

Fairness Notions on Hardware Resource Configuration

by

Aravind Vellora Vayalapra

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2023

© Aravind Vellora Vayalapra 2023

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

To meet performance and energy efficiency demand of modern workloads, specialized hardware accelerators implemented on FPGAs or ASICs have found adoption in modern servers and Systems-on-Chip (SoC). These hardware accelerators cater to a wide range of applications such as Machine Learning, databases, analytics and networking. Prior studies have shown excessive underutilization, utilization ranging from 60% to as low as 20%. Sharing hardware among multiple users can mitigate underutilization but makes achieving QoS requirements a challenge.

In this work, we explore hardware sharing on a GPU and employ game theory to present a formal model of sharing. We present notions of fairness, a performance model for heterogeneous hardware systems, and present a novel market-based mechanism to allocate and configure hardware resource. We implement the design on the open source Vortex GPU, and evaluate the system with 5 machine learning workloads – Resnet, AlexNet, YoloNet, K-Means clustering and multi-layer perceptron training. The evaluation showed that the market-based mechanism meets proportional QoS and we provide a theoretical guarantee. We observe up to 2x speedup and up to 10% higher utilization compared to traditional methods of providing equal resources. The system also chooses hardware configuration that maximizes the collective welfare of the users.

Acknowledgements

I would like to thank my supervisors Nachiket Kapre and Seyed Majid Zahedi for their insight and inspiration. I am grateful for their guidance, support, patience for my reasons and impatience for my excuses.

I would like to thank my readers, Andrew Morton and Ziqiang Patrick Huang for their time and feedback.

I would also like to express my gratitude to my family for their constant encouragement.

Finally, I would like to thank Hadi, Bernard, Mohammed, Lavanya, Ahmed and other comrades.

Dedication

To comradeship.

Table of Contents

Author's Declaration	ii
Abstract	iii
Acknowledgements	iv
Dedication	v
List of Figures	x
List of Tables	xii
1 Introduction	1
2 Background and Related Works	5
2.1 Hardware Accelerators	5
2.1.1 Vortex GPU	6

2.2	Sharing	9
2.2.1	Amdahl's Law	10
2.2.2	Fisher Markets	13
2.2.3	Collective Utility Functions	14
3	Utility Model and Mechanism	17
3.1	Motivation	18
3.2	Illustration and Overview	20
3.3	Throughput Model	22
3.3.1	PE Definition	23
3.3.2	Throughput Invariant	24
3.3.3	Formal Definitions	25
3.3.4	Modified Amdahl's Law for a Cluster	27
3.3.5	Linear Model	28
3.3.6	User model	29
3.3.7	Partitioning and Allocation	32
3.4	Fisher Market Allocation	34
3.4.1	Market model	34
3.4.2	Optimal Bids	35
3.4.3	Aggregating preferences — Social Welfare	39
3.5	Summary	40

4	Implementation	41
4.1	Modification to the Architecture	43
4.1.1	Heterogeneous Cores	43
4.1.2	Estimating Parallelization Ratio	43
4.1.3	Other Modification	45
4.1.4	Vortex GPU and the performance model	45
4.2	System Implementation	45
4.2.1	Challenges and Summary	48
4.2.2	System Implementation Details	49
5	Evaluation	53
5.1	Methodology	54
5.1.1	Benchmarks	54
5.1.2	Setting	54
5.1.3	Configuration Space and Parameters	55
5.1.4	User Sets	55
5.1.5	Metrics and Comparisons	56
5.2	Evaluation	56
5.2.1	Parallelizable fraction estimation	56
5.2.2	Market mechanism	58

5.2.3	3 Configuration Case Study	59
5.2.4	Utilization	64
5.3	Comparison to other Allocation Mechanisms	65
6	Conclusions and Future Work	70
	References	72

List of Figures

2.1	Vortex GPU Architecture Overview [89]: The GPU is organized into several clusters. Each cluster has a set of cores that it supports with a number of threads. Relevant configuration parameters are highlighted in red.	7
3.1	A simple illustration of the model. Two users submit two workloads to two different clusters. Each cluster comprises a set of PEs.	23
3.2	System Overview: Three configurations are characterized and partitioned. CFG1 has 4 clusters and the partitions are illustrated as PAR CFG1. A configuration is then chosen based on MNW and is sent to the scheduler to enforce.	33
4.1	Overview of the end to end Vortex system: Path shows the different steps from application to GPU.	46
4.2	Overview of the DE10 SoC Layout for a sample 2 cluster configuration . . .	47
5.1	Expected F values measured for various benchmarks	57
5.2	Execution time measured for benchmarks running in isolation and deviation from the prediction of the model. The expected F value measures are accurate and can predict speedup to within a 5% error margin.	58

5.3	Prices for clusters with the sample workload and (ϵ) for each iteration. Prices converge rapidly — at 15 iterations the error is within 10^{-4} . Clusters are priced based on their capability and relative demand. Cluster 1 and Cluster 2 that have equal capability and have the same relative demand, so are priced equally.	60
5.4	User Set 1 speedup and allocation metrics. In CFG1 and CFG2, User 1 receives shares on the higher thread count clusters and is excluded from the FPU cores and User 2 and User 3 are allocated shares on Cluster 2 and Cluster 3. In all configurations, User 2 receives a higher proportion due to its higher weight.	61
5.5	User Set 2 speedup and allocation metrics. In CFG1 and CFG2, User 2 and User 3 compete for high thread count clusters, Cluster 3 and Cluster 4. User 1 therefore receives some time on FPU clusters. In CFG3, Cluster 1 and Cluster 2 do not have FPUs. User 1 strategically bids higher for Cluster 1 and Cluster 2 permitting it to run on a larger thread count without competing with User 2 and User 3 on Cluster 3 and Cluster 4.	63
5.6	User Set 3 speedup metrics. Since the users are running the same workload, in all configurations, the speedup is distributed strictly proportional to the user’s weights.	64
5.7	Utilization for User Sets across different configurations	65
5.8	Speedup with Entitlements across different configurations. Ratio to Fisher Market speedup is shown in red. In all cases, users receive speedups below Fisher Market speedup.	67
5.9	Speedup with Weighted Equalized Speedup across different configurations.	68
5.10	Utilization for User Sets across different configurations	69

List of Tables

4.1	Performance Counters to estimate F	44
4.2	Register mapping of Vortex Control Signals	50
5.1	Sample configuration. Cluster 1 and Cluster 2 have 2 cores with floating point units (FPUs). Cluster 3 and Cluster 4 have no FPU, but have larger sizes with 4 and 8 cores respectively.	59
5.2	Sample Users. User 1 runs Resnet on all clusters. User 2 runs k-Means on the floating point clusters and AlexNet on the clusters. User 3 runs MLP on the floating point clusters and YoloNet on the remaining clusters.	59
5.3	Configurations used. All configurations have 4 clusters, and each comma separated value indicates the value corresponding to the cluster.	60
5.4	User Set 1. All users run the same workload on all clusters; User 1 runs Resnet, User 2 runs k-Means and User 3 runs MLP.	61
5.5	User Set 2. User 1 runs Resnet jobs on all clusters. User 2 runs k-Means on Cluster 1 and Cluster 2 with FPU cores, and runs AlexNet on the remaining clusters. User 3 runs MLP on Cluster 1 and Cluster 2 also using FPU cores, and runs YoloNet on the remaining clusters.	62
5.6	User Set 3. All users run the same Resnet workload on all clusters.	63

Chapter 1

Introduction

To meet the performance and energy efficiency demands of modern compute workloads, specialized hardware accelerators are adopted in modern servers and Systems-on-Chip (SoC). These accelerators can be deployed as Field Programmable Gate Array (FPGA) overlays or Application Specific Integrated Circuit (ASICs). For instance, the popular Convolutional Neural Network (CNN) or Large Language Models (LLMs) in machine learning can be mapped to systolic arrays (TPU) [47] or adapted to run on general purpose Graphics Processing Units (GPGPU) [18, 27]. We find analogous developments in analytics [39, 109, 55], database systems [33, 86] and network acceleration [14, 111, 5, 67, 42, 68].

Such hardware accelerators, while promising, suffer from underutilization; underutilization reports range from 20% - 90% [99, 83, 44]. In [44], Jeon et al. found up to 90% underutilization and an average of 46%. The community has explored various solutions to improve utilization [101, 102, 99] that can be, broadly, classified into one of two methods — aggressive additional parallelizing of applications or, sharing hardware resource with multiple users [17, 83, 72, 62, 106]. Parallelizing an application increases the number of hardware units that can be concurrently utilized and, consequently, improves hardware utilization [83, 62, 106]. Models such as Amdahl’s [38] law (and its variants [38, 32, 73]) predict diminishing improvements in speedup from increased parallelism. Amdahl’s law predicts that with sufficient parallelism, a workload is limited by its serial portion.

Sharing hardware resources with multi-users has been employed to improve utilization [105, 51, 99, 17, 72]; different users' workloads are independent, and they can consume hardware resources concurrently. Sharing is not constrained by Amdahl's law, as different users' workloads are inherently independent. When users simultaneously use hardware (referred to as multi-tenancy on hardware), sharing must minimize the detriment of performance of a single user, often called a Quality-of-Service (QoS) guarantee.

In shared systems, QoS guarantee can be evaluated with respect to entitlements, the proportional share of resource per user. If each user receives at least the same utility from the shared system as they would from their entitlement, the QoS guarantee is met [36, 50, 31]. Redistributing underutilized resources while meeting the QoS guarantee has been studied theoretically [1, 28] and in deployed systems [30, 31, 104].

When hardware resource are homogeneous (all hardware units are the same) the challenges are fewer. Preferences of each user are symmetric for all hardware units, and shares on one hardware unit can be substituted for another. Distributing resources equally among users is often sufficient to adhere to QoS requirements of users [28]. In some cases, it may be desirable to have some unequal allocation to improve utilization. For example, consider allocating CPU resources on a server with 2 identical CPUs and 4 users in a 10-second window. Allocating each user 5 seconds on one CPU is sufficient to guarantee their entitlement and QoS requirement. To improve utilization and speedup, the system can consider the parallelism of the users' workloads. Assume two users (User 1 and User 2) have highly parallel workloads and two users (User 3 and 4) have fully serial workloads. The system gives 2.5 seconds on 2 CPUs to the highly parallel workloads, and 5 seconds on 1 CPU to the serial workloads.

In a heterogeneous setting, these challenges are pronounced; users have asymmetric demand on different resources. In such systems, equal distributions may deviate considerably from QoS requirements [28]. In the previous example, assume that one CPU has special hardware units that speedup up floating point instructions by 2x compared to the other. If User 1-4 have no use of floating point, equal allocation meets the QoS guarantee. If, however, User 1-2 are entirely floating point, allocating these users to the non-floating point CPU misses the QoS guarantee by 25% and the floating point hardware is not utilized.

In contrast to a homogenous setting, to substitute a resource in a heterogeneous setting, a mechanism needs to account for relative utility between resources per user. The system needs to know if specialized hardware is available, the proportion of the workload that can use the hardware, and the improvement in performance that the specialized hardware provides across all users [103].

To study sharing in multi-user systems, recent work has used game theory [30, 29, 31, 104, 81]. In game theory, availability of resources, their relative utility, and user preferences can be formally defined. Atop these definitions, methods and techniques in game theory explore ways of allocating resources and their merits. In particular, the QoS guarantees can be formulated as fairness notions among users. Compared to other methods, formal guarantees on QoS (as mathematical proofs) can be derived with game theory; it can formally guarantee that users will not be better off with any other allocation (that there are “no justified complaints” [20]). It allowed for the formulation and study of systems with multiple resources [30, 1, 29], substitutable resources [103], dynamic allocations [29, 28] in the presence of adversarial users [75].

Notably, to allocate CPU cores in the datacenter setting, Zahedi et al. [104] presented algorithms based on market and trading mechanisms. They present a performance model to characterize the utility of a server to users, and presented a Fisher Market mechanism to allocate resource to agents. This, however, is limited to CPUs on servers and specialized hardware are unaccounted for in this treatment.

Other sharing methods can be seen in Operating Systems (OS). OS kernels enforce fairness principals on CPU time. For instance, Linux kernel implements the completely fair scheduler (CFS) [70] for fair sharing of CPUs. CPU sharing schemes such First-Come-First-Serve (FCFS), stride scheduling and lottery scheduler have been proposed and studied [91, 92]. Recently, runtime sharing of specialized hardware has been discussed and studied [72, 81, 99]. Application of queuing theory and game-theoretic principals have described desirable metrics and algorithms to achieve such methods [81, 30, 65]. Such work focuses on several aspects, ranging from load balancing to fairness measures. Queuing and scheduling work however limits its discussion to the runtime behavior of such systems.

In this work, we focus on the specific challenges with sharing and, investigate unexplored opportunities for improved performance and utilization. In particular, we ask and address two questions;

- for a set of users, how must hardware resources be allocated, and
- what hardware configuration is most suitable for the set of users.

The first question involves allocating hardware resources to different users to achieve fairness. The allocation impacts the runtime (speedup) of each user, and we investigate fair allocations of hardware accelerators and present algorithms to achieve them. We employ well studied market mechanisms where the market sets prices on hardware resource and users bid on them.

The second studies the choice of the architecture based on fairness. While allocations maybe fair on a given hardware architecture, we additionally consider the choice of architecture and its impact on fairness. Particularly, in heterogeneous systems, the special function offered by different hardware units, and the number of each hardware units, among other factors, can change the fairness of the final allocation. We present notions of fairness to choose ideal configuration candidates.

The key contributions of this work are the following.

- We develop a mechanism of (1) allocating resources and, (2) configuring hardware, and provide a game-theoretic analysis of its fairness properties.
- We present an implementation of the mechanism on the configurable Vortex GPU [89] using TVM [15] and OpenCL [66] to design machine learning workloads.
- We construct three sets of multiple users with (1) one workload per user, (2) multiple workloads per user and (3) same workload across all users, and evaluate the performance and utilization of each set with different hardware configurations.
- We study the mechanism with different hardware configuration by varying the presence of floating point units, different frequencies (80-120 MHz) and number of execution units.

Chapter 2

Background and Related Works

In this chapter, we provide the background of this work and related work. We discuss hardware accelerators, GPU architectures, hardware performance models, specifically Amdahl's Law, utility theory, fairness measures and mechanisms.

2.1 Hardware Accelerators

The last two decades has seen diminishing improvements in single-threaded performance of CPUs and has been noted as a deviation from Moore's law [22, 24, 63]. Concurrently, computation-hungry data-parallel algorithms able to scale to multiple threads to keep Moore's Law alive [22]. For example, CNNs adopted in applications ranging from computer vision [59, 98], self-driving [69, 16] and weather prediction [108] can scale to 100s of cores. In the absence of continued scaling predicted by Moore's law, accommodating the performance and energy efficiency needs of such application required hardware accelerators [97]. GPGPUs and FPGAs have been widely deployed in data centers and cloud offerings [47, 17, 18]. Even custom ASICs like Tensor Processing Unit (TPUs) have been introduced. The development of open source instruction sets like RISC-V [95] have made development and interoperability of hardware accelerators with software easier. Hardware

accelerators vary widely in capabilities, but they often comprise an array of computational blocks running in parallel. Each block comprises one or more arithmetic units, registers and memory. The blocks are stitched together with interconnection logic, along with any system level entities (caches, arbiters, Direct-Memory-Access (DMA) engines etc.).

GPGPUs are a specific class of accelerators that comprises an array of several multi-processors [17]. Each multi-processor comprises numerous parallel arithmetic units and registers executing instructions in lockstep, leading to a data-parallel computational paradigm referred to as Single-Instruction-Multiple-Threads. In this work, we look at Vortex GPU [89], a state-of-the-art configurable RISC-V GPU as an overlay on an FPGA.

2.1.1 Vortex GPU

Vortex GPU [89, 88] is an open-source highly configurable GPU architecture developed for research needs. Vortex uses the RISC-V instructions set architecture with modification for GPU control logic. Vortex GPU is organized as several independent cores grouped together as independent clusters. The clusters have a shared L3 cache, which caches data from the system memory. Cores in each cluster are configurable with a shared L2 cache. A core can execute several threads concurrently with a shared instruction and data cache. Figure 2.1 provides a high-level overview of the organization of the architecture.

The Vortex GPU uses a warp based single-instruction-multi-thread (SIMT) architecture. We summarize the salient features of the reference implementation of the architecture provided for Intel Stratix 10 FPGA (Stratix 10) [57].

Organization and Terminology

The Vortex GPU is organized into clusters, cores, threads and warps.

- **Warps.** Vortex warps are similar to thread blocks in other GPU architectures. A warp represents a collection of logical threads that execute the same instruction.

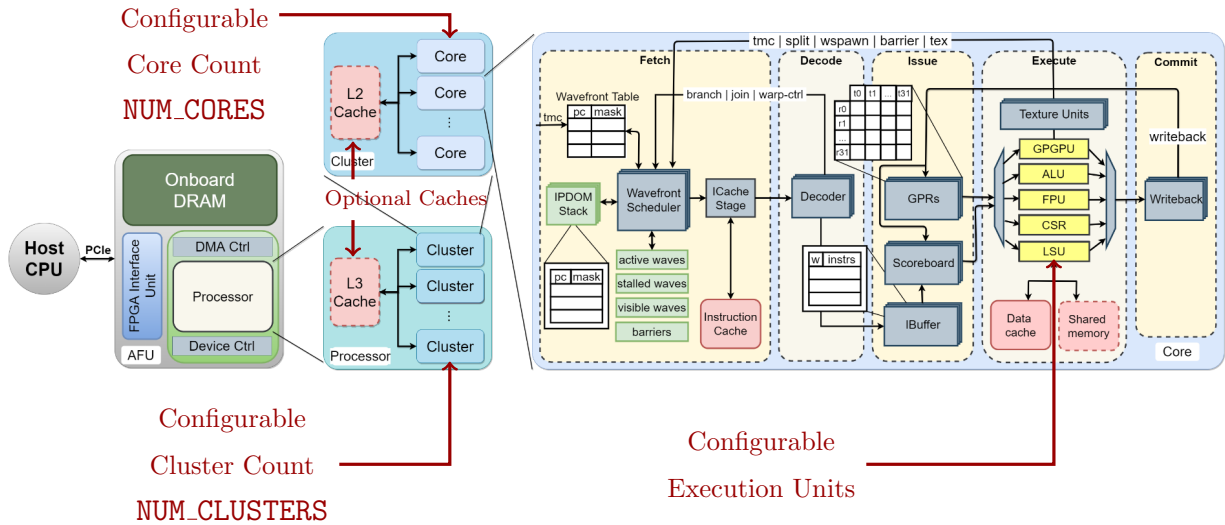


Figure 2.1: Vortex GPU Architecture Overview [89]: The GPU is organized into several clusters. Each cluster has a set of cores that it supports with a number of threads. Relevant configuration parameters are highlighted in red.

