Group 2 Project– Version 1 – May 30, 2018

CS533 Spring 2018

Data Movement and Power Measurements

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

Goals:

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

Background:

In a typical “pipeline” process, data is move 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 work tests this idea.

Hypothesis:

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:

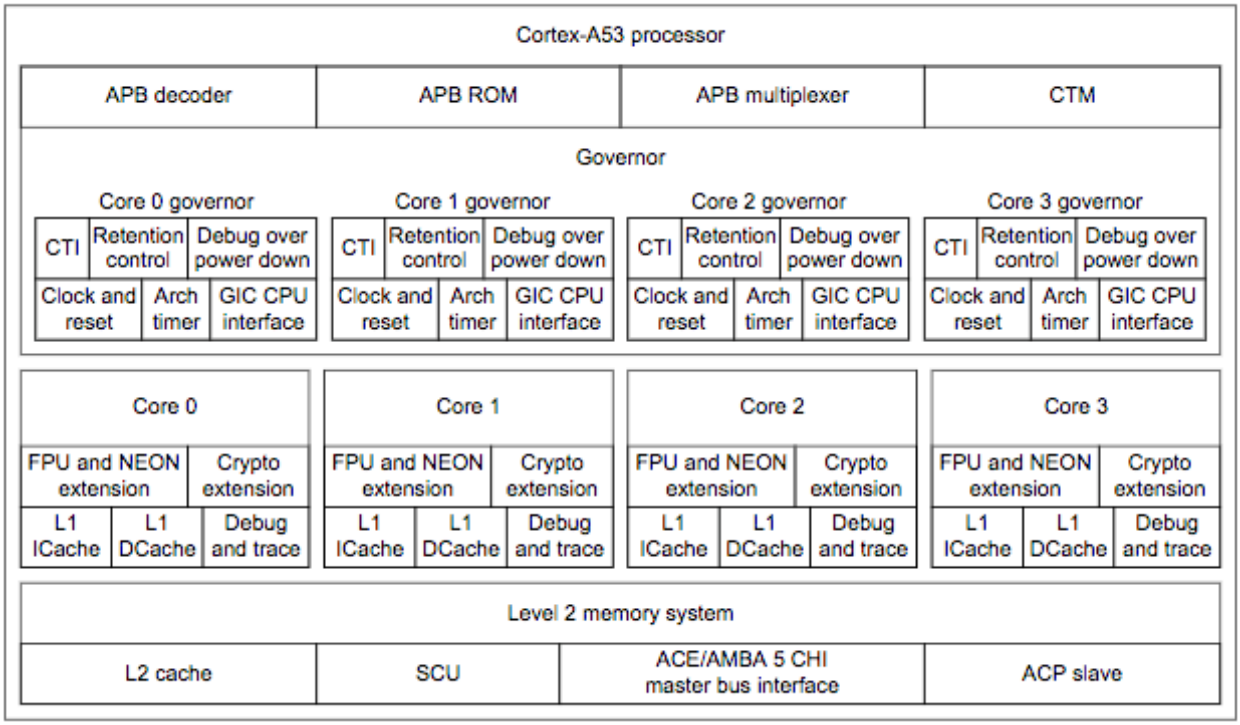
<Need to write>

Hardware Platform: Raspberry Pi 3[1]

- SoC: Broadcom BCM2837
- CPU: 4 × ARM Cortex-A53, 1.2GHz
- GPU: Broadcom VideoCore IV
- RAM: 1GB LPDDR2 (900 MHz)
- Networking: 10/100 Ethernet, 2.4GHz 802.11n wireless
- Bluetooth: Bluetooth 4.1 Classic, Bluetooth Low Energy
- Storage: microSD
- GPIO: 40-pin header, populated
- Ports: HDMI, 3.5mm analogue audio-video jack, 4× USB 2.0, Ethernet, Camera Serial Interface (CSI), Display Serial Interface (DSI)

System overview – block diagram:[2]

Figure 2.1. Cortex-A53 processor block diagram



[1] https://www.raspberrypi.org/magpi/raspberry-pi-3-specs-benchmarks/

[2] http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0500e/BABCFDAH.html

Memory Configuration

L1 Cache details[3]:

The L1 Instruction memory system has the following key features:

- Instruction side cache line length of 64 bytes.
- 2-way set associative L1 Instruction cache.

128-bit read interface to the L2 memory system.

The L1 Data memory system has the following features:

- Data side cache line length of 64 bytes.
- 4-way set associative L1 Data cache.
- 256-bit write interface to the L2 memory system.
- 128-bit read interface to the L2 memory system.
- Read buffer that services the *Data Cache Unit* (DCU), the *Instruction Fetch Unit* (IFU) and the TLB.
- 64-bit read path from the data L1 memory system to the datapath.
- 128-bit write path from the datapath to the L1 memory system.
- Support for three outstanding data cache misses.
- Merging store buffer capability. This handles writes to:
  - Device memory.
  - Normal Cacheable memory.
  - Normal Non-cacheable memory.
- Data side prefetch engine.

L2(LLC) Cache details[4]:

The L2 memory system consists of an:

- Integrated *Snoop Control Unit* (SCU), connecting up to four cores within a cluster. The SCU also has duplicate copies of the L1 Data cache tags for coherency support. The L2 memory system interfaces to the external memory system with either an AMBA 4 ACE bus or an AMBA 5 CHI bus. All bus interfaces are 128-bits wide.
- Optional tightly-coupled L2 cache that includes:
  - Configurable L2 cache size of 128KB, 256KB, 512KB, 1MB and 2MB.
  - Fixed line length of 64 bytes.
  - Physically indexed and tagged cache.
  - 16-way set-associative cache structure.
  - Optional ACP interface if an L2 cache is configured.
  - Optional ECC protection.

The L2 memory system has a synchronous abort mechanism and an asynchronous abort mechanism, see *External aborts handling*.

[3] http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0500e/BABCFDAH.html
[4] http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0500e/BABCFDAH.html

Software Configuration

Operating System

- Kernel – Linux 4.14.34-v7+
- Customer built[5,6]
    - Add performance and power counter support
        - Need to capture all the details.
    - If possible, enable L2 cache, disable GPU(if able)

    MORE DETAILS HERE

Measurement Tools

- perf
    - Built with custom kernel
    - <Need to describe the specific counters used>
- schedtool
    - Used to set nice values and core affinity, etc.

Benchmarks

Data Movement Types

There are two types of data movement to be considered: Instructions and Data. The instructions must be fetched from, at a minimum, L1 Cache. This movement of data is unavoidable. If we assume that instructions for all benchmarks fit in L1 Cache, we can eliminate the corresponding power as it is equal across all benchmarks. All benchmarks should also start out with an equivalent amount of data in main memory and load it into L1 cache and LLC in the same way. Also, the operations on the data must be equivalent so that the power used by the core is the same across benchmarks.

Measurement Types

1. Baseline - With-in Core – only accessing Core Registers
2. Between Core and L1 Cache
    a. Data limit of 32KB
3. Between a single Core and LLC
4. Between Cores via LLC
    a. Core 1 L1 Cache -> LLC -> Core 2 L1 Cache
5. Between Cores via Main Memory
    a. Core 1 L1 Cache -> LLC -> Main Memory -> LLC -> Core 2 L1 Cache

---

[5] https://www.raspberrypi.org/documentation/linux/kernel/building.md
[6] https://www.raspberrypi.org/documentation/linux/kernel/configuring.md

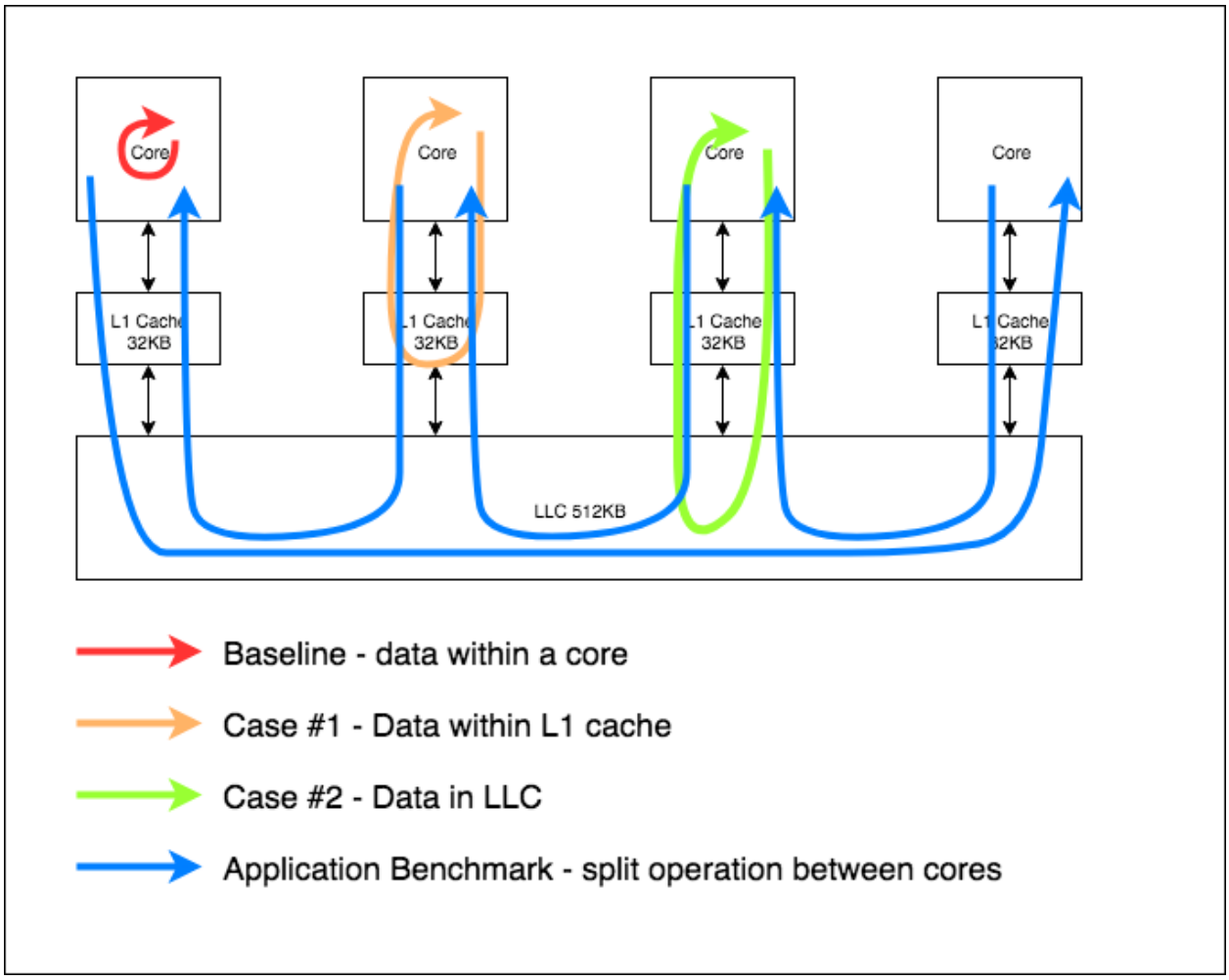
Measurement Methodology

Goals

1. Get power cost of moving a single value between the core and various levels of memory.
2. Get power cost of moving a single value from core to core.

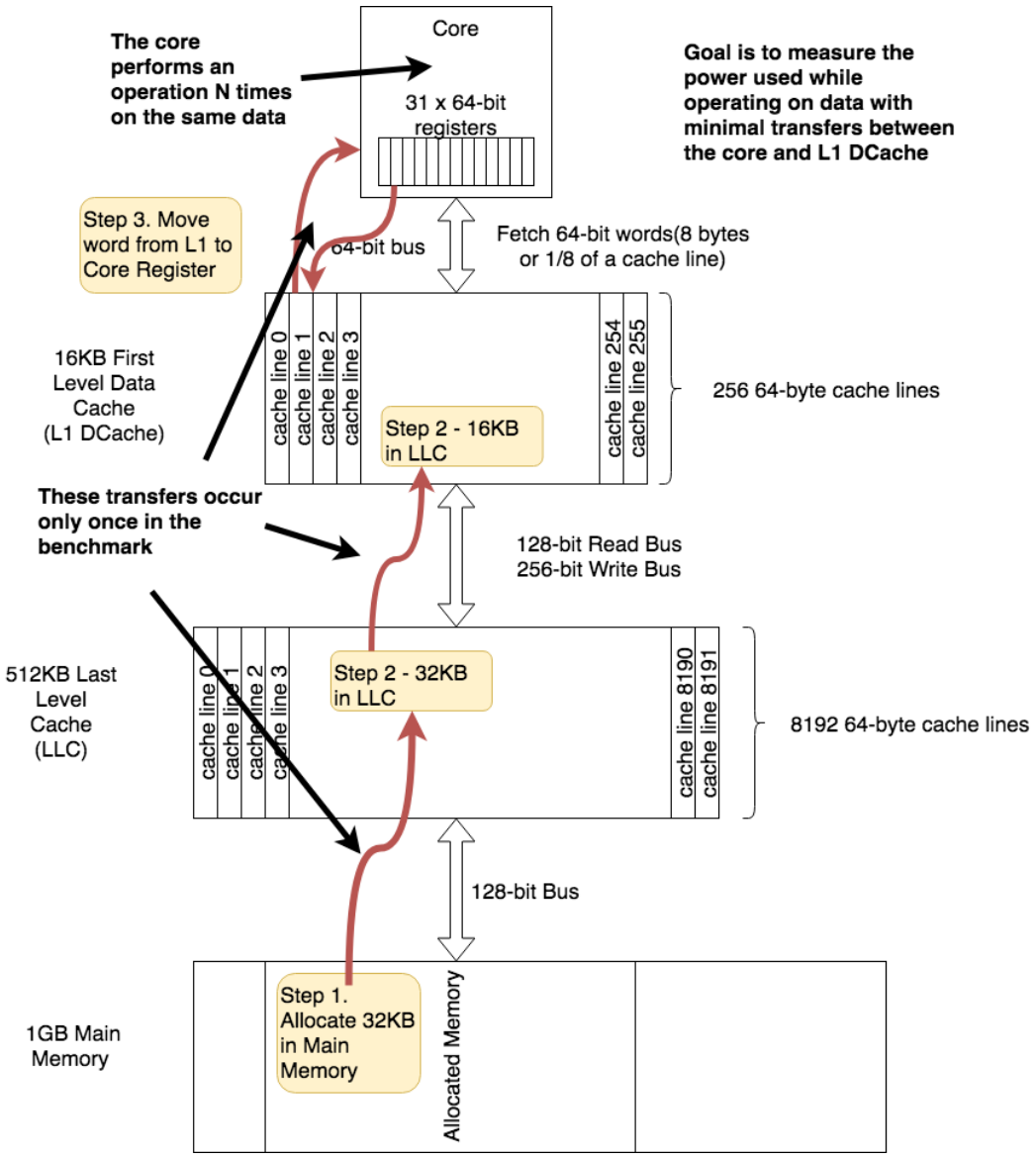
General Steps

1. Move instructions into iCache
2. Move data into Core <- source depends on benchmark goal
3. Operate on data
4. Move data out of Core <- destination depends on benchmark goal

