Writing Components for PAPI-C

Vince Weaver

vweaver1@eecs.utk.edu

16 July 2012

Abstract

This document describes the somewhat complicated task of writing components for PAPI-C.

Contents

1 getting started with PAPI

Check out PAPI git.

Built

check components

configure –with-components=”example rapl”

build

check components

2 where to start with modules

There’s an example component. Good for layout. *don’t* use it as a starting point. ALl it does is return a few never changing values. Not a good ref unless that’s what your component does.

If reading HW perf hardware: perf If reading system-wide value from /proc or /sys: If reading value that happens once per CPU If reading value that reads x86 MSR: If reading value from USB/serial: If reading in output of an executable tool:

2.1 configure scripts

Please don’t use autoconf/automake GNU configure scripts unless absolutely necessary.

Using configure makes things difficult if cross compiling or else if people plan to compile on one machine and move to another (which happens often on clusters that often have separate build nodes, or for distributions like RedHat that build RPMs and distribute them).

The proper way to make configuration changes is to haev an environment variable or some other way to change behavior. (See how vmware component enables/disables pseudo perf counters).

The only time configure might be necessary is when a header or library is needed to be linked to. Even in that case, if at all possible make this detected at runtime. Otherwise it is extra steps to build the component and most users won’t bother or else will break the build by enabling a component but forgetting to run configure (FIXME... can we make configure descend all dirs by default?).

3 Code Layout

Code style.

header file or not? CPU ones do to build system. Components do not, as mainly for sharing code and we aren’t sharing.

Make all that aren’t exported static.

Any that aren't static, prepend with _. This avoids collisions oin names, for example "example_init". There are more fancy things you should do to avoid exporting these symbols (see Redhat document).

strucs vs typedef.

4 Data Structures

5 Functions to Implement

5.1 .start

5.2 .stop

5.3 .read

5.4 .write

5.5 .reset

Be careful when implementing a _reset function. As of PAPI-4.4 PAPI_reset() only calls the underlying component _reset() when an EventSet is running. Otherwise the call is ignored. This is due to historical reasons.

5.6 .init_component

This function is called once, at PAPI_init() time.

The first thing this routine should do is detect if the hardware supported by your component exists. If it doesn't, it should return an error. It can also set _vector.cmp_info.disabled_reason so that a user of `papi_component_avail` can see why it is disabled.

This function should allocate any resources used by your component.

Typically it also sets up any native events you might have.

It should set _vector.cmp_info.num_native_events and _vector.cmp_info.num_cntrs.

5.7 .shutdown_component

Called at PAPI_shutdown() time. Should unallocate everything that was allocated in _init_component()

5.8 .init_thread

This function is called whenever a thread is initialized.

It should allocate and set up any per-thread info your component might have.

5.9 .shutdown_thread

Called at thread destroy time. Should deallocate everything done at _init time.

5.10 .init_control_state

Control state is an eventset. This is called any time a new eventset is created.

5.11 .ctl

5.12 .ntv_code_to_name

Takes an internal code; pass into it the name.

You probably should check ranges here in case it is out of bounds.

5.13 .ntv_code_to_descr

Takes an internal code; pass into it the description

You probably should check ranges here in case it is out of bounds.

5.14 .allocate_registers

This function is called when an event is added to an eventset.

It is meant for components where there might be constraints. You can check here if the new eventset will properly map to hardware, as well as to allocate any additional counter resources.

5.15 .update_control_state

This function is called whenever an EventSet has a change made. This is when an event is added, one is removed, or if cleanup eventset is called.

Also this is called at PAPI_start() time. This is inefficient but done to catch the case that other things can change the eventset state and PAPI doesn't rigorously remember to call it in all instances.

6 Tests

Always do tests. Otherwise it is hard for others to know if your code works, or even yout to know. ALso it lets PAPI devs know if some change they make breaks things.

7 Other

8 Submitting to PAPI