Elastic Multi-Resource Fairness: Balancing Fairness and Efficiency in Coupled CPU-GPU Architectures

Shanjiang Tang[1], BingSheng He[2], Shuhao Zhang[2,4], Zhaojie Niu[3]
[1]School of Computer Science & Technology, Tianjin University
[2]School of Computing, National University of Singapore
[3]Interdisciplinary Graduate School, Nanyang Technological University
[4]SAP Research & Innovation, Singapore
[1]tashj@tju.edu.cn, [2]hebs@comp.nus.edu.sg, {[2]tonyzhang19900609,[3]nzjemail}@gmail.com

***Abstract*—Fairness and efficiency are two important concerns for users in a shared computer system, and there tends to be a tradeoff between them. Heterogeneous computing poses new challenging issues on the fair allocation of computational resources among users due to the availability of different kinds of computing devices (e.g., CPU and GPU). Prior work either considers the fair resource allocation separately for each computing device or is unable to balance flexibly the tradeoff between the fairness and system utilization.**

In this work, we consider an emerging heterogeneous computing system with coupled CPU and GPU into a single chip. We first show that it is essential to have a new fair policy for coupled CPU-GPU architectures that is capable of considering both the CPU and the GPU as a whole in fair resource allocation and being aware of the system utilization maximization. We then propose a fair policy called Elastic Multi-Resource Fairness (EMRF) for coupled CPU-GPU architectures, by modeling CPU and GPU as two resource types and viewing the resource fairness problem as a multi-resource fairness problem. It extends DRF by adding a knob that allows users to tune and balance fairness and performance flexibly, and considers the fair allocation of computational resources as a whole for CPU and GPU devices. We show that EMRF satisfies fairness properties of *sharing incentive*, *envy-freeness* and *pareto efficiency*. Finally, we evaluate EMRF using real experiments, and the results show that EMRF can achieve better performance and fairness.

***Index Terms*—Coupled CPU-GPU Architecture, Fairness, Performance, EMRF, APU.**

I. Introduction

The recent advancement of accelerator technologies (e.g., GPU, FPGA and DSP) has made heterogeneous computing become common and a trend for High Performance Computing (HPC) systems [31]. By removing the PCI-e bus, the coupled CPU-GPU architecture (CCGA) [4], which integrates CPU and GPU into a single chip, is a new emerging heterogeneous computing system that aims to achieve extreme-scale, cost-effective, and power-efficient high performance computing.

Sharing computing devices (e.g., CPU and GPU) is essential to achieve high resource utilization and cost efficiency for a heterogeneous computing system. First, it generally involves a high degree of parallelism for a single GPU that consists of many computing cores, and individual users often cannot fully utilize it exclusively [26]. Second, even for a single user, the resource demand of its workload is varying over time, implying that it is difficult to keep the high utilization all the time. Resource sharing can address this problem by allowing multiple users concurrently running their workloads in the system so that overloaded users can possess unused resources from underloaded users. Hence, obtaining high utilization by resource sharing is also an effective manner for high cost efficiency.

GFLOPS (Giga FLoating-point Operations Per Second) is a key metric widely used to measure and compare the capacities for different computing devices. However, there are several challenges for CCGA in providing performance isolation and Quality of Service (QoS) guarantees. Due to the significant architectural differences between CPU and GPU cores, it is not suitable to simply consider CCGA as a black box with a certain capacity of aggregate GFLOPS. The system performance is essentially associated with the relative frequencies of float-point operations at which the applications invoke with different kinds of computing devices. Moreover, the performance is also related to how CPU and GPU capacities are divided across different applications.

In addition to system efficiency (i.e., performance), fairness is also a major concern for users in the *shared* heterogeneous computing environment. Previous studies have shown that there is a general tradeoff between the fairness and system efficiency in resource allocation [20]. Strictly keeping the fairness among users can result in allocations with low system efficiency. On the contrary, pursing for a high system efficiency is often at the expense of compromised fairness.

One of the most popular fair policies is *proportional resource sharing*, which allocates resources to users in proportion to their *weights* [34]. It has been widely applied to a variety of computer components such as storage systems [37] and network link bandwidth [13]. When it comes to the heterogeneous computing, although there is some work for CPU only [33] and GPU only [7], [21], all of them consider the fair resource allocation separately for each computing device. For the computation in the CCGA, a user's workload is computed on both computing devices simultaneously. From a user's perspective, it is most likely that the user mainly concerns with the final GFLOPS allocation and performance improvement entirely received from the system, rather than the result of *separate* resource allocation and performance improvement from each computing device. It means that we

SC16; Salt Lake City, Utah, USA; November 2016
978-1-4673-8815-3/16/$31.00

should *not* consider them separately as the single resource fair allocation for each computing device. Instead, we should consider CPU and GPU of CCGA as a whole in resource allocation. Typically, we show that it can be modeled as a multi-resource fair allocation problem for CPU and GPU allocation in the CCGA.

In multi-resource allocation, Dominant Resource Fairness (DRF) is one of the most popular fair allocation policies [15]. It achieves fairness by equalizing the share of each user's dominant resource, referring to as the resource that is most heavily used (as a fraction of its capacity) by a user. Although there are a number of extensions [8], [23], [25], [36] (See Section VII), the impact on the system efficiency receives little attention. Many recent studies have shown that there tends to be a tradeoff between fairness and efficiency in multi-resource allocation [20]. DRF and its extensions are prone to over-constrain the system for meeting rigid fairness requirements, resulting in allocations with poor system efficiency.

In this paper, we propose Elastic Multi-Resource Fairness (EMRF), an elastic fairness-efficiency allocation policy, to be aware of the tradeoff between fairness and efficiency for CCGA in resource allocation. It allows users to balance fairness and system efficiency flexibly in the CCGA with a knob argument in the range of $[0, 1]$. EMRF then improves the system efficiency while guaranteeing the QoS of δ-fairness with the user's setting of knob, where δ is the maximum difference of GFLOPS allocations between any two users in the CCGA. To the best of our knowledge, EMRF is the first fair policy that integrates CPU and GPU as a whole in resource allocation for CCGA. We show that EMRF can ensure that each user in the CCGA can at least get the amount of resources as that under the exclusively partitioned non-sharing environment ("Sharing Incentive"). It can also ensure that every user prefers to its own allocation and no user envies the allocation of others ("Envy-freeness"). Moreover, EMRF enables the system fully utilized by ensuring that it is impossible for a user to get more resources without decreasing the resource of at least one user ("Pareto Efficiency"). We evaluate EMRF with testbed experiments in an AMD APU platform. The experimental results show that our approach is highly elastic and can achieve high efficiency and fairness in the CCGA. We conjecture that the results and findings in this paper can be extended to supercomputers and clusters.

Organization. Section II overviews the background of CCGA and describes a computing model on the CCGA. Section III motivates our work by showing the difficulties of achieving both fairness and efficiency in the CCGA, followed by formal definitions of several desirable fairness properties in Section IV. We describe and analyze our model and approach in Section V. The experimental evaluation is given in Section VI. We review the related work in Section VII. Finally, Section VIII concludes the paper.

II. Background and Preliminary

This section starts by reviewing Coupled CPU-GPU Architecture (CCGA), and then introduces a heterogeneous computing model for CCGA.

A. Coupled CPU-GPU Architectures (CCGA)

In traditional *discrete* CPU-GPU heterogeneous computing system, the GPU has separated memory space from the CPU, and the data transfer between the CPU and the GPU is achieved via a connection bus (e.g., PCIe bus). It tends to incur large communication overheads between CPU and GPU, and meanwhile requires a lot of programming effort, since the data that is manipulated by both the CPU and the GPU have to be explicitly managed by the programmer carefully [24]. To resolve these limitations, Coupled CPU-GPU Architectures (CCGA) has emerged, which integrates the CPU and the GPU into a single chip and allows the CPU and the GPU to communicate with each other through the shared physical memory by featuring shared memory space between them. This reduces the data transfer time significantly, especially for those applications that need large communication between the CPU and the GPU, since it enables the CPU and the GPU to transfer their data in the shared physical memory only without going through the connection bus and allows them to have *zero-copy* for the shared data [4]. Having these merits, CCGA is a trend for future heterogeneous computing system and many such processors have been produced, including AMD APU [10], Ivy Bridge [12] and Intel Sandy Bridge [39].

To better support the programming for CPU-GPU heterogeneous computing, OpenCL [16] is proposed by industry and becomes popular in recent years, as a unified programming model for both CPU and GPU computing units. It enables applications written in OpenCL to run on either CPU or GPU devices, and also allows multiple applications to share and execute on each device at the same time [38]. Due to the limited chip area, the GPU in the CCGA is usually less powerful than the high-end GPU in discrete architectures. Thus, the GPU in the CCGA usually cannot deliver a dominant performance speedup as it does in discrete architectures. An application must carefully assign workloads to both processors and keep them busy for the maximized speedup (named as *co-run computation*) [11], [17], [18], [41].

