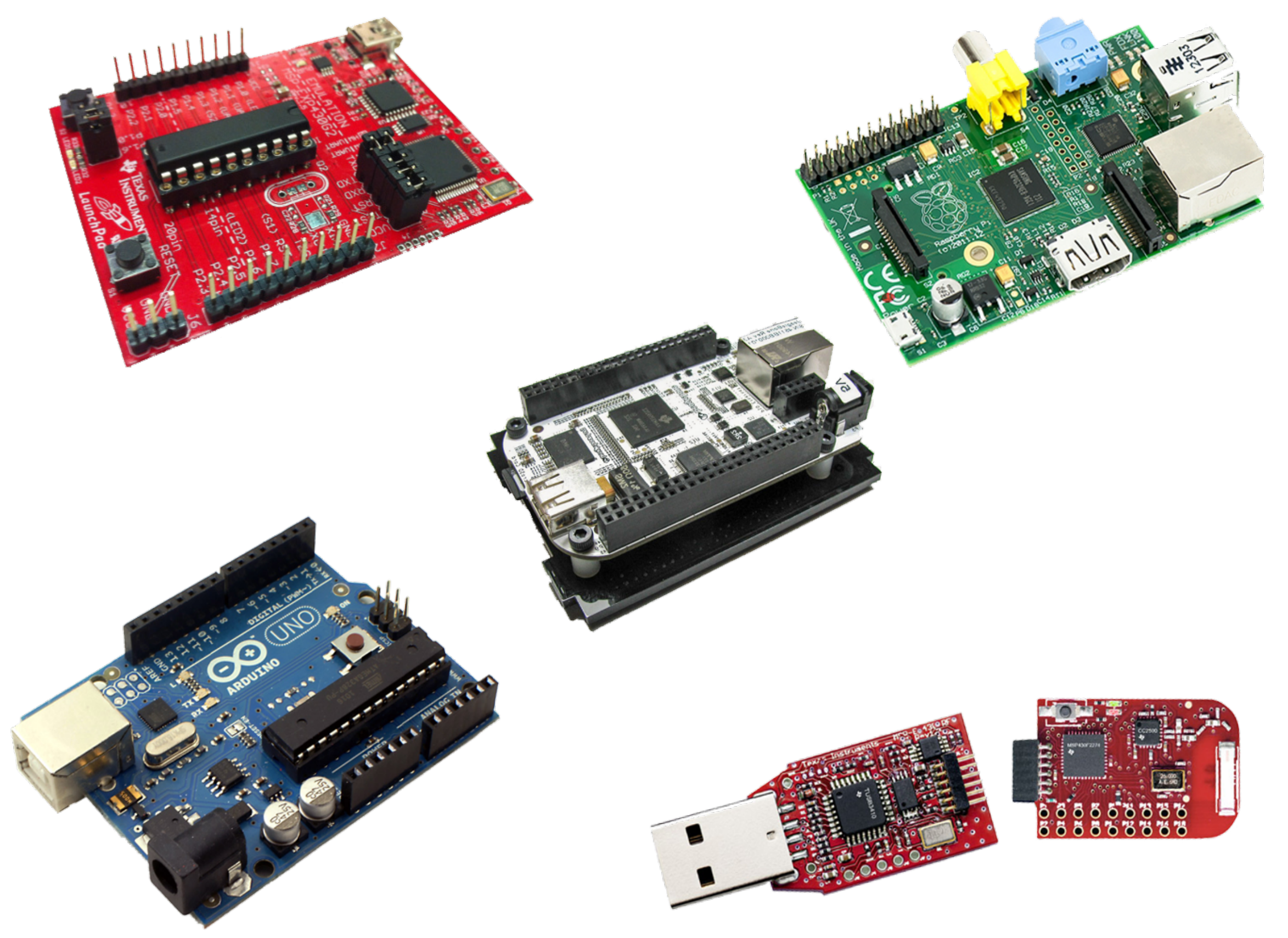
Application-Level Energy Modeling for Embedded Processors

