Evaluating pseudo-random SRAM for AI applications in GPU cache

KWAKU ASARE

MASTER´S THESIS

DEPARTMENT OF ELECTRICAL AND INFORMATION TECHNOLOGY

FACULTY OF ENGINEERING | LTH | LUND UNIVERSITY

Evaluating pseudo-random SRAM for AI applications in GPU cache

Kwaku Asare
kw3883as-s@student.lu.se

Xenergic AB

Supervisor LTH: Iman Ghotbi

Supervisor Xenergic: Tom Johansson

Examiner: Pietro Andreani

June 4, 2024

© 2024
Printed in Sweden
Tryckeriet i E-huset, Lund

Acknowledgements

I would like to express my gratitude to the following individuals and institutions:
Tom Johannson (Engineer at Xenergic AB); Your insights and time were invaluable. Thank you for guiding me through this project.
Babak Mohammadi (Engineer and Founder of Xenergic AB); Your vision, expertise and support significantly contributed to our success.
Xenergic AB: Thank you for providing me with the opportunity to work on this industry project. The experience gained was invaluable.
Academic Supervisor: Iman Ghotbi (Faculty of Engineering, Lund University): Your guidance during my thesis project was instrumental.
Lund University: To my classmates, lecturers, and the entire university community, thank you for fostering an environment of growth and learning.
Lastly, I extend my heartfelt thanks to my family for their unwavering support and encouragement throughout my two-year Master's program.

Abstract

General Purpose Graphics Processing Units (GPGPUs) have become the prevalent processor for AI/ML and other large computational problems because parallel processing has not reached a hardware limit, unlike single-threaded processing. The goal of the thesis is to investigate the suitability of a novel SRAM architecture for implementation in a GPU without extensive GPU architecture changes. This architecture features groups of SRAM cells called zones, which are leveraged to perform pipelined SRAM read operations to reduce dynamic energy consumption. This thesis examines the implementation of an energy-efficient SRAM in GPU caches and analyzes its energy saving and performance under AI/ML workloads in a GPU. GPGPU-Sim [1] simulator was used to run all the benchmarks. The simulator was modified to output all memory accesses made to L1 and L2 cache. Thereafter, the memory accesses were used by the software cache model to analyze the performance and energy of all the workloads run. Hybrid SRAM implementations with a small capacity conventional SRAM in tandem with the Pseudo-random SRAM (PR-SRAM) were investigated to check penalty cycles and reduction in dynamic energy consumption. A penalty cycle is a stall in the pipeline caused by repeated access to a specific zone. The penalty cycle rate is the number of penalty cycles per 100 accesses. The hybrid implementation featured a 33% increase in energy consumption versus the pure PR-SRAM implementation. This increase in energy consumption was the cost of reducing the penalty cycle rate by 43%. The effect of cache replacement policy on the performance of the hybrid design was also investigated, Least Recently Used (LRU) achieved the lowest penalty cycle rate. The gains were also measured against the complexity required to perform necessary operations with the hybrid cache.

Popular Science Summary

AI has become unavoidable in daily life; it is used to recommend services to us, in virtual assistants, healthcare, finance, and more. The meteoric rise of AI and its applications can be attributed to the technological advancements that have allowed processing power to catch up with the computational requirements of AI training.
These AI/ML models require large compute resources and are still limited by hardware, as current models are designed to take advantage of all the available hardware resources. Graphics Processing Units (GPUs) are the prevalent choice of processors for specific repetitive workloads with parallelism due to the GPU structure. This has led to a great increase in the computing power of graphics cards (GPUs) which has eclipsed the growth of computing power in Central Processing Units (CPUs). Performance increases in GPUs can be attributed to advancements in manufacturing processes and an increase in onboard memory.
To achieve these performance gains, improvements have been made across all aspects, including memory. Larger memory is required to store the growing amount of data being processed. Data transfer rates for all memory hierarchy levels have also been increased to improve latency, while wider buses and newer protocols have been implemented to improve memory throughput. The pursuit of increasing performance often results in energy efficiency taking a backseat. This thesis investigates the benefits of using a new memory design to read data efficiently with minimal impact on performance.
Scrutinising the benefits of Xenergic's new memory in a GPU; required the use of an accurate and detailed GPU simulator. A performance model was programmed to calculate the energy consumption of the new memory against a conventional memory. The model used details obtained from the simulator running benchmarks. The model also implemented a dual-memory design to look into performance and energy-saving trade-offs of different memory configurations. This thesis project found considerable energy reduction with the new memory design.

List of Abbreviations

AI	Artificial Intelligence
ASIC	Application Specific Integrated Circuit
CPU	Central Processing Unit
CU	Compute Unit
DRAM	Dynamic Random Access Memory
GPU	Graphics Processing Unit
GPGPU	General Purpose Graphics Processing Unit
EDA	Electronic Design Automation
HPC	High Performance Computing
IC	Integrated Circuit
LFU	Least Frequently Used
LRU	Least Recently Used
ML	Machine Learning
NN	Neural Net
PR-SRAM	Pseudo Random - Static Random Access Memory
RNG	Random Number Generator
SA	Sense Amplifier
SIMD	Single Instruction Multiple Data
SRAD	Speckle Reducing Anisotropic Diffusion
SRAM	Static Random Access Memory
SM	Streaming Multiprocessor
SP	Streaming Processor
SOTA	State of the Art

Table of Contents

1 Introduction **1**
1.1 Thesis Motivation . . . 1
1.2 Thesis Scope . . . 2
1.3 Thesis Outline . . . 2

2 Graphics Processing Units **3**
2.1 GPGPU Architecture . . . 3

3 Cache **7**
3.1 GPU Cache . . . 8
3.2 GPU Cache Operation . . . 9
3.3 Static Random Access Memory . . . 9
3.4 Pseudo Random SRAM . . . 10

4 Methodology **13**
4.1 Project Flow . . . 13
4.2 Benchmarks . . . 14
4.3 SRAM Performance Model . . . 17

5 Results and Analysis **19**
5.1 GPGPU-Sim . . . 19
5.2 Performance model . . . 21
5.3 Analysis on level 1 GPGPU cache . . . 28

6 Conclusion **31**

7 Future Work **33**

References **35**

A Benchmarks **39**

B Cache placement policies **41**

C Conflict detector **43**

List of Figures

2.1 Nvidia SM with 4 processing blocks [2] 5
2.2 Nvidia Kepler Architecture die illustration [3] 6

3.1 Cache placement policy 8
3.2 The memory hierarchy of the Nvidia GV100 9
3.3 The 6 Transistor SRAM cell [4] 10
3.4 PR-SRAM 2 cycle pipeline operation 11
3.5 PS-SRAM waveform for 4 cycle pipeline 11

4.1 Thesis project flow 14
4.2 Artificial neural network [5] 15
4.3 2D convolution operation [6] 15
4.4 2D convolution vs 3D convolution [7] 16
4.5 Overall SRAM performance model flowchart 17

5.1 GPGPU-Sim overall architecture 19
5.2 GV100 Cache organisation 20
5.3 L2 cache partitioning and cache address allocation 20
5.4 Comparative analysis of PR-SRAM and conventional SRAM 22
5.5 Hybrid SRAM waveform 23
5.6 Illustration of the 1st zone conflict 24
5.7 Second zone conflict illustration 25
5.8 Analysis of conventional SRAM capacity effects on penalty cycle rate and energy consumption 27
5.9 Different zoning methods for PR-SRAM 29
5.10 Analysis of conventional SRAM capacity effects on penalty cycle rate and energy consumption 30

List of Tables

2.1 Nvidia GPU components [2] 4

5.1 Penalty rate for different replacement policies 26
5.2 Summary of replacement policies 26
5.3 The ratio of total accesses to total number of clock cycles required by each benchmark 28
5.4 Summary of penalty cycles for different zoning implementations . . . 28

Chapter 1

Introduction

Dennard's Scaling Law is the trend of maintaining constant power density in integrated circuits (ICs) as transistors shrank in size [8]. Robert Dennard observed that as transistor dimensions shrank so did their dynamic power consumption. The operating frequency could be increased to improve performance because dynamic power consumption decreased, and as a result power density remained constant. Moore's Law is the observation made by Gordon Moore that the density of transistors in an integrated circuit board doubled approximately every two years [9]. The consequences of both of these laws are explosive growth in the performance capability of integrated circuits. Dennard's scaling ended around 2005 to 2007 when clock frequency could not be increased without causing significant power consumption and heating issues due to high leakage current at smaller transistor sizes. Moore's Law, however, held, that more transistors could be added to processors. This has led the industry to pivot to parallel computing paradigms to increase performance as a result modern CPUs feature parallelism but not to the extent seen in GPUs.

1.1 Thesis Motivation

General-Purpose Graphics Processing Units (GPGPUs) have become the ubiquitous processor for large computation problems such as Electronic Design Automation (EDA) [10], bioinformatics [11], cryptography [12] and more. AI/ML workloads involve repeatedly performing a limited set of operations on a large amount of data. This allows for AI/ML processing to be carried out in parallel processors such as GPGPUs and Application Specific Integrated Circuits (ASIC). GPGPU price-to-performance ratio beats out the higher performance and energy efficiency of ASICs. The main drawback to the widespread adoption of ASICs apart from inflexibility is the very high price to design. As a result AI/ML processing is performed with GPGPUs. The AI/ML boom has necessitated GPGPU capabilities to be pushed further, therefore and more research and resources have been poured in to produce innovations (in terms of interconnect protocols, cache organisation, and more) that continue to push the boundaries of GPGPU memory size, operating frequency, and performance throughput. Energy efficiency is another aspect that, while important, usually takes a backseat to raw performance gains. According to J. Tan [13], the L2 cache of GPUs consumes a large amount of chip energy due to the large SRAM structure it requires and its inefficient use in GPUs. This work concerns investigating the energy consumption of the new SRAM implementation and comparing its performance impact with a standard SRAM architecture.

1.2 Thesis Scope

There has been a lot of research into different caching policies for performance gains [14][15][16][17]; however, the scope of this thesis is on the SRAM used in GPGPU caches.
To perform this analysis detailed inner working of caches under AI/ML workloads were required. There are a number of different ways of performing this comparison given that a lot of crucial information related to the GPU architecture is closed source. A GPU simulator capable of simulating every clock cycle was used to run relevant benchmarks and all cache accesses were retrieved for analysis.
The analysis of the cache accesses was performed with a cache performance model, a Python program that calculates accumulated penalty cycles and energy consumption. The analysis was also used to confirm details about the GPU used such as the number of ports per cache bank and number of Streaming Multiprocessor (SM) cores in the simulated GPU. The model was also used to compare the energy consumption and performance impact between a conventional and a pseudo-random SRAM (PR-SRAM).
A hybrid architecture was also investigated to compare energy consumption and performance readings with PR-SRAM. The hybrid architecture used a combination of a small-capacity standard SRAM with the PR-SRAM in an attempt to achieve an optimal balance.

1.3 Thesis Outline

The next chapter provides an overview of GPUs and their architecture. Chapter 3 elaborates on caches, their operation, composition, and GPU caches. The Chapter also expounds on the architecture and operation of the PR-SRAMs. In Chapter 4, the methodology applied in this project is explained. The results and analysis of this project are presented in Chapter 5. A conclusion of this project is drawn and discussed in Chapter 6 and Chapter 7 elaborates on the future work that can be done for this project.

