PowerTrader: Enforcing Autonomous Power Management for Future Large-Scale Many-Core Processors

Hang Lu, Guihai Yan, *Member, IEEE*, Yinhe Han, *Member, IEEE*, and Xiaowei Li, *Senior Member, IEEE*

Abstract—Existing power management approaches for modern many-core processors resort to "centralized" design concept, aiming to optimize chip performance under fixed power budget. Unfortunately, the centralized power management approach, which usually relies on a dedicated on-chip power manager, faces various limitations such as poor scalability and high implementation overhead, and hence cannot be deployed in future large-scale manycores. This article proposes *PowerTrader*, an *autonomous power management* scheme. PowerTrader endows each core with self autonomy to issue the power control at any time to harvest the desirable power quota through negotiating with vicinity cores. It does not incur the overheads introduced by power allocation and statistics collection that are inevitable in centralized approaches, meanwhile chip power consumption could be well kept beneath the preset power budget. This article also elaborates on the key design tradeoff in autonomous power management (i.e., Mean-Time-to-Stable versus application power efficiency), and provides thorough design space exploration to justify the efficacy of the proposed approach. Experimental results show that PowerTrader achieves substantial improvements in both performance and power, and exhibits superior scalability compared with the state-of-the-arts.

Index Terms—Many-core architectures, power management, on-chip interconnection networks (NoC)

1 Introduction

Along with the rapid growth of chip integration, power consumption has become a first-order design constraint in modern many-core processors. Ever-growing power consumption not only leads to increased energy and packaging costs, but also results in high die temperatures that may, in the worst case, jeopardize chip performance and reliability. Therefore, modern many-core processors seek to exploit dedicated power management schemes to carefully restrain the overall chip power consumption to stay beneath a certain power budget, and at the same time optimize performance, a.k.a. boosting the power efficiency of manycores.

To reach the power efficiency frontier, existing power management for manycores faces two grand challenges: the first one comes from the *scalability* limitation. Most of the solutions nails "centralized power management", which is initially designed for single-core or multi-core processors where scalability problem is not that critical. In these approaches, a centralized "power manager", usually implemented as an on-chip coprocessor [1] or firmware [2], [3], is employed to account for the power budget allocation of each core for the imminent control interval, with the help of dedicated power allocation algorithms [4] or prediction models [3], [5]. However, the complexity of these schemes, though not an issue in multicores, increases drastically and could not be simply ignored in *manycores*; for example, configuring a 100-core processor in the central manager may result in a temporal complexity of hundreds of seconds for some recently proposed schemes [6], [7]. While commercial trends have unveiled that future many-core processors will feature 100+ (i.e., Tilera Mx100 [8], Ambric MPPA [9]), or even 1,000+ (i.e., Adapteva-4096 [10]) number of cores, the scalability problem will be further exacerbated.

The scalability problem is not merely manifested in configuring optimal power state for the target cores. Runtime statistics collection and transmission, as another major bane, also stunts the deployment of the centralized approach. Extra I/Os or buses must be implemented on-chip, in addition to the already-existing data paths, for the delivery of runtime statistics of interest such as cache misses (i.e., MPKI) or core-level utilizations (i.e., IPC) back to the manager; final power state decisions also need to be delivered, in return, to each voltage/frequency controller to finalize the management. Such statistics and control information turn-around, however, does not scale well either, because the prolonged wires placed to facilitate the power manager not only complicate chip layout, but also result in an elevated data collection latency that will eventually lead to stale power state adaptations for the target cores.

In this article, we propose a novel "autonomous" power management approach, namely *PowerTrader*, to address these challenges in tandem. We use the term "autonomous" because PowerTrader no longer reports core runtime statistics

• *The authors are with the Institute of Computing Technology, Chinese Academy of Sciences, Kexueyuan South No. 6, Zhongguancun, Beijing 100190, China. E-mail: {luhang, yan, yinhes, lxw}@ict.ac.cn.*

Manuscript received 5 Feb. 2017; revised 11 Apr. 2017; accepted 30 Apr. 2017. Date of publication 5 May 2017; date of current version 13 Dec. 2017. (Corresponding author: Hang Lu.)
Recommended for acceptance by S. Pasricha.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TMSCS.2017.2701795

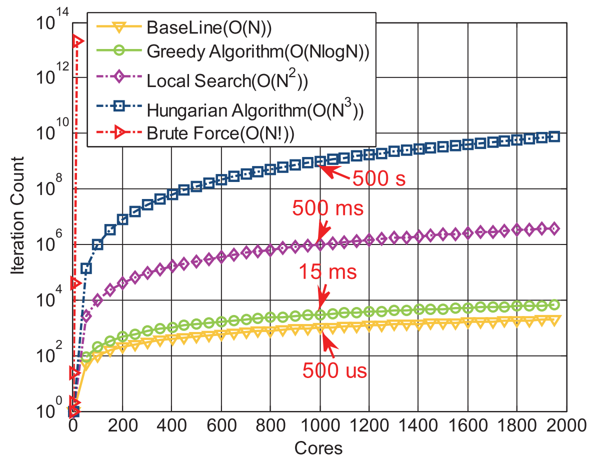


Fig. 1. Complexity comparison for state-of-the-art power allocation algorithms.[1] For future large-scale many-core processors designed for high performance computing, power management would be fatal if using existing centralized approaches.

back to the power manager, and then passively waits for the turn-around decision, a commonly-exercised strategy in centralized approach with poor scalability. Instead, each core is endowed with per-core autonomy that allows to spontaneously "probe and acquire" the desirable power quota at any time during manycore operation, and meanwhile chip power budget is still well guaranteed. The per-core autