Power And Data Movement

GitHub: https://github.com/jasongraalum/CS533_Spring2018_Group2_Project

- Alex Kelly
-Ajinkya Shinde
-Jason Graalum
-Shikha Shah

Goals:

- Write benchmark codes that target specific configurations expected to be power efficient or power inefficient.
- Conduct a study using the benchmarks to actually measure the power consumption.

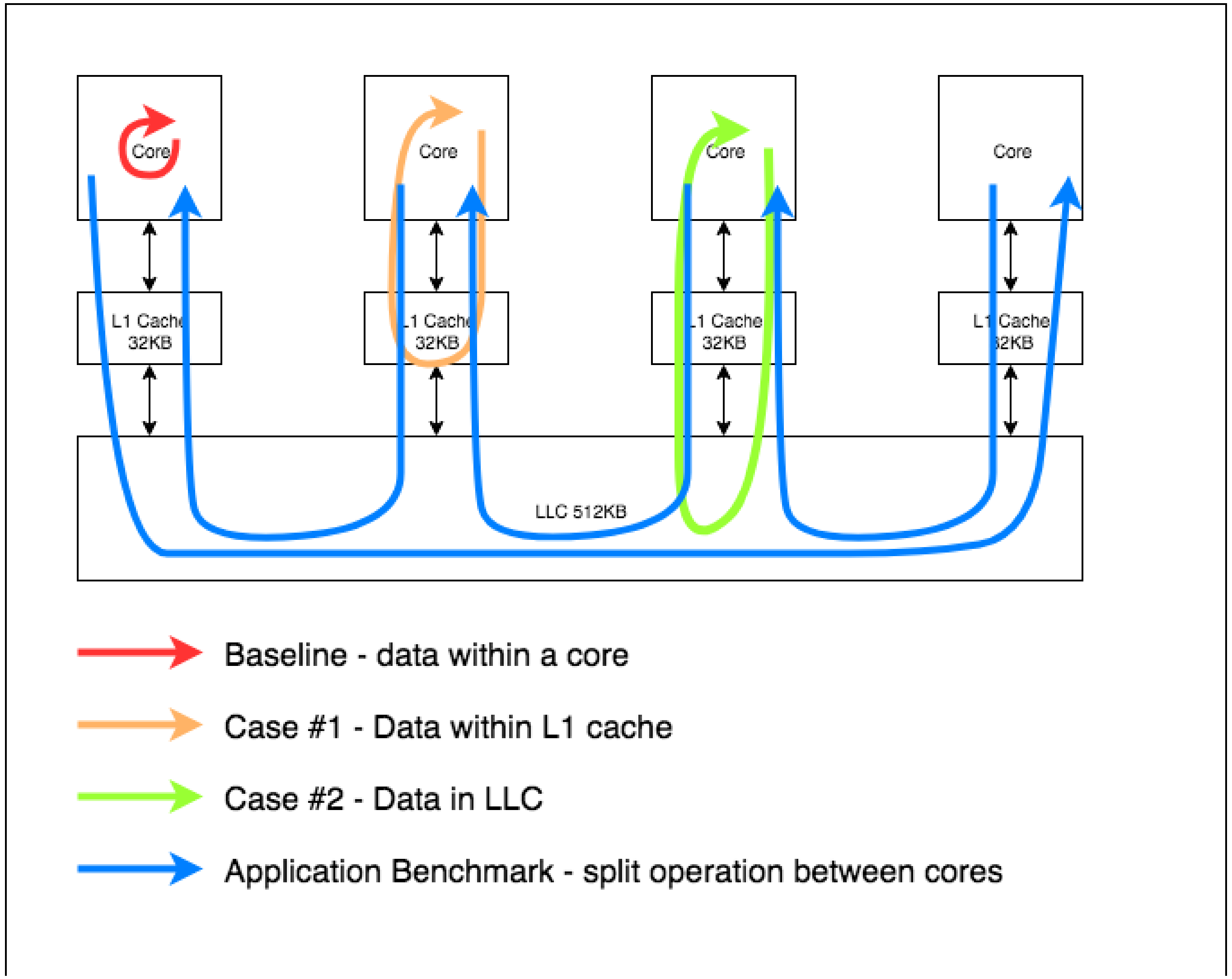
Hypothesis & Decision

- Assumed that power consumption increases with the increase in data movement and makes code less efficient with degradation in performance.
- Well, How to measure?

