ReBudget: Trading Off Efficiency vs. Fairness in Market-Based Multicore Resource Allocation via Runtime Budget Reassignment

Xiaodong Wang José F. Martínez

Computer Systems Laboratory
Cornell University
Ithaca, NY 14853 USA
http://m3.csl.cornell.edu

Abstract

Efficiently allocating shared resources in computer systems is critical to optimizing execution. Recently, a number of market-based solutions have been proposed to attack this problem. Some of them provide provable theoretical bounds to efficiency and/or fairness losses under market equilibrium. However, they are limited to markets with potentially important constraints, such as enforcing equal budget for all players, or curve-fitting players' utility into a specific function type. Moreover, they do not generally provide an intuitive "knob" to control efficiency vs. fairness.

In this paper, we introduce two new metrics, Market Utility Range (MUR) and Market Budget Range (MBR), through which we provide for the first time theoretical bounds on efficiency and fairness of market equilibria under *arbitrary* budget assignments. We leverage this result and propose *ReBudget*, an iterative budget re-assignment algorithm that can be used to control efficiency vs. fairness at run-time. We apply our algorithm to a multi-resource allocation problem in multicore chips. Our evaluation using detailed execution-driven simulations shows that our budget re-assignment technique is intuitive, effective, and efficient.

Keywords Multicore architectures; Market equilibria; Scalable resource allocation; Efficiency; Fairness.

1. Introduction

Devising scalable chip-multiprocessor (CMP) designs is an important goal for the upcoming manycore generation. A key challenge to scalability is the fact that these cores will share hardware resources, be it on-chip cache, pin bandwidth, the chip's power budget, etc. Prior work has shown that freely contending for shared resources can penalize system performance [4, 12, 15]. Thus, allocating resources efficiently among cores is key.

Unfortunately, single-resource, and more generally uncoordinated resource allocation, can be significantly suboptimal, due to its inability to model the interactions among resources [4]. A few solutions have been proposed to coordinate resource allocation across multiple resources, and their performance estimation methods range from trial runs [1, 15, 22], to artificial neural networks [4], and to analytical models [12–14]. Unfortunately, these all rely on centralized mechanisms (e.g., global hill-climbing) to optimize system throughput, essentially exploring the global search space sequentially, which may be prohibitively expensive, particularly in large-scale systems.

More recently, a number of market-based approaches have been introduced. Chase et al. propose a static market [11], where the players reveal to the resource supplier the amount of money each is willing to pay as a function of allocated service units, and the central market then allocates the available computing resources so that monetary profit is maximized. Still, because the maximization process is done by the supplier centrally, it is unclear whether it could deal with a large-scale system efficiently.

In the context of distributed computing clusters, Lai et al. propose a market-based solution to resource allocation that allows players to adjust their bids dynamically in response to the others bids to that resource [23]. Resource allocation is done in a largely distributed manner, which enables the system to scale better than centralized approaches. Recently, in our XChange work [37], we also propose one such dynamic market in the context of CMPs, and similarly show that XChange is scalable due to its largely distributed nature: Instead of making the resource allocation decision globally, each core in the CMP is actively optimizing its resource assignment largely independently of each other, and partic-

ASPLOS '16, April 2–6, 2016, Atlanta, Georgia, USA..
Copyright © 2016 ACM 978-1-4503-4091-5/16/04…$15.00.
http://dx.doi.org/10.1145/2872362.2872382

ipants demands are reconciled through a relatively simple pricing strategy.

XChange also shows that it can achieve a good balance between system efficiency and fairness. The study is purely empirical, however, and thus it does not provide any guarantees on the loss of efficiency and fairness. It is well-known, for example, that market mechanisms in equilibrium can sometimes be highly inefficient—this is known as *Tragedy of Commons* [18]. Therefore, a number of research efforts have focused on quantifying the efficiency loss compared to the optimal resource allocation, known as the *Price of Anarchy (PoA)*. For example, Zhang studies a market where all players have the same amount of money (budget) to purchase resources, and he finds that the overall system efficiency in such a market can be low ($1/\sqrt{N}$ of the maximum feasible utility, where N is the number of market players), but fairness is high (0.828-approximate envy-free, a measure of fairness) [40]. This is consistent with our empirical observations in the XChange work [37].

Recently, Zahedi and Lee propose a resource allocation mechanism named *elasticities proportional* (EP) for CMPs [39], which does provide game-theoretic guarantees such as Pareto efficiency, envy-freeness, etc. However, such guarantees rely on the assumption that an application's utility can be accurately curve-fitted to a Cobb-Douglas function, where the coefficients are used as the "elasticities" of resources. Our XChange work shows that EP can in fact perform worse than expected when such curve-fitting is not well suited to the applications. In addition, although EP is proven to be Pareto-efficient, its efficiency loss compared to global optimality is not quantified.

To improve system efficiency while sacrificing some fairness, our XChange work discusses a *wealth redistribution* technique, which varies the players' budget based on an estimation of their potential for performance gain. However, XChange's wealth redistribution is an on/off technique, providing no "knob" to control the efficiency vs. fairness trade-off. It is also not backed up by a theoretical result that would provide bounds for this trade-off. To the best of our knowledge, there is no theoretic study that is able to quantify the loss of *both* efficiency and fairness under an arbitrary budget assignment.

Contributions

The contributions of this paper are as follows:

— We introduce a new *Market Utility Range (MUR)* metric, which helps us establish a theoretical bound for efficiency loss of a market equilibrium under a constrained budget. Specifically, we show that, if $\text{MUR} \geq 0.5$, then $\text{PoA} \geq (1 - \frac{1}{4\text{MUR}}) \geq 0.5$ (i.e., the efficiency is guaranteed to be at least 50% of the optimal allocation); and that if $\text{MUR} < 0.5$, then $\text{PoA} \geq \text{MUR}$.

— We introduce a new *Market Budget Range (MBR)* metric, which helps us evaluate the fairness of a market equilibrium under a constrained budget. We show that any market equilibrium is $(2\sqrt{1+\text{MBR}} - 2)$-approximate envy-free.

— We propose *ReBudget*, a budget re-assignment technique that is able to systematically control efficiency and fairness in an adjustable manner. We evaluate ReBudget on top of XChange, using a detailed simulation of a multicore architecture running a variety of applications. Our results show that ReBudget is efficient and effective. In particular, it can achieve 95% of the maximum feasible efficiency. Furthermore, when combined with the analysis using MUR and MBR metrics, it can provide worst-case fairness guarantees.

2. Market Framework

This paper adopts the general market-based resource allocation framework proposed in our XChange work [37]. In this section, we describe our efficiency and fairness models in the context of that framework.

Assume a market consisting of N players and M resources. Each player i has a utility function $U_i(\mathbf{r_i})$ when it is allocated $\mathbf{r_i} = (r_{i1}, r_{i2}, \ldots, r_{iM})$ resources. We assume that a player's utility functions is concave, nondecreasing, and continuous. Note that this may not always hold in the CMP context (e.g., the utility of the allocated cache space [2, 37]). We describe how we address this issue later in Section 4.

Every player i is allowed to bid b_{ij} for resource j, and the sum of its bids cannot exceed its budget B_i (i.e., the total amount of money it is allowed to spend): $\sum_j b_{ij} \leq B_i$. The market reconciles those bids by adopting a *proportional* allocation scheme, which is widely used [23, 40] and considered fair. The market first collects bids from all the players, and then determines the price of each resource j as follows:

$$p_j = \frac{\sum_{i=1}^{N} b_{ij}}{C_j} \tag{1}$$

where C_j represents the total amount of resource j. As a result, player i gets r_{ij} units of resource j proportionally to its bid: $r_{ij} = \frac{b_{ij}}{p_j}$.

The essence of this type of market-based approach is that it is a largely distributed mechanism: the players independently traverse their local search space to find the bids that maximize their own utilities, bringing the market toward a resource allocation that is Pareto-optimal [25]. In order to find such optimal bids, each player needs to solve an optimization problem, which can be modeled as follows: Given the price p_j of resource j announced by the market, player i is able to compute the sum of other players' bids to that resource: $y_{ij} = \sum_{i' \neq i} b_{i'j} = p_j \times C_j - b_{ij}$. By making the simplification that other players do not change their bids and therefore y_{ij} are constants, player i is able to make a prediction on the amount of resource r_{ij} it can get if it changes its bids from b_{ij} to b'_{ij}:

$$r_{ij} = \frac{b'_{ij}}{b'_{ij} + y_{ij}} C_j \tag{2}$$

Combining this equation with player i's utility function $U_i(\mathbf{r_i})$, the player can obtain its utility function vs. its bids: $U_i(\mathbf{b_i}) = U_i(\frac{b_{ij}}{b_{ij}+y_{ij}}C_j)$. Then the optimization problem a player needs to solve is:

$$\begin{aligned} \text{maximize} \quad & U_i(\mathbf{b}_i) \\ \text{subject to} \quad & \sum_j b_{ij} \leq B_i \end{aligned} \tag{3}$$

Using the Lagrangian multiplier method, we conclude that if such optimal bids exist, there exists a player-specific constant $\lambda_i > 0$ such that, for any resource j:

$$\frac{\partial U_i}{\partial b_{ij}} \begin{cases} = \lambda_i & \text{if} \quad b_{ij} > 0 \\ < \lambda_i & \text{if} \quad b_{ij} = 0 \end{cases} \tag{4}$$

Intuitively, we define λ_{ij} to be the rate of utility change (marginal utility) if player i changes its bid on resource j by one unit: $\lambda_{ij} = \frac{\partial U_i}{\partial b_{ij}}$. From Equation 4, if player i submits non-zero bids on different resources j, the λ_{ij} for all those resources are the same, and equal to λ_i in Equation 4. For resources with zero bids, their λ_{ij} is necessarily smaller than λ_i. Otherwise, it contradicts the condition that the bids maximize the player's utility.

