Sharing Incentives and Fair Division for Multiprocessors

The trend in datacenter computing is toward large, shared hardware platforms, which poses two challenges to architects: sharing fairly and sharing multiple resources. Drawing on economic game theory, the authors rethink fairness in computer architecture and propose Resource Elasticity Fairness to find fair allocations that ensure sharing incentives, envy-freeness, Pareto efficiency, and strategy proofness in large systems.

Seyed Majid Zahedi
Benjamin C. Lee
Duke University

Datacenter servers are often poorly utilized, running at less than 30 percent of peak capability,[1] such that power is amortized over little computation. To improve energy efficiency and address energy disproportionality, software tasks should share hardware. Policies for fair resource allocation (or a lack thereof) determine whether users have incentives to participate in dynamically managed systems. In such shared systems, computer architects encounter two challenges: allocating resources fairly and allocating multiple resources in a coordinated fashion. To address these challenges, this article presents a game-theoretic framework that encourages strategic users to share hardware.

Conventional wisdom in computer architecture assumes that users must share system resources, which overlooks strategic behavior. Users who dislike the allocation policy can choose to participate in another system that provides exclusive access to private, possibly less capable, machines. Moreover, prior policies for fairness equally distribute the performance penalties from sharing,[2,3] but equal slowdowns fail to incentivize sharing between envious users. Finally, prior mechanisms equalize slowdown with heuristics, which lack provable guarantees.[4]

In this work, we rethink fairness in computer architecture, seeking a more robust approach that uses the economic definition of fairness. This article provides a fair, multi-resource allocation policy called Resource Elasticity Fairness (REF), which guarantees four fundamental, game-theoretic desiderata:

- *Sharing incentives* (SI) ensures that users perform no worse than under an equal division of resources—that is, $1/n$ division for n users. Without SI, strategic users would not participate in the shared system. Instead, the n users would prefer exclusive access to their own machines, each with $1/n$-th the resources, an inefficient outcome.
- *Envy-freeness* (EF) ensures that each user prefers her own allocation over other users' allocations. In the economic definition of fairness, an allocation is equitable if it is envy free. This notion of equity is intuitive; humans often proclaim an outcome "unfair" after observing another preferable outcome. Shared resources are allocated equitably when no user experiences envy.

Published by the IEEE Computer Society

- *Pareto efficiency* (PE) allocates resources such that the system cannot improve any user's performance without harming another's. If an allocation is not Pareto efficient, some reallocation could improve system performance. With PE, REF identifies allocations that perform well subject to SI and EF constraints.
- *Strategy-proofness in the large* (SPL) ensures that users cannot improve their performance by misreporting their hardware preferences when the number of users in a shared system is large. In practice, we find that tens of users who share a server with multiple sockets, cores, and threads comprise a sufficiently large system for SPL.

Thus, we present a new framework for reasoning about fair resource allocation in computer architecture. Collectively, our findings establish robust foundations for the fair division of multiple hardware resources.

Fairness and Cobb-Douglas utility

We seek fair resource allocations that provide sharing incentives such that users prefer sophisticated allocation policies to a simple but inefficient policy of equal division. Users that share want fair division, which is defined in economic theory by EF and PE.[5] The REF allocation policy provides these assurances when software preferences for hardware can be modeled by Cobb-Douglas utility functions.

Equation 1 defines utility within the Cobb-Douglas preference domain. Multiple users share a system with R types of hardware. Let $x = (x_1, \ldots, x_R)$ denote a user's hardware allocation, and let $u(x)$ denote her utility from that allocation. The parameters $\alpha = (\alpha_1, \ldots, \alpha_R)$ quantify the elasticity with which a user demands a resource. If $\alpha_r > \alpha_{r'}$, then the user benefits more from resource r than from resource r'. These parameters are tailored and define each user's resource demands.

$$u(x) = \alpha_0 \prod_{r=1}^{R} x_r^{\alpha_r} \quad (1)$$

The exponents α capture nonlinearity and help model diminishing marginal returns in utility, which are prevalent in computer architecture. Examples include Amdahl's law for multicore parallelism, limited locality as cache allocations increase, and limited communication intensity as memory bandwidth allocations increase. The product captures substitution effects and interdependencies between resource allocations. For example, a user might substitute memory bandwidth for cache capacity or vice versa.