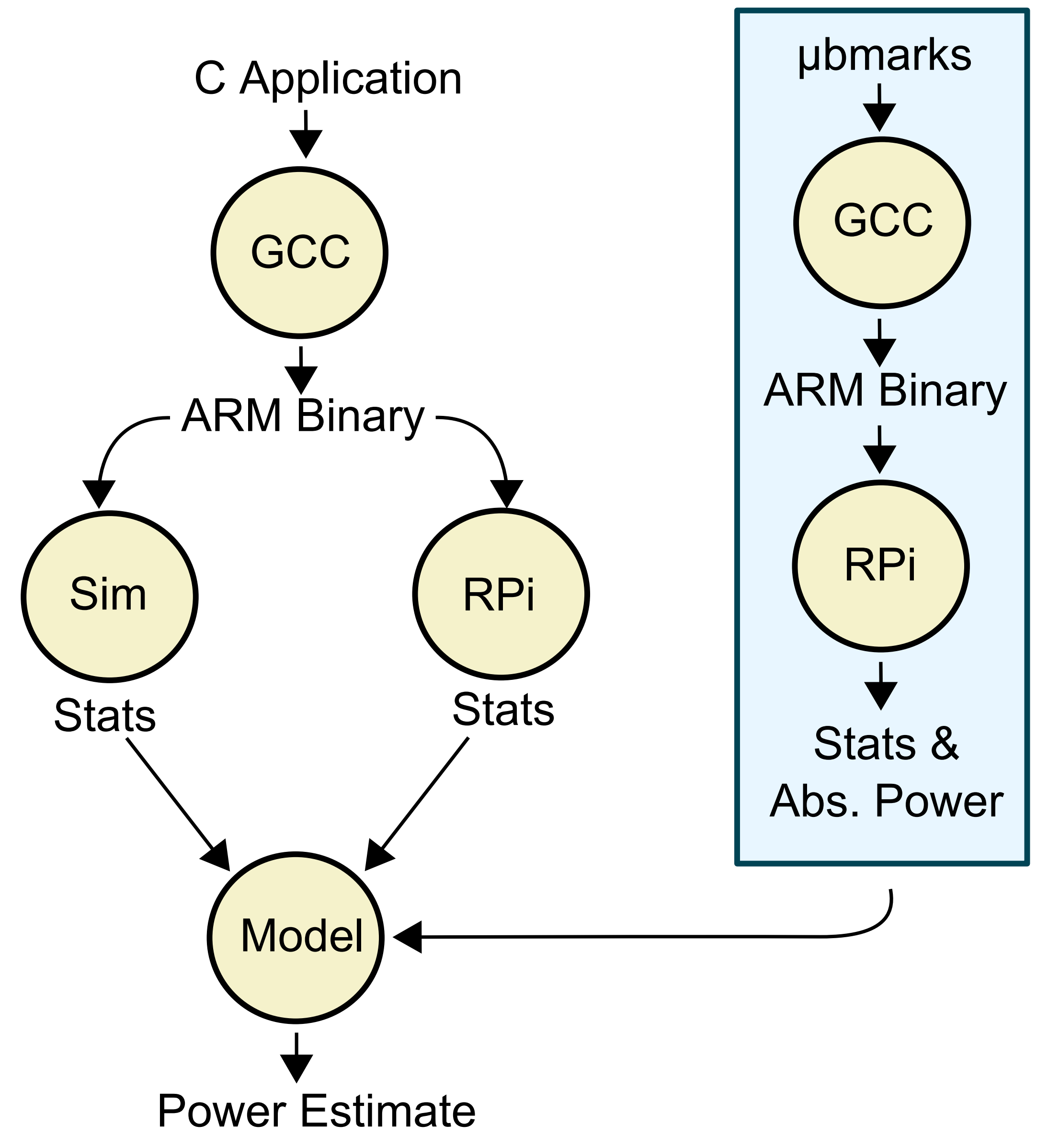
Einar Veizaga and Christopher Batten
Computer Systems Laboratory
School of Electrical and Computer Engineering

1 Motivation

Current computing devices have a high demand for both energy efficiency and performance. Although continued development has led to exponential increase in computational performance, high performance usually comes at the expense of energy efficiency, which is why designers are highly interested in studying this trade-off. Ingenuity in computer design, such as introducing multicore processors, is working to allow computational performance to develop unhindered by power constraints. To aid in the endeavor of achieving both high performance and high energy efficiency, this study focuses on techniques to **aid the development of more energy efficient software.**