Let's use an illustrative example. Assume player i bids on two resources, with $\lambda_{i1} = 1$ and $\lambda_{i2} = 2$. If the player moves one unit of bid from resource 1 to 2, its utility can be increased by 1 unit (-1 from resource 1 and +2 from resource 2). As a result, the current bids are not optimal, and the player can keep improving its utility by adjusting its bids until λ_{ij} are equal, or the bids on one of the resources drop to zero, beyond which no further profitable movement is possible.

2.1 Market Equilibrium

Market equilibrium is a state where all the players have no incentive to change their bids to improve utilities, and the resource prices remain stable. It is a desirable state because it is proven to be Pareto-optimal (i.e., no other resource allocation can make any one individual better off without making at least one individual worse off [25]). In order to find a market equilibrium, we adopt an iterative bidding–pricing process, similar to the one used in our recent XChange work [37]. Such a process can be divided into two steps: (1) the market broadcasts the current resource prices to all the players, and (2) the players adjust their bids to maximize their own utilities. These two steps are repeated iteratively until the market converges—i.e., for any player, its utility changes negligibly between two iterations. (In our implementation we detect this globally by monitoring prices instead, and assume convergence when they fluctuate within 1%.)

Zhang [40] has shown that a market equilibrium always exists in a *strongly competitive market*, where for any resource j, there always exists at least two players placing non-zero bids.

Lemma 1. *An equilibrium always exists for a strongly competitive market. The market equilibrium may not be unique.*

2.2 Efficiency

Given the existence of a market equilibrium, its system efficiency, also known as social welfare, is an important metric.

Definition 1. *The efficiency of a system is defined as the sum of players' utilities: Efficiency* $= \sum_i U_i(\mathbf{r}_i)$.

Nissan et al. [25] show that the efficiency of a market equilibrium can be low. Papadimitriou introduces the concept of *Price of Anarchy* (PoA) [26], which is the lower bound of the efficiency of a market equilibrium compared with that of the optimal allocation. Mathematically, let $\mathbf{r}_i^*$ denote a feasible resource allocation which maximizes the system efficiency, Ω be the set of resource allocation in market equilibrium (recall that a market equilibrium may not be unique), and $\mathbf{r}^n \in \Omega$ be a market equilibrium outcome. Let us also define optimal efficiency OPT $= \sum_i U_i(\mathbf{r}_i^*)$, and efficiency in market equilibrium Nash$(\mathbf{r^n}) = \sum_i U_i(\mathbf{r}_i^n)$, the Price of Anarchy is then defined as:

Definition 2. *PoA* $= \min_{\mathbf{r^n} \in \Omega} \frac{\mathit{Nash}(\mathbf{r^n})}{\mathit{OPT}}$

Note that PoA is a lower bound, which means that the efficiency of any market equilibrium $\mathbf{r^n}$ is guaranteed to be greater than PoA $\times$ OPT.

Zhang [40] studies PoA in a market with a proportionally balanced budget, where a player is given a budget in proportion to its maximum utility, i.e., the utility when it owns all resources. Zhang then shows:

Lemma 2. *The equilibrium in a market with a proportionally balanced budget has a Price of Anarchy PoA* $= \Theta(\frac{1}{\sqrt{N}})$.

Recall that N is the number of players; thus, Lemma 2 tells us that PoA will worsen with the number of players, and thus it can be prohibitively low in a large market.

2.3 Fairness

Envy-freeness (EF) is widely used to evaluate fairness of a resource allocation in real life [5, 35], and it's recently introduced by Zahedi and Lee in the context of resource allocation in CMPs [39]. It is a metric to evaluate how much a player desires others' resources compared to what it owns.

Definition 3. *Envy-freeness (EF) of an allocation* $\mathbf{r} = (\mathbf{r}_1, \ldots \mathbf{r}_N)$ *is EF*$(\mathbf{r}) = \min_{i,j} \frac{U_i(\mathbf{r}_i)}{U_i(\mathbf{r}_j)}$.

By definition, a resource allocation is envy-free when $EF \geq 1$—i.e., players prefer their own resources to those of others (at worst, they like them equally). Although a market equilibrium is provably envy-free under some form of utility constraints [39], in general it is not. And although it might seem that a market equilibrium would be generally fair as long as every player is endowed with the same budget, Zhang shows that this is not guaranteed [40]. As a result, Zhang defines c-approximate envy-free ($c \leq 1$) as follows:

Definition 4. *A market is c-approximate envy-free, if the envy-freeness of any market equilibrium is larger than c, i.e., for any player i,* $U_i(\mathbf{r}_i) \geq c \cdot \max_j U_i(\mathbf{r}_j)$.

It is straightforward that the equilibrium is more *"fair"* if c is closer to 1 (players envy others less). Zhang then proves [40]:

Lemma 3. *If each player in the market has the same budget, market equilibrium is at least 0.828-approximate envy-free. The bound is tight in the worst case.*

Note that the bounds of efficiency and fairness proven by Zhang [40] may not apply at the same time. Recall that Lemma 2 requires a proportionally balanced budget assignment (i.e., a player's budget is proportional to its maximum achievable utility), while Lemma 3 assumes every player has an equal budget. However, note that in the multicore resource allocation problem that we study, the utility function is in fact a value normalized to the maximum utility (discussed in Section 4.1.1). As a result, the maximum utility is 1 for all the players, and therefore these two markets are equivalent within the scope of this paper.

Thus, by combining Lemma 2 and Lemma 3 in our context, we find although a market with equal budget for all players has a good fairness guarantee, its efficiency can be low ($1/\sqrt{N}$ of optimal allocation in the worst case). In the following sections, we study how budget assignment across players affects the theoretical bound of efficiency and fairness, and how to utilize such theory to design a budget re-assignment scheme to trade off efficiency and fairness systematically.

3. Theoretical Results

In this section, we introduce two new metrics that will serve our goal: *Market Utility Range (MUR)* and *Market Budget Range (MBR)*. By measuring MUR and MBR in the market equilibrium, we can quantitatively understand the bound of loss in efficiency and fairness. In addition, MUR and MBR can be used as a guidance to adjust the budgets across players so that we can control the trade-off between efficiency and fairness more effectively.

3.1 Efficiency

According to Equation 4, in a market with a budget constraint, if player i bids optimally to maximize its utility, its marginal utility of bids $\frac{\partial U_i}{\partial b_{ij}}$ is a player-specific value λ_i, identical for all resources j with non-zero bids. Our intuition is that, the larger the λ_i *variation* across players, the higher "potential" there is to increase system efficiency by budget re-assignment, and therefore the lower PoA efficiency guarantee the current market has.

Consider the following examples: Assume a budget-constrained market with two players (A and B), such that $\lambda_A = 1$ and $\lambda_B = 3$ in equilibrium. It is intuitive that if the market moves 1 unit of budget from A to B, the market's overall efficiency in equilibrium (which is the sum of A's and B's utilities) is likely to increase (e.g., by 2 units, -1 from A and +3 from B). Consider, instead, that $\lambda_A = 1$ and $\lambda_B = 2$. In that case, the same budget re-assignment also points toward a market efficiency increase, but the improvement may be lower than in the first case. Finally, consider that λ_A and λ_B are equal. In that case, the intuitive expectation is that a budget re-assignment will not have an effect in overall market efficiency.

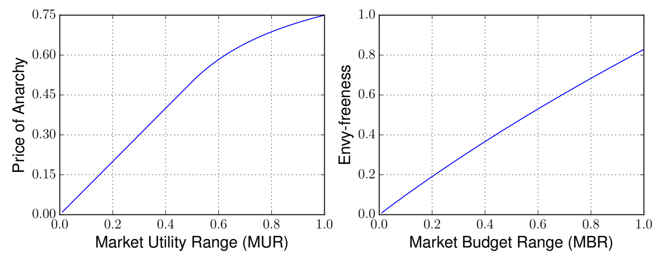


Figure 1: Relationship between Price of Anarchy and Market Utility Range (left), and Envy-freeness and Market Budget Range (right), based on Theorem 1 and Theorem 2, respectively.