Vortex can be configured to track a fixed number of warps during synthesis. The primary effect of the number of warps is on the number of simultaneous contexts that the GPU can keep track.

- Threads.** Threads in Vortex are physical entities within a GPU core that simultaneously execute instructions in lock step. All threads in a core execute the same instruction, but each thread runs in its own thread context. Increasing the number of threads in a core increases the number of execution units and register files to manage context per core. Each thread in a core has one execution unit of each type such as an arithmetic and logic unit (ALU), floating point unit (FPU) etc. and one register file. Each thread in a core is assigned a unique number.
- Cores.** Vortex cores are similar to independent CPU cores, each with a collection of execution units and register files for threads, instruction fetch and control units, memory write back units and, dedicated L1 caches for instruction and data. Each core is numbered uniquely across the GPU.

- **Clusters.** A cluster is a collection of cores with a shared L2 cache. It must be noted that clusters do not affect the logical numbering of cores. Clusters only affect the organization of the cores and cache. This allows for instructions compiled for a fixed core GPU cluster (for instance 32) to be used across different cluster core configurations (8 clusters of 4 cores or 4 clusters of 8 cores).

Execution Units

Vortex cores are configured with 32-bit integer arithmetic units (ALUs) and are configurable with several capabilities. Optionally, 64-bit units can be instantiated. Importantly, Vortex cores can be configured to instantiate 32-bit floating point units (FPUs). Vortex FPUs are designed to use the Digital Signal Processing (DSP) blocks available on the FPGA for performance [37]. Experiments described in Chapter 4 employ these to make cores less and more capable. The Vortex system also consists of “texture units” targeted for graphics workloads [88].

Clock Domains

In hardware architectures, it is common for different units to run with different clocks, and all hardware block running with a given clock is said to belong to the same clock domain [19]. Multiple clock domains allow for a reduction in congestion for clock routing and allows for faster regions of a chip to run at higher clock rate. In the Vortex GPU, all clusters, core and threads in the system run on the same clock. At the chip level there are three major clocks — system memory clock, HPS-FPGA clock, and Vortex top level clock.

Core Numbering

In the Vortex GPU, cores are uniquely numbered with a `COREID`. The numbering is available to the application to read at runtime through a RISC-V Control and Status Register (CSR) [95, 94] named `CSR_COREID`. In addition, `CSR_NCORES` advertises the number of cores available. These values are fixed when the GPU is synthesized and does not change.

Intel OPAE

The top level implementation of Vortex on Stratix 10 uses the Intel Open Programmable Acceleration Engine (OPAE) [40]. OPAE provides developers with interfaces to perform common tasks such as data movement, writing control signal and communicating to and from the hardware block. In compliance with OPAE, Vortex implements an OPAE Accelerator Function Unit (AFU) wrapper around the Vortex GPU for control and data transfer. The system allows for communication using the PCI-e capabilities [12] of the underlying FPGA development board. OPAE is however noted to have several restrictions, particularly without an Intel Xeon CPU [40].

2.2 Sharing

It has been noted that custom hardware acceleration suffer from underutilization [101, 102, 99]. Sharing hardware resources is one solution to mitigate underutilization [17, 72, 99]. With multiple users, each user has an independent workload that can be run concurrently to keep hardware busy. A corollary of sharing, however, is that a given user may experience diminished performance. Increased utilization from sharing must therefore be in tandem with an analysis of QoS of each user. Prior work in sharing has explored the idea of providing guarantees on QoS, by doing resource allocation and scheduling [30, 31, 92, 104].

To formally study the outcomes of sharing, we can game theory methods [104, 26, 30, 1]. Game theory can be employed (1) to formalize users and resources, (2) to study desirable properties, and (3) to formulate ways of allocating resource allocation appropriate for the setting. In general, users have different preference of resources, and the resources are limited.

Utility Function

In game theory, such preferences can be quantified using a utility function. Assume a set of resource R are to be allocated. We can model the utility of the user i with a function

u_i such that if the user prefers resource a to b , then $u_i(a) > u_i(b)$. Such utility functions have been shown to exist (von Neumann-Morgenstern Utility Theorem [90]).

$$u_i : R \rightarrow \mathbb{R}$$

Mechanisms

Once a model of utility is formulated, algorithms are used to allocate the resource to different users. Various measures and optimally targets of such algorithms have been studied [26, 1, 73, 30]. In game theory and economic market theory, such algorithms are resource allocation mechanism (often referred to as mechanisms). Each user that derives a certain utility from a resource is said to have demand on that resources. A mechanism then uses such demand to allocate available resources.

In this work, we model utility of users based on the performance of the users' workloads. There are several prior performance models that quantify improvements in performance when consuming hardware resources [38, 6, 32, 73]. We begin by evaluating Amdahl's law and its applicability to the hardware accelerators. To allocate resource, this work uses a market-based mechanism where hardware resources are assigned prices and users bid on the resources.

2.2.1 Amdahl's Law

Amdahl's law [6] characterizes the speedup of a task when hardware improvements decrease the turnaround time of a fraction of instructions in a workload. In particular, Amdahl's law has been used to characterize the speedup when a task is being run on a parallel machine with respect to a serial machine.

Amdahl defines speed up as the ratio of old and new execution times. If the old execution time is T_{old} and the new execution time T_{new} , then the speedup is the following.

$$\text{Speedup} = \frac{T_{old}}{T_{new}}$$

Assume a task currently being run on a single-core machine is run on P such parallel cores. Amdahl’s law claims the following; if F fraction of the original task can be parallelized, the execution time of that portion will improve by a factor of P . However, the remaining $1 - F$ portion (referred to as the serial portion) remains at the same execution time. This behavior can be quantified to a speedup value.

$$s = \frac{1}{1 - F + \frac{F}{P}}$$

or equivalently,

$$s = \frac{P}{P(1 - F) + F}$$

An implication of Amdahl’s law is that increasing the number of hardware units have diminishing returns. When a task is being parallelized, every additional core provides a smaller speedup improvement than the previous core. If the task is infinitely parallelized, the task is still bottlenecked by the fraction of serial portion.

$$s_{\infty} = \frac{1}{1 - F}$$

Several shortcomings of this model have been identified [38, 87, 32], two of which are the subject of this work.

1. First, Amdahl’s parallelization law does not account for heterogeneous cores and special capabilities. In particular, it does not capture the trade-off between larger, more capable cores and smaller, less capable cores. For instance, parallelizing a floating point numeric job on a system with one core with a floating point unit (FPU) and three integer only cores will provide a speedup distinct from that of one with 4 integer only cores.

2. Second, Amdahl’s law assumes that workloads do not scale, which may not hold in practice. Increasing the number of cores is accompanied by an increase in the size of workloads. We look at two specific amendments to law borne from these shortcomings.

Amdahl’s Law in the Multicore Era

Hill et al. [38] investigated the speedup of multiple core architectures with certain ceilings. First, they defined a base core equivalent (BCE), at most n of which can be fit on a chip due to any number of constraints. They then define a trade of — the resource of r BCEs can be traded off to create a larger, specialized core with $perf(r)$ serial performance that consumes r resources and a larger area. For instance, if the resource of two integer cores can be combined to create a floating point core with 1.5 times the serial performance, then $perf(2) = 1.5$. They note that if $perf(r) > r$, then the bigger core must be used since the performance will be strictly higher, and if, $perf(r) < r$ then the recommendation depends on the parallel portion of execution. They consider cases where all the cores are the same (symmetric case) and where one core is larger (asymmetric case).

The respective speedups can be expressed in the following manner.

$$s_{symmetric} = \frac{1}{\frac{1-f}{perf(r)} + \frac{f \cdot r}{perf(r) \cdot n}}$$

$$s_{asymmetric} = \frac{1}{\frac{1-f}{perf(r)} + \frac{f}{perf(r) + n - r}}$$

Gustafson’s Law

Gustafson et al. [32] noted that an increase in compute power inspires in the user an increase in workload (problem) size. By way of example, we can examine this in a modern image classification problem involving convolutional neural network (CNN) inference.

Convolution neural networks have been used to classify images. Assume a user running such an image classification workload moves from a single server system to two server systems. Speedup can be measure using Amdahl’s law by evaluating the instructions in the CNN workload that be parallelized to two machines. However, in such a scenario, the user may elect to classify more images instead. The user instead of speeding up one image classification task will rather run two image classification.

In [32], they notice that for such workloads, the serial portions of the execution remain the same and that the parallel portions scale linearly. They introduce the notion of scaled speedup and define it.

Scaled speedup,

$$s = P + (1 - P) \cdot (1 - f)$$

Note: for consistency the notation has been changed.

2.2.2 Fisher Markets

Market mechanisms model trading of resources by users. In the simplest case, users (participants in the market) begin with certain resources called entitlements. Users then trade to achieve a higher utility. When no users prefer to trade, the market is said to be in an equilibrium. In this work, we focus particularly on Fisher markets [107, 79].

In a Fisher market,

- there are n buyers for a set of m resources,
- each buyer i starts off with an entitlement e_i and has a utility function u_i ,
- each resource j has a price p_j at some t , and
- user i receives a share x_{ij} for this resource for the amount $b_{ij} = x_{ij} \cdot p_j$.

In Fisher markets, utilities are a function of the share of resources allocated to the user [79]. A market equilibrium in Fisher markets occurs when each buyer maximizes her utility for the price of resources, and the resources are not in deficit or surplus (i.e. all resources have been allocated). The prices at market equilibrium are called market clearing prices.

Proportional Response Dynamic

To compute the market equilibrium of Fisher Markets, Proportional Response Dynamic (PRD) has been proposed [107]. PRD is an iterative method of computing the prices at market equilibrium. At each iteration, prices of resources are computed based on the bids of users. In response to the new prices, each user updates her bid to maximize her utility. The process continues until the price converges. Fisher markets have been studied with several classes of utility functions, in particular linear utilities and utilities with constant elasticity of substitution (CES) [64, 79].

For CES utilities, it has been shown that a fixed point of this procedure (i.e $\mathbf{p} = \text{next}(\mathbf{p})$) is a market equilibrium and that the procedure converges [107].

Amdahl's Bidding

In [104], Zahedi et al. presented a Fisher Market to allocate data center cores using Amdahl's law. They present Amdahl's utility, a non CES utility that each user derives from a server core based on Amdahl's law. They present Amdahl's Bidding (AB), a PRD procedure for users with this utility function, and show that such a procedure has desirable properties.

2.2.3 Collective Utility Functions

In a multi-user setting, for a certain allocation of resources, different users will have different utilities. To compare one resource allocation to another, a collective utility function (CUF)

is used [82]. A CUF, sometimes referred to as welfare, is a function that aggregates users' utility to a real number. For n users, a CUF is a function $W : \mathbb{R}^n \rightarrow \mathbb{R}$. A higher value of the W is said to have a higher collective utility or higher welfare. We examine three CUFs.

Maximum Social Welfare

The utilitarian CUF, also known as social welfare, is the sum of utilities of all users. In social welfare, a resource allocation R is preferred to R' if R increases the utility of any user.

$$\text{CUF}_s(R) = \sum_i u_i(R)$$

The maximum social welfare (MSW) is the maximum across all resource allocation of social welfare.

$$\text{MSW} = \max_R \text{CUF}_s(R)$$

Max-Min Share

A short-coming of social welfare is that it does not account for how the utilities are distributed. For instances, for two users, utilities 100,10 has a higher social welfare than 50,50. In contrast, we have the egalitarian CUF.

The egalitarian CUF, also known as min-share, is the minimum of utility among the users. Here, a resource allocation R is preferred to R' only if the lowest utility is higher (50,50 is preferred to 100,10).

$$\text{CUF}_m(R) = \min_i u_i(R)$$

Max-min utility (MMU) is maximum possible minimum utility.

$$\text{MMU} = \max_R \text{CUF}_m(R)$$

Maximum Nash Welfare

Consider three resource distributions that give the utility profiles (100, 10), (50, 50) and (65, 43) respectively. The profile 100, 10 has the highest social welfare and 50, 50 has the highest egalitarian welfare. We can see that the profile (63, 43) offers a trade-off between the other two utilities where there is a low, but non-zero difference in utilities and a higher, but not the highest welfare. Nash CUF is suitable to account for such trade-offs [11].

The Nash CUF, also known as Nash Welfare, is the product of utilities of all user.

$$\text{CUF}_N(R) = \prod u_i(R)$$

The maximum Nash welfare (MNW) is the maximum across all resource allocation of Nash welfare.

$$\text{MNW} = \max_R \text{CUF}_N(R)$$

Chapter 3

Utility Model and Mechanism

To distribute hardware resources among users, we require a resource allocation mechanism. Several categories of mechanisms have been studied extensively in various settings with diverse objectives [104, 28, 30, 29, 31]. Such mechanisms account for available resources, demand on the resources, usefulness (utility) of the resources to the users and the desirable metrics to enforce.

For instance, consider Shortest-Job-First (SJF) [8] an OS job scheduler. In game theory, SJF can be viewed a mechanism that distributes CPU time. SJF mechanism allocates CPU time to different tasks running on an OS kernel, accounts for the run/wait time of each task, models utility of task inversely to its wait time (i.e. in SJF, the shorter the wait time of a task the better the task is) and, SJF achieves optimal wait times [8].

While mechanisms with various attributes have been studied, the emphasis in this presentation is of mechanisms that enforce fairness notions. To design and study such mechanisms require the following.

- **A utility model.** To enforce metrics (such as fairness notions), a mechanism must be able to evaluate the utility of a resource allocation to participating users.

- **Metrics of fairness.** One or more fairness notions may be desirable for a given setting; these must be carefully considered when designing the mechanism.

This chapter describes a proposal of a utility model, a discussion of the desirable metrics of fairness and a proposal of a mechanism.

The key presentations in this chapter are the following.

- Methods of evaluating the utility of a hardware configuration to a user.
- A mechanism to partition hardware and aggregate preference across multiple users on a hardware configuration.
- An evaluation of the model and proposed mechanism.

3.1 Motivation

Modelling and mechanism design involves several considerations listed below.

PE as allocation units. A hardware resource can be modelled as a collection of processing elements (PEs) that perform computations. A processing element can perform a set of operations and instructions can be issued to run the operations.

For instance, in addition to LUTs and memory units, modern FPGAs comprises DSP blocks. Designs often have PEs that use one or more DSP blocks capable of independently evaluating instructions [37]. The instructions maybe explicit, as with soft CPU cores [48], or maybe implicit, often by way of transport mechanisms [56, 96]. Similarly, AI engines comprise vector blocks, collections of multipliers and other hardware that can be independently addressed [18, 80, 2].

This in our proposed utility model, a PE is the unit of allocation; each user is assigned a PE for a certain time, and receives utility by consuming the PE. This allocation is the fraction of time for which the user can consume the PE (for some fixed window of time).

Bandwidth model of PEs. On the same hardware, PEs with different capabilities may exist; often PEs have specialized arithmetic units that perform specific tasks. For instance, some PEs may have floating point units (FPUs) while other PEs may have high precision integer units. In addition to the special capabilities of the PEs, they may also differ in the physical frequency ranges with which they can run. One PE maybe able to run at 3 GHz, while another can run at 3.2 GHz. To allocate PEs, we require a model to compare the relative performances of PEs.

To accomplish the comparison, we use a throughput performance model of PEs. For a given user, the model reports the throughput performance estimates of the PEs with appropriate modifications to Amdahl’s law, a well established heuristic [6, 38, 4, 87]. Similar to Amdahl’s law, we use the parallelizable fraction of a user’s workload to estimate speedup of the user workload.

Allocating resources on a configuration. While we focus on fairness of configuration and leave the allocation and scheduling of resources on a configuration to a runtime scheduling entity, configuration choices must be informed by the runtime scheduling algorithm.

To illustrate this constraint, consider a scenario with one user running a highly parallel throughput oriented job. With a single user, the desirable configuration is one that maximizes the throughput of the job. Consider the following cases.

1. In one case, a perfect load balanced scheduler [65] is run (for simplicity, assume load here is the number of tasks run of the job). A load balanced scheduler attempts to even the load among all PEs, and consequently the throughput provided by this configuration is that of the minimum throughput among the PEs.
2. In another case, a perfectly greedy scheduler is run. A greedy scheduler will run tasks as soon as a PE is vacant. The throughput of this configuration is that of the average performance of the PEs.

Since multiple users are being simultaneously seated on the hardware, accurate estimate of utilities require estimating the impact of runtime sharing of the hardware. In particular, the share of the PEs that each agent receives needs to be computed. While certain predictions can be made of the behavior of runtime schedulers, poor runtime schedule may inhibit many possible configurations, and sensitivity to predictions of the scheduler may lead to poor configuration choices.

To overcome this, in addition to the configuration mechanism, the system a) provides a partitioning mechanism that partitions the hardware among users and, b) requires that conforming schedulers enforce this partition.

3.2 Illustration and Overview

In this section, we provide an illustration to summarize the key details in this chapter.

Utility and Resource Model. To evaluate fairness, we model hardware configurations as a collection of PEs grouped together as a cluster. Each user has an independent workload to run on a cluster. **A key insight of this work is that a configuration can be modelled as a public good [11] for which users have private valuations.** Once the workload is submitted, we compute the performance of the workload on that cluster, using our performance model presented in Section 3.3.

Speedup in particular has been shown to be a favorable utility for users running throughput-driven programs, and several models of speedup and trends have been studied for distributed systems [25, 84] and computer architecture [6, 38, 4, 87]. We make the following observations.

1. Each PE can compute a set of functions (operations) and has an operating frequency. If a PE A has a higher frequency and can provide the same set of functions compared to PE B, the user will experience a higher speedup by using PE A. Similarly, if some function can be computed faster (fewer cycles) by PE A, the user will experience a higher speedup by using PE A.

2. The number of PEs in a cluster allows for parallelism. Parallelism increases speedup and has been modelled by Amdahl’s law. Using the parallelize fraction of user’s workload F , we can compute the speedup for a given set of PEs. We modify Amdahl’s law to account for the PEs and their specialty to compute speedup.

In Section 3.3.2 we incorporate (1) into our model by introducing an invariant called apparent throughput. If PE A runs at 100 MHz and executes 2 instructions per cycle and PE B runs at 200 MHz and executes 1 instruction per cycle, to the user they both provide a throughput of 200 instructions per second. Similarly, an instance of PE A and an instance of PE B combined provides additive throughput.

In Section 3.3.4 and 3.3.5 we look at the parallelism in (2) and modify well established heuristics, Amdahl’s Law [6] and Gustafson’s Law [32].

User Weights. In a system, users have extrinsic weights associated with them. The weights may come from the administrator of the system, requiring certain users to be preferred over others, and may even reflect an order of priority [36]. It is important in discussion of fairness for such weights to be considered.

