

# Implementing IBM POWER4, POWER4+, POWER5 and PowerPC 970FX Support in the Oprofile tool

Linux Performance Considerations

*IBM's Linux Technology Center*

Carl Love, Maynard Johnson, Bill Buros

## Abstract

The goal of this paper is to discuss the support added to OProfile for the IBM® POWER4™, POWER4+™, POWER5™ and PowerPC® 970FX processors. The paper describes the existing OProfile support in the Linux™ Kernel, and the details of how the support was added for the IBM POWER4, POWER4+, POWER5 and PowerPC 970FX processors are covered. The restrictions on specifying multiple events in OProfile on these processors is discussed in section 2 and is of interest to the general OProfile user and the maintainers. The rest of the paper is intended for the OProfile maintainer.

A unique feature of OProfile is its ability to profile based on specific hardware event counters. Specifying the events in OProfile for Power is slightly different than for other architectures. Examples of the flexibility and some of the limitations of the OProfile implementation are covered. The paper is intended for individuals looking to understand how OProfile interacts with the operating system kernel and the underlying Power hardware in extracting profiling information.

## 1 Overview of the OProfile tool and support added for POWER4, POWER4+, POWER5 and PowerPC 970FX processors

OProfile is a tool for profiling source code. The tool has been ported to a number of architectures including Intel™ x86, x86\_64, AMD as well as several others. The hardware performance counters are used to generate the profile based on any of the available performance counter events for the processor. A time-based profile can be obtained by profiling based on cycles. The use of the tool is explained in the user manual [1] available from the OProfile Web site, <http://oprofile.sourceforge.net>, in the Docs subdirectory. The generic internal implementation of OProfile is discussed in [2]. There is a paper [3] on the use of OProfile by William Cohen. The paper [4] by John Engle discusses using OProfile on the POWER architecture. And a recent paper [5] by Rob Nava contrasts OProfile with gprof and tprof on the POWER systems.

This paper specifically talks about the POWER4 POWER4+, POWER5 and PowerPC 970FX implementation of OProfile. Currently, these are the only POWER architectures supported by OProfile. This paper specifically discusses:

- the extensions to the OProfile tool needed to support the POWER4, POWER4+, POWER5 and PowerPC 970FX architectures
- the format of the event specification files
- the limitations the architecture puts on selecting multiple events
- the ongoing work for the Power architecture

This paper is not intended to be a user guide to help the new user to learn how to use OProfile except for explaining the nuances of selecting multiple events.

## **2 User guide to selecting multiple events.**

This section discusses the unique requirements for specifying multiple events on the IBM POWER4, POWER4+, POWER5 and PowerPC 970FX processors. These restrictions are a result of how the hardware performance counters were implemented. The architecture of the hardware performance counters is described to give the user an understanding of why there are restrictions for specifying multiple events in OProfile. This section is written as a guide for the general user as well as for the OProfile maintainer.

### **2.1 Overview of the performance counter architecture**

The performance counters consist of three control registers (**MMCR0**, **MMCR1** and **MMCR2**) and a set of event count registers. There are eight event count registers for POWER4, POWER4+ and PowerPC 970FX processors. There are six performance counters for POWER5 processor. The control registers contain fields to specify the various events to be counted by each counter, a set of bits to route the performance counter event signals to the counter, a bit to start/stop the counters, as well as a set of configuration bits. Due to routing conflicts in the signal multiplexers (MUX), it is not possible to arbitrarily configure any event in any counter, especially when setting up multiple events. Given a set of events to be programmed in each of the counters, it is not an easy task to determine if all of the events can be mapped to a counter (program the MUXs). A group consists of the six or eight events, depending on the hardware, that can be programmed simultaneously. For each group, the MMCR values are specified. The groups and events are specified in two files. The format of these files does not lend itself to the OProfile implementation very well. The details of how the event and MMCR values get mapped is discussed later in this paper. What is important for this discussion is the fact that there are valid groups of events that can be measured at the same time.