Methodology

Three Cases

- Baseline(red line)
  - Load data into core and operate
- L1 Cache Loads(orange line)
  - On each operation, load data from L1 to core
- LLC Cache Loads(green line)
  - On each operation, load data from LLC to L1 to core


Core
Core
Core
Core
L1 Cache 32KB
L1 Cache 32KB
L1 Cache 32KB
L1 Cache 32KB
LLC 512KB
Baseline - data within a core
Case #1 - Data within L1 cache
Case #2 - Data in LLC
Application Benchmark - split operation between cores

Data Movement

- Initialization
  - For all cases, we need a consistent configuration cost
  - Memory allocation and initialization
  - Array size == 2X L1 Cache Size
- Baseline
  - Simple operation(addition) on value in core registers
- L1 Cache
  - Alternate between two L1 cache lines
- LLC
  - Load from a new cache line on each access
  - Twice as many cache lines in the array as in L1
  - Guarantees that every load causes a L1 cache miss

Experimental Setup

- Used bash script to run multiple cases
- Took photo of power meter in between each run
- Found good use for Algorithms Textbook

out_case1.100.3.end.jpg
7/361 · 1280 × 720 · 572.2 kB · 48%
INTRODUCTION TO ALGORITHMS THIRD EDITION
BenQ

Real-World(ish) Data-Flow Cases:

What kind of data-intensive applications are common in the wild?

- Databases
- Analytics / Machine Learning
- Network traffic

How do different data-flow patterns in these cases affect power/performance?

How can we test something like this?

Big Data: Map, Filter, Reduce

Simple high-level algorithms for dealing with massive data asynchronously

One of the most popular cases of functional programming used in practice

Google and Amazon like these a lot

Used everywhere all the time

NLP: Markov-chain N-Gram

Very simple but powerful probabilistic natural language model

Build a probability model of all possible next words given a sequence of N words

Underpins a lot of machine translation, text synthesis, and sentiment analysis

Lots of data to crunch: Needs huge corpuses to be even remotely useful

Markov Ouroboros: “Realistic” test

- Process a big corpus of data (Tweets)
- Filter and Map to generate n-grams, then stack them with a reduce step
- Build a simplistic markov-chain bigram model with all the data
- Sample new text from the model and feed it right back through the first step

Markov Ouroboros: Why?

Moves a whole bunch of data around

Different kinds of data processing

- Aggregation
- Filtering
- Comparisons

Highly entropic data, so likely to cause cause cache misses

Markov Ouroboros: Cases

Baseline: Do it on one core

- How much power does just spinning up the python runtime take?

Parallel: Split the data, put it on 4 cores

- With full processor utilization, how much more power do we use?

Cascading: Move the data between cores

- Does moving the data around increase power consumption?

Markov Ouroboros: Details

Does the size/amount of data matter?

- Cache vs Memory

What about complexity?

- Will the power usage go down if the models regulaize the data?

Controlling for the disk read

What kind of processing do you cascade?

- Different steps?
- “Juggling” data?

Markov Ouroboros: “Results”

Working test case, but...

Power monitor stopped working

Only subjective results

Spoilers: Different processes are a hard test case to get right

But working test code which we will make available if anyone’s curious

Results

To perform the actual analysis , we use the **perf** tool

Collect statistics - vary loop iterations i. e 100, 400, 800 etc.

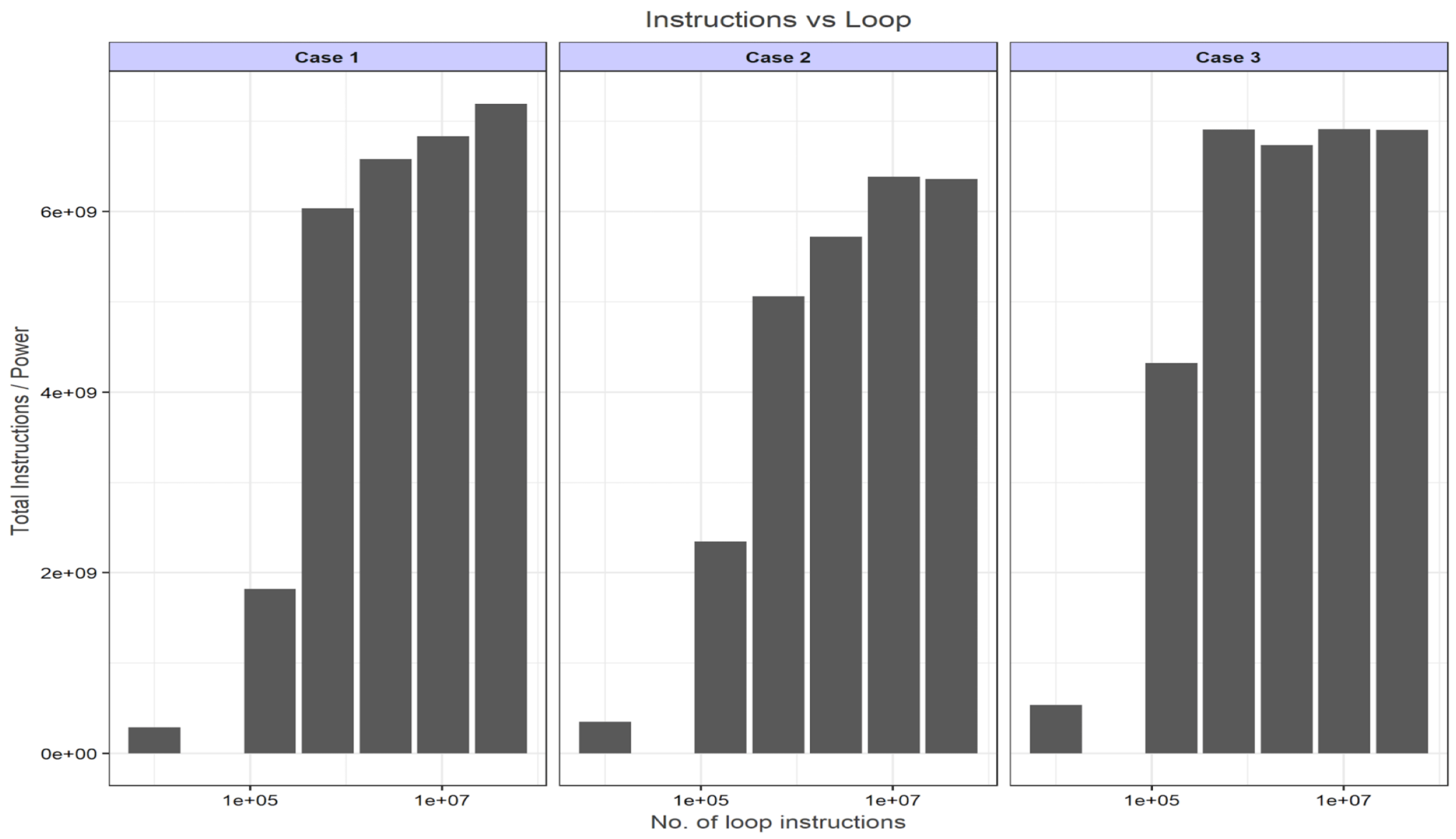
Identified key metrics for the analysis and plotted as :

1. Power consumed per instruct. in all of the cases.
2. Power consumed while data movement Main mem -> L1
3. Power consumed while storing data back to L1(on cache miss) .
4. Power consumed while data movement Mem -> LLC -> L1
5. Power consumed while storing data back to LLC (on cache miss)