Market Mechanism. For a set of users and their weights, we allocate time-shares on each cluster. Each cluster has some specialized PE(s), a fixed number of PEs and an operating frequency. As we examined above, a user’s preference (speedup) on the cluster is based on (a) the use or specialty of the PE to the user’s workload and (b) the parallel fraction of the users’ workload. If a user has a highly parallel workload, that user may have a demand for a high number of PEs but few specialty PEs. Similarly, a user with specialized functions (floating point arithmetic for instance), but lowly parallel, may have a demand for specialized PEs (FPUs) but fewer PEs.

To quantify these relative preferences and allocate resources, we use a market mechanism called the Fisher market [79] (see Section 3.4. In a market mechanism, the PEs are allocated prices and users are given a budget based on their weights (entitlements). Given these prices, each user bids a portion of their budget to acquire time-shares on each PE. Once the bids have been placed, the market then computes the allocation for each user,

and based on the demand new prices are computed. This procedure is repeated until the market reaches an equilibrium; all PEs are fully allocated and prices no longer change.

Fairness Notions. It is desirable for any allocation to be fair (meets performance requirements for an individual user) and be efficient (some notion of performance of all users). In this work, we look at the following measures of fairness.

- **Pareto Efficiency (PE).** An allocation is Pareto Efficient if a user cannot get a higher speedup from a different allocation without at least one other user experiencing a slow-down. For some allocation A if a different allocation B can provide a higher speedup to one user and at least the same speedup as A to the other users, then desirably the market mechanism should not converge to A. We prove this to be true.
- **Sharing Incentive (SI).** A market provides Sharing Incentive if a user will receive at least the same utility from participating in the market as their entitlement. If the market does not have Sharing Incentive, it will be outperformed by a simple weighted share. We prove that our market provides Sharing Incentive.

The following sections provide formal details of the model and mechanism.

3.3 Throughput Model

The system is modelled as a collection of configurable independent PEs organized as clusters. A processing element is a unit that can perform a set of operations, and instructions can be issued to run the operations. In this model, PEs are configurable instruction processors with arithmetic units. The model characterizes each PE based on its capability and its frequency. In addition, the number of such PEs that are present also impact performance.

In general, this model applies to any hardware configuration which can be broken into constituent PEs. This may be an FPGA overlay or an ASIC configuration to be taped-out.

We motivate preliminaries and definitions used in the model in the next two sections. Section 3.3.3 formally defines the model.

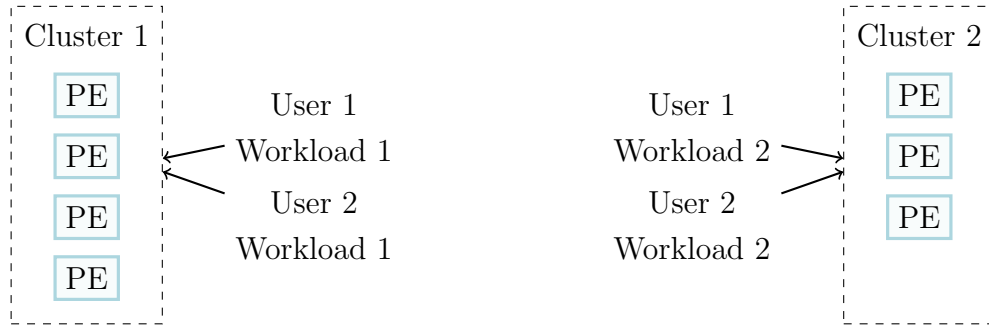


Figure 3.1: A simple illustration of the model. Two users submit two workloads to two different clusters. Each cluster comprises a set of PEs.

3.3.1 PE Definition

Intrinsic (Clock) Frequency. Each PE has an associated clock domain and clock frequency that it operates on. This is modelled with a positive real value.

$$f \in \mathbb{R}^+$$

f is a characterization of the clock frequency of the PE.

Retiring rate. PEs may process functions with varying latency. For a given workload (set of instructions) we can compute the retiring rate; the average rate at which functions (operations) of the workload are computed. The retiring rate is a measure of specialty of the PE, as noted above.

Utility requirement from the PE model

If the utility is to reflect the frequency performance of the system, such a treatment, all else equal, needs to satisfy the following.

1. PE with higher frequency must provide a higher utility. Let PE A and PE A' be instances of design D running at frequency f_A and $f_{A'}$ respectively. If $f_A < f_{A'}$ then $u_i(A) < u_i(A')$.

For instance, considers PEs A and B that evaluate some arbitrary 32 bit function c . Let A run with frequency 200Mhz while B runs at 100Mhz. All else equal, a user must receive a higher utility from utilizing A.

2. PE with higher turnaround time of operation must provide a lower utility. Let PE A and PE A' be hardware design computing an arbitrary c at the same frequency. Assume A and A' computes c in t_A and $t_{A'}$ cycles respectively. If $t_{A'} < t_A$ Then $u_i(A) < u_i(A')$.

For instance, assume PE A computes f in 20 cycles and B computes in 10 cycles. All else equal, a user receives a higher utility from utilizing B.

3. Utility from two PEs must be greater than or equal to the utility from only one PE. Assume utility to user i from PE A is $u_i(A)$ and PE A' and $u_i(A')$. Then $u_i((A, A')) \geq \max\{u_i(A), u_i(A')\}$. Desirably, $u_i((A, A')) = u_i(A) + u_i(A') - \epsilon$.

For instance, assume we have as system with PE A and PE B. If a user is assigned both A and B, the user should not be worse off than just having A or B (they should be able to ignore one without penalty).

Cluster and Configuration

A cluster is a collection of instances of PEs. Each PE in a cluster is numbered, and a cluster is considered unique. Similarly, a configuration is a collection of clusters, and the clusters are also numbered. The distinction between a cluster and configuration is in their treatment by the speedup model used and is explored in Sections 3.3.4 and 3.3.5.

3.3.2 Throughput Invariant

The specialty of a PE can be quantified as a product of its operating frequency and retiring rate. For instance, a PE A running at 200 MHz retiring 1 instruction every cycle and PE

A' running at 100 MHz retiring 2 instructions every cycle deliver identical throughput.

To account for such equivalences, we introduce an invariant — the apparent throughput. The apparent throughput is an invariant between two settings that deliver identical make spans to a workload. In the setting above, both PE A and PE A' have that same apparent throughput of 200.

In particular, two types of apparent throughput are of importance; one permitting comparison between PEs, and one permitting comparison of a collection of PEs.

- **Apparent Throughput of a PE.** Apparent Throughput is the product of the frequency of a PE and its retiring rate. Two PEs with the same apparent throughput provide identical utility to a user.
- **Apparent Parallel Throughput of PEs.** A collection of PEs have an Apparent Parallel Throughput that is the sum of their Apparent Throughput. To motivate, assume a highly parallel job. For such a job, every PE can simultaneously run instructions and each instruction is serviced at the rate of the apparent throughput of the executing PE; apparent rate of execution is therefore the sum.

3.3.3 Formal Definitions

The system is assumed to have $n \in \mathbb{Z}^+$ users. We can uniquely map users to $[n] = \{1, \dots, n\}$.

Definition 3.3.1. *The set $\mathbf{Users} = [n]$ is the set of users in the system.*

Definition 3.3.2. *A **PE** is the 2-tuple (f, r) where $f \in \mathbb{R}^+$, and $r : \mathbf{Users} \rightarrow \mathbb{R}_0^+$. f is the clock frequency of the PE and r is the number of instructions retired per second (instructions per cycle) by the PE for a given user. \mathbf{PEs} is the set of all PE.*

A configuration is a collection of unique PEs with various capabilities. To define a configuration, we map each PE to a unique integer.

Definition 3.3.3. A cluster c is the function $c : [m_1] \rightarrow \text{PEs}$ for some $m_1 \in \mathbb{Z}^+$. A configuration is a function $C : [m_2] \rightarrow \text{CLUSTERS}$ for some $m_2 \in \mathbb{Z}^+$.

Definition 3.3.4. The share x is the function $x : \text{USERS} \rightarrow \mathbb{R}^m$ where m is the number of clusters. For a user i , her share vector is denoted by x_i and x_{ic} is her share on cluster c .

For a given system, only certain configurations may be valid. (For instance, an FPGA device may have area constraints that prohibit certain configurations). Additionally, only a subset of valid configurations might be subject to consideration.

Definition 3.3.5. The finite set of valid configurations being considered is denoted by **CONFIGURATIONS**.

Definition 3.3.6. For $p = (f, r) \in \text{PEs}$, the apparent throughput of p is $\tilde{f}_p : \text{USERS} \rightarrow \mathbb{R}_0^+$, where $\forall i \in \text{USERS}, f_p(i) = f \cdot r(i)$.

Definition 3.3.7. For a user, the apparent parallel throughput is $\hat{f} : \text{USERS} \times \text{CONFIGURATIONS} \rightarrow \mathbb{R}_0^+$. For $i \in \text{USERS}, c \in \text{CONFIGURATIONS}$

$$\hat{f}(i, c) = \sum_{\langle j, (f, p) \rangle \in c} \tilde{f}_p(i)$$

Let $P = |c|$. We can define the average parallel frequency.

Definition 3.3.8. The average parallel frequency is

$$\bar{f}(i, c) = \frac{\hat{f}(i, c)}{P}$$

Demonstration of Invariance. In Section 3.3.2 we motivated the definitions as invariant used for comparison. We formally demonstrate the invariance here. Consider a configuration c on some hardware device and a workload (set of instructions) W that is arbitrarily parallel. Let w_i be the workload on PE $c(i)$. Let $i = \textit{slowest}$, be the slowest PE. We can then evaluate the makespan T of the workload.

$$\begin{aligned}
T &= w_{slowest} \cdot \frac{1}{\tilde{f}_{slowest}} \\
W &= \sum_i \left[\frac{w_{slowest} \cdot \tilde{f}_i}{\tilde{f}_{slowest}} \right] \approx \sum_i \frac{w_{slowest} \cdot \tilde{f}_i}{\tilde{f}_{slowest}} \\
W &= T \sum_i \tilde{f}_i
\end{aligned}$$

The relation $\frac{W}{T}$ is the apparent parallel throughput. Notice, however, no assumptions were made about the PEs in the configurations. If a configuration c_1 has only one PE, the apparent parallel throughput is the apparent throughput of the PE. A workload running on a different configuration with the same apparent parallel throughput will have the same make-span. This concludes the proof.

The average frequency can be used to reduce a configuration to a 2-tuple similar to the PE as (\bar{f}, P) .

3.3.4 Modified Amdahl's Law for a Cluster

In [38], Hill proposes a modification of Amdahl's Law for the multi-core setting with one specialty core. In the hardware configuration setting, however, there maybe several specialized hardware. Here, we introduce a modified Amdahl's law for the general hardware configuration setting with heterogeneous PEs. The modification allows computing the speedup of execution within one cluster.

Definition 3.3.9. *Parallelizable ratio F_i of user i is the fraction of the workload of user i that can be parallelized.*

Using the system definitions, we can now derive the speedup a user i will receive by utilizing a cluster c . Let the parallel fraction of the workload W be F_i , and let $P = |c|$ be

the cluster size. We can derive the makespan, and consequently the speedup with respect to a base PE ($f_{base}, \mathbf{1}$).

$$T = (1 - F_i) \cdot \frac{1}{\tilde{f}_p(i)} + F_i \cdot \frac{1}{\hat{f}(i, c)}$$

and the speedup is,

$$s'_i(c) = \frac{1}{(1 - F_i) \cdot \frac{f_{base}}{\tilde{f}_p(i)} + F_i \cdot \frac{f_{base}}{\hat{f}(i, c)}}$$

For simplicity we can assume that the serial portion of execution, tasks will run on a random PE. Then,

$$s_i(c) = \frac{\bar{f}_i}{f_{base}} \frac{P}{P(1 - F_i) + F_i} \quad (3.1)$$

Now assume that the user i only receives a share of resource x_i on the cluster c . We can define x_i as follows. Consider some time duration Q during which the cluster is allocated to the users. x_i is the fraction of time Q during which user i can use *one* PE. If the user uses multiple PEs (in the parallel portions), then the usage during that portion is P times higher. To obtain the speedup, we can replace P with the share of cores available to the user x_i .

We can then modify the equation as¹,

$$s_i(x_i \cdot c) = \frac{\bar{f}_i}{f_{base}} \frac{x_i}{x_i(1 - F_i) + F_i} \quad (3.2)$$

3.3.5 Linear Model

While Amdahl's law models the behavior of a fixed workload running with different parallelism, it does not capture speed up trends of a user fully. This was noted by Gustafson in

¹For simplicity, we espouse the notation $x \cdot c$ to depict x share on a cluster c .

[32], to motivate the formulation of Gustafson’s law. They note that while parallelization of fixed workloads can be modelled by Amdahl’s law, an increase in compute resources is accompanied by an increase in the size of the workload; more resource causes more jobs to run.

This can be easily demonstrated with modern accelerator workloads, where there are several independent jobs that can be executed simultaneously. For instance, consider a CNN inference workload run by a server. It is typical for a server to have several outstanding requests from multiple clients. In this case, there are inference requests to be done on several images and not just one. While it may be possible to parallelize a network further on the addition of a CPU core, the additional CPU core can run inference on another image instead.

This motivates the linear section of the model; when multiple clusters are being allocated, users will run independent job on each cluster that they are allocated and the speedup grows linearly.

For a given configuration C and users share vector x_i , we define the speedup of a user for the cluster as the following.

$$s_i(x_i \cdot C) = \sum_c s_i(x_{ic} \cdot c) \tag{3.3}$$

3.3.6 User model

For throughput oriented users, the utility to the user is that of the speedup of its jobs. We can now model the user’s utility function.

Definition 3.3.10. *Amdahl’s Utility is the utility a user i receives from a configuration C and is equal to the speedup.*

$$u_i(c) = s_i(x_i \cdot C)$$

We can examine whether this utility model conforms to the requirement outlined in Section 3.2.1. The requirements are trivially true for the linear model, so we explore the case of PEs within a cluster here.

Theorem 3.3.11. *For the utility function $u_i(c)$,*

1. *all else equal PEs with higher frequencies are preferred, and*
2. *all else equal PEs with lower turnaround times are preferred.*

Proof. 1. Let PE A and PE A' be instances of generic design D running at frequency f_A and $f_{A'}$ respectively such that $f_A < f_{A'}$.²

$$\begin{aligned} u_i(A) &= \frac{r f_A}{f_{base}} \\ u_i(A') &= \frac{r f_{A'}}{f_{base}} \\ \implies u_i(A) &< u_i(A') \end{aligned}$$

2. Let PE A and PE A' be instances computing some function g and have the same frequencies f_c . A and A' computes g in t_A and $t_{A'}$ cycles respectively, and let $t_A > t_{A'}$.

$$\begin{aligned} r_A &= t_A \\ r_{A'} &= t_{A'} \\ u_i(A) &= \frac{r_A f_c}{f_{base}} \\ u_i(A') &= \frac{r_{A'} f_c}{f_{base}} \\ \implies u_i(A) &< u_i(A') \end{aligned}$$

□

²For simplicity, the utility from a PE A is being denoted as $u(A)$.

The third requirement is only conditionally met.

Theorem 3.3.12. *For generic PE A and PE A' with $u_i(A) < u_i(A')$ and, cluster c containing both.*

$$u_i(c) \geq u_i(A') \iff F \geq 1 - \frac{\tilde{f}_A}{\tilde{f}_{A'}}$$

Proof. Consider PE A and PE A' with $u_i(A) < u_i(A')$ and, cluster c containing both.

$$\begin{aligned} c &= \{(1, A), (2, A')\} \\ u_i(A) &= \frac{\tilde{f}_A}{f_{base}} \\ u_i(A') &= \frac{\tilde{f}_{A'}}{f_{base}} \\ u_i(c) &= \frac{\tilde{f}_A + \tilde{f}_{A'}}{2 \cdot f_{base}} \frac{2}{2(1-F) + F} \\ \frac{u_i(c)}{u_i(A')} &= \left(1 + \frac{\tilde{f}_A}{\tilde{f}_{A'}}\right) \cdot \frac{1}{2(1-F) + F} \end{aligned}$$

If $u_i(c) \geq u_i(A')$,

$$\begin{aligned} \frac{u_i(c)}{u_i(A')} &\geq 1 \\ \implies F &\geq 1 - \frac{\tilde{f}_A}{\tilde{f}_{A'}} \end{aligned}$$

If $F \geq 1 - \frac{\tilde{f}_A}{\tilde{f}_{A'}}$,

$$\begin{aligned} \implies 2 - F &\geq 1 + \frac{\tilde{f}_A}{\tilde{f}_{A'}} \\ \implies 1 &\geq 1 + \frac{\tilde{f}_A}{\tilde{f}_{A'}} \cdot \frac{1}{2(1-F) + F} \\ \frac{u_i(c)}{u_i(A')} &\geq 1 \end{aligned}$$

□

In general, **we cannot assume that an additional PE will increase the utility of an agent.**

Homogeneous Cluster. Assume now that all PEs in a cluster provide the same utility.

$$\begin{aligned} \frac{\tilde{f}_A}{\tilde{f}_{A'}} &= 1 \\ \implies 1 - \frac{\tilde{f}_A}{\tilde{f}_{A'}} &= 0 \leq F \end{aligned}$$

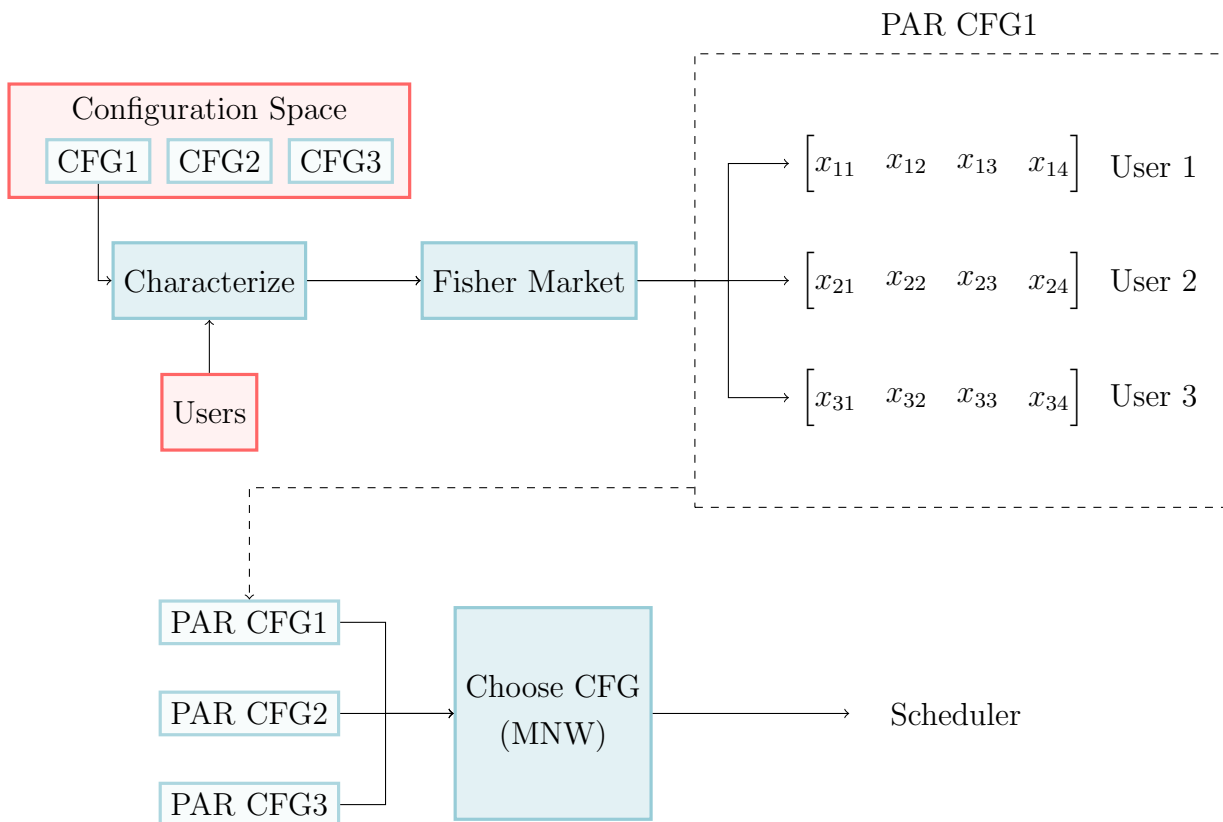
Clusters with the same PEs, therefore, satisfy all the desiderata of a utility function. In Section 4, all evaluations are performed with such clusters.

