REF: Resource Elasticity Fairness with Sharing Incentives for Multiprocessors

Seyed Majid Zahedi
Duke University
seyedmajid.zahedi@duke.edu

Benjamin C. Lee
Duke University
benjamin.c.lee@duke.edu

Abstract

With the democratization of cloud and datacenter computing, users increasingly share large hardware platforms. In this setting, architects encounter two challenges: sharing fairly and sharing multiple resources. Drawing on economic game-theory, we rethink fairness in computer architecture. A fair allocation must provide sharing incentives (SI), envy-freeness (EF), and Pareto efficiency (PE).

We show that Cobb-Douglas utility functions are well suited to modeling user preferences for cache capacity and memory bandwidth. And we present an allocation mechanism that uses Cobb-Douglas preferences to determine each user's fair share of the hardware. This mechanism provably guarantees SI, EF, and PE, as well as strategy-proofness in the large (SPL). And it does so with modest performance penalties, less than 10% throughput loss, relative to an unfair mechanism.

Categories and Subject Descriptors B.3.2 [*Hardware*]: Memory Structures; C.1.2 [*Processor Architectures*]: Multiple Data Stream Architectures; F.2.2 [*Theory of Computation*]: Analysis of Algorithms and Problem Complexity

Keywords Multiprocessor architectures, fair sharing, economic mechanisms, game theory

1. Introduction

Datacenter platforms are often poorly utilized, running at less than 30% of peak capability [1]. With poor utilization, server power is amortized over little computation. To address this inefficiency, software must share hardware. Mechanisms for fair resource allocation (or a lack thereof) determine whether users have incentives to participate in dynamic, shared hardware platforms. In this setting, architects encounter two challenges: sharing fairly and sharing multiple resources.

We rethink fairness in resource allocation for computer architecture. Adopting the game-theoretic definition, a fair hardware allocation is one in which

- *all users perform no worse than under an equal division,*
- *no user envies the allocation of another, and*
- *no other allocation improves utility without harming a user.*

Our resource allocation strategy relies on robust game theory, encouraging users to share hardware and ensuring equitable allocations when they do. Conventional wisdom, on the other hand, assumes that users have no choice but to share. In this setting, prior efforts devise mechanisms to equally distribute performance penalties from sharing, which is not equitable [14, 28].

Drawing on economic game theory, we present a fair, multi-resource allocation mechanism. This mechanism and its resulting allocations provide key game-theoretic properties. First, the mechanism provides sharing incentives (SI), ensuring that each agent is at least as happy as they would be under an equal division of shared resources. Without SI, agents would not participate in the proposed sharing mechanism. Instead, they would rather equally and inefficiently divide the hardware. Supposing agents share a system, they will desire a fair division of the hardware.

In economic game theory, a fair allocation is defined to be envy-free (EF) and Pareto efficient (PE) [37]. An allocation is EF if each agent prefers his own allocation to other agents' allocations. Equitable sharing is defined by EF for all agents. An allocation is PE if we cannot improve an agent's utility without harming another agent.

Finally, a mechanism to allocate hardware should be strategy-proof (SP), ensuring that agents cannot gain by misreporting their preferences. Without SP, strategic agents may manipulate the hardware allocation mechanism by lying. In practice, SP may be incompatible with SI, EF, and

ASPLOS '14, March 1–5, 2014, Salt Lake City, Utah, USA.

http://dx.doi.org/10.1145/2541940.2541962

PE [19]. But there exist allocation mechanisms that are approximately SP as long as many agents share a system. We refer to this weaker guarantee as strategy-proofness in the large (SPL).

Thus, we present a new framework for reasoning about fair resource allocation in computer architecture. Our contributions include the following:

- **Cobb-Douglas Utility in Computer Architecture.** We show that Cobb-Douglas utility functions are well suited to model user performance and preferences for multiple hardware resources. Given Cobb-Douglas utilities, we detail conditions for SI, EF, and PE. (§3)
- **Fair Allocation for Computer Architecture.** We present a new mechanism to fairly allocate multiple hardware resources to agents with Cobb-Douglas utilities. We prove its game-theoretic properties (SI, EF, PE, SPL) and describe its implementation. (§4)
- **Case Study for Cache Size and Memory Bandwidth.** We apply the mechanism to fairly allocate cache size and memory bandwidth. We evaluate with cycle-accurate processor and memory simulators for diverse application suites, including PARSEC, SPLASH-2x, and Phoenix MapReduce. (§5)
- **Performance Trade-offs.** We compare our mechanism against prior approaches that equalize slowdown, describing how the latter violates game-theoretic fairness. Our mechanism provides fairness with modest penalties ($< 10\%$ throughput loss) relative to a mechanism that does not provide SI, EF, PE, and SPL. (§5)

Without loss of generality, we evaluate our multi-resource allocation mechanism for cache size and memory bandwidth. In future, the mechanism can support additional resources, such as the number of processor cores. Collectively, our findings establish robust foundations for the fair division of multiple hardware resources.

2. Motivation and Background

We present a mechanism for allocating shared resources. This mechanism guarantees SI, EF, PE, and SPL. And we demonstrate its ability to allocate last-level cache capacity and off-chip memory bandwidth. Our mechanism design relies on two fundamental insights about utility functions for computer architecture.

First, we use Cobb-Douglas utility functions to accurately capture hardware performance. For example, $u = x^{\alpha_x} y^{\alpha_y}$ models performance u as a function of resource allocations for cache capacity x and memory bandwidth y. The exponents α capture non-linear trends and model performance elasticity (i.e., sensitivity) for each hardware resource. For example, if $\alpha_x > \alpha_y$, the agent prefers cache capacity to memory bandwidth.

Second, we design a mechanism that uses each agent's reported resource elasticity α to determine his fair share of hardware. Given Cobb-Douglas utilities, the fair share can be expressed in a closed-form equation. Thus, the mechanism is computationally trivial. Yet the resulting allocation provably guarantees each of the desired game-theoretic properties: SI, EF, PE, and SPL.

Game-theoretic versus Heuristic Fairness. Our approach addresses fundamental limitations in prior work. Prior mechanisms consider each user's performance penalty incurred from sharing [12, 13]. They then allocate a resource, such as memory bandwidth, trying to equalize slowdown [28]. While this approach produces equal outcomes, it is not fair in the economic sense. Our rigorous, game-theoretic analysis shows that equalizing slowdown provides neither SI nor EF.

Without these properties, strategic users would have no incentive to share. They would prefer an equal division of memory bandwidth rather than receive an equal slowdown guarantee from the allocation mechanism. Allocating multiple resources with heuristics, such as hill-climbing [4], is even more difficult and provides even fewer guarantees.

Cobb-Douglas versus Leontief. Cobb-Douglas allows us to guarantee fairness in computer architecture for the first time. Although Leontief [9, 15, 21] provides the same guarantees in distributed systems, they do not apply in a more fine-grained analysis of hardware behavior for two reasons.

First, unlike Leontief, Cobb-Douglas utilities capture diminishing returns and substitutability. Both of these effects are prevalent in architecture, whether in Amdahl's Law for multi-core parallelism [20], in data locality for cache sizing, or in communication intensity for bandwidth allocation. In these settings, linear Leontief preferences of the form $u = min(x_1/\alpha_1, x_2/\alpha_2)$ are ineffective.

Second, consider the complexity of Cobb-Douglas and Leontief. We use classical regression to fit log-linear Cobb-Douglas to architectural performance. In contrast, since Leontief is concave piecewise-linear, fitting it would require non-convex optimization, which is computationally expensive and possibly NP-hard [36]. Note that [9, 15, 21] did not encounter these difficulties because they assume that agents in a distributed system provide a demand vector (e.g., 2CPUs, 4GB-DRAM). Fitting architectural performance to Leontief is equivalent to finding the demand vector for substitutable microarchitectural resources (e.g., cache and memory bandwidth), which is conceptually challenging.

3. Fair Sharing and Cobb-Douglas

A mechanism for fair sharing should guarantee several game theoretic properties. First, the mechanism must provide sharing incentives (SI). Without such incentives, software agents would prefer equally divided resources to a sophisticated mechanism that shares hardware more efficiently.

If agents do intelligently share, they will want a fair division. In economic game theory, a fair allocation is envy-free (EF) and Pareto efficient (PE) [37]. We present an allocation mechanism that provides SI, EF, and PE for hardware resources given software agents with Cobb-Douglas utility.

Cobb-Douglas Utility. Suppose multiple agents share a system with several types of hardware resources $1, \ldots, R$. Let $x_i = (x_{i1}, \ldots, x_{iR})$ denote agent i's hardware allocation. Further, let $u_i(x_i)$ denote agent i's utility. Equation (1) defines utility within the Cobb-Douglas preference domain.

$$u_i(x_i) = \alpha_{i0} \prod_{r=1}^{R} x_{ir}^{\alpha_{ir}} \quad (1)$$

The exponents α introduce non-linearity, useful for capturing diminishing marginal returns in utility. The product models interactions and substitution effects between resources. The user requires both resources for progress because utility is zero when either resource is unavailable.

The parameters $\alpha_i = (\alpha_{i1}, \ldots, \alpha_{iR})$ quantify the elasticity with which an agent demands a resource. If $\alpha_{ir} > \alpha_{ir'}$, then agent i benefits more from resource r than from resource r'. These parameters are tailored to each agent and define her demand for resources.

With Cobb-Douglas utility functions, we reason about agents' preferences. Consider two allocations x and x' for agent i.