### **2.2 Profiling on multiple events**

OProfile allows multiple hardware triggered events to be used for profiling at the same time. This is done by specifying the events in the same opcontrol line. For example, if the user wanted

to measure L2 cache hits (PM\_DATA\_FROM\_L2) and processor cycles (PM\_RUN\_CYC) at the same time on a given machine, the command line would look like the following:

```
opcontrol --event=PM_DATA_FROM_L2_G44:10000 --event=PM_RUN_CYC_G44:10000
```

Note that if the user were to specify the events in separate opcontrol commands, only the event in the last opcontrol command would be programmed. This is due to how OProfile was implemented and is not specific to the processor.

To create a profile for multiple events, the user must specify a single group that contains each of the desired events. This requires being able to uniquely specify each event in the group since the same event can occur in multiple groups. Uniquely specifying each event was done by appending the group number to the end of the event name in each group. In the example above, both events belong to group 44 as indicated by the `_G44` suffix.

### **2.3 The number of available events**

The initial implementation included a subset of all available groups. This was due to a limitation in the size of the postprocessing event number variable. Version 0.9.1 and earlier of OProfile support up to 256 events. A patch to remove this restriction has been accepted into the OProfile code base. The restriction will be removed starting with the next release of the OProfile user code.

### **2.4 Setting the user and kernel mode.**

OProfile has support for setting up each counter to collect in user mode, kernel mode or both user and kernel modes. However, the architecture of the POWER performance counters does not allow the user and kernel mode to be set for each counter. Rather, all counters count in the same mode. If any of the counters have user mode specified for it, then all of the counters will count in user mode. The same is true for the kernel mode.

### **2.5 Hypervisor mode profiling**

By default, profiling in hypervisor mode is always enabled. There was no explicit control added to enable or disable hypervisor mode. Samples collected when the processor is running in hypervisor mode show up in the output listing under the **hypervisor\_bucket** entry. Typically, this should be a small percentage of the total samples.

### **2.6 Call graph support**

Call graph support for the POWER4, POWER4+, POWER5 and PowerPC 970FX processors was added to the kernel in version 2.6.17-rc1. Prior to this kernel there was no call graph support for these processors.

The call graph utility gives information on the function calls. For each function, the name of the caller function and the number of times it called the function is given. The name the various functions and number of times these functions were called from each function is given. See the OProfile manual for further details.

The gprof utility, as described in the paper “Contrasting Linux on Power Profilers” by Rob Nava, provides a call-graph capability, but it does incur a fair degree of overhead.

### **3 History of POWER4, POWER5 and PowerPC 970FX support for OProfile**

Anton Blanchard added basic support to OProfile for capturing time based profiles. This support used the performance counters to capture addresses every **N** cycles. No performance counter events could be used to collect a profile. Support for the event counters was added by Carl Love in early 2004 and was tested extensively internally in IBM.

After the changes were used and tested sufficiently within IBM, work was started to submit the kernel and user level changes to the community. This work was done by Carl Love and Maynard Johnson. Carl worked with Anton Blanchard to get the needed kernel changes submitted and accepted into the kernel. Maynard Johnson worked to get the changes to the user level tool accepted. This required a restructuring of how the event information was stored and processed, as well as adding the architecture-specific calls to the user code. These changes were included into the 0.8.2 version of the OProfile user code. Patches were pushed to SLES for Power 4 OProfile support in SLES9 rc1. The kernel changes were accepted into 2.6.9-rc3. In November of 2004, the kernel changes were accepted into SLES 9 and Red Hat REL 4, which followed the acceptance of all of the changes into the mainline kernel.

Currently, the tools chain group is responsible for providing long term support for the POWER specific support in Oprofile.

### **4 Implementation of Power support in OProfile**

The previous sections mentioned the differences in the POWER4, POWER4+, POWER5 and PowerPC 970FX support for OProfile. These differences include:

- how the Power events are programmed as part of a group of events
- how the user and kernel modes cannot be set individually for each counter
- sampling addresses in hypervisor mode is always enabled.

The user level OProfile code can be obtained from the OProfile home page. The code must be compiled and then the OProfile utilities are installed on the system. The details of this is described in the OProfile manuals. When the source code is downloaded and untar-ed, the files that specify the events are located in the **events/ppc64** directory. Within this directory, there is a subdirectory for each architecture. The source code directory, **events/ppc64/power4**, contains three files: **unit\_masks**, **events** and **event\_mappings**.