Cobb-Douglas utility accurately models performance, and we illustrate its effectiveness in a recurring example. Suppose two users share a system with 24 Gbytes per second (GBps) of memory bandwidth and a 12-Mbyte cache. Let (x_1, y_1) denote memory bandwidth and cache capacity allocated to the first user, and let (x_2, y_2) denote allocations to the second. Let u_1 and u_2 measure instruction throughput. Cobb-Douglas accurately models performance as well as relative elasticities for cache and memory bandwidth. In our example, we fit the following utility functions for `canneal` and `freqmine`, benchmarks from the Parsec suite.

$$u_1 = x_1^{0.6} y_1^{0.4}$$
$$u_2 = x_2^{0.2} y_2^{0.8}$$

Figure 1 illustrates the space of feasible allocations when two users divide cache capacity and memory bandwidth. Allocations that satisfy certain conditions ensure game-theoretic desiderata. We enumerate these conditions for SI, EF, and PE. We can easily determine whether an allocation is fair, given users' utility functions. Although the conditions for fairness are general, their specific effects on Cobb-Douglas supply intuition. We illustrate their effect for two users, and these conditions extend to many users.

Sharing incentives

An allocation policy should provide SI such that users are at least as happy as they would be under an equal division of the resources. Without SI, users would prefer to partition hardware equally. However, doing so would not reflect software diversity and heterogeneous hardware preferences. Hardware would be misallocated, and throughput would suffer.

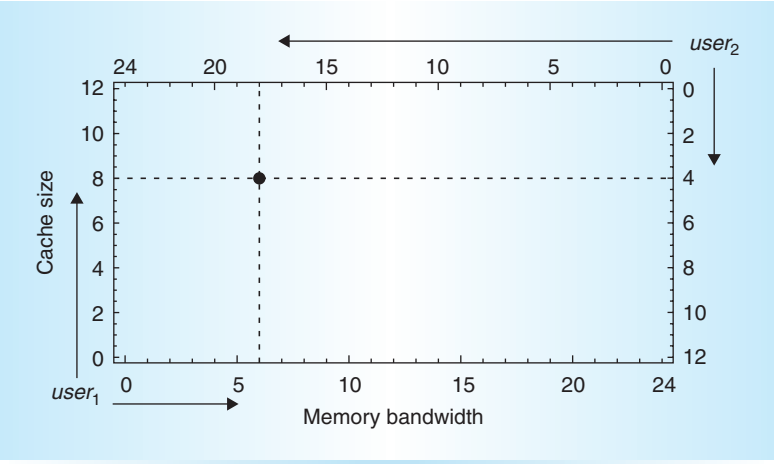

Figure 1. In an Edgeworth box, height and width show total capacity for cache and bandwidth. The origins of users 1 and 2 are at the lower left and upper right corners, respectively. Each point is a feasible allocation. If user 1 gets 6 Gbytes per second (GBps) and 8 Mbytes of bandwidth and cache, user 2 is left with 18 GBps and 4 Mbytes.

Consider our example with cache size and memory bandwidth. In the inequalities in Equations 2 and 3, user 1 compares her allocation (x_1, y_1) against equally partitioning 24 GBps of bandwidth and 12 Mbytes of cache. The allocation policy incentivizes user 1 to share if she weakly prefers (x_1, y_1) to equal division. Similarly, user 2 should weakly prefer (x_2, y_2). A policy that provides SI allocates hardware to satisfy such constraints for each user.

$$x_1^{0.6} y_1^{0.4} \geq \left(\frac{24\,\text{GBps}}{2}\right)^{0.6} \left(\frac{12\,\text{Mbytes}}{2}\right)^{0.4} \quad (2)$$

$$x_2^{0.2} y_2^{0.8} \geq \left(\frac{24\,\text{GBps}}{2}\right)^{0.2} \left(\frac{12\,\text{Mbytes}}{2}\right)^{0.8} \quad (3)$$

Envy-freeness

Allocations are envy free if no user prefers another user's allocation. In economic theory, such allocations are equitable, and equity is a prerequisite for fairness.[5] In our recurring example, EF allocations for user 1 are those for which $u_1(x_1, y_1) \geq u_1(x_2, y_2)$. Note that $(x_2, y_2) = (24 - x_1, 12 - y_1)$. An EF policy allocates hardware to avoid envy for each user, satisfying the following constraints, which produce the EF regions in Figure 2 for our example with two users.

