IP WINTER 2019

Shivam Aggarwal(2016195)
Raghavv Goel(2016179)

Acknowledgments

We would like to express our deepest appreciation to all those who provided us the possibility to complete this report. We are grateful to Mitali Sinha [Ph.D. Student] for her constant assistance with the project. We are thankful to everyone who provided expertise that greatly assisted the research work and supported the project.

We are really grateful of Dr. Sujay Deb, for his vision and constant support without which we wouldn't have been able to do this.

Contents

Chapter 1

Introduction

There were two themes which are as follows:

1 Implementing a simulator which can simulate any type of architecture for accelerators used for Deep learning.

2 Implementing dynamic cache partitioning for multi accelerator based systems.

For the first one, a number of recent and past work was read. Analysis was made and a good flavour of how to model a simulator just for even a TPU(Tensor Processing Unit) will be made. A TPU is a dedicated unit in the chip for the purpose of fast matrix multiplication, implying that if we have a neural network which needs to be trained/tested, then we can push the training part or testing part to TPU and it'll give the results a lot faster than a CPU does.

For the purpose of matrix multiplication, the TPU uses systollic arrays, which flow data in a wave like technique, along with storing already multiplied data, the matrix multiplication becomes really fast.

Then, we worked on GEM5-Aladdin to perform dynamic cache partitioning of the last level cache(LLC). The idea behind cache partitioning is motivated from multi core systems(more than 1 CPU) having same amount of LLC cache but using not fully using the memory of the LLC allotted to them.

The same idea was extended for a multi core system with multiple accelerators. The dynamicness is brought in by an reinforcement learning(RL) based agent.

Chapter 2

TPU Based Simulator

2.0.1 Architecture of TPU

2.0.2 Scale-Sim

SCALE-SIM (Systolic CNN Accelerator Simulator) is a cycle-accurate, systolic array based CNN accelerator simulator. SCALE-SIM exposes various micro-architectural features such as array size, array aspect ratio, scratchpad memory size, dataflow mapping strategy, as well as system integration parameters such as memory bandwidth.

Given a convolution neural network topology and certain architecture parameters, SCALE sim is capable of estimating the following:

- Run time in cycles
- Average utilization
- On-chip memory requirements (SRAM accesses)
- Off-chip interface bandwidth requirements (DRAM accesses, and DRAM bandwidth requirement).
- Benchmark used: MLPerf

Tool can help with:

- Analysing the effect of various dataflows (how data is represented in the systolic arrays) - Output Stationary, Input stationary, Weight Stationary.
- Identifying factors essential for sizing on-chip memory.
- Effect of systolic array size/aspect ratio
- Explore the trade-offs b/w Scaling-Up and Scaling-down. Scaling-up - Making the array bigger and scaling-down - having more arrays and dividing the compute among them.

Validation: SCALE-SIM is validated against an in-house RTL model for a systolic array implementing OS dataflow. The workload they chose is a Mat-Mat multiplication of matrices with same size as the array.

The tool takes two files as inputs from the user, a config file and other is a topology file.

- Config file- architecture specification details
- Topology- specifies the CNN architecture.

The following can be improved:

- It is important to understand the accelerator in the context of the entire Systems-on-a-chip (SoC). Scale-sim doesnt provide that flexibility.
- The simulators scope is limited to analysing the effect of arrays hyperparameters on the system performance. Not possible to apply any optimization techniques within the current framework.
- Cannot modify the current system architecture. For eg, you cannot add a custom cache, change how instructions are performed, etc.

Why do we need a systolic-array based simulator within gem5?

1. **GEM5**
   Discrete-event simulation platform with numerous models
   - CPU models at various performance/accuracy trade-off points
   - Multiple ISAs: x86, ARM, Alpha, Power, SPARC, MIPS
2. **Two memory system models: Ruby and classic (M5)**
   - Including caches, DRAM controllers, interconnect, coherence protocols, etc.
   - I/O devices: disk, Ethernet, video, etc.
3. **Full system or app-only (system-call emulation).**
4. **Cycle-level modeling (not cycle accurate):**
   - Accurate enough to capture first-order performance effects.
   - Flexible enough to allow prototyping new ideas reasonably quickly.