The files are copied to the default **/usr/local/share/oprofile/ppc64/power4** directory, or a location specified by the user when the tool is installed. These files support comments that start with the # symbol and continue to the end of the line.

## 4.1 The format of the performance counter event files

There are three files that are used to specify the events. The first file is the **unit\_masks** file. The second is the **events** file and the third is the **event\_mapping** file. The **unit\_masks** and the **events** files are standard files for all architectures. The third file is specific to the POWER4, POWER4+, POWER5 and PowerPC 970FX implementation.

### 4.1.1 The unit\_masks file

The **unit\_masks**, specifies additional qualifiers for the Intel architecture. The file is not used for the POWER architecture however, it must exist for compatibility with the OProfile tool. The file contents are as follows:

```
# ppc64 POWER4 possible unit masks
#
name:zero type:mandatory default:0x0
      0x0 No unit mask
```

Note that there is only one mask listed by the name of zero. The event file parser expects there to be a unit mask specified for every event.

### 4.1.2 The events file

The **events** file contains an entry for every event. The following is an example of an event entry:

```
event:0x10 counters:0 um:zero minimum:10000 name:PM_RUN_CYC_GRP1 :
(Group 1 pm_slice0) Processor Cycles gated by the run latch
```

The event consists of the following items:

- **event:** event number
- **counters:** counters that the event can be programmed in
- **um:** name of the unit mask for the event
- **minimum:** minimum reset count value
- **name:** name of the event
- description of the event.

The format supports specifying that the event can be programmed into one of several counters, or all counters. However, for POWER4, POWER4+, POWER5 and PowerPC 970FX, the event is defined to be in a specific counter by the group definition. There is a unique event number for every supported event.

On most platforms, the event number is the value that is programmed into the event control register to count the event. However, POWER4, POWER4+, POWER5 and PowerPC 970FX processors, the event number is used to determine the MMCR registers settings for the group that the event belongs to.

### 4.1.3 The event\_mappings file

The MMCR register setting for the event can be found in the **event\_mappings** file. An entry from the event mappings file is as follows:

```
event:0x10 mmcr0:0x00000D0E mmcr1:0x000000004A5675AC mmcra:0x00022001
```

The line consists of the **event** tag followed by the event number, the **MMCR0**, **MMCR1**, and **MMCRA** tags and their respective values. The MMCR0 and MMCRA values are 32 bits while the MMCR1 value is a 64-bit value. The MMCR values are identical for the event numbers that correspond to events from the same group. By convention, the **event\_mappings** and **events** files are organized by group.

It is possible to add or delete groups by editing the **events** and **event\_mappings** files. If the files are edited in the source code directory, the **make install** command must be run again to copy the files to the installed directory. By default the installed directory is **/usr/local/share/oprofile/ppc64/**, but the user can specify an alternate directory when the tool is installed. If the files are edited in the installed directory, the changes will be lost if the tool is reinstalled or upgraded.

## 5 OProfile user tool support for POWER4, POWER4+, POWER5 and PowerPC 970FX architecture

The data collection process for OProfile is done using the **opcontrol** bash script command. The script can be found in the **utils** source code subdirectory.

## 5.1 OProfile userspace code

The OProfile user code consists of the following components:

- User interface scripts and programs
- Profiler daemon, `oprofiled`

The user interface accepts input from the user, turning this input into commands written to the pseudo-filesystem at `/dev/oprofile`. The OProfile kernel driver responds to these commands to perform the profiling task. When profiling is complete, the daemon takes the buffered data from the kernel driver and persists it into the filesystem, creating the sample data. This sample data is post-processed by user interface programs that create reports and annotates source files.

## 5.2 User interface: `opcontrol`

OProfile's main user interface is the `opcontrol` script, which is used to initialize, set up, start and stop the profiler daemon, and start and stop profiling. Most of this script is architecture-independent with the exception of the `do_param_setup` function, which is invoked when the user runs the `opcontrol --start` command.