$$x_1^{0.6} y_1^{0.4} \geq (24 - x_1)^{0.6} (12 - y_1)^{0.4} \quad (4)$$

$$x_2^{0.2} y_2^{0.8} \geq (24 - x_2)^{0.2} (12 - y_2)^{0.8} \quad (5)$$

Fundamentally, there always exist at least three EF allocations, which are illustrated by points in the middle and at the corners of Figure 1. The middle equally divides resources between users; no user envies the other. The corners at (0 GBps, 12 Mbytes) and (24 GBps, 0 Mbytes) allocate all of one resource to one user and all of the other resource to the other user. With these allocations, neither user derives utility, because both cache and memory bandwidth are required for computation, yet neither user is envious. These obvious EF allocations perform poorly and call for a mechanism to find more efficient divisions of hardware that exist within the intersection of EF regions.

Pareto efficiency

An allocation is Pareto efficient if increasing one user's utility necessarily decreases another's utility. If an allocation is not Pareto efficient, another allocation exists that could improve aggregate utility. To find PE allocations, we begin with indifference curves that identify sets of allocations that provide the same utility. The gradient at each point on the indifference curve quantifies the marginal rate of substitution (MRS), the rate at which a user substitutes one resource for another without utility loss. In our recurring example, Equation 6 gives the marginal rate of substitution for user 1:

$$MRS_{1,xy} = \frac{\partial u_1 / \partial x_1}{\partial u_1 / \partial y_1} = \left(\frac{0.6}{0.4}\right)\left(\frac{y_1}{x_1}\right) \quad (6)$$

In a PE allocation, the two users have the same MRS, and their indifference curves are tangent. If this were not the case, a user could adjust her allocation, travel along her indifference curve, and substitute resources based on her MRS without affecting utility. Simultaneously, these substitutions would increase the other user's utility and improve overall performance. The contract curve is the set of all PE allocations. In our example, PE

allocations lie on the contract curve in Figure 2 and satisfy the following constraints:

$$MRS_{1,xy} = \left(\frac{0.6}{0.4}\right)\left(\frac{y_1}{x_1}\right) = \left(\frac{0.2}{0.8}\right)\left(\frac{y_2}{x_2}\right) = MRS_{2,xy}$$

Both origins in Figure 2 are PE allocations because one user's utility is 0 and the other's is maximized. At the origin, increasing one user's utility necessarily harms the other user's utility. Although such allocations are Pareto efficient, they are neither desirable nor fair, because the user with zero utility is envious. Thus, we need a mechanism that finds both EF and PE allocations.

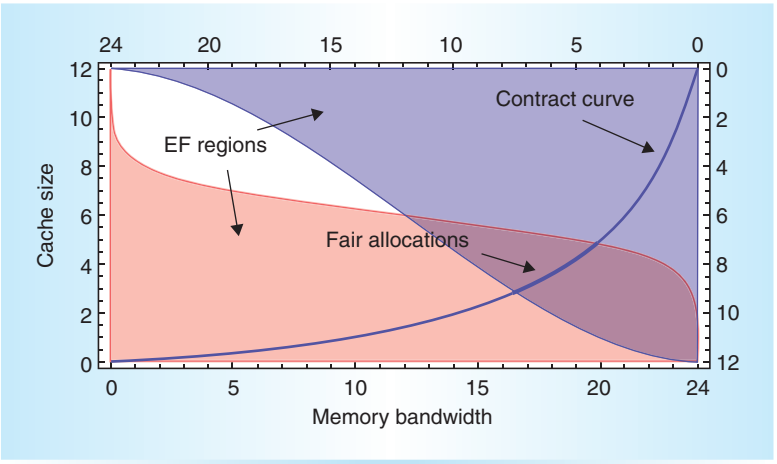

Figure 2. Envy-free regions satisfy Equations 4 and 5. Pareto-efficient allocations lie on the contract curve. Fair allocations are both envy free and Pareto efficient.

Resource Elasticity Fairness

Many feasible allocations provide the game-theoretic desiderata of SI, EF, and PE, despite their constraints and conditions, and we present a simple mechanism to find one. Specifically, we model performance with Cobb-Douglas utility, rescale elasticities, and calculate resource shares in proportion to users' elasticities. We calculate shares with a closed-form expression and enforce those shares with prior mechanisms, such as weighted fair queueing[6] or lottery scheduling.[7]

First, we fit the profile and characterize the performance of user i for varied allocations. We fit a Cobb-Douglas utility function $u_i(x_i) = \alpha_{i0}\prod_{r=1}^{R} x_{ir}^{a_{ir}}$ by taking the log of both sides and performing a linear regression.