- If $u_i(x) > u_i(x')$, then $x \succ_i x'$ (strictly prefer x to x')
- If $u_i(x) = u_i(x')$, then $x \sim_i x'$ (indifferent to x and x')
- If $u_i(x) \geq u_i(x')$, then $x \succsim_i x'$ (weakly prefer x to x')

Cobb-Douglas preferences are a good fit for resources in computer architecture. They capture diminishing marginal returns and substitution effects in ways that linear Leontief preferences, which prior work uses [15], cannot.

Example with Cache and Memory. We use a recurring example to illustrate the allocation of multiple resources given Cobb-Douglas preferences. Consider processor cache size and memory bandwidth. Agents see diminishing marginal returns from larger caches since software tasks exhibit limited exploitable locality. Depending on its data access locality, software tasks can substitute cache size for memory bandwidth and vice versa.

Suppose a system has 24GB/s of memory bandwidth and 12MB cache. This setting is representative of a quad-core processor with two DDRx channels. The system is shared by two users or agents. Let (x_1, y_1) denote the memory bandwidth and cache capacity allocated to the first user. Similarly, let (x_2, y_2) denote the second user's allocation. Suppose users' utilities are described by Equation (2).

$$u_1 = x_1^{0.6} y_1^{0.4} \qquad u_2 = x_2^{0.2} y_2^{0.8} \quad (2)$$

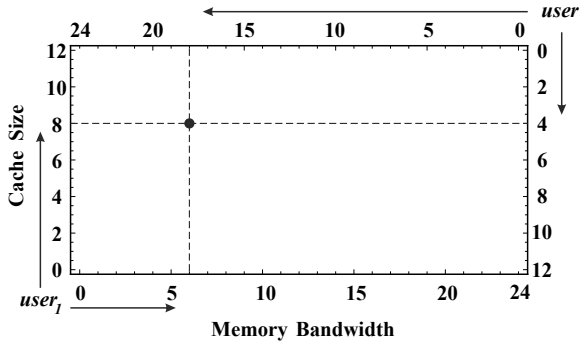


Figure 1. Edgeworth Box Example. Box height shows total cache size and box width shows total memory bandwidth. User 1's origin is at the lower left corner and User 2's origin is at the upper right corner. Each point in this box corresponds to a feasible resource allocation to users.

User 1 runs an application that exhibits bursty memory activity but little data re-use. For user 1, memory bandwidth x_1 is more useful than cache capacity y_1. In contrast, user 2 makes good use of its cache capacity y_2. We use profilers and regression to derive these utility functions (§4.4).

Software behavior translates into hardware demands, which in turn are reflected in the utility functions. These utility functions are representative of realistic applications. For example, u_1 and u_2 accurately model the relative cache and memory intensities for `canneal` and `freqmine` from the PARSEC benchmarks (§5).

Visualization with Edgeworth Boxes. To visualize feasible resource allocations, we use the Edgeworth box [11]. Figure 1 illustrates the allocation of two resources to two users. User 1's origin is at the lower left corner and User 2's origin is at the upper right corner. The total amount of cache is the height of the box and the total amount of memory bandwidth is the width. Therefore, each feasible allocation of resources can be represented as a point in the Edgeworth box. If user 1 gets 6GB/s memory bandwidth and 8MB cache, user 2 is left with 18GB/s memory bandwidth and 4MB cache.

The Edgeworth box includes all possible allocations. But only some of these allocations are fair. And only some of these provide sharing incentives. Thus, desired game-theoretic properties (sharing incentives, envy-freeness, and Pareto efficiency) define constraints on the allocation space. We use the Edgeworth box to visualize these constraints, beginning with sharing incentives.

3.1 Sharing Incentives (SI)

Sharing hardware is essential to increasing system utilization and throughput. An allocation mechanism should provide sharing incentives (SI) such that agents are at least as happy as they would be under an equal split of the resources. Without SI, users would prefer to partition hardware equally. But an equal partitioning would not reflect software diversity

and heterogeneous hardware demands. Resources may be mis-allocated, leaving throughput unexploited.

Formally, let C_r denote the total capacity of resource r in the system. Suppose an allocation mechanism provides agent i with resources $x_i = (x_{i1}, \ldots, x_{iR})$. For a system with N users, this mechanism provides SI if

$$(x_{i1}, \ldots, x_{iR}) \succsim_i \left(\frac{C_1}{N}, \ldots, \frac{C_R}{N}\right) \tag{3}$$

for each agent $i \in [1, N]$. In other words, each agent weakly prefers its allocation of hardware to an equal partition.

Whether an allocation is preferred depends on the utility functions. Consider our example with cache size and memory bandwidth. User 1 compares its allocation (x_1, y_1) against equally splitting 24GB/s of bandwidth and 12MB of cache. If user 1 always weakly prefers (x_1, y_1), then the allocation mechanism provides user 1 an incentive to share.

$$x_1^{0.6} y_1^{0.4} \geq \left(\frac{24\text{GB/s}}{2}\right)^{0.6} \left(\frac{12\text{MB}}{2}\right)^{0.4} \tag{4}$$

$$x_2^{0.2} y_2^{0.8} \geq \left(\frac{24\text{GB/s}}{2}\right)^{0.2} \left(\frac{12\text{MB}}{2}\right)^{0.8} \tag{5}$$

In our example with two agents, Equations (4)–(5) must be satisfied. User 1 must receive allocations that satisfy Equation (4). Simultaneously, user 2 must receive allocations that satisfy Equation (5). A mechanism that provides SI will identify allocations that satisfy both constraints.

3.2 Envy-Freeness (EF)

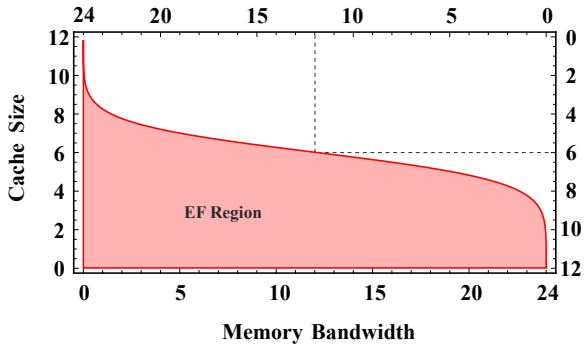
Envy is the resentment of another agent's allocation combined with a desire to receive that same allocation. Allocations are envy-free (EF) if no agent envies another. Such allocations are considered equitable and equity is a game-theoretic requirement for fairness [37].

Specifically, suppose agent i is allocated x_i. This allocation is EF if agent i prefers its allocation to any other agent's allocation and has no desire to swap. That is, $x_i \succsim_i x_j, \forall j \neq i$. In this comparison, each agent considers herself in the place of other agents and evaluates their allocations in the same way she judges her own allocation.

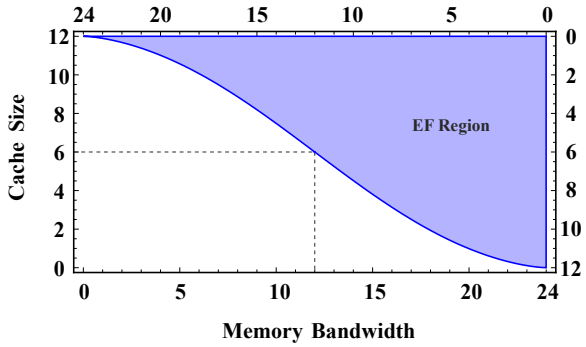
In our cache and bandwidth example, the EF allocations for user 1 are those for which $u_1(x_1, y_1) \geq u_1(x_2, y_2)$. Note that $(x_2, y_2) = (24 - x_1, 12 - y_1)$. Thus, allocations that satisfy Equation (6) are EF for user 1. And Figure 2(a) illustrates regions in which these allocations are found. Similarly, Equation (7) and Figure 2(b) describe the set of EF allocations for user 2. A mechanism that satisfies EF will identify allocations that satisfy both constraints.

$$x_1^{0.6} y_1^{0.4} \geq (24 - x_1)^{0.6} (12 - y_1)^{0.4} \tag{6}$$

$$x_2^{0.2} y_2^{0.8} \geq (24 - x_2)^{0.2} (12 - y_2)^{0.8} \tag{7}$$



(a) Envy-free Allocations for User 1



(b) Envy-free Allocations for User 2

Figure 2. Visualizing Envy-freeness (EF). EF allocations for user 1 satisfy $x_1^{0.6} y_1^{0.4} \geq (24 - x_1)^{0.6} (12 - y_1)^{0.4}$. Those for user 2 satisfy $x_2^{0.2} y_2^{0.8} \geq (24 - x_2)^{0.2} (12 - y_2)^{0.8}$. The mid-point (12GB/s, 6MB) and two corners (24GB/s, 0MB), and (0GB/s, 12MB) are always EF.

There are always at least three EF allocations, which are illustrated by the middle point and two corner points. The middle point corresponds to the situation in which all resources all equally divided between users. No user envies the other.

The corners correspond to situations in which all of one resource is given to one user and all of the other resource is given to the other. Both users derive zero utility and do not envy each other. In our example, the two corner allocations are (0GB/s, 12MB) and (24GB/s, 0MB). Users derive zero utility because both cache and memory are required for computation.

None of these obvious EF allocations is attractive. The middle point divides resources equally without accounting for differences in user utility. In this setting, system throughput could likely be improved. And corner points are clearly not useful. Thus, we need a mechanism to identify more effective EF allocations.