Consequently, let's define *Market Utility Range (MUR)* as follows:

Definition 5. *Maximum Utility Range is the maximum variation of marginal utility* λ_i *across the market players,* $MUR = \frac{\min_i \lambda_i}{\max_i \lambda_i}$

By using such definition, we can prove that:

Theorem 1. *If MUR* ≥ 0.5, *the Price of Anarchy of the market equilibrium PoA* $\geq (1 - \frac{1}{4MUR}) \geq 0.5$, *i.e., the overall market efficiency is guaranteed to be at least 50% of optimal allocation; If MUR* < 0.5, *PoA* $\geq$ *MUR.*

(The detailed proof can be found later in Section A.1.)

Not only does MUR provide a lower bound of overall market efficiency, it can also be used to guide budget re-assignment to help improve the overall market efficiency. As shown in the examples above, by moving a portion of budget from a player i with lower λ_i to another player i' with higher $\lambda_{i'}$, MUR moves toward 1.[1] As a result, the PoA guarantee increases, and the actual market efficiency hopefully increases accordingly.

3.2 Envy-freeness

A likely side effect of assigning different budgets to different players is that it may impact fairness negatively. It is straightforward that the player with the highest budget is able to purchase more resources than others, and therefore it is likely to be "envied" by others. Hence, we hypothesize that a valuable indicator of envy (or envy-freeness) of a market in equilibrium is the variation of the budget across players:

[1] It is provable that player i's marginal utility of bids λ_i decreases monotonically with a larger budget.

Definition 6. *Market Budget Range is the maximum variation in budget across players, MBR* $= \frac{\min_i B_i}{\max_i B_i}$.

Note that MBR is defined as the minimum budget divided by the maximum budget, so that larger budget variation means lower MBR value. Based on this definition, we can prove the following:

Theorem 2. *A market equilibrium with budget range MBR is* $(2\sqrt{1+MBR}-2)$*-approximate envy-free.*

(A short proof sketch can be found in Section A.2.)

The key insight here is that, by combining Theorem 1 and Theorem 2, we can attempt to adjust the trade-off between efficiency and fairness: As Figure 1 shows, the more aggressively we re-assign the budget to make MUR closer to 1, the higher system efficiency we may achieve. However, it creates a larger variation in players' budgets, which in turn may hurt fairness. Note that such budget re-assignments do not *guarantee* an actual improvement in either the efficiency or the envy-freeness. Nevertheless, our expectation is that, by tightening the efficiency/envy-freeness bounds according to our MUR and MBR theorems, the resulting allocation at equilibrium will tend to move in the desired direction. Therefore, we envision that an algorithm, by using MUR and MBR together, can try to fine-tune a market's trade-off between efficiency and fairness. We show one such algorithm ReBudget in Section 4.2.

4. ReBudget Framework

In this section, we first describe the basic framework for market-based resource allocation. We then describe ReBudget, a practical heuristic based on the theoretical results in Section 3 to assign budgets across players, so that an adjustable trade-off between system efficiency and fairness can be accomplished.

4.1 Market-based Approach

Our basic market-based resource allocation framework, described in Section 2, is a dynamic proportional market. In this framework, the goal is to find a market equilibrium using an iterative bidding–pricing procedure, after which resources are allocated proportionally to bids. This mechanism is detailed in Section 2.1.

Shared cache space and on-chip power are two of the most frequently targeted resources in the literature [6, 9, 24, 27, 34, 38], and our evaluation of ReBudget will focus on these two resources. Our mechanism, however, is a general framework: As long as the resource's utility function can be accurately modeled, and such utility function is non-decreasing, continuous, and concave (or can be made concave), the results of this paper can be applied. (Note that prior studies show that the utility of cache is often non-continuous—e.g., if partitioned by cache ways—and non-concave [2, 37], which is not consistent with our theoretical

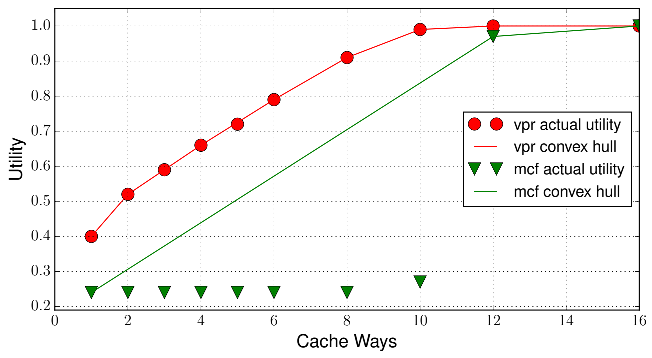


Figure 2: Normalized utility under different cache allocation, running at the highest possible frequency. The x-axis is the number of cache ways enabled. Section 5 describes the setup.

assumptions. We describe how to address such issue later in Section 4.1.1.)

At any point in time, we guarantee that each core will be given a minimum amount resources: one cache region (128 kB), and the power to run at minimum frequency (800 MHz in our setup, Section 5) for free. The remaining cache capacity and power budget are allocated using market-based mechanism. This is to guarantee that each application is able to at least run regardless of its purchasing power.

We now address the two major challenges in designing the market: how to model the players' utility, and how the players bid to maximize their utility.

4.1.1 Utility Function

In the multicore resource allocation problem that we study, we define an application's utility to be its IPC, normalized to the IPC when it's running alone (and thus owns all the resources): $U_i(\mathbf{r_i}) = \text{IPC}(\mathbf{r}_i)/\text{IPC}_{\text{alone}}$. Thus, the possible values of U_i are between 0 and 1. To figure out the performance–resource relationship of the utility function, we adopt the monitoring technique of our recent XChange work [37]: We divide the total execution time of an application into compute and memory phases. The length of the memory phase under different cache allocations is estimated using UMON shadow tags [27] and a critical path predictor [24]. The length of the compute phase and the corresponding power consumption is estimated using the power model developed by Isci et al. [9]. The sum of both phases is an estimation of the execution time given a particular cache-power allocation. Note that this is all modeled dynamically online; no prior off-line profiling is needed whatsoever [37].

Recall that in Section 2, in order to apply the theoretical results, the utility function of a player is required to be concave, continuous, and non-decreasing in shape. However, in computer architecture, this is not always true. On the one hand, power is known to be concave [9, 24], and fine-grained enough to regard it as continuous. For example, Intel's RAPL technique allows setting the CPU power at a granularity of 0.125W [20]. On the other hand, it is well-known that cache capacity is a non-concave, and non-continuous resource [2, 27].

Figure 2 shows the cache utilities of two representative applications, *mcf* and *vpr*. The markers are the utilities (normalized IPC) of each application when it is given different cache ways (no change in power budget). From the figure, we can make two observations:

First, such a utility function is not continuous, as it is partitioned by cache ways which are relatively coarse-grained. To make it continuous, we adopt *Futility Scaling* by Wang and Chen [36], a feedback control mechanism that can precisely keep the partition size close to a target at the granularity of cache lines. We empirically set the allocation granularity to 128 kB, and we call this a *cache region*.

Second, cache utility may not be concave. Although *vpr* shows a concave utility function, *mcf* clearly is not: its normalized utility is flat at 0.2 for 1 to 10 cache ways, and suddenly increases to 1.0 once it secures 12 ways (1.5MB). This is because *mcf*'s working set size is 1.5 MB, and 12 cache ways or more will satisfy its need by reducing the L2 cache miss rate to be almost zero. To address this issue, we apply *Talus*, a technique to convexify cache behavior [2]. Talus works roughly as follows: First, based on an application's actual cache utility, its "convex hull" is derived, which is the convex set of the cache utility. The cache allocation points on the hull are called "point of interest" (*PoI*), which are the desired allocations. Next, to make cache utility continuous on the convex hull, Talus divides the cache partition of a core into two "shadow" partitions. Given an arbitrary cache partition target, Talus first finds its two neighboring *PoIs*, and then adjusts the size of the shadow partitions accordingly. The access stream is also divided correspondingly into the two shadow partitions. More details can be found in Talus [2]. As shown in Figure 2, Talus effectively convexifies the cache behavior to a convex hull, which satisfies the requirement of being concave and non-decreasing.

4.1.2 Bidding Strategy

Now that we have constructed a utility function for each player, according to the market's bidding–pricing procedure (Section 2), the next problem is how each player finds its optimal bids to maximize its utility. Because both cache and power utilities are concave, heuristics such as hill climbing are appropriate in finding optimal solutions [2]. Therefore, we adopt a simple hill-climbing technique as follows:

1. Each player i splits its budget B_i into equal bids b_{ij} across all resources. In addition, S, which is the amount by which a player will shift its budget across resources, is set to be half of the bid.