3.3.7 Partitioning and Allocation

Overview. For fair configuration, configurations needs to be analyzed with the frequency-utility model. For a given configuration, PEs can be partitioned in two ways. First, PEs maybe partitioned spatially — a PE is being used by one user. Second, PEs may be partitioned temporally — a user is assigned a set of PEs for some fixed duration of time. To guarantee fairness, these partitions need to be analyzed.

To this end, the system is constructed in the following way. We introduce a Fisher Market mechanism to evaluate optimal shares x_i for each user i for a configuration. We run Fisher market mechanisms on every configuration being considered and choose the configuration that delivers the highest Nash Welfare [26]. An additional requirement of the system is that it enforces the Fisher Market shares to guarantee optimal runs. See Figure 3.2.

Figure 3.2: System Overview: Three configurations are characterized and partitioned. CFG1 has 4 clusters and the partitions are illustrated as PAR CFG1. A configuration is then chosen based on MNW and is sent to the scheduler to enforce.



3.4 Fisher Market Allocation

In [104], Zahedi et al. introduced a Fisher Market mechanisms to allocate datacenter cores to users. They introduce the Amdahl’s Bidding (AB) procedure as a mechanism to allocate cores in a shared datacenter. Here, we introduce a similar bidding procedure for the utility and performance models in the hardware configuration setting. To solve for the market equilibrium, we use Proportional Response Dynamics (PRD) [107].

Entitlements. As noted previously, each user has an extrinsic weight or priority. In a game-theoretic market setting, this can be modelled as entitlements. Each user is entitled to time on the PEs proportional to their weight. If all agents have equal weight, then they each are entitled to an equal amount of time on the core.

Budget. In a Fisher market, each user has a starting budget. Higher the budget, higher the share of resources that the agent is able to procure. We can therefore set the initial budget of each user proportional to the weight of the user.

3.4.1 Market model

A Fisher market is designed with n participants, each deriving utility according to the Amdahl’s utility function $u_i(C)$ on a configuration C . To find the market equilibrium, we run PRD for Fisher Markets, an iterative algorithm that solves for updates on prices and bids until convergence criteria are met.

PRD Bidding Procedure — Summary

At the beginning of the procedure, the users are allocated a budget proportional to their weight. Each cluster is set to an arbitrary price. A user then bids some amount on each cluster at every round. Based on the bid, shares and new prices are computed for each cluster. The process is repeated till convergence. Each user is given a budget b_i proportional to her weight.

We summarize the bidding procedure here (see Section 3.4.2 for the derivation and exposition). At iteration t , the price of cluster c is

$$p_c(t) = \sum_i \frac{b_{ic}}{P} \quad (3.4)$$

We then calculate the following marginal utility factors (see Definition 3.4.1).

$$x_{ic}(t) = \frac{b_{ic}(t)}{p_c(t)}$$

$$U_{ic}(t) = \sqrt{\frac{F_{ic}}{f_{ic}} \cdot c(t)} u_{ic}(x_{ic}(t) \cdot c)$$

$$U_i(t) = \sum_j U_{ic}(t)$$

The new bid is then,

$$b_{ic}(t+1) = b_i \cdot \frac{U_{ic}(t)}{U_i(t)}$$

The iteration continues until a convergence criterion is met at time t^* and price p_c^* ; for some ϵ the following holds.

$$|p_c(t^*) - p_c(t^* - 1)| < \epsilon$$

3.4.2 Optimal Bids

The market equilibrium occurs when there are no more PEs units to be allocated (the market clears) and all bids maximizes a user's utility. We define these formally.

Market clears

For allocation x_{ic} on cluster c , sum of shares of all users must equal the number of cores.

$$\sum_i x_{ic}^* = P_c \quad (3.5)$$

Bid maximizes utility

For the equilibrium price, shares x_i must be a solution to the following optimization problem.

$$\begin{aligned} \max \quad & u_i(x_i) \\ \text{subject to} \quad & x_{ic} = b_{ic}/p_c \\ & \sum b_{ic} \leq b_i \\ & b_{ic} \geq 0 \end{aligned} \quad (3.6)$$

For simplicity, we define a bidding factors that quantities the preference of a user to one cluster.

Definition 3.4.1. *The bid factor of user i on cluster c is U_{ic} and total bid factor of the user is U_i for a given price vector and share.*

$$\begin{aligned} U_{ic} &= \sqrt{\frac{F_{ic}}{f_{ic}} p_c(t) \cdot s_{ic}(x_i c)} \\ U_i &= \sum_c U_{ic} \end{aligned}$$

Theorem 3.4.2. *A user exhausts her budget.*

$$\sum_c b_{ic}(t) = b_i, \quad \forall t$$

Proof. By way of contradiction, let some bid total $b'_i < b_i$ be the optimal total bid. Choose an arbitrary c . Consider a new bid for c , $b'_{ic} = b_{ic} + b_i - b'_i$

Cluster are being bid independently. Therefore, her utility from the other clusters remain unchanged. Her new share of c is x'_{ic} .

$$\begin{aligned} x'_{ic} = \frac{b'_{ic}}{p_c} &> \frac{b_{ic}}{p_c} \\ x'_{ic} &> x_{ic} \end{aligned}$$

By Theorem 3.3.11, we know $u_{ic}(x_i) < u_{ic}(x_i + \epsilon)$. The user has bid for a lower share, a contradiction. \square

Theorem 3.4.3. *For a given price vector p the optimal bids are*

$$b_{ij} = b_i \cdot \frac{U_{ic}}{U_i}$$

Proof. The optimal bid is a solution to 3.6. We use the method of Lagrangian multiplier and introduce λ_i .

$$\begin{aligned} \frac{\partial u_i}{\partial x_{ij}} &= \lambda_i \cdot p_{ij} \\ \implies b_{ic}^2 &= \lambda_i \cdot \frac{F_{ic}}{f_{ic}} \cdot p_c \cdot s_{ic}(x_{ic} \cdot c) \end{aligned}$$

For two clusters A and B ,

$$\frac{b_{iA}^2}{b_{iB}^2} = \frac{U_{iA}^2}{U_{iB}^2}$$

By Theorem 3.4.2, we know that the users exhausts the budget.

$$\begin{aligned} b_i &= \sum_c b_{ic} \\ \implies b_{ic} &= b_i \cdot \frac{U_{ic}}{U_i} \end{aligned}$$

\square

Theorem 3.4.4. (*SI*) *Users have an incentive to share.*

Proof. For user i , let x_{ic}^* be the market allocated share and let $y_{ic} = \frac{b_i}{B} \cdot P_c$ be her entitlement, where $B = \sum b_i$ is the total budget. Market clearance can be used with PRD price iterations to compute the total price. We see that since users exhaust their budget (Theorem 3.4.2), total price is total budget.

$$\sum_c P_c \cdot p_c^* = \sum_c P_c \cdot \left(\sum_i \frac{b_{ic}^*}{P_c} \right) = B$$

At equilibrium price, we can calculate the entitled budget

$$\sum_c y_{ic} \cdot p_c^* = \frac{b_i}{B} \sum_c P_c \cdot p_c^* = b_i$$

i.e. y_{ic} is a valid allocation of at price p_c^* . However, we know x_{ic}^* is optimal.

$$u_i(x_i^*) \geq u_i(y_i^*)$$

□

Theorem 3.4.5. (PE) *The mechanism is Pareto efficient.*

Proof. By way of contradiction, assume some allocation x'_i Pareto dominates x_i^* . By definition,

1. all users have at least the same utility, $u_i(x'_i) \geq u_i(x_i^*)$, and
2. one user has strictly higher utility, $u_i(x'_i) > u_i(x_i^*)$.

Since for a user i , x_i^* is optimal for prices p , she must not be able to afford x'_i . Necessarily,

$$\begin{aligned} \text{If } u_i(x'_i) \geq u_i(x_i^*) &\implies \sum_c p_c^* x'_{ic} \geq b_i \\ \text{If } u_i(x'_i) > u_i(x_i^*) &\implies \sum_c p_c^* x'_{ic} > b_i \end{aligned}$$

We can sum over all budgets. Since Pareto dominance requires at least one user to have a higher utility, the inequality is strict.

$$\sum_i \sum_c p_c^* x'_{ic} > \sum_i b_i$$

Using Theorem 3.4.2, Equation 3.4 and some algebra we arrive at the following.

$$\sum_c \sum_i x'_{ic} > \sum_c P_c$$

Total allocation is greater than the number of cores available, which is a contradiction. \square

3.4.3 Aggregating preferences — Social Welfare

Once a set of configurations have been identified, we can run the market mechanism on each configuration and evaluate the utility of a configuration to each user. We now need to aggregate the preferences and choose the ideal configuration.

Various aggregate utilities have been studied. For hardware configuration, we recommend Nash Welfare based on the following observations.

1. Hardware configurations behave like a public good [11, 26, 13] . While independent users use a given configuration to various degrees, the configuration itself is chosen once ahead of time.
2. Since the hardware configuration have longer persistence, it is desirable for users with very low utility to skew the preference down. Ideally, if a hardware is unusable for a given user (her utility is zero), it should not be picked.

In public good settings, Nash welfare has been identified a suitable aggregate of preferences.

Definition 3.4.6. *Nash welfare of a configuration C is defined as*

$$NW(C) = \prod_{i \in \text{USERS}} u_i(C)$$

The ideal configuration is C^* is then the configuration that maximizes Nash Welfare called the Maximum Nash Welfare (MNW) solution.

$$MNW(C^*) = \max_{C \in \text{CONFIGURATIONS}} NW(C) \tag{3.7}$$

3.5 Summary

In the system, we run a collection of configurations through a market mechanism. The market mechanism allocates configuration resources (clusters) and generates share vectors for each user. The system then chooses the optimal configuration that maximizes Nash Welfare. We configure to this optimal Nash Welfare providing configuration, and the share vectors are provided to the runtime scheduler to enforce.

Chapter 4

Implementation

This chapter details a case study of the mechanism outlined in Chapter 3 on the Vortex GPGPU architecture. Vortex is a parameterized configurable open source GPU architecture [89, 21]. Vortex allows for GPUs to be configured into clusters and cores, with configurable cache sizes, core capabilities (FPU, texture units etc.) and routing. The following is an overview of the implementation.

- **GPU.** The GPU is deployed on DE10-Pro, a Stratix 10 FPGA platform [57, 41]. Stratix 10 consists of a configurable FPGA fabric and a Hard-Processor-System with an ARM core. We configure an instance of the Vortex GPU on the FPGA along with associated control logic. The ARM core runs Linux and processes responsible for controlling the GPU, moving data and characterizing the GPU. We also add new performance counters to the GPU.
- **Front-End.** We use TVM [15] and OpenCL [66] to implement a front-end for user applications. We make modification to Portable OpenCL (POCL) [43] compiler to (a) perform runtime compilation and linking, and (b) extricate the system from OPAE [40].

- **Characterization and Resource Allocation.** On the Hard-Processor-System (HPS) we deploy a runtime on Linux to manage the GPU instance. It runs the mechanism outlined in Chapter 3 and deploys workloads with its recommended allocations. In addition, this stage also extracts performance counters from the GPU and characterizes F .

To evaluate the mechanism, the architecture needed several modifications. We summarize the requirements and consequent revisions here.

- **Heterogeneous Cores.** The mechanism outlined in Chapter 3 is aimed for heterogeneous cores. While Vortex GPU can be configured with different execution units (FPU, Texture units etc.), all cores are homogeneous. In our implementation, we relax this constraint. We allow the GPU to have heterogeneous clusters; all cores within a cluster are the same, but different clusters can instantiate different core types.
- **Clusters with Independent Clocks.** Once heterogeneous clusters can be instantiated, the clusters may have different maximum frequencies. In Vortex GPU, the entire GPU runs on a single clock. To allow for optimal performance, we run each cluster on an independent clock. Shared caches are run on an independent clock, and act as the clock domain crossings. Varied frequencies allow us to further exercise the performance model.
- **Estimating Parallelization Ratio.** The mechanism requires the parallelization ratio F , to allocate resource. We introduce new performance counters to Vortex GPU core to allow us to estimate the parallelization ratio F of a workload.

We present the implementation details of the above, other modifications, system implementation and evaluation.

4.1 Modification to the Architecture

4.1.1 Heterogeneous Cores

To achieve heterogeneity, Vortex GPU level architecture was modified to instantiate different clusters. Two major revisions are required here; generating clock signals, and memory systems calls. Following is a summary of the changes and design decisions made.

- **L3 cache.** We introduced asynchronous FIFOs to park memory requests and responses to the L3 cache. L2 cache from each cluster forward requests to the L3 cache on the cluster’s clock domain. The request is then enqueued on a dedicated request FIFO. The dequeue end of the FIFO runs on the L3 cache clock domain. We introduced an analogous FIFO for responses.
- **Control and Status Registers.** We need new CSRs, (`CSR_CLUSTER_CAP`) to hold flags for cluster capabilities. The CSR is intended for the application to query capabilities of a cluster. The flags have no-inherent meaning and use of the CSR is application specific; in the sample workloads, we use the CSR to check for FPU and cache size. The application software then choose specific code paths depending on the capabilities.

4.1.2 Estimating Parallelization Ratio

In Chapter 3, the mechanism uses the parallelization ratio F of a workload to allocate PEs. The parallelization ratio however is not readily apparent and needs to be estimated. In [49], the Karp-Flatt metric was proposed to estimate F .

For an application, if a speedup s is observed with P cores P , then Karp-Flatt metric estimates F as the following.

$$F_P = \left(1 - \frac{1}{s}\right) \left(1 - \frac{1}{P}\right)^{-1} \quad (4.1)$$

Table 4.1: Performance Counters to estimate F

Performance Counter	Description
CYCLE_THREAD_X	Counter increments when X threads are being executed in that cycle, values of X are 1, 2, ..., P for P threaded core.
TOTAL_ACTIVE_CYCLES	Counter increments when any thread is active.

Applications can be profiled by running multiple iterations with differing values of P and the expected value of F can be computed, a method that has been used in [104]. In this work, we introduce and use performance counters to measure the parallelization ratio. Consider a cluster with m cores. In each core, we require the performance counters presented in Table 4.1.

We introduce the notion of X -parallelization ratio $F^{(X)}$; $F^{(X)}$ is the fraction of instructions that are executed with exactly X threads. Using the performance counters, we can measure $F^{(X)}$.

$$F^{(X)} = \frac{\text{CYCLE_THREAD_X}}{\text{TOTAL_ACTIVE_CYCLES}} \quad (4.2)$$

We can express the speedup of the workload in terms of $F^{(X)}$.

$$s = \frac{1}{\sum_X \frac{F^{(X)}}{X}}$$

Using some algebra, we arrive at an estimate for F .

$$F = \frac{P}{P-1} \left(1 - \sum_{X=1}^P \frac{F^{(X)}}{X} \right) \quad (4.3)$$

4.1.3 Other Modification

For general operability and improved performance, we make several other changes. Salient of them are the following.

- **Removed OPAE-AFU.** As noted in Section 2.1.1, Vortex relies on OPAE and OPAE-AFU wrappers for data movement and OPAE has several restrictions on non-Intel host machines [40]. We host our implementation is on a server with AMD Threadripper 1920WX, and consequently we remove OPAE due to its poor compatibility. We construct a new top-level for this chip, relying on the on-chip ARM-A53 CPU on Stratix 10 to manage the GPU and drive instructions to the GPU from the host.
- **Memory lanes.** We introduce multiple DDR4 memories to the top level and added a system level cache (L4-cache) capability. In the reference design, Vortex provides primitive support for multiple memory chips at the top level. To exercise the GPU with larger datasets, we add support for 3 dedicate memory lanes, which use the onboard system memory.

4.1.4 Vortex GPU and the performance model

In context of the performance model, the following are the relevant equivalencies. Each cluster in the Vortex GPU is a cluster in the performance model. A workload is launched on a cluster and its performance is governed by the modified Amdahl’s law. Each core is a PE in the performance model, responsible for computation.

4.2 System Implementation

We implement the system on DE10-Pro [41] Stratix 10 Accelerator. We instantiate the Vortex GPU on the FPGA fabric and use a Hard-Processor-System (HPS) to control

the GPU. Figures 4.1 and 4.2 provide an overview of the system and hardware layout, respectively. We present the challenges in system implementation and summary of the design choices.

