# Group 2 Project– Final Report

# Data Movement and Power Measurements

# CS533 Spring 2018

## Portland State University

## June 9, 2018

Group Members:

* Alex Kelly <keap@cs.pdx.edu>
* Shikha Shah <shikha2@pdx.edu>
* Ajinkya Shinde <ajsh2@pdx.edu>
* Jason Graalum <jgraalum@pdx.edu>

GitHub Repo: <https://github.com/jasongraalum/CS533_Spring2018_Group2_Project>

# Introduction

In the world of PBs and TBs data, if power is not consumed in an appropriate way, it’s harmful in all aspects like performance, energy loss, cost of storage and contributes highly towards environment pollution. It is always assumed that with increase in data, power consumption is more, but in actual it depends on what type of operation is performed on the data you consist of and what type of memory accessed to perform the same.

In a typical “pipeline” process, data move is from processor to processor where different operations are performed. The assumption is that there is an efficiency gain by allowing the processors (or cores) specialize – think automobile assembly line. However, there may be a flaw in this thinking. As each core is general purpose, the efficiency gains, if any, from specialization, may be small. Given a large enough data set, it may be more efficient to move the operations, instead of the data, from core to core. Our project hence here tests this idea using raspberry pi which helps to avoid the noise while the actual code is running on OS.

# Goal

This project intends to write the benchmark codes which states in what situation power consumed is high with respect to data movement. And Performance is calculated with respect to the number of instructions executed and operation performed with respect to time.

# Hypothesis

It is assumed that as the number of context switches increase, amount of power consumed also increases and similarly if the computation fits within cache, the power consumed should be very less compared to the consumption when the execution has the transfer of information from cache to main memory.

Hence, given a large enough data set, relative to the size of the local memory (local being relative itself), the power consumption will decrease if the operation is moved from between cores as compared to moving the data between cores.

# Methodology

There are two main challenges to evaluating the power usage with in a system. The first is finding a measurable metric which is directly related to the case to be evaluated. In our case, measuring the power due to data movement between the core, L1 cache, LLC and main memory must be done indirectly. This is because we do not have physical access to the busses or the ability to physically isolated them. The second challenge is due to the complex interaction between the core, CPU, computer hardware system and the operating system. Each of these impact the total power of the system and introduces “noise” in our power measurement. An attempt to account for and eliminate this noise is difficult. To address these two challenges, we made several methodology decisions.

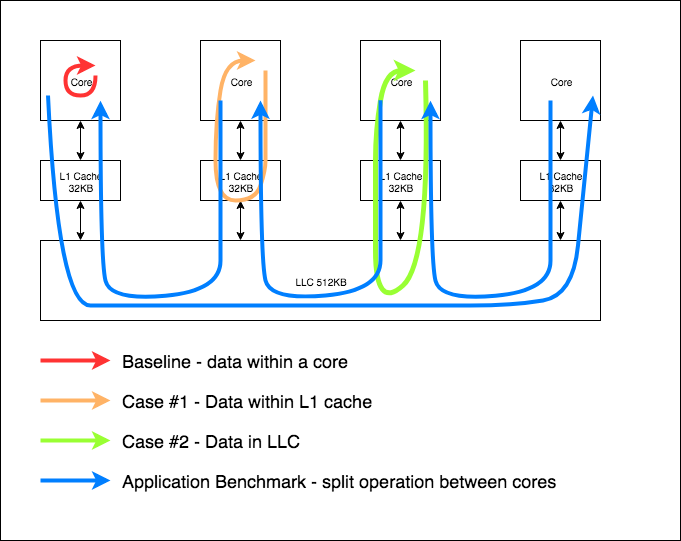
The first decision needing to be made was which hardware platform to use. We decided on the Raspberry Pi 3 mainly due to access. Also, using a system which we had complete control over allowed us to experiment with the operating system and job control. For instance, we were able to disable many of the background jobs running to help eliminate some of the noise in the power measurements. Also, we were able to hook up the power source to a power meter to directly measure the system power. Finally, the Raspberry Pi 3 has well defined(and small) cache sizes which helped limit the size of the overall testcases. The hardware and software specifications are listed below