Chapter 2
Graphics Processing Units

GPUs are specialized electronic circuits designed to accelerate image and graphics processing which require a parallel structure. Their parallel architecture was leveraged to solve exceedingly parallel problems at a higher rate compared to normal CPUs, making them a popular option for processing large amounts of data that can be split. The biggest advantage of parallel computing is that currently it is limited by the workload and not a physical constraint. Embarrassingly parallel problems are problems that require little to no effort to split the problem into a number of parallel tasks [18]. The architecture of GPUs, primarily focused on graphics was modified to improve the performance at a greater range of tasks, leading to the emergence of the aforementioned General-Purpose Graphics Processing Units (GPGPUs). These modifications include implementing higher capacity and bandwidth memory compared to standard GPUs as well as caches optimized for general computing tasks.
State of the Art (SOTA) GPGPUs typically feature a large number of Compute Units (CUs) for processing a large amount of data concurrently. For example, OpenAI's SOTA large language model Chatgpt ran on General-Purpose GPUs and was trained with around 570GB of data on thousands of GPUs[19]. These advanced GPGPUs are utilized in High-Performance Computing (HPC) clusters, where they can be interconnected with other GPU nodes to pool resources and tackle much larger problems. GPU clusters have gained popularity due to their efficient resolution of AI/ML, scientific, and engineering workloads [20]. Nvidia's SOTA H100 GPU features 80 GB of memory, capable of 67 floating point teraFLOPS, and utilises NVLink protocol to interconnect with other GPUs to pool memory and cores [21].

2.1 GPGPU Architecture

GPGPU architecture features thousands of processing cores grouped into tens of multiprocessing cores. They also feature a large capacity on-chip cache as well as a high bandwidth and capacity on-board memory. GPGPUs have a general structure that is replicated across all designs. The significant difference between GPU architectures is the component type and number of components implemented into a design. Newer GPGPU architectures simply use a larger amount of multiprocessing cores paired with larger and faster memory components. Some application-specific

GPU Component	Quadro GV100
GPCs	6
SMs/GPC	14
TPCs/GPC	7
SMs	84
TPCs	42
FP32 Cores/SM	64
FP64 Cores/SM	32
GPU Boost clock	1530 MHz
Memory Interface	4096 bit
Memory	16 GB
L2 Cache Size	6144 KB
L1 Cache/SM	128KB
Shared Memory/SM	Up to 96KB
Register File/SM	256KB

Table 2.1: Nvidia GPU components [2]

processing cores (accelerators) have also been added to efficiently process specific workloads. The tensor cores shown in figure 2.1 were accelerators introduced to speedup operations involving multidimensional arrays. The drawbacks of parallelisation are still present in GPUs. Tasks that have a lot of data dependency or cause resource contention between different parallel tasks significantly impact performance. GPUs employ dependency analysis during code compilation to reduce data dependencies. A thread is a group of the smallest schedulable instructions. In GPUs, 32 consecutive threads are grouped into thread blocks. These thread blocks are called warps in Nvidia's documentation. These warps perform the same operation on all threads. However, during execution, a condition may be met that performs different operations on part of the warp. This is warp divergence limits parallelism. The Nvidia Quadro GV100 GPU is a relatively modern GPGPU designed with AI workflows in mind [2], therefore it would be interesting to investigate. The hardware architecture of the GV100's Single Instruction Multiple Thread (SIMD) core is depicted in Figure 2.1, and the specifications of the GPU are detailed in table 2.1. Nvidia refers to the SIMD as a Streaming Multiprocessor (SM) in their documentation and groups a number of them into GPU Processing Clusters (GPCs). The table lists the number of floating-point (FP) 32-bit and 64-bit cores in each SM, cache sizes, the number of SMs in the GV100 GPU, as well as the number of Texture Processing Clusters (TPCs)

GPUs have several types of memory to provide the necessary capacity and transfer speed to supply the thousands of computing cores. The GV100 has larger register files than level 1 caches to reduce context-switching latency. Context-switching allows GPUs to hide the latency of cache accesses by changing the thread being processed to another that has required data to be processed. Context switching is a process whereby a GPU changes between different tasks. It is crucial for

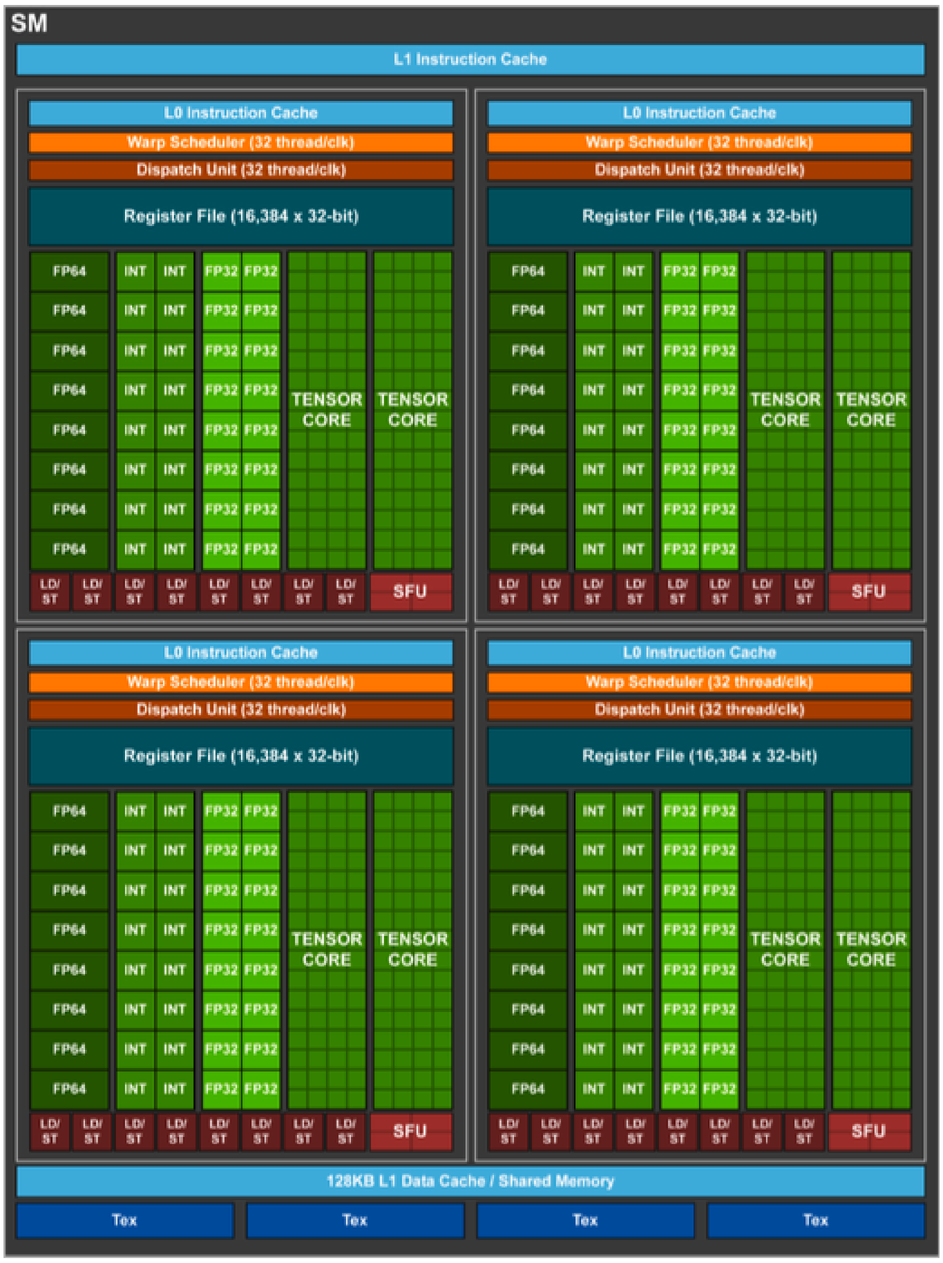

SM
L1 Instruction Cache
L0 Instruction Cache
Warp Scheduler (32 thread/clk)
Dispatch Unit (32 thread/clk)
Register File (16,384 x 32-bit)
FP64
INT
FP32
TENSOR CORE
LD/ ST
SFU
128KB L1 Data Cache / Shared Memory
Tex

Figure 2.1: Nvidia SM with 4 processing blocks [2]

resource utilisation and multitasking in a GPU.
GPUs leverage multilevel caches to optimise for improved performance. Two levels are typically utilised in caches with the larger level 2 cache used by all GPU cores and a number of smaller, faster level 2 caches used by groups of GPU cores as

shown in figure 2.1.
In Figure 2.2, a chip micrograph of the GTX680 Kepler Architecture GPU die is shown [3]. The middle band shown in multi-colours is occupied by the second level cache.

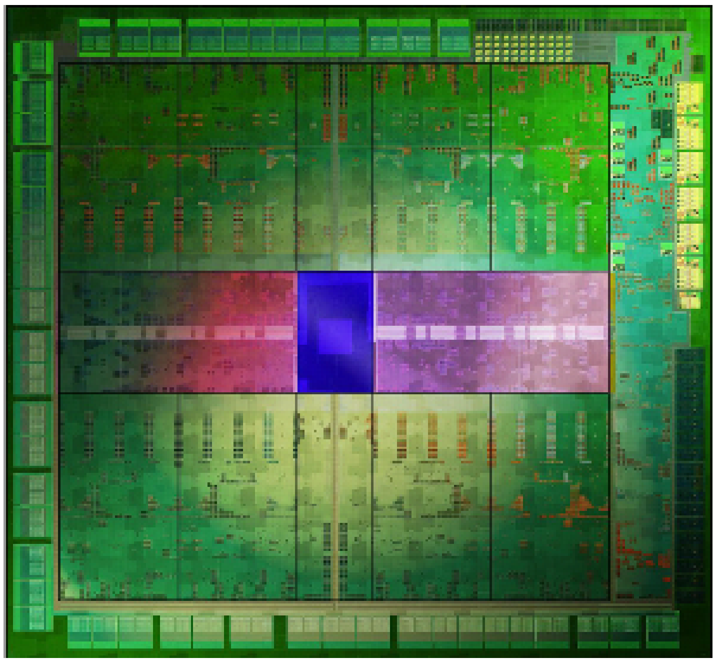


Figure 2.2: Nvidia Kepler Architecture die illustration [3]

Chapter 3

Cache

The memory hierarchy of GPUs starts with Dynamic Random Access Memory. GPGPUs utilise a higher capacity and speed DRAM compared to GPUs to store a larger amount of data on board. The next level in the GPU memory hierarchy is the multilevel cache then the register file. The multilevel cache has level 1 comprising the fastest cache but lower capacity, the following levels of cache have a higher capacity but lower speed. All cache levels are much faster than the DRAM but smaller in capacity. The register files in GPUs are at the top of the memory hierarchy. The cache is a high-speed data storage component type that forgoes large data capacity for varying levels of speed up. Cache utilises temporal and spatial locality of data to store data that is frequently required or data that might be required next. The low storage capacity of the cache necessitated an investigation of different utilisation policies designed to take advantage of the limited size. A brief description of these policies is also listed in the appendix as they were utilised in the hybrid SRAM architecture.