Next, we rescale elasticities and utilities as shown in Equation 7. Parameters α in the Cobb-Douglas utility are known as elasticities. For each user i, we rescale elasticities so that they sum to 1. Rescaling lets us compare users' elasticities on the same scale.

$$\hat{\alpha}_{ir} = \frac{\alpha_{ir}}{\sum_{r=1}^{R}\alpha_{ir}}, \hat{u}_i(x_i) = \prod_{r=1}^{R} x_{ir}^{\hat{\alpha}_{ir}} \quad (7)$$

Finally, we use rescaled elasticities to determine the fair share for each user i and resource r. Because REF allocates in proportion to elasticity, users that benefit more from resource r will receive a larger share of the total resource capacity C_r.

$$x_{ir} = \frac{\hat{\alpha}_{ir}}{\sum_{j=1}^{N}\hat{\alpha}_{jr}} \times C_r \quad (8)$$

In our cache and bandwidth example, two users provide Cobb-Douglas utility functions with elasticities that are already scaled and summed to 1 (for example, $u_1 = x_1^{0.6}y_1^{0.4}$). The REF mechanism examines both users' memory bandwidth elasticities ($\alpha_{1x} = 0.6$ and $\alpha_{2x} = 0.2$) and allocates this resource proportionally. Similarly, the REF mechanism allocates last-level cache capacity.

$$x_1 = \left(\frac{0.6}{0.8}\right) \times 24 = 18\,\text{GBps},$$

$$y_1 = \left(\frac{0.4}{1.2}\right) \times 12 = 4\,\text{Mbytes},$$

$$x_2 = \left(\frac{0.2}{0.8}\right) \times 24 = 6\,\text{GBps},$$

$$y_2 = \left(\frac{0.8}{1.2}\right) \times 12 = 8\,\text{Mbytes}.$$

The REF mechanism guarantees SI, EF, and PE. We sketch the proofs and refer the reader to our prior work for details.[8] Equation 8 expresses the mechanism's allocation in closed form, which makes the calculation trivial, once user performance is modeled as a Cobb-Douglas utility. The allocation is

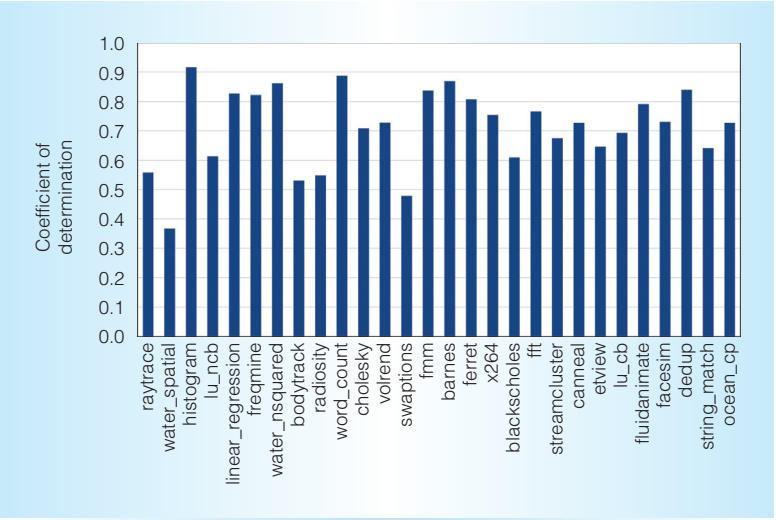

Figure 3. Coefficient of determination measures goodness of fit for Cobb-Douglas utility. Larger values are better.

equivalent to the Nash bargaining solution. The Nash bargaining solution is, in turn, equivalent to the Competitive Equilibrium from Equal Incomes (CEEI) solution for homogeneous utility functions, such as Cobb-Douglas.[9] Because the CEEI solution provides SI, EF, and PE for rescaled Cobb-Douglas utilities,[5] REF provides these properties as well.

REF is also strategy proof in the large. An allocation mechanism is strategy proof (SP) if a user can't gain by misreporting her utility. Unfortunately, for Cobb-Douglas, SP is too restrictive a property, and no mechanism can provide both PE and SP.[10] However, our REF mechanism satisfies a weaker property, SPL. When there are many users in the system, users have no incentive to lie about their elasticities α. In theory, SPL holds when an individual user's elasticity is far smaller than the sum of all users' elasticities. In practice, tens of agents suffice to provide SPL. For example, SPL holds when 64 tasks share a large system, which is a realistic setting because modern servers have four processor sockets (= 64 threads) that share eight to 12 memory channels (>100 GBps of bandwidth).