#### Hardware Platform: Raspberry Pi 3[[1]](#footnote-1)

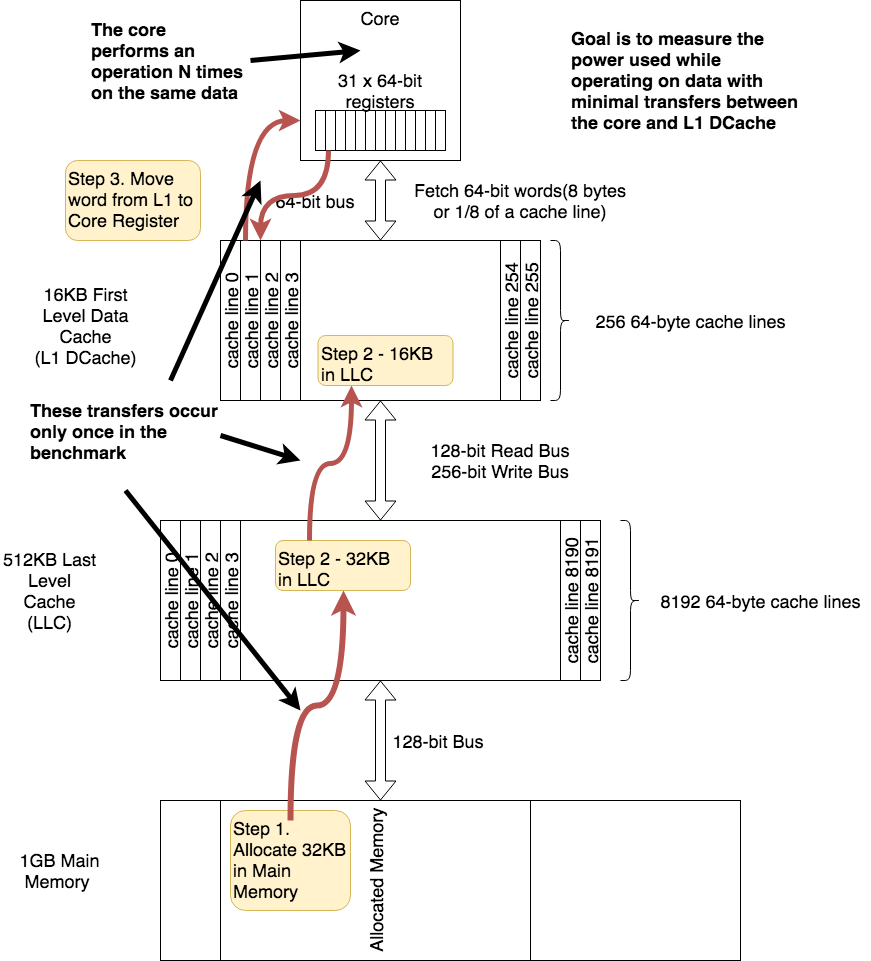
* SoC: Broadcom BCM2837
* CPU: 4 × ARM Cortex-A53, 1.2GHz
* GPU: Broadcom VideoCore IV
* RAM: 1GB LPDDR2 (900 MHz)

#### Software Configuration

* Kernel – Linux 4.14.34-v7+
* Customer built[[2]](#footnote-2),[[3]](#footnote-3)
* Add performance and power counter support
  + Need to capture all the details.



The second decision was which metrics to measure. An obvious choice was the total system power. We used a simple USB power meter as the Raspberry Pi 3 is powered through a standard Micro-USB port. The downside to using such a power meter is the relatively lower resolution. Combined with the fact that the Raspberry Pi 3 draws a low amount of quiescent power introduced even more noise into our measurements. Finally, we selected a set of system counter metrics to evaluate the operation of the cores. Using the Linux “perf” tool allowed us to record the number of instructions and the number of cache loads and misses per testcase. (In addition to “perf”, we also utilized the “schedtool” Linux tool. This tool allowed us to set the priority(niceness) of the running testcase to the highest level of 39. This decreased the chance that other jobs might interrupt a running test which would add additional noise to our measurements.