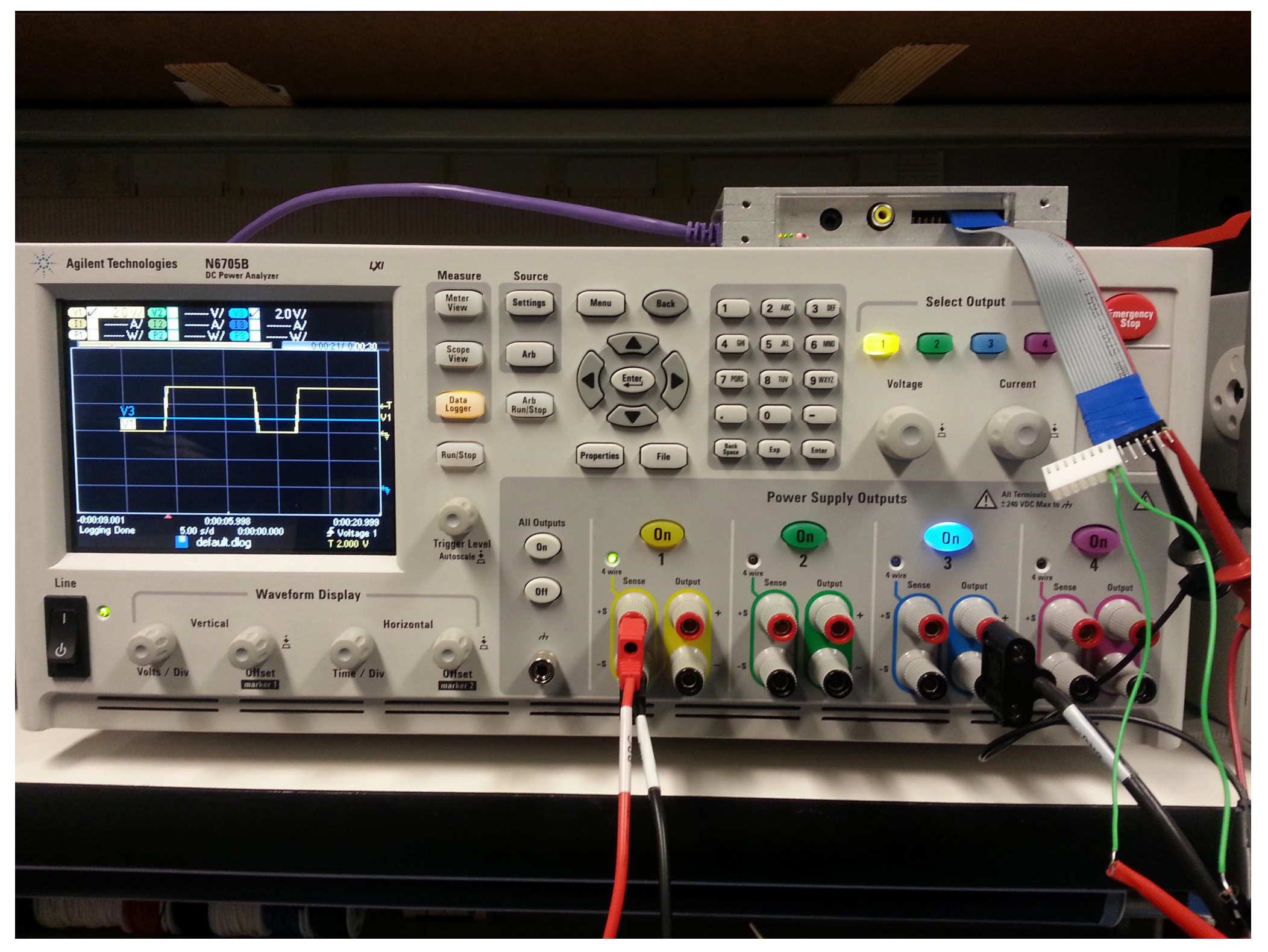
2 Methodology

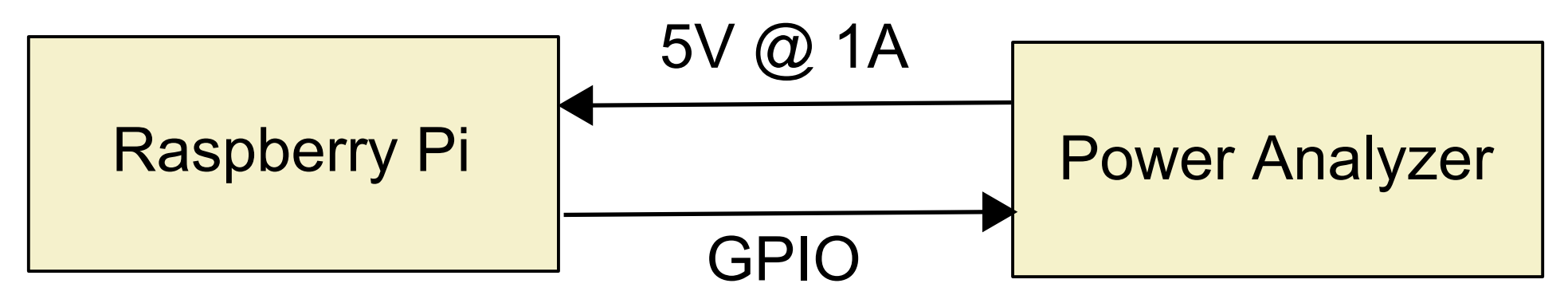


3 Experiental Setup

The study comprised of a Raspberry Pi containing the ARM1176JZF-S processor powered by an Agilent Technologies DC Power Analyzer. This allows us to collect detailed power consumption measurements of the Raspberry Pi as it runs.

We have a hardware-based and simulator-based setup which provides us all the information required to characterize the energy consumption of architectural events.