3.3 Pareto Efficiency (PE)

Pareto efficiency (PE) is another game-theoretic property that must be satisfied by a fair resource allocation [37]. An allocation is PE if increasing one user's utility necessarily

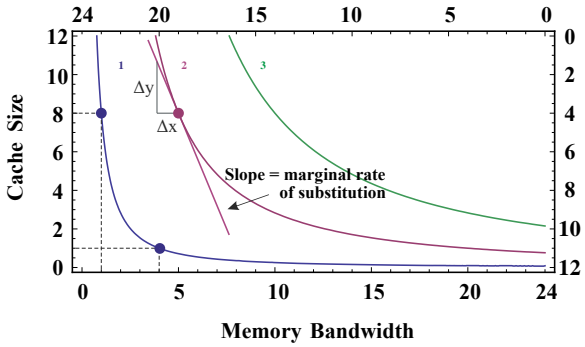


Figure 3. Cobb-Douglas Indifference Curves. On a given indifference curve, allocations provide the same utility. On different curves, utility of I_3 is greater than that of I_1. Slopes illustrate the marginal rate of substitution.

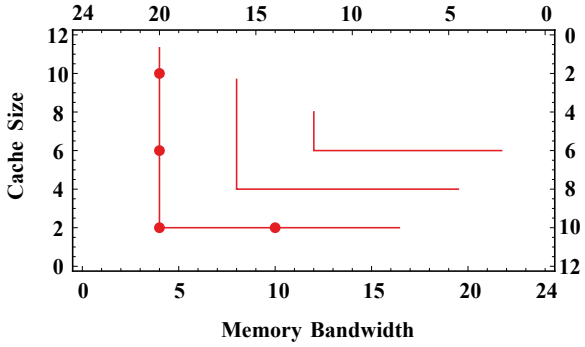


Figure 4. Leontief Indifference Curves. Resources are perfect complements and the marginal rate of substitution is either zero or infinity.

decreases another's utility. If an allocation is not PE, there exists another allocation that should have been chosen to improve total system utility.

More precisely, consider an allocation $x = (x_1, \ldots, x_N)$ for N agents. Allocation x is PE if there exists no other feasible allocation x' that all agents i weakly prefer ($x'_i \succsim_i x_i$) and at least one agent j strictly prefers ($x'_j \succ_j x_j$). Finding PE allocations is inherently linked to navigating trade-offs between substitutable resources.

Substitution Effects. An indifference curve depicts the allocations that are substitutable for one another. Figure 3 shows three indifference curves for user 1. Allocations on the same curve provide the same utility. Allocations on different curves provide different utilities. The utility of I_1 is less than that of I_2, and the utility of I_2 is less than that of I_3. Therefore, all allocations on I_2 and I_3 are strictly preferred to those on I_1.

The Leontief preferences used in prior work do not permit substitution [15]. Suppose user 1 demands 2GB/s of memory bandwidth and 1MB of cache. With this demand vector, the user's Leontief utility function is shown in Equation (8).

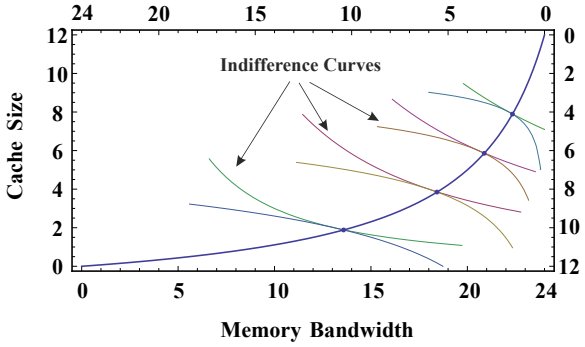


Figure 5. Visualizing Pareto Efficiency (PE). The contract curve includes all PE allocations for which the marginal rate of substitution (MRS) for both utility functions are equal.

Under Leontief, resources are perfect complements, leading to the L-shaped indifference curves in Figure 4.

$$u_1 = min\{x_1, 2y_1\} \tag{8}$$

User 1 demands bandwidth and cache in a 2:1 ratio. If the allocated ratio differs, then extra allocated resources are wasted. For example, user 1 derives the same utility from (4GB/s, 2MB) as it does from disproportional allocations such as (10GB/s, 2MB) or (4GB/s, 10MB). Leontief preferences do not account for marginal benefits from disproportional allocations. Nor do they allow for substitution in which more cache capacity compensates for less memory bandwidth.

In contrast, substitution is modeled by Cobb-Douglas preferences as illustrated by indifference curves' slopes in Figure 3. For instance, user 1 can substitute an allocation of (4GB/s, 1MB) for an allocation of (1GB/s, 8MB). Such flexibility provides the allocation mechanism with more ways to provide the same utility, which is particularly important as the set of feasible allocations are constrained by the conditions for SI, EF, and PE.

Marginal Rates of Substitution. The marginal rate of substitution (MRS), is the rate at which the user is willing to substitute one resource for the other. Visually, the MRS is the slope of the indifference curve. If MRS=2, the user will give up two units of y for one unit of x. Under Leontief preferences, the MRS is either zero or infinity; the user has no incentive for substitution. But under Cobb-Douglas preferences, the MRS is more interesting. In our cache and bandwidth example, the marginal rate of substitution for user 1 is given by Equation (9).

$$MRS_{1,xy} = \frac{\partial u_1/\partial x_1}{\partial u_1/\partial y_1} = \left(\frac{0.6}{0.4}\right)\left(\frac{y_1}{x_1}\right) \tag{9}$$

For any PE allocation, the MRS for the two users must be equal. Visually, this means users' indifference curves are tangent for PE allocations. Suppose curves were not tangent

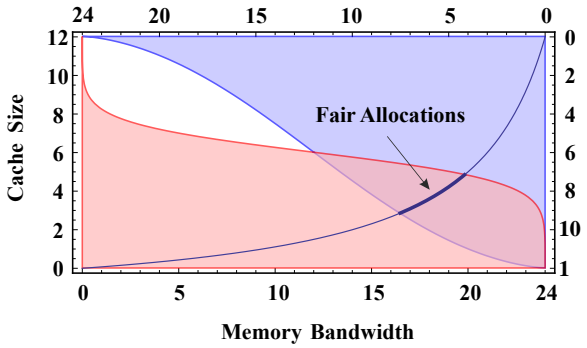


Figure 6. Fair Allocation Set All the points on the intersection of envy-free sets and the contract curve correspond to the fair allocations.

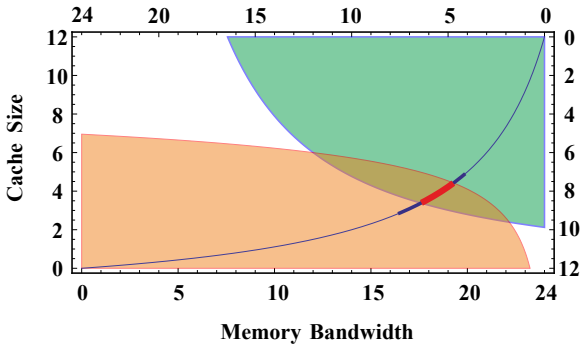


Figure 7. Visualizing Sharing Incentives (SI). Satisfying the sharing incentive property limits the set of feasible fair allocations.

for a particular allocation. Then a user i could adjusts its allocation and travel along its indifference curve, substituting resources based on its MRS without affecting u_i. But the substitution would take the other user to a higher utility.

The MRS determines the contract curve, which shows all PE allocations. Figure 5 shows the contract curve and illustrates tangency for three allocations. With the tangency condition, formal conditions for PE is easily formulated. In our example, allocations (x_1, y_1) and (x_2, y_2) are PE if the users' marginal rates of substitution are equal; Equation (10) must be satisfied.

$$\left(\frac{0.6}{0.4}\right)\left(\frac{y_1}{x_1}\right) = \left(\frac{0.2}{0.8}\right)\left(\frac{y_2}{x_2}\right) \tag{10}$$

As seen in Figure 5, both origins are PE allocations. At these points, one user's utility is zero and the other's is maximized. Increasing a user's utility, starting from zero, necessarily decreases the other user's utility. While PE, these allocations are neither desirable nor fair. The user with zero utility envies the other user's allocation. Thus, we need a mechanism that identifies both PE and EF allocations.

4. Resource Elasticity Fairness (REF)

We present a fair allocation mechanism that satisfies three game-theoretic properties: sharing incentives (SI), envy-freeness (EF), and Pareto efficiency (PE). We begin with the space of possible allocations. We then add constraints to identify allocations with the desired properties.

Economic game theory defines a fair allocation as one that is equitable (EF) and efficient (PE) [37]. Figure 6 illustrates the effect of these constraints. Each user identifies its EF allocations. And the contract curve identifies PE allocations. The intersection of these three constraints define feasible, fair allocations. Figure 7 shows that SI further constrains the set of fair allocations.

Formally, finding fair multi-resource allocations given Cobb-Douglas preferences can be modeled as the following feasibility problem for N agents and R resources.

$$\begin{aligned}
&\text{find} && x && \\
&\text{subject to} && u_i(x_i) \geq u_i(x_j) && i,j \in [1,N] \\
& && \frac{\alpha_{ir}}{\alpha_{is}}\frac{x_{is}}{x_{ir}} = \frac{\alpha_{jr}}{\alpha_{js}}\frac{x_{js}}{x_{jr}} && i,j \in [1,N]; r,s \in [1,R] \\
& && u_i(x_i) \geq u_i(C/N) && i \in [1,N] \\
& && \sum_{i=1}^{N} x_{ir} \leq C_r, && r \in [1,R]
\end{aligned} \tag{11}$$

where C/N is $(C_1/N, \ldots, C_R/N)$. In this formulation, the four constraints enforce EF, PE, SI, and capacity.

4.1 Procedure for Fair Allocation

To solve the multi-resource allocation problem, we present a mechanism to determine each agent's fair share of the hardware. N agents share R resources. For each agent i, we determine its allocation $x_i = (x_{i1}, \ldots, x_{iR})$ with the following procedure, which satisfies all constraints in Equation (11).