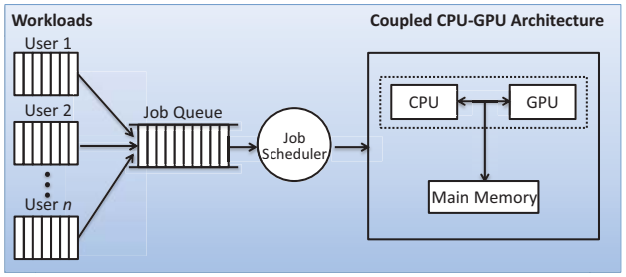
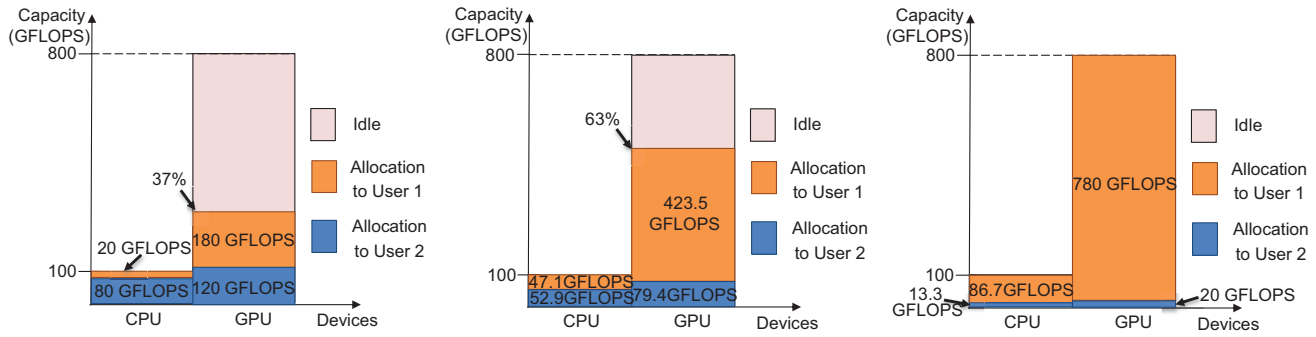


Fig. 1: The heterogeneous computing model on the CCGA.

B. Heterogeneous Computing Model

Figure 1 illustrates the heterogeneous computing model for CCGA. We consider the APU, of which the CPU and the GPU are coupled in the same chip and all data accesses go through a unified north bridge (UNB) that connects the CPU, the GPU, and the main memory. In this study, we limit our focus



(a) Proportional resource allocation result under 1:1 resource allocation ratio, with the GPU utilization of 37%.

(b) The DRF allocation with the GPU utilization of 63%. The allocation ratio is 32 : 9.

(c) Both CPU and GPU are 100% utilized, given the allocation ratio of 26 : 1.

Fig. 2: Allocations in the CCGA of Example 1. The capacities of CPU and GPU are 100 and 800, respectively. The CPU-to-GPU workload ratios for User 1 and User 2 are $1/9$ and $4/6$, respectively.

on applications written in OpenCL, for which the CPU/GPU computing dominants the performance and the impact of data transfer is negligible.

Given GFLOPS as a widely used metric in HPC for measuring and comparing the computing resource capacities across different computing devices, we use it in our following resource allocation to represent the amount of allocated resources. Let C_{cpu} and C_{gpu} represent the computing capacity (measured in GFLOPS) of CPU and GPU, respectively. Those capacities represent the total amount of resources that can be allocated to users in the CCGA. Each user has a sequence of jobs to compute. To fully utilize CCGA, the tasks of a job can be assigned and execute on both CPU and GPU devices concurrently (i.e., *co-run* computation). Let ξ_i^{cpu} and ξ_i^{gpu} denote the fraction of the workload of user i dispatched to CPU and GPU, respectively. By definition, it has $\xi_i^{gpu} = 1 - \xi_i^{cpu}$. The ratio of the workload allocated to CPU to that of GPU for user i is defined as *CPU-to-GPU workload ratio*, denoted by ρ_i (i.e., $\rho_i = \xi_i^{cpu}/\xi_i^{gpu}$). It is generally different for different applications.

Suppose that there are n users and a job queue that receives arriving jobs from each user and then dispatches them to the internal scheduling buffer by a job scheduler. The job scheduler maintains a certain number of submitted jobs in its internal scheduling buffer. It monitors the number of operations performed in both CPU and GPU for each job, and dynamically assigns *weights* to users in accordance with the measured allocation ratios. Based on these weights, the scheduler dynamically controls the resource allocations of CPU and GPU to each user so that the computed number of operations of a user is proportional to its weight. The weights are computed according to the fairness policies of EMRF for maximizing system utilization. Particularly, we consider the resource sharing across multiple users/jobs in the GFLOPS dimension instead of the time dimension, since the current GPU kernel written in OpenCL is non-interruptible, i.e., it cannot be stopped once it is submitted, which makes it impossible for time division sharing approach.

Let F_i^{cpu} and F_i^{gpu} denote the amount of resources in GFLOPS that user i received from the CPU and the GPU, respectively. Then the total number of allocated resources F_i in GFLOPS for user i in the CCGA is $F_i = F_i^{cpu} + F_i^{gpu}$. For the sake of load balancing computation between the CPU and the GPU for a user i's workload, we make the ratio of the allocated resources of CPU in GFLOPS to that of GPU equal to its CPU-to-GPU workload ratio ρ_i, i.e., $F_i^{cpu}/F_i^{gpu} = \rho_i$. By learning from the traditional multi-resource allocation problem where there is a fixed ratio among different resource types for a task resource demand (e.g., CPU and memory) [15], we can model CPU and GPU allocation as a multi-resource allocation problem (namely, *CPU-GPU multi-resource allocation* problem). Our allocation in the following sections is based on this.

III. Motivations

In the *single-resource* allocation, the fairness is achieved among users through dividing the system resources to users in proportion to their assigned weights. For example, if there is a CPU of 200 GFLOPS capacity shared equally by two users, the system will allocate 100 GFLOPS to each user. Many existing fair schedulers like Weighted Fair Queuing [13] and Proportional Resource Sharing [34] are work-conserving schedulers, which can ensure that the system is 100% fully utilized whenever there are pending workloads in the system. It implies that the system utilization is not a problem in the single-resource fairness.

However, when it comes to the CCGA, the above claim for system utilization might no longer hold. Fairness in sharing CCGA is relatively more complex and challenging, since the loads of users' applications can be scheduled to both CPU and GPU devices for concurrent execution in the CCGA. The resource utilization of computing devices highly depends on their CPU-to-GPU workload ratios and the allocation ratios (the relative fraction of system capacity in GFLOPS allocated to users) in the CCGA. If the users with high GPU loads are assigned with relatively small allocation ratios, it may result in the low utilization for the GPU device due to insufficient resource requests. Similar case does also hold for the CPU device. On the contrary, keeping high usage of both CPU and

GPU devices generally causes the unfairness problem, since it may need to adjust the allocations in a manner that just starves some users.

To demonstrate these points, let's consider the following examples for proportional resource sharing and DRF allocations.

Example 1. *Consider a CCGA with the capacities of CPU and GPU to be* 100 *GFLOPS and* 800 *GFLOPS, respectively. It is shared by users* 1 *and* 2 *equally with CPU-to-GPU workload ratios of* $\rho_1 = 1/9$ *and* $\rho_2 = 4/6$*, respectively.*

Proportional Resource Sharing Allocation. We start with the basic policy for a single resource type [34]. Figure 2(a) illustrates the allocation results for Example 1 by using proportional resource sharing. The two users receive the same number of GFLOPS under the proportional resource sharing allocation with the allocation ratio of $1 : 1$. There is an allocation of 200 GFLOPS to each user. Specifically, user 1 receives 20 GFLOPS from the CPU device and 180 GFLOPS from the GPU device. User 2 obtains 80 GFLOPS and 120 GFLOPS from the CPU and GPU devices, respectively. The CPU device is 100% fully utilized, whereas the utilization of GPU device is only 37% due to the relatively small allocation ratio for user 1.

DRF Allocation. In Example 1, the dominant resource of user 1 is the GPU device ($0.1 \cdot F_1/100 < 0.9 \cdot F_1/800$), whereas the dominant resource of user 2 is the CPU device ($0.4 \cdot F_2/100 > 0.6 \cdot F_2/800$). According to DRF, the fairness is achieved by equalizing the dominant shares for user 1 and 2, i.e., $0.9 \cdot F_1/800 = 0.4 \cdot F_2/100$. As shown in Figure 2(b), the resulting allocations for user 1 and 2 are $F_1 = 470.6$ GFLOPS and $F_2 = 132.3$ GFLOPS, respectively. Still, the GPU device is underutilized, with approximately 63% utilization only.

The above allocation policies just consider the issue of how to achieve some measure of fairness by setting users' allocation ratios, but do not deal with the impact of how such settings on system utilization. If we want to increase the system utilization, it needs to adjust the allocations of the users. As shown in Figure 2(c), both CPU and GPU devices can be fully utilized if the system allocates 866.7 GFLOPS to user 1 (86.7 from the CPU and 780 from the GPU), and 33.3 GFLOPS to user 2 (13.3 from the CPU and 20 from the GPU). In such 100% utilization case, it requires a $26 : 1$ allocation ratio (i.e., giving more allocation ratio to user 1), and increases user 1's resources from 470.6 (Figure 2 (b)) to 866.7 (Figure 2 (c)) GFLOPS whereas decreasing user 2's resources from 132.3 to 33.3 GFLOPS (being *unfair* for user 2).

Moreover, the system utilization also highly relies on users' competing workloads, and can be even worse under proportional resource sharing and DRF policies with more users joining in the system. To illustrate it, we modify Example 1 by adding a third user (with $\rho_3 = 5/5$) to the system, as shown by Example 2 below,

Example 2. *We extend Example 1 by adding user 3 with* $\rho_3 = 5/5$.

With proportional resource sharing policy, each user in Example 2 receives 100 GFLOPS (10 from the CPU and 90 from the GPU for user 1; 40 from the CPU and 60 from the GPU for user 2; 50 from the CPU and 50 from the GPU for user 3) with $1 : 1 : 1$ allocation ratio. It makes user 1 with *CPU-to-GPU workload ratio* of $\rho_3 = 1/9$ being seriously constrained by the allocation ratio, and reduces the GPU utilization from 37% (Figure 2(a)) to 25%. If using DRF policy, the system would allocate 303.8, 86.5 and 69.2 GFLOPS to users 1, 2 and 3 respectively with the allocation ratio of $160 : 45 : 36$, and the GPU is 44.93% utilized. In contrast, when changing the allocation ratio to $87 : 2 : 1$, both devices are 100% utilized, with allocations of 870, 20 and 10 GFLOPS for users 1, 2, and 3, respectively.

Summary. Through the examples, we have the following observations. First, pursuing 100% fairness is prone to result in poor resource utilization, and reversely achieving for high resource utilization is generally at the expense of fairness. That is, there tends to be a tradeoff between fairness and efficiency in resource allocation for users in the CCGA. Second, it can be more serious for the tradeoff problem between fairness and efficiency when there are more users in the system.