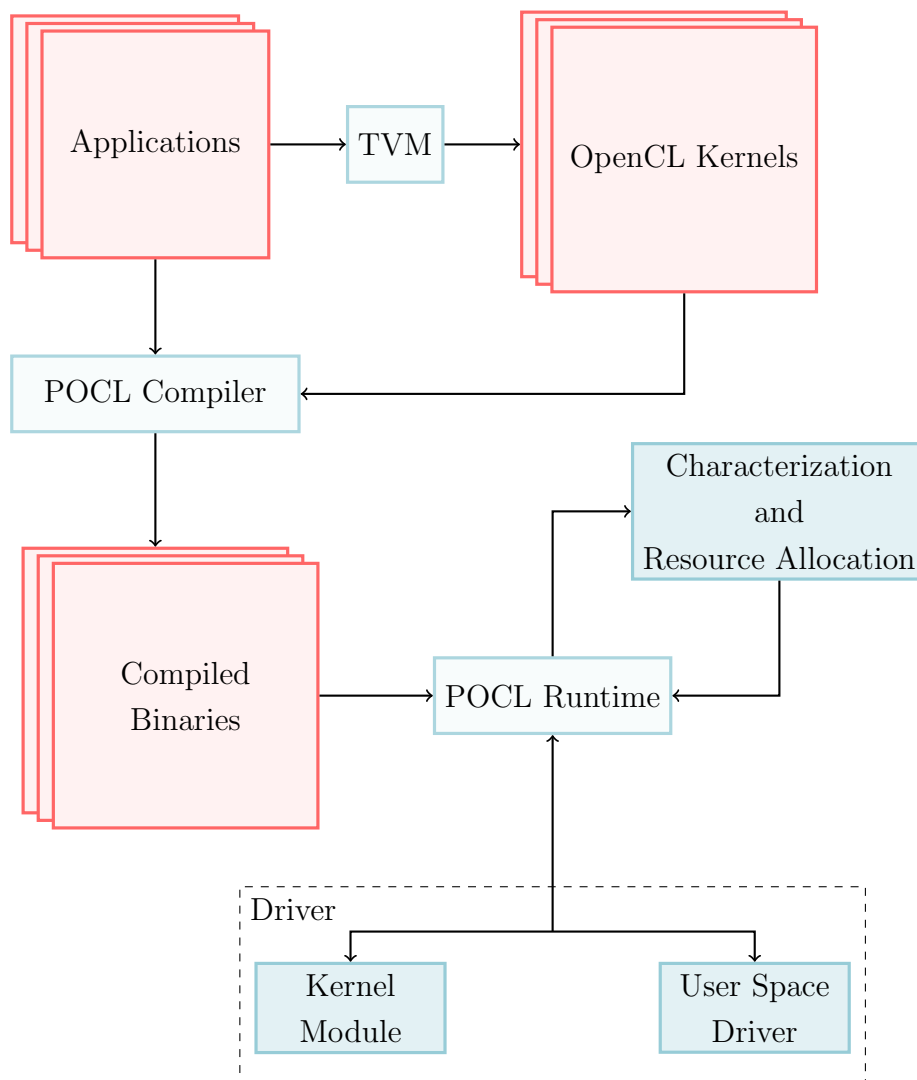


Figure 4.1: Overview of the end to end Vortex system: Path shows the different steps from application to GPU.

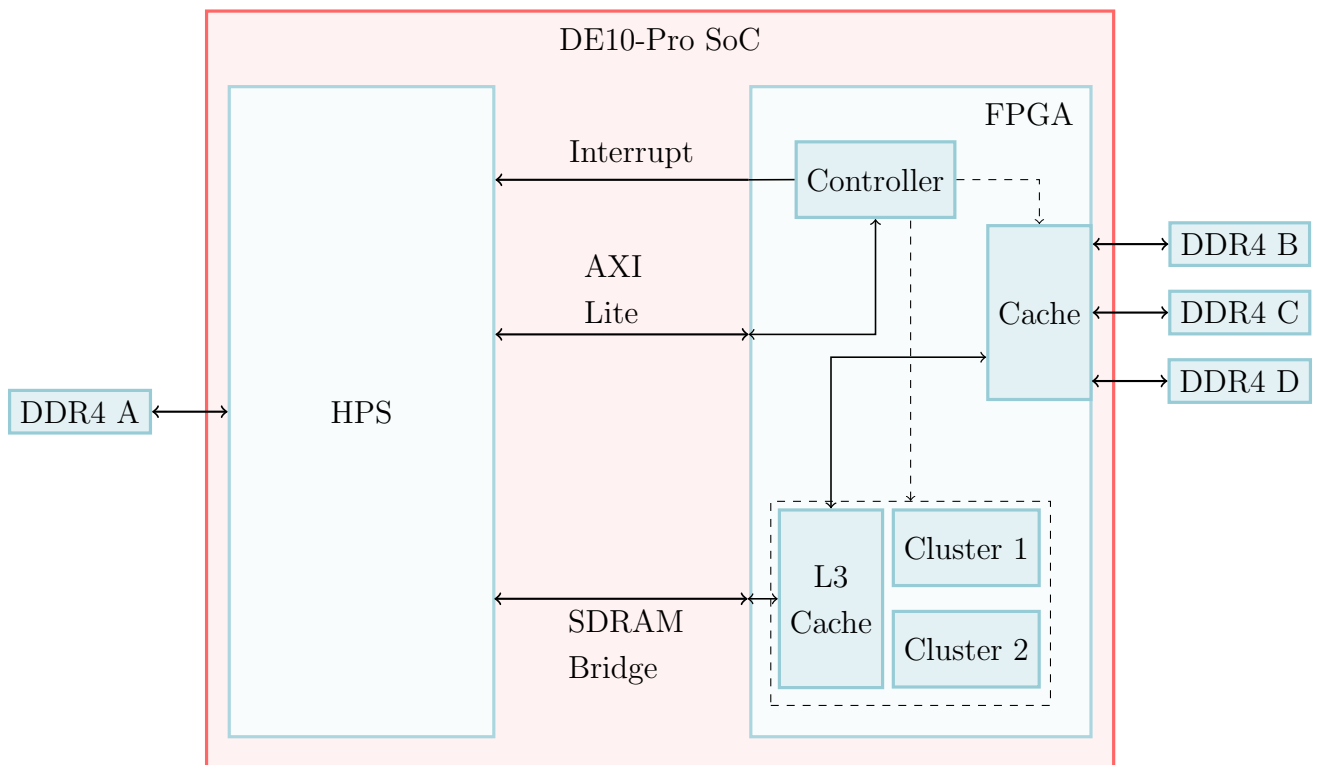


Figure 4.2: Overview of the DE10 SoC Layout for a sample 2 cluster configuration

4.2.1 Challenges and Summary

The system implementation was met with several challenges across various stages of the execution stack; the front end application layer, compiler, OS runtime, GPU runtime and hardware.

Hardware Integration

Due to challenges with Intel OPAE, Vortex AFU wrappers were removed from the implementation. This introduced challenges in data movement and control. With AFU, control and data movement signals were created in compliance with Intel OPAE specifications. In its absence, control logic now needs to be implemented in the Hard Processor System (HPS) and hardware modules are needed on the Vortex GPU system.

For data movement, we implemented support for three dedicated DDR4 memory units for the GPU; the GPU has sole access to these memory units. A memory bridge was instantiated to allow the GPU to access HPS memory. This DRAM is shared memory between the GPU and HPS, and is for data and instructions movement. To orchestrate the GPU, we introduce a controller unit capable of setting address, reporting status of clusters and enable and reset logic. The controller also generates an interrupt signal which interrupts the HPS when the GPU idles.

Frontend Compiler

The Portable OpenCL (POCL) [43] compiler was used to design benchmarks for the GPU. The GPU executes OpenCL kernels organized into work groups. For machine learning workloads, we used TVM [15] to compile neural network descriptions to OpenCL kernels. We investigated modifications of PyTorch, a more widely used machine learning framework [74], for compatibility with Vortex, but TVM showed the following advantages. TVM compiled to native OpenCL kernels that the POCL was able to compile. TVM generated

OpenCL kernels and data flow graph on these kernels. This allowed us to compile kernels, package them into separate instruction binaries and to make scheduling decisions on these kernels.

Runtime

On the HPS, we run Linux with changes to the kernel and U-boot [23] bootloader. The kernel and bootloader were updated to enable sharing of the DRAM memory by enabling the SDRAM bridge. The SDRAM bridge supports simple base-and-bound like memory translation [7]; we set appropriate values in these registers such that the GPU would be able to issue read and writes to the DRAM without triggering bus faults.

To orchestrate the GPU, we design custom drivers on Linux. The driver is responsible for servicing interrupts, notifying the runtime to issue new kernels to execute and to allocate memory. The driver ensures that memory regions of different kernels (of the same or different users) do not overlap in the three dedicate memories. Shared memory presents an additional challenge; the GPU expects memory to be contiguous, a constraint that cannot be guaranteed with Linux memory paging. To mitigate, we instantiate hugepages [71] and the shared memory between the GPU and HPS are restricted to these pages. The driver is also responsible for allocating memory on these huge pages to the kernels and users.

4.2.2 System Implementation Details

Memory

The DE10-Pro has 4 8GB DDR4 RAM chips on board. For data movement, the primary DDR4 (shared RAM) is used by the HPS. The other 3 DDR4 units (dedicated RAM) are allocated for sole use by the GPU. On chip RAM on the FPGA fabric is used a cache between the three DDR4 and GPU.

Table 4.2: Register mapping of Vortex Control Signals

Address	Register	Description
0xFFC00F00	Enable-and-Reset	Bit 0 enables the GPU and setting 1 will trigger a reset
0xFFC00F04	Interrupt Status Register	When an interrupt is served, register read 0x1. On service, driver writes 0x0.
0xFFC00F08	Cluster Status Register	Register indicates with clusters are active.
0xFFC00F0C	Instruction Start Address	Starting address of the Vortex instructions

Controller Logic

To control the Vortex GPU system from the ARM HPS, we implement 3 FPGA-HPS connection. First, we use FPGA-to-SDRAM connections to service memory requests from the Vortex GPU to the HPS subsystem. Second, we use AXI lite bus to communicate with the controller logic in the Vortex GPU subsystem. Lastly, Vortex GPU can interrupt the HPS to notify the HPS of events.

The controller logic is as a simple state machine, and we provide the register mapping of the logic in Table 4.2.

The controller also manages the generation of interrupts — Vortex subsystem notifies the HPS when one of the clusters are idling (when all the cores in the cluster does not have any more warps to execute) by triggering an interrupt.

Floating point intrinsics

Some workloads under evaluation require floating point units but, as noted in 2.1.1, not all cores are guaranteed to have an FPU. For purposes of comparison, we use SoftFloat [34]; SoftFloat provides 32-bit floating point support for architectures with 32-bit ALUs.

We also introduce modifications to the compiler stage to appropriately lower the floating point instructions to use these intrinsics.

U-boot

For booting and managing the SoC (System-on-Chip), we use U-Boot [23]. The vendor-provided baseline implementation is modified in two major ways. First, memory protection regions were updated on the HPS to allow for the GPU to communicate with the shared RAM. Second, the existing system was limited to 4GB of memory and did not support the 8GB RAM required. Additional changes were also made to the device tree to allow for interrupts from the Vortex system and to support hugepages [10, 71] for shared memory.

Linux Driver

The HPS system runs Linux with a Debian Root File System [52]. We implement two drivers in the Linux system — a kernel space driver and user space driver. The kernel space driver is responsible for forwarding interrupt notifications to the user space system. In addition, it manages the FPGA HPS bridges, namely the AXI Lite bridge and SDRAM bridge. Address space translation is done here to compute the correct addresses as seen by various buses. The kernel level driver is also responsible for translating hugepage addresses.

We write the critical section of the driver to run in user space. We exploit Linux’s `mmap` calls to allow for user-space mapping of the Vortex controller memory region. The user-space driver also instantiates hugepages. User space drivers have been shown to have better performance profile and lower heads due to syscall-bypass [110, 85, 45, 100].

The Stratix 10 system allows only for offset and base-and-bound [7] like address translation from the FPGA subsystem. Vortex GPU expects memory regions to be contiguous; for shared memory however, Linux may allocate pages for staging data and instructions in fragmented pages [10], restricting the size of workloads’ data and instruction section to the page size of the OS (which by default is 4KiB [7]). To provide larger memory to the

GPU, we employ hugepages for shared memory. Four huge pages of size 1GB each were instantiated on the shared memory. A custom memory allocator was written to sit atop the hugepages for shared memory allocation.

OpenCL

To program workloads (or to compile a workload) we used OpenCL [66]. The Portable OpenCL (POCL) [43] compiler and runtime were used as the OpenCL compilation and runtime system. The OpenCL port for Vortex backends did not provide runtime compilation, a feature required for evaluation. In addition, the provided OpenCL drivers used a Vortex driver API based on the Intel OPAE [40] framework. Due to the new implementation huge pages and multiple workloads running at the same time, the driver API was no longer appropriate.

The new driver is non-blocking, where kernel launch and other housekeeping calls are not blocked. New instructions to be issued to the GPU are staged and held in queue. If the queue is full, the application can optionally block; this allows for the application to monitor the Vortex GPU and deploy kernels concurrently.

TVM

We write tensor and Convolutional Neural Network (CNN) workloads with TVM [15]. TVM implements an executor layer responsible for backend execution of the compiled Tensor representation called Graph-Executor. We modify the Graph-Executor to compile to Vortex OpenCL and to use the POCL compiler as the OpenCL implementation.

Chapter 5

Evaluation

In this chapter, we evaluate the mechanism and implementation (see Chapter 3 and Chapter 4). We construct benchmarks, evaluate the market implementation and measure the performance and fairness of the market allocation. Key findings are the following.

- We find that the market converges rapidly; for sample workloads, in 15 iterations, the market converged to prices within 10^{-4} .
- The performance model is reliable; the model predicts runtime of the benchmarks to within a 5% error.
- Across all experiments, the market allocation provides at least entitlement performance to all users, and up to 2x increase in performance.
- The market delivers upto 10% higher utilization compared to entitlement and weighted-equal speedup.

5.1 Methodology

5.1.1 Benchmarks

We use the following representative workloads in the evaluation. The benchmarks provide a mix of highly parallel (inference) and mostly serial workloads (clustering and training). The clustering and serial workloads are also capable of issuing 32-bit floating point instructions.

- **Inference workloads.** We run inference on three networks; Resnet-50 [35], YoloNet [78], and AlexNet [53]. We use TVM models of these networks with pre-trained weights with reduced 8-bit unsigned integer precision. The models were then compiled to OpenCL kernel.
- **Clustering workload.** We run k-Means clustering [3] written in OpenCL on the Vortex GPU. The algorithm parameter is $k = 7$ with 32-bit floating point precision.
- **Training workload.** We train a multi-layer perceptron (MLP) [76] with a set of sampled 10000 data points. We design the training workload in TVM and export it to OpenCL. Additional modification were made to account for some compiler restrictions to optimize the constituent kernels.

5.1.2 Setting

We deployed a Vortex GPU configuration on the Stratix 10 FPGA. Several collections of workloads are programmed to run on the Vortex GPU, all workloads organized as kernels. A user requests a specific workload be run on a specific cluster (this is typical of specialized hardware settings where specific functions target specific functional units in the hardware). A scheduling application then launches the constituent jobs of each of the workloads to the corresponding cluster.

We dump metrics from the hardware and use it to run the market algorithms outlined in Chapter 3. The metrics are evaluated by a runtime engine and trigger reconfiguration

events in the FPGA. Characterization learns the parallelization ratio of the users' workloads and across iterations converge to a final configuration.

5.1.3 Configuration Space and Parameters

We run all experiments on 4-cluster configurations. We change three parameters to generate the configuration space; the number of threads, number of cores and FPU capability. The configurations are synthesized for the Stratix 10 FPGA, where each cluster is running within 5MHz of its maximum frequency.

5.1.4 User Sets

Since we are evaluating sharing, we require a set of representative users that are running the benchmarks in Section 5.1.1. We generate three user sets.

User Set 1. We design the first user set to evaluate how the mechanism provisions specialty resources. We construct three users, where each user runs one workload. We choose Resnet, k-Means, and MLP to represent a highly parallel, somewhat parallel and mostly serial workload respectively. Only k-Means and MLP uses FPU resources and, therefore, User 2 and User 3's allocation provides insight into whether the mechanism respects preferences.

User Set 2. We design the second user set to evaluate whether the mechanism inspires strategic bidding and how users are compensated for lack of resources on one cluster. We construct three users, where some user runs different workload. User 1 continues to run Resnet as with the previous user set. We choose k-Means, and MLP to run on the FPU clusters by User 2 and User 3 respectively. User 2 and User 3 will also run AlexNet and YoloNet on the remaining clusters. Cluster 3 and Cluster 4 will have demand from all three users to run highly parallel workloads.

User Set 3. The third user set evaluates whether the mechanism respects entitlements and delivers weighted speedup when all workloads are the same. In this set, three users run the same highly parallel Resnet workload.

5.1.5 Metrics and Comparisons

For each user, we measure the speedup across different configurations and each user’s allocations of resources. We compare the market against weighted equal speedup and entitlement only sharing. Speedup ratio of market allocations to entitlement demonstrates sharing incentive. In addition to speedup, we measure the aggregate utilization of resources.

The performance of the market is evaluated by the convergence rate. We run the market for several iterations and report the price and convergence threshold at each iteration.

5.2 Evaluation

5.2.1 Parallelizable fraction estimation

The parallelizable ratio of the benchmarks presented in Section 5.1.1 is measured; this is then compared with the speedup model outlined in Chapter 3.

Setting

We run the benchmarks on 4 different hardware configurations — 4 Core - 4 Thread (4C4T), 4 Core - 8 Thread (4C8T), 8 Core - 4 Thread (8C4T), and 8 Core - 8 Thread (8C8T). We extract the performance counters (see Section 4.1) to measure the expected speedup and estimate F .

On the 8C-8T, we sweep the number of threads for each benchmark. For the purpose of evaluation, we vary the input size to normalize the runtime to 1000ms runtime for 1 core. For the inference workloads, this corresponds to the batch size. For k-Means and MLP we change the number of data points. We then measure the deviation from the esimated F .

Measurement Results

The distribution of F values are shown in Figure 5.1 and distribution of times are shown in Figure 5.2.