Cache Placement Policy

Cache placement policies determine how a cache is divided and where data can be placed within a cache. There are three placement policies [22], namely:

- Directly mapped : A cache is divided into a number of sets that can only hold a single cache block. A block is placed into a particular set depending on the block's address in the main memory. This placement policy is power efficient and requires less hardware as searching for a block simply requires checking a single location in the cache.
- Fully associative : The cache has a single set that stores all the cache blocks. The cache blocks can be placed anywhere in the cache so that the cache can be fully utilised. This policy is the most computationally expensive to implement as the entire cache must be searched during a cache request.
- Set-associative : This policy offers a mix of directly mapped and fully associative placement policies. The cache is divided into a number of sets which hold a fixed number of cache blocks. During a cache request the necessary set is calculated then all blocks within the set must be searched. The number of cache blocks that can occupy a set is referred to as associativity. This policy also does not fully utilise a cache.

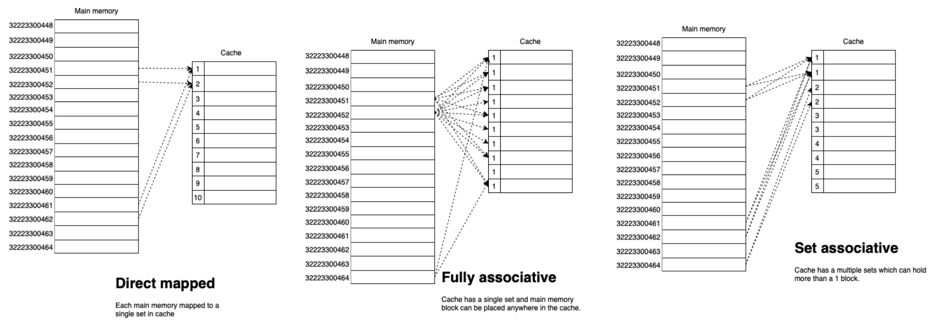


Figure 3.1: Cache placement policy

The cache placement policies are illustrated in figure 3.1.

Cache Replacement Policy

Cache replacement policies describe which cache blocks in a full cache set should be removed to make space for another cache block. There are several different cache replacement policies such as first in first out, first in last out, and simple eviction. This thesis project used the following policies:

- Random : This policy randomly selects a cache block within a set and discards it.
- Least Recently Used (LRU) : This policy removes the least recently used cache block within a set.
- Least Frequently Used (LFU): This policy keeps track of how often each cache block is accessed and replaces the least used block.

LRU and LFU produce more consistent results, however require processing overhead to implement. Depending on the cache access pattern, LRU and LFU can result in wasted space as a cache block will not be removed. For example, a block that is repeatedly accessed at the beginning of a program then never again will not be removed if an LFU policy is implemented.

3.1 GPU Cache

SOTA GPGPU caches are designed with features that are common with CPU caches such as multilevel caches, large cache capacities, and more [23]. The cache of the GPU is managed by memory controllers while the shared memory is a type of scratchpad memory that is managed by code being run on the GPU. The shared memory uses part of the same physical L1 cache. This arrangement is specific to GV100 (Volta architecture) and can differ between GPGPUs.

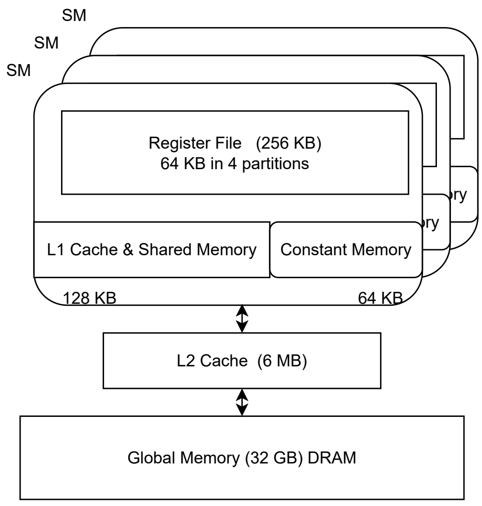


Figure 3.2: The memory hierarchy of the Nvidia GV100

3.2 GPU Cache Operation

All data transfers to and from GPU go through the L2 cache as shown in figure 3.2. The L1 data cache is used to reduce memory access latency to lower levels in memory hierarchy while shared memory is used to store data that is shared between threads. The constant cache is used for read accesses to read-only data. Cache word line size is 128 bytes and is used to transfer cache lines between caches, registers, and processing cores.

3.3 Static Random Access Memory

Caches are typically implemented with SRAM. SRAM is a form of Random Access Memory that employs circuitry to store bits. It is a volatile memory, requiring a power supply to maintain data stored in the circuits. There are a variety of circuit architectures that accomplish storing bits and the 6 transistors shown in Figure 3.3 is the most commonly used SRAM cell architecture. Other architectures replace 2 of the transistors with resistors or are in a different configuration with varying numbers of transistors from 4 transistors to more than 10 transistors to store a single bit [24][25][26]. The different architectures have unique benefits and drawbacks such as manufacturing complexity, latency, and die area.

3.3.1 Operation

The SRAM cell in figure 3.3 is a single port bitcell with 3 modes of operation namely standby, reading, and writing. The transistors M_1 and M_2 form an inverter and transistors M_3 and M_4 form another. The inverters are looped and connected to each bit line through pass transistors M_5 and M_6.

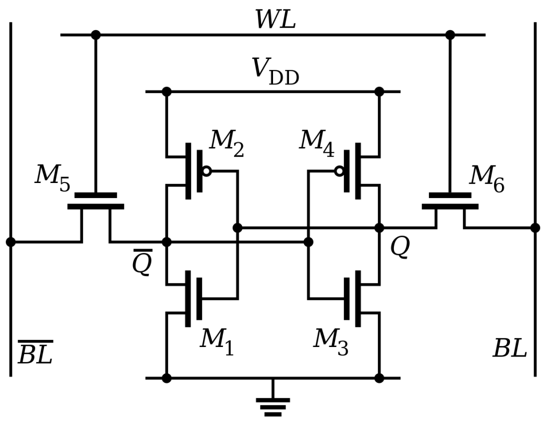


Figure 3.3: The 6 Transistor SRAM cell [4]

Standby

In this mode of operation the word line is not asserted and transistors M_5 and M_6 operate in the off state. The value of the bitcell remains latched in a positive feedback loop. The value of the bitcell will be stored at either inverter output (designer's choice) as long the inverters remain powered.

Reading

In this mode of operation, the bit lines BL and $\overline{BL}$ are pre-charged to supply voltage then the word line is asserted. The asserted word line turns on the pass transistors M_5 and M_6 which causes one of the bit lines to drop by discharging through the bitcell. The drop in voltage is detected by a sense amplifier(SA) connected to both bit lines. The value read by the SA is determined by which bit line experienced a drop in voltage. The SA is a circuit designed to amplify the small changes in voltages found in the bit lines.

Writing

In this mode of operation, the value being written is applied to BL and the inversion of the value is applied to $\overline{BL}$. The word line is asserted high and the value is latched between the 2 inverters. This is achieved through sizing of transistors to have stronger NMOS pass transistors than the PMOS transistors (M_2 & M_4) to override the value already stored in the bitcell.

3.4 Pseudo Random SRAM

The pseudo-random SRAM (PR-SRAM) is a novel SRAM architecture that trades latency for a significant reduction in dynamic energy consumption. PR-SRAM is also capable of operating at a higher operating clock frequency compared to conventional SRAM. The PR-SRAM read operation is illustrated in figure 3.4, which highlights the main drawback of this SRAM architecture. It requires accesses to addresses that are spaced far enough apart to not cause penalty cycles which occur when reads are made to the same zone. The waveform shown in figure 3.5

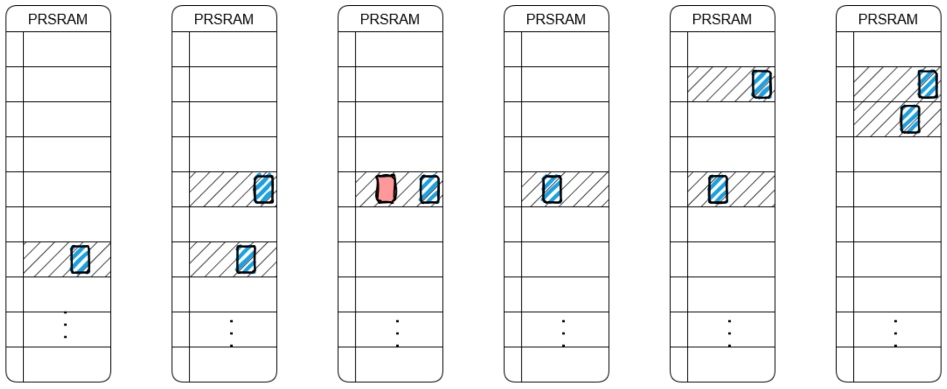


Figure 3.4: PR-SRAM 2 cycle pipeline operation

highlights the access conflict of the read operation for 4 cycle pipeline implementation. Figure 3.5 shows a 4 read cycle pipelined PR-SRAM. The first 4 reads

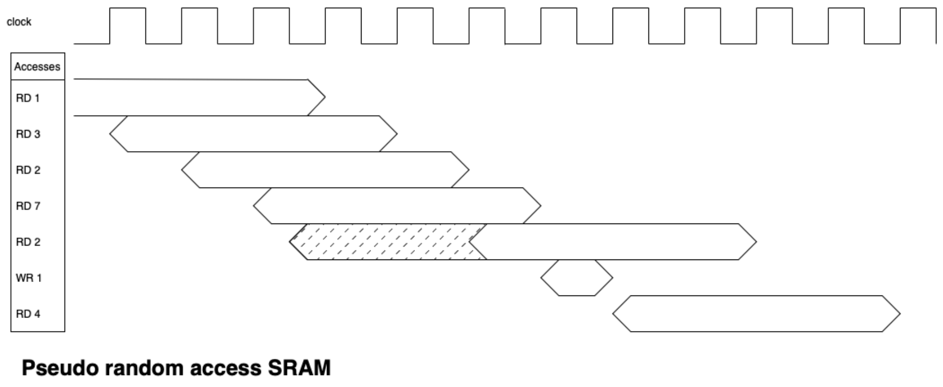


Figure 3.5: PS-SRAM waveform for 4 cycle pipeline

are performed on different SRAM zones and are processed immediately. The next read instruction results in a zone conflict as zone 2 is already being accessed. All accesses to the SRAM are stalled until zone 2 is cleared. The number of penalty cycles or cycles the SRAM is stalled for is shown in a hatched pattern in figure 3.5. The best-case scenario implementation of PR-SRAM would require a predetermined access pattern that can be utilised to overcome the main PR-SRAM drawback which results in penalty cycles. Another weakness with the PR-SRAM is the increase in latency as data will always take longer from request to receiving unless the PR-SRAM is run at a higher clock frequency. Because of these weaknesses, its suitability for use in GPGPUs has to be analysed.

Chapter 4

Methodology

Exploring the suitability of the PR-SRAM in GPGPUs required a way to test the PR-SRAM under GPU workloads. Nvidia GPUs run Compute Unified Device Architecture (CUDA) code. When CUDA code is run on a GPU, it is compiled to parallel thread execution code (ptx), then the CUDA compiler compiles the ptx code into streaming assembler code (sass). Streaming assembler code is optimized machine code specific to the target GPU architecture. Nvidia provides profiling tools for programmers to check different performance metrics of GPU running their code. The tools are capable of outputting .ptx and .sass snippets of code being run, however, the code does not contain any information referencing cache addresses being accessed. The memory profiling tool they provide also does not show which cache addresses were being accessed as they are hardware managed by the memory controllers. Short of having access to closed source Hardware Description Language (HDL) of a GPU, retrieving all cache addresses would require a GPU simulator capable of accurately simulating the operations of GPU caches. Since most details of GPUs are closed source, researchers have reverse-engineered details of GPUs including GPU cache through microbenchmarking like Saksham [27] did to determine the L2 cache set addressing in the GTX1080 GPU. Their results were used to create more accurate GPU models to simulate.

4.1 Project Flow

Figure 4.1 shows the project flow used to retrieve cache accesses and to analyse the energy consumption and performance metrics of the different runs. GPGPU-Sim[1] was the chosen GPU simulator for this thesis. The source code of GPGPU-Sim had to be altered to write out all cache addresses utilised as well as other information related to each cache access. GPGPU-Sim used a memory-fetch object to store all details about an individual memory request and a memory configuration object to store memory configuration about a specific memory module being accessed. The memory-fetch object stored details about each individual memory request such as the access type, data size, multiprocessor core number, and more. The memory configuration object kept track of the configuration of the caches, the configuration included the number of cache lines, set indexing algorithm, and cache associativity number among other attributes. Most of the memory fetch and memory configuration attributes were written out to databases and analysed by

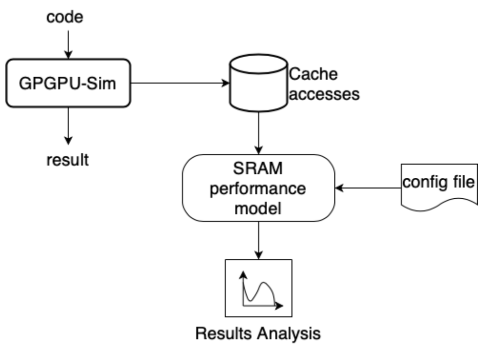


Figure 4.1: Thesis project flow

the SRAM performance model.

4.2 Benchmarks

This thesis project focused on AI/ML workloads, therefore the benchmarks that were run are some of the most frequently used operations in AI/ML applications. A brief explanation of the algorithms used is highlighted in the appendix. The benchmarks utilised in this project are listed below:

- Neural Net (NN)
- 2 Dimensional Convolution (CONV2D)
- 3 Dimensional Convolution (CONV3D)
- Back-propagation (BACKPROP)
- Speckle Reducing Anisotropic Diffusion (SRAD)

4.2.1 Neural Net

Neural networks are computing models made to function like neural networks in human brains [28]. They comprise a large number of nodes called artificial neurons, which are also modeled after neural circuity in biology. The artificial neurons have a number of inputs, weights, and a bias which are summed and the result is sent out as an output. They are arranged into layers as shown in figure 4.2.

Each neuron processes its inputs and forwards the result to the next layer. The weights and biases are adjusted during the ML process. This basis of operation allows neural networks to perform various tasks such as predictive modeling, pattern recognition, medical diagnosis, and more. SOTA neural networks are composed of exceptionally large networks such as the Palm Language Model trained on 540 billion parameters [29]. Parameters in a neural network refer to the sum of all the inputs to each individual neuron and the weights and biases of every neuron in the

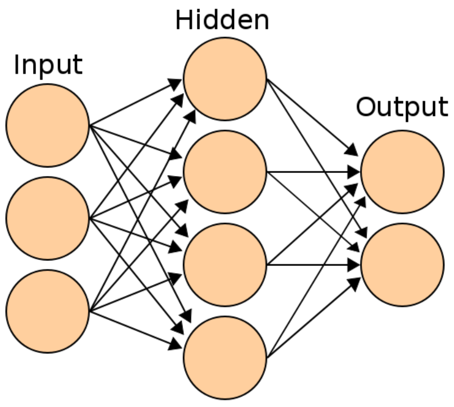


Figure 4.2: Artificial neural network [5]

neural network. These large networks require large processing power and are run on numerous SOTA GPGPUs or ASICs. This benchmark runs a trained neural net to perform number recognition on a number of images.

4.2.2 CONV2D

Convolution is a mathematical operation performed on 2 functions which results in a function that is the integral of the product of the 2 functions after reflecting one of them in the y-axis. Discrete convolution is used in signal processing and is visually represented in figure 4.3. The functions being convoluted are discrete values in a 2-dimensional array which produces a 2-dimensional array as the result. Convolution is an important operation with uses in probability, image processing,

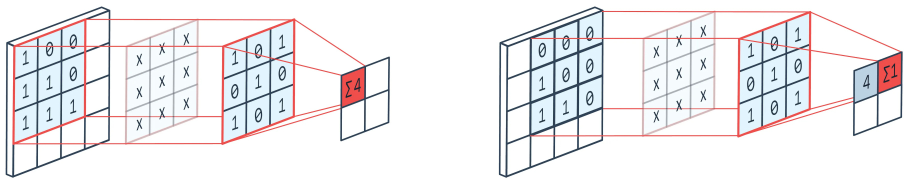
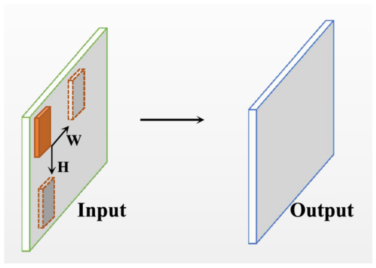


Figure 4.3: 2D convolution operation [6]

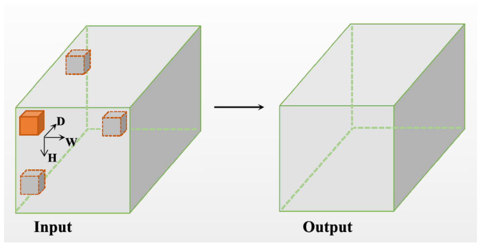
signal processing and more. Convolution is used in convolutional neural networks which apply multiple cascaded convolution kernels in AI applications. It is a highly parallel and computational operation which would be important for benchmarking GPGPU for AI/ML applications.

4.2.3 CONV3D

3 dimensional convolution is the same operation as 2D convolution except the arrays have an extra dimension.



(a) 2D Convolution



(b) 3D Convolution

Figure 4.4: 2D convolution vs 3D convolution [7]

Figure 4.4 shows the difference between 2D and 3D convolution. 3D and higher dimensional convolution are being used in increasingly more complex AI/ML neural nets. From figure 4.4, the 3D convolution would have more repeated accesses as an input value that could be used 3 times rather than 2 times. The performance difference between 2D and 3D convolution in PR-SRAM would also be intriguing to observe.

4.2.4 BACKPROP

Back-propagation is short for backward propagation of errors and is an algorithm used in supervised ML to train NN using the gradient descent method. This operation calculates the gradient of an error function with respect to neuron weights and updates respective neuron weights (parameters) with the result. The result of the operation is also used by previous layer neurons to calculate their respective errors. This calculation propagates backward throughout the network and is repeated numerous times during the learning process. The algorithm was invented in 1962 by Frank Rosenblatt [30] but saw limited use due to the computational cost. Back-propagation performs the same operation on a large amount of data repeatedly, therefore it is easily exploitable by the SIMD structure of GPGPUs. It is the main ML algorithm in use to train AI. This benchmark performs backpropagation of a face recognition benchmark[31].

4.2.5 SRAD

Speckle reducing anisotropic diffusion is not particularly used in AI/ML. It was added as image processing is another wide use case of GPUs. The results of SRAD can be used with AI to perform medical diagnosis. A speckle is a locally correlated noise that affects imaging applications such as medical ultrasound imaging[32]. SRAD utilises partial differential equations (PDEs) to remove speckles from images without destroying important image features. SRAD is performed in several stages with synchronisation requirements therefore each kernel operates on a single stage. This benchmark performs the SRAD operation which includes: image extraction, image processing, and image compression. This benchmark was chosen for this project to observe PR-SRAM effects with an image processing algorithm.

4.3 SRAM Performance Model

The SRAM performance model is a python project that uses the cache addresses output of the GPGPU-Sim to calculate the dynamic energy consumed over the run of the benchmark. The cache addresses that are stored in a database are accompanied with the address in the GPU DRAM, cache set number, cache bank number, clock cycle, GPU core and more. A parser object is called to read the

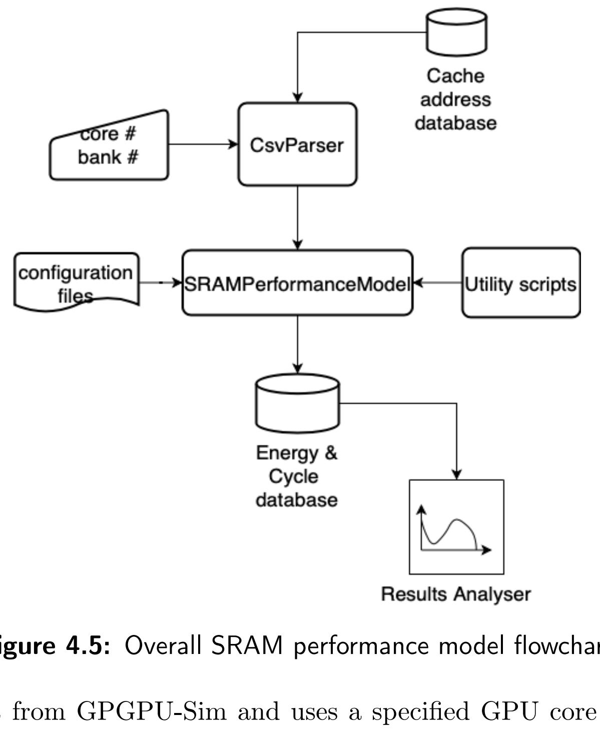


Figure 4.5: Overall SRAM performance model flowchart

database result from GPGPU-Sim and uses a specified GPU core number or L2 cache bank number to parse and sort all accesses. The level 1 cache database is parsed with a GPU core number as level 1 caches are unique to GPU cores. The result from the CsvParser is used by the Performance model to calculate energy and conflict cycles for all accesses. A configuration file with information such as

PR-SRAM pipeline length, clock frequency, and energy ratio is read by the model. The energy ratio is the ratio of low-energy standard SRAM dynamic power consumption to the PR-SRAM dynamic power consumption. It was extrapolated and standardised from in-house energy datasheets for conventional low-power SRAM and the PR-SRAM at Xenergic. The model implemented a pipeline to perform PR-SRAM zone conflict detection. Accesses are loaded into the pipeline on its specified clock cycle. If there is no cache access at a particular clock cycle then no operation is inserted into the pipeline to continue SRAM operations already in the pipeline. The zone conflict detection algorithm is also elaborated upon in the appendix.

Results from the SRAM performance model showed a higher than expected number of zone conflicts. A hybrid SRAM model was proposed to reduce the number of zone conflicts. It was implemented with a list of different conventional SRAM placement models. A utility script was used to test and implement different PR-SRAM zoning algorithms. The script also implements another write-after-read (WAR) hazard detector as a write operation that takes a single clock cycle may complete before a read operation finishes. The results of the model are stored in a database listing the test configuration of each individual run as well as the accumulated energy and penalty cycles. A separate results analyser is used to standardise, summarise and plot the results required.