Evaluation

We evaluate REF from three perspectives. First, we show that Cobb-Douglas utilities accurately model chip multiprocessor performance. We apply a log transformation to linearize the Cobb-Douglas function and to accurately fit to the empirical data with least squares. Second, we show that REF provides game-theoretic desiderata, whereas equalizing slowdowns does not. Beyond the proofs in our prior work,[8] we illustrate SI and EF with examples. Finally, we show that REF guarantees game-theoretic definitions of fairness with little impact on performance.

Fitting Cobb-Douglas utility

Performance for each user's application is measured in terms of instructions per cycle, which we profile with cycle-accurate simulation. Given profiles for varied cache size and memory bandwidth allocations, we construct a model, $u = a_0 x^{a_x} y^{a_y}$, where u is application performance, x is memory bandwidth, and y is cache size. Although a nonlinear relationship exists between Cobb-Douglas utility and resource allocations, a logarithmic transformation produces a linear model. Least-squares regression provides an estimate for each elasticity parameter α_*.

We evaluate the regression fit by reporting the coefficient of determination (R-squared), which measures how much variance in the empirical data is captured by the model. R-squared approaches 1.0 as fit improves. Figure 3 shows that Cobb-Douglas accurately models performance with R-squared of 0.7 to 1.0 for most applications. Applications such as `radiosity` whose models exhibit low R-squared exhibit negligible performance variance as allocations change and have no trend for Cobb-Douglas to capture.

Interpreting Cobb-Douglas utility

Elasticities quantify the extent to which a user demands a resource. In a multiresource setting, elasticities quantify the relative importance of each resource to user performance. We rescale Cobb-Douglas elasticities as described in Equation 7 and plot them in Figure 4. An application derives more utility from cache than it does from memory bandwidth when $\alpha_{cache} > \alpha_{mem}$ (for example, `raytrace`). In contrast, an application finds memory bandwidth more useful when $\alpha_{mem} > \alpha_{cache}$ (for example, `dedup`). Thus, we can classify applications into two groups on the basis of their resource elasticities. Applications in group M demand memory

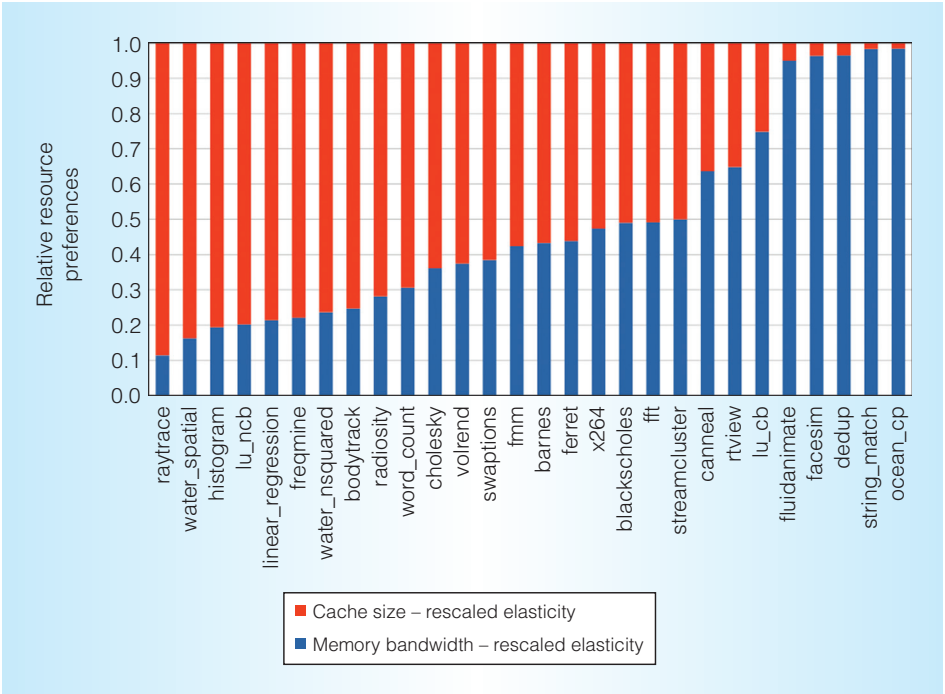

Figure 4. Resource elasticities from Equation 7 show the relative importance of cache size and memory bandwidth. Workloads for which $\alpha_{mem} > 0.5$ are classified M. Otherwise, they are classified C.

bandwidth intensely ($\alpha_{mem} > 0.5$), and those in group C demand cache intensely ($\alpha_{cache} > 0.5$).