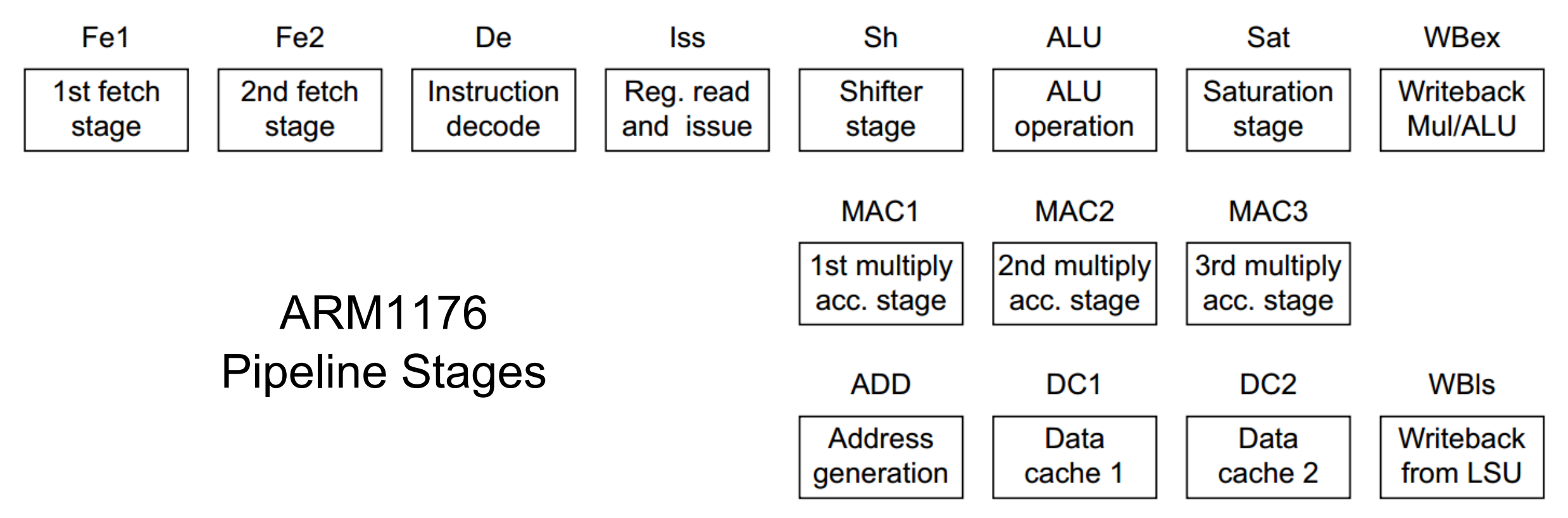




Hardware-Based Experimental Setup

The ARM1176 provides specialized hardware registers which can be configured as counters to measure hardware events such as cache misses and branch mispredictions.

These performance counters allow us to not only verify the validity of our energy benchmarks, but also provides the necessary statistics to calculate metrics such as cache miss ratios, data dependency stalls and branch mispredictions.