For each event passed into `opcontrol` by way of the `--event` parameter, the `do_param_setup` function obtains the following information about the event using the `ophelp` program:

- event value
- eligible hardware counters
- unit mask
- minimum count
- domain(s)
- architecture-specific information (optional)

The `ophelp` program has some architecture-specific code that obtains additional event mapping information. Currently, only POWER4, POWER4+, POWER5 and PowerPC 970FX needs this mapping, which consists of the MMCR values associated with a group of events. This mapping is returned by the `ophelp` program along with the event value and stored in the `EVENT_STR` local variable. This variable is checked by the `opcontrol` script function `check_event_mapping_data` to see if mapping values are present. If so, then all of the events specified by the user are validated to ensure they are in the same group.

At the end of the `do_param_setup` function, the architecture-independent event setup values are written to the pseudo-filesystem `/dev/oprofile` entry by way of the `set_ctr_param` function. Some of the architecture-independent values such as the unit mask is not used for the POWER4, POWER4+,

POWER5 and PowerPC 970FX processors. Then, if there are mapping values contained in the EVENT\_STR those are also written out to the /dev/oprofile entry as key/value pairs. The last piece of work done by the **opcontrol –start** command is to set the enable bit at the top-level of the /dev/oprofile entry. The OProfile kernel driver detects that profiling is enabled and uses the information from the /dev/oprofile entries to set up the performance counters to perform the desired profiling task.

### 5.3 OProfiled daemon

When the user stops profiling with the **opcontrol –stop** command, the top-level enable bit is cleared and profiling is stopped. The user then must invoke the **opcontrol –dump** command to get the sample data from the kernel. The profile daemon, oprofiled, takes the raw profiling data from the kernel driver and creates a sample file for each module or executable for which samples were collected. The sample files are stored in the /var/lib/oprofile/samples/current/ directory. By default, all samples for a given binary are stored in the same sample file.

The user can get separate sample files per thread, per application, per library or for the kernel using the following command prior to starting profiling:

```
opcontrol –separate=[none,lib,kernel,thread,cpu,all].
```

### 5.4 User interface: oprofile and opannotate

Once the profile is dumped, the user can generate reports from the sample files using the **oprofile** program. The user can get system wide reports or filtered reports that show only selected binaries.

Another tool, **oprofile-annotate**, is used to identify lines in source code or assembly listings where samples are taken. For source code annotation, the program must be compiled with debug information (for example, -g option for gcc and IBM's xlc/xlf compilers).

## 6 OProfile kernel code

The following discussion was written based on the code for the 2.6.14 and earlier kernels. Since the 2.6.14 kernel there been some changes. The most significant is that the path names changed from **arch/ppc64/oprofile** to **arch/powerpc/oprofile**.

The OProfile kernel code for POWER4, POWER4+, POWER5 and PowerPC 970FX is found in the **arch/ppc64/oprofile** directory. The key files are **Makefile**, **op\_impl.h**, **common.c**, **op\_model\_power4.c**. The code in the **op\_model\_power4.c** file is used for POWER4, POWER4+, POWER5 and PowerPC 970FX processors. The name is held over from the original architecture

supported in OProfile. Additionally, there is a file for the rs64 architecture that is not discussed in this report since event based profiling support was not added for this architecture.

The **op\_impl.h** file contains the definitions for several data structures and functions as described below.

- **op\_counter\_config** structure contains the parameters for each counter such as the event number, count, kernel/user mode, unit mask. The user and kernel fields are not used in this structure; they are there for compatibility with other processors.
- **op\_system\_config** structure contains the system-wide settings for the counters, the MMCR register values, kernel and user mode flags.
- **op\_ppc64\_model** structure is where the generic OProfile functions (start, stop, handle\_interrupt and so on.) are mapped to the Power architecture specific functions.
- **ctr\_read** is an inline function that reads the performance counters as defined in this file.
- **ctr\_write** is an inline function for writing values to the performance counters.

