RAPL

Before you use RAPL, be sure to call the rapl_init function. See 'general libmsr use' for more details.

Setting a Power Bound

1. Create a rapl_limit struct
2. Set the limits in that struct
3. Call the function to set the limit on the socket and domain you desire, pass in your limit struct

Setting RAPL Limits

```
struct rapl_limit limit1, limit2, dramlimit, pp0limit, pp1limit;
unsigned socket = 0;

limit1.watts = 95;
limit1.seconds = 1;
limit1.bits = 0;  // Leave this as zero, its explanation is beyond the scope of this article
limit2.watts = 120;
limit2.seconds = 3;
limit2.bits = 0;  // Leave this as zero, its explanation is beyond the scope of this article
set_pkg_rapl_limit(socket, &limit1, NULL);   // Set only the lower PKG limit on socket 0
set_pkg_rapl_limit(socket, NULL, &limit1);   // Set only the upper PKG limit on socket 0
set_pkg_rapl_limit(socket, &limit1, &limit2);  // Set both PKG limits on socket 0

dramlimit.watts = 50;
dramlimit.seconds = 1;
dramlimit.bits = 0;  // Leave this as zero, its explanation is beyond the scope of this article
set_dram_rapl_limit(socket, &dramlimit);  // Set the DRAM limit for socket 0

pp0limit.watts = 100;
pp0limit.seconds = 5;
pp0limit.bits = 0;   // Leave this as zero, its explanation is beyond the scope of this article
pp1limit.watts = 80;
pp1limit.seconds = 10;
pp1limit.bits = 0;   // Leave this as zero, its explanation is beyond the scope of this article
set_pp_rapl_limit(socket, &pp0limit, &pp1limit);  // Set the power planes limits for socket 0
```

Reading a Power Bound

This works the same as setting the power bound, but you call the respective 'get' function.

Getting RAPL Limits

```
struct rapl_limit limit1, limit2, dramlimit, pp0limit, pp1limit;
unsigned socket = 1;
get_pkg_rapl_limit(socket, &limit1, &limit2);   // Get both power limits for socket 1
get_dram_rapl_limit(socket, &dramlimit);        // Get DRAM limit for socket 1
get_pp_rapl_limit(socket, &pp0limit, &pp1limit);  // Get the power plane limits for socket 1
```

Reading Used Power

Note: The read_rapl_data function is no longer used for this. Now we use poll_rapl_data, which must be called twice on a socket to calculate watts/deltas.

There are two ways to do this.

Reading Used Power

```
struct rapl_data * rdata, * data1;     // In this example, rdata was passed into either rapl_init or
rapl_storage
unsigned socket = 0;
poll_rapl_data(socket, data1);   // Update the rapl data. Watts/deltas are relative to the last time
this function was called
dump_rapl_data(data1, stdout);   // Display everything in data1.
// Since poll_rapl_data has only been called once, these should all be 0
dump_rapl_data(&rdata[socket], stdout); // This will display the same data as the previous call

poll_rapl_data(socket, data1);   // This will calculate Watts/deltas relative to the last
poll_rapl_data call
dump_rapl_data(data1, stdout);   // Display everything in data1. This time, there should be values
for watts
dump_rapl_data(&rdata[socket], stdout); // This will display the same data as the previous call
```

The rapl_data Struct

This struct contains tons of data.

struct rapl_data

```
struct rapl_data{

    uint64_t old_pkg_bits;  // holds the bits previously stored in the MSR_PKG_ENERGY_STATUS
register

    uint64_t ** pkg_bits;   // holds the bits currently stored in the MSR_PKG_ENERGY_STATUS register

    uint64_t old_dram_bits;   // holds the bits previously stored in the MSR_DRAM_ENERGY_STATUS
register

    uint64_t ** dram_bits;  // holds the bits currently stored in the MSR_DRAM_ENERGY_STATUS
register

    double old_pkg_joules; // this holds the previous energy value stored in MSR_PKG_ENERGY_STATUS
register represented in joules

    double pkg_joules;   // this holds the current energy value stored in MSR_PKG_ENERGY_STATUS
register represented in joules

    struct timeval old_now;  // this holds the timestamp of the previous rapl data measurement

    struct timeval now;    // this holds the timestamp of the current rapl data measurement

    double elapsed;   // this holds the amount of time elapsed between the two timestamps

    double pkg_delta_joules;  // this represents the change in energy for PKG between rapl data
measurements

    double pkg_watts;    // this represents the change in power for PKG between rapl data
measurements

    uint64_t * pkg_perf_count;  // pkg performance counter

    uint64_t flags;

    // DRAM

    uint64_t * dram_perf_count; // this is a count of how many times dram performance was capped due
to imposed limits

    double dram_delta_joules;  // this represents the change in energy for DRAM between rapl data
measurements

    double dram_watts;   // this represents change in power for DRAM between rapl data measurements

    double old_dram_joules; // this holds the current energy value stored in MSR_DRAM_ENERGY_STATUS
register represented in joules

    double dram_joules;     // this holds the current energy value stored in MSR_DRAM_ENERGY_STATUS
register represented in joules

    // PP0
```

```
        uint64_t ** pp0_bits;
        uint64_t old_pp0_bits;
        double pp0_joules;
        double old_pp0_joules;
        double pp0_delta_joules;
        uint64_t * pp0_policy;
        uint64_t * pp0_perf_count;
        double pp0_watts;
    // PP1
        uint64_t ** pp1_bits;  // energy bits
        uint64_t old_pp1_bits;  // old energy bits
        double  pp1_joules;  // energy
        double old_pp1_joules;  // old energy
        double pp1_delta_joules; // delta energy
        uint64_t * pp1_policy;  // policy
        double pp1_watts;
    };
```

There is a centralized array of rapl_data structs used by RAPL. You can access it by using the rapl_data function.

rapl_storage

```
struct rapl_data * rapl = NULL;
rapl_storage(&rapl, NULL);
```

Related articles

- The Batch Interface
- RAPL
- General LIBMSR Use