IV. Fair Sharing Properties in CCGA

From the economic point of view, a *good* allocation policy for fair sharing in the CCGA should satisfy several game theoretic properties, including sharing incentive, envy-freeness, and pareto efficiency [15], [35]. We re-define those properties for CCGA.

A. Sharing Incentive (SI)

Resource sharing is an effective approach to improve the system utilization and efficiency by allowing overloaded users to possess the unused resources from underloaded users [29], [30], [28]. To enable resource sharing among users sustainably, an allocation policy should satisfy *sharing incentive (SI)*, which ensures that each user in the shared system can get at least the resources it would get under a statically equal division of the computing system. Otherwise, users would prefer to split the computing system equally and use their own partitions exclusively without sharing.

The GFLOPS obtained by user i in the CCGA can be represented by the vector $\mathbf{F}_i = \langle F_i^{cpu}, F_i^{gpu} \rangle$. Moreover, let $\overline{F_i^{cpu}}$ and $\overline{F_i^{gpu}}$ represent the received resources in GFLOPS for user i from CPU and GPU devices under its own non-sharing partition of CCGA, respectively. Formally, an allocation policy satisfies SI if the following holds for each user $i \in [1, n]$,

$$\mathbf{F}_i \succeq \overline{\mathbf{F}}_i{}^{1}. \tag{1}$$

where $\overline{\mathbf{F}}_i = \langle \overline{F_i^{cpu}}, \overline{F_i^{gpu}} \rangle$ denotes the GFLOPS received for user i under its own non-sharing partition of CCGA.

B. Envy-freeness (EF)

Envy-freeness (EF) is another important criterion for measuring the fairness of an allocation policy. It means that no user envies the allocation of any other users. That is, each

[1]For any two vectors $\mathbf{a}$ and $\mathbf{b}$, we say $\mathbf{a} \succeq \mathbf{b}$ iff $a_i \geqslant b_i$ for $\forall i$.

user prefers its own allocation to the allocation of any other user. To achieve EF, we should ensure that each user cannot increase its resource allocation by exchanging its allocation with any other user.

Formally, let $\mu_i(\mathbf{F}_i)$ be the resources in GFLOPS achieved by user i under the vector $\mathbf{F}_i$. Then EF is guaranteed for an allocation policy if

$$\mu_i(\mathbf{F}_i) \geqslant \mu_i(\mathbf{F}_j), (\forall j \in [1,n] \wedge i \neq j). \tag{2}$$

for any two users $i, j \in [1, n]$. Formula (2) is an indication for EF that every user i works the best under its own resource allocation vector compared to using all other users.

C. Pareto Efficiency (PE)

Pareto efficiency (PE) is essential for allocation efficiency and high resource utilization. An allocation policy is PE if it is not feasible for a user to increase its resource allocation without decreasing the resource allocation of at least one other user. That is, there is no other feasible allocation for which at least one user is strictly better off and all other users are at least as well off.

Formally, let $\mathbf{F} = \langle \mathbf{F}_1, \mathbf{F}_2, ..., \mathbf{F}_n \rangle$ be the allocation result for n users generated by an allocation policy. The policy is PE only when it is true that, for any feasible allocation $\tilde{\mathbf{F}} = \langle \tilde{\mathbf{F}}_1, \tilde{\mathbf{F}}_2, ..., \tilde{\mathbf{F}}_n \rangle$, if $\mu_i(\tilde{\mathbf{F}}_i) > \mu_i(\mathbf{F}_i)$ for some user i, there must exist a user j satisfying $\mu_j(\tilde{\mathbf{F}}_j) < \mu_j(\mathbf{F}_j)$.

V. Elastic Multi-Resource Fairness

In this section, we describe our resource allocation model called *Elastic Multi-Resource Fairness(EMRF)*, which is an elastic fairness-efficiency model that can balance the tradeoff between fairness and allocation efficiency flexibly as needed. We analyze EMRF and show that it meets all the desirable properties (i.e., SI, EF, PE) shown in Section IV.

A. Allocation Model

We first define some terms used in our model. The *fair share* of a user is defined as the resources (GFLOPS) it receives if each of the resources is divided equally among all the users. Let's denote the fair share of user i by S_i. Denote the weight of user i in the CCGA as w_i. Then the total amount of CPU and GPU resources for user i after equal partition are $C_{cpu} \cdot \frac{w_i}{\sum_{k=1}^{n} w_k}$ and $C_{gpu} \cdot \frac{w_i}{\sum_{k=1}^{n} w_k}$, respectively. We can compute fair share with the progressive filling method [15], which increases all users' shares at the same rate, until at least one resource is saturated. That is, for user i, it must have $\max\{\frac{S_i \cdot \xi_i^{cpu}}{C_{cpu} \cdot \frac{w_i}{\sum_{k=1}^{n} w_k}}, \frac{S_i \cdot \xi_i^{gpu}}{C_{gpu} \cdot \frac{w_i}{\sum_{k=1}^{n} w_k}}\} = 1$. Hence,

$$S_i = \frac{1}{\max\{\frac{\xi_i^{cpu}}{C_{cpu} \cdot \frac{w_i}{\sum_{k=1}^{n} w_k}}, \frac{\xi_i^{gpu}}{C_{gpu} \cdot \frac{w_i}{\sum_{k=1}^{n} w_k}}\}}.$$

Let $\gamma_i^{cpu} = \frac{\xi_i^{cpu}}{C_{cpu}}$ and $\gamma_i^{gpu} = \frac{\xi_i^{gpu}}{C_{gpu}}$, then we can rewrite the above formula as

$$S_i = \frac{w_i}{\sum_{k=1}^{n} w_k \cdot \max\{\gamma_i^{cpu}, \gamma_i^{gpu}\}}. \tag{3}$$

Moreover, we call an allocation *fair* when the resources received by each user $i \in [1, n]$ in the shared CCGA is proportional to its own fair share. That is, the fairness is achieved if the following proposition is true,

$$\frac{F_i}{S_i} = \frac{F_j}{S_j}, \forall i, j \in [1, n]. \tag{4}$$

Let f_i denote the share of dominant (bottleneck) resources received by user i in the CCGA. The allocation shares in CPU and GPU for user i are $\frac{F_i \cdot \xi_i^{cpu}}{C_{cpu}}$ and $\frac{F_i \cdot \xi_i^{cpu}}{C_{cpu}}$, respectively. By definition, we have

$$f_i = \max\{\frac{F_i \cdot \xi_i^{cpu}}{C_{cpu}}, \frac{F_i \cdot \xi_i^{gpu}}{C_{gpu}}\} = F_i \cdot \max\{\gamma_i^{cpu}, \gamma_i^{gpu}\}. \tag{5}$$

According to Formula (3), we can deduce that,

Lemma 1. *$\forall i, j \in [1, n]$, if $\frac{F_i}{S_i} = \frac{F_j}{S_j}$, there must be $\frac{f_i}{w_i} = \frac{f_j}{w_j}$, and vice versa.*

Proof: According to Formula (5), $\frac{f_i}{w_i} = \frac{f_j}{w_j} \Leftrightarrow \frac{F_i \cdot \max\{\gamma_i^{cpu}, \gamma_i^{gpu}\}}{\frac{w_i}{w_j}} = \frac{F_j \cdot \max\{\gamma_j^{cpu}, \gamma_j^{gpu}\}}{w_j} \Leftrightarrow \frac{F_i \cdot \frac{w_i}{\sum_{k=1}^{n} w_k \cdot S_i}}{w_i} = \frac{F_j \cdot \frac{w_j}{\sum_{k=1}^{n} w_k \cdot S_j}}{w_j} \Leftrightarrow \frac{F_i}{S_i} = \frac{F_j}{S_j}$. ∎

By combining Formula (4) and Lemma 1, it provides us a key information that achieving the fairness of the overall resources for users in the CCGA is equivalent to guaranteeing the fairness on their (weighted) dominant resource share. Thus, the resource fairness problem can be converted to the dominant resource fairness problem, which can be addressed with Dominant Resource Fairness (DRF) [15] by ensuring that,

$$\frac{f_1}{w_1} = \frac{f_2}{w_2} = \cdots = \frac{f_n}{w_n}. \tag{6}$$

Formula (6) shows us that there is a proportional relationship between a user's dominant share and its received resources. Let f_i^{max} and F_i^{max} denote the maximum share of dominant resource and the corresponding overall resource for user i under the DRF allocation. With the progressive filling approach, the allocation of DRF terminates only when there is at least one resource saturated (e.g., CPU resource is saturated in Figure 2(b)). In that case, we cannot further increase each user's dominant resource, i.e., the dominant resource share and the overall resource are maximized for each user. We thus have,

$$\max\{\sum_{i=1}^{n} F_i \gamma_i^{cpu}, \sum_{i=1}^{n} F_i \gamma_i^{gpu}\} = 1. \tag{7}$$

We can get F_i^{max} by resolving F_i with Formula (5) (6) (7), i.e.,

$$F_i^{max} = \frac{w_i}{\max\{\sum_{k=1}^{n} \frac{w_k \gamma_k^{cpu}}{\max\{\gamma_k^{cpu}, \gamma_k^{gpu}\}}, \sum_{k=1}^{n} \frac{w_k \gamma_k^{gpu}}{\max\{\gamma_k^{cpu}, \gamma_k^{gpu}\}}\}} \cdot \frac{1}{\max\{\gamma_i^{cpu}, \gamma_i^{gpu}\}}. \tag{8}$$

However, DRF only seeks for 100% fairness in resource allocation without considering its impact on resource utilization, making its allocation efficiency tend to be poor. For example, as illustrated in Figure 2(b), the resource utilization of GPU device under DRF allocation is only 63%. On the contrary, pursuing for high resource utilization could also result in *poor* fairness. Figure 2(c) shows an allocation of 100% high resource utilization for both CPU and GPU devices, but its fairness is much poor since the share of dominant resource

for user 1 is $780/800 = 97.5\%$ whereas for user 2 is only $13.3/100 = 13.3\%$. It implies that there tends to be a tradeoff between the fairness and allocation efficiency in multi-resource allocation [20].