It makes sense to add infrastructure within the simulator to take advantage of these features.

2.0.3 Recent Systolic Array Based Architecture

- Googles TPU
- MITs Eyeriss
- Neurocube
- Flexflow
- MAESTRO
- Pipelayer

2.0.4 HyPar

A communication model where the communication comes from amount of communication between different partitions. Dynamic Programming is used to find the partition of each layer HMC(hybrid memory cube) based DNN accelerator. It is a hybrid between data parallelism and model parallelism for training.

The amount of power and time consumed when accessing data from the main memory is very high. These frequent off chip memory accesses need to be prevented when the amount of data is high and bandwidth between memory and CPU is small. When training Deep Neural Networks(DNN) the amount of training data is very high, so HyPar helps to explore more DNN based architectures. Also, there is a need for multiple accelerator to explore coarse grain acceleration. It was also proved that only data parallelism or only model parallelism performed wore than hybrid.

Data Parallelism: Inter communication between accelerator gradient computation

Model Parallelism: Inter communication after forward computation(propagation).

All of the following accelerators(Eyeriss, Maestro, Neurocube, Flexflow) focus on the intra layer computation, only one layer computation is performed at one time. Now, for an array of accelerators the input data for a particular layer is produced by the outputs of other layers. Hence, the computation and data flow affect the data movement.Therefore, we need to determine partitions and the data flow.

An optimization problem is formulated: minimize the total communication during training. For this the authors propose, an HMC based architecture which has stacked DRAM dies and logic dies along with high bandwidth.

For evaluation, 3 data sets and 10 networks chosen from fully connected(SFC), only convolutional(SCON)

3 data sets and 10 networks chosen from fully connected(SFC), only convolutional to mix of both simulations were event driven. computation cost, memory accesses and communication between vaults were modelled. A vault is an Eyeriss accelerator and it's local memory. It was

observed that fully connected layers are like model parallelism and convolutional layers are like data parallelism.

Finally, the results were that Model Parallelism had the most total communication: 8.88GB, followed by Data Parallelism and finally with Hypar.

2.0.5 Real Time Graphical Simulation of Systolic Array

A system designer must be able to observe the movement of every piece of data as it traverses through the at all tunes to verify whether a given algorithm is correctly mapped into the corresponding array architecture. He must also be able to see the results from the operations performed on each piece of data by any of the cells. Furthermore, for debugging purposes, he must be able to load into the registers of every cell at any one time and see the values of all the control signals present in that cell. In short, he must have the most detailed view of the entire system that may consist of many arrays composed of many types of cells This perspective must be available to him with the accuracy of single pulses during the entire simulation process.

To meet the above requirements, a new breed of simulator - a systolic array simulator - has been developed and built by us to aid a hardware or software designer in the task of designing, evaluating, debugging, verifying, etc. It was deemed essential that this simulator should be graphics based, hence its name Systolic Arrays Graphical Simulator(SAGS for short).

From the very beginning, SAGS was designed to simulate the systolic systems of any configuration. These configurations are specified to SAGS by means of script files. A script file contains all of the vital information about the system: its number of arrays, their types and sizes, the way they are linked together. and the microprograms used in each cell. A script file also specifies when and where the input data and the control signals should be fed into, and the output data taken from the system. SAGS allows for systems with multiple input control, and output data streams. Each input or control mean is stored into ASCII files prior to being accessed by SAGS. Similarly, SAGS outputs are written into ASCII files.

2.0.6 Systolic Array Architecture in Bluespec

coded and simulated four solutions in Bluespec to verify the accuracy of each of the solutions, these guys used RISC V architecture in Bluespec

- Solution 1: Perform matrix-matrix multiplication using 2D systolic array of processor elements.
- Solution 2: Perform matrix-matrix multiplication using linear 1D array of processor elements.
- Solution 3: Perform matrix-matrix multiplication using single processor element.