REF versus equal slowdown

The REF mechanism uses Cobb-Douglas elasticities to allocate multiple resources in a coordinated fashion, ensuring game-theoretic desiderata. We compare REF against a mechanism that allocates to equally distribute the performance penalties (that is, slowdowns) from sharing, a common objective in computer architecture.[11,12] We find that equal slowdowns fail to incentivize sharing and mitigate envy between strategic users.

Figure 5 presents an example with two applications in different groups—`barnes` (C) and `canneal` (M). `Barnes` prefers cache to bandwidth, whereas cannel prefers bandwidth to cache. This example shows how a mechanism that equalizes slow-downs fails to satisfy SI and EF for `canneal`, which receives less than half of both resources in the system. In this setting, `canneal` would be unwilling to participate in a shared system that equalizes slow-downs. `Canneal` would rather receive a static allocation with half the hardware. Moreover, `canneal` envies `barnes`' allocation. In contrast, REF allocates more than half of the memory bandwidth to `canneal`, giving it an incentive to share. REF also ensures EF by exploiting substitution effects, which are modeled by Cobb-Douglas, as `canneal` is offered more cache in exchange for less bandwidth.

Figure 6 presents a second example with applications from the same group—`freqmine` (C) and `linear regression` (C), which both prefer cache capacity to memory bandwidth. However, because `linear` exhibits far more memory activity than `freqmine`, a mechanism that equalizes slowdowns must allocate far more of both resources to `linear`. In this setting, `freqmine` prefers a static, equal division of resources over an allocation policy that equalizes slowdown. `Freqmine` will not share the system with dynamic resource allocation; the system does not provide SI. Even if `freqmine` were willing to share resources with `linear`, it would prefer `linear`'s allocation over its own; the system does not provide EF. In contrast, REF divides

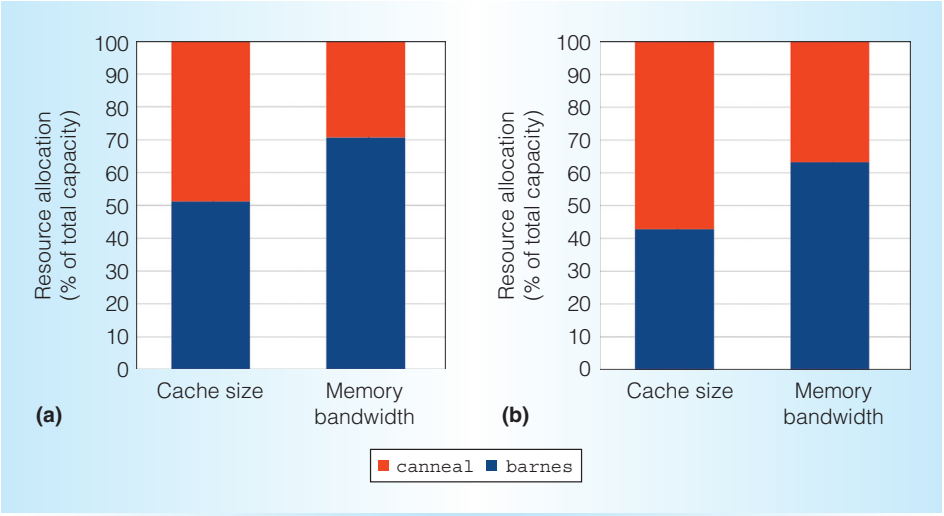

Figure 5. Allocations for `barnes` and `canneal` show that equalizing slowdowns can fail to satisfy sharing incentives (SI) and envy-freeness (EF), providing `canneal` less than half of both resources. Resource Elasticity Fairness (REF) satisfies SI and EF.