B. EMRF Allocation Policy

We propose a fairness policy called *Elastic Multi-Resource Fairness (EMRF)* that allows users to flexibly balance fairness and allocation efficiency in multi-resource allocation. The basic idea is that, instead of strictly keeping fairness as DRF does in multi-resource allocation, we trade fairness for increasing allocation efficiency by allowing some degree of unfairness. Here, we define two terms: *hard fairness* and *soft fairness*. The *hard fairness* refers to that all users get equal share in resource allocation, i.e., it requires that Formula (4) must be strictly satisfied. In contrast, the *soft fairness* allows some degree of $\delta(\delta \geqslant 0)$ unfairness in resource allocation among users. Formally, we define δ-*fairness* by modifying Formula (4) as

$$|\frac{F_i}{S_i} - \frac{F_j}{S_j}| \leqslant \delta, \forall i, j \in [1, n]. \tag{9}$$

In summary, DRF returns the allocation of *hard* fairness, whereas EMRF is aware of fairness-efficiency tradeoff and considers *soft* fairness so as to leave some optimization space for allocation efficiency.

Design of EMRF Policy. A tradeoff balancing allocation can be viewed as a combination of *fairness-oriented* allocation (i.e., purely for fairness guarantee) and *efficiency-oriented* allocation (i.e., purely for efficiency optimization). In EMRF, we provide a knob $\eta(0 \leqslant \eta \leqslant 1)$ for users to control and balance such two allocations flexibly. The EMRF first performs the fairness-oriented allocation for soft fairness guarantee. After that, it makes the efficiency-oriented allocation for efficiency optimization with the remaining idle resources across users.

In the fairness-oriented allocation, instead of achieving the hard fairness of F_i^{max}, we guarantee the soft fairness of $F_i^{max}\eta$ for each user i by using DRF. After the fairness-oriented allocation, the system remains $\mathbf{C}^{'} = \langle C_{cpu}^{'}, C_{gpu}^{'} \rangle$ idle resources for the successive efficiency-oriented allocation, where $C_{cpu}^{'} = C_{cpu} - \sum_{k=1}^{n} F_k^{max}\xi_k^{cpu}\eta$ and $C_{gpu}^{'} = C_{gpu} - \sum_{k=1}^{n} F_k^{max}\xi_k^{gpu}\eta$ for users. Let $F_i^{'}$ denote the resource received by user i under the efficiency-oriented allocation. Then we have

$$F_i = F_i^{max}\eta + F_i^{'}. \tag{10}$$

for each user i. The fairness-oriented allocation becomes dominant (i.e., benefit for fairness optimization) in EMRF allocation when η is large. On the contrary, the small value of η benefits more for efficiency optimization.

In the following, we first show that EMRF is a $\delta(\delta \geqslant 0)$-fairness, determined by the knob η. Next we introduce the efficiency-oriented allocation of EMRF.

Theorem 1. *EMRF is a δ-fairness policy where*

$$\delta = \max_{1 \leqslant i \leqslant n} \{ \frac{\max\{\gamma_i^{cpu}, \gamma_i^{gpu}\}}{\frac{w_i}{\sum_{k=1}^{n} w_k} \cdot \max\{\frac{\xi_i^{cpu}}{C_{cpu}^{'}}, \frac{\xi_i^{gpu}}{C_{gpu}^{'}}\}} \}.$$

Proof: We start by soft fairness definition. For any two users $i, j \in [1, n]$,

$$\begin{aligned} |\frac{F_i}{S_i} - \frac{F_j}{S_j}| &\leqslant \max_{\forall i,j \in [1,n]} \{|\frac{F_i}{S_i} - \frac{F_j}{S_j}|\} \\ &= \max_{\forall i,j \in [1,n]} \{|\frac{(F_i^{max}\eta + F_i^{'})}{S_i} - \frac{(F_j^{max}\eta + F_j^{'})}{S_j}|\} \\ &= \max_{\forall i,j \in [1,n]} \{|(\frac{F_i^{'}}{S_i} - \frac{F_j^{'}}{S_j}) + (\frac{F_i^{max}}{S_i} - \frac{F_j^{max}}{S_j})\eta|\}. \end{aligned} \tag{11}$$

According to Formula (6) and Lemma 1, we have $\frac{F_i^{max}}{S_i} = \frac{F_j^{max}}{S_j}$. Thus, we have

$$\max_{\forall i,j \in [1,n]} \{|\frac{F_i}{S_i} - \frac{F_j}{S_j}|\} = \max_{\forall i,j \in [1,n]} \{|\frac{F_i^{'}}{S_i} - \frac{F_j^{'}}{S_j}|\} = \max_{1 \leqslant i \leqslant n} \{\frac{F_i^{'}}{S_i}\} - \min_{1 \leqslant j \leqslant n} \{\frac{F_j^{'}}{S_j}\}. \tag{12}$$

Now our proof turns to be finding an upper bound δ satisfying that $\max_{1 \leqslant i \leqslant n}\{\frac{F_i^{'}}{S_i}\} - \min_{1 \leqslant j \leqslant n}\{\frac{F_j^{'}}{S_j}\} \leqslant \delta$ for all feasible allocations given the idle resource vector of $\mathbf{C}^{'}$.

For any feasible allocation $\langle F_1^{'}, F_2^{'}, \cdots, F_n^{'} \rangle$, its allocation stops when at least one resource saturated, i.e.,

$$\max\{\frac{\sum_{i=1}^{n} F_i^{'}\xi_i^{cpu}}{C_{cpu}^{'}}, \frac{\sum_{i=1}^{n} F_i^{'}\xi_i^{gpu}}{C_{gpu}^{'}}\} = 1. \tag{13}$$

Moreover, for all feasible allocations, the maximum value of $F_i^{'}$ for user i occurs when it exclusively possesses all the idle resource $\mathbf{C}^{'}$. In that case, there is no resource allocation for other users, i.e., $\forall j \in [1, n], F_j^{'} = 0$ if $j \neq i$. According to Formula (13), we get the maximum value of $F_i^{'}$ as follows

$$F_i^{'max} = 1/\max\{\frac{\xi_i^{cpu}}{C_{cpu}^{'}}, \frac{\xi_i^{gpu}}{C_{gpu}^{'}}\}. \tag{14}$$

We now can get the upper bound δ for Formula (12) regarding all feasible allocations in the case of Formula (14), i.e.,

$$\begin{aligned} \delta &= \max_{1 \leqslant i \leqslant n} \{\frac{F_i^{'}}{S_i}\} - \min_{1 \leqslant j \leqslant n} \{\frac{F_j^{'}}{S_j}\} = \max_{1 \leqslant i \leqslant n} \{\frac{F_i^{'max}}{S_i}\} \\ &= \max_{1 \leqslant i \leqslant n} \{\frac{\max\{\gamma_i^{cpu}, \gamma_i^{gpu}\}}{\frac{w_i}{\sum_{k=1}^{n} w_k} \cdot \max\{\frac{\xi_i^{cpu}}{C_{cpu}^{'}}, \frac{\xi_i^{gpu}}{C_{gpu}^{'}}\}}\}. \end{aligned}$$

Finally, according to Formula (11) (12), we have $|\frac{F_i}{S_i} - \frac{F_j}{S_j}| \leqslant \delta$ and our proof completes. ∎

Theorem 1 shows an important relationship between the knob η and the upper bound of unfairness degree δ for EMRF. In practice, given a knob value η, we can estimate its unfairness upper bound δ for EMRF. Reversely, given the maximum unfairness degree δ, we can calculate the knob value η.

Efficiency-Oriented Allocation. The efficiency-oriented allocation of EMRF is to find a feasible allocation $\langle F_1^{'}, F_2^{'}, \cdots, F_n^{'} \rangle$ that maximizes the system utilization under the idle resource vector of $\mathbf{C}^{'}$. Moreover, for any two users i and j with the same *CPU-to-GPU workload ratio* (i.e., $\rho_i = \rho_j$), switching resources between them has no impact on efficiency but fairness. In order for better fairness, we still keep Formula (4) hold for any two users i and j given that $\rho_i = \rho_j$. It can be modeled as a linear programming optimization problem with n unknowns in its formulation representing the allocations of

n users below,

Efficiency Allocation Optimization.

$$\textbf{Maximize} \quad \sum_{i=1}^{n} F_i^{'}. \tag{15}$$

subject to:

$$\sum_{i=1}^{n} F_i^{'} \xi_i^{cpu} \leqslant C_{cpu}^{'}, \qquad \sum_{i=1}^{n} F_i^{'} \xi_i^{gpu} \leqslant C_{gpu}^{'}. \tag{16}$$

$$\frac{F_i^{'}}{S_i} = \frac{F_j^{'}}{S_j}. \qquad (\rho_i = \rho_j \forall i, j \in [1, n]) \tag{17}$$

Solving the linear program, we can get the optimal (largest) value, denoted by $F^{'max}$, for the objective function of Formula (15), i.e., $F^{'max} = \sum_{i=1}^{n} F_i^{'}$. Then the total resources or allocation efficiency (i.e., $\sum_{i=1}^{n} F_i$) achieved by all users can be maximized according to Formula (10).

According to Theorem 1, we can now conclude that EMRF is a knob-based elastic fairness-efficiency allocation policy that can maximize the system utilization whereas guarantee the δ-fairness, under the given knob η.