Chapter 5

Results and Analysis

5.1 GPGPU-Sim

GPGPU-Sim[1] is one of many research-backed open-source GPU simulators available. It was used for this thesis as it was a popular homogeneous GPU simulator that produced cycle-accurate results with a high correlation to actual GPUs. Homogeneous simulators simulate only a GPU, while heterogeneous simulators simulate CPU and GPU which isn't necessary for this project.

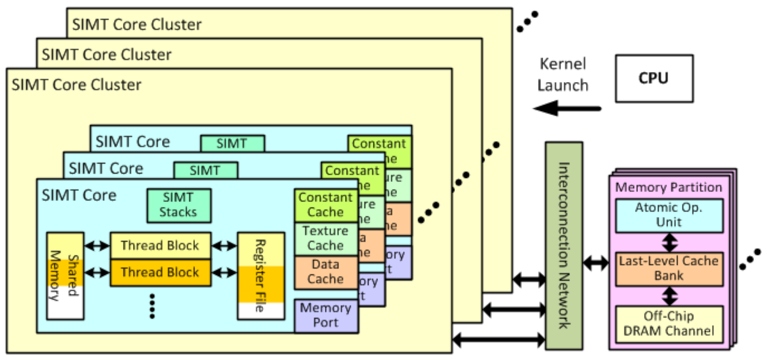


Figure 5.1: GPGPU-Sim overall architecture

The overall architecture of the GPU model used in the simulator is shown in 5.1. It is composed of multiple single instruction multiple thread clusters, an interconnect network, and a memory partition. The simulator features a fetch-decode and instruction issue in the front end and uses a per-warp SIMT stack to perform execution. The simulator uses a scoreboard algorithm to perform write-after-write(WAW) and read-after-write(RAW) hazards on a per-warp basis. Register access in GPGPU-Sim is implemented with an operand collector which is based on Nvidia patents [33]. The GV100 was simulated as it was the most up-to-date and tested GPGPU available on the simulator. Utilised the findings from [27] and the GV100 configuration file, to determine the cache organisation as shown in figure 5.2. The GV100 has a 6 MB L2 cache as was summarised in table 2.1. It is divided into 64, single-port cache banks. Each L2 cache bank is a 24-way set associative cache with 768 words of 128B. The cache banks are individually

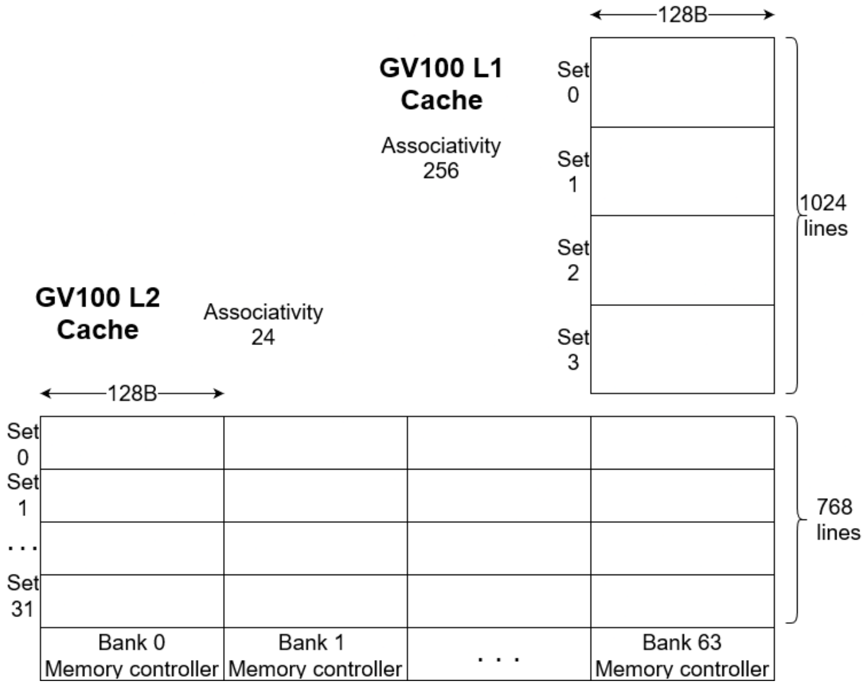


Figure 5.2: GV100 Cache organisation

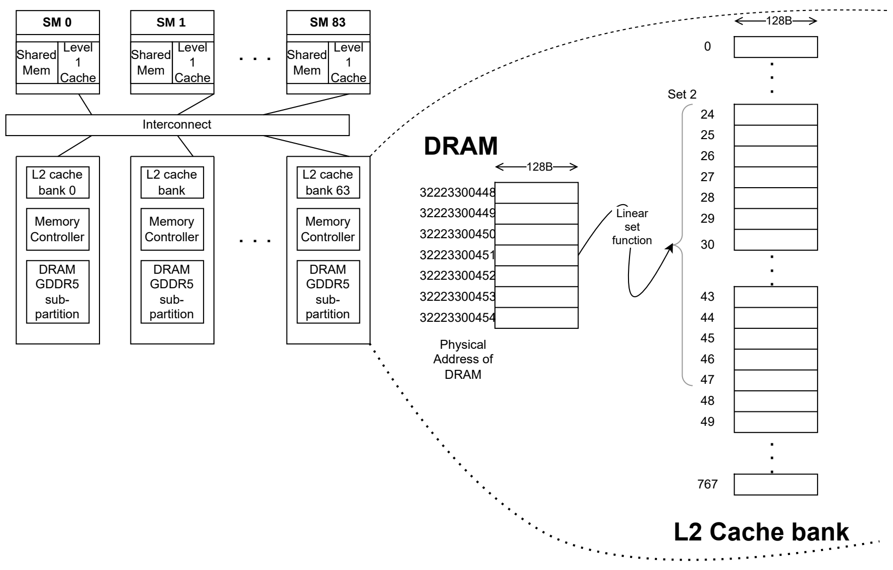


Figure 5.3: L2 cache partitioning and cache address allocation

managed by a memory controller that also controls a specific partition of the DRAM. The L1 cache in each SM is a 256-way set associative cache of 1024, 128B words. The cache replacement and set policies were already found by previous researchers and implemented into the simulator. The set placement in the L2 cache is assigned through a hashing algorithm applied to the physical address of data requested from DRAM. Figure 5.3 shows how the data fetched from DRAM is assigned to the L2 cache.

5.2 Performance model

The performance simulator was used to test a conventional SRAM against a PR-SRAM with the retrieved access sequences from GPGPU-Sim. Cache requests to cache bank number 2 in the GV100 L2 cache were used for the performance model in figure 5.4a, to compare 5 different benchmarks with and without the PR-SRAM. An explanation of the benchmarks is written in the methodology. There is also a summary which lists their explanations in the appendix.

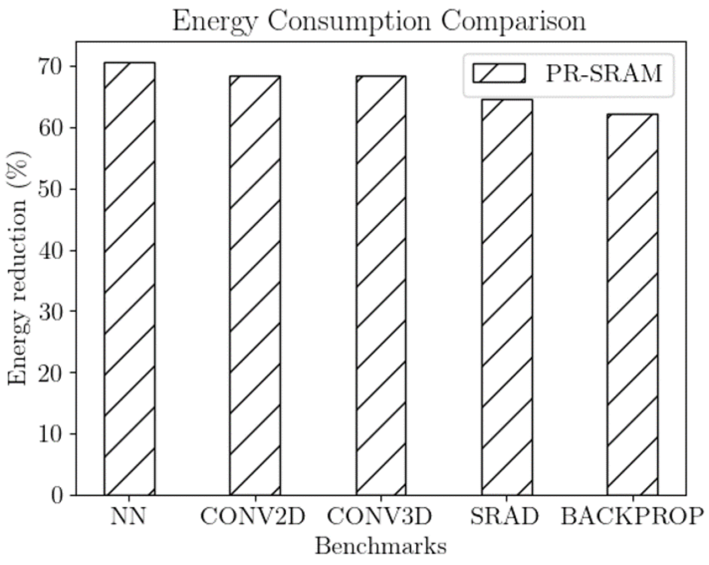
There was a greater than 60% energy reduction using PR-SRAM for all benchmarks with a pipeline factor of 4. Figure 5.4a shows a constant value for PR-SRAM energy consumption because, for each benchmark, a ratio was calculated between the 2 SRAM types.
The effect of pipelining can be seen in figure 5.4b where the number of penalty cycles is shown for each benchmark. A deeper PR-SRAM pipeline architecture results in greater energy efficiency than a smaller PR-SRAM pipeline. However, a deeper pipeline increased the chances of a zone conflict as there would be more accesses in flight. The deeper pipeline also increased the number of penalty cycles when a zone conflict inevitably occurred. From figure 5.4b, the 16-cycle pipeline resulted in orders of magnitude higher penalty cycles compared to a 2-cycle pipeline. The number of penalty cycles is also quite dependent on the code being run on the GPGPU which could impact performance if the number of penalty cycles is high. To counteract this effect, this thesis also proposes a hybrid SRAM architecture that can be used to save energy with non-conflicting reads and reduce penalty cycles accumulated by servicing repeated cache accesses with a conventional SRAM.

5.2.1 Hybrid SRAM

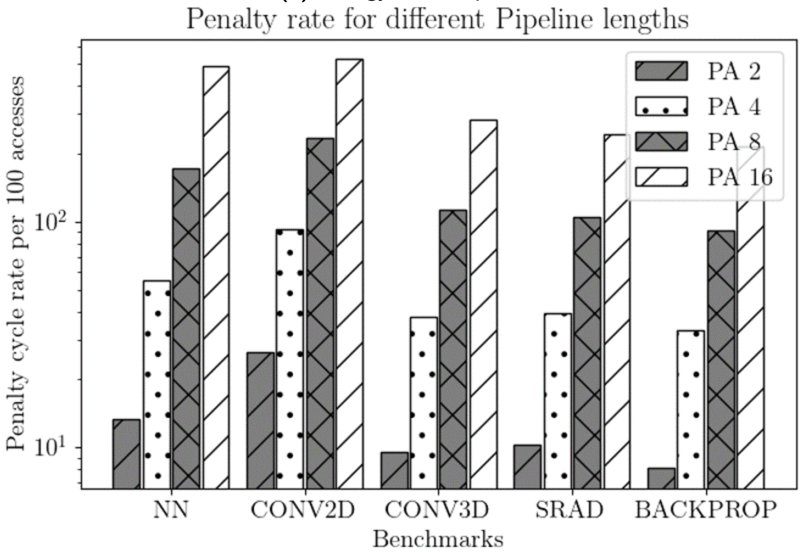
The hybrid SRAM model created in this section used the higher capacity PR-SRAM to perform all SRAM accesses. The conventional SRAM will be used with accesses that lead to zone conflicts. During a zone conflict, the zone conflict is checked and if the zone conflict is due to a re-access of the same address already being read, it is redirected to the conventional SRAM which will perform all subsequent operations for that specific operation. This procedure is illustrated in the waveform in the figure 5.5.

