Performance Counters

What are performance counters?

Performance counters are registers that count occurrences of specific hardware events. They are particularly useful for tracking optimization and efficiency of a program throughout its runtime. When attempting to achieve soft real time performance, these metrics help a programmer track what performance is left on the table.

This tutorial will demonstrate how to use the PAPI performance counter library to integrate benchmarking in a program.

Installing PAPI

These steps are derived from the arm developer hub linked here:
https://learn.arm.com/install-guides/papi/

For RPI 4 Users
See the thread here on how to apply the patch for PAPI to work before continuing: patch

Prerequisite - Install perf:

```
sudo apt install linux-perf
```

1. Clone the PAPI repository:

```
git clone https://github.com/icl-utk-edu/papi/
cd papi/src
```

2. Configure and compile the source code:

```
./configure && make
```

3. Configure and compile the source code:

```
sudo make install
```

PAPI should be now be installed.

A brief description of how PAPI tracks counters:

PAPI uses “event sets” where hardware events are added prior to benchmarking. These counters usually measure the occurrences of specific hardware events such as cache reads/writes, cache hits/misses, branch mispredicts, total cpu cycles, and more. Since counters are often hardware-specific, PAPI includes roughly 100 generic default events that map to the host processors actual events. With the pi4, these were fully supported. However, the pi5 seems to

not map the generic events to the hardware specific ones. Luckily, it's easy to check if your hardware mapped the generic events, and if not, which native events are supported on your hardware.

To check which generic events are supported:
1. Go to the 'utils' directory in the papi repo

```
cd papi/src/utils
```

2. Run the 'papi_avail' executable:

```
./papi_avail
```

If generic events aren't supported, check the available native events:
1. Go to the 'utils' directory in the papi repo

```
cd papi/src/utils
```

2. Run the 'papi_native_avail' executable:

```
./papi_native_avail
```

Using the PAPI Library

Start off by initializing the PAPI library:

```
// setup PAPI library and make sure it's initialized
if (PAPI_library_init(PAPI_VER_CURRENT) != PAPI_VER_CURRENT) {
    exit(1);
}
```

Now initialize an integer as your event set:

```
int event_set = PAPI_NULL;
if (PAPI_create_eventset(&event_set) != PAPI_OK) { // create the event set
        exit(1);
    }
```

Adding Generic Events

If your processor supports the generic PAPI events, it's relatively straightforward to add them.

First create a list of the PAPI events you want to track with a separate array to track their corresponding values. In this tutorial, it's simply L1 misses, L2 cache misses and total instruction count.

```
int events[3] = {PAPI_L1_DCM, PAPI_L2_DCM, PAPI_TOT_INS};
long long values[3];
```

Now use a loop to add the events to the set:

```
// add each event counter to the set of measured events
    for (int i = 0; i < 3; i++) {
        if (PAPI_add_event(event_set, events[i]) != PAPI_OK) {
            exit(1);
        }
    }
```

Adding Native Events

If your processor doesn't support the PAPI events or you need a processor-specific native event, PAPI allows you to add native events to a PAPI event set. In the case of the Pi5, the perf hardware events need to be used instead of the papi generics.

First create a list of the native events and another array for their corresponding values:

```
char *events_str[3]  = {"perf::PERF_COUNT_HW_CACHE_L1D:MISS",
"perf::PERF_COUNT_HW_CACHE_LL:MISS", "PERF_COUNT_HW_INSTRUCTIONS"};
long long values[3];
```

Now create a PAPI event list and convert these events to PAPI-compatible events

```
int events[3];
for (int i = 0; i < 3; i++) {
    if (PAPI_event_name_to_code(events_str[i], &events[i]) != PAPI_OK) {
        exit(1);
    }
}
```

Finally add the events to the set:

```
// add each event counter to the set of measured events
    for (int i = 0; i < 3; i++) {
        if (PAPI_add_event(event_set, events[i]) != PAPI_OK) {
            exit(1);
        }
    }
```

Starting the Counters

To start the counters themselves

```
PAPI_start(event_set); // start the hardware event counters
```

Often a hardware timer is a useful metric alongside event counters. This is done simply with the following function call

```
long long start_time = PAPI_get_real_nsec(); // start a hardware timer to
measure execution time
```

Stopping the Counters and Seeing the Result

Stop the hardware counters and the timers, specifying the previously created values array to store the results:

```
/*Execution code*/

long long end_time = PAPI_get_real_nsec(); // stop the hardware timer

// stop the cache/instruction counters
if (PAPI_stop(event_set, values) != PAPI_OK) {
    fprintf(stderr, "Error stopping PAPI!\n");
    return 1;
}

double elapsed_time = (end_time - start_time) / 1e9; // calculate
execution time in seconds
```

The values array will have corresponding values to the events created in the events array used to make the event set. In others words, the number that the event at event[0] occurs is stored in values[0].

In the case of the array used in the tutorial, the results can simply be printed to stdout like so

```
// display the benchmark information
printf("L1 Data Cache Misses: %lld\n", values[0]);
printf("L2 Data Cache Misses: %lld\n", values[1]);
printf("Total Instructions: %lld\n", values[2]);
printf("Execution Time: %.9f seconds\n", elapsed_time);
```

Shutting down

Before ending the program, make sure to shutdown PAPI:

```
PAPI_shutdown();
```