- Solution 4: Perform matrix-matrix multiplication using linear bidirectional 2D systolic array of processor elements.

Bluespec refers to a language and associated tools which are being used for all aspects of hardware system design . specification, synthesis, modeling, and verification. The language, BSV (Bluespec SystemVerilog), is based on a new model of computation for hardware, where all behavior is described as a set of rewrite rules, or Guarded Atomic Actions. Unlike the process/thread model of Verilog, VHDL and SystemC, or the sequential model of C/C++, all behavior of a BSV program can be understood in terms of atomic rule firings. This computational model has a long pedigree in formal specification and verification systems (e.g., Dijkstra's Guarded Commands, UNITY, TLA+, and EventB), and BSV makes it available for hardware design.

2.0.7 Efficient way to simulate hardware accelerators

In order to incorporate efficiently our accelerator module we have developed a set of device drivers. The accelerator is activated through programmed I/Os that provide the start address and the size of the array used for the descriptors of the accelerator. The CPU can then sleep until an interprocessor interrupt from the accelerator is delivered to indicate task completion; this approach allows for full overlap of the two sub-simulations. In addition, an ioctl function was developed in order to achieve efficient user-kernel space communication; this novel function can mainly perform the following tasks (additional helper commands were developed but cannot be listed in this paper due to the page limit):

- QUERY SET DATA - Initialise the Direct Memory Access (DMA) copy transaction from Host (Kernel Space) to Accelerator Wrapper.
- QUERY GET DATA - Initialise the DMA copy transaction from Accelerator Wrapper to Host (Kernel Space)
- QUERY CALL DEVICE - Call the SystemC accelerator (executing the specified application) .

Finally, an interrupt handler was implemented in order to receive appropriate interrupts from the Accelerator Wrapper, such as the SystemC accelerator finish signal, the memcpy finish signal, etc.

GEM5 provides a Memory Bus to interconnect all architecture components such as CPUs, Caches, RAMs as well as I/O devices using master and slave ports. Our novel Accelerator Wrapper device was attached to the GEM5 system using one master and one slave port. Specifically, one bus master port was connected to the peripheral I/O Accelerator Wrapper port pio in order to read and write into the accelerators wrapper registers; similarly, one bus slave port was connected to the DMA Accelerator Wrapper port in order to write/read large amounts of data to/from the accelerator device memory

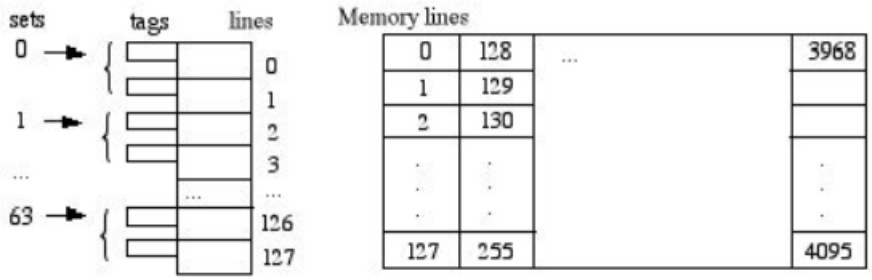
Another important remark is that normally in every SystemC simulator call, the full-system must boot the OS from scratch, while, and more importantly, no notion of synchronisation exists in such a file-based exchange. In contrary, the proposed method resolves efficiently all those issues since the full-system simulator boots the OS only once and, since it communicates with the SystemC simulator through programmed I/Os and DMA engines, full global synchronisation is supported. Moreover, the proposed approach is orders of magnitude faster that the conventional one

Chapter 3

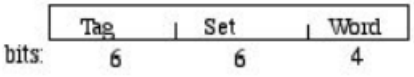
Dynamic Cache Partitioning

3.0.1 Introduction

Set-way associative cache: This scheme is a compromise between the direct and associative schemes. Here, the cache is divided into sets of tags, and the set number is directly mapped from the memory address (e.g., memory line j is mapped to cache set j mod 64), as suggested by the diagram below:



The memory address is now partitioned to like this:



Here, the "Tag" field identifies one of the 64 different memory lines in each of the 64 different "Set" values. Since each cache set has room for only two lines at a time, the search for a match is limited to those two lines (rather than the entire cache). If there's a match, we have a hit and the read or write can proceed immediately. Otherwise, there's a miss and we need to replace one of the two cache lines by this line before reading or writing into the cache. (The "Word" field again select one from among 16 addressable words inside the line.)