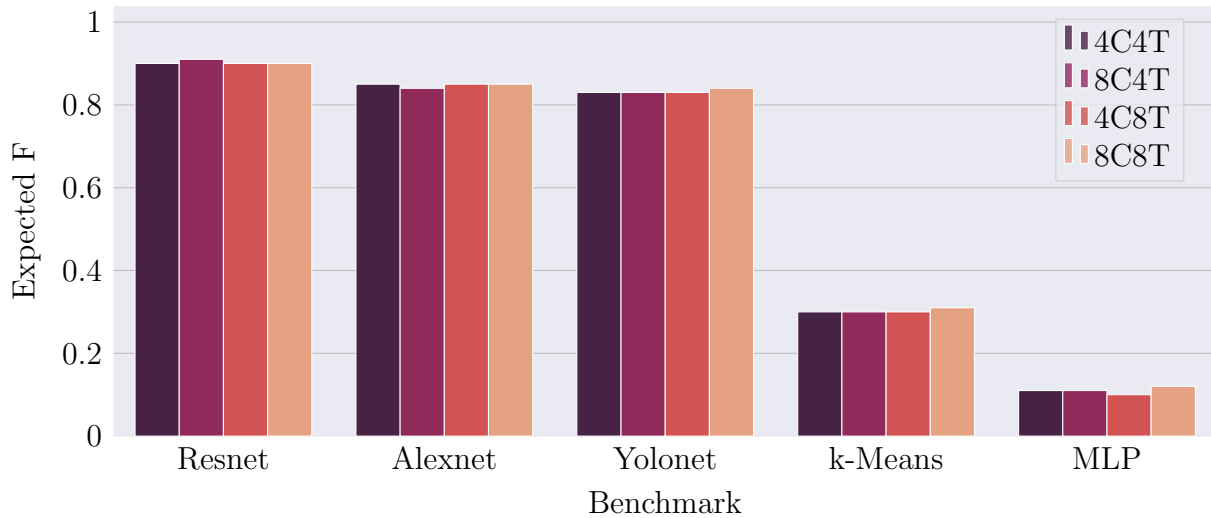


Figure 5.1: Expected F values measured for various benchmarks

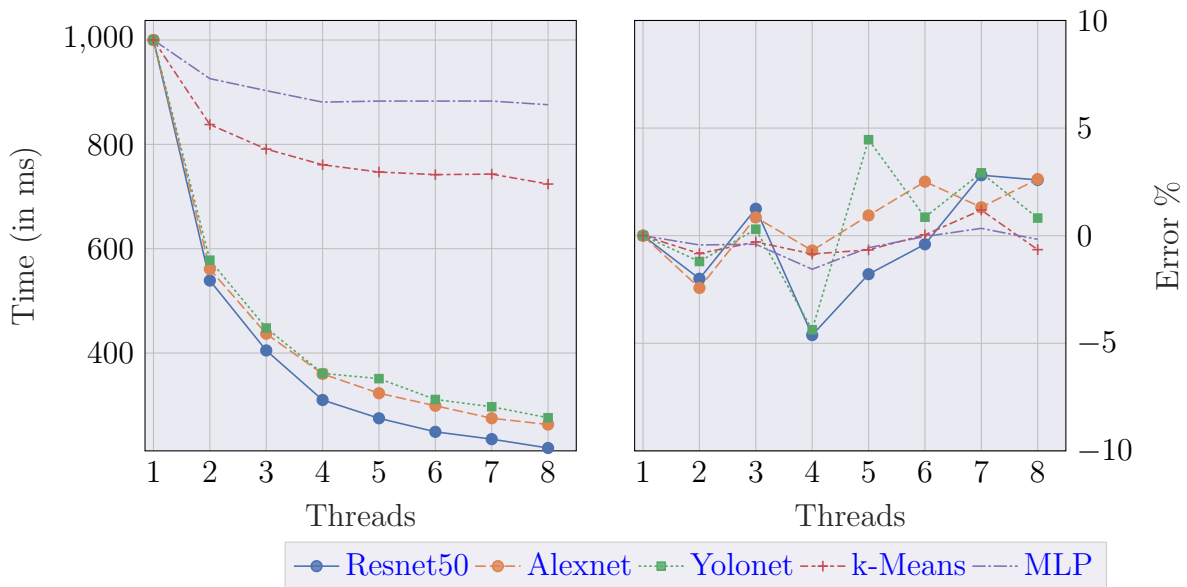


Figure 5.2: Execution time measured for benchmarks running in isolation and deviation from the prediction of the model. The expected F value measures are accurate and can predict speedup to within a 5% error margin.

5.2.2 Market mechanism

Setting

The performance of the market is measured to evaluate the convergence of prices, and to measure the number of iterations required to converge. Metrics are evaluated on the following configuration.

A set of 3 users are modelled that have the following workloads they would like to run.

We evaluate the performance of the market by looking at the number of iterations to converge to a price.

Cluster	Cores	Capability
Cluster 1	2	Floating point
Cluster 2	2	Floating point
Cluster 3	4	-
Cluster 4	8	-

Table 5.1: Sample configuration. Cluster 1 and Cluster 2 have 2 cores with floating point units (FPUs). Cluster 3 and Cluster 4 have no FPU, but have larger sizes with 4 and 8 cores respectively.

User	Cluster 1	Cluster 2	Cluster 3	Cluster 4
User 1	Resnet	Resnet	Resnet	Resnet
User 2	k-Means	k-Means	AlexNet	AlexNet
User 3	MLP	MLP	YoloNet	YoloNet

Table 5.2: Sample Users. User 1 runs Resnet on all clusters. User 2 runs k-Means on the floating point clusters and AlexNet on the clusters. User 3 runs MLP on the floating point clusters and YoloNet on the remaining clusters.

Measurements

We see that the market converges rapidly. The market converges in iterations on the order of 100 for 10^{-5} change in price. Figure 5.3 shows the number of iterations to converge to achieve a convergence difference ϵ and prices at the iterations.

5.2.3 3 Configuration Case Study

Setting

Here we perform end-to-end measurements of the system and evaluate the configuration scheme. We consider the following configuration space as a sample.

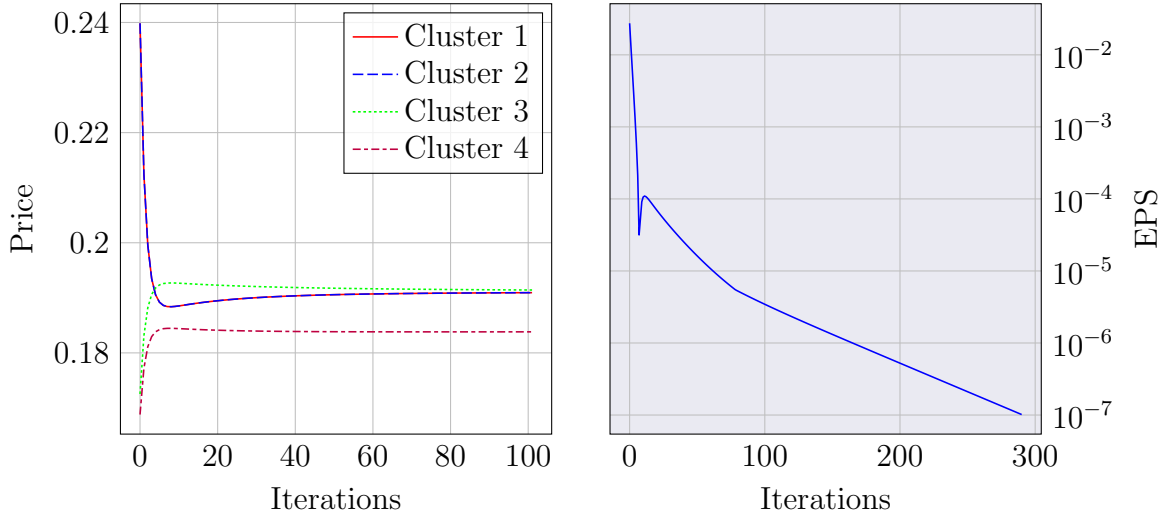


Figure 5.3: Prices for clusters with the sample workload and (ϵ) for each iteration. Prices converge rapidly — at 15 iterations the error is within 10^{-4} . Clusters are priced based on their capability and relative demand. Cluster 1 and Cluster 2 that have equal capability and have the same relative demand, so are priced equally.

Configuration ID	Cores	Threads	Features	Frequency (MHz)
CFG1	2,2,4,8	4,4,4,4	FPU,FPU,-	80, 80,120,110
CFG2	2,2,4,8	4,8,2,4	FPU,FPU,-	80, 73,120,110
CFG3	2,2,4,8	8,8,4,4	-	80, 80,110,110

Table 5.3: Configurations used. All configurations have 4 clusters, and each comma separated value indicates the value corresponding to the cluster.

Arbitrarily, we assume initially that the workloads have a parallelization ratio of 80%. The characterization (see Section 4) learns F after 1 iteration and we report converged values here.

We consider three user sets. In User Set 1 a user has one workload to run. In User Set 2, users have different workloads. Finally, in User Set 3, all users are running the same workload. In each case users have weights (corresponding to entitlements) of 1, 4 and 1 respectively.

Measurement

User Set 1 is summarized in Table 5.4. We measure speedup for the different configurations. Maximum Nash Welfare configuration was found to be CFG2.

User	Cluster 1	Cluster 2	Cluster 3	Cluster 4
User 1	Resnet	Resnet	Resnet	Resnet
User 2	k-Means	k-Means	k-Means	k-Means
User 3	MLP	MLP	MLP	MLP

Table 5.4: User Set 1. All users run the same workload on all clusters; User 1 runs Resnet, User 2 runs k-Means and User 3 runs MLP.

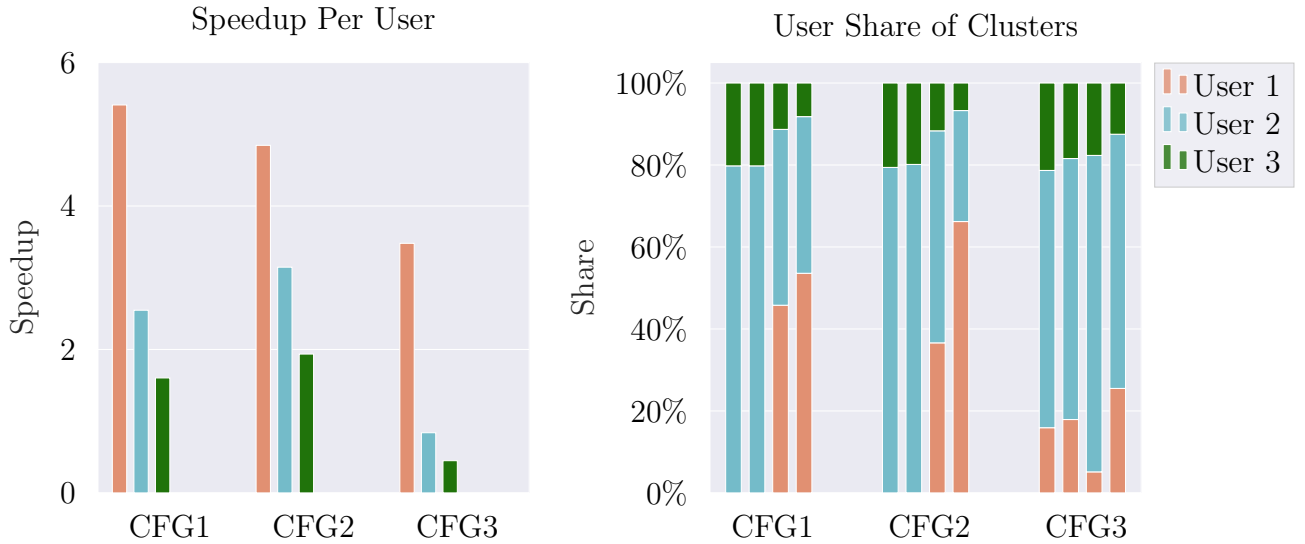


Figure 5.4: User Set 1 speedup and allocation metrics. In CFG1 and CFG2, User 1 receives shares on the higher thread count clusters and is excluded from the FPU cores and User 2 and User 3 are allocated shares on Cluster 2 and Cluster 3. In all configurations, User 2 receives a higher proportion due to its higher weight.

Observations. k-Means and MLP uses floating point instructions and prefer to run on clusters with FPUs. Resnet jobs are highly parallel and prefer a higher thread count. In

CFG1 and CFG2, we observe that Resnet workloads are entirely excluded from the FPU clusters. MLP and k-Means jobs share the FPU clusters. To compensate, we see that the mechanism gives User 1 a higher proportion on Cluster 3 and Cluster 4. In CFG3, since no cluster has an FPU, we see no such bias. However, Cluster 4 has higher number of threads. The highly parallel Resnet job receives a higher share of Cluster 4 and a lower share of other clusters.

User Set 2 is summarized in Table 5.5.

User	Cluster 1	Cluster 2	Cluster 3	Cluster 4
User 1	Resnet	Resnet	Resnet	Resnet
User 2	k-Means	k-Means	AlexNet	AlexNet
User 3	MLP	MLP	YoloNet	YoloNet

Table 5.5: User Set 2. User 1 runs Resnet jobs on all clusters. User 2 runs k-Means on Cluster 1 and Cluster 2 with FPU cores, and runs AlexNet on the remaining clusters. User 3 runs MLP on Cluster 1 and Cluster 2 also using FPU cores, and runs YoloNet on the remaining clusters.

Observations. In this User Set, User 2 and User 3 have a mix of low parallelism job (k-Means and MLP respectively) and higher parallelism job (AlexNet and YoloNet respectively); User 2 and User 3 have demands on clusters with higher cores, not just User 1. Here, we therefore expect User 1 to receive some time on FPU clusters compared to the previous set. This is reflected in our observation in CFG1 and CFG2. In CFG3, we see strategic bidding from User 1. Unlike User Set 1, User 2 and User 3 have a demand of the highly parallel cores in cluster 3 and cluster 4. User 1 therefore strategically bids higher for cluster 1 and 2 where User 1 and User 3 are running low parallelism jobs.

User Set 3 is summarized in Table 5.6.

Observations. In this User Set, all users are running the highly parallel Resnet job. We expect the speedup of all jobs to be equal to their entitlement. We observe that User 1 and User 3 have the same speedup across all configurations, confirming that users with equal entitlements receive equal utility. User 2 has an entitlement 4 times higher than User

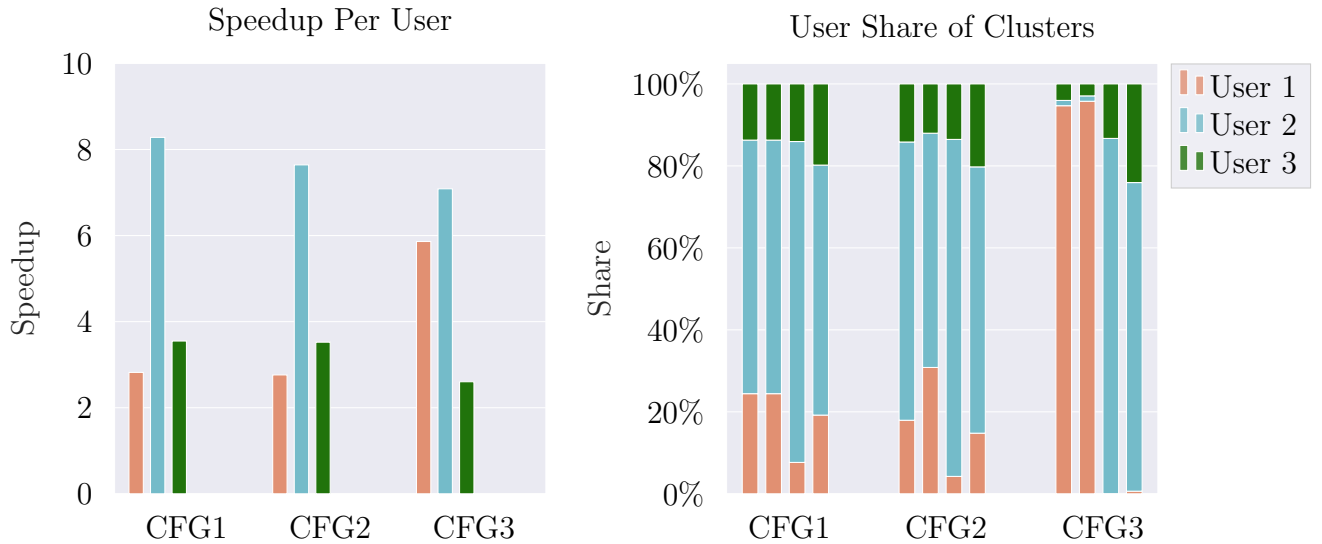


Figure 5.5: User Set 2 speedup and allocation metrics. In CFG1 and CFG2, User 2 and User 3 compete for high thread count clusters, Cluster 3 and Cluster 4. User 1 therefore receives some time on FPU clusters. In CFG3, Cluster 1 and Cluster 2 do not have FPUs. User 1 strategically bids higher for Cluster 1 and Cluster 2 permitting it to run on a larger thread count without competing with User 2 and User 3 on Cluster 3 and Cluster 4.

User	Cluster 1	Cluster 2	Cluster 3	Cluster 4
User 1	Resnet	Resnet	Resnet	Resnet
User 2	Resnet	Resnet	Resnet	Resnet
User 3	Resnet	Resnet	Resnet	Resnet

Table 5.6: User Set 3. All users run the same Resnet workload on all clusters.

1 and User 3. User 2 must receive a speedup that is $\frac{2}{3}$ higher than User 1 and User 3. We observe this in Figure 5.6.

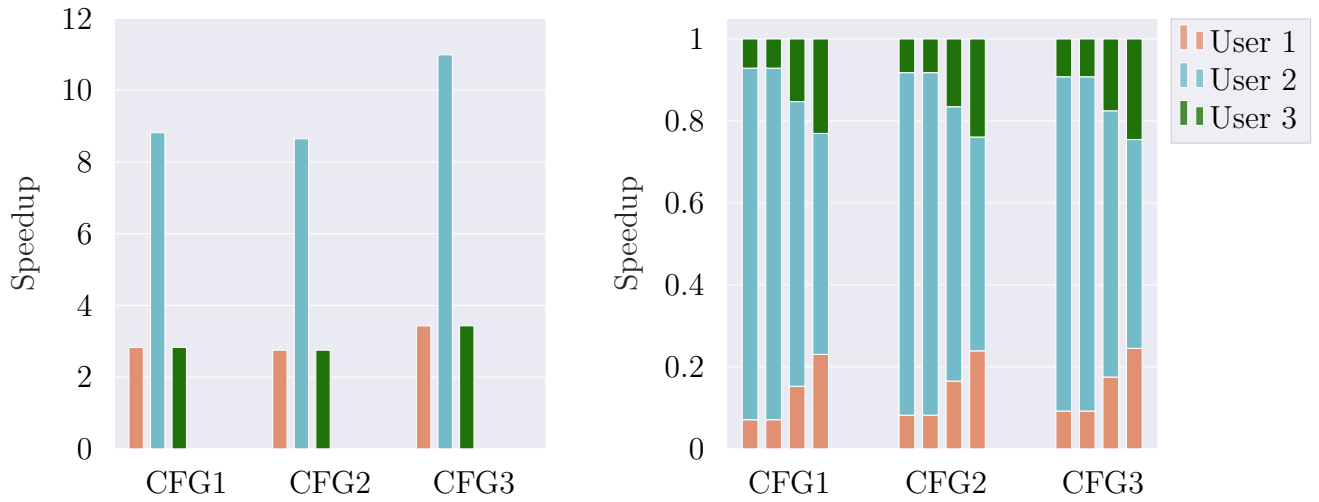


Figure 5.6: User Set 3 speedup metrics. Since the users are running the same workload, in all configurations, the speedup is distributed strictly proportional to the user’s weights.

5.2.4 Utilization

The utilization of the cores are presented in Figure 5.7. The market delivers a high resource utilization, particularly in cases where all jobs are highly parallel. Markedly, utilization of cores with a mix of low parallelism and high parallelism jobs are high.