Let's now take a look at former Example 1 to see how EMRF policy works. Suppose the knob here is $\eta = 0.5$. It has $\gamma_1^{cpu} = 0.1/100, \gamma_1^{gpu} = 0.9/800, \gamma_2^{cpu} = 0.4/100, \gamma_2^{gpu} = 0.6/800, S_1 = 444.4, S_2 = 125.0, F_1^{max} = 470.5, F_2^{max} = 132.4$. We then have $C_{cpu}^{'} = 50, C_{gpu}^{'} = 548.5$ and $\delta = 1.125$. Therefore, we have the following efficiency allocation optimization problem: maximizing $F_1^{'} + F_2^{'}$ subject to $F_1^{'} \cdot 0.1 + F_2^{'} \cdot 0.4 \leqslant 50$ and $F_1^{'} \cdot 0.9 + F_2^{'} \cdot 0.6 \leqslant 548.5$. By solving the linear program, it returns $F_1^{'} = 500, F_2^{'} = 0$. According to Formula (10), we have $F_1 = 735.3$ and $F_2 = 66.2$, which increases the GPU resource utilization of DRF from 63% to 88% at the expense of $\delta = 1.125$-fairness.

C. Scheduling System Implementation

By modeling the resource allocation problem as a linear programming problem, we solve the problem and develop a scheduling system. The linear program of Formula (15) for allocation optimization can be solved by using traditional optimization solvers. For efficiency, this study uses GNU Linear Programming Kit (GLPK) [3]. The ratios of these allocations make up the weights to a weighted fair scheduler (e.g., WFQ [13] and PRS [34]) that allocates resources based on users' weights.

At runtime, the allocations (i.e., the weights to the weighted scheduler) for users need to be updated dynamically, being adaptive to system changes. Likewise, if there is a significant change of *CPU-to-GPU workload ratio* for users' workloads, we should re-compute the allocations for users. Algorithm 1 gives the pseudo-code for our EMRF implementation. It maintains and updates the system status periodically, including the remaining idle resources, the number of *active* users (i.e., refers to those with running jobs in the system) and their statistics of *CPU-to-GPU workload ratios*, etc (Line 1). Next, it periodically compute the suitable allocations for users, as the input of relative weights to the weighted fair scheduler, by invoking the linear program optimization solver GLPK (Line 2-3). Lastly, it allocates resources to users dynamically according to their computed weights (Line 4-9). To enable efficiently concurrent kernel execution across different applications at runtime in the CCGA, we adopt the kernel slicing technique [42] in our scheduling system. It slices a kernel of an application into multiple sub-kernels (namely *slices*) and dynamically allocates resources to slices for different applications. The slice size has balanced the overhead and performance gains of slicing. For more details about slicing, we refer the readers to the paper [42].

Algorithm 1 Implementation of Elastic Multi-Resource Fairness (EMRF)

1: Maintain the statistics of *CPU-to-GPU workload ratio* for each user i over a pre-configured time window ΔT.
2: Invoke the linear program optimization solver *GLPK* for efficient allocation optimization (Formula (15)) to compute allocations for users over window ΔT periodically, according to Formula (10).
3: Use the computed allocations (denoted by F_i^* for user i) in Line 2 as relative weights for users in the following allocation.
4: Find user i with the lowest normalized resource share, i.e., $F_i/F_i^* = \min_{1 \leqslant k \leqslant n} F_k/F_k^*$. ▷ Resource allocation based on max-min heuristic.
5: $D \leftarrow$ demand of user i's next task.
6: **if** $F_i + D \leqslant C^{cpu} + C^{gpu}$ **then**
7: $\quad F_i = F_i + D$.
8: **else**
9: $\quad$ System is full and the allocation stops.

D. Analysis of Essential Properties

In this section, we start to analyze EMRF formally with the three essential fair sharing properties (i.e., SI, EF, PE) listed in Section IV. We first show that by configuring $\eta \geqslant S_i/F_i^{max}$, each user under the EMRF allocation can receive at least the amount of resources when using its own partition of resources exclusively.

Theorem 2. (Sharing Incentive): *The EMRF allocation obtained by solving Formula (15) and (10) is SI when* $\eta \geqslant \frac{S_i}{F_i^{max}}$.

Proof: We start by specifying the vector $\mathbf{F}_i$ and $\overline{\mathbf{F}}_i$ defined in Section IV-A as follows,

$$\mathbf{F}_i = \langle F_i \xi_i^{cpu}, F_i \xi_i^{gpu} \rangle, \quad \overline{\mathbf{F}}_i = \langle S_i \xi_i^{cpu}, S_i \xi_i^{gpu} \rangle.$$

According to Formula (10), we have

$$\mathbf{F}_i = \langle (F_i^{max}\eta + F_i^{'}) \xi_i^{cpu}, (F_i^{max}\eta + F_i^{'}) \xi_i^{gpu} \rangle$$

Then,

$$\mathbf{F}_i - \overline{\mathbf{F}}_i = \langle (F_i^{max}\eta + F_i^{'} - S_i) \xi_i^{cpu}, (F_i^{max}\eta + F_i^{'} - S_i) \xi_i^{gpu} \rangle$$

When $\eta \geqslant \frac{S_i}{F_i^{max}}$, it holds $(F_i^{max}\eta + F_i^{'} - S_i)\xi_i^{cpu} \geqslant 0$ and $(F_i^{max}\eta + F_i^{'} - S_i)\xi_i^{gpu} \geqslant 0$. We therefore have $\mathbf{F}_i \succeq \overline{\mathbf{F}}_i$ and our proof completes. ■

Next we show that under the EMRF allocation, no user envies other users' allocations.

Theorem 3. (Envy-freeness): *Every user under the EMRF allocation prefers its own allocation to others.*

Proof: Let's start proof by contradiction to suppose that user i envies user j under EMRF policy. Then there must be

$$\mu_i(\mathbf{F}_i) < \mu_i(\mathbf{F}_j). \tag{18}$$

As preparation for the following proof, we first deduce the formula for $\mu_i(\mathbf{F}_j)$. Similar to the deduction of Formula (3), we adopt the progressive filling approach and the allocation stops when one resource is saturated under the resource vector of $\mathbf{F}_j = \langle F_j^{cpu}, F_j^{gpu} \rangle$. Then there is $\max\{\frac{\mu_i(\mathbf{F}_j)\cdot\xi_i^{cpu}}{F_j^{cpu}}, \frac{\mu_i(\mathbf{F}_j)\cdot\xi_i^{gpu}}{F_j^{gpu}}\} = 1$. Hence,

$$\begin{aligned}\mu_i(\mathbf{F}_j) &= \frac{1}{\max\{\frac{\xi_i^{cpu}}{F_j^{cpu}}, \frac{\xi_i^{gpu}}{F_j^{gpu}}\}} = \frac{1}{\max\{\frac{\rho_i}{(\rho_i+1)F_j^{cpu}}, \frac{1}{(\rho_i+1)F_j^{gpu}}\}} \\ &= \frac{1}{\max\{\frac{\rho_i}{(\rho_i+1)}\cdot\frac{(\rho_j+1)}{\rho_j F_j}, \frac{1}{(\rho_i+1)}\cdot\frac{(\rho_j+1)}{F_j}\}} \\ &= \frac{(\rho_i+1)F_j}{(\rho_j+1)\max\{\frac{\rho_i}{\rho_j}, 1\}}. \end{aligned} \tag{19}$$

We consider the following three cases,

(I). $\rho_i = \rho_j$: we have $\frac{F_i^{'}}{S_i} = \frac{F_j^{'}}{S_j}$ according to Formula (17). Then there is $\frac{F_i}{S_i} = \frac{F_j}{S_j}$ according to Formula (10). By switching the allocation between user i and j, there should be $\frac{\mu_i(F_j)}{S_i} = \frac{\mu_j(F_i)}{S_j}$ according to the constraint of Formula (17). Moreover, according to Formula (19), we have $\mu_i(\mathbf{F}_j) = F_j$ and $\mu_j(\mathbf{F}_i) = F_i$. We therefore have $\frac{F_j}{S_i} = \frac{F_i}{S_j}$. Then it is true that $\frac{F_i}{S_i}/\frac{F_j}{S_i} = \frac{F_j}{S_j}/\frac{F_i}{S_j}$. It shows that $F_i = F_j \Rightarrow \mu_i(\mathbf{F}_i) = \mu_i(\mathbf{F}_j)$, which violates Formula (18) and shows that the assumption does not hold.

(II). $\rho_i > \rho_j$**:** Formula (19) is equivalent to $\mu_i(\mathbf{F}_j) = \frac{(\rho_i+1)F_j}{(\rho_j+1)\frac{\rho_i}{\rho_j}} = \frac{\rho_i\rho_j+\rho_j}{\rho_i\rho_j+\rho_i}\cdot F_j < F_j$. Similarly, we can deduce that $\mu_j(\mathbf{F}_i) < F_i$. Then after exchanging the allocation between user i and j, we have $\mu_i(\mathbf{F}_j)+\mu_j(\mathbf{F}_i) < F_i+F_j$, violating the utilization/efficiency maximization requirement in Section V-B and therefore the assumption is not true.

(III). $\rho_i < \rho_j$**:** we can equally transform Formula (19) to $\mu_i(\mathbf{F}_j) = \frac{(\rho_i+1)F_j}{(\rho_j+1)} < F_j$. Likewise, we can get $\mu_j(\mathbf{F}_i) = \frac{(\rho_j+1)F_i}{(\rho_i+1)} < F_i$. If swapping the allocation between user i and j, there will be $\mu_i(\mathbf{F}_j)+\mu_j(\mathbf{F}_i) < F_i+F_j$, which violates the utilization/efficiency maximization requirement and therefore the assumption does not hold.

Based on the analysis of the above three cases, we conclude that EMRF policy is envy-freeness. ■

Moreover, we show that the allocation of EMRF is efficient, making that no user can improve its allocation without decreasing that of other users.