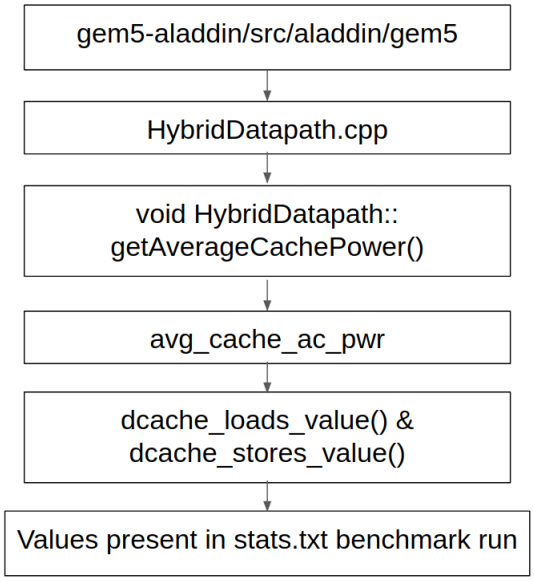


Figure 3.1: How loads and stores are used for finding the power estimate

3.0.2 Area and Power Estimates

A tool called CACTI, was used for estimating the area and power of the caches. In case of area, there was no dynamic variables involved and hence it was just a mapping. In case of power, both the static and dynamic power needed to be calculated.

Avg cache pwr = (dcacheLoads.value()*cacheReadEnergy + dcacheStores.values()*cacheWriteEnergy)/cycles

Cacti uses .cfg files to get the configuration of the system. After the benchmark has ran it takes the values from the stats.txt. The area and power estimate can be seen through two files generated in the output folder of the benchmark ran: test_aes_power_stats.txt test_aes_summary.

3.0.3 Static Cache Partitioning

Based on the description outlined in the section 3.0.1, L2 cache partitioning was added to the gem5-Aladdin simulator. Cache partitioning assigns a subset of cache ways to each core such that a core is limited to its assigned subset of ways for allocating lines in the cache. Partitioning is only implemented for LRU replacement therefore, the cache must be configured to use the LRU tag in order for partitioning to be enabled. In order to find which CPU has requested to allocate a line in L2, the code looks for the CPU names in MasterName. This way, the CPU names in the Python script must be the same as the constant CPU strings defined in allocateBlock. It is possible to make the code more flexible by passing the partitions configuration and CPU MasterNames from the Python script to the C++ code and avoiding any hard coded constant. I implemented the static-cache partitioning in both classic and ruby memory model.