Results - Analysis

Analysis - 1 : Power consumed per instructions after incr. in loop iteration

Should increase exponentially. Same are the results as shown below

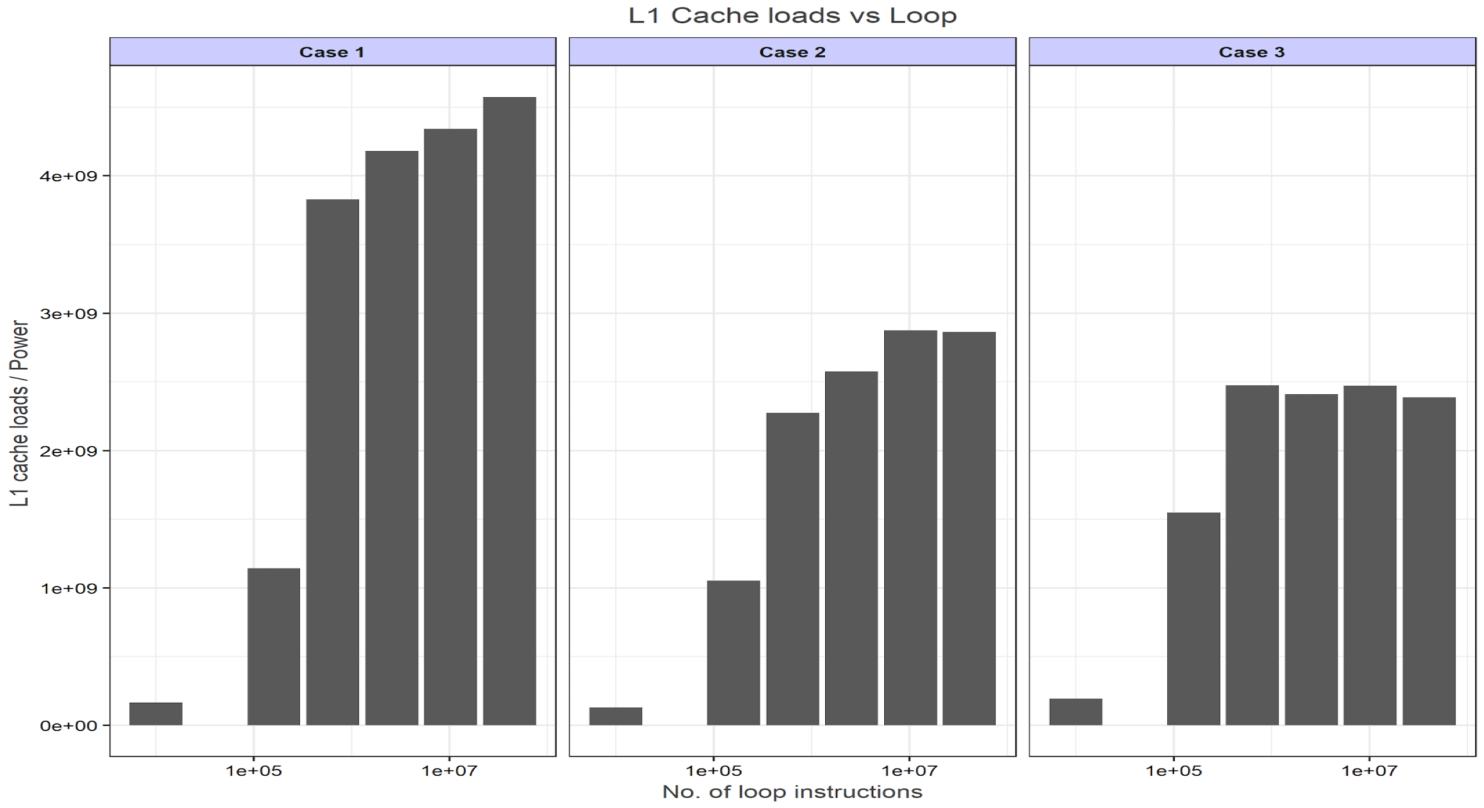


Results - Analysis

Analysis - 2 : Power consumed while moving data from main mem -> L1

For case 1 , there shouldn't be any change. With exp. incr. for case 2 & 3 .