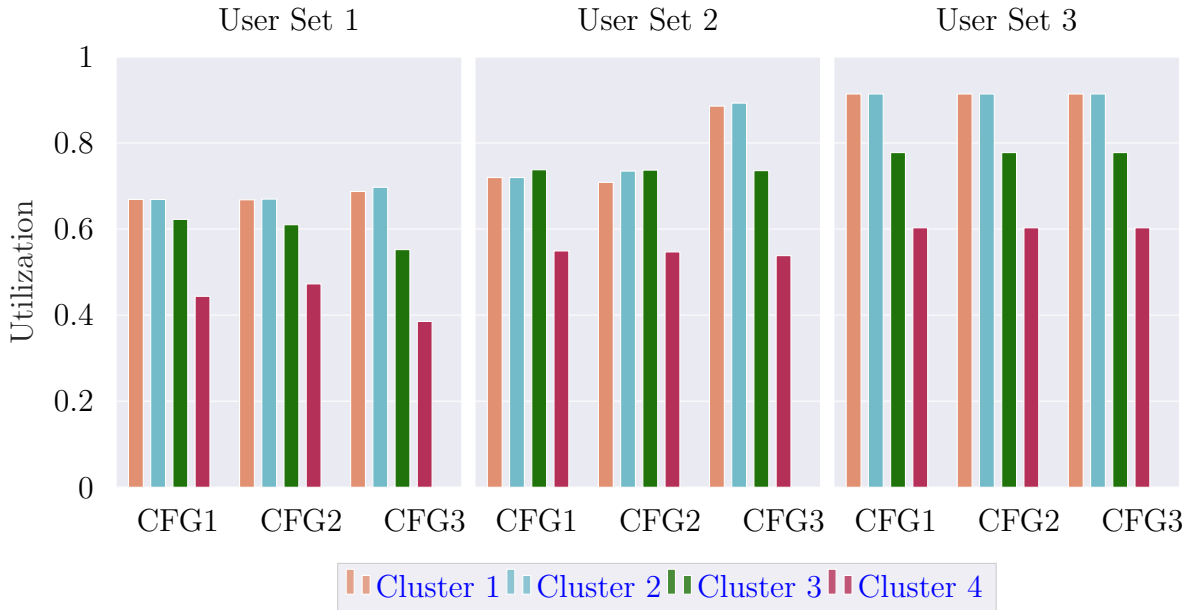


Figure 5.7: Utilization for User Sets across different configurations

5.3 Comparison to other Allocation Mechanisms

We compare the Fisher Market mechanism to two allocation schemes, weighted equalization of speedup and entitlement. For comparison, we implement these as run-time mechanisms. For weighted equalization of speedup, we perform iterations to measure F and predict the shares required to achieve equal speedup. We compare only against the final iteration and treat early iterations as warm up for the algorithm.

- **Entitlement.** In entitlement runs, shares per cluster of each user are set to their entitlement (proportional to weight). To implement this system, we run a user’s jobs until she exhausts her time. We then switch to the next user on that cluster. This is repeated for a fixed duration (10 seconds).
- **Weighted Equal Speedup (weighted max-min).** In weighted equal speedup, each user is given shares such that their speedups are equal normalized to the entitlements. For comparison, we run all jobs first with equal shares and estimate F and

\bar{f} . We then solve for shares by using the following optimization program for each cluster.

$$\begin{aligned} \max \quad & u_{c,\min} \\ \text{subject to} \quad & u_{c,\min} = \min \frac{u_{ic}(x_{ic})}{w_i} \end{aligned} \tag{5.1}$$

The jobs are run with the new shares. If the speedups are not equal, the algorithm is run again with the new expected speedup. In practice, we observe that this iteration only occurs once.

We compare them across two metrics — speedup and utilization.

Speedup

Speedup of Weighted Equal Speedup and Entitlement are shown in Figure 5.9 and Figure 5.8 respectively. Equalizing speedup favors the low parallelism workloads. Users with lower parallelism require a much higher share to achieve equal speedup. Equalizing speedup therefore requires provisioning more cores to low parallelism workloads, in strict contrast to the market. Across all user sets, we see a weighted allocation of speedup, with User 2 receiving 4 times higher speedup in all cases.

Equalizing entitlements was shown to be no better than Fisher Market allocations (Theorem 3.4.4). In User Set 1, we observe that when users request specific capabilities, Fisher market is able to deliver a higher speed up compared to their entitlement for users that do not require the specialty cores. In a Fisher market setting, users trade their specialty cores for non-specialty cores (and vice versa). In all cases, Fisher market outperforms entitlements, validating the implementation.

Utilization

Fisher Market also delivers a higher utilization of clusters compared to other schemes. In User Set 1, Fisher market provides a higher utilization in all cases. This is expected since

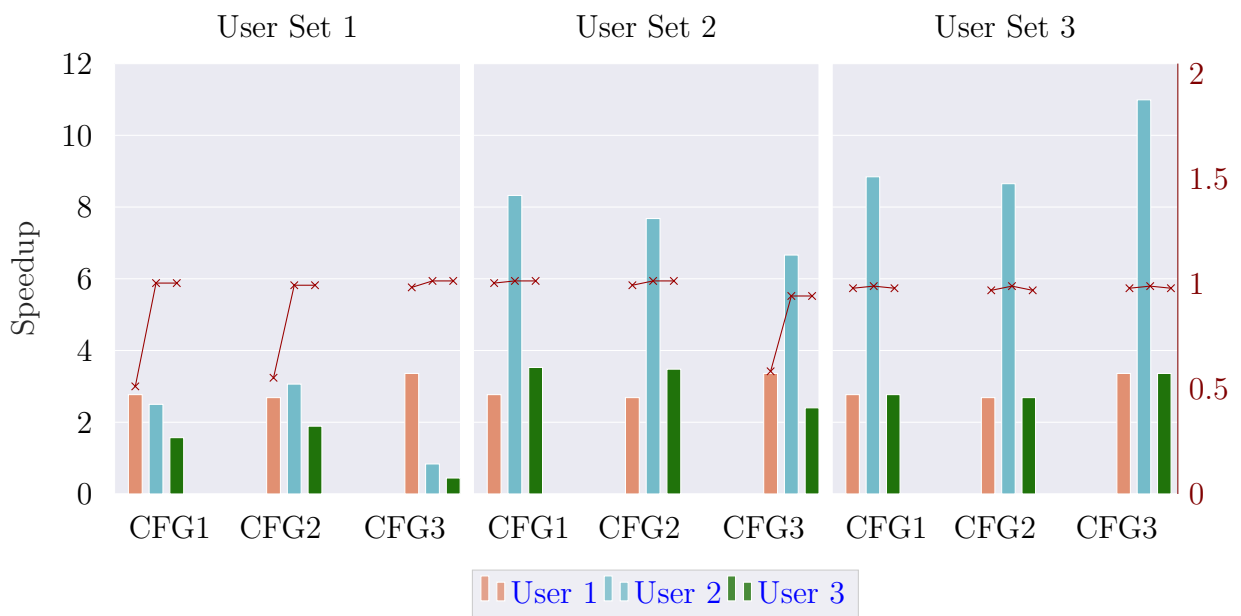


Figure 5.8: Speedup with Entitlements across different configurations. **Ratio to Fisher Market speedup is shown in red.** In all cases, users receive speedups below Fisher Market speedup.

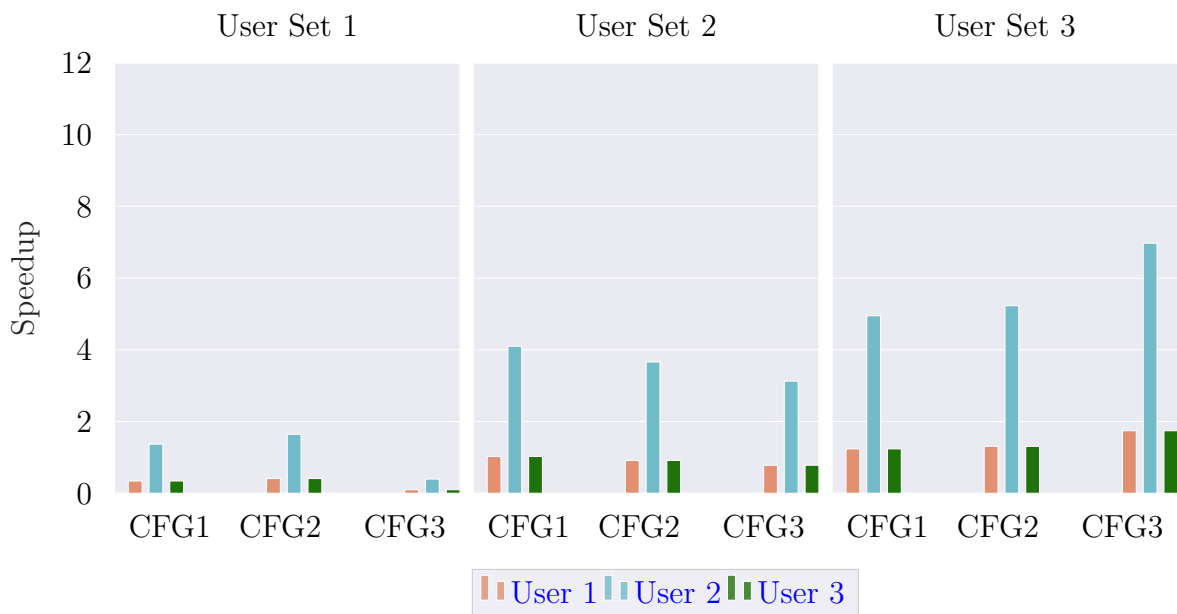


Figure 5.9: Speedup with Weighted Equalized Speedup across different configurations.

each user has a workload that is specifically skewed; highly specific, low parallelism or highly parallel, low specificity. In a market, a highly parallel user can trade her specialty cores for more cores, and in general improves utilization. In User Set 2, users have a mix of highly parallel and lowly parallel task. Here we expect lowly parallel tasks to assert more demand and thus a reduction in utilization. We observe that Fisher market is better performing or comparable to other schemes. In User Set 3, all users are running the same workload, so all schemes will deliver the same utilization.

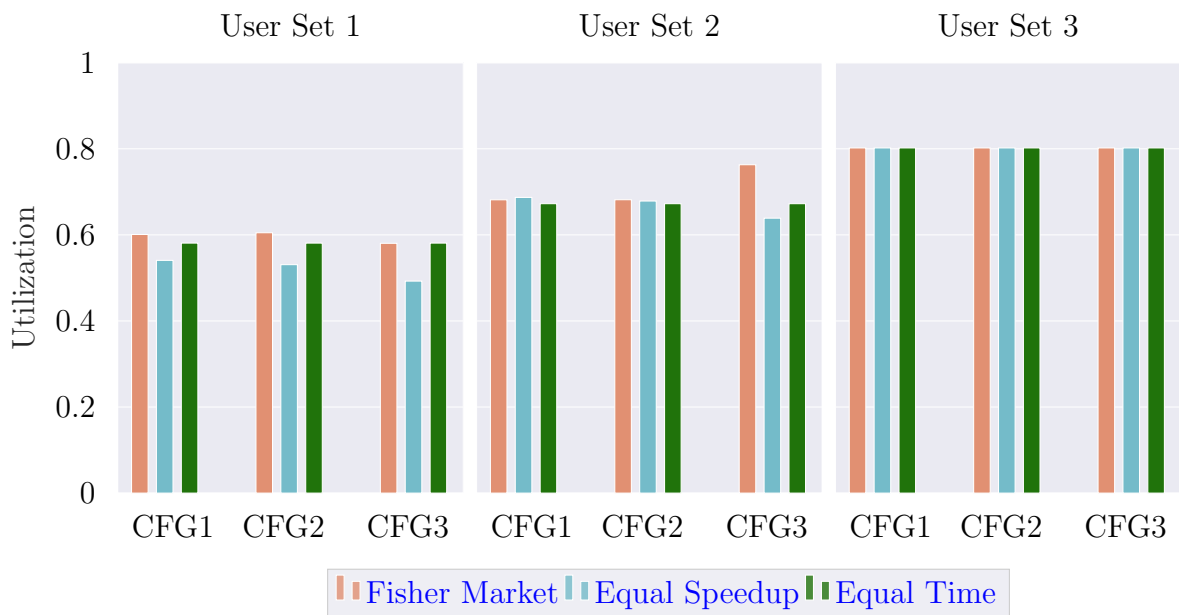


Figure 5.10: Utilization for User Sets across different configurations

Chapter 6

Conclusions and Future Work

We explore multi-user sharing of hardware accelerators and present an algorithm to allocate hardware resources to users. The algorithm is based on Fisher markets, where hardware resources are allocated prices and users bid on hardware resources. The user bids are used to allocate shares of the hardware resources, and the iterative algorithm terminates when the prices converge. We then use the notion of Nash Welfare to select the configuration that maximizes cumulative utility to all users. We provide theoretical guarantees on fairness and efficiency of the market mechanism.

In our evaluation, we find that the market provides at least the speedup from the user's entitlements (if there were no sharing mechanism) and up to 2x higher speedup per user. The allocations from the market mechanism improve utilization by up to 10%.

Future Work

Several extensions of this work remain to be explored and can be the subject of future work. We highlight a few here.

First, the performance model and allocations consider execution units only. Other resource types, such as memory bandwidth, cache and storage are not analyzed. Bottlenecks

from such resources may impede users deriving maximum utility from the cores. A promising future direction is to explore allocation of such resources in the hardware accelerator setting.

Second, beyond the implementation on Vortex, this work can extend to other architectures that similarly provide high configurability. Recently, extensible target-agnostic compiler frameworks such as MLIR has emerged [54]. Using MLIR can relax the reliance on OpenCL in our implementation, allowing for better analysis and optimizations.

References

- [1] No agent left behind: Dynamic fair division of multiple resources. *Journal of Artificial Intelligence Research*, 51:579–603, 2014.
- [2] Sagheer Ahmad, Sridhar Subramanian, Vamsi Boppana, Shankar Lakka, Fu-Hing Ho, Tomai Knopp, Juanjo Noguera, Gaurav Singh, and Ralph Wittig. Xilinx first 7nm device: Versal ai core (vc1902). In *Hot Chips Symposium*, pages 1–28, 2019.
- [3] Mohiuddin Ahmed, Raihan Seraj, and Syed Mohammed Shamsul Islam. The k-means algorithm: A comprehensive survey and performance evaluation. *Electronics*, 9(8):1295, 2020.
- [4] Mohammed A Noaman Al-hayanni, Fei Xia, Ashur Rafiev, Alexander Romanovsky, Rishad Shafik, and Alex Yakovlev. Amdahl’s law in the context of heterogeneous many-core systems—a survey. *IET Computers & Digital Techniques*, 14(4):133–148, 2020.
- [5] AMD. Amd pensando™ infrastructure accelerators, 2021. <https://www.amd.com/en/accelerators/pensando>.
- [6] Gene M Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, spring joint computer conference*, pages 483–485, 1967.
- [7] Thomas Anderson and Mike Dahlin. *Operating Systems: Principles and Practice, volume 1: Kernel and Processes*. Recursive books, 2014.

- [8] Remzi H Arpaci-Dusseau and Andrea C Arpaci-Dusseau. *Operating systems: Three easy pieces*. Arpaci-Dusseau Books, LLC, 2018.
- [9] Krste Asanović and David A Patterson. Instruction sets should be free: The case for risc-v. *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2014-146*, 2014.
- [10] Arkaprava Basu, Jayneel Gandhi, Jichuan Chang, Mark D Hill, and Michael M Swift. Efficient virtual memory for big memory servers. *ACM SIGARCH Computer Architecture News*, 41(3):237–248, 2013.
- [11] Eric Budish. The combinatorial assignment problem: Approximate competitive equilibrium from equal incomes. *Journal of Political Economy*, 119(6):1061–1103, 2011.
- [12] Ravi Budruk, Don Anderson, and Tom Shanley. *PCI express system architecture*. Addison-Wesley Professional, 2004.
- [13] Ioannis Caragiannis, David Kurokawa, Hervé Moulin, Ariel D Procaccia, Nisarg Shah, and Junxing Wang. The unreasonable fairness of maximum nash welfare. *ACM Transactions on Economics and Computation (TEAC)*, 7(3):1–32, 2019.
- [14] Adrian M Caulfield, Eric S Chung, Andrew Putnam, Hari Angepat, Jeremy Fowers, Michael Haselman, Stephen Heil, Matt Humphrey, Puneet Kaur, Joo-Young Kim, et al. A cloud-scale acceleration architecture. In *2016 49th Annual IEEE/ACM international symposium on microarchitecture (MICRO)*, pages 1–13. IEEE, 2016.
- [15] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Haichen Shen, Meghan Cowan, Leyuan Wang, Yuwei Hu, Luis Ceze, et al. {TVM}: An automated {End-to-End} optimizing compiler for deep learning. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 578–594, 2018.
- [16] Syed OwaisAli Chishti, Sana Riaz, Muhammad BilalZaib, and Mohammad Nauman. Self-driving cars using cnn and q-learning. In *2018 IEEE 21st International Multi-Topic Conference (INMIC)*, pages 1–7. IEEE, 2018.

- [17] Jack Choquette, Wishwesh Gandhi, Olivier Giroux, Nick Stam, and Ronny Krashinsky. Nvidia a100 tensor core gpu: Performance and innovation. *IEEE Micro*, 41(2):29–35, 2021.
- [18] Eric Chung, Jeremy Fowers, Kalin Ovtcharov, Michael Papamichael, Adrian Caulfield, Todd Massengill, Ming Liu, Daniel Lo, Shlomi Alkalay, Michael Haselman, Maleen Abeydeera, Logan Adams, Hari Angepat, Christian Boehn, Derek Chiou, Oren Firestein, Alessandro Forin, Kang Su Gatlin, Mahdi Ghandi, Stephen Heil, Kyle Holohan, Ahmad El Hussein, Tamas Juhasz, Kara Kagi, Ratna K. Kovvuri, Sitaram Lanka, Friedel van Megen, Dima Mukhortov, Prerak Patel, Brandon Perez, Amanda Rapsang, Steven Reinhardt, Bitu Rouhani, Adam Sapek, Raja Seera, Sangeetha Shekar, Balaji Sridharan, Gabriel Weisz, Lisa Woods, Phillip Yi Xiao, Dan Zhang, Ritchie Zhao, and Doug Burger. Serving dnns in real time at datacenter scale with project brainwave. *IEEE Micro*, 38(2):8–20, 2018.
- [19] André DeHon, Joshua Adams, Michael DeLorimier, Nachiket Kapre, Yuki Matsuda, Helia Naeimi, Michael Vanier, and Michael Wrighton. Design patterns for reconfigurable computing. In *12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 13–23. IEEE, 2004.
- [20] Danny Dolev, Dror G Feitelson, Joseph Y Halpern, Raz Kupferman, and Nathan Linial. No justified complaints: On fair sharing of multiple resources. In *proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, pages 68–75, 2012.
- [21] Fares Elsabbagh, Blaise Tine, Priyadarshini Roshan, Ethan Lyons, Euna Kim, Da Eun Shim, Lingjun Zhu, Sung Kyu Lim, et al. Vortex: Opencl compatible risc-v gpgpu. *arXiv preprint arXiv:2002.12151*, 2020.
- [22] Joel Emer, Mark D Hill, Yale N Patt, J Yi Joshua, Derek Chiou, and Resit Sendag. Single-threaded vs. multithreaded: Where should we focus? *IEEE Micro*, 27(6):14–24, 2007.
- [23] DENX Software Engineering. U-boot, 2022. <https://www.denx.de/wiki/U-Boot/>.