- **Fit Cobb-Douglas Utility.** Profile and characterize agent i's performance for various resource allocations. Fit a Cobb-Douglas utility function $u_i(x_i) = \alpha_{i0} \prod_{r=1}^{R} x_{ir}^{\alpha_{ir}}$.
- **Re-scale Elasticities.** Parameters α in the Cobb-Douglas utility function are known as elasticities. For each agent i, re-scale its elasticities so that they sum to one.

$$\hat{\alpha}_{ir} = \frac{\alpha_{ir}}{\sum_{r=1}^{R} \alpha_{ir}} \tag{12}$$

- **Re-scale Utilities.** Redefine the Cobb-Douglas utility function with re-scaled elasticities $\hat{u}_i(x_i) = \prod_{r=1}^{R} x_{ir}^{\hat{\alpha}_{ir}}$.
- **Allocate in Proportion to Elasticity.** Examine re-scaled Cobb-Douglas utilities and use their elasticities to determine fair share for each agent i and resource r.

$$x_{ir} = \frac{\hat{\alpha}_{ir}}{\sum_{j=1}^{N} \hat{\alpha}_{jr}} \times C_r \tag{13}$$

In effect, this allocation mechanism quantifies elasticity α to determine the extent each resource improves an agent's utility. Re-scaling elasticities allows us to compare values for different agents on the same scale. By allocating in proportion to elasticity, agents that benefit more from resource r will receive a larger share of the total C_r.

In our cache and bandwidth example, two users provide Cobb-Douglas utility functions with elasticities. These elasticities are already scaled and sum to one (e.g., $u_1 = x_1^{0.6} y_1^{0.4}$). To determine the memory bandwidth allocation, we examine both user's bandwidth elasticity ($\alpha_{1x} = 0.6, \alpha_{2x} = 0.2$) and allocate proportionally.

$$x_1 = \left(\frac{0.6}{0.8}\right) \times 24 = 18\text{GB/s}, \quad y_1 = \left(\frac{0.4}{1.2}\right) \times 12 = 4\text{MB}$$
$$x_2 = \left(\frac{0.2}{0.8}\right) \times 24 = 6\text{GB/s}, \quad y_2 = \left(\frac{0.8}{1.2}\right) \times 12 = 8\text{MB}$$

4.2 Fairness and Sharing Incentives

The proportional elasticity mechanism has several attractive properties. The mechanism promotes sharing and guarantees fairness by satisfying conditions for SI, EF, and PE. We sketch the proofs for these properties.

First, we show that the allocation is a Nash bargaining solution. Observe that the allocation from Equation (13) is equivalent to finding an allocation that maximizes the product of re-scaled utilities $\hat{u}$. This equivalence can be shown by substituting re-scaled Cobb-Douglas utility functions into Equation (14) and using Lagrange multipliers for constrained optimization.

$$\max \prod_{i=1}^{N} \hat{u}_i(x_i) \quad \text{subject to} \quad \sum_{i=1}^{N} x_{ir} \leq C_r \tag{14}$$

In game theory, the bargaining problem asks how agents should cooperate to produce Pareto efficient outcomes. Nash's solution is to maximize the product of utilities [27, 30], which is equivalent to Equation (14) and our allocation mechanism. Thus, our mechanism produces an allocation that is also a Nash bargaining solution.

Next, we show that our allocation is also a Competitive Equilibrium from Equal Outcomes (CEEI), a well-known microeconomic concept for fair division. In CEEI, users are initially assigned equal resource allocations. Based on user preferences, prices are assigned to resources such that users trade and the market clears to produce an allocation.

The CEEI solution picks precisely the same allocation of resources as the Nash bargaining solution for homogeneous utility functions [26]. Let $x = (x_1, \ldots, x_R)$ be a vector of resources. Utility function u is homogeneous if $u(kx) = ku(x)$ for some constant k. Our re-scaled Cobb-Douglas utilities are homogeneous because $\sum_{r=1}^{R} \hat{\alpha}_r = 1$. For this reason, our allocation is a solution to both the Nash bargaining problem and CEEI.

Finally, a CEEI allocation is known to be fair, satisfying both EF and PE [37]. CEEI solutions also satisfy SI because users start with an equal division of resources. Users would only deviate from this initial division if buying and selling resources in the CEEI market would increase utility. Thus, users can do no worse than an equal division and CEEI provides SI.

In summary, our allocation mechanism is equivalent to the Nash bargaining solution, which is equivalent to the CEEI solution. Because the CEEI solution provides SI, EF, and PE for re-scaled Cobb-Douglas utility functions, the proportional elasticity mechanism provides these properties as well.

4.3 Fairness and Strategy-Proofness in the Large

The proportional elasticity mechanism is strategy-proof in the large. An allocation mechanism is strategy-proof (SP) if a user cannot gain by mis-reporting its utility functions. Unfortunately, SP is too restrictive a property for Cobb-Douglas utility functions. For these preferences, no mechanism can provide both PE and SP [19]. However, our mechanism does satisfy a weaker property, strategy-proofness in the large (SPL). When there are many users in the system, users have no incentive to lie about their elasticities α.

First, we define large. A large system has many users such that the sum of all agents' elasticities for any resource is much bigger than 1. In such a system, any one user's resource elasticity is small relative to the sum of all agents' elasticities for the resource. More formally, the system is large if $1 \ll \sum_j \alpha_{jr}$, for all resources r.

Next, suppose user i decides to lie about her utility function, reporting α'_{ir} instead of the true value α_{ir} for resource r. Given other users' utilities, user i would choose to report the α'_{ir} that maximizes her utility.

$$\frac{\partial}{\partial \alpha'_{ik}} \prod_{r=1}^{R} \left(\frac{\alpha'_{ir}}{\alpha'_{ir} + \sum_{j \neq i} \alpha_{jr}} C_r \right)^{\alpha_{ir}} = 0 \quad \forall k \in [1, R] \tag{15}$$

In her best scenario, user i knows all other users' utilities and $\alpha_{jr}, \forall j \neq i$. Thus, by mis-reporting α'_{ir}, user i can precisely affect her proportional share of resource r. Yet, when user i receives her allocation, she evaluates it with α_{ir}, which reflects her true utility from resource r. Thus, the product in Equation (15) reflects user i's utility from lying.

User i attempts to maximize this utility from lying, taking partial derivatives with respect to α'_{ir}. But it can be proven that this optimization produces $\alpha'_{ir} \approx \alpha_{ir}$ when $1 \ll \sum_j \alpha_{jr}$ for all resources r.[1] Thus, in a large system, our allocation mechanism is approximately strategy proof. A user cannot benefit by lying about her utility.

In theory, SPL holds when an individual agents elasticity is much smaller than the sum of all agents elasticities.

[1] See §A for the proof

In practice, we find that tens of agents are sufficient to provide SPL. In other words, a strategic agent performing the optimization of Equation (15) will not deviate from her true elasticity.

For example, consider 64 tasks sharing a large system. This is a realistic setting since modern servers can have four processor sockets (= 64 threads) that share eight-twelve memory channels (> 100 GB/s of bandwidth). Suppose each of the 64 tasks elasticities are uniformly random from (0,1). We analyze Equation (15) and find that SPL holds.

4.4 Implementing the Mechanism

To implement the proportional elasticity mechanism, we need Cobb-Douglas utilities. We describe the process for deriving these utilities based on performance profiles and statistical regression. We also describe how proportional shares can be enforced by leveraging known resource schedulers.

Profiling Performance. Suppose a user derives utility from performance. Without loss of generality, we measure performance as the number of instructions committed per cycle (IPC). Execution time, speed-ups over a baseline, and energy efficiency would all exhibit similar trends.

The user profiles its performance as a function of allocated resources. These profiles reveal the rate of diminishing returns and identify resource substitutability. For example, the user samples from the allocation space to determine sensitivity to cache size and memory bandwidth. These profiles provide the data needed to derive utilities.

Performance can be profiled in several ways. First, consider off-line profiling in which a user runs software while precisely varying the available hardware. For example, a user can co-locate its task with synthetic benchmarks that exert tunable pressure on the memory hierarchy [25]. Thus, profiles would quantify cache and bandwidth sensitivity.

Also off-line, the user might rely on cycle-accurate, full-system simulators. These simulators combine virtual machines, such as QEMU, with hardware timing models to accurately model processor and memory [33, 35]. Simulated and physical hardware may report different performance numbers. But simulators can accurately report trends and elasticities, identifying hardware resources that are more important for performance. We value relative accuracy over absolute accuracy when profiling hardware preferences.

Finally, consider on-line profiling. Without prior knowledge, a user assumes all resources contribute equally to performance. Such a naive user reports utility $u = x^{0.5}y^{0.5}$. As the system allocates for this utility, the user profiles software performance. And as profiles are accumulated for varied allocations, the user adapts its utility function.

Fitting Cobb-Douglas Utility. Given performance profiles for varied hardware allocations, each user fits her Cobb-Douglas utility function in the form of $u = \alpha_0 \prod_{r=1}^{R} x_r^{\alpha_r}$. For example, let u be IPC, let x_1 be cache capacity, and let x_2 be memory bandwidth.

Fitting the utility function means identifying elasticities $\alpha = (\alpha_0, \ldots, \alpha_R)$ that best relate performance to the resources. We fit α with regression. Specifically, we apply a log transformation to linearize Cobb-Douglas. After this transformation, we have a standard linear model with parameters α as shown in Equation (16). Parameters are fit with least squares.

$$\log(u) \quad = \quad \log(\alpha_0) + \sum_{r=1}^{R} \alpha_r \log(x_r) \qquad (16)$$

Allocating Proportional Shares. We re-scale elasticities from each user's Cobb-Douglas utility function and compute proportional shares. The novelty of our mechanism is not in proportional sharing but in how we identify the proportions based on Cobb-Douglas elasticities to ensure SI, EF, and PE. After the procedure determines proportional shares for each user, we can enforce those shares with existing approaches, such as weighted fair queuing [8] or lottery scheduling [38].