Theorem 4. (Pareto Efficiency): *The allocation of EMRF is pareto efficient.*

Proof: By contradiction, let's suppose that the allocation $\mathbf{F} = \langle \mathbf{F}_1, \mathbf{F}_2, \cdots, \mathbf{F}_n \rangle$ under the EMRF policy is *not* pareto efficient, i.e., it must exist a feasible allocation $\tilde{\mathbf{F}} = \langle \tilde{\mathbf{F}}_1, \tilde{\mathbf{F}}_2, ..., \tilde{\mathbf{F}}_n \rangle$ satisfying that $\mu_i(\tilde{\mathbf{F}}_i) \geqslant \mu_i(\mathbf{F}_i)$ for all user i, and $\mu_j(\tilde{\mathbf{F}}_j) > \mu_j(\mathbf{F}_j)$ for some user j. Recall in Section V-A, our EMRF policy follows the progressive filling approach and the allocation stops when one resource is fulfilled. That is, for allocation $\mathbf{F}$, we have

$$\max\{\sum_{i=1}^{n}\mu_i(\mathbf{F}_i)\gamma_i^{cpu}, \sum_{i=1}^{n}\mu_i(\mathbf{F}_i)\gamma_i^{gpu}\} = 1.$$

Then there is $\sum_{i=1}^{n}\mu_i(\mathbf{F}_i)\gamma_i^{cpu} < \sum_{i=1}^{n}\mu_i(\tilde{\mathbf{F}}_i)\gamma_i^{cpu}$ and $\sum_{i=1}^{n}\mu_i(\mathbf{F}_i)\gamma_i^{gpu} < \sum_{i=1}^{n}\mu_i(\tilde{\mathbf{F}}_i)\gamma_i^{gpu}$. It gives us that,

$$\max\{\sum_{i=1}^{n}\mu_i(\tilde{\mathbf{F}}_i)\gamma_i^{cpu}, \sum_{i=1}^{n}\mu_i(\tilde{\mathbf{F}}_i)\gamma_i^{gpu}\} > 1.$$

for allocation $\tilde{\mathbf{F}}$, which is *not* a feasible allocation and implies that the premise is false. Hence, EMRF is pareto-efficient. ■

VI. Evaluation

We start with experimental setup by describing our computing platform and workloads. Next, we give the experimental results for our approach.

A. Experimental Setup

We conduct a testbed experiment in a Linux machine consisting of an AMD A8-3870K APU, with its specification detailed in Table I. The machine has 8 GB DRAM, and its OS is Ubuntu 15.10. For the experimental workload, we consider three types of application programs from Rodinia benchmarks suite [41], and follow the previous study on categorizing co-run computation on APU: 1). *co-run friendly program* (referring to the program that achieves the best performance when using both CPU and GPU devices together to deal with the application), including Gaussian Elimination (GE), K-means (KM) and Heart Wall (HW); 2). *CPU-dominant program* (referring to the program that achieves the best performance when all its workload runs on CPU), including Myocyte (MY) and k-Nearest Neighbors (KNN); 3). *GPU-dominant program* (referring to the program that gets the best performance when all its workload runs on GPU), including B+Tree (BT), CFD Solver (CFD) and LU Decomposition (LUD). The detailed descriptions of them are presented in Table III. Based on the benchmarks in Table III, we generate a set of workloads, as shown in Table II, for the following experimental evaluation.

Platform	**A8-3870K**	
	CPU	GPU
# Cores	4	400
Clock Frequency(MHz)	800	600
Peak FLOPS (GFLOPS)	24	480
Zero copy buffer (MB)	512 (shared)	
Cache size(MB)	4 (shared)	

TABLE I: The configuration of testbed platform.

Workloads	**Applications**	**Workload Category**
M_1	GE, KM, HW, MY, BT	Co-run friendly + GPU-dominant + CPU-dominant
M_2	GE, KM	Co-run friendly + Co-run friendly
M_3	HW, MY	Co-run friendly + CPU-dominant
M_4	KM, BT	Co-run friendly + GPU-dominant
M_5	KNN, CFD	CPU-dominant + GPU-dominant

TABLE II: A set of different workloads generated from Rodinia benchmarks listed in Table III.

B. Experiment Results

We start by evaluating EMRF in a testbed Linux system, and compare its performance with WFQ (Weighted Fair Queuing) [13] and DRF [15]. Next we show the system efficiency and soft fairness results for EMRF under different knobs.

Applications	Description	Category
Gaussian Elimination (GE)	A classic algorithm used to solve systems of linear equations with a sequence of operations made on the associated matrix of coefficients [2]. We consider a matrix size of 4096×4096.	Co-run friendly
K-means (KM)	A popular clustering algorithm in data mining. It identifies related data points by associating each data point with its nearest cluster, computing new cluster centroids, and iterating until convergence. We take KDD CUP dataset [5] as its input data.	Co-run friendly
Heart Wall (HW)	It tracks the movement of a mouse heart over a sequence of ultrasound images to record response to the stimulus [27]. The number of frame is set to 20.	Co-run friendly
Myocyte (MY)	A biology simulator that models cardiac myocyte (heart muscle cell) and simulates its behavior. It can identify potential therapeutic targets which may be useful for the treatment of heart failure [27]. In our experiment, we set its simulation time to be 2000.	CPU-dominant
k-Nearest Neighbors (KNN)	A non-parametric lazy learning algorithm used for classification and regression. It finds the k-nearest neighbors from an unstructured data set by calculating the Euclidean distance from the target latitude and longitude, and evaluating the k nearest neighbors iteratively. We synthesize an input data with 5000 records using its provided tool.	CPU-dominant
B+Tree (BT)	An n-ary tree data structure often used in the implementation of database indexes. We use the mil data set [6] as its input data.	GPU-dominant
CFD Solver (CFD)	an unstructured grid finite volume solver for the three-dimensional Euler equations for compressible fluid flow [9]. The data set fvcorr.domn.097K [1] is taken as input in our experiment.	GPU-dominant
LU Decomposition (LUD)	A method to calculate the solutions of a set of linear equations by factoring a matrix as the product of a lower triangular matrix and an upper triangular matrix. We consider a matrix of 2048×2048.	GPU-dominant

TABLE III: The description of Rodinia benchmarks used in the paper (categorized according to [41]).

Finally, we evaluate various policies under different numbers of users.

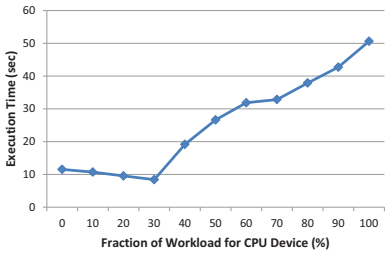


Fig. 3: The execution time for the HW benchmark under different workload distributions.

1) Throughput and System Efficiency: Recall in Section VI-A, there are three types of application programs. In this section, we conduct two sets of experiments to evaluate our approach. One set is to mix five representative benchmarks (e.g., GE, KM, HW, MY, BT) from Table III. The other set is to study all possible combinations of co-run computation on the shared environment (i.e., *Co-run friendly program* vs *Co-run friendly program*, *Co-run friendly program* vs *CPU dominant program*, *Co-run friendly program* vs *GPU dominant program*, and *CPU dominant program* vs *GPU dominant program*). The knob of EMRF policy is configured to $\eta = 0.1$.

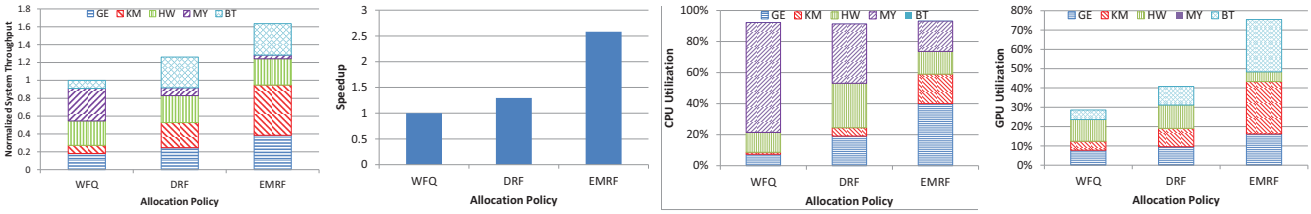
Mixed Benchmarks Evaluation. This experiment employs M_1 from Table II for five users with different weighted shares (i.e., $2:1:3:4:1$) in the APU system, each submitting a distinct benchmark (e.g., GE, KM, HW, MY, BT). Each benchmark does the computation in the co-run execution manner, i.e., some portion of its workload running on CPU and others on the GPU. Different ratios of workload distribution between CPU and GPU devices can have a significant impact on the performance of an application. For example, as shown in Figure 3, the execution time for HW benchmark is varying under different workload distributions. There tends to be a suitable workload distribution for an application, which in fact has also been explored and discussed by existing work [41]. In our experiment, for *co-run friendly program* benchmarks GE, KM and HW, their suitable ratios ξ_i^{cpu} of workload distribution on the CPU device are $21\%, 7\%$ and 24%, respectively. Moreover, we set $\xi_i^{cpu} = 0\%$ for the *GPU-dominant program* benchmark BT and $\xi_i^{cpu} = 100\%$ for the *CPU dominant program* benchmark MY.

Figure 4(a) shows the throughput results (normalized to WFQ) for different allocation policies. The system throughput (i.e., the aggregated throughput of five benchmarks) of WFQ is the lowest (worst) of the three allocation policies. The problem is that, the WFQ throttles the GPU-bounded applications (e.g., KM, BT) severely, resulting in a poor GPU utilization of only 28% shown in Figure 4(d). This is because that the capacity of CPU is much smaller than that of GPU. In order to achieve the strict $2:1:3:4:1$ fairness allocation ratio across five benchmarks, WFQ lets CPU-bounded workloads (i.e., MY) possess much larger amount of CPU throughput (87%) than GPU-bounded workloads (i.e., KM), as shown in Figure 4(c). It causes the allocation of GPU-bounded workloads to be throttled on the CPU device, making it unable to maximally utilize the GPU device and thereby resulting in the low utilization for GPU device as illustrated in Figure 4(d).

DRF performs much better than WFQ (i.e., as high as 41% GPU utilization of DRF in Figure 4(c)). It guarantees that the dominant (weighted) fair shares are equal across different benchmarks.