ARM1176
Pipeline Stages

Software-Based Experimental Setup

We use the the Gem5 processor simulator to measure dynamic instruction counts. The statistics provided by Gem5 allow us to take our energy table and begin to aggregate the energy usage per instruction or event. Thus we can provide an estimate of the energy consumption of the application.

4 Energy Microbenchmarks

We create microbenchmarks for all events we wish to characterize. For an add instruction, for example, we create a program whose runtime computation is comprised almost entirely of adds. Then, knowing the exact number of instructions executed as well as the average power of the program allows us to estimate the energy per instruction.

```
__attribute__((noinline))
int add()
{
   int i;
   for( i = 0; i < 10000000; i++ )
   {
   __asm__ __volatile__(" add  R0, R1, R2\n\t"
                        " add  R1, R2, R4\n\t"
                        " add  R2, R4, R5\n\t"
                        " add  R4, R5, R6\n\t"
                        " add  R5, R6, R7\n\t"
                        " add  R6, R7, R8\n\t"
                        " add  R7, R8, R9\n\t"
                        " add  R8, R9, R0\n\t"
                        " add  R9, R0, R1\n\t"
                        //...(100 instructions total)
                       );
   }
}
```

Energy per Event

The ARM ISA contains hundreds of instructions; for this study we focused on the instructions most preeminent in two matrix multiplication algorithms.

Instruction / Event	Energy per Instruction / Event	CPI
NOP	140 pJ	1
ADD	139 pJ	1
CMP	184 pJ	1
LSL	281 pJ	2
MOV	184 pJ	1
MUL	296 pJ	2
RSB	196 pJ	1
SUB	196 pJ	1
Branch Mispredict	1301 pJ	-
Cache Miss	93x10^6 pJ	-

Acknowledgements and References

I'd like to thank Christopher Batten, Ji Kim, Chris Torng, Berkin Ilbeyi, Shreesha Srinath, and the rest of the CSL team for providing invaluable guidance, advice, and support.

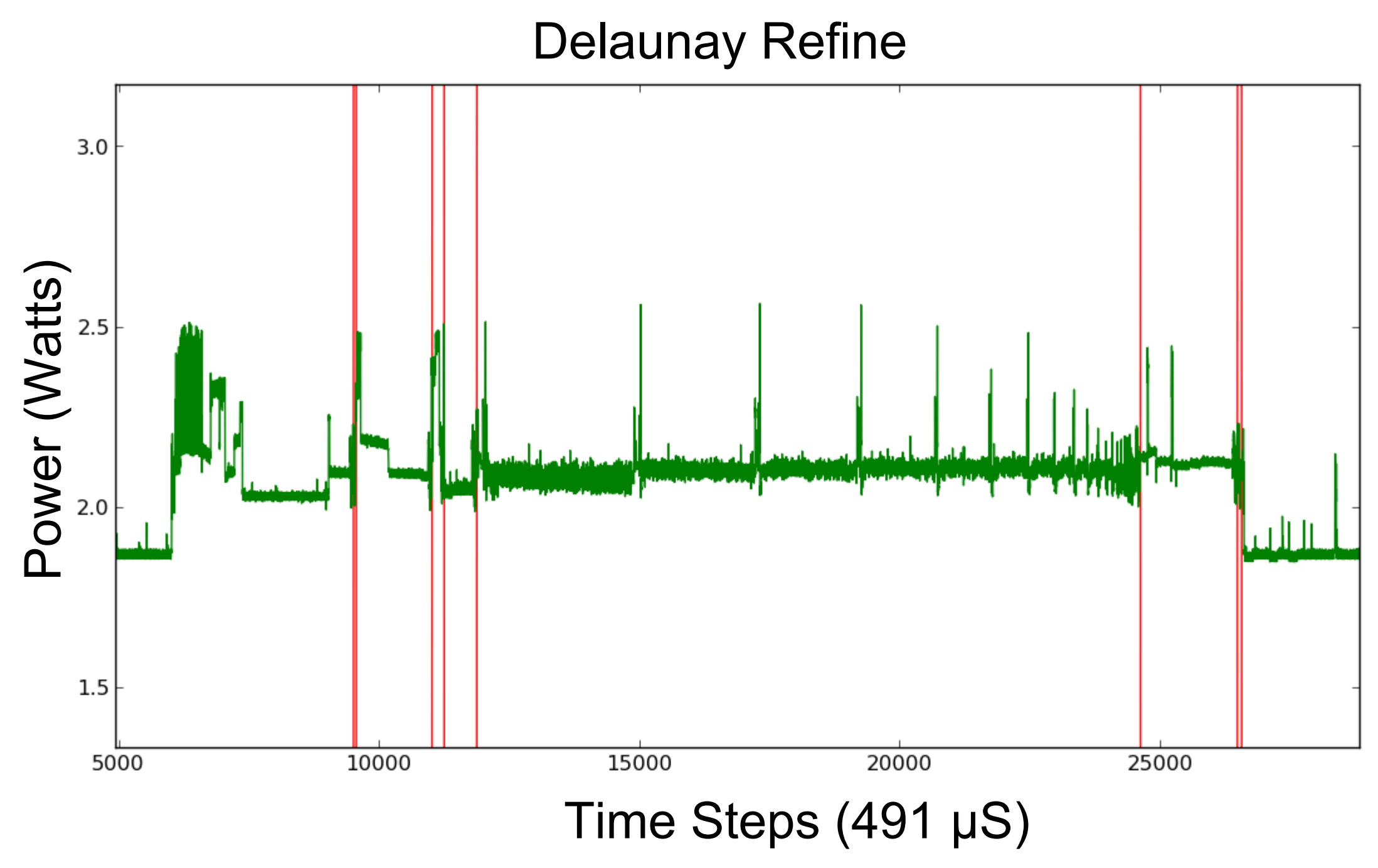
P.J. Drongowski, Sand Software Sound. http://sandsoftwaresound.net/