Normal zone conflict

This section expounds on the first zone conflict indicated in figure 5.5. The hybrid PR-SRAM in figure 5.6 has 4 zones which hold 2 data blocks each. The conven-



(a) Energy consumption



(b) Penalty cycles rate

Figure 5.4: Comparative analysis of PR-SRAM and conventional SRAM

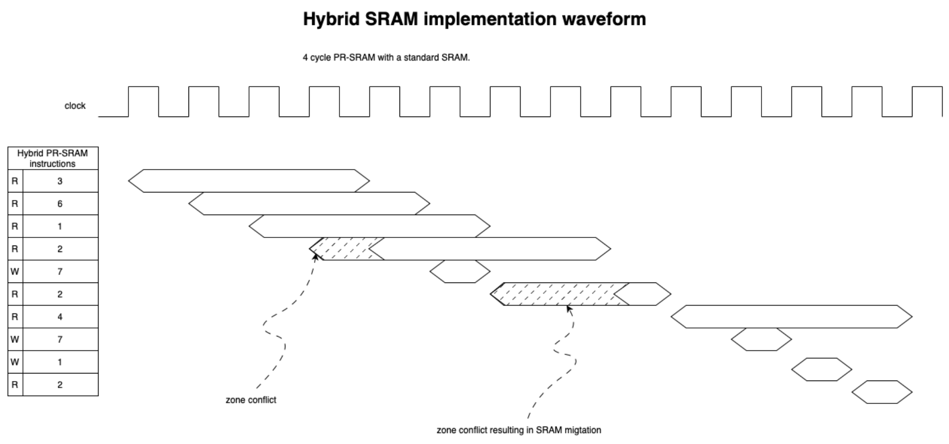

Hybrid SRAM implementation waveform
4 cycle PR-SRAM with a standard SRAM.
clock
Hybrid PR-SRAM instructions
R 3
R 6
R 1
R 2
W 7
R 2
R 4
W 7
W 1
R 2
zone conflict
zone conflict resulting in SRAM migtation

Figure 5.5: Hybrid SRAM waveform

tional SRAM in this example is fully associative and can store 4 data blocks. The PR-SRAM already has data in all locations while the conventional SRAM is empty. Figure 5.6, follows the instructions shown in figure 5.5. The first 3 read operations to addresses 3, 6 and 1 were initiated without causing any zone conflicts. The next operation, a read to address 2, however resulted in a zone conflict. Zone 2 was already occupied by a read operation on address 3, therefore the pipeline was stalled for 1 additional penalty cycle as shown in figure 5.5. Since the requested address, 2 is different from the address already in the pipeline, 3, there was no utilisation of the conventional SRAM.

SRAM migrating zone conflict

This section details which zone conflicts result in migrating data to the conventional SRAM. Rather than migrating the data of all zone conflicts to the conventional SRAM, only the same access conflicts were moved to the conventional SRAM. This prevented unnecessary transfers to the conventional SRAM, which would have occurred during every zone conflict and wasted resources. Figure 5.7 depicts the zone conflict that causes the requested data to be moved to the conventional SRAM. The read operation to address 7 is started then in the next clock cycle, the read instruction to address 2 causes a zone conflict. After zone conflict has been detected the conflicting addresses are checked. In this case, since they are the same address, the result from the preceding access will be forwarded to the conventional SRAM. The conflicting request will be completed with the data in conventional SRAM which will fulfill all subsequent requests to the data block that was stored in address 2. As shown in the waveform in figure 5.5 the next request to address 2, is completed in a single clock by the conventional SRAM storing the data in address 1 (in the conventional SRAM).

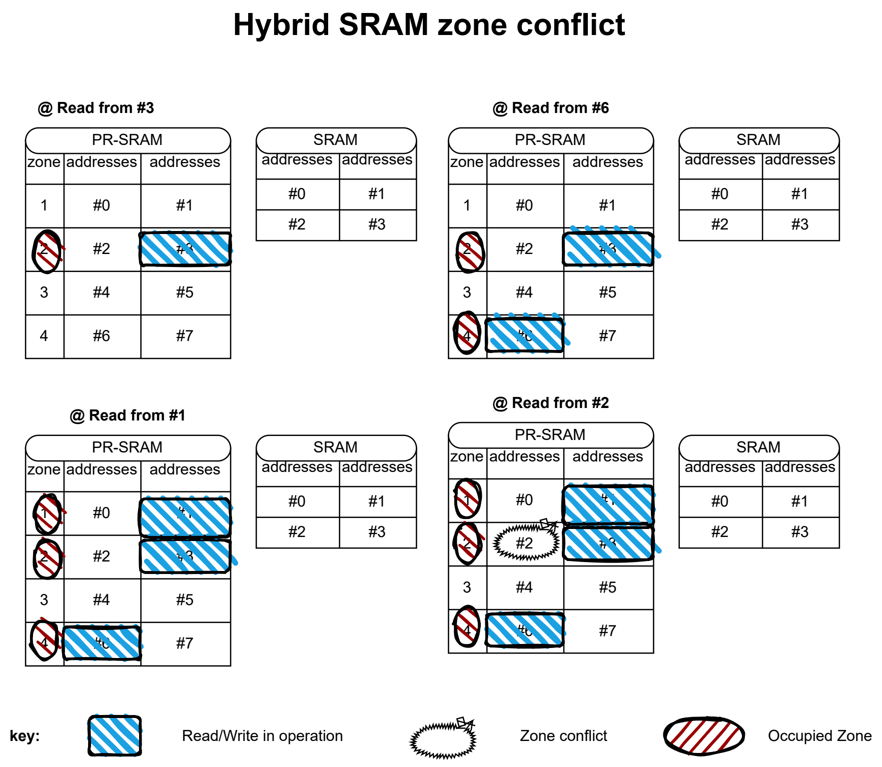


Figure 5.6: Illustration of the 1st zone conflict

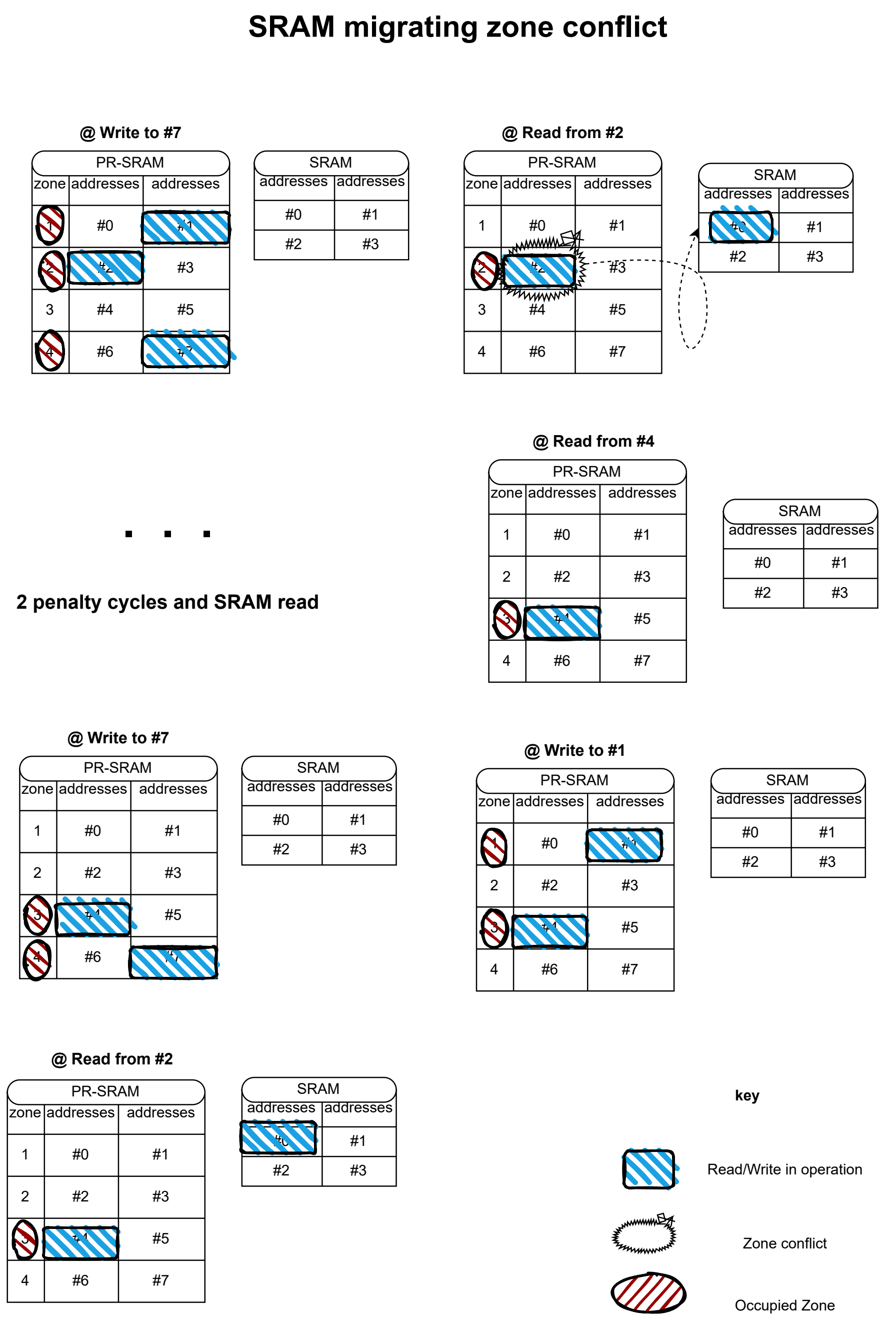


Figure 5.7: Second zone conflict illustration

Conventional SRAM analysis

The following section investigates different caching policies and SRAM sizes to find the configuration's effect on energy and performance.
Several benchmarks were performed with different conventional SRAM capacities to analyze the effect of its capacity. The disjoint in figures 5.8b and 5.8a is because the graphs start with the results of a pure PR-SRAM instead of the smallest possible hybrid SRAM design. Figure 5.8a shows the effect of varying the size of the conventional SRAM on the number of penalty cycles, while figure 5.8b shows its effect on energy. The pure PR-SRAM with 0Kb conventional SRAM had about 55 penalty cycles for every 100 SRAM accesses while a 2Kb conventional SRAM reduced the penalty rate to 30 cycles. For the same reduction in penalty cycles, the energy consumption increased by about 1.3 times as shown in figure 5.8b. As the conventional SRAM is increased, more accesses are performed by it and the energy consumption increases as a result. A hybrid SRAM could be configured to use just the PR-SRAM to provide greater energy efficiency when running a latency-insensitive program on the GPGPU.