4.5 Alternative Fair Mechanisms

There may exist multiple allocations x that satisfy the fairness conditions in Equation (11). Our mechanism for proportional elasticity is only one possible mechanism for one possible solution. Alternative mechanisms may also produce fair allocations but increase computational complexity. Suppose we follow prior work in computer architecture and seek fair allocations that maximize system throughput.

To evaluate throughput for a multi-programmed system, architects define the notion of weighted progress, which divides each application's multi-programmed IPC by its single-threaded IPC [12]. Weighted system throughput is the sum of each user's weighted progress. This is the metric used to evaluate prior work on memory scheduling and multiprocessor resource management [4, 29].

$$\sum_{i=1}^{N} \frac{\text{IPC}(x_i)}{\text{IPC}(C)} \quad \approx \quad \sum_{i=1}^{N} \frac{u_i(x_i)}{u_i(C)} \quad = \quad \sum_{i=1}^{N} U(x_i) \qquad (17)$$

We adapt this notion of normalized throughput, expressing it in terms of our utility functions. This means dividing utility for an allocation in the shared machine $u_i(x_i)$ by utility when given all of the machine's capacity $u_i(C)$. Let $U(x_i) = u_i(x_i)/u_i(C)$ define the notion of weighted utility, which is equivalent to the notion of slowdown in prior work [4, 29].

Fair Allocation for Utilitarian Welfare. Rather than allocate in proportion to elasticities, we could allocate to maximize utilitarian welfare. Instead of finding x subject to fairness conditions in Equation (11), we would optimize $\max \sum_i U_i(x_i)$ subject to the same conditions. While $\max \sum_i U_i(x_i)$ is computationally intractable, $\max \prod_i U_i(x_i)$

is similar but tractable with geometric programming.[2] But this mechanism would be more computationally demanding than our closed-form solution in Equation (13).

Yet a utilitarian mechanism is interesting. Overall system performance is an explicit optimization objective. A utilitarian mechanism likely provides the allocation that achieves the highest performance among all fair allocations. In effect, utilitarian allocations provides an empirical upper bound on fair performance.

Fair Allocation for Egalitarian Welfare. We could also find fair allocations to optimize egalitarian welfare. In Equation (11), we would optimize max-min $U_i(x_i)$ subject to fairness conditions. As before, geometric programming can perform this optimization but this mechanism would be more computationally demanding than our closed-form solution.

Egalitarian welfare is interesting because it optimizes for the least satisfied user. EF and PE define conditions for a fair allocation. But these conditions say nothing about equality in outcomes. An allocation could be fair but the difference between the most and least satisfied user in the system could be large. The max-min optimization objective mitigates inequality in outcomes, perhaps at the expense of system welfare. Egalitarian allocations might provide an empirical lower bound on fair performance.

Unfair Allocation. Finally, we could neglect game-theoretic fairness and ignore constraints imposed by SI, EF, and PE. In this setting, we would maximize welfare subject only to capacity constraints. Note that optimizing egalitarian welfare without fairness conditions is equivalent to the objective in prior work [29], which equalizes users' weighted progress such that $\max_i U_i(x_i) \ / \ \min_j U_j(x_j) \rightarrow 1$. The max-min objective for egalitarian welfare causes the denominator to approach the numerator. Assessing performance of unfair allocations reveals the penalty we must pay for SI, EF, and PE.

5. Evaluation

We evaluate the proportional elasticity mechanism when sharing the last-level cache and main memory bandwidth in a chip-multiprocessor. In this setting, we evaluate several aspects of the mechanism. First, we show that Cobb-Douglas utilities are a good fit for performance. Then, we interpret utility functions to identify applications that prefer cache capacity (C) and memory bandwidth (M).

Finally, we compare the proportional elasticity mechanism against an equal slowdown mechanism, which represents conventional wisdom. We find that equal slowdown fails to guarantee game-theoretic fairness. On the other hand, proportional elasticity guarantees SI, EF and PE with only modest performance penalties relative to an unfair approach.

[2] Cobb-Douglas is a monomial function (i.e., function with the form $f(x) = ax_1^{\alpha_1} x_2^{\alpha_2}, \ldots, x_m^{\alpha_m}$). And geometric programming can maximize monomials [5].

Table 1. Platform Parameters

Component	Specification
Processor	3 GHz OOO cores, 4-width issue and commit
L1 Cache	32 KB, 4-way set associative, 64-byte block size, 2-cycle latency
L2 Cache	[128 KB, 256 KB, 512 KB, 1 MB, 2 MB], 8-way set associative, 64-byte block size, 20-cycle latency
DRAM Controller	Closed-page, Queue per rank, Rank then bank round-robin scheduling
DRAM Bandwidth	[0.8 GB/s, 1.6 GB/s, 3.2 GB/s, 6.4 GB/s, 12.8 GB], single channel

5.1 Experimental Methodology

Simulator. We simulate the out-of-order cores using the MARSSx86 full system simulator [33]. We integrate the processor model with the DRAMSim2 simulator [35] to simulate main memory. To characterize application sensitivity to allocated cache size and memory bandwidth, we simulate 25 architectures spanning combinations of five cache sizes and five memory bandwidths. The platform parameters are described in Table 1.

Given simulator data, we use Matlab to fit Cobb-Douglas utility functions. Our mechanism includes a closed-form expression for each agent's fair allocation. But to evaluate other mechanisms that require geometric programming, we use CVX [16], a convex optimization solver.

Workloads. We evaluate our method on 24 benchmarks from PARSEC and SPLASH-2x suites [2]. We further evaluate applications from the Phoenix system for MapReduce programming [34], including histogram, linear regression, string match, and word count. For PARSEC 3.0 benchmarks, we simulate 100M instructions from the regions of interest (ROI), which are representative application phases identified by MARSSx86 developers. Phoenix applications we simulate 100M instructions from the beginning of the map phase.

5.2 Fitting Cobb-Douglas Utility

Each application is associated with a user. Application performance is measured as instructions per cycle (IPC). Using cycle-accurate simulations, we profile each benchmark's performance. Given these profiles for varied cache size and memory bandwidth allocations, we perform a linear regression to estimate utility functions.

For each application, we use a Cobb-Douglas utility function $u = \alpha_0 x^{\alpha_x} y^{\alpha_y}$ where u is application performance measured with IPC, x is memory bandwidth, and y is cache size. Although a non-linear relationship exists between Cobb-Douglas utility and resource allocations, a logarithmic transformation produces a linear model (Equa-

tion (16)). Least squares regression estimates the resource elasticities α for each benchmark.

To evaluate this fit, we report the coefficient of determination (R-squared), which measures how much variance in the data set is captured by the model. R-squared $\rightarrow 1$ as fit improves. Figure 8(a) shows that most benchmarks are fitted with R-squared of 0.7-1.0, indicating good fits. Benchmarks with low R-squared, such as `radiosity`, have negligible variance and no trend for Cobb-Douglas to capture.

We consider representative workloads with high and low R-squared values in Figure 8, which plots simulated and fitted IPC. Cobb-Douglas utilities accurately track IPC and reflect preferences for cache and memory bandwidth. Even workloads with lower R-squared values, such as `radiosity`, do not deviate significantly from true values.

In practice, the proportional elasticity mechanism never uses the predicted value for u to allocate hardware. It only uses the fitted parameters for α to determine fair shares. Thus, Cobb-Douglas fits need only be good enough to assess resource elasticities and preferences. But good predictions for u give confidence in the accuracy of fitted α.

We expect Cobb-Douglas utility functions to generalize beyond cache size and memory bandwidth. After applying log transformations to performance and each of the resource allocations, our approach to fitting the utility function is equivalent to prior work in statistically inferred microarchitectural models [23]. Prior work accurately inferred performance models with more than ten microarchitectural resources, which suggests our application of Cobb-Douglas utilities will scale as more resources are shared.

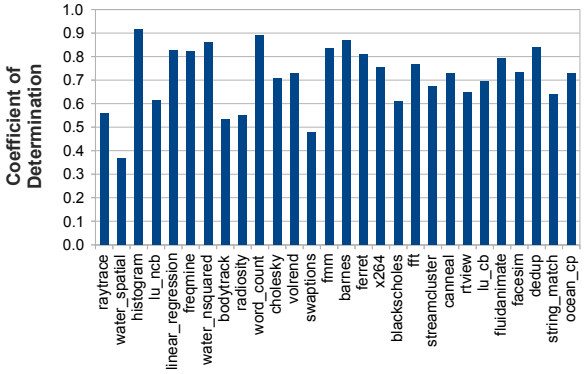
5.3 Interpreting Cobb-Douglas Utilities

After fitting Cobb-Douglas utilities, we re-scale elasticities as described in Equation (12). Resource elasticity quantifies the extent to which an agent demands a resource. In other words, in a multi-resource setting, elasticities quantify the relative importance of each resource to an agent.