2. Each player i computes the marginal utility λ_{ij} of all resources j. According to the optimality condition in Equation 4, if a player's bids are such that they maximize the player's utility, then the marginal utilities of all resources which receive non-zero bids are necessarily identical—in other words, the player has no incentive to re-allocate its budget across resources. Otherwise, if λ_{ij} varies for different resources under the current bids, the player will move an amount S of money from a resource k with lower λ_{ik} to another one k' with higher $\lambda_{ik'}$, and such a move will tend to equalize the marginal utility of these two resources (recall that the player's utility function is concave, which means that marginal utility λ_{ij} decreases as bid b_{ij} increases).

3. S is halved, and the process is repeated, until one of the following two conditions is met: *(a)* Either λ_{ij} of the resources stays the same (within 5% difference); or *(b)* S is smaller than 1% of the total budget. Because S decreases exponentially with every step, and λ_{ij} is monotonic, such an algorithm will quickly reach an optimal bids to the resources.

4.2 Budget Re-assignment Algorithm

The mechanism discussed so far applies to individual players in the context of a general budget-constrained market-based mechanism. In this section, we describe ReBudget, a heuristic that works on top of the above mechanism. ReBudget assigns different budgets to players in an adjustable manner, so that a trade-off between system efficiency and fairness can be made.

As is discussed Section 3, MUR and MBR are good metrics to indicate multicore efficiency and fairness. Using MUR, we can identify the lower bound of system efficiency, as shown in Theorem 1. Moreover, due to the concavity of utility, player i's λ_i increases if its budget B_i is reduced. As a result, by reducing the budget of a player with low λ_i, system MUR, which is the maximum variation of marginal utility λ_i across the market players, will move closer to 1, potentially yielding a higher efficiency. On the other hand, however, by creating a larger budget variation across players, MBR will decrease (recall from Section 3.2 that larger budget variation will decrease MBR), therefore opening the door for the level of fairness (i.e., envy-freeness) to decrease as well.

In ReBudget, we attempt to maximize efficiency while guaranteeing a certain level of fairness. The system administrator can set a lowest acceptable envy-freeness level, and using Theorem 2, the minimum MBR can be computed. Then, the budgets of the players are re-assigned based on their λ_i value under market equilibrium: those with lower λ_i will get a reduction in budget. We define a player to be "low λ_i" if its λ_i is smaller than 50% of the maximum λ_i—recall that, in Theorem 1, we find that when MUR is smaller than 0.5, the PoA guarantee starts to decrease linearly. However, at any point in time, the budget variation across players has to be maintained higher than the set MBR value.

We design an iterative method with exponential back-off:

(1) MBR is computed based on the lowest acceptable fairness level set by the administrator. To start the bidding process for the market, each player is assigned an equal budget B. The amount of budget re-assignment, named *step*, is initialized to be $(1 - \text{MBR}) \cdot \frac{B}{2}$. (2) Players then use their budget to conduct the algorithm we described in Section 4.1.2 to reach a market equilibrium. (3) λ_i of each player is collected. If a player i's λ_i value is lower than 50% of the maximum λ_i in the market, its budget is reduced by one *step*. (4) *step* is halved, and the algorithm returns to (2) to find a market equilibrium again. When *step* is smaller than 1% of each player's initial budget, or when no player's budget is decreased, the resulting market equilibrium is the final outcome.

This algorithm has two advantages: (1) The highest possible budget of any player is B, and the lowest possible budget is MBR $\cdot B$ (if the player gets a budget decrease in all iterations). Therefore, the maximum variation of players' budget will stay within MBR, and the fairness level set by the designer is guaranteed. (2) The exponentially decreasing *step* ensures that the process is fast, so that the market is still efficient and scalable to deal with large-scale systems.

The above exponential back-off greedy algorithm is a show case of how MUR and MBR can be used to guide budget re-assignment to optimize the system. As we will show in Section 6, such an algorithm is in fact fast and efficient in practice.

4.3 Implementation

We now discuss the hardware and software implementation of ReBudget. On the hardware side, ReBudget requires:

1. Hardware monitors to model an application's utility–resource relationship. As is discussed in Section 4.1.1, we adopt the same monitoring hardware as we do in XChange. As a result, the overhead is 3.7 kB per core [37], and the total overhead of the monitors is less than 1% of the total cache.
2. Extra states per partition and per cache line for partitioning the cache. We adopt Futility Scaling to partition the L2 cache. and it incurs around 1.5% storage overhead of the total cache [36].

On the software side, in order to handle the changing resource demands due to context switches and application phase changes, our budget re-assignment algorithm described in Section 4.2 is triggered every 1 ms to re-allocate resources. Similar to XChange, such an algorithm can be piggybacked to the Linux kernel's APIC timer interrupt, with a low runtime overhead [37].

5. Experimental Methodology

5.1 Architectural Model

We evaluate ReBudget using SESC [28], a highly detailed execution-driven simulator, which we modified in-house to suit our experimental setup. We model 4-way out-of-order cores; Table 1 shows the most important parameters of the CMP. We also faithfully model Micron's DDR3-1600 DRAM timing [19].

Table 1: System configuration.

Chip-Multiprocessor System Configuration	
Number of Cores	8 / 64
Power Budget	80 W / 640 W[a]
Shared L2 Cache Capacity	4 MB / 32 MB
Shared L2 Cache Associativity	16 / 32 ways
Memory Controller	2 / 16 channels
Core Configuration	
Frequency	0.8 GHz - 4.0 GHz
Voltage	0.8 V - 1.2 V
Fetch/Issue/Commit Width	4 / 4 / 4
Int/FP/Ld/St/Br Units	2 / 2 / 2 / 2 / 2
Int/FP Multipliers	1 / 1
Int/FP Issue Queue Size	32 / 32 entries
ROB (Reorder Buffer) Entries	128
Int/FP Registers	160 / 160
Ld/St Queue Entries	32 / 32
Max. Unresolved Branches	24
Branch Misprediction Penalty	9 cycles min.
Branch Predictor	Alpha 21264 (tournament)
RAS Entries	32
BTB Size	512 entries, direct-mapped
iL1/dL1 Size	32 kB
iL1/dL1 Block Size	32 B / 32 B
iL1/dL1 Round-Trip Latency	2 / 3 cycles (uncontended)
iL1/dL1 Ports	1 / 2
iL1/dL1 MSHR Entries	16 / 16
iL1/dL1 Associativity	direct-mapped / 4-way
Memory Disambiguation	Perfect

[a] We anticipate multicore chips with different number of cores will not be fabricated under the same technology, and thus expect the power consumption per core to decrease with the technology. For simplicity, in our evaluation, we use a chip TDP of 10 W per core and a 65-nm power model.

We use Wattch [7] and Cacti [30] to model the dynamic power consumption of the processors and memory system. We adopt Intel's power management approach for Sandy Bridge [29] to approximate the static power consumption as a fraction of the dynamic power that is exponentially dependent on the system temperature [10]. The run-time temperature of the chip multiprocessor is estimated using Hotspot [31] integrated with SESC.

Our multicore chip is able to regulate two shared on-chip resources: power budget and shared last-level cache. The power budget is regulated via per-core DVFS, similar to Intel's RAPL technique [20]. Each core can run at a frequency between 800 MHz and 4 GHz, as long as the total power consumption remains within $p \times 10$ W, where p is the number of processor cores. The last-level (L2) cache is partitioned using Futility Scaling by Wang and Chen [36], at the granularity of 128 kB (one cache region). The total L2 cache capacity is $p \times 512$ kB. Due to the overhead of UMON shadow tags [27], we limit its stack distance to be 16, i.e., the shadow tags can estimate the miss rate for the cache with capacities from 128 kB to 2 MB. We empirically observe that very few of our applications benefit from cache regions larger than 2 MB, and even if they do, such a large cache region is usually not affordable given the limited budget of each player. With a dynamic sampling rate of 32, the shadow tags take up 3.6 kB per core, which is less than 1% of the L2 cache size.

We guarantee that each core will have at least one cache region, and sufficient power budget to allow it to run at the minimum frequency (800 MHz). The remaining resources will be entirely distributed (i.e., no leftovers) based on a market-based resource allocation decision.

Workload Construction

We use a mix of 24 applications from SPEC2000 [32] and SPEC2006 [33] to evaluate our proposal. Each application is cross-compiled to MIPS ISA with -O2 optimization using gcc 4.6.1. For all simulations, we use Simpoints [8] to pick the most representative 200-million dynamic instruction block of each application.

Our evaluation is based on multiprogrammed workloads because we anticipate to allocate resources at the granularity of cores. For multithreading workloads, we can still allocate the resources at thread granularity if each thread is running on a different core.[2] Another choice is to allocate resources at the granularity of applications. All the threads of one application may share the same resources, which is a reasonable assumption, because the demand of the threads tend to be similar across threads of a parallel application in many programming models.