The limited capacity of the conventional SRAM used in this hybrid solution cannot store all the required conflicting addresses. The SRAM was configured with different replacement policies to assess each policy with each benchmark. The obtained results are listed in the table 5.1. A summary analysis of the hardware

	NN	CONV2D	CONV3D	SRAD	BACKPROP
LRU	3.23	36.82	20.29	33.25	18.55
LFU	11.92	37.36	20.88	34.41	18.24
RAND	3.73	36.96	20.43	33.60	18.55
PR-SRAM	54.87	92.3	38.01	39.3	33.23

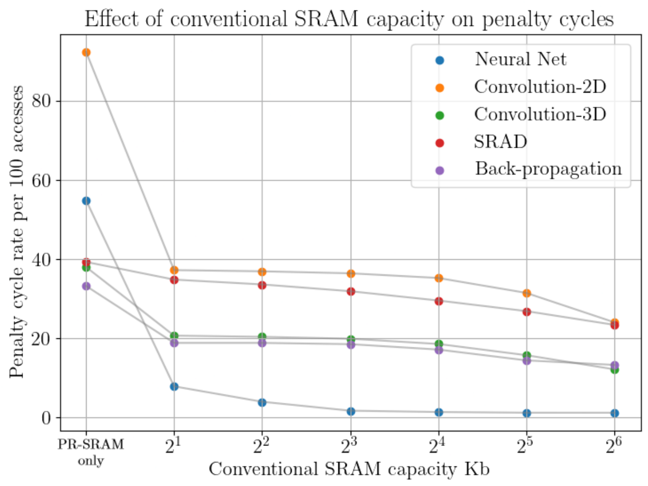
Table 5.1: Penalty rate for different replacement policies

requirements and processing required for implementation of the different policies is shown in table 5.2. The hardware requirements to implement a random re-

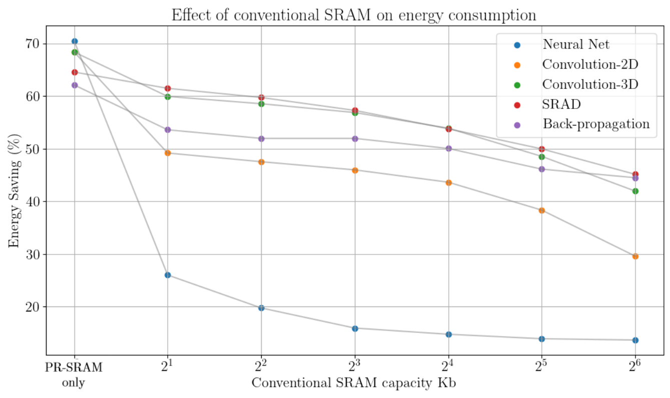
	LFU	LRU	Random
Hardware Requirement	High	High	Low
Processing Overhead	High	Medium	Low
Miss rate (averaged)	24.562	22.428	22.654

Table 5.2: Summary of replacement policies

placement cache are quite low. The random number generator (RNG) is the most important component for the random placement and it is an uncomplicated circuit. There are numerous methods of implementing an RNG in hardware such as through a pair of cross-coupled inverters[34], a free running oscillator[35] or thermal noise sampler[36]. This can be achieved with a couple dozen NAND gates. However, a combination of different methods is usually used to generate a random number depending on the requirements of randomness the circuit requires[37]. The



(a) Penalty cycle rate effect



(b) Effect on energy consumption

Figure 5.8: Analysis of conventional SRAM capacity effects on penalty cycle rate and energy consumption

hardware requirements to implement LFU and LRU are much higher than that of random placement cache as it requires SRAM access history. Shift registers and other storage circuitry are required for the implementation of these replacement algorithms. LFU has the highest processing overhead as it requires a history of every location in SRAM and has to update every time a location is accessed. LRU could be implemented with a shift register always shifting out the least recently used address. This would require less processing as compared to LFU. The level of processing required to implement the different replacement policies was obtained from another research paper [38].
The overall impact of PR-SRAM latency on the performance of the GPGPU cache

is minimal due to the amount of idle time between bursts of cache usage. The ratio of total cycles to the number of accesses is shown in table 5.3

Benchmark	NN	CONV2D	CONV3D	SRAD	BACKPROP
Access rate	0.027	0.080	0.095	0.099	0.059

Table 5.3: The ratio of total accesses to total number of clock cycles required by each benchmark

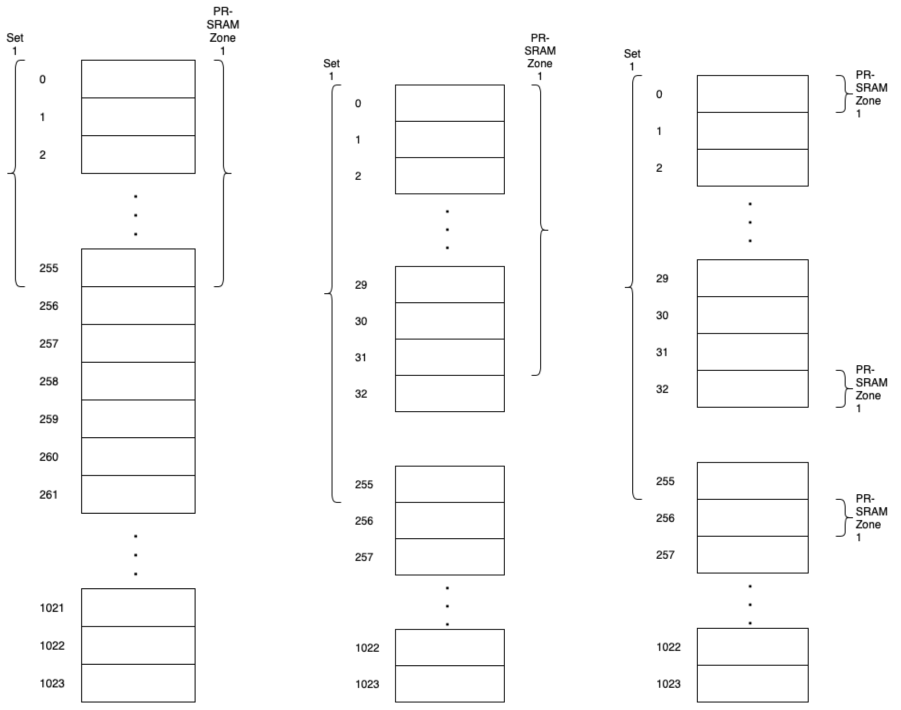
5.3 Analysis on level 1 GPGPU cache

Results analysed in the previous section were obtained from analysis of the level 2 GPGPU cache which was zoned according to the sets already implemented. Aligning the PR-SRAM zoning with sets in the cache produced better results with fewer zone conflicts and the number of cache sets allowed for easier PR-SRAM zoning. The level 1 cache of the GV100 has 4 sets with an associativity of 256. Four zones would almost always result in a zone conflict so the PR-SRAM had to be implemented with more than 4 zones.
The different zoning methods were investigated to prevent the excessive penalty rate, they are visually represented in figure 5.9. PR-SRAM Zoning 1 in table 5.4 represents the set defined zoning, whereby PR-SRAM zones are aligned with the 4 sets in the level 1 cache as depicted in figure 5.9a. Zoning methods 2 and 3 reduce the number of penalty cycles by increasing the number of zones to 32. Their implementation is shown in figures 5.9b and 5.9c respectively. Zone method 2 implements 32 zones with each zone made of 32 contiguous locations while zone method 3 implements zones by zoning locations over strides of 32. The sets are used by the GPU during use to quickly locate data within a cache, while the PR-SRAM zones are used by the PR-SRAM memory controller to perform pipelined SRAM accesses. The high level 1 cache associativity of 256 creates a high cache with high utilisation. This can be used in conjunction with smaller PR-SRAM zones to improve cache utilisation and reduce the penalty cycle rate.
Table 5.4 summarises the results from running the same benchmarks for a set-defined implementation and for two different 32 zone implementations. The fol-

PR-SRAM Zoning	Set-defined	Contiguous	Non-contiguous
Average Penalty Rate	106.56	99.94	38.40

Table 5.4: Summary of penalty cycles for different zoning implementations

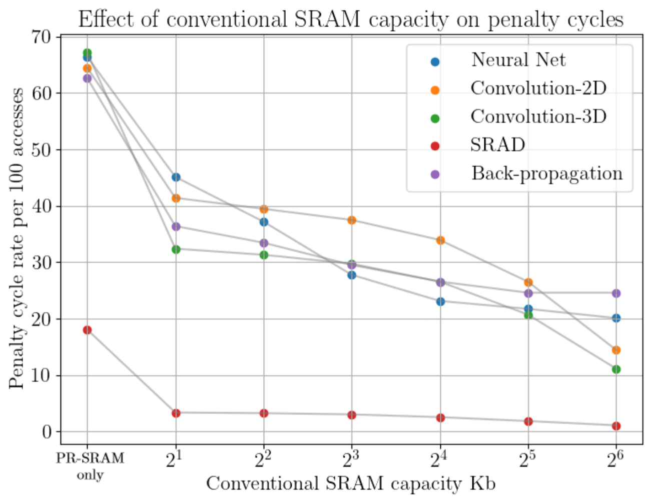
lowing data shown in the plots were obtained for a PR-SRAM using the non-contiguous zoning method in the hybrid SRAM design in the level 1 cache. The penalty rate of the PR-SRAM implementation in the level 1 cache is shown in figure 5.10a. The first point of the different benchmarks shows the penalty rate of the pure PR-SRAM. The trend of increasing the conventional SRAM The PR-SRAM



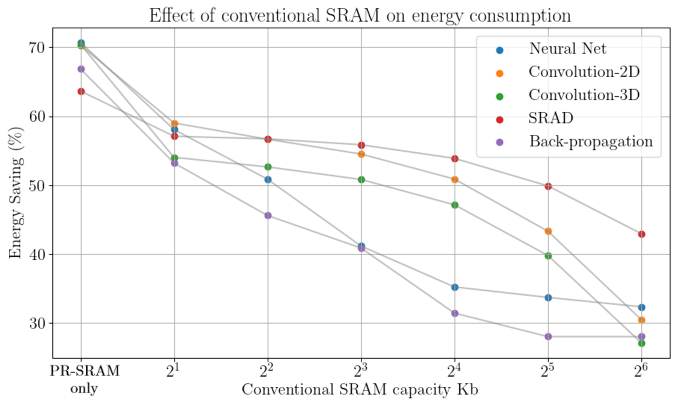
(a) Set defined zoning **(b)** Contiguous zoning **(c)** Non-contiguous zoning

Figure 5.9: Different zoning methods for PR-SRAM

energy saving is shown in figure 5.10b. The first point for each of the benchmarks is the energy saving of a pure PR-SRAM. The subsequent points of the graphs show the effect of increasing the conventional SRAM capacity.



(a) Penalty cycle rate effect



(b) Effect on energy consumption

Figure 5.10: Analysis of conventional SRAM capacity effects on penalty cycle rate and energy consumption

Chapter 6

Conclusion

To investigate the suitability of PR-SRAM in GPGPU caches, GPGPU cache access patterns were obtained from a GPU simulator running code. The L1 and L2 GPGPU cache accesses were used in a software model to obtain energy and cycle readings for different AI/ML focused benchmarks. The benchmarks used were neural-net, convolution, back-propagation, and SRAD. This project found greater than 60% reduction in dynamic energy consumption when a PR-SRAM was utilised over a conventional SRAM. The drawback found was an increase in latency (55 more cycles(penalty cycles), every 100 cache accesses) in PR-SRAM which did not significantly impact performance as GPGPU caches are not in continuous use in every cycle. This project also proposed several implementations of a hybrid SRAM design that could be used to balance penalty cycles and dynamic energy consumption. Hybrid designs incorporating 2kB, 4kB, 8kB, and 16kB standard SRAM were used to investigate its effect on penalty cycles and energy consumption. The hybrid design also implemented LRU, LFU, and random replacement standard SRAM to probe its effect on penalty cycles.

Chapter 7

Future Work

This project focused on the caches of GPGPUs specifically the Nvidia GV100. GPGPUs caches are not accessed as frequently as caches in CPUs as a result the effect on performance of implementing PR-SRAM is negligible. GPGPUs have large register files which are implemented with SRAM [39]. Further work can be done to investigate:

- the energy and performance effect of implementing register files with PR-SRAM rather than conventional SRAM
- the leakage power which would be affected by the additional complexity of the hybrid designs
- incorporating an accurate model of the PR-SRAM into a GPU simulator
- AI/ML cache accesses to obtain a discernible access sequence that can be used to reduce penalty cycles
- theoretical gains with a GPU compiler and architecture suited for PR-SRAM caches

The final work that could be done is a real-world implementation and comparison of a GPU with the PR-SRAM and another with the conventional SRAM.