Stefanos Kaxiras and Margaret Martonosi. *Computer Architecture Techniques for Power-Efficiency*. 2008

ARM infocenter. http://infocenter.arm.com/

5 Results

Problem Based Benchmark Suite



The Delaunay refine algorithm is one of many applications that we are working to estimate the power consumption of. Software controlled GPIO signals allow us to delimit key areas of code in the power plot.

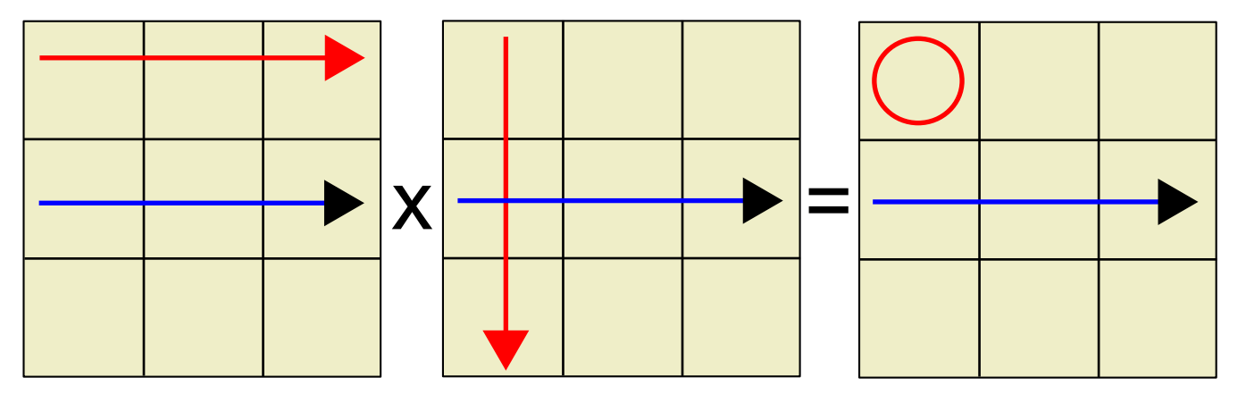
Matrix Multiplication

Naive.c

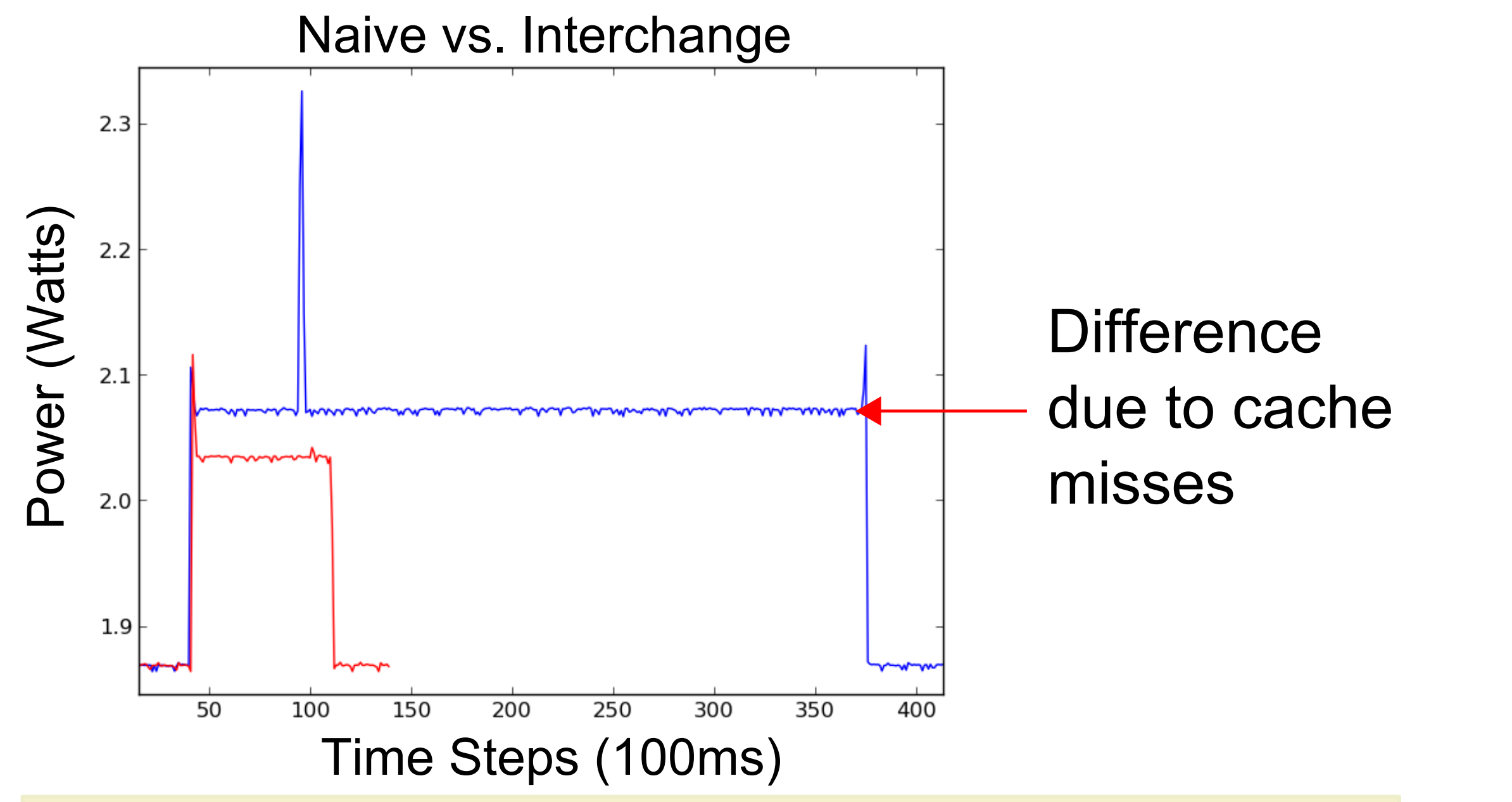
```
for (i = 0 ; i < MSIZE ; i++) {
  for (j = 0 ; j < MSIZE ; j++) {
    float sum = 0.0 ;
    for (k = 0 ; k < MSIZE ; k++) {
      sum = sum +
           (matrix_a[i][k] * matrix_b[k][j]) ;
    }
    matrix_r[i][j] = sum ;
  }
}
```

Interchange.c

```
 for (i = 0 ; i < MSIZE ; i++) {
   for (k = 0 ; k < MSIZE ; k++) {
     for (j = 0 ; j < MSIZE ; j++) {
       matrix_r[i][j] = matrix_r[i][j] +
         matrix_a[i][k] * matrix_b[k][j]) ;
     }
   }
 }
```



Row major order takes advantage of spatial locality



	Measured	Estimated	%Error
Naive	7.01 J	7.5 J	7%
Interchange	1.10 J	1.25 J	13%