To construct our multiprogrammed workloads, we classify the 24 applications into four classes based on profiling: *Cache-sensitive* (C), *Power-sensitive* (P), *Both-sensitive* (B), and *None* (N). Then, we create six categories of multiprogrammed workloads: *CPBN*, *CCPP*, *CPBB*, *BBNN*, *BBPN*, and *BBCN*. For each category, we randomly generate 40 workloads for 8- and 64-core configurations. The random generation works as follows: for an 8-core (64-core) configuration, 2 (16) applications are randomly selected from each application class (e.g., *CPBN* means 2 (16) applications in each of *C*, *P*, *B*, and *N*, whereas *CCPP* will have 4 (32) applications in *C* and 4 (32) in *P*).

Performance Metrics

A key issue in resource allocation is the figure of merit. As is discussed in Section 4.1, we define an application's utility to be its IPC normalized by the IPC when it's running alone: $U_i(\mathbf{r_i}) = \mathrm{IPC}(\mathbf{r}_i)/\mathrm{IPC}_{\mathrm{alone}}$. The system efficiency can therefore be computed as:

$$\text{Efficiency} = \sum_{i=1}^{N} U_i = \sum_{i=1}^{N} \frac{\mathrm{IPC}_i(\mathrm{r}_i)}{\mathrm{IPC}_i^{\mathrm{alone}}} \tag{5}$$

We realize that this is exactly *weighted speedup*, a common metric to measure system throughput, and has been widely accepted by the community. Therefore, our *Price of Anarchy* study becomes meaningful in computer architecture: it guarantees the lower bound of throughput in market equilibrium.

A system could achieve high throughput (i.e., weighted speedup) by starving one or two applications while benefiting all the others; however, system fairness would suffer as a result, providing a bad experience to some users. Therefore, we evaluate fairness using *envy-freeness* shown in Definition 3, which is a widely used metric in economy, and has recently been introduced by Zahedi and Lee to evaluate fairness in multicore chips [39].

6. Evaluation

We evaluate our proposal in two phases. In the first phase, we analytically study the effectiveness of ReBudget by assuming that the applications' utility functions can be perfectly modeled and convexified. For this phase only, we extensively profile each application using our simulation infrastructure. (Recall that ReBudget does not require off-line profiling; our second phase models utility functions at run-time using hardware monitors, as described in Section 4.1.1.)

We sample 90 cache+power configuration points, with {1-6, 8, 10, 12, 16} cache regions (10 possible allocations) and {0.8,1.2,1.6,. . ., 4.0} GHz (9 possible allocations). For each point, we collect average IPC and power consumption.[3] Then, we derive the convex hull of cache and power, and assume that their utilities are perfectly concave and continuous. Finally, we analytically evaluate the system efficiency and fairness by applying ReBudget and other competing mechanisms to all 240 bundles.

In the second phase, we evaluate ReBudget in a simulated CMP environment (SESC). The application's utility is monitored at run-time, using the technique described in Section 4.1.1. We apply *Talus* [2] and *Futility Scaling* [36] to make cache behavior concave and continuous. Because of practical simulation time constraints, we randomly select one application bundle per category, and run it using detailed simulations. We use these results to validate our first phase evaluation.

We conduct all the experiments on 8- and 64-core CMP configurations, and find that the results are similar. Therefore, we omit the results for the 8-core configuration, and focus on the large-scale 64-core configuration.

[2] Skewing resources within threads in a multi-threaded application (e.g., to alleviate synchronization imbalance) is beyond our scope, can be incorporated orthogonally to our approach, and has been studied elsewhere [3, 16].

[3] Without loss of generality, our evaluation assumes that allocating more than 16 cache regions (2 MB) does not yield any significant additional utility to any application. This is reasonable for the input sets of the applications studied, and it allows us to complete profiling in a reasonable time.

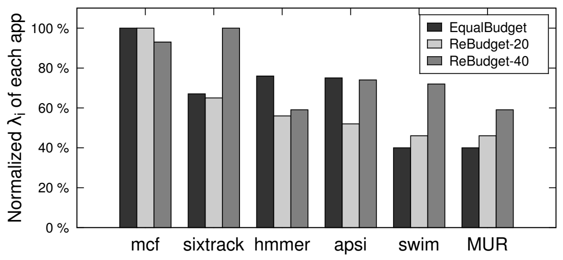


Figure 3: Marginal utility λ_i of each application in a sample *BBPC* bundle. The multiple copies of the same application in the bundle behave essentially the same way, so only one of each is shown. λ_i is normalized to the maximum λ_i in the bundle. MUR metric is also shown.

The allocation mechanisms evaluated are as follows: *EqualShare*, where resources are equally partitioned among all processor cores. Two XChange mechanisms [37]:[4] *EqualBudget*, where resources are partitioned using a market-based procedure assuming equal budgets for all players; and *Balanced*, where each player receives a budget proportional to the utility difference between its maximum (2 MB L2 cache and 4.0 GHz frequency) and minimum (128 kB and 800 MHz) possible allocations, normalized to the former. *ReBudget-step*, where resources are allocated using our ReBudget mechanism such that, at the end of the first iteration (where all players run with equal budget), each player is assigned either its original budget or *step* less. (Recall that *step* is halved in each subsequent iteration.) The initial budget is set to be 100 for all players. Finally, *MaxEfficiency*, which is the resource allocation maximizing system efficiency, is obtained by running an infeasible very fine-grained hill-climbing search (recall that all utilities are concave).

6.1 Efficiency

For the first phase evaluation, Figure 4 reports the efficiency and envy-freeness for EqualShare, MaxEfficiency, as well as XChange's EqualBudget and Balanced, and ReBudget with different aggressiveness (i.e., *step*) based on run-time feedback. The bundles are ordered by the efficiency of EqualShare. We observe that the efficiency of EqualShare compared to MaxEfficiency suddenly increases for the last 40 bundles. Most of these bundles fall into the category of BBPN. EqualShare works well for BBPN workloads because 75% of the apps in the bundle are power sensitive, and equally distributing the power in EqualShare works reasonable well. In addition, although the efficiency can be improved by giving more cache to "B" apps, they are not as cache-sensitive as "C" apps. Therefore, EqualShare in cache performs better for BBPN bundles than others such as CPBN.

[4] Note we convexify applications' utility, which is an improvement over the original XChange.

6.1.1 EqualBudget vs. EqualShare

We first compare the efficiency between EqualBudget market-based mechanism with EqualShare allocation. Figure 4a shows that in a 64-core configuration, 37% of the workloads in EqualBudget are able to achieve 95% of the social welfare of the MaxEfficiency, and over 90% bundles are within 90%. This proves that the market-based mechanism is robust, efficient, and scalable in this setup.

However, there are still 10% of the applications below 90% of the welfare in optimal allocation. We observe that over half of these workloads fall in the category of *BBPC*, where the number of applications favoring both resources (50%) is much higher than the number of applications favoring power (25%) or cache (25%) only. This is reminiscent of the well-known *Tragedy of Commons* [18]. MaxEfficiency strongly favors the apps which prefer only one resource, and the "B" apps, which need both resources, are sacrificed for the sake of higher system efficiency. The EqualBudget mechanism allows all the applications to fairly contend with each other, even though the price is an efficiency that is not as good as MaxEfficiency's.

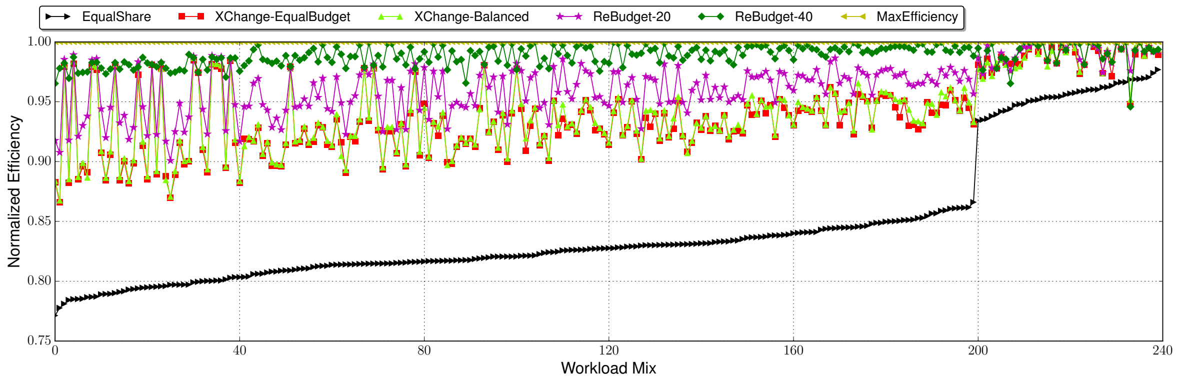
We look closely at an 8-core experiment using a BBPC bundle that contains four "B" apps (*apsi* and *swim*, 2 copies each), two "C" apps (2 copies of *mcf*), and two "P" apps (*hmmer* and *sixtrack*). The overall efficiency of such bundle with EqualBudget is 90% of MaxEfficiency, and we find its MUR $= 0.40$. Figure 3 shows λ_i value of each app at market equilibrium. We can find that with EqualBudget, "B" app *swim* has the lowest λ_i value, indicating that it is over-budgeted and not use its money efficiently; on the other hand, "C" app *mcf* has the highest λ_i value, showing that a budget increase can lead to a high utility gain.