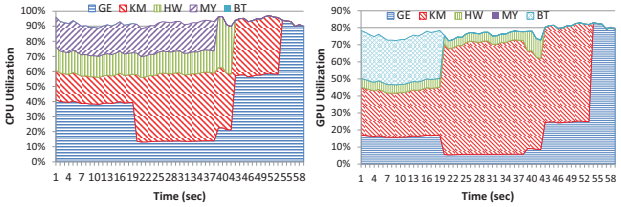
EMRF performs the best. It improves the system throughput significantly by adjusting the allocation weight across five benchmarks so that the GPU device is maximally utilized, as high as 75% GPU utilization for EMRF shown in Figure 4(d). Moreover, Figure 4(b) gives the speedup result for each policy relative to WFQ, i.e., the *speedup* is defined as the ratio of the execution time of WFQ to that of the corresponding policy. It shows that EMRF achieves the best performance result among the three scheduling policies, all of which are attributed to the dynamic resource allocation mechanism of EMRF.

We study the resource utilization of each device in more depth. Figure 5 shows the utilizations of CPU and GPU for five benchmarks under EMRF policy over time. Initially, there are five benchmarks executed concurrently on the APU device under the EMRF scheduling policy until BT completes (Figure 5(b)) at the 19^{th} second. In that case, EMRF adjusts the allocation among the remaining four active benchmarks so that the released CPU and GPU resources are possessed. At the 38^{th} second, MY finishes (Figure 5(a)) and likewise, EMRF



(a) System Throughput. (b) Speedup (relative to WFQ). (c) Average CPU Utilizations. (d) Average GPU Utilizations.

Fig. 4: The experimental results for five users running different benchmarks in a shared APU machine under different fair allocation policies. For the EMRF policy, its knob value is configured to be 0.1.



(a) CPU utilization. (b) GPU utilization.

Fig. 5: Stacked chart showing the resource utilization for five benchmarks over time under EMRF policy.

scheduler adjusts the resource allocations among GE,KM and HW. The whole dynamic allocation repeats until all benchmarks complete.

Different Co-run Combinations. As illustrated in Figure 6, we further extend our experiment to consider all possible combinations of different types of application programs by considering M_2, M_3, M_4 and M_5 from Table II, respectively. Figure 6(a) gives the throughput results of two *co-run friendly program* benchmarks GE and KM with equal share under different allocation policies, which are normalized over that of WFQ. It shows that EMRF achieves better results than others, since it is able to adjust the allocations between co-run friendly programs so as to maximize the system utilization. Figures 6(b) shows the results of the co-run friendly program (e.g., HW) sharing with the CPU-dominant program (e.g., MY), whereas Figure 6(c) gives the results of the co-run friendly program (e.g., KM) and the GPU-dominant program (e.g., BT). In these two cases, our EMRF achieves the best performance results among the three scheduling policies. Compared to the WFQ and DRF that consider the fairness only, EMRF additionally considers the efficiency with the attempt to maximize both CPU and GPU utilization in these two cases. Finally, Figure 6(d) gives the sharing case of CPU-dominant program (e.g., KNN) and GPU-dominant program (e.g., CFD). In this case, WFQ performs the worst, since it constrains the GPU allocation for GPU-dominant program BT in order to strictly keep the same allocation as that of CPU-dominant program KNN. However, DRF and EMRF achieves the same better performance, since in this case they both can allow KNN and CFD to freely possess CPU and GPU resources respectively without constraints.

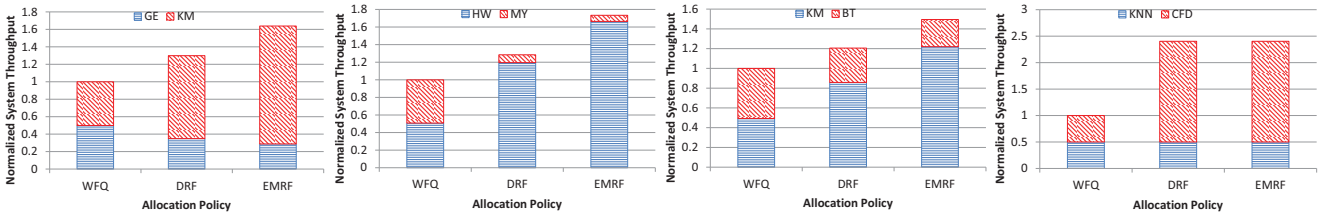
2) EMRF Results Under Different Knobs: EMRF is a knob-based elastic allocation policy that can balance fairness and efficiency flexibly. In this section, we show the impacts of different knob configurations on the system efficiency and fairness with the mix of all five workloads. Note in Section V-B that EMRF is to maximize the system efficiency while guaranteeing the *soft* fairness. Here we define a term *soft fairness degree* to quantify the soft fairness. The smaller value of soft fairness degree indicates the better fairness result.

Figure 7 shows the normalized results of throughput (relative to that when knob is 1) and soft fairness for EMRF with different knobs. It favors the throughput (or efficiency) but worsens the fairness (i.e., the soft fairness degree is large) when knob value is very small. In contrast, as we increase the knob value, the fairness can be better at the expense of efficiency. It indicates that, by controlling such a knob, users can flexibly balance the tradeoff between fairness and efficiency with our EMRF.

3) Evaluation on Different Numbers of Users: This section evaluates the throughput under different numbers of users. Figure 8 presents the experimental results for different policies. Particularly, we consider three EMRF policy instances by varying the knob values, namely, EMRF-0.8, EMRF-0.5, and EMRF-0.1, under different knob settings of 0.8, 0.5 and 0.1, respectively. We have the following observations. First, for each allocation policy, there is a decreasing trend for its throughput curve as more users join in the system. The reason is that, the resource competition and fairness constraint become more serious as we increase the number of users, leading to lower resource utilization. Second, DRF outperforms WFQ, whereas EMRF is better than DRF. Typically, as we decrease the knob, EMRF achieves much better throughput results in all users cases. Third, by comparing EMRF with different knob values, it shows that the throughput curve becomes increasingly stable as the knob becomes smaller. This is because that smaller knob value leads to more room or freedom for efficiency optimization in all users cases. Moreover, as discussed previously in Section VI-B2, DRF is indeed a special case of EMRF given that its knob value is one, explaining why the curve of DRF drops fast compared to the three EMRF policy instances.

VII. Related Work

Heterogeneous Computing Schedulers. There are a number of studies on task scheduling in heterogeneous computing. Wang et al. [38] proposed a fine-grained fair sharing scheduler named Simultaneous Multikernel (SMK), which can increase



(a) *Co-run friendly program* vs *Co-run friendly program*. (b) *Co-run friendly program* vs *CPU dominant program*. (c) *Co-run friendly program* vs *GPU dominant program*. (d) *CPU dominant program* vs *GPU dominant program*.

Fig. 6: The normalized throughput results for four possible combinations of co-run programs under different allocation policies. We normalize them over that of WFQ. We configure the knob value of EMRF policy to be 0.1.

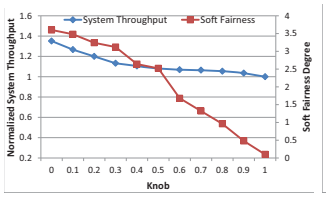


Fig. 7: The system efficiency and soft fairness for EMRF under different knobs.

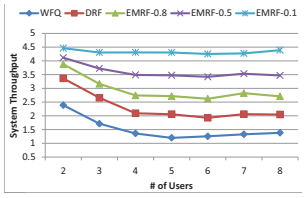


Fig. 8: The throughput results for different allocation policies under different numbers of users.

resource utilization while maintaining the resource fairness among kernels by scheduling kernels from different applications dynamically. Observing that GPU memory is a critical performance factor for applications, Jog et al. [21] proposed a First-ready Round-robin FCFS (FR-RR-FCFS) memory scheduler to improve both fairness and system performance for concurrent GPGPU applications. Aguilera et al. [7] examined several different ways to characterize "fairness" for GPU spatial multitasking, by balancing individual application's performance and overall system performance. Zhang et al. [41] had a performance study of scheduling tasks of an application to both CPU and GPU simultaneously by developing a benchmark suite called Rodinia. Moreover, there are also some optimization studies on the performance improvement for specific applications, frameworks or algorithms on heterogeneous platforms, including [11], [17], [18]. To summarize, all of these existing studies focus on performance and fairness optimization for either CPU only or GPU only. However, there is no work that systematically studies the tradeoff between the performance and fairness in heterogeneous computing. Our proposed EMRF scheduler can address it, since it is an elastic tradeoff scheduler that can balance the performance and fairness flexibly for users via combining CPU and GPU as a whole.

Multi-Resource Fairness. In cluster computing, DRF is the most popular fair policy in the literature for multi-resource allocation [15], [36], which achieves fair allocation of multiple resources on the basis of dominant shares. The attractiveness of DRF stems from its good properties including sharing incentive, envy freeness, and pareto efficiency. It has been implemented in many computing frameworks, such as YARN [32] and Mesos [19]. Subsequently, there have been a lot of extensions and generalizations for DRF. Wang et al. [36] extended DRF to a heterogeneous distributed system consisting of a number of heterogeneous machines. Kash et al. [22] extended the DRF model to a dynamic setting where users can join the system over time but will never leave. Bhattacharya et al. [8] generalized DRF to support hierarchical scheduling. Liu et al. [23] relaxed DRF policy by proposing a Reciprocal Resource Fairness to allow the trade among different types of resources between users. Dolev et al. [14] proposed an alternative to DRF by considering the global system bottleneck resource. Parkes et al. [25] proposed several schemes to extend DRF, and particularly focused on the case of indivisible tasks. Considering that the resource demand vector required by DRF is hard to get in computer architectures, Zahedi et al. [40] proposed an alternative multi-resource policy based on Cobb-Douglas utility function for multiprocessors. Wang et al. [35] considered the multi-tiered storage consisting of SSD and HDD and proposed a bottleneck-aware allocation policy to balance fairness and efficiency for users. In comparison with the previous studies, we consider CCGA and focus on the tradeoff balancing between the multi-resource fairness and efficiency by proposing an EMRF policy. It extends the DRF policy for CCGA to allow users to flexibly tune and balance the tradeoff with a knob.