The **common.c** file defines the functions for starting and stopping the counters for the POWER4, POWER4+, POWER5 and PowerPC 970FX processors. When OProfile is started one of the first routines to run is the **op\_ppc64\_create\_files** function. The function creates files in the OProfile file system for passing data between the user and kernel. This file system is similar to the **/proc** file system in that there are files that can be read and written by the user level tool to pass information to and from the kernel. From the user side, a value is written to the file in the **/dev/oprofile** directory that corresponds to the value in the **op\_system\_config** structure to be set. The value written to the file is stored in the element of the **op\_system\_config** with that name. The values include the MMCR0, MMCR1, MMCR2 register settings. The directory also includes the enable\_kernel and enable\_user mode entries as well as other entries that are part of the standard OProfile implementation. The **oprofile\_arch\_init** function is also called to set the OProfile function pointers for the hardware mode, processor type and number of counters. The file contains the **op\_ppc64\_setup** function. This function reserves the performance counter hardware for use by OProfile, calls the **model->reg\_setup** function to compute the hardware register values, and calls the **model->cpu\_setup** routine. The **model->reg\_setup** is a function pointer to the **power4\_reg\_setup** function defined in the **op\_model\_power4.c** file. Similarly, **model->cpu\_setup** is a function pointer to the **power4\_cpu\_setup** function in the **op\_model\_power4.c** file.

The **op\_model\_power4.c** file contains the functions to set up the performance counter register values (**power4\_reg\_setup**), set up the power processors (**power4\_cpu\_setup**), start the counters, stop the counters (**power4\_stop**), capture the sample address, determine if the address is a user, kernel or a hypervisor address, and handle the performance counter interrupt **power4\_handle\_interrupt**.

The **power4\_reg\_setup** routine is passed the **sys** structure containing the value of the MMCR registers, user and kernel mode. This information is needed in the **power4\_cpu\_setup** routine however, the **sys** structure is not passed to that routine. Hence, it was necessary to create three

global variables to hold the MMCR settings that could be accessed by the **power4\_cpu\_setup** routine. The global MMCR registers are initialized with their corresponding value from the **sys** structure. The user and kernel mode bits are set in the MMCR0 register. The reset value for each counter is initialized in this routine.

The **power4\_cpu\_setup** routine takes the global MMCR values and sets the bit to stop the counters, enable interrupts on overflow and so on. The values are then written to the hardware register for the processor. At this point, the counters on each processor are configured to count the desired events, the counters are stopped, and the reset value for each counter has been set but the counters have not been started. This routine is called for each processor in the system.

The **power4\_start** routine is called to start the counters. The MMCR0 register is read first. The routine then loads the reset value into each counter that will be used to count the event. The user tool determines which counters will be used and passes the information using the **/dev/oprofile** file system. If the counter is not used, the reset value is set to 0 to minimize how often it will generate an interrupt. The bit in the MMCR0 register to enable counting is set and the updated MMCR0 value is written to the hardware register. The **oprofile\_running** flag is set. This routine is run by each processor.

The **power4\_stop** routine reads the MMCR0 register, changes the bit to stop the counters, and writes it back to the register. This routine is run by each processor.

There are three dummy routines defined in the **op\_model\_power4.c** file, **hypervisor\_bucket**, **rtas\_bucket**, and **kernel\_unknown\_bucket**. When the interrupt handler captures an address, the address could correspond to the processor running user, kernel, hypervisor or RTAS code. The interrupt routine attempts to determine the code region that was running when the sample was taken. If the interrupt routine can't determine the mode the processor was running in, the address is replaced with the address of the **kernel\_unknown\_bucket** function. Because this function is defined within the kernel space, this will be a valid kernel address for the post processor. The OProfile post processor currently has no way of mapping an RTAS or hypervisor address to its corresponding code. It can only deal with user and kernel addresses. Hence, if the address corresponds to one of these regions, the address is replaced with the address of the corresponding dummy function that is defined in the kernel space, making these addresses show up as kernel addresses. The important thing is the user can tell how many samples were captured while the processor was running hypervisor and RTAS code.

The **power4\_handle\_interrupt** routine handles the hardware interrupt. The hardware interrupt occurs when the most significant bit of the hardware counter register changes from a 0 to a 1. Note that the hardware counters are 32 bits wide. The hardware saves the program counter address at the time the interrupt occurs in a shadow register called the SIAR register. The first routine called by the interrupt handler is the **get\_pc** routine. The **get\_pc** routine reads the SIAR register. The address checked to see if it is a hypervisor address or an RTAS address and if so, the address is discarded and replaced with the corresponding dummy function address as described above. If the address was in user space, which in this context includes the kernel, the address is returned. If the address was some other region, the address is replaced with the address of the **kernel\_unknown\_bucket** dummy function. The interrupt handler then calls the **get\_kernel** function to determine if the

address returned by the `get_pc` routine is in the user or kernel space then the `is_kernel` flag is set to true; otherwise it is set to false.