However, the results are : -

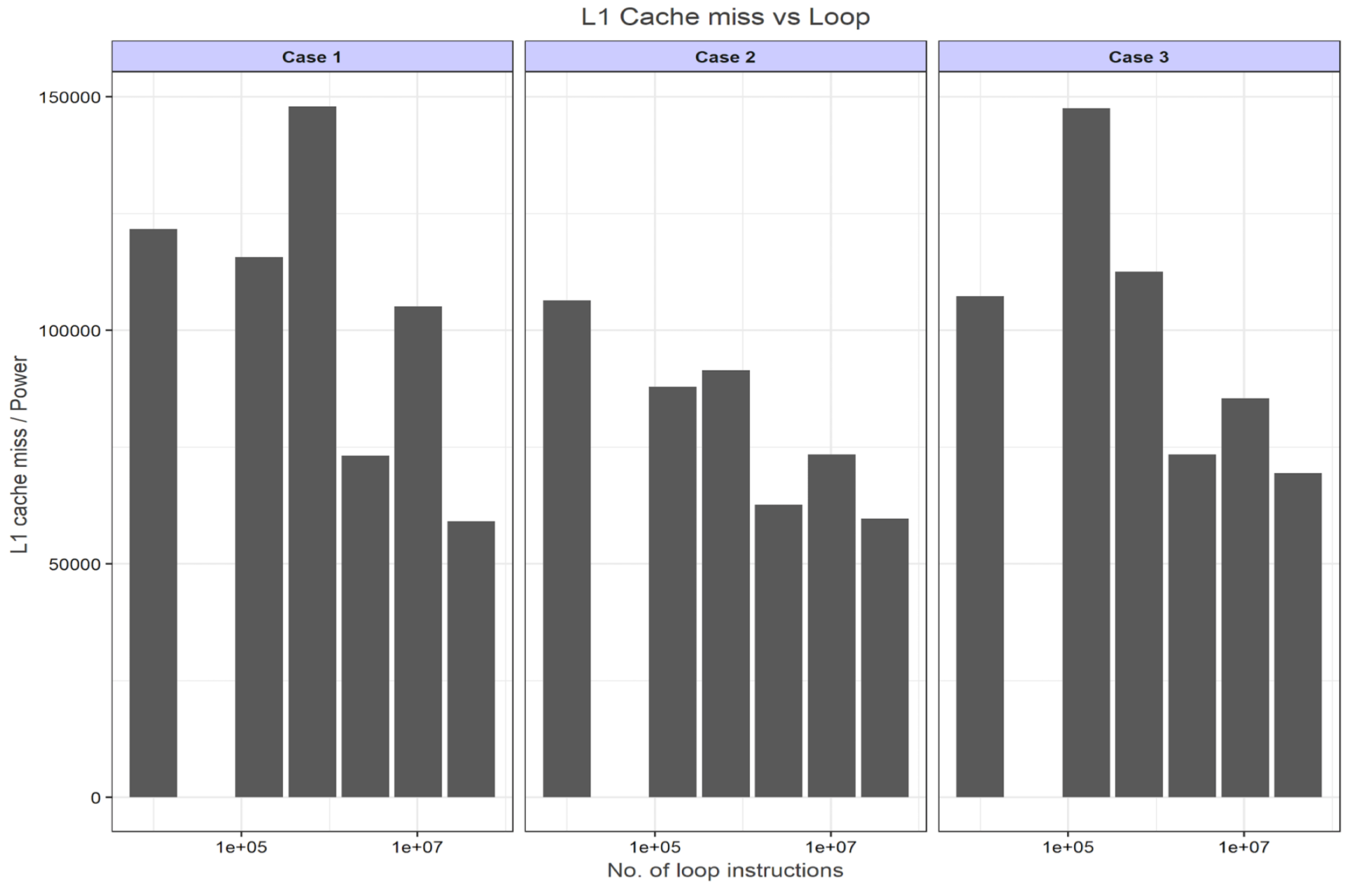


Results - Analysis

Analysis - 3 : Power consumed while storing data back to L1

Here, on cache miss , there shouldn't be any change in case 1 , with exp. incr for other cases.