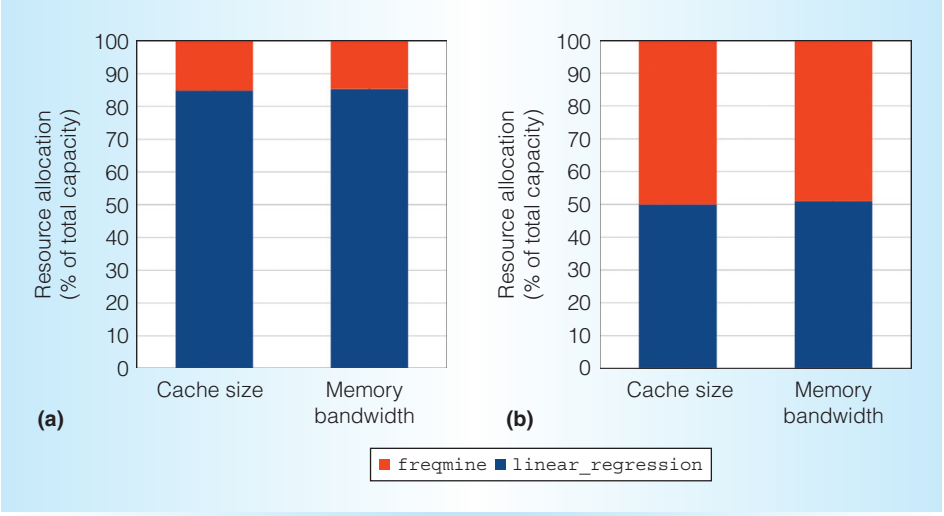

Figure 6. Allocations for `freqmine` and `linear regression` show that equalizing slowdowns can fail to satisfy SI and EF, providing `freqmine` less than half of both resources. REF satisfies SI and EF.

resources almost equally between benchmarks to ensure SI and EF, even when `linear` demands those resources more intensely. REF ensures game-theoretic fairness at the expense of performance in this setting.

Fairness versus efficiency

Tradeoffs between fairness and efficiency are inevitable, yet we find that REF guarantees game-theoretic desiderata with only modest performance penalties. We evaluate performance in terms of weighted throughput,[3] which is defined as an application's utility when sharing divided by the utility when not. Specifically, $U(x) = u(x)/u(C)$, in which x is an allocation from the shared system, and C is the total system capacity. To evaluate REF, we compare and contrast weighted throughput for several allocation policies that make very different tradeoffs:

1. *Max welfare without fairness.* This policy allocates resources to maximize welfare, subject to capacity constraints. We maximize Nash social welfare, $\Pi_i U_i(x_i)$, with geometric programming to obtain an empirical upper bound on performance.
2. *Equal slowdown without fairness.* This policy allocates resources to maximize minimum welfare across all users. We optimize a max-min objective function on $U_i(x_i)$ for users *i,* which minimizes the gap between the best and worst performing users and is equivalent to equalizing slowdown.
3. *Max welfare with fairness.* This policy allocates resources to maximize welfare, subject to SI, EF, and PE constraints. We maximize Nash social welfare, $\Pi_i U_i(x_i)$, with geometric programming to obtain an empirical upper bound on performance with game-theoretic fairness.
4. *Resource elasticity fairness.* This policy allocates resources according to REF. We fit Cobb-Douglas models, rescale elasticities, and allocate resources in proportion to elasticities.

Policies that maximize welfare (1 and 3) provide upper bounds on performance, but their implementation is impractical, requiring geometric programming and convex optimization. Although less computationally intensive, a policy that equalizes slowdowns (2) also requires iterative optimization. In contrast, resource elasticity fairness (4) uses closed-form expressions to calculate each user's share of the resources in a computationally trivial mechanism. This comparison emphasizes online analysis during allocation and neglects offline analysis for fitting Cobb-Douglas models, which does not affect the critical path.

Policies that neglect fairness can perform well. Maximizing welfare without fairness (1), the least restrictive policy, allocates resources to maximize aggregate throughput and provides an empirical upper bound on performance. Relative to this bound, equalizing slowdowns (2) penalizes performance by optimizing worst-case performance. These penalties grow as the number of users increases, because equalizing slowdowns allocates resources to favor the least satisfied user. A larger system is more likely to include a dissatisfied user who is very different from the others and demands a large share of the resources. Recall, however, that equalizing slowdowns fails to ensure SI and EF.

The two policies that ensure game-theoretic fairness (3, 4) perform comparably, which is a compelling finding because REF allocations perform as well as those resulting from explicitly optimizing performance, subject to SI, EF, and PE constraints. Although fairness penalizes performance, the penalties are modest. REF performance is often within 90 percent of policies that pursue performance alone (1) or equalize slowdowns (2). Indeed, REF can outperform policies that equalize slowdown as the number of users increases; our simulations indicate superior REF performance for systems with as few as eight users.