After the interrupt handler determines the address range, it then proceeds to re-initialize the hardware. The PMM bit is reset to a 1 to enable counting of marked instructions. This bit gets cleared by the interrupt. The performance counters are now examined to see which counter overflowed. If the counter that overflowed does not have the enabled flag set to indicate that OProfile set up the counter with an event to profile on, the sample is not stored and the counter is reset to zero. If the counter is enabled, then the sample is saved by the `oprofile_add_pc` routine. The `oprofile_add_pc` function saves the address, the counter number and the `is_kernel` flag. The MMCR0 register is read, the enable interrupt bit is re-enabled, the enable counter bit is set and the updated MMCR0 value is written back to the hardware register.

## 7 Kernel changes needed to support new POWER processor

There are a couple of things that need to be done to add kernel support for a new POWER™ processor. The assumption is that the basic performance counter architecture remains the same. Specifically, the counter control is still done by the MMCR registers; the control and data register sizes do not change.

The `oprofile_arch_init` routine in the `common.c` file needs to have a new case statement entry for the new architecture. If it is just an update to an existing processor, then it might be possible to just add a new case entry and use the same code to set the number of counters, processor type and model entry. This is the only explicit place where the processor type is checked.

Some of the counter control bits moved from POWER4 to POWER5. Hence, it is important to check that the bits set by OProfile have not changed. If they do move, then code may need to be added to deal with setting the bits on an architecture-specific basis.

Additionally, there are some expected changes in the next POWER5+™ architecture as to which registers will be available for use. Some of the counters might be reserved for use by the hypervisor. If this happens, then new `evs` and `gps` files will be generated that will only use the available counters. From those files, the event and event\_mapping files will need to be updated. The user level tool will take care of only specifying counters in the legal range based on the new event file. It might be necessary to change the number of available counters in this case to insure that OProfile doesn't try to reset the value of a counter that has been reserved.

## 8 Current limitations in the POWER4, POWER4+, POWER5 and PowerPC 970FX support of OProfile

**Profiling the hypervisors.** OProfile only processes user and kernel addresses. There have been some discussions about extending this to include hypervisor addresses. This is a need given the

open source hypervisor project Xen. Currently, no details are available on how this would be done. Work would have to be done in the postprocessor to process the hypervisor addresses. One key issue is that there is no documented interface for mapping hypervisor addresses to function names.

**Call-graph support.** As mentioned earlier, call-graph support is only available in the 2.6.17-rc1 and newer kernels.

## 9 Summary

OProfile is a community-based tool for providing user and kernel code profiles. The profile is generated by periodically sampling the program counter. The performance counters are used to generate interrupts to cause OProfile to collect samples. Support for POWER4, POWER4+, POWER5 and PowerPC 970FX event-based profiling was added by Anton Blanchard from the kernel team, Maynard Johnson from the tools chain group and Carl Love from performance tools group.

The tool is still missing callgraph support. There is also a need to work with the OProfile maintainers to support samples from the both the Xen and IBM pSeries® hypervisors.

This tool is one of the primary tools used to identify the code where the processor spends time. It is important to continue to support this tool for the success of IBM's customers and benchmarking efforts.

## 10 Trademarks

The following are registered trademarks of the IBM Corporation in the United States and/or other countries:

IBM

POWER4

POWER4+

POWER5

POWER5+

PowerPC

pSeries

Linux is a trademark of Linus Torvalds in the United States and other countries.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.

## References

- [1] *OProfile manual* by John Levon, <http://oprofile.sourceforge.net>, 2004.
- [2] *OProfile Internals* by John Levon, <http://oprofile.sourceforge.net>, 2004.
- [3] *Tuning Programs with OProfile* by William E. Cohen, <http://people.redhat.com/wcohen>, 2004.
- [4] *Identify Performance Bottlenecks with OProfile for Linux on POWER* by John Engle, <http://www-128.ibm.com/developerworks/library/l-pow-oprofile/>, 2005.
- [5] *Contrasting Linux on Power Profilers* by Rob Nava, <http://www.ibm.com/developerworks/linux/library/l-pow-profilers.html>, 2006.