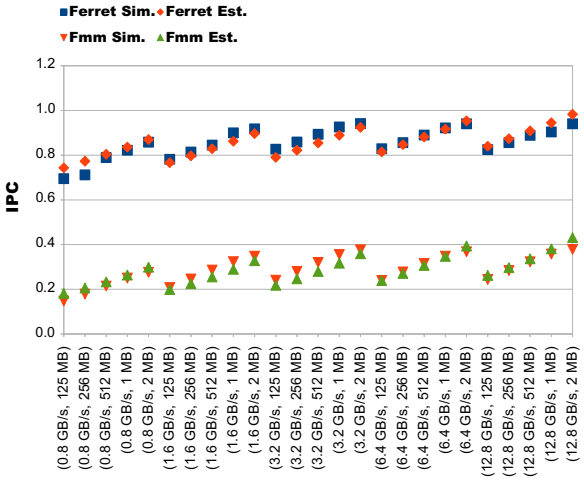
Figure 9 depicts re-scaled elasticities for our workloads. If $\alpha_{cache} > \alpha_{mem}$, then the workload derives more utility from cache size than it does from memory bandwidth (e.g., `raytrace`). In contrast, if $\alpha_{mem} > \alpha_{cache}$, then the workload finds memory bandwidth more useful (e.g., `dedup`).

Given resource elasticities, we can classify workloads into two groups. Workloads in group M demand memory bandwidth and $\alpha_{mem} > 0.5$. Workloads in group C demand cache capacity and $\alpha_{cache} > 0.5$. This classification differentiates how workloads re-use data in their cache and whether they exhibit bursty memory behavior.

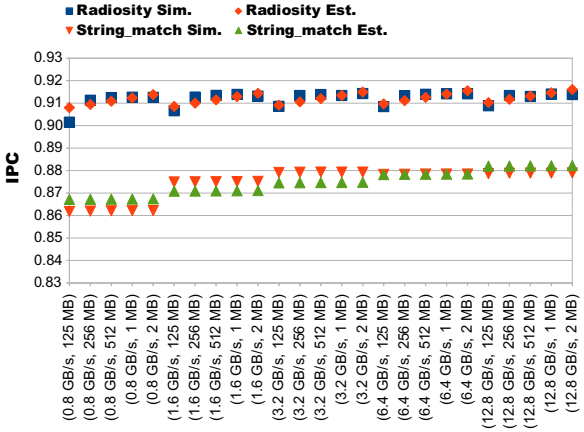
For example, `facesim`, `fluidanimate`, and `streamcluster` exhibit streaming behavior [3]. Increasing the cache size would only marginally increase performance. Streaming workloads clearly prefer memory bandwidth and this preference is reflected in their resource elasticities in Figure 9.



(a) Coefficient of determination (R-squared) measures goodness of fit for Cobb-Douglas utility functions. Larger values are better.



(b) Simulated versus fitted Cobb-Douglas performance for varied cache size, memory bandwidth allocations. Representative workloads with high R-squared.



(c) Simulated versus fitted Cobb-Douglas performance for varied cache size, memory bandwidth allocations. Representative workloads with low R-squared.

Figure 8. Evaluating Cobb-Douglas utilities for cache size and memory bandwidth. $u = \alpha_0 x^{\alpha_x} y^{\alpha_y}$ where u is IPC performance, x is memory bandwidth, and y is cache size. Cobb-Douglas is fit by finding α with method of least squares.

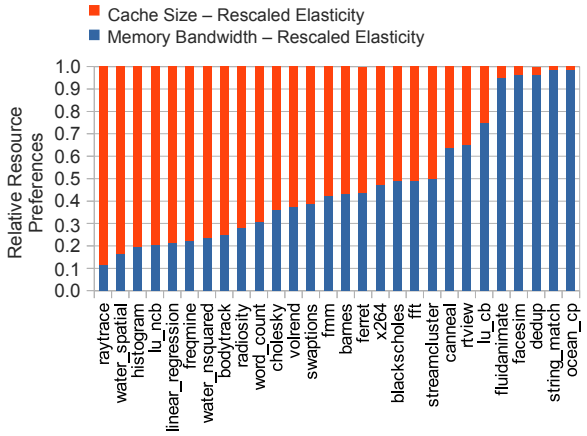


Figure 9. Resource Preferences and Elasticities. Rescaled elasticities from Equation (12) show relative importance of cache size and memory bandwidth for each workload. Workloads for which $\alpha_{mem} > 0.5$ are classified M. Otherwise, they are classified C.

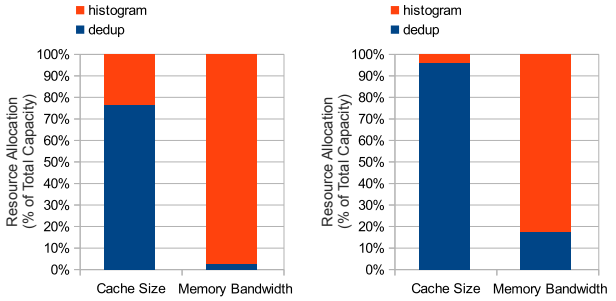


Figure 10. Allocations for histogram (C) and dedup (M) show that equal slowdown can satisfy SI, EF, and PE in some scenarios.

5.4 Proportional Elasticity versus Equal Slowdown

Having demonstrated accurate Cobb-Douglas utility models, we now evaluate our mechanism that allocates in proportion to elasticity. We compare against a mechanism that allocates for equal slowdown, a commonly used approach in computer architecture that seeks to equally distribute the performance penalties from sharing [29], [4].

We compare proportional elasticity and equal slowdown with a series of representative examples. In the first example, all desirable properties (SI, EF, PE) are satisfied by both proportional elasticity and equal slowdown. But an equal slowdown mechanism cannot guarantee these properties. We present two other examples where both SI and EF are violated by an equal slowdown mechanism.

Example 1: C-M satisfies SI, EF, PE. Consider a system shared by `histogram` from group C and `dedup` from group M, which prefer cache capacity and memory bandwidth, respectively. Figure 10 illustrates allocations as a percentage

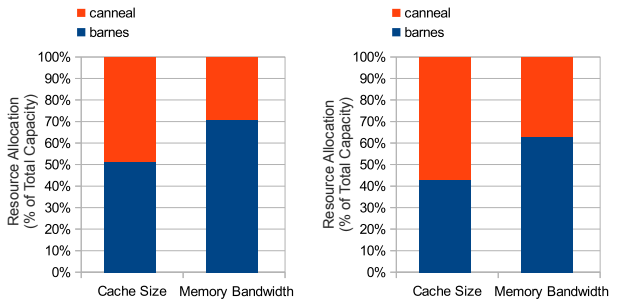


Figure 11. Allocations for barnes (C) and canneal (M) show that equal slowdown can fail to satisfy SI and EF. Equal slowdown provides canneal less than half of both resources, which satisfies neither SI nor EF. Proportional elasticity satisfies SI, EF, and PE.

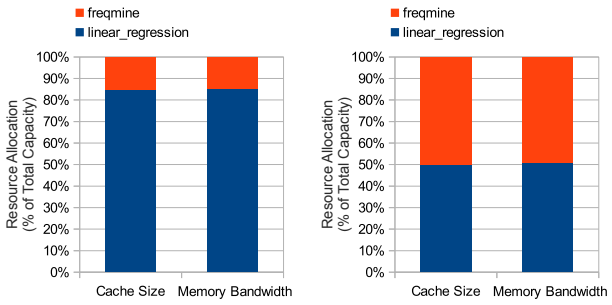
of total capacity for an equal slowdown mechanism and our proportional elasticity mechanism.

Both mechanisms allocate more cache capacity to `histogram` (C) and memory bandwidth to `dedup` (M). Consider a chip multiprocessor with 12MB cache and 24GB/s of memory bandwidth. We can compute the allocations and evaluate the conditions for SI, EF, and PE in Equation (11). In this particular case, the equal slowdown allocation satisfies all game-theoretic conditions for fairness. And, of course, we have proven that the proportional elasticity allocation is fair.

Unfortunately, while an equal slowdown mechanisms may provide SI and EF in this case, it cannot guarantee them. We cannot even generalize the properties of equal slowdown for broad classes of workloads. While equal slowdown happens to provide SI and EF for `histogram` (C) and `dedup` (M), it may not do so for other pairs of C and M workloads.

Example 2: C-M violates SI and EF. Figure 11 considers the allocations for `barnes` (C) and `canneal` (M). Barnes prefers cache size to memory bandwidth whereas cannel prefers bandwidth to cache. This example shows how an equal slowdown mechanism fails to satisfy SI and EF for `canneal`, which receives less than half of both resources in the system. In this setting, `cannel` would not be willing to participate in an equal slowdown mechanism and would rather statically receive half the hardware resources. Moreover, `canneal` envies `barnes`'s allocation. In contrast, our proportional elasticity mechanism allocates more than half of the memory bandwidth to `canneal`, giving it an incentive to share.

Example 3: C-C violates SI and EF. Finally, Figure 12 considers two workloads from the same group. In this, case `freqmine` (C) and `linear_regression` (C) both prefer cache capacity to memory bandwidth. But `freqmine` exhibits less memory activity than `linear`. To equalize slowdowns, `linear` must receive far more of both resources.



(a) Equal slowdown (b) Proportional elasticity

Figure 12. Allocations for freqmine (C) and linear_regression (C) show that equal slowdown can fail to satisfy SI and EF. Equal slowdown provides freqmine less than half of both resources, which satisfies neither SI nor EF. Proportional elasticity satisfies SI, EF, and PE.

In this setting, `freqmine` would not be willing to share the system, preferring an equal split of the resources rather than participate in an equal slowdown mechanism. Even if `freqmine` had been willing to share resources with `linear`, it would prefer `linear`'s allocation over its own. Thus, the allocation from equal slowdown is far from equitable. On the other hand, proportional elasticity divides resources almost equally between benchmarks to satisfy SI and EF.

However, proportional elasticity seems inefficient. It allocates resources equally when one user needs them more. Although the equal slowdown mechanism does not provide game-theoretic fairness, it likely provides higher system throughput in this example. Thus, in some cases, proportional elasticity pays a throughput penalty to provide game-theoretic fairness.