6.1.2 XChange-Balanced

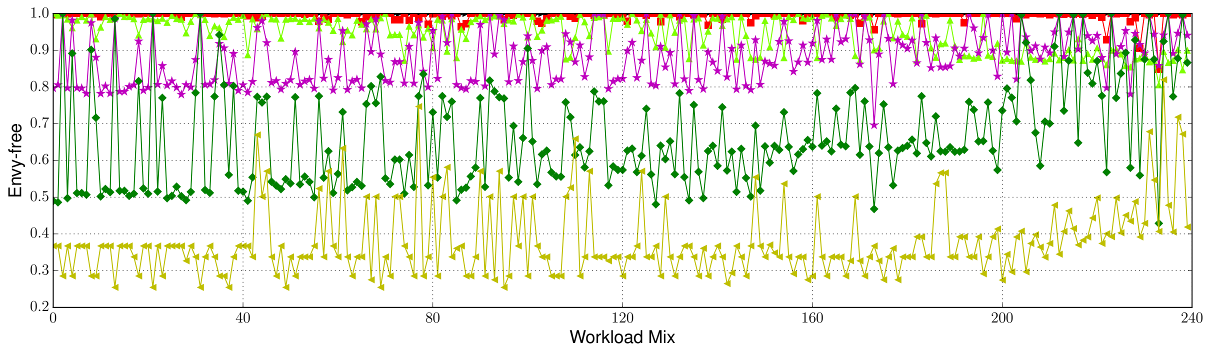
XChange's Balanced budget assignment is an intuitive way to distribute budget among players to improve efficiency [37]. However, Figure 4a shows that it does not outperform EqualBudget in efficiency too much, but in fact it loses in fairness, as shown in Figure 4b. The reasons are: (1) With the exception of "N"-type apps, which are not sensitive to any resources, the performance difference between minimum and maximum utility of most apps are similar, especially when we give out minimum resources for each player for free. Therefore, the budget assignment is not very different from EqualShare. (2) Blindly setting the player's budget B_i proportionally to his "potential," while ignoring the shape of utility and MUR metric, is ineffective.

6.1.3 ReBudget

We evaluate the our ReBudget mechanism proposed in Section 4.2. We test different aggressiveness, by setting the amount of budget decrease at each iteration (i.e., *step*) to be 20 and 40. Figure 4a clearly shows that by re-assigning the budget more aggressively, efficiency will improve for all



(a) 64-core Efficiency. Higher is better.



(b) 64-core Envy-freeness. Higher is better.

Figure 4: Comparison of system efficiency (weighted speedup) and envy-freeness among the proposed mechanisms in a 64-core configuration. System efficiency results are normalized to MaxEfficiency. Workloads are ordered by the efficiency of EqualShare.

bundles. Also, for all the 240 bundles, ReBudget-40 achieves 95% of system efficiency of MaxEfficiency.

We look closely at the same 8-core bundle we study in Section 6.1.1. For ReBudget-20, Figure 3 shows that *swim*, whose λ_i at 0.40 is the lowest under EqualBudget, has its value increased to 0.46, because its budget drops from 100 (every player's initial budget) to 61.25 units (minimum budget under ReBudget-20). On the other hand, *mcf*, which has the highest λ_i, has its budget unchanged (100). The budget of other apps are lowered to around 80, because their λ_i values are significantly lower than *mcf*. Note that λ_i of *apsi* and *hmmer* decrease, even though their budgets are reduced. This is because the budget cut of *swim* makes the prices of resources drop significantly. As a result, although *apsi* and *hmmer*'s budgets are reduced, they can actually afford more resources, and their λ_i decreases. Overall, the MUR of ReBudget-20 increases to 46%, compared to 40% in EqualBudget. Correspondingly, the efficiency of ReBudget-20 increases to 96% of MaxEfficiency.

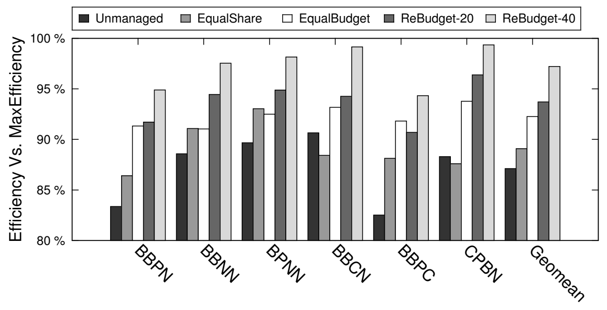
In ReBudget-40, *swim*, whose λ_i is still the lowest in ReBudget-20, get a further budget cut to 20. As a result, its λ_i value increases from 0.46 to 0.72, as shown in Figure 3. *mcf* in this case is no longer the highest in λ_i value: *sixtrack*'s budget is decreased to 30 and it starts to request for more money. Therefore, MUR of the system is increased to 0.59, and the efficiency is now 99% of MaxEfficiency.

6.2 Fairness

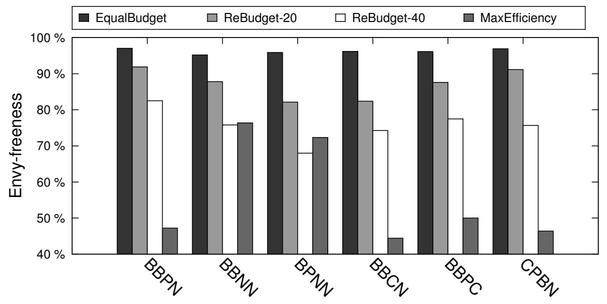
We use envy-freeness as the metric to evaluate the fairness, as is discussed in Section 2.3. We first look at the fairness comparison between EqualBudget and MaxEfficiency. As expected, Figure 4b shows that EqualBudget is almost envy-free, where in the worst case, it is still 0.93-approximate envy-free. On the contrary, MaxEfficiency is unfair, which is typically 0.35-approximate envy-free.

Regarding the XChange-Balanced mechanism, the envy-freeness of most workloads stays at 0.9, where in the worst-case it is 0.86-approximate envy-free. It is not as good as EqualBudget, and considering its trivial efficiency gain, and no control over the aggressiveness in making trade-off between efficiency and fairness, we consider such a mechanism to be ineffective.

On the other hand, the envy-freeness of ReBudget has a direct relationship with its aggressiveness. Figure 4b shows that the typical envy-freeness of ReBudget-20 and ReBudget-40 are 0.8 and 0.5, respectively, and none of the



(a) 64-core Efficiency in SESC. Higher is better.



(b) 64-core Envy-freeness in SESC. Higher is better.

Figure 5: Comparison of system efficiency (weighted speedup) and envy-freeness among the proposed mechanisms in a simulated 64-core configuration. System efficiency results are normalized to MaxEfficiency.

bundles violates the theoretic guarantee provided by Theorem 2 (0.53 and 0.19). We notice that there is a gap between the theory and reality. This is because what Theorem 2 states is a theoretic lower bound, which should stand in all cases. Such bound is tight, and it is not hard to construct a market to reach it[5]. Although it does not happen on the applications we use, it could happen in the real life. In addition, we show the envy-freeness of ReBudget-20 is consistently higher than ReBudget-40 for all bundles. Therefore, besides the theoretic guarantee, MBR can be used as an accurate indicator of system fairness.

Combined with the findings in Section 6.1, we can conclude that the more aggressive budgets are adjusted, the higher efficiency, and correspondingly the lower fairness it is achieved. This is appealing, because system designers and administrators can use the *step* as a “knob” to trade off one for the other.

6.3 Simulation Results

Besides the above analytical results, we implement ReBudget in architectural simulator SESC. Figure 5 shows the system efficiency and envy-freeness of the competing mechanisms. Such results are consistent to our analytical evaluation above: ReBudget improves system efficiency over EqualBudget by sacrificing fairness, and the more aggressive budget is re-assigned, the more efficiency improvement it is achieved. On the other hand, EqualBudget achieves the highest in envy-freeness, and MaxEfficiency, which targets at maximizing system efficiency, is the worst in fairness. ReBudget successfully maintains its rank between these two extremes, and aggressiveness gradually hurts system performance as expected.

[5] Zhang shows an example in EqualBudget case [40].

