Measuring Energy Consumption with the Energy Measurement Library

A. Cabrera, F. Almeida, J. Arteaga, V. Blanco

HPC Group
<cap@pcg.ull.es>
Universidad de La Laguna
Spain

September 2014
This work was co-funded by the Spanish Ministry of Education and Science (through TIN2011-24598 project and through the FPU program) and the European Fund for Regional Development (FEDER).
It also has been developed in the framework of the NESUS European network COST-IC-1305 and the Spanish network CAPAP-H4.
Outline

1. Background
2. Energy Measurement Library
3. Overhead Experimentation
   - Hardware Configuration
   - Methodology
4. Results
   - Communications
   - Computation Experiments
   - Nvidia Tesla Measurements
5. Conclusions
Outline

1. Background

2. Energy Measurement Library

3. Overhead Experimentation
   - Hardware Configuration
   - Methodology

4. Results
   - Communications
   - Computation Experiments
   - Nvidia Tesla Measurements

5. Conclusions
Measurement Tools

Classification of measurement tools

- External devices
- Intranode devices
- Hardware counters
External Devices

Some examples

- Power meters
- Metered power distribution units (PDUs)
### External Devices

<table>
<thead>
<tr>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>No overhead</strong></td>
<td><strong>Coarse system-level data</strong></td>
</tr>
</tbody>
</table>
Intranode Devices

Highly customized tools that measure energy within a node

Some examples

- Linux Energy Attribution and Accounting Platform (LEA$^2$P)
- PowerMon2
- PowerPack

A. Cabrera, F. Almeida, J. Arteaga, V. Blanco

Measuring Energy with EML
Intranode Devices

Advantages
- More accurate (per-component) data

Disadvantages
- Scalability
- Cost
Hardware Counters

Hardware providing consumption data through an API

Some examples

- Nvidia Management Library (NVML)
- Intel Running Average Power Limit (RAPL)
- Intel Manycore Platform Software Stack (MPSS)
## Hardware Counters

### Advantages
- Abstraction
- Simplicity
- Precision

### Disadvantages
- Not always available
- **Heterogeneous interfaces**
Every tool has its own:
- Software interface
- Choice of metric (power/energy)
- System of units
- Adequate sampling rate

Standards are needed!
...why not read energy events from PAPI?
...why not read energy events from PAPI? (they didn't exist)
Performance API (PAPI)

Can now access some hardware energy counters

Limitations

- Scope limited to hardware counters
- Lower-level abstraction
1 Background

2 Energy Measurement Library

3 Overhead Experimentation
   - Hardware Configuration
   - Methodology

4 Results
   - Communications
   - Computation Experiments
   - Nvidia Tesla Measurements

5 Conclusions
EML abstracts away tool-specific details

- Specific software interface calls
- Quantity reported (instant power vs cumulative energy)
- Units reported
- Sampling rate

It also provides convenient data acquisition and exporting
Goals

- Portable instrumentation
- Convenient data acquisition
- Low overhead
- Easy to extend
- Open source
EML was first implemented as a C++ library based on the factory method pattern

**Shortcomings**

- No device discovery functionality
- C code difficult to instrument
EML has been rewritten in C and many issues addressed

**Additions**
- Run-time autodetection of supported measurement devices
- Further simplified model
- JSON exporting of raw measurement data
- Open sourced under the GPL

https://github.com/hpc-ull/eml
Current Implementation

Initial device support

- Intel CPUs Sandy Bridge and later (through Intel RAPL)
- Intel Xeon Phi from the host (through Intel MPSS 3.x)
- Nvidia Fermi and Kepler cards (through NVML)
Stopwatch-like instrumentation of relevant code sections
Launches data gathering threads
C-style encapsulation (opaque types with related functions)
```
#include <eml.h>
#include <stdlib.h>

int main() {
    emlInit();
    // get total device count and allocate result handles
    size_t count;
    emlDeviceGetCount(&count);
    emlData_t* data[count];

    emlStart();
    // ...do work...
    emlStop(data);
    // ...use data...
    emlShutdown();
}
```
for (size_t i = 0; i < count; i++) {
    double consumed, elapsed;
    emlDataGetConsumed(data[i], &consumed);
    emlDataGetElapsed(data[i], &elapsed);
    emlDataFree(data[i]);