However, the results are : -

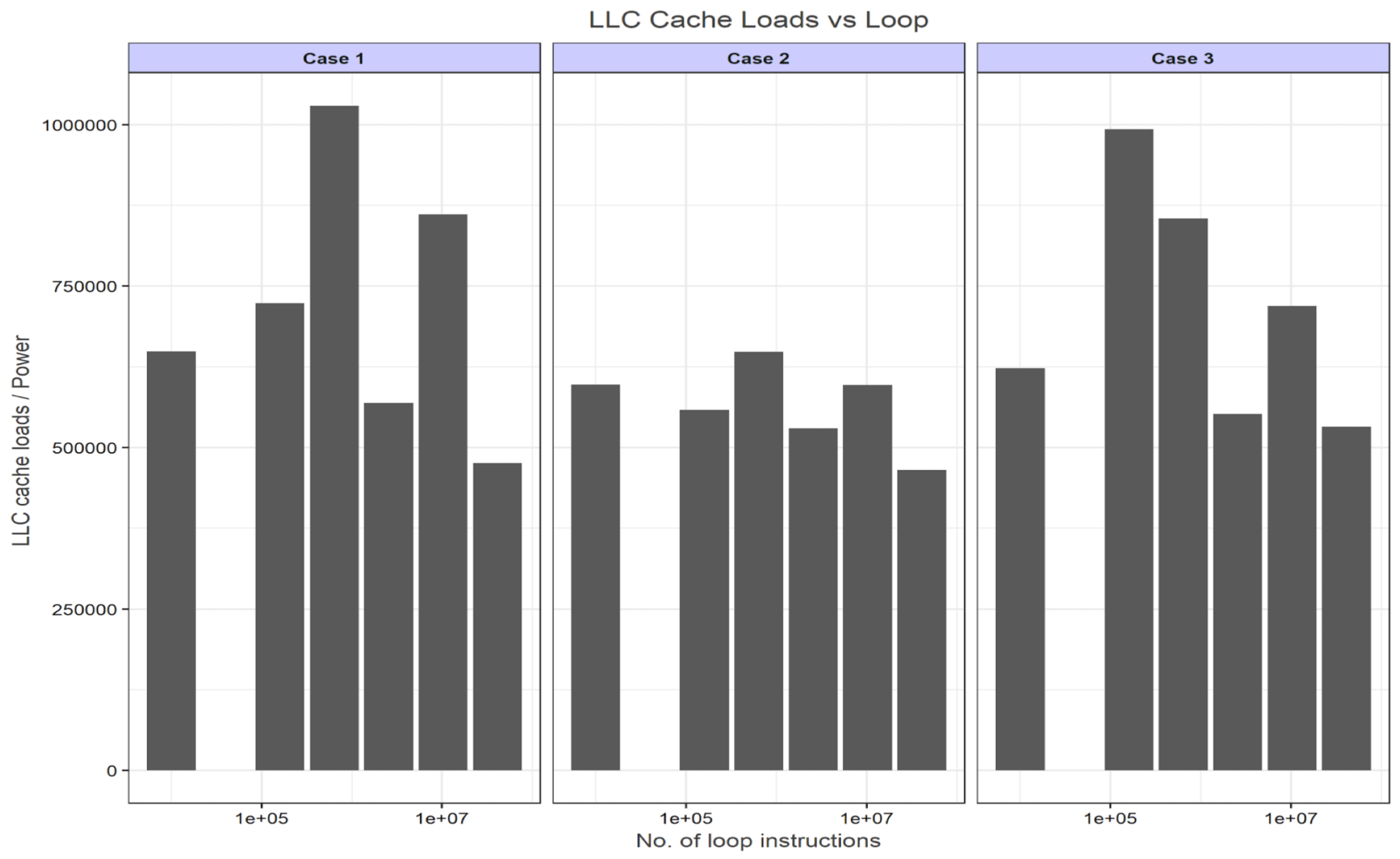


Results - Analysis

Analysis - 4 : Power consumed while data movement from Mem->LLC->L1

For case 1, 2 there shouldn't be any relative change , with exp. incr. in case 3

However, the results are : -

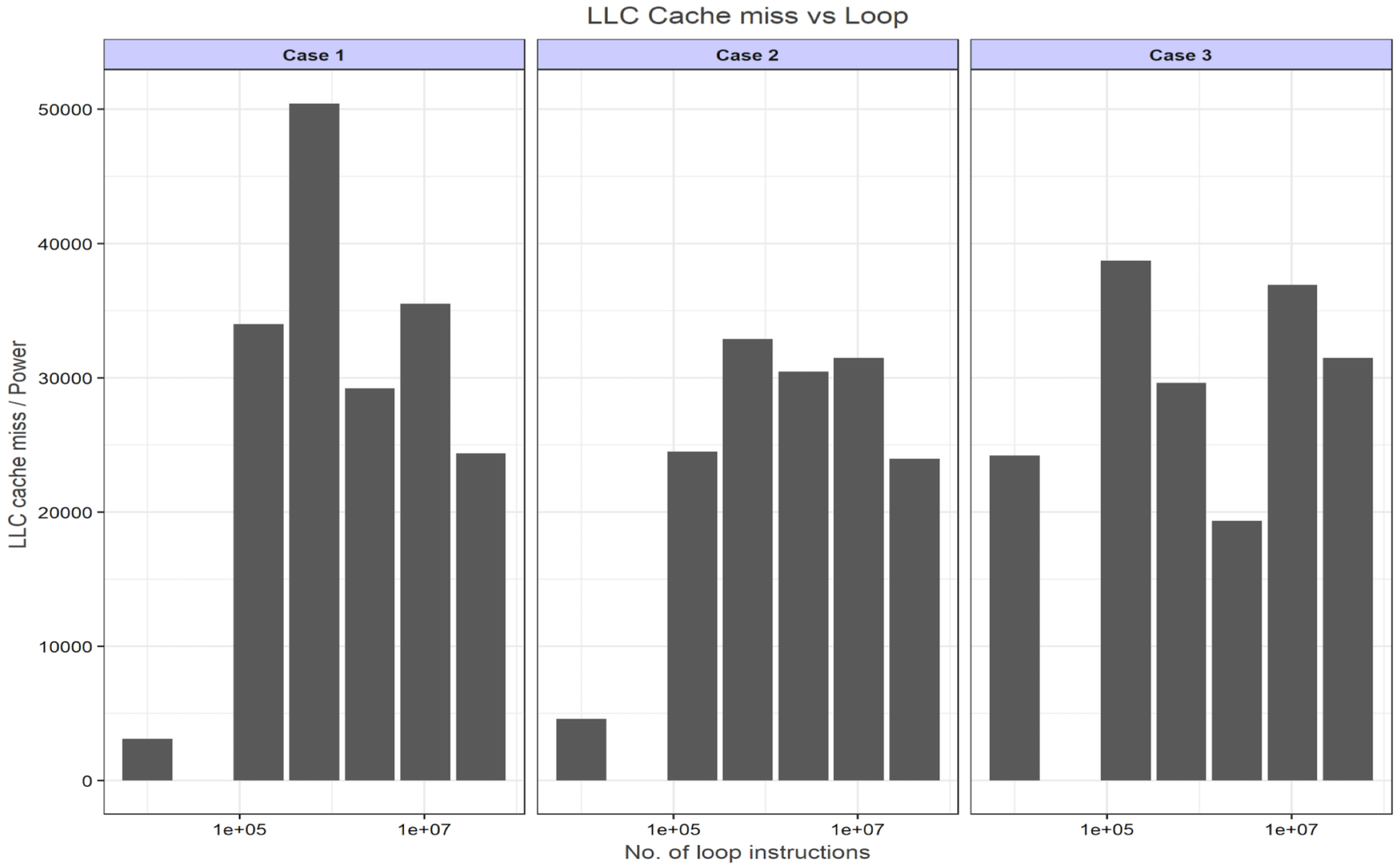


Results - Analysis

Analysis - 5 : Power consumed while getting data back at LLC ->Main Mem.