6.4 Convergence

A very important aspect of the market-based mechanism is how fast our algorithm is in finding the equilibrium allocation. To the best of our knowledge, there is no theoretic lower bound on the convergence time. However, in reality, we find that EqualBudget and XChange-Balanced converge within 3 iterations for 95% of the bundles. ReBudget mechanism spends a few more iterations, because it needs to re-converge after budget adjustment. However, the exponential back-off in budget change guarantees that the ReBudget process converges fast. These findings are in line with prior studies (e.g., Feldman et al. find the convergence time for a dynamic market is ≤ 5 iterations [17]). We also adopt a fail-safe mechanism for the very rare cases that market cannot converge: we simply terminate the equilibrium finding algorithm after 30 iterations.

7. Conclusion

In this paper, we have introduce two new metrics, *Market Utility Range (MUR)* and *Market Budget Range (MBR)*, which help us establish a theoretical bound for the loss of efficiency and fairness of a market equilibrium under a constrained budget, respectively. We have proposed *ReBudget*, a budget re-assignment technique that is able to systematically control efficiency and fairness in an adjustable manner. We evaluate ReBudget on top of our earlier XChange proposal for market-based resource management in CMPs, using a detailed simulation of a multicore architecture running a variety of applications. Our results show that ReBudget is efficient and effective. In particular, when combined with the proposed MUR and MBR metrics, ReBudget is effective at maximizing efficiency under worst-case fairness constraints.

A. Proofs

A.1 Proof of Theorem 1

Johari and Tsitsiklis [21] study the PoA for a market where the players are not constrained by a budget, and the players are maximizing their actual payoff: i.e., its utility minus its resource costs:

$$\text{maximize } U_i(\mathbf{b}_i) - \sum_j b_{ij} \tag{6}$$

We introduce *market utility range* metric MUR $= \frac{\min_i\{\lambda_i\}}{\max_i\{\lambda_i\}}$ into their proof, so that such proof can be generalized to a market with budget constraint. In a market equilibrium allocation $\mathbf{r}^n$ where player i bids $\mathbf{b}_i^n$, there exists $\lambda_i > 0$ for this player, such that for any resources j:

$$\begin{aligned}\frac{\partial U_i(\mathbf{r}_i^n)}{\partial b_{ij}} &= \frac{\partial U_i(\mathbf{r}_i^n)}{\partial r_{ij}}\frac{\partial r_{ij}}{\partial b_{ij}} \\ &= \frac{\partial U_i(\mathbf{r}_i^n)}{\partial r_{ij}}\frac{1}{p_j^n}(1-\frac{r_{ij}^n}{C_j})\begin{cases} = \lambda_i & \text{if} \quad b_{ij} > 0 \\ < \lambda_i & \text{if} \quad b_{ij} = 0\end{cases}\end{aligned} \tag{7}$$

Let $V_i(\mathbf{r}_i) = \sum_j \frac{\partial U_i(\mathbf{r}_i^n)}{\partial r_{ij}}(r_{ij} - r_{ij}^n) + U_i(\mathbf{r}_i^n)$. We find $\mathbf{r}^n$ for U_i is also the equilibrium allocation for V_i:

$$\frac{\partial V_i(\mathbf{r}_i^n)}{\partial b_{ij}} = \frac{\partial V_i(\mathbf{r}_i^n)}{\partial r_{ij}}\frac{\partial r_{ij}}{\partial b_{ij}} = \frac{\partial U_i(\mathbf{r}_i^n)}{\partial r_{ij}}\frac{\partial r_{ij}}{\partial b_{ij}}.$$

Define $\text{Nash}(U)$ to be the system efficiency in market equilibrium, where players have utility function U_i. Because the market equilibrium allocation for U_i is also the equilibrium allocation for V_i, $\text{Nash}(U) = \text{Nash}(V)$. Also define $\text{OPT}(U)$ to be the maximum feasible efficiency of the market. Due to the concavity of the utility function U_i, we have $V_i \geq U_i$ for any resource allocation $\mathbf{r}_i$. Therefore, $\text{OPT}(V) \geq \text{OPT}(U)$, and:

$$\text{Nash}(U)/\text{OPT}(U) \geq \text{Nash}(V)/\text{OPT}(V)$$

Define $\alpha_{ij} = \frac{\partial U_i(\mathbf{r}_i^n)}{\partial r_{ij}}$, $\beta_i = U_i(\mathbf{r}_i^n) - \sum_j \alpha_{ij} r_{ij}^n$, and $V_i(\mathbf{r}_i) = \sum_j \alpha_{ij} r_{ij} + \beta_i$. Let $W_i(\mathbf{r}_i) = \sum_j \alpha_{ij} r_{ij}$, $B = \sum_i \beta_i$. Then we have:

$$\frac{\text{Nash}(U)}{\text{OPT}(U)} \geq \frac{\text{Nash}(V)}{\text{OPT}(V)} \geq \frac{\text{Nash}(V) - B}{\text{OPT}(V) - B} = \frac{\text{Nash}(W)}{\text{OPT}(W)} \tag{8}$$

Next, we show the *Price of Anarchy* assuming players have utility W_i. The social welfare $W = \sum_i W_i(\mathbf{r}_i) = \sum_i \sum_j \alpha_{ij} r_{ij} = \sum_j \sum_i \alpha_{ij} r_{ij}$. It is easy to find that the optimal social welfare is achieved if for resource j, all C_j is given to the player i that has maximum α_{ij}:

$$\text{OPT}(W) = \sum_j C_j \max_i\{\alpha_{ij}\}$$

For market equilibrium allocation r^n, we have $\text{Nash}(W) = \sum_j \sum_i \alpha_{ij} r_{ij}^n$. Define $\text{Nash}_j = \sum_i \alpha_{ij} r_{ij}^n$. From Equation 7, each player has $\alpha_{ij}\frac{1}{p_j^n}(1-\frac{r_{ij}^n}{C_j}) = \lambda_i$ for any resource j if he submits a non-zero bid to it. Therefore, $\alpha_{ij} \geq p_j^n \lambda_i$. Without loss of generality, we define $\alpha_{1j} = \max_i\{\alpha_{ij}\}$. Then we have:

$$\begin{aligned}\text{Nash}_j &= \alpha_{1j} r_{1j}^n + \sum_{i\neq 1} \alpha_{ij} r_{ij}^n \geq \alpha_{1j} r_{1j}^n + \sum_{i \neq 1} p_j^n \lambda_i r_{ij}^n \\ &\geq \alpha_{1j} r_{1j}^n + p_j^n \min_i\{\lambda_i\}(C_j - r_{1j}^n) \\ &\geq \alpha_{1j} r_{1j}^n + \alpha_{1j}\frac{\min_i\{\lambda_i\}}{\lambda_1}(C_j - r_{1j}^n)(1-\frac{r_{1j}^n}{C_j}) \\ &\geq \alpha_{1j} r_{1j}^n + \alpha_{1j}\frac{\min_i\{\lambda_i\}}{\max_i\{\lambda_i\}}(C_j - r_{1j}^n)(1-\frac{r_{1j}^n}{C_j}) \\ &= \alpha_{1j}[\frac{\text{MUR}}{C_j}(r_{1j}^n - (1-\frac{1}{2\text{MUR}}))^2 + C_j(1-\frac{1}{4\text{MUR}})]\end{aligned}$$

Therefore, if $\text{MUR} \geq \frac{1}{2}$, $\text{Nash}_j \geq \alpha_{1j} C_j (1 - \frac{1}{4\text{MUR}})$, and:

$$\begin{aligned}\text{Nash}(W) = \sum_j \text{Nash}_j &\geq (1-\frac{1}{4\text{MUR}})\sum_j \max_i\{\alpha_{ij}\} C_j \\ &= (1-\frac{1}{4\text{MUR}})\text{OPT}(W) \geq \frac{1}{2}\text{OPT}(W)\end{aligned}$$

If $\text{MUR} < \frac{1}{2}$, $\text{Nash}_j > \alpha_{1j} C_j \cdot \text{MUR}$, then:

$$\text{Nash}(W) > \text{MUR} \cdot \text{OPT}(W)$$

Combined with Equation 8, we have:

$$\begin{aligned}\text{MUR} \geq \frac{1}{2} : \frac{\text{Nash}(U)}{\text{OPT}(U)} \geq \frac{\text{Nash}(W)}{\text{OPT}(W)} \geq (1-\frac{1}{4\text{MUR}}) \\ \text{MUR} < \frac{1}{2} : \frac{\text{Nash}(U)}{\text{OPT}(U)} \geq \frac{\text{Nash}(W)}{\text{OPT}(W)} \geq \text{MUR}\end{aligned}$$

A.2 Proof of Theorem 2

We introduce *market budget range* metric $\text{MBR} = \frac{B_i}{\max_j\{B_j\}}$ into Zhang's proof of envy-freeness with equal budget [40], so that our results apply to a market with an arbitrary budget assignment for players. The idea for proving a market is c-approximate envy-free is to prove that at market equilibrium, for any feasible bid vector $\mathbf{z}$ with $0 \leq z_j \leq y_{ij}^n$, and it is always the case that $\sum_j z_j = \max_i\{B_i\}$, $U_i(\mathbf{r}_i^n) \geq c \cdot U_i(\frac{z_j}{b_{ij}^n + y_{ij}^n})$.