    // query each device name to print it alongside results
    emlDevice_t* dev;
    emlDeviceByIndex(i, &dev);
    const char* devname;
    emlDeviceGetName(dev, &devname);
    printf("%s : %gJ in %gs\n", devname, consumed, elapsed);
}
emlStart();
for (int i = 0; i < N; i++) {
  emlStart();
  // ...do work...
  emlStop(inner_data[i]);
}
emlStop(outer_data);
Outline

1. Background

2. Energy Measurement Library

3. Overhead Experimentation
   - Hardware Configuration
   - Methodology

4. Results
   - Communications
   - Computation Experiments
   - Nvidia Tesla Measurements

5. Conclusions
The *Bejeque* Cluster

<table>
<thead>
<tr>
<th>Node</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 x Intel(R) Xeon(R) @ 3.20GHz (Sandy Bridge)</td>
</tr>
<tr>
<td>8 Cores each</td>
</tr>
<tr>
<td>20MB L3 cache</td>
</tr>
<tr>
<td>64GB RAM</td>
</tr>
<tr>
<td>gcc 4.4.5-8</td>
</tr>
<tr>
<td>Intel MSR RAPL Interface</td>
</tr>
</tbody>
</table>
Performed Experiments

Total Energy

SYSTEM

SOCCKET (Processor)

CORE

RAPL0

RAPL1

8 cores per Socket
The Bejeque Cluster

GPU

- 1 x Nvidia Tesla M2090
- 512 cores @ 1.3 GHz
- 6 GB of GDDR5 Memory
- CUDA 4.1
- NVML interface
Methodology

Instrumented applications

- OSU Microbenchmarks for communication overhead
- Matrix multiplication implementations
  - Sequential
  - OpenMP
  - CUDA shared memory arrays
  - CUDA global memory
Methodology

Process

1. Code was instrumented with EML calls
2. Both instrumented and non-instrumented versions executed through *eml-consumed*
   - wrapper reporting a command’s consumption similar to the Unix *time* command
Instrumented code for the Sandy Bridge experiments

```c
#include <omp.h>
#include <eml.h>

void matmul_omp(float *C, float *A, float *B, int N) {
    int i, j, k;

    #pragma omp parallel for private(j, k) shared(A, B, C, N)
        for (i = 0; i < N; i++)
            for (j = 0; j < N; j++)
                for (C[i*N+j] = 0.0; k = 0; k < N; k++)
                    C[i*N+j] += A[i*N+k] * B[k*N+j];
}
```
int main(int argc, char *argv[]) {
    // ... Matrix initialization ...
    size_t count;
    emlInit();
    emlDeviceGetCount(&count);
    emlDevice_t* devices[count];
    emlData_t* data[count];

    emlStart(); // EML Measurement Start
    matmul_omp(C, A, B, N); // Matrix Mult
    emlStop(data); // EML Measurement Stop

    // ... Data postprocessing ...
    emlShutdown();
    return 0;
}

A. Cabrera, F. Almeida, J. Arteaga, V. Blanco
Measuring Energy with EML
#include "common.h"
#include "matrix_common.h"
#include <eml.h>

__global__
void matmul_kernel(float *C, float *A, float *B, int N) {
    int i = blockIdx.y * blockDim.y + threadIdx.y;
    int j = blockIdx.x * blockDim.x + threadIdx.x;

    if((i<N) && (j<N))
    {
        C[i*N+j] = 0;
        for(int k = 0; k < N; k++)
            C[i*N+j] += A[i*N+k] * B[k*N+j];
    }
}
int main(int argc, char *argv[]) {
    // EML Preparation
    size_t count;
    emlInit();
    check_error(emlDeviceGetCount(&count));
    emlDevice_t* devices[count];
    emlData_t* data[count];