Similar to analysis -4, for case 1, 2 there shouldn't be any relative change , with exp. incr. in case 3

However, the results are : -



Conclusion

- Measuring power is easy. Measuring power <u>accurately</u> is **hard**!
- To measure the power of internal data movements, large amounts of data movement is needed.
- But, the OS gets in the way – time outs and context switches – which adds noise to the data
- Overall, the benchmarks did what they were designed to do – move data between the core, L1 cache and LLC.
- Future work would include modifications to the kernel to limit time-outs and context switches as well as move to a system which allowed more access to power data

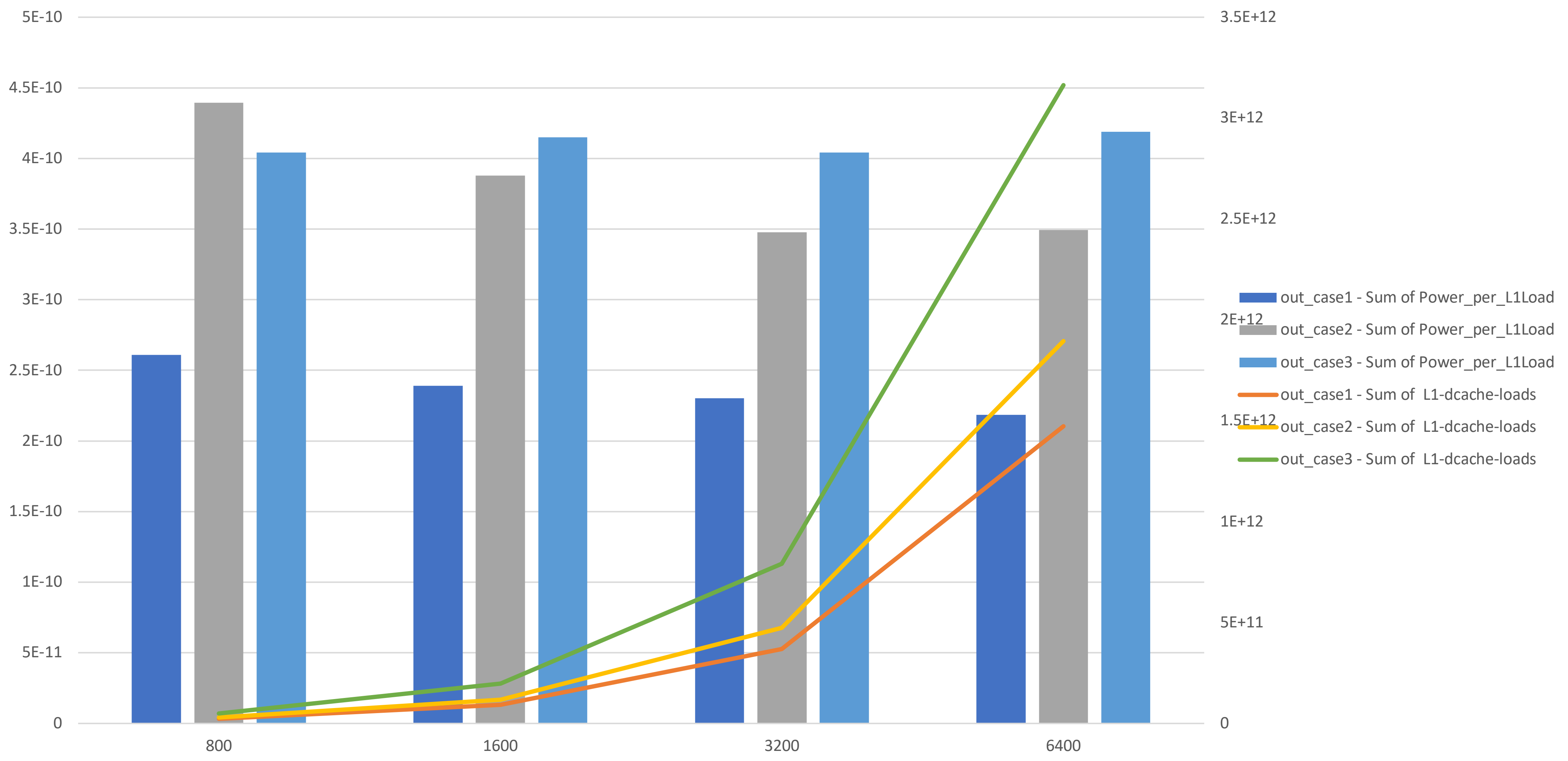
Back up slides

Raspberry Pi 3

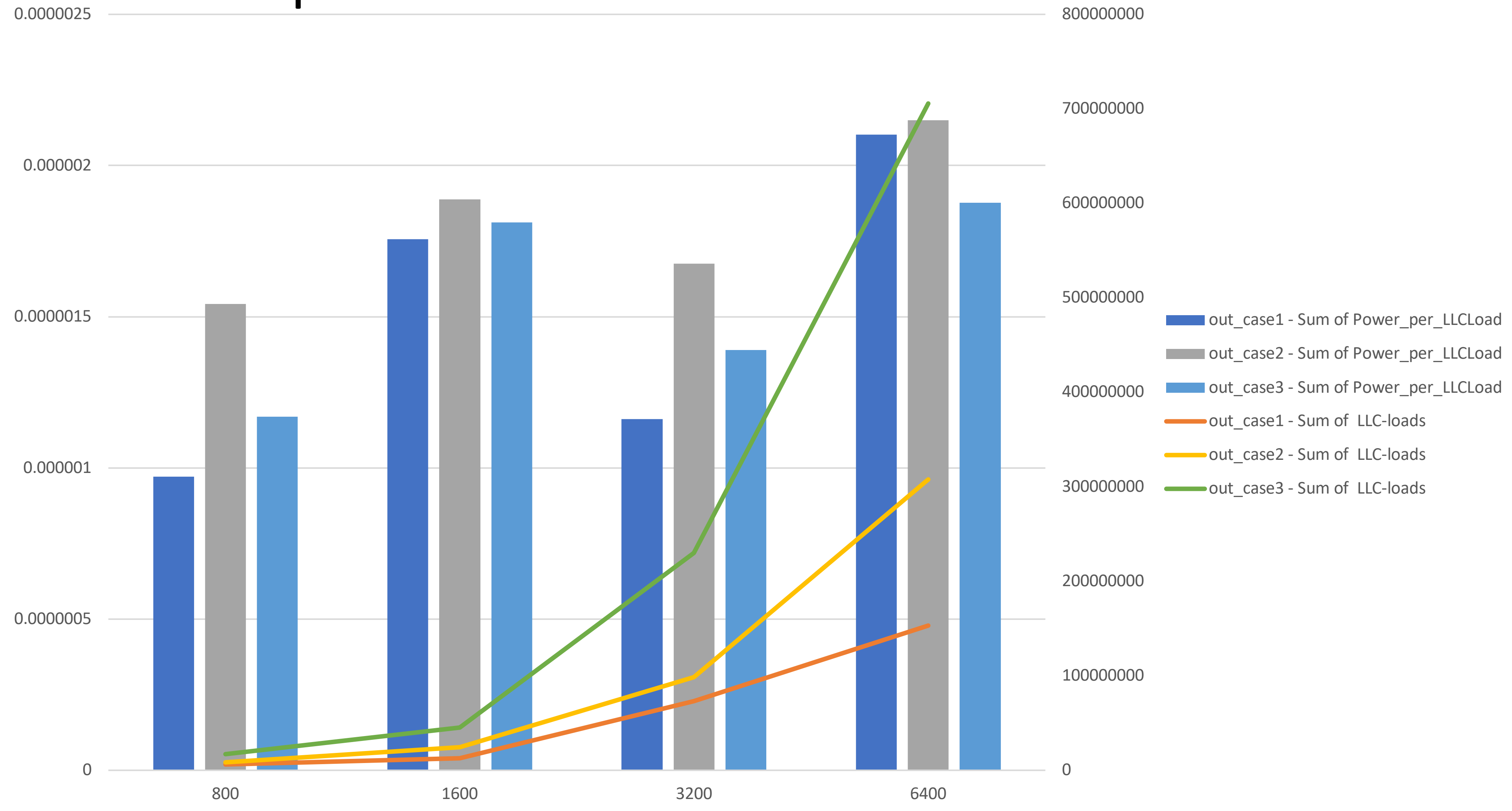
- Why did we use a Raspberry Pi 3?
  - Ease of use
    - Easy Access
    - Easy Configuration
  - Power measurement through USB power supply

- Was it a good decision?

Power per L1 Cache Load



Power per LLC Cache Load


out_case1 - Sum of Power_per_LLCLoad
out_case2 - Sum of Power_per_LLCLoad
out_case3 - Sum of Power_per_LLCLoad
out_case1 - Sum of LLC-loads
out_case2 - Sum of LLC-loads
out_case3 - Sum of LLC-loads