select one runnable (thread/process) in Call Tree Table, and choose Find ... from context menu.

For example, you want to locate the method "startsWith(String)". In the Find dialog, input the method name "startsWith", and click Find Next.
The first invocation which contains name "startsWith" is found and selected in Call Tree Table.

Select this invocation, and choose **Highlight in Execution Flow** in context menu.
This invocation will be highlighted in Execution Flow Graph.

5.6.7 Filter Invocations

The call trace data file contains a mess of runnables (threads/processes) and invocations. You can filter some of them out by using invocation filter. There are two ways support filtering, and they are Filter Runnables, and Filter Methods.
You can select the runnables to display in Filter Runnable dialog page.

You can define rule to include or exclude some methods whose names match some pattern. In the following example, all methods whose names start with "java/" will be excluded.
5.6.8 Drill Down one Invocation

When you open one call trace data file, you can view all invocations in it. If you want to narrow the scope to analyze, you can select one invocation and choose to drill down from Execution Flow Graph or Call Tree Table.

After you choose **Drill Down in Execution Flow Graph**, the selected invocation will be displayed in new Execution Flow Editor Page.
After you choose **Drill Down in Call Tree**, the selected invocation will be displayed in new Call Tree Editor Page.
6. Appendix A - sample profiling session

This process will walk through the typical usage of VPA to analyze problems, using the Profile Analyzer plug-in and some existing data. Where can you download the existing data? Look for the Sample ETM File at the bottom of the web page: http://www.alphaworks.ibm.com/tech/vpa/download.

The picture below shows the file being saved to a workspace directory.

![Profile Analyzer window showing file being saved](image)

If you don’t have the Profile Analyzer directory, try this:
Once we have the Profile Analyzer folder in view, it is sometimes necessary to REFRESH it. Just Right click and select REFRESH.
If you don’t have a Navigator tab, adding a new view is easy with VPA!
Just select Window => Show View => Other…

Select Navigator. It will open it as a tab in the bottom right window. Just drag it to the top left where we want it.
You will then need to navigate to D:\eclipse\workspace\Profile Analyzer to see your data.
Open the SAMPLE ETM by double clicking on it in the Navigator view.
The initial view will look something like this:

![Image of initial view](image1)

Expand Process > Module on the top right window by clicking the + sign:

![Image of expanded view](image2)
Once expanded, we can see that on the 4-way test system, our single threaded Java test case left 3 of the processors idling, leaving the ticks nicely divided amongst the processors. The Java test case (Process ID 410f4) took 23.24% of the total ticks. A tick is a sampled address recording where the system was executing code. JITCODE is where the work for this Java test case is being done, and we expect this due to Jitting of the Java methods.

The bar chart on the bottom left of VPA shows this well. Within the Java Process, we took a full 96.87% of the ticks in Jitted code. (If not shown, just click the bar chart icon on the Samples Distribution tab):
Almost everything with VPA is a mouse click away!!!

Double clicking any tab will change to a maximized view of that tab. Double clicking the same tab will change it back.

Try it by double clicking the tab for the window on the top right:

This makes it easier to see everything.
Double click the same tab to go back to the 4 pane view:

Select PID 410f4 and click the + to expand and see the modules within this process:
As we noted earlier, the JITCODE module is showing the most ticks as expected for the Java process. On the right hand side we see a breakdown of ticks for the JITCODE module.

Clicking on the JITCODE Module expands the window on the right, showing a breakdown of ticks in the JITCODE. The hottest symbols are shown in descending order. Notice that thirtytwo() is the biggest contributor at almost 52%:

If you look at the Samples Distribution view in the lower left corner, you will see that the information now corresponds to the JITCODE symbols. You can also switch between pie and bar charts.
DOUBLETICK ON thirtytwo() AND NOTICE THAT THE DISASSEMBLY/OFFSET INFORMATION IN THE BOTTOM RIGHT WINDOW IS NOW FILLED IN FOR THIS METHOD. CLICKING ON ANY OTHER METHOD ON THE TOP RIGHT WILL CHANGE EVERYTHING WE ARE SEEING IN THE GRAPH AND IN THIS DISASSEMBLY TO CORRESPOND TO THE METHOD SELECTED:

We need to get the Disassembly/Offset Information as a full screen, so double click that tab:
Double click on the tab
We now see a full window view of this:

Notice the colored bar on the right. This is the Hotness Bar.

- **Red** is used for any symbol that represents at least 20% of the total tick count.
- **Magenta** is used for any symbol that represents between 5% and 20% of the total tick count.
- **Blue** is used for any symbol that uses less than 5% of the total tick count.

Darker shades show higher activity than lighter shades.
We want to see the code with the most activity, so click on the % column to do a quick sort:

If you would like to use other Profile Analyzer views that is not currently shown. Simply go to Window->Show View -> Other …
Expand Profile Analyzer to see all the available views ...

And as an example .. click on **Temporal Profiling** to load the view...
And you should now have a screen that looks like this …

Temporal Profiling now appears as a view.