The Batch Interface

Libmsr Version 0.2.1

This is an advanced feature, if you are just using Libmsr for power monitoring/limiting or performance monitoring, you probably don't need to know this.

When using Libmsr you may need to access a large custom list of MSRs. Doing this with the traditional pread/pwrite method has a lot of overhead for anything over 2 MSRs. Therefore, we have created an optimized way to access a list of MSRs.

Note: This feature is subject to change (it will probably become easier).

Note 2: There are currently 11 user batches available numbered USR_BATCH0 - USR_BATCH10.

Using a Batch

1. Create an array of references to the raw data using uint64_t
2. Allocate memory for the batch using allocate_batch and give it the batch number and a size
3. Load the array of references for your batch using load_core_batch. Pass in the MSR the array will have data for, the array, and the batch number.
4. For socket level registers use load_socket_batch. For thread level registers use load_thread_batch.

This will create a batch that reads MSR 0x19C on every core

Creating a Single MSR Batch

```
// Each pointer in this array will reference the data from an MSR
uint64_t ** val = (uint64_t **) malloc(num_cores() * sizeof(uint64_t *));
// This will allocate the space for a batch
allocate_batch(USR_BATCH0, num_cores());
// This will set all the pointers in the array val

// to reference the data for accessing msr 0x19C
load_core_batch(0x19C, val, USR_BATCH0);
```

This will create a batch that reads MSRs 0x309, 0x30A, and 0x30B on every thread

Creating a Multiple MSR Batch

```
uint64_t ** val1 = (uint64_t **) malloc(num_devs() * sizeof(uint64_t *));
uint64_t ** val2 = (uint64_t **) malloc(num_devs() * sizeof(uint64_t *));
uint64_t ** val3 = (uint64_t **) malloc(num_devs() * sizeof(uint64_t *));
// Note that you multiply size by the number of different MSRs in your batch
allocate_batch(USR_BATCH4, 3 * num_devs());

// Add MSR 0x309 to the usr batch and have the val1 array reference the data
load_thread_batch(0x309, val1, USR_BATCH4);
// Add MSR 0x30A to the usr batch and have the val2 array reference the data
load_thread_batch(0x30A, val2, USR_BATCH4);
// Add MSR 0x30B to the usr batch and have the val3 array reference the data
load_thread_batch(0x30B, val3, USR_BATCH4);
```

Reading and Writing with a Batch

```
// To read a batch and access its data

// Based on USR_BATCH0 in 'Creating a Single MSR Batch'

read_batch(USR_BATCH0);
int i;
for (i = 0; i < num_cores(); i++)
{
fprintf(stdout, "Data for MSR %x is %lx\n", 0x19C, *val[i]);
}
```

```
// To assign data to a batch and write
for (i = 0; i < num_devs(); i++)
{
val1[i] = 1;
val2[i] = 2;
val3[i] = 3;
}

// Based on USR_BATCH4 in 'Creating a Multiple MSR Batch'

write_batch(USR_BATCH4);
```

Deleting a Batch

Deleting A Batch
free_batch(USR_BATCH0);

Related articles

- PCI Configuration Registers (CSRs)
- The Batch Interface
- RAPL
- Performance Counters
- General LIBMSR Use