This trade-off between game-theoretic fairness and performance efficiency is fundamental to the mechanisms. The equal slowdown mechanism seeks to equalize normalized performance. If one more unit of a resource significantly improves `linear`'s performance and only modestly improve `freqmine`'s, the equal slowdown mechanism favors `linear`. And overall throughput should increase. In the next section, we quantify the performance penalty incurred by adding constraints for SI, EF, and PE.

5.5 Performance Penalty from Fairness

We investigate the performance lost due to game-theoretic fairness conditions. We define an agent's individual performance as $U_i(x_i) = u_i(x_i)/u_i(C)$, which divides utility when sharing by utility when not. U_i is equivalent to the notion of weighted throughput [12] except that we use utility functions rather than IPC. For each allocation policy, we compare weighted system throughput in Equation (17) calculated from utility functions fitted to simulator data.

- **Max Welfare w/o Fairness.** Find an allocation that maximizes welfare subject only to capacity constraints. We use Nash social welfare ($\prod_i U_i(x_i)$), which is tractably maximized with geometric programming. This mechanism provides an empirical upper bound on performance.
- **Equal Slowdown w/o Fairness.** Find an allocation that maximizes the minimum $U_i(x_i)$. This max-min objective function is equivalent to equalizing slowdown by closing the gap between the best and worst performing agents.
- **Max Welfare w/ Fairness.** Find an allocation that maximizes welfare subject to SI, EF, and PE conditions. We use Nash social welfare ($\prod_i U_i(x_i)$), which is tractably maximized with geometric programming.
- **Proportional Elasticity w/ Fairness.** Allocate in proportions based on elasticities in Cobb-Douglas utility functions. Allocations are proven to provide SI, EF, and PE.

Table 2. Workload Characterization

Name	Benchmarks	C/M
WD1	histogram, linear_regression, water_nsquared, bodytrack	4C
WD2	radiosity, fmm, facesim, string_match	2C-2M
WD3	lu_cb, fluidanimate, facesim, dedup	4M
WD4	fft, streamcluster, canneal, word_count	3C-1M
WD5	streamcluster, facesim, dedup, string_match	1C-3M
WD6	histogram, linear_regression, water_nsquared, bodytrack, freqmine, word_count, x264, dedup	7C-1M
WD7	histogram, canneal, rtview, bodytrack, radiosity, word_count, linear_regression, water_nsquared	6C-2M
WD8	radiosity, word_count (2), canneal, rtview, freqmine, x264, dedup	5C-3M
WD9	radiosity (2), word_count, canneal, rtview, fmm, facesim, string_match	4C-4M
WD10	water_nsquared, barnes, ferret, lu_cb (2), fluidanimate, facesim, dedup	3C-5M

Thus, we compare two allocations with and without game-theoretic fairness. Note that our mechanism is computationally trivial based on the closed-form expression in Equation (13). In contrast, the other mechanisms require geometric programming and convex optimization.

Figure 13 presents weighted system throughput when four applications share cache and memory bandwidth. The workloads have different characteristics, as shown in Table 2. Performance penalties are larger when the allocation mechanism imposes more constraints. Least restricted, maximizing welfare without any fairness constraints provides an

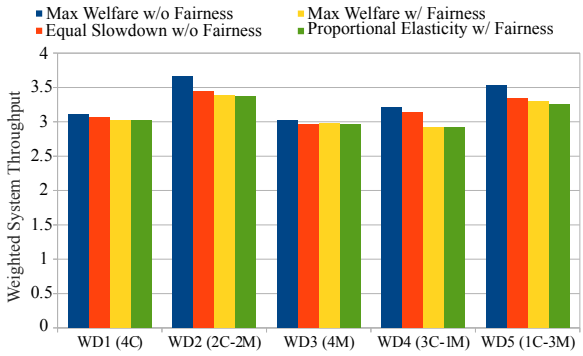


Figure 13. Performance Comparison for 4-core System. Penalties for game-theoretic fairness are less than 10%.

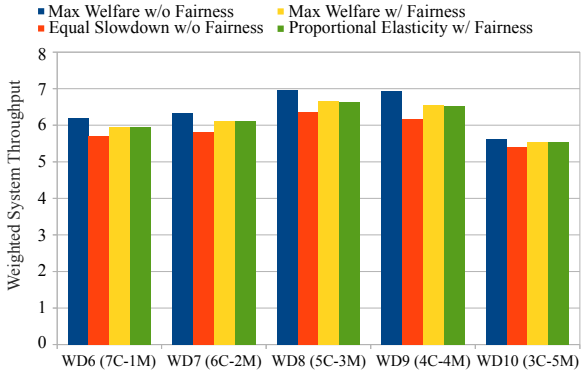


Figure 14. Performance Comparison for 8-core System. Penalties for game-theoretic fairness are less than 10%. Proportional elasticity out-performs equal slowdown.

empirical upper bound on throughput. Relative to this upper bound, equal slowdown optimizes worst-case performance, thereby lowering overall throughput. Yet, despite its lower performance, an equal slowdown mechanism does not guarantee game-theoretic fairness.

Among the two mechanisms that provide fairness with SI, EF and PE, we find no performance difference, which is a compelling result. First, our proportional elasticity mechanism is as good as explicitly optimizing throughput subject to fairness. Second, proportional elasticity provides fair performance in a complexity effective way. Our mechanism simply calculates fair shares whereas other mechanisms would require geometric programming.

The price for game-theoretic fairness is small. First, compare maximizing welfare with and without fairness. Constraints for SI, EF, and PE reduces throughput by less than 10%. Second, compare equal slowdown to proportional elasticity. With less than a 7% throughput penalty, proportional elasticity provides game-theoretic guarantees.

Figure 14 further presents throughput for an eight-core system in which eight applications share cache and memory bandwidth. We select five representative workloads to compare allocation mechanisms. In this setting, constraints for game-theoretic fairness reduce throughput by less than 10%.

More interesting, in an eight-core setting, equal slowdown may perform worse than proportional elasticity. Poor performance for an equal slowdown mechanism may be due to optimizing allocations to favor the least satisfied user (i.e., max-min u_i). As the number of users increases, the opportunity cost of favoring the least satisfied user also increases. Thus, not only does an equal slowdown mechanism fail to provide game-theoretic fairness, it may also perform worse than proportional elasticity in large systems with many agents.

6. Related Work

Computer Science and Economics. The fair resource allocation problem has been extensively studied in computer science and economics. While most prior studies focus on fairly allocating a single resource, Ghodsi et al. propose Dominant Resource Fairness (DRF) for fair, multi-resource allocation [15]. DRF satisfies SI, PE, EF and SP for Leontief preferences. Leveraging Leontief properties, Parkes et al. [32] and Joe-Wong et al. [21] extend DRF. Dolev et al. propose an alternative notion of multi-resource fairness [9]. Gutman et al. analyze fairness frameworks and present computational tractable algorithms [18].

While Leontief preferences might be appropriate for distributed systems [15], they cannot capture important trends in hardware architecture. Leontief utilities are linear and do not allow substitution between multiple resources. Moreover, specifying a demand vector for resources, which is required by DRF, is not always possible. In this paper, we consider fair, multi-resource allocation under Cobb-Douglas preferences, which are more realistic in computer architecture.

Fairness in Computer Architecture. Nesbit et al. [31] propose a memory scheduler to address fairness for a single memory resource. Other architects use an unfairness index [13, 28]. A variety of memory scheduling heuristics optimize this metric to fairly share memory bandwidth [4, 7, 10, 22, 29]. The unfairness index quantifies the ratio between the maximum and the minimum performance slowdown among workloads sharing the system. The allocation is considered fair if workloads experience equal slowdowns. However, we find that equal slowdowns cannot guarantee game-theoretic properties (e.g., SI, EF, PE).

We consider fairness in a multi-resource setting. Coordinating multi-resource allocation is more challenging due to substitution effects. Bitirgen et al. [4] consider multiple resources, relying on machine learning to predict performance for different allocations at run-time. Their objective is system throughput not fairness. Moreover, their learning technique is likely more computationally demanding than our equation for fair shares.

Resource Allocation in Datacenters. Within datacenters, market mechanisms allocate resources to maximize welfare, which is defined as user utility minus power cost [6], [24]. Guevara et. al. [17] apply market mechanisms for heterogeneous resources. Whereas these market mechanisms enhance welfare, we provide game-theoretic fairness.

7. Conclusions

Our results motivate new thinking in fairly allocating hardware resources. Rather than assume users must share hardware, we must provide allocation mechanisms to encourage sharing. We show that Cobb-Douglas utilities are well suited to modeling user preferences in computer architecture. For Cobb-Douglas utilities, we present an allocation mechanism that provides sharing incentives, envy-freeness, Pareto efficiency, and strategy-proofness in the large. By linking hardware resource management to robust, game-theoretic analysis, computer architects can qualitatively change the nature of performance guarantees in hardware platforms shared by strategic users.