    // .. Matrix and CUDA Initialization ..

    emlStart();
// Memory allocation
HANDLE_ERROR(cudaMalloc(&d_A, bytes));
HANDLE_ERROR(cudaMalloc(&d_B, bytes));
HANDLE_ERROR(cudaMalloc(&d_C, bytes));

// Host initializing
Initialize(A, N*BLOCK_SIZE, N*BLOCK_SIZE);
Initialize(B, N*BLOCK_SIZE, N*BLOCK_SIZE);

// Device initializing
HANDLE_ERROR(cudaMemcpy(d_A, A, bytes, cudaMemcpyHostToDevice));
HANDLE_ERROR(cudaMemcpy(d_B, B, bytes, cudaMemcpyHostToDevice));

dim3 dimBlock(BLOCK_SIZE, BLOCK_SIZE);
dim3 dimGrid(N, N);
CUDA Matrix Multiplication

matmul_kernel<<<dimGrid, dimBlock>>>(d_C, d_A, d_B, N*BLOCK_SIZE);

HANDLE_ERROR(cudaMemcpy(C, d_C, bytes, cudaMemcpyDeviceToHost));

// EML Measurement Stop
emlStop(data);

// .. Data Retrieving and memory deallocation..
emlShutdown();

return 0;
}
1 Background

2 Energy Measurement Library

3 Overhead Experimentation
   • Hardware Configuration
   • Methodology

4 Results
   • Communications
   • Computation Experiments
   • Nvidia Tesla Measurements

5 Conclusions
Performed Experiments

- Point to point communication (\textit{osu\_latency})
- Broadcast (\textit{osu\_bcast})
Comparison between instrumented and non instrumented versions

- Energy 1.53% higher for instrumented same socket latency
- Energy 0.60% higher for instrumented different socket latency
- Energy 1.09% higher for instrumented broadcast
Performed Experiments

- Sequential Matrix Multiplication
- OpenMP Matrix Multiplication
Sequential Matrix Multiplication

- Not very precise due to RAPL limitations
  - Matrix Multiplication uses 1 core
  - RAPL measures the entire socket
- Energy up to 10.59% higher for instrumented matrix multiplication
OpenMP Matrix Multiplication

- Very precise due to RAPL nature
  - Matrix Multiplication uses 16 cores
  - RAPL measures the entire sockets
- Energy up to 0.81% higher for instrumented matrix multiplication
Performed Experiments

- Global Memory Matrix Multiplication
- Shared Memory Matrix Multiplication

The executions are example codes provided by Nvidia instrumented with EML.
Global Memory Matrix Multiplication

- Constant energy overhead due to NVML calls
- Energy up to 9.53% higher for very low size problems (Absolute error, 9 Joules)
- Energy 0.99% higher for size of 8000 rows
Shared Memory Matrix Multiplication

- Same constant energy overhead due to NVML calls
- Energy up to 15.98% higher for very low size problems (Absolute error, 9 Joules)
- Energy 0.52% higher for size of 8000 rows
Comparative

- Sequential energy consumption is not comparable. Lack of precision.
- CUDA Examples consume much less than Sandy Bridge versions
- Cuda Shared memory is the less energy consuming (1487, 24 $J N = 8000$)
1. Background

2. Energy Measurement Library

3. Overhead Experimentation
   - Hardware Configuration
   - Methodology

4. Results
   - Communications
   - Computation Experiments
   - Nvidia Tesla Measurements

5. Conclusions
Conclusions

- EML is a practical tool for energy consumption analysis
- Low enough overhead to fulfill its role
Future Work

- Support for more devices (including out-of-node)
  - Metered PDUs
  - Instrumented mobile targets
- Integration with interposition techniques
- Complementary data postprocessing tools
THANKS

https://github.com/hpc-ull/eml