We introduce computer architects to a rich body of knowledge of economics and game theory, permitting qualitatively new performance guarantees in hardware platforms shared by strategic users. Without loss of generality, we evaluate our multiresource allocation mechanism for cache size and memory bandwidth. We expect Cobb-Douglas utility functions to generalize beyond cache size and memory bandwidth. In the future, computer architects might extend REF to support additional resources, such as the number of processor cores.

REF provides a policy and mechanism for single (that is, one) resource allocation for a system in which user participation and behavior is static. However, in practice, systems are dynamic—users arrive and leave, and software behavior exhibits phases. Extending notions of sharing incentives and envy-freeness to accommodate temporal variations is daunting. We will need new policies that define game-theoretic desiderata across multiple time periods, new mechanisms that adapt to the past and anticipate the future, and new solution concepts to model strategic behaviors and system dynamics. Collectively, we argue for new thinking in the fair allocation of computational resources. Rather than assume users must share, we argue that polices must encourage sharing. As distributed computing proliferates, whether in the

cloud or between mobile devices, game theory provides an essential framework for shaping user behavior in shared systems. MICRO

Acknowledgments

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 authors and do not necessarily reflect the views of these sponsors.

References

1. L.A. Barroso and U. Hölzle, *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*, Morgan & Claypool, 2009.
2. R. Gabor, S. Weiss, and A. Mendelson, "Fairness Enforcement in Switch on Event Multithreading," *ACM Trans. Architecture and Code Optimization*, vol. 4, no. 3, 2007, article 15.
3. S. Eyerman and L. Eeckhout, "System-Level Performance Metrics for Multiprogram Workloads," *IEEE Micro*, vol. 28, no. 3, 2008, pp. 42–53.
4. O. Mutlu and T. Moscibroda, "Stall-Time Fair Memory Access Scheduling for Chip Multiprocessors," *Proc. Int'l Symp. Microarchitecture*, 2007, pp. 146–160.
5. H. Varian, "Equity, Envy, and Efficiency," *J. Economic Theory*, vol. 9, no. 1, 1974, pp. 63–91.
6. A. Demers, S. Keshav, and S. Shenker, "Analysis and Simulation of a Fair Queueing Algorithm," *ACM SIGCOMM Computer Communication Rev.*, vol. 19, no. 4, 1989, pp. 1–12.
7. C.A. Waldspurger and W.E. Weihl, "Lottery Scheduling: Flexible Proportional-Share Resource Management," *Proc. USENIX Conf. Operating Systems Design and Implementation* (OSDI 94), 1994, p. 1.
8. S.M. Zahedi and B.C. Lee, "REF: Resource Elasticity Fairness with Sharing Incentives for Multiprocessors," *Proc. Int'l Conf. Architectural Support for Programming Languages and Operating Systems*, 2014, pp. 145–160.
9. H. Moulin, *Fair Division and Collective Welfare*, MIT Press, 2004.
10. K. Hashimoto, "Strategy-Proofness Versus Efficiency on the Cobb-Douglas Domain of Exchange Economies," *Social Choice and Welfare*, vol. 31, no. 3, 2008, pp. 457–473.
11. R. Bitirgen, E. Ipek, and J.F. Martinez, "Coordinated Management of Multiple Interacting Resources in Chip Multiprocessors: A Machine Learning Approach," *Proc. Int'l Symp. Microarchitecture*, 2008, pp. 318–329.
12. O. Mutlu and T. Moscibroda, "Parallelism-Aware Batch Scheduling: Enhancing Both Performance and Fairness of Shared DRAM Systems," *Proc. Int'l Symp. Computer Architecture* (ISCA 08), 2008, pp. 63–74.

Seyed Majid Zahedi is a PhD student in the Department of Computer Science at Duke University. His research focuses on microeconomics and game theory for hardware resource allocation and management. Zahedi has an MS in electrical and computer engineering from the University of Tehran. He is a member of IEEE. Contact him at zahedi@cs.duke.edu.

Benjamin C. Lee is an assistant professor in the Department of Electrical and Computer Engineering at Duke University. His research interests include energy-efficient design, emerging technologies, statistical methods, and algorithmic economics for computer architecture. Lee has a PhD in computer science from Harvard University. He is a member of IEEE and the ACM. Contact him at benjamin.c.lee@duke.edu.