A. Strategy Proofness in the Large

Suppose user i decides to lie about her utility function, reporting α'_{ir} instead of the true value α_{ir} for resource r. Given other users' utilities, user i would choose to report the α'_{ir} that maximizes her utility. As mentioned in §4, in a large system $1 \ll \sum_j \alpha_{jr}$, for all resources r. Since $\alpha'_{ir} \leq 1$, we have $\alpha'_{ir} \ll \sum_j \alpha_{jr}$. Therefore:

$$u_i = \prod_{r=1}^{R} \left(\frac{\alpha'_{ir}}{\alpha'_{ir} + \sum_{j \neq i} \alpha_{jr}} C_r \right)^{\alpha_{ir}} \approx \prod_{r=1}^{R} \left(\frac{\alpha'_{ir}}{\sum_{j \neq i} \alpha_{jr}} C_r \right)^{\alpha_{ir}} = A_i \prod_{r=1}^{R} \alpha_{ir}^{\prime \alpha_{ir}},$$

where $A_i = \prod_{r=1}^{R} C_r / \sum_{j \neq i} \alpha_{jr}$ is a constant. Let us consider the following optimization problem:

$$\begin{aligned} &\text{maximize} && \prod_r \alpha_{ir}^{\prime \alpha_{ir}} \\ &\text{subject to} && \sum_r \alpha'_{ir} = 1. \end{aligned}$$

Now, consider the Lagrangian form:

$$L(\alpha', \lambda) = \prod_r \alpha_{ir}^{\prime \alpha_{ir}} - \lambda(1 - \sum_r \alpha'_{ir}),$$

where λ is a Lagrange multiplier. Then, based on the KKT conditions:

$$\frac{\partial L}{\partial \alpha'_{ir}} = \alpha_{ir} \frac{\prod_r \alpha_{ir}^{\prime \alpha_{ir}}}{\alpha'_{ir}} - \lambda = 0 \qquad \forall r$$
$$\frac{\partial L}{\partial \lambda} = 1 - \sum_r \alpha'_{ir} = 0$$

The solution to these equations is $\alpha'_{ir} = \alpha_{ir}$, for all resources r.

Acknowledgments

We sincerely thank Vincent Conitzer for his feedback during various stages of this project. We also thank Marisabel Guevara, Blake Hechtman and the anonymous reviewers for their feedback. This work is supported by NSF grants CCF-1149252 (CAREER) and CCF-1337215 (XPS-CLCCA). This work is also supported by STARnet, a Semiconductor Research Corporation program, sponsored by MARCO and DARPA. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of these sponsors.

References

[1] Luiz André Barroso and Urs Hölzle. The datacenter as a computer: An introduction to the design of warehouse-scale machines. *Synthesis Lectures on Computer Architecture*, 4(1):1–108, 2009.

[2] Christian Bienia. *Benchmarking Modern Multiprocessors*. PhD thesis, Princeton University, January 2011.

[3] Christian Bienia, Sanjeev Kumar, Jaswinder Pal Singh, and Kai Li. The parsec benchmark suite: Characterization and architectural implications. In *Proc. International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pages 72–81, 2008.

[4] Ramazan Bitirgen, Engin Ipek, and Jose F Martinez. Coordinated management of multiple interacting resources in chip multiprocessors: A machine learning approach. In *Proc. International Symposium on Microarchitecture (MICRO)*, 2008.

[5] Stephen Boyd, Seung-Jean Kim, Lieven Vandenberghe, and Arash Hassibi. A tutorial on geometric programming. *Optimization and Engineering*, 8(1):67–127, 2007.

[6] Jeffrey S Chase, Darrell C Anderson, Prachi N Thakar, Amin M Vahdat, and Ronald P Doyle. Managing energy and server resources in hosting centers. In *Proc. Symposium on Operating System Principles (SOSP)*, 2001.

[7] Reetuparna Das, Onur Mutlu, Thomas Moscibroda, and Chita R Das. Application-aware prioritization mechanisms for on-chip networks. In *Proc. International Symposium on Microarchitecture (MICRO)*, pages 280–291. IEEE, 2009.

[8] Alan Demers, Srinivasan Keshav, and Scott Shenker. Analysis and simulation of a fair queueing algorithm. In *ACM SIGCOMM Computer Communication Review*, volume 19, pages 1–12. ACM, 1989.

[9] Danny Dolev, Dror G Feitelson, Joseph Y Halpern, Raz Kupferman, and Nathan Linial. No justified complaints: On fair sharing of multiple resources. In *Proc. Innovations in Theoretical Computer Science Conference*, pages 68–75. ACM, 2012.

[10] Eiman Ebrahimi, Chang Joo Lee, Onur Mutlu, and Yale N Patt. Fairness via source throttling: a configurable and high-performance fairness substrate for multi-core memory systems. In *Proc. International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 335–346. ACM, 2010.

[11] Francis Ysidro Edgeworth. *Mathematical psychics: An essay on the application of mathematics to the moral sciences*. Number 10. C. Kegan Paul & Company, 1881.

[12] Stijn Eyerman and Lieven Eeckhout. System-level performance metrics for multiprogram workloads. *Micro, IEEE*, 28(3):42–53, 2008.

[13] Ron Gabor, Shlomo Weiss, and Avi Mendelson. Fairness and throughput in switch on event multithreading. In *Proc. International Symposium on Microarchitecture (MICRO)*, pages 149–160. IEEE, 2006.

[14] Ron Gabor, Shlomo Weiss, and Avi Mendelson. Fairness enforcement in switch on event multithreading. *ACM Transactions on Architecture and Code Optimization (TACO)*, 4(3):15, 2007.

[15] Ali Ghodsi, Matei Zaharia, Benjamin Hindman, Andy Konwinski, Scott Shenker, and Ion Stoica. Dominant resource fairness: fair allocation of multiple resource types. In *Proc. USENIX NSDI*, 2011.

[16] Michael Grant and Stephen Boyd. Cvx: Matlab software for disciplined convex programming, version 1.21, 2010.

[17] Marisabel Guevara, Benjamin Lubin, and Benjamin C Lee. Navigating heterogeneous processors with market mechanisms. In *Proc. International Symposium on High Performance Computer Architecture (HPCA)*, pages 95–106, 2013.

[18] Avital Gutman and Noam Nisan. Fair allocation without trade. In *Proc. International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 719–728, 2012.

[19] Kazuhiko Hashimoto. Strategy-proofness versus efficiency on the cobb-douglas domain of exchange economies. *Social choice and welfare*, 31(3):457–473, 2008.

[20] Mark D Hill and Michael R Marty. Amdahl's law in the multicore era. *Computer*, 41(7):33–38, 2008.

[21] Carlee Joe-Wong, Soumya Sen, Tian Lan, and Mung Chiang. Multi-resource allocation: Fairness-efficiency tradeoffs in a unifying framework. In *Proc. INFOCOM*, pages 1206–1214. IEEE, 2012.

[22] Yoongu Kim, Dongsu Han, Onur Mutlu, and Mor Harchol-Balter. Atlas: A scalable and high-performance scheduling algorithm for multiple memory controllers. In *Proc. International Symposium on High Performance Computer Architecture (HPCA)*, pages 1–12. IEEE, 2010.

[23] Benjamin Lee and David Brooks. Accurate and efficient regression modeling for microarchitectural performance and power prediction. In *Proc. International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2006.

[24] Benjamin Lubin, Jeffrey O Kephart, Rajarshi Das, and David C Parkes. Expressive power-based resource allocation for data centers. In *Proc. International Joint Conferences on Artificial Intelligence (IJCAI)*, pages 1451–1456, 2009.

[25] Jason Mars, Lingjia Tang, Robert Hundt, Kevin Skadron, and Mary Lou Soffa. Bubble-up: Increasing utilization in modern warehouse scale computers via sensible co-locations. In *Proc. International Symposium on Microarchitecture (MICRO)*, pages 248–259. ACM, 2011.

[26] Hervé Moulin. *Fair division and collective welfare*. MIT press, 2004.

[27] Abhinay Muthoo. *Bargaining theory with applications*. Cambridge University Press, 1999.

[28] Onur Mutlu and Thomas Moscibroda. Stall-time fair memory access scheduling for chip multiprocessors. In *Proc. International Symposium on Microarchitecture (MICRO)*, pages 146–160. IEEE Computer Society, 2007.

[29] Onur Mutlu and Thomas Moscibroda. Parallelism-aware batch scheduling: Enhancing both performance and fairness of shared dram systems. In *Proc. International Symposium on Computer Architecture (ISCA)*, pages 63–74. IEEE Computer Society, 2008.

[30] John F Nash Jr. The bargaining problem. *Econometrica: Journal of the Econometric Society*, pages 155–162, 1950.

[31] Kyle J Nesbit, Nidhi Aggarwal, James Laudon, and James E Smith. Fair queuing memory systems. In *Proc. International Symposium on Microarchitecture (MICRO)*, pages 208–222. IEEE, 2006.

[32] David C Parkes, Ariel D Procaccia, and Nisarg Shah. Beyond dominant resource fairness: extensions, limitations, and indivisibilities. In *Proc. Conference on Electronic Commerce (EC)*, pages 808–825. ACM, 2012.

[33] Avadh Patel, Furat Afram, Shunfei Chen, and Kanad Ghose. MARSSx86: A Full System Simulator for x86 CPUs. In *Design Automation Conference 2011 (DAC'11)*, 2011.

[34] Colby Ranger, Ramanan Raghuraman, Arun Penmetsa, Gary Bradski, and Christos Kozyrakis. Evaluating MapReduce for multi-core and multiprocessor systems. In *Proc. International Symposium on High Performance Computer Architecture (HPCA)*, pages 13–24. IEEE, 2007.

[35] Paul Rosenfeld, Elliott Cooper-Balis, and Bruce Jacob. Dramsim2: A cycle accurate memory system simulator. *Computer Architecture Letters*, 10(1):16–19, 2011.

[36] Alejandro Toriello and Juan Pablo Vielma. Fitting piecewise linear continuous functions. *European Journal of Operational Research*, 219(1):86–95, 2012.

[37] Hal Varian. Equity, envy, and efficiency. *Journal of economic theory*, 9(1):63–91, 1974.

[38] Carl A Waldspurger and William E Weihl. Lottery scheduling: Flexible proportional-share resource management. In *Proc. USENIX Conference on Operating Systems Design and Implementation (OSDI)*, page 1. USENIX Association, 1994.