To prove this, we define the player i's budget range: $\text{MBR}_i = \frac{B_i}{\max_j\{B_j\}}$. Similar to Zhang [40], we construct a matrix $\{\hat{b}_{jk}\}$ for $0 \leq j,k \leq M$, where M is the number of resources. Such matrix satisfies the following conditions:

$$\begin{aligned}\hat{b}_{jj} &= \min_j\{b_{ij}, z_j \cdot \text{MBR}_i\}, \\ \sum_k \hat{b}_{jk} &= b_{ij}, \\ \sum_j \hat{b}_{jk} &= z_k \cdot \text{MBR}_i\end{aligned}$$

The rest of the proof follows Zhang [40], combined with the fact that $\text{MBR}_i = \frac{B_i}{\max_j\{B_j\}} \geq \frac{\min_j B_j}{\max_k B_k} = \text{MBR}$.

Finally, we get that for any player i,

$$\begin{aligned}U_i(\mathbf{x}_i^n) &\geq (2\sqrt{1+\text{MBR}_i} - 2) \cdot U_i(\mathbf{z}) \\ &\geq (2\sqrt{1+\text{MBR}} - 2) \cdot U_i(\mathbf{z})\end{aligned}$$

Acknowledgments

We are grateful to Éva Tardos for her feedback during various stages of this project. We thank also the anonymous reviewers for their thoughtful feedback, which helped improve the paper. This work was supported in part by AFOSR Award FA9550-15-1-0311, and a gift from Qualcomm.

References

[1] M. Becchi and P. Crowley. Dynamic thread assignment on heterogeneous multiprocessor architectures. In *Intl. Conf. on Computing Frontiers (FC)*, 2006.

[2] N. Beckmann and D. Sanchez. Talus: A simple way to remove cliffs in cache performance. In *Intl. Symp. on High Performance Computer Architecture (HPCA)*, 2015.

[3] A. Bhattacharjee and M. Martonosi. Thread criticality predictors for dynamic performance, power, and resource management in chip multiprocessors. In *Intl. Symp. on Computer Architecture (ISCA)*, 2009.

[4] R. Bitirgen, E. Ipek, and J. F. Martínez. Coordinated management of multiple interacting resources in chip multiprocessors: A machine learning approach. In *Intl. Symp. on Microarchitecture (MICRO)*, 2008.

[5] S. J. Brams and A. D. Taylor. *Fair Division: From cake-cutting to dispute resolution*. Cambridge University Press, 1996.

[6] J. Brock, C. Ye, C. Ding, Y. Li, X. Wang, and Y. Luo. Optimal cache partition-sharing. In *Intl. Conf. on Parallel Processing (ICPP)*, 2015.

[7] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: a framework for architectural-level power analysis and optimizations. In *Intl. Symp. on Computer Architecture (ISCA)*, 2000.

[8] B. Calder, T. Sherwood, E. Perelman, and G. Hamerley. Simpoint. http://www.cs.ucsd.edu/ calder/simpoint/, 2003.

[9] A. B. Canturk Isci, C.-Y. Cher, P. Bose, and M. Martonosi. An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget. In *Intl. Symp. on Microarchitecture (MICRO)*, 2006.

[10] P. Chaparro, J. González, and A. González. Thermal-effective clustered microarchitectures. In *Workshop on Temperature-Aware Computer Systems*, 2004.

[11] J. S. Chase, D. C. Anderson, P. N. Thakar, A. M. Vahdat, and R. P. Doyle. Managing energy and server resources in hosting centers. In *ACM Symp. on Operating Systems Principles (SOSP)*, 2001.

[12] J. Chen and L. K. John. Predictive coordination of multiple on-chip resources for chip multiprocessors. In *Intl. Conf. on Supercomputing (ICS)*, 2011.

[13] J. Chen, L. K. John, and D. Kaseridis. Modeling program resource demand using inherent program characteristics. In *Intl. Conf. on Measurement and Modeling of Computer Systems (Sigmetrics)*, 2011.

[14] J. Chen, A. Nair, and L. John. Predictive heterogeneity-aware application scheduling for chip multiprocessors. *IEEE Trans. on Computers*, 63, 2012.

[15] S. Choi and D. Yeung. Learning-based SMT processor resource distribution via hill-climbing. In *Intl. Symp. on Computer Architecture (ISCA)*, 2006.

[16] E. Ebrahimi, R. Miftakhutdinov, C. Fallin, C. J. Lee, J. A. Joao, O. Mutlu, and Y. N. Patt. Parallel application memory scheduling. In *Intl. Symp. on Microarchitecture (MICRO)*, 2011.

[17] M. Feldman, K. Lai, and L. Zhang. A price-anticipating resource allocation mechanism for distributed shared clusters. In *Intl. Conf. on Electronic Commerce (EC)*, 2005.

[18] G. Hardin. The tragedy of the commons. *Science*, 162(3859): 1243–1248, 1968.

[19] M. T. Inc. 2Gb DDR3 SDRAM component data sheet: MT41J256M8. http://www.micron.com/parts/dram/ddr3-sdram/mt41j256m8da-125, July 2012.

[20] Intel Corporation. Intel 64 and IA-32 Architectures Software Developer's Manuals. http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html, 2014.

[21] R. Johari and J. N. Tsitsiklis. Efficiency loss in a network resource allocation game. *Mathematics of Operations Research*, 29(3):407–435, 2004.

[22] R. Kumar, K. I. Farkas, N. P. Jouppi, P. Ranganathan, and D. M. Tullsen. Single-isa heterogeneous multi-core architectures: The potential for processor power reduction. In *Intl. Symp. on Microarchitecture (MICRO)*, 2003.

[23] K. Lai, L. Rasmusson, E. Adar, L. Zhang, and B. A. Huberman. Tycoon: An implementation of a distributed, market-based resource allocation system. *Multiagent and Grid Systems*, 1, 2005.

[24] R. Miftakhutdinov, E. Ebrahimi, and Y. N. Patt. Predicting performance impact of dvfs for realistic memory systems. In *Intl. Symp. on Microarchitecture (MICRO)*, 2012.

[25] N. Nisan, T. Roughgarden, E. Tardos, and V. V. Vazirani. *Algorithmic game theory*. Cambridge University Press, 2007.

[26] C. Papadimitriou. Algorithms, games, and the Internet. In *Intl. Symp. on Theory of computing*, 2001.

[27] M. K. Qureshi and Y. N. Patt. Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches. In *Intl. Symp. on Microarchitecture (MICRO)*, 2006.

[28] J. Renau, B. Fraguela, J. Tuck, W. Liu, M. Prvulovic, L. Ceze, S. Sarangi, P. Sack, K. Strauss, and P. Montesinos. SESC simulator, January 2005. http://sesc.sourceforge.net.

[29] E. Rotem, A. Naveh, D. Rajwan, A. Ananthakrishnan, and E. Weissmann. Power-management architecture of the intel microarchitecture code-named sandy bridge. *IEEE Micro*, 32 (2):0020–27, 2012.

[30] P. Shivakumar and N. P. Jouppi. CACTI 3.0: an integrated cache timing, power, and area model. Technical report, HP Western Research Labs, 2001.

[31] K. Skadron, M. R. Stan, K. Sankaranarayanan, W. Huang, S. Velusamy, and D. Tarjan. Temperature-aware microarchitecture: Modeling and implementation. *ACM Trans. on Architecture and Code Optimization (TACO)*, 1(1):94–125, 2004.

[32] Standard Performance Evaluation Corporation. SPEC CPU2000. http://www.spec.org/cpu2000/, 2000.

[33] Standard Performance Evaluation Corporation. SPEC CPU2006. http://www.spec.org/cpu2006/, 2006.

[34] G. E. Suh, S. Devadas, and L. Rudolph. A new memory monitoring scheme for memory-aware scheduling and partitioning.

In *Intl. Symp. on High Performance Computer Architecture (HPCA)*, 2002.

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

[36] R. Wang and L. Chen. Futility scaling: High-associativity cache partitioning. In *Intl. Symp. on Microarchitecture (MICRO)*, 2014.

[37] X. Wang and J. F. Martínez. XChange: A market-based approach to scalable dynamic multi-resource allocation in multicore architectures. In *Intl. Symp. on High Performance Computer Architecture (HPCA)*, 2015.

[38] Y. Xie and G. H. Loh. PIPP: promotion/insertion pseudo-partitioning of multi-core shared caches. In *Intl. Symp. on Computer Architecture (ISCA)*, 2009.

[39] S. M. Zahedi and B. C. Lee. Ref: Resource elasticity fairness with sharing incentives for multiprocessors. In *Intl. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2014.

[40] L. Zhang. The efficiency and fairness of a fixed budget resource allocation game. In *Automata, Languages and Programming*, pages 485–496. Springer, 2005.