References

[1] M. Khairy, Z. Shen, T. Aamodt, and T. Rogers, "Accel-sim: An extensible simulation framework for validated gpu modeling," pp. 473–486, 05 2020.

[2] NVIDIA, "Nvidia tesla v100 gpu architecture, the world's most advanced data center gpu," Tech. Rep. WP-08608-$001_v 1.1, NVIDIACooporation, August$2017.

[3] "NVIDIA Launches First GeForce GPUs Based on Next-Generation Kepler Architecture — nvidianews.nvidia.com." `https://nvidianews.nvidia.com/news/nvidia-launches-first-geforce-gpus-based-on-next-generation-kepler-architecture`. [Accessed 16-04-2024].

[4] Inductiveload, "Sram cell (6 transistors)," 2009. [Online; accessed May 7, 2024].

[5] Colin M L Burnett, "Artificial neural network," 2006. [Online; accessed May 7, 2024].

[6] Peltarion. `https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/blocks/2d-convolution-block/`, 2020. [Accessed 07-05-2024].

[7] C. Leng, Q. Ding, C. Wu, A. Chen, H. Wang, and H. Wu, "Bdnet: a method based on forward and backward convolutional networks for action recognition in videos," *The Visual Computer*, pp. 1–15, 10 2023.

[8] R. H. Dennard, F. H. Gaensslen, H.-N. Yu, V. L. Rideout, E. Bassous, and A. R. Leblanc, "Design of ion-implanted MOSFET's with very small physical dimensions," *Proc. IEEE Inst. Electr. Electron. Eng.*, vol. 87, pp. 668–678, Apr. 1999.

[9] G. E. Moore, "Cramming more components onto integrated circuits," 1965.

[10] L. Lerner, "Viewpoint: Mass gpus, not cpus for eda simulations," Apr 2009.

[11] M. C. Schatz, C. Trapnell, A. L. Delcher, and A. Varshney, "High-throughput sequence alignment using graphics processing units," *BMC Bioinformatics*, vol. 8, p. 474, Dec. 2007.

[12] O. Harrison and J. Waldron, "Efficient acceleration of asymmetric cryptography on graphics hardware," in *Progress in Cryptology – AFRICACRYPT 2009*, Lecture notes in computer science, pp. 350–367, Berlin, Heidelberg: Springer Berlin Heidelberg, 2009.

[13] J. Tan, K. Yan, S. L. Song, and X. Fu, “Energy-efficient gpu l2 cache design using instruction-level data locality similarity,” *ACM Trans. Des. Autom. Electron. Syst.*, vol. 25, aug 2020.

[14] L. Djinevski, S. Arsenovski, S. Ristov, and M. Gusev, “Performance drawbacks for matrix multiplication using set associative cache in gpu devices,” in *2013 36th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pp. 193–198, 2013.

[15] J. Fang, Q. Fan, X. Hao, Y. Cheng, and L. Sun, “Performance optimization by dynamically altering cache replacement algorithm in cpu-gpu heterogeneous multi-core architecture,” in *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pp. 723–726, 2017.

[16] J. Alsop, M. D. Sinclair, S. Bharadwaj, A. Dutu, A. Gutierrez, O. Kayiran, M. LeBeane, B. Potter, S. Puthoor, X. Zhang, T. T. Yeh, and B. M. Beckmann, “Optimizing gpu cache policies for mi workloads,” in *2019 IEEE International Symposium on Workload Characterization (IISWC)*, pp. 243–248, 2019.

[17] K. Choo, W. Panlener, and B. Jang, “Understanding and optimizing gpu cache memory performance for compute workloads,” in *2014 IEEE 13th International Symposium on Parallel and Distributed Computing*, pp. 189–196, 2014.

[18] M. Herlihy and N. Shavit, *The art of multiprocessor programming, revised reprint.* Morgan Kaufmann, June 2012.

[19] A. Hughes, “Chatgpt: Everything you need to know about openai’s gpt-4 tool,” *BBC Science Focus*, 2023.

[20] A. K. Z. Fan, F. Qiu and S. Yoakum-Stove, “Gpu cluster for high performance computing in proc. acm/ieee conference on supercomputing,” 2004.

[21] NVIDIA, “Nvidia h100 tensor core gpu architecture,” tech. rep., NVIDIA Cooporation, 2023.

[22] R. L. Mattson, J. Gecsei, D. R. Slutz, and I. L. Traiger, “Evaluation techniques for storage hierarchies,” *IBM Syst. J.*, vol. 9, no. 2, pp. 78–117, 1970.

[23] D. Rákos, “Understanding GPU caches 2013; RasterGrid — rastergrid.com.” `https://www.rastergrid.com/blog/gpu-tech/2021/01/understanding-gpu-caches/`, January 2021. [Accessed 07-05-2024].

[24] H. Noguchi, S. Okumura, T. Takagi, K. Kugata, M. Yoshimoto, and H. Kawaguchi, “0.45-v operating v<inf>t</inf>-variation tolerant 9T/18T dual-port SRAM,” in *2011 12th International Symposium on Quality Electronic Design*, IEEE, Mar. 2011.

[25] Y. Morita, H. Fujiwara, H. Noguchi, Y. Iguchi, K. Nii, H. Kawaguchi, and M. Yoshimoto, “Area optimization in 6T and 8T SRAM cells considering vth variation in future processes,” *IEICE Trans. Electron.*, vol. E90-C, pp. 1949–1956, Oct. 2007.

[26] S. V. Kosonocky and A. Bhavnagarwala, “Quasi-static random access memory,” 12 2005.

[27] S. Jain, I. Baek, S. Wang, and R. Rajkumar, "Fractional gpus: Software-based compute and memory bandwidth reservation for gpus," in *2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pp. 29–41, 2019.

[28] L. Hardesty, "Explained: Neural networks — news.mit.edu." `https://news.mit.edu/2017/explained-neural-networks-deep-learning-0414`, 2017. [Accessed 07-05-2024].

[29] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann, *et al.*, "Palm: Scaling language modeling with pathways," *Journal of Machine Learning Research*, vol. 24, no. 240, pp. 1–113, 2023.

[30] C. von der Malsburg, "Frank rosenblatt: Principles of neurodynamics: Perceptrons and the theory of brain mechanisms," *Brain Theory*, pp. 245–248, 01 1986.

[31] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *2009 IEEE International Symposium on Workload Characterization (IISWC)*, pp. 44–54, 2009.

[32] Y. Yu and S. Acton, "Speckle reducing anisotropic diffusion," *IEEE Transactions on Image Processing*, vol. 11, no. 11, pp. 1260–1270, 2002.

[33] S. Liu, J. E. Lindholm, M. Y. Siu, B. W. Coon, and S. F. Oberman, "Operand collector architecture," 11 2010.

[34] V. B. Suresh, "On-Chip true random number generation in nanometer cmos," 2012.

[35] B. Acar and S. Ergün, "A random number generator based on irregular sampling and transient effect ring oscillators," in *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5, 2020.

[36] N. C. Laurenciu and S. D. Cotofana, "Low cost and energy, thermal noise driven, probability modulated random number generator," in *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 2724–2727, 2015.

[37] P. Monteiro, L. Oliveira, and J. Casaleiro, "True random number generator implemented in 130 nm cmos nanotechnology," in *2022 International Young Engineers Forum (YEF-ECE)*, pp. 52–56, 2022.

[38] Q. Javaid, Department of Computer Science & Software Engineering, International Islamic University, Islamabad, A. Zafar, M. Awais, M. A. Shah, Department of Computer Science, COMSATS Institute of Information Technology, Islamabad, Department of Computer Science, COMSATS Institute of Information Technology, Islamabad, and Department of Computer Science, COMSATS Institute of Information Technology, Islamabad, "Cache memory: An analysis on replacement algorithms and optimization techniques," *Mehran Univ. Res. J. Eng. Technol.*, vol. 36, pp. 831–840, Oct. 2017.

[39] Q. Qian, "Register caching for energy efficient gpgpu tensor core computing," Dec 2023.

[40] A. Bakhoda, G. L. Yuan, W. W. L. Fung, H. Wong, and T. M. Aamodt, "Analyzing cuda workloads using a detailed gpu simulator," in *2009 IEEE International Symposium on Performance Analysis of Systems and Software*, pp. 163–174, 2009.

[41] S. Grauer-Gray, L. Xu, R. Searles, S. Ayalasomayajula, and J. Cavazos, “Auto-tuning a high-level language targeted to gpu codes,” in *2012 Innovative Parallel Computing (InPar)*, pp. 1–10, 2012.

Appendix A

Benchmarks

This thesis initially focused on AI/ML workloads, however the versatility of GPGPUs requires benchmarking the caches with a variety of algorithms. A brief explanation of the algorithms used for GPGPU benchmarking is as follows:

NN This is a neural net benchmark created by [40]. It performs number recognition on a set of 28*28 images with an already trained neural network.

CONV2D This a 2 dimensional convolution benchmark used from [41].

CONV3D This is a 3 dimensional convolution benchmark used from [41].

SRAD This stands for Speckle Reducing Anisotropic Diffusion, it is used in radar imagining applications to remove locally correlated noise without destroying important image features [31].

BACKBPROP This is a back propagation benchmark used for machine learning applications. [31]

Appendix B

Cache placement policies

The limited size of caches means that a cache word would be requested while the cache is full, therefore a cache word would have to be replaced. The conventional SRAM used in the hybrid design makes use of these cache placement policies to determine which cache word should be replaced.

RAND This is the random cache placement policy which places and replaces requested cache words randomly within a cache.

LRU Least Recently Used is a cache placement policy that always replaces the least recently used cache word with the requested cache word

LFU Least Frequently Used is another placement policy that keeps track of accesses to every cache location to determine and replace the least frequently used word with

Appendix C

Conflict detector

The zone conflict detection algorithm is illustrated in the pseudo-code shown below. The conflict detector was implemented as a class and called upon when running the performance on PR-SRAM.

Algorithm 1 Class Definition: Conflict detector

```
1: procedure INITIALIZE($size$)
2:     Constructor method to initialize the conflict detector with attributes and to implement dual cache
3:     $self.size \leftarrow size$
4:     $self.register \leftarrow [placeholder] * size$
5:     . . .
6:     $self.standard\_cache \leftarrow self.cache\_dict[self.cachename](size)$
7: end procedure
8: procedure SLIDE($value$)
9:     Method called to slide next value into the pipleine
10:    $self.register \leftarrow self.registe + [value]$
11:    return CONFLICT_DETECTION()
12: end procedure
13: procedure RUN(access)
14:    Method called every tick clock cycle to run the pipeline
15:    if data is in the standard cache then
16:        advance pipeline for a single cycle
17:    else
18:        return SLIDE(access)
19: end procedure
20: procedure CONFLICT_DETECTION()
21:    if there's no bank conflict then
22:        return 1 clock cycle
23:    else
24:        return required stalled cycles
```

Printed by Tryckeriet i E-huset, Lund 2024

Series of Master´s theses
Department of Electrical and Information Technology
LU/LTH-EIT 2024-980
http://www.eit.lth.se