VIII. Conclusion and Future Work

Heterogeneity is a trend in achieving energy-efficient computing. By removing PCI-e bus, coupled CPU-GPU architectures have demonstrate promising results in various applications. Still, fairness in sharing the CPU and the GPU on such architectures is an open problem. In this paper, we show for coupled CPU-GPU architectures that, it is essential to consider both CPU and GPU as a whole in fair resource allocation rather than separately for each computing device as previous studies did. This is because both CPU and GPU are computing devices and used for computation simultaneously in heterogeneous computing. To the best of our knowledge, this is first work that combines CPU and GPU devices as a whole in fair resource allocation for coupled CPU-GPU architectures. We cast the heterogeneous allocation problem to the multi-resource fairness allocation problem and consider the tradeoff between fairness and efficiency. We find that the approaches proposed by previous studies are heuristics, which cannot truly tell and guarantee the QoS of δ-fairness mentioned in this paper. We propose an elastic multi-resource fairness (EMRF)

to address it. It can allow users to flexibly balance fairness and efficiency using a knob while guaranteeing δ-fairness (See Theorem 1 in Section V-B). We also show that it satisfies several desirable properties including sharing incentive, envy freeness and pareto efficiency. We evaluate our method with real experiments, showing that our approach can achieve high efficiency and fairness.

ACKNOWLEDGEMENT

We thank the anonymous reviewers for their constructive comments. This work is in supported by National Natural Science Foundation of China (No.61303021). Bingsheng He's work is partially supported by an NUS startup grant in Singapore. Shuhao Zhang's work is partially funded by the Economic Development Board and the National Research Foundation of Singapore.

REFERENCES

[1] Fvcorr.domn.097k dataset. In *https://github.com/kkushagra/rodinia/tree/master/data/cfd*.

[2] Gaussian elimination. In *https://en.wikipedia.org/wiki/Gaussian_elimination*.

[3] Gnu linear programming kit (glpk). In *https://www.gnu.org/software/glpk/*.

[4] Heterogeneous system architecture. In *https://en.wikipedia.org/wiki/Heterogeneous_System_Architecture*.

[5] The kdd cup datasets. In *http://www.cs.cornell.edu/projects/kddcup/datasets.html*.

[6] Mil dataset for b+ tree. In *https://github.com/kkushagra/rodinia/tree/master/data/b+tree*.

[7] P. Aguilera, K. Morrow, and N. S. Kim. Fair share: Allocation of gpu resources for both performance and fairness. In *ICCD'14*, pages 440–447, Oct 2014.

[8] A. A. Bhattacharya, D. Culler, E. Friedman, A. Ghodsi, S. Shenker, and I. Stoica. Hierarchical scheduling for diverse datacenter workloads. In *SOCC'13*, pages 4:1–4:15, New York, NY, USA, 2013. ACM.

[9] J. Blazek. *Computational fluid dynamics: principles and applications*. Butterworth-Heinemann, 2015.

[10] A. Branover, D. Foley, and M. Steinman. Amd fusion apu: Llano. *IEEE Micro*, 32(2):28–37, Mar. 2012.

[11] L. Chen, X. Huo, and G. Agrawal. Accelerating mapreduce on a coupled cpu-gpu architecture. In *SC'12*, pages 25:1–25:11, Los Alamitos, CA, USA, 2012. IEEE Computer Society Press.

[12] S. Damaraju, V. George, S. Jahagirdar, T. Khondker, R. Milstrey, S. Sarkar, S. Siers, I. Stolero, and A. Subbiah. A 22nm ia multi-cpu and gpu system-on-chip. In *ISSCC'12*, pages 56–57, Feb 2012.

[13] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. In *SIGCOMM'89*, pages 1–12, New York, NY, USA, 1989. ACM.

[14] D. Dolev, D. G. Feitelson, J. Y. Halpern, R. Kupferman, and N. Linial. No justified complaints: On fair sharing of multiple resources. In *ITCS'12*, pages 68–75, New York, NY, USA, 2012. ACM.

[15] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica. Dominant resource fairness: Fair allocation of multiple resource types. In *NSDI'11*, pages 323–336, Berkeley, CA, USA, 2011. USENIX Association.

[16] K. O. W. Group et al. The opencl specification, version 1.2, 15 november 2011. *Cited on pages*, 18(7):30.

[17] J. He, M. Lu, and B. He. Revisiting co-processing for hash joins on the coupled cpu-gpu architecture. *Proc. VLDB Endow.*, 6(10):889–900, Aug. 2013.

[18] J. He, S. Zhang, and B. He. In-cache query co-processing on coupled cpu-gpu architectures. *Proc. VLDB Endow.*, 8(4):329–340, Dec. 2014.

[19] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica. Mesos: A platform for fine-grained resource sharing in the data center. In *NSDI'11*, pages 295–308, Berkeley, CA, USA, 2011. USENIX Association.

[20] C. Joe-Wong, S. Sen, T. Lan, and M. Chiang. Multiresource allocation: Fairness-efficiency tradeoffs in a unifying framework. *IEEE/ACM Trans. Netw.*, 21(6):1785–1798, Dec. 2013.

[21] A. Jog, E. Bolotin, Z. Guz, M. Parker, S. W. Keckler, M. T. Kandemir, and C. R. Das. Application-aware memory system for fair and efficient execution of concurrent gpgpu applications. In *GPGPU-7*, pages 1:1–1:8, New York, NY, USA, 2014. ACM.

[22] I. Kash, A. D. Procaccia, and N. Shah. No agent left behind: Dynamic fair division of multiple resources. In *AAMAS'13*, pages 351–358, Richland, SC, 2013. International Foundation for Autonomous Agents and Multiagent Systems.

[23] H. Liu and B. He. Reciprocal resource fairness: Towards cooperative multiple-resource fair sharing in iaas clouds. In *SC'14*, pages 970–981, Piscataway, NJ, USA, 2014. IEEE Press.

[24] S. Mittal and J. S. Vetter. A survey of cpu-gpu heterogeneous computing techniques. *ACM Comput. Surv.*, 47(4):69:1–69:35, July 2015.

[25] D. C. Parkes, A. D. Procaccia, and N. Shah. Beyond dominant resource fairness: Extensions, limitations, and indivisibilities. *ACM Trans. Econ. Comput.*, 3(1):3:1–3:22, Mar. 2015.

[26] V. T. Ravi, M. Becchi, G. Agrawal, and S. Chakradhar. Supporting gpu sharing in cloud environments with a transparent runtime consolidation framework. In *HPDC'11*, pages 217–228, New York, NY, USA, 2011. ACM.

[27] L. G. Szafaryn, K. Skadron, and J. J. Saucerman. Experiences accelerating matlab systems biology applications. In *Proceedings of the Workshop on Biomedicine in Computing: Systems, Architectures, and Circuits*, pages 1–4. Citeseer, 2009.

[28] S. Tang, B. S. Lee, and B. He. Towards economic fairness for big data processing in pay-as-you-go cloud computing. In *CloudCom'14*, pages 638–643, Dec 2014.

[29] S. Tang, B. S. Lee, and B. He. Fair resource allocation for data-intensive computing in the cloud. *IEEE Transactions on Services Computing*, PP(99):1–1, 2016.

[30] S. Tang, B.-s. Lee, B. He, and H. Liu. Long-term resource fairness: Towards economic fairness on pay-as-you-use computing systems. In *ICS'14*, pages 251–260, New York, NY, USA, 2014. ACM.

[31] R. Ubal, B. Jang, P. Mistry, D. Schaa, and D. Kaeli. Multi2sim: A simulation framework for cpu-gpu computing. In *PACT'12*, pages 335–344, New York, NY, USA, 2012. ACM.

[32] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O'Malley, S. Radia, B. Reed, and E. Baldeschwieler. Apache hadoop yarn: Yet another resource negotiator. In *SOCC'13*, pages 5:1–5:16, New York, NY, USA, 2013. ACM.

[33] C. A. Waldspurger. Lottery and stride scheduling: Flexible proportional-share resource management. Technical report, Cambridge, MA, USA, 1995.

[34] C. A. Waldspurger and W. E. Weihl. Lottery scheduling: Flexible proportional-share resource management. In *OSDI'94*, Berkeley, CA, USA, 1994. USENIX Association.

[35] H. Wang and P. Varman. Balancing fairness and efficiency in tiered storage systems with bottleneck-aware allocation. In *FAST'14*, pages 229–242, Berkeley, CA, USA, 2014. USENIX Association.

[36] W. Wang, B. Li, and B. Liang. Dominant resource fairness in cloud computing systems with heterogeneous servers. In *INFOCOM'14*, pages 583–591, April 2014.

[37] Y. Wang and A. Merchant. Proportional-share scheduling for distributed storage systems. In *FAST'07*, pages 4–4, Berkeley, CA, USA, 2007. USENIX Association.

[38] Z. Wang, J. Yang, R. Melhem, B. Childers, Y. Zhang, and M. Guo. Simultaneous multikernel: Fine-grained sharing of gpgpus. *Computer Architecture Letters*, PP(99):1–1, 2015.

[39] M. Yuffe, E. Knoll, M. Mehalel, J. Shor, and T. Kurts. A fully integrated multi-cpu, gpu and memory controller 32nm processor. In *ISSCC'11*, pages 264–266, Feb 2011.

[40] S. M. Zahedi and B. C. Lee. Ref: Resource elasticity fairness with sharing incentives for multiprocessors. In *ASPLOS'14*, pages 145–160, New York, NY, USA, 2014. ACM.

[41] F. Zhang, J. Zhai, W. Chen, B. He, and S. Zhang. To co-run, or not to co-run: A performance study on integrated architectures. In *MASCOTS'15*, pages 89–92, Oct 2015.

[42] J. Zhong and B. He. Kernelet: High-throughput gpu kernel executions with dynamic slicing and scheduling. *IEEE Trans. Parallel Distrib. Syst.*, 25(6):1522–1532, June 2014.