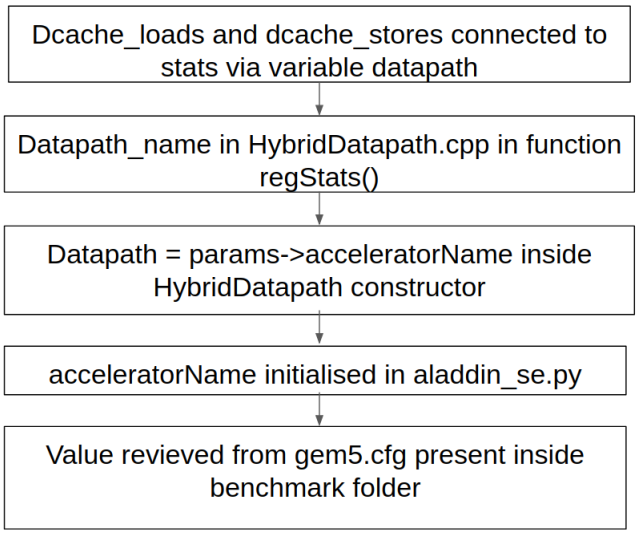


Figure 3.2: The connection between HybridDatapath and stats.txt

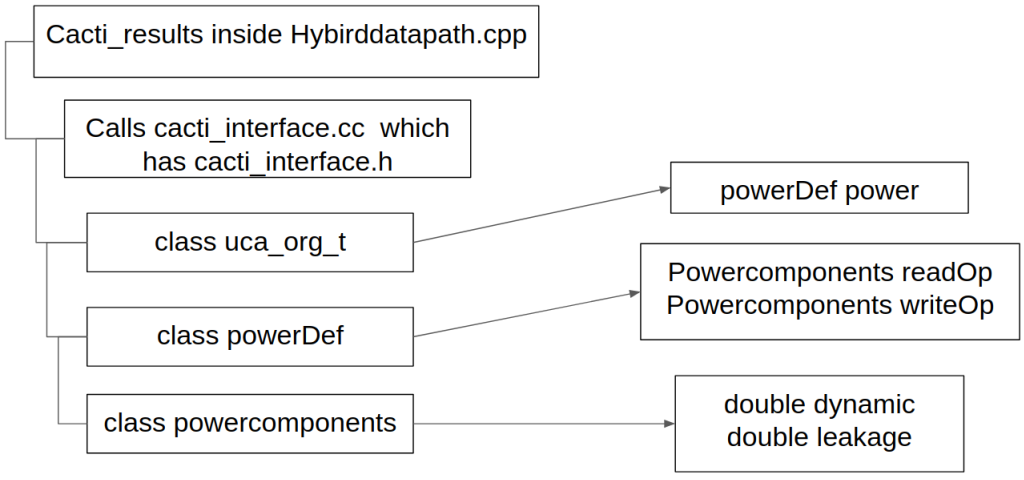


Figure 3.3: The way cacti has defined objects and classes for estimating power

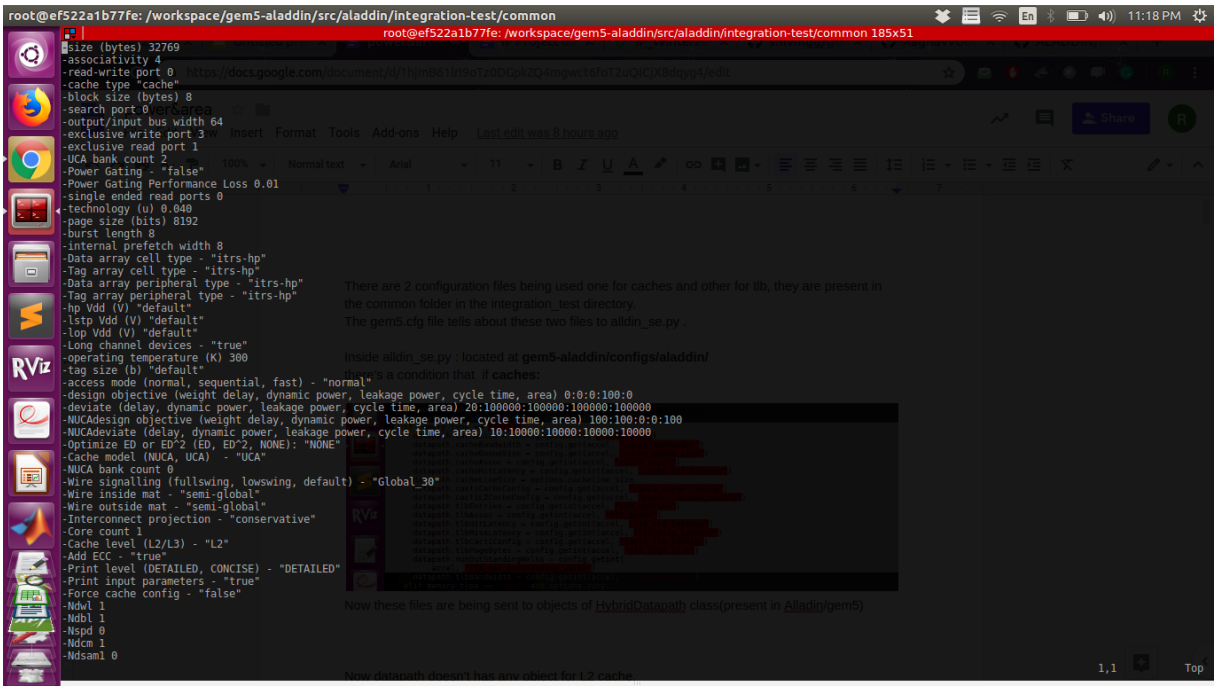


Figure 3.4: The parameters inside a .cfg file which is an input for the cacti

3.0.4 Ruby Memory Model

The Ruby memory system model provides gem5 a flexible infrastructure capable of accurately simulating a wide variety of memory systems. In particular, Ruby supports a domain specific language called SLICC (Specification Language for Implementing Cache Coherence) where one can define many different types of cache coherence protocols. Essentially SLICC defines the cache, memory, and DMA controllers as individual per-memory-block state machines that together form the overall protocol. By defining the controller logic in a higher level language, SLICC allows different protocols to incorporate the same underlining state transition mechanisms with minimal programmer effort.

Within the Ruby memory system, all objects are connected to each other via MessageBuffers. The two pictures below highlight the difference between the two memory models:

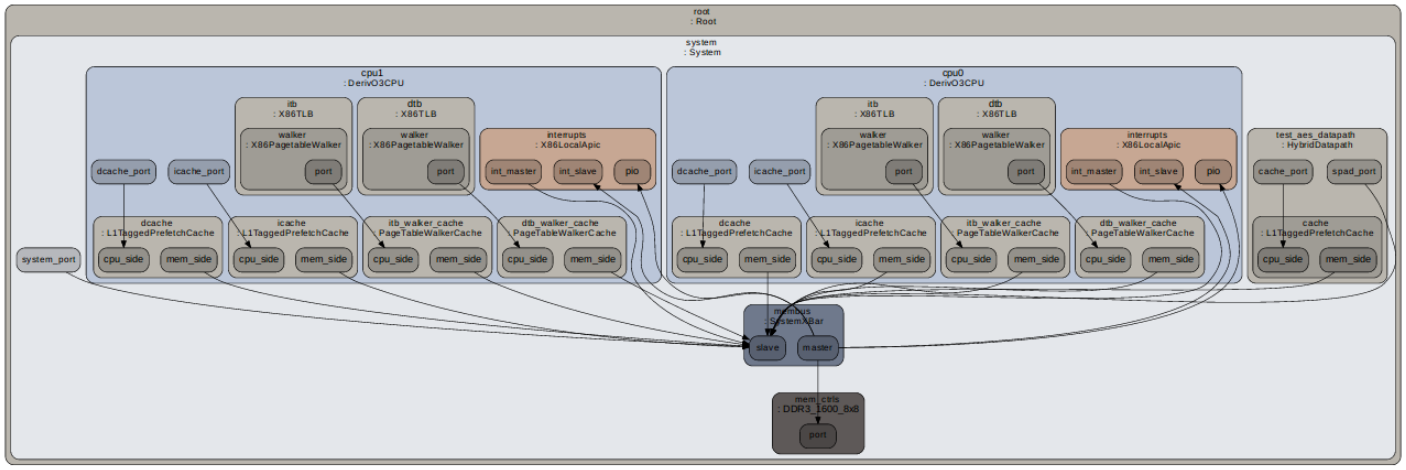


Figure 3.5: Classical Memory model

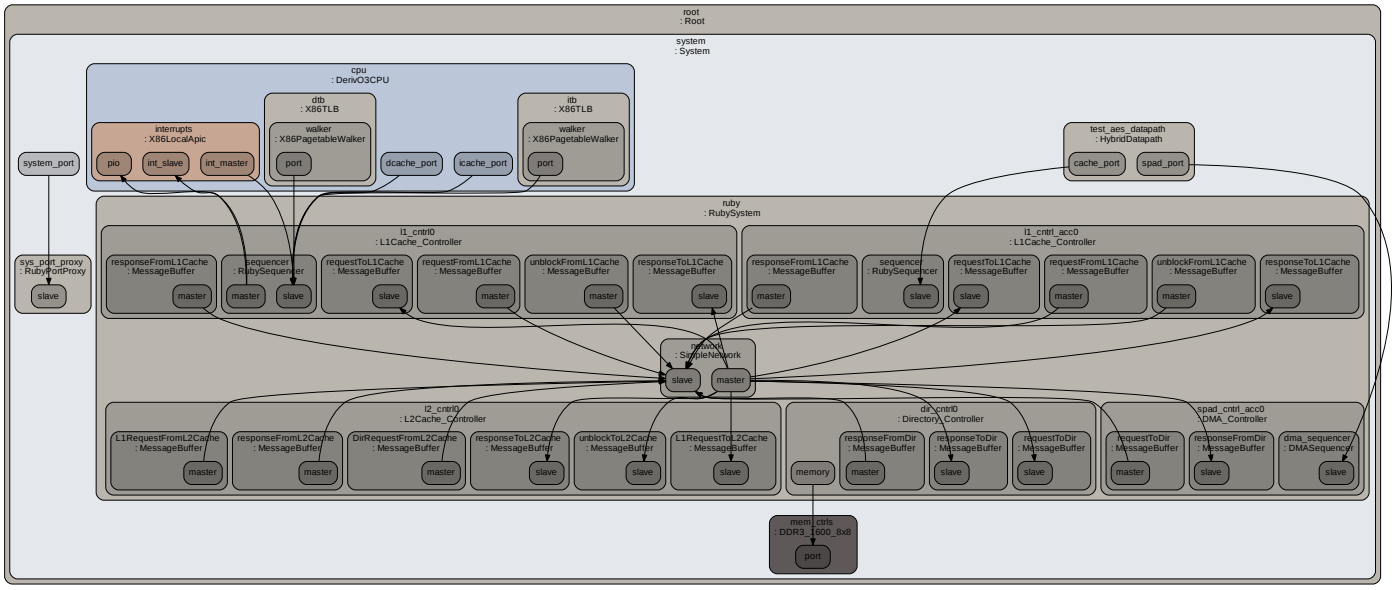


Figure 3.6: Ruby Memory model

Implementation of the cache partitioning technique in the two memory models are poles apart. The implementation details can looked here: Static-cache-partitioning in ruby memory model

Ruby-memory model specific implementation details:

- /src/mem/protocol/MESI_Two_Level-L1cache.sm
- /src/mem/protocol/MESI_Two_Level-L2cache.sm
- /src/mem/protocol/RubySlicc_Types.sm
- src/mem/ruby/structures/CacheMemory.hh

- src/mem/ruby/structures/CacheMemory.cc

Major functions that we needed to change in CacheMemory:

- bool cacheAvail()
- AbstractCacheEntry* allocate()

Other important related papers:

- SCP: Shared Cache Partitioning for High-Performance GEMM
- Accelerating Concurrent Workloads with CPU Cache Partitioning
- CoPart: Coordinated Partitioning of Last-Level Cache and Memory Bandwidth for Fairness-Aware Workload Consolidation on Commodity Servers

3.0.5 Initial framework for dynamic cache partitioning

We have surpassed the value of current curTicks in the code and want to pass the same to a dynamic controller. Now, this agent will be outside the simulator and called after every fixed number of curticks. The value of the curticks can be checked using the program file:src/sim/simulate.cc. We ran a small experiment to call a function from a program file (which is outside the simulator) and print its values during the simulation process. It worked perfectly.

Changes can be found here: dynamic-cache-partitioning-step1.patch

Next, we need to work on the framework for dynamically changing the parameters defining the cache partitions for the CPU cores.

We are working on a more constructive policy as of now.