- [24] Hadi Esmaeilzadeh, Emily Blem, Renee St Amant, Karthikeyan Sankaralingam, and Doug Burger. Dark silicon and the end of multicore scaling. *IEEE micro*, 32(3):122–134, 2012.
- [25] Stijn Eyerman and Lieven Eeckhout. System-level performance metrics for multiprogram workloads. *IEEE Micro*, 28(3):42–53, 2008.
- [26] Brandon Fain, Kamesh Munagala, and Nisarg Shah. Fair allocation of indivisible public goods. In *Proceedings of the 2018 ACM Conference on Economics and Computation*, EC '18, page 575–592, New York, NY, USA, 2018. Association for Computing Machinery.
- [27] Jeremy Fowers, Kalin Ovtcharov, Michael Papamichael, Todd Massengill, Ming Liu, Daniel Lo, Shlomi Alkalay, Michael Haselman, Logan Adams, Mahdi Ghandi, et al. A configurable cloud-scale dnn processor for real-time ai. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pages 1–14. IEEE, 2018.
- [28] Rupert Freeman, Seyed Majid Zahedi, Vincent Conitzer, and Benjamin C Lee. Dynamic proportional sharing: A game-theoretic approach. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 2(1):1–36, 2018.
- [29] Ali Ghodsi, Vyas Sekar, Matei Zaharia, and Ion Stoica. Multi-resource fair queueing for packet processing. In *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*, pages 1–12, 2012.
- [30] Ali Ghodsi, Matei Zaharia, Benjamin Hindman, Andy Konwinski, Scott Shenker, and Ion Stoica. Dominant resource fairness: Fair allocation of multiple resource types. In *8th USENIX symposium on networked systems design and implementation (NSDI 11)*, 2011.
- [31] Ali Ghodsi, Matei Zaharia, Scott Shenker, and Ion Stoica. Choosy: Max-min fair sharing for datacenter jobs with constraints. In *Proceedings of the 8th ACM European Conference on Computer Systems*, pages 365–378, 2013.

- [32] John L Gustafson. Reevaluating amdahl’s law. *Communications of the ACM*, 31(5):532–533, 1988.
- [33] Robert J Halstead, Bharat Sukhwani, Hong Min, Mathew Thoennes, Parijat Dube, Sameh Asaad, and Balakrishna Iyer. Accelerating join operation for relational databases with fpgas. In *2013 IEEE 21st Annual International Symposium on Field-Programmable Custom Computing Machines*, pages 17–20. IEEE, 2013.
- [34] John Hauser. Softfloat. <http://HTTP.CS.Berkeley.EDU/~jhauser/arithmetric/softfloat.html>, 1997.
- [35] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.
- [36] Gary J Henry. The unix system: The fair share scheduler. *AT&T Bell Laboratories Technical Journal*, 63(8):1845–1857, 1984.
- [37] Don Lahiru Nirmal Hettiarachchi, Venkata Salini Priyamvada Davuluru, and Eric J Balster. Integer vs. floating-point processing on modern fpga technology. In *2020 10th Annual Computing and Communication Workshop and Conference (CCWC)*, pages 0606–0612. IEEE, 2020.
- [38] Mark D. Hill and Michael R. Marty. Amdahl’s law in the multicore era. *Computer*, 41(7):33–38, 2008.
- [39] Joost Hoozemans, Johan Peltenburg, Fabian Nonnemacher, Akos Hadnagy, Zaid Al-Ars, and H Peter Hofstee. Fpga acceleration for big data analytics: Challenges and opportunities. *IEEE Circuits and Systems Magazine*, 21(2):30–47, 2021.
- [40] Intel. the open programmable acceleration engine (opae), 2018. <https://01.org/opae>.
- [41] Intel, 2019. <https://www.intel.com/content/www/us/en/partner/showcase/offering/a5b3b000000TcqfAAC/de10pro-stratix-10-gxsx-accelerator.html>.

- [42] Intel. Intel fpga smartnic, 2023. <https://www.intel.com/content/www/us/en/products/details/fpga/platforms/smartnic.html>.
- [43] Pekka Jääskeläinen, Carlos Sánchez de La Lama, Erik Schnetter, Kalle Raiskila, Jarmo Takala, and Heikki Berg. pocl: A performance-portable opencl implementation. *International Journal of Parallel Programming*, 43:752–785, 2015.
- [44] Myeongjae Jeon, Shivaram Venkataraman, Amar Phanishayee, Junjie Qian, Wencong Xiao, and Fan Yang. Analysis of {Large-Scale}{Multi-Tenant}{GPU} clusters for {DNN} training workloads. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, pages 947–960, 2019.
- [45] EunYoung Jeong, Shinae Wood, Muhammad Jamshed, Haewon Jeong, Sunghwan Ihm, Dongsu Han, and KyoungSoo Park. {mTCP}: a highly scalable user-level {TCP} stack for multicore systems. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, pages 489–502, 2014.
- [46] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, and Jonathan Ross. In-datacenter performance analysis of a tensor processing unit. 2017.
- [47] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th annual international symposium on computer architecture*, pages 1–12, 2017.

- [48] Nachiket Kapre. Custom fpga-based soft-processors for sparse graph acceleration. In *2015 IEEE 26th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pages 9–16. IEEE, 2015.
- [49] Alan H. Karp and Horace P. Flatt. Measuring parallel processor performance. *Commun. ACM*, 33(5):539–543, may 1990.
- [50] Judy Kay and Piers Lauder. A fair share scheduler. *Communications of the ACM*, 31(1):44–55, 1988.
- [51] Ahmed Khawaja, Joshua Landgraf, Rohith Prakash, Michael Wei, Eric Schkufza, and Christopher J Rossbach. Sharing, protection, and compatibility for reconfigurable fabric with amorphos. In *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*, pages 107–127, 2018.
- [52] Martin F Krafft. *The Debian system: concepts and techniques*. No Starch Press, 2005.
- [53] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.
- [54] Chris Lattner, Mehdi Amini, Uday Bondhugula, Albert Cohen, Andy Davis, Jacques Pienaar, River Riddle, Tatiana Shpeisman, Nicolas Vasilache, and Oleksandr Zinenko. Mlir: Scaling compiler infrastructure for domain specific computation. In *2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, pages 2–14. IEEE, 2021.
- [55] Joo Hwan Lee, Hui Zhang, Veronica Lagrange, Praveen Krishnamoorthy, Xiaodong Zhao, and Yang Seok Ki. Smartssd: Fpga accelerated near-storage data analytics on ssd. *IEEE Computer architecture letters*, 19(2):110–113, 2020.
- [56] Topi Leppänen, Atro Lotvonen, Panagiotis Mousouliotis, Joonas Multanen, Georgios Keramidas, and Pekka Jääskeläinen. Efficient opencl system integration of non-blocking fpga accelerators. *Microprocessors and Microsystems*, 97:104772, 2023.

- [57] David Lewis, Gordon Chiu, Jeffrey Chromczak, David Galloway, Ben Gamsa, Valavan Manohararajah, Ian Milton, Tim Vanderhoek, and John Van Dyken. The stratix™ 10 highly pipelined fpga architecture. In *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 159–168, 2016.
- [58] Baolin Li, Viiay Gadepally, Siddharth Samsi, and Devesh Tiwari. Characterizing multi-instance gpu for machine learning workloads. In *2022 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 724–731. IEEE, 2022.
- [59] Ming Liang and Xiaolin Hu. Recurrent convolutional neural network for object recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3367–3375, 2015.
- [60] Wang Lie and Wu Feng-yan. Dynamic partial reconfiguration in fpgas. In *2009 Third International Symposium on Intelligent Information Technology Application*, volume 2, pages 445–448, 2009.
- [61] Jiaxin Lin, Kiran Patel, Brent E Stephens, Anirudh Sivaraman, and Aditya Akella. Panic: A high-performance programmable nic for multi-tenant networks. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 243–259, 2020.
- [62] Xinhan Lin, Shouyi Yin, Fengbin Tu, Leibo Liu, Xiangyu Li, and Shaojun Wei. Lcp: A layer clusters paralleling mapping method for accelerating inception and residual networks on fpga. In *Proceedings of the 55th Annual Design Automation Conference*, pages 1–6, 2018.
- [63] Igor L Markov. Limits on fundamental limits to computation. *Nature*, 512(7513):147–154, 2014.
- [64] Daniel McFadden. Constant elasticity of substitution production functions. *The Review of Economic Studies*, 30(2):73–83, 1963.

- [65] Michael Mitzenmacher. The power of two choices in randomized load balancing. *IEEE Transactions on Parallel and Distributed Systems*, 12(10):1094–1104, 2001.
- [66] Aaftab Munshi. The opencl specification. In *2009 IEEE Hot Chips 21 Symposium (HCS)*, pages 1–314. IEEE, 2009.
- [67] Netronome. Agilio cx smartnics, 2023. <https://www.netronome.com/products/agilio-cx/>.
- [68] NVIDIA. Nvidia mellanox innova-2 flex, 2023. <https://www.nvidia.com/en-us/networking/ethernet/innova-2-flex/>.
- [69] Zhenchao Ouyang, Jianwei Niu, Yu Liu, and Mohsen Guizani. Deep cnn-based real-time traffic light detector for self-driving vehicles. *IEEE transactions on Mobile Computing*, 19(2):300–313, 2019.
- [70] Chandandeep Singh Pabla. Completely fair scheduler. *Linux Journal*, 2009(184):4, 2009.
- [71] Ashish Panwar, Aravinda Prasad, and K Gopinath. Making huge pages actually useful. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 679–692, 2018.
- [72] Jason Jong Kyu Park, Yongjun Park, and Scott Mahlke. Chimera: Collaborative pre-emption for multitasking on a shared gpu. *ACM SIGARCH Computer Architecture News*, 43(1):593–606, 2015.
- [73] David C Parkes, Ariel D Procaccia, and Nisarg Shah. Beyond dominant resource fairness: Extensions, limitations, and indivisibilities. *ACM Transactions on Economics and Computation (TEAC)*, 3(1):1–22, 2015.
- [74] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

- [75] James Pita, Manish Jain, Fernando Ordóñez, Milind Tambe, Sarit Kraus, and Reuma Magori-Cohen. Effective solutions for real-world stackelberg games: When agents must deal with human uncertainties. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pages 369–376, 2009.
- [76] Marius-Constantin Popescu, Valentina E Balas, Liliana Perescu-Popescu, and Nikos Mastorakis. Multilayer perceptron and neural networks. *WSEAS Transactions on Circuits and Systems*, 8(7):579–588, 2009.
- [77] M. Quraishi, E. Tavakoli, and F. Ren. A survey of system architectures and techniques for fpga virtualization. *IEEE Transactions on Parallel & Distributed Systems*, 32(09):2216–2230, sep 2021.
- [78] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.
- [79] Tim Roughgarden. Algorithmic game theory. *Communications of the ACM*, 53(7):78–86, 2010.
- [80] Aaron Severance and Guy GF Lemieux. Embedded supercomputing in fpgas with the vectorblox mxp matrix processor. In *2013 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS)*, pages 1–10. IEEE, 2013.
- [81] Muhammad Shafique, Lars Bauer, Waheed Ahmed, and Jörg Henkel. Minority-game-based resource allocation for run-time reconfigurable multi-core processors. In *2011 Design, Automation & Test in Europe*, pages 1–6. IEEE, 2011.
- [82] Nisarg Shah. Reverting to simplicity in social choice. *The Future of Economic Design: The Continuing Development of a Field as Envisioned by Its Researchers*, pages 39–44, 2019.

- [83] Yongming Shen, Michael Ferdman, and Peter Milder. Overcoming resource underutilization in spatial cnn accelerators. In *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–4. IEEE, 2016.
- [84] Allan Snavely and Dean M Tullsen. Symbiotic jobscheduling for a simultaneous multithreaded processor. In *Proceedings of the ninth international conference on Architectural support for programming languages and operating systems*, pages 234–244, 2000.
- [85] Livio Soares and Michael Stumm. {FlexSC}: Flexible system call scheduling with {Exception-Less} system calls. In *9th USENIX Symposium on Operating Systems Design and Implementation (OSDI 10)*, 2010.
- [86] Bharat Sukhwani, Hong Min, Mathew Thoennes, Parijat Dube, Balakrishna Iyer, Bernard Brezzo, Donna Dillenberger, and Sameh Asaad. Database analytics acceleration using fpgas. In *Proceedings of the 21st international conference on Parallel architectures and compilation techniques*, pages 411–420, 2012.
- [87] Xian-He Sun and Yong Chen. Reevaluating amdahl’s law in the multicore era. *Journal of Parallel and distributed Computing*, 70(2):183–188, 2010.
- [88] Blaise Tine, Varun Saxena, Santosh Srivatsan, Joshua R Simpson, Fadi Alzammam, Liam Cooper, and Hyesoon Kim. Skybox: Open-source graphic rendering on programmable risc-v gpus. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, pages 616–630, 2023.
- [89] Blaise Tine, Krishna Praveen Yalamarthy, Fares Elsabbagh, and Kim Hyesoon. Vortex: Extending the risc-v isa for gpgpu and 3d-graphics. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 754–766, 2021.
- [90] John Von Neumann and Oskar Morgenstern. Theory of games and economic behavior, 2nd rev. 1947.

- [91] Carl A Waldspurger and William E Wehl. Lottery scheduling: Flexible proportional-share resource management. In *Proceedings of the 1st USENIX conference on Operating Systems Design and Implementation*, pages 1–es, 1994.
- [92] Carl A Waldspurger and William E Wehl. Stride scheduling: deterministic proportional-share resource management. 1995.
- [93] Kaibo Wang, Kai Zhang, Yuan Yuan, Siyuan Ma, Rubao Lee, Xiaoning Ding, and Xiaodong Zhang. Concurrent analytical query processing with gpus. *Proceedings of the VLDB Endowment*, 7(11):1011–1022, 2014.
- [94] Andrew Waterman, Yunsup Lee, Rimas Avizienis, David A Patterson, and Krste Asanovic. The risc-v instruction set manual volume ii: Privileged architecture version 1.7. *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2015-49*, 2015.
- [95] Andrew Waterman, Yunsup Lee, David A Patterson, and Krste Asanovic. The risc-v instruction set manual, volume i: User-level isa, version 2.0. *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2014-54*, 2014.
- [96] Xuechao Wei, Cody Hao Yu, Peng Zhang, Youxiang Chen, Yuxin Wang, Han Hu, Yun Liang, and Jason Cong. Automated systolic array architecture synthesis for high throughput cnn inference on fpgas. In *Proceedings of the 54th Annual Design Automation Conference 2017*, pages 1–6, 2017.
- [97] R Stanley Williams. What’s next?[the end of moore’s law]. *Computing in Science & Engineering*, 19(2):7–13, 2017.
- [98] Yuheng Wu, Bin Zheng, and Yongting Zhao. Dynamic gesture recognition based on lstm-cnn. In *2018 Chinese Automation Congress (CAC)*, pages 2446–2450. IEEE, 2018.
- [99] Wencong Xiao, Romil Bhardwaj, Ramachandran Ramjee, Muthian Sivathanu, Nipun Kwatra, Zhenhua Han, Pratyush Patel, Xuan Peng, Hanyu Zhao, Quanlu Zhang, et al. Gandiva: Introspective cluster scheduling for deep learning. In *13th USENIX*

- Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 595–610, 2018.
- [100] Ziyue Yang, James R Harris, Benjamin Walker, Daniel Verkamp, Changpeng Liu, Cunyin Chang, Gang Cao, Jonathan Stern, Vishal Verma, and Luse E Paul. Spdk: A development kit to build high performance storage applications. In *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 154–161. IEEE, 2017.
- [101] Tsung Tai Yeh, Amit Sabne, Putt Sakdhnagool, Rudolf Eigenmann, and Timothy G Rogers. Pagoda: Fine-grained gpu resource virtualization for narrow tasks. *ACM SIGPLAN Notices*, 52(8):221–234, 2017.
- [102] Gingfung Yeung, Damian Borowiec, Adrian Friday, Richard Harper, and Peter Garaghan. Towards {GPU} utilization prediction for cloud deep learning. In *12th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 20)*, 2020.
- [103] Seyed Majid Zahedi and Benjamin C Lee. Ref: Resource elasticity fairness with sharing incentives for multiprocessors. *ACM Sigplan Notices*, 49(4):145–160, 2014.
- [104] Seyed Majid Zahedi, Qiuyun Llull, and Benjamin C. Lee. Amdahl’s law in the datacenter era: A market for fair processor allocation. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 1–14, 2018.
- [105] Yue Zha and Jing Li. Virtualizing fpgas in the cloud. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS ’20*, page 845–858, New York, NY, USA, 2020. Association for Computing Machinery.
- [106] Chen Zhang, Peng Li, Guangyu Sun, Yijin Guan, Bingjun Xiao, and Jason Cong. Optimizing fpga-based accelerator design for deep convolutional neural networks. In *Proceedings of the 2015 ACM/SIGDA international symposium on field-programmable gate arrays*, pages 161–170, 2015.
- [107] Li Zhang. Proportional response dynamics in the fisher market. *Theoretical Computer Science*, 412(24):2691–2698, 2011.

- [108] Bin Zhao, Xuelong Li, Xiaoqiang Lu, and Zhigang Wang. A cnn-rnn architecture for multi-label weather recognition. *Neurocomputing*, 322:47–57, 2018.
- [109] Shijie Zhou and Viktor K Prasanna. Accelerating graph analytics on cpu-fpga heterogeneous platform. In *2017 29th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, pages 137–144. IEEE, 2017.
- [110] Zhe Zhou, Yanxiang Bi, Junpeng Wan, Yangfan Zhou, and Zhou Li. Userspace bypass: Accelerating syscall-intensive applications. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*, pages 33–49, 2023.
- [111] Noa Zilberman, Yury Audzevich, Georgina Kalogeridou, Neelakandan Manihatty-Bojan, Jingyun Zhang, and Andrew Moore. Netfpga: Rapid prototyping of networking devices in open source. *ACM SIGCOMM Computer Communication Review*, 45(4):363–364, 2015.