The final decision was which instructions to use to show significant differences in power consumption between the different data movements. We expected that there is increased power consumption due to loading data from the caches. Four test cases were decided on which would test four different data movements.

The first three test cases attempt to implement simple data movements between the core, the L1 cache and the Last Level Cache(LCC). We start with a baseline testcase(case #1) which attempts to limit the total data movement by operating on data currently resident in the core. The power measured during this test case should be limited to the quiescent power – i.e. idle power plus the power used in the operation. To be an effective baseline, all setup and operations must be consistent across all test cases. This means that we execute the same computation, which in our case is a simple addition. Also, the memory allocation must be consistent. This memory allocation is dictated by case 3 which requires the most amount of memory.

Figure : Test Case #1 Data Movements

The next two cases measure the power between the core and L1 cache and the core, L1 cache and LLC respectively.

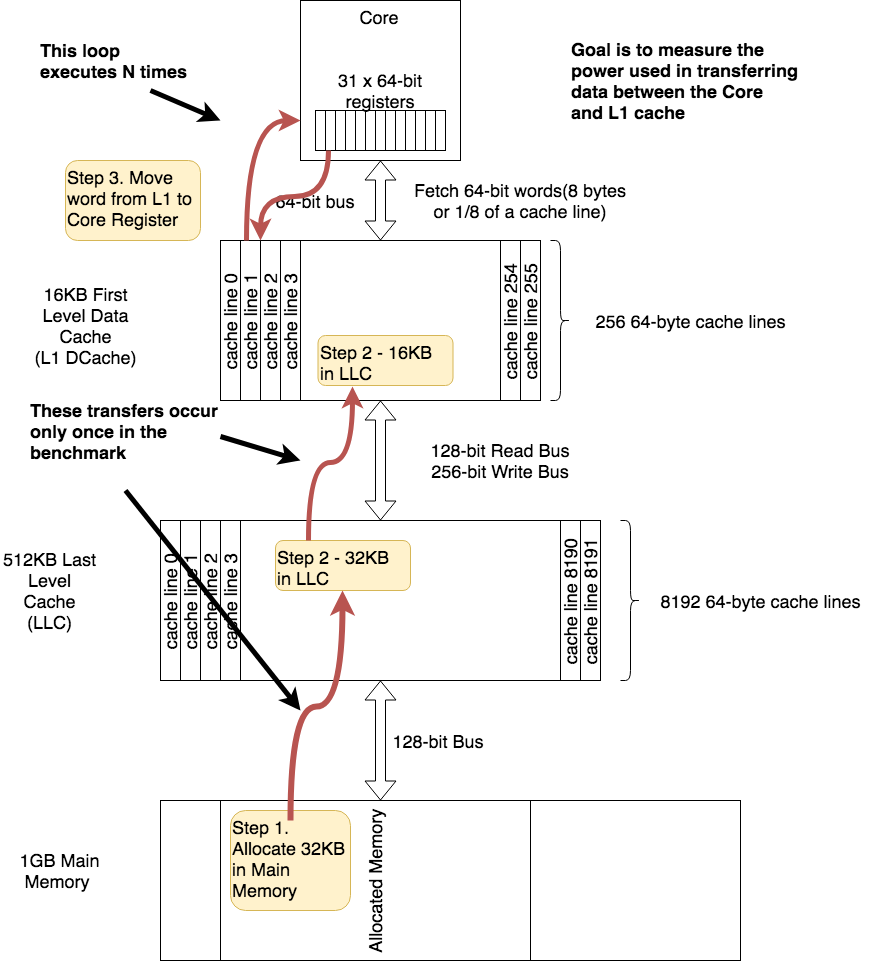
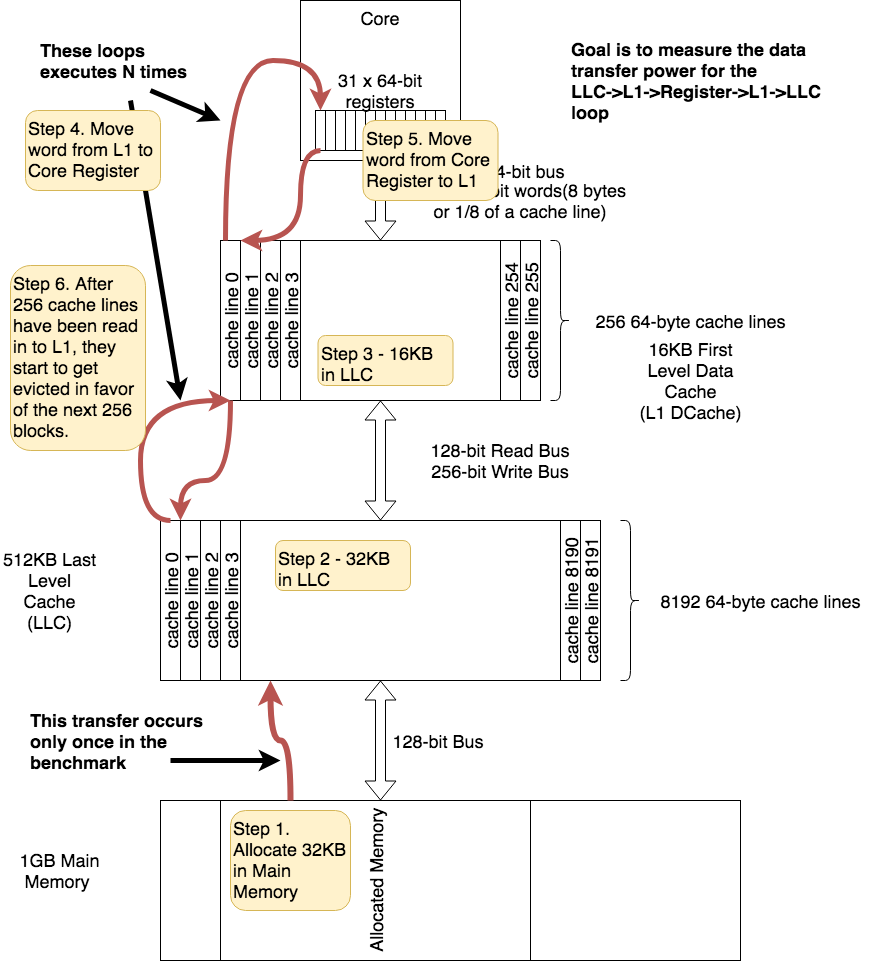