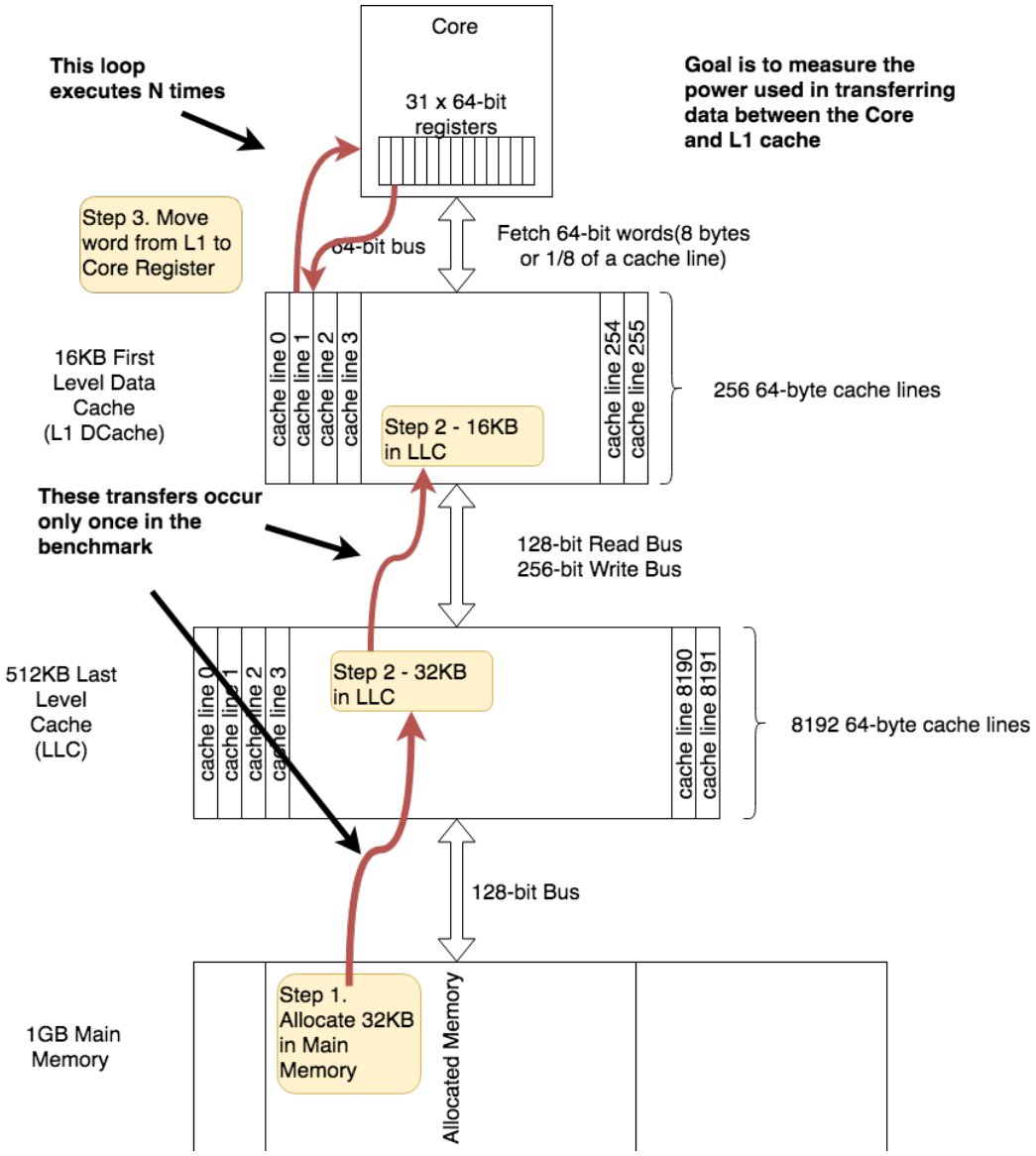


Cases

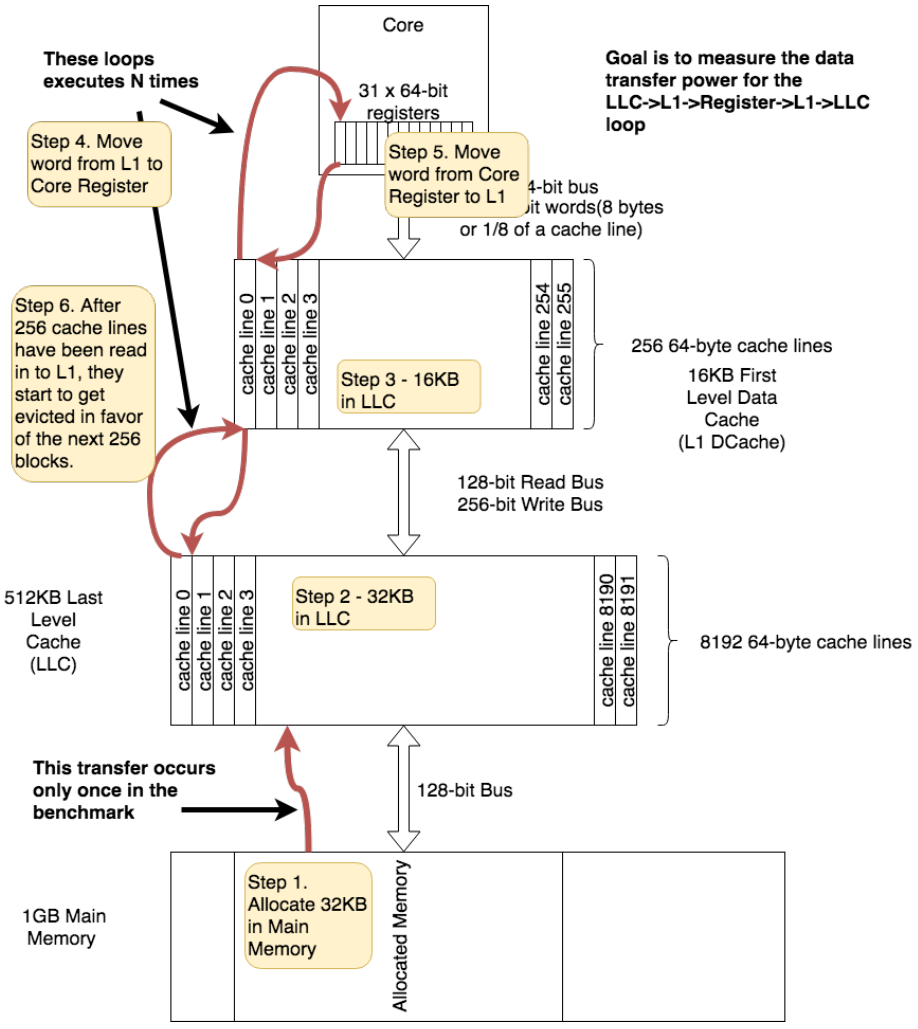
1. Case #1 - With-in Core <- this is the baseline power
    a. Load instructions from main memory into L1 cache
    b. Move data into LCC and L1 cache from main memory
    c. Move data into core register
    d. Operate on in-register data – avoid moving data out of Core registers
        i. Perform N operations
    e. Output data to Main Memory
    f. Read perf counters



2. Case #2 - Between Core and L1 cache
    a. Load instructions from main memory into L1 cache
    b. Load data into LCC and L1 cache from main memory
    c. Load data into core register
        i. Accessing the same cache line is OK as long as the data is moved from L1 to a core register
    d. Operate on data
    e. Store data in L1
        i. Be careful not to access LLC
    f. Repeat last three steps N times
    g. Output data to Main Memory
    h. Read perf counters



3. Case #3 - Between Core and LLC
    a. Load instructions from main memory into L1 cache
    b. Load data into LCC and L1 cache from main memory
    c. Load data into core register
        i. Each access must cause a load from LLC – there are 256 cache lines in the data cache of each core – each of 64-bytes
    d. Operate on data
    e. Store data in L1 and LCC
        i. Must be sure to push data to LCC
    f. Repeat last three steps N times
    g. Output data to Main Memory
    h. Read perf counters



4. Application Benchmark – Core to Core Data Movement
    a. Baseline
        i. Move all data into L1 cache and LCC from main memory
            1. Data set should be large enough that multiple cache misses will occur
        ii. Execute 4 operations on all the data on a single core.
        iii. Store data in main memory from single core L1
    b. Split Data
        i. Move ¼ of the data from main memory into L1 cache of each core
        ii. Execute 4 operations on all the data in each core
            1. Large data – multiple fills of L1 cache
        iii. Store data in main memory from each core L1 cache
    c. Core to Core
        i. Move ¼ of the data into each L1 cache of each core
        ii. Execute 1 operation in each core
        iii. Store data back in main memory
        iv. Shift data to each core and repeat for each of the 3 remaining operations

Results

<PROBABLY NEED SOME RESULTS>