Figure : Test Case #2 Data Movement

To test the L1 cache power in case #2, we must force the loading of memory between the L1 cache and the core registers. To achieve this we load from a unique location for each loop of the test case. As shown in the diagram to the left, on each loop iteration, data is loaded into the core from alternating L1 cache lines. Because we limit the access from just two cache lines, we expect that there will be limited data moved between the L1 cache and LLC.



For case 3, we need to force a L1 cache miss so that a new cache block is loaded from the LLC on each iteration. To do this, we need to be sure to access a memory location from a different L1 cache block on each loop iteration. We accomplish this be allocating an array that is twice the size of the L1 cache. (This is the amount of memory allocated for all testcases so that we are working with the same baseline.)

Figure : Test Case #3 Data Movement

We need twice the size of L1 cache to guarantee that every access causes an L1 cache miss. On the first half, each cache line is loaded from the LLC into the L1. Each increment the array offset by the size of a cache line to ensure that we are accessing a different cache line. On the second half of the iterations, each cache line is replaced and loaded with a new block. Therefore on every access, a new cache line should be loaded from the LLC.

1. https://www.raspberrypi.org/magpi/raspberry-pi-3-specs-benchmarks/ [↑](#footnote-ref-1)
2. https://www.raspberrypi.org/documentation/linux/kernel/building.md [↑](#footnote-ref-2)
3. https://www.raspberrypi.org/documentation/linux/kernel/configuring.md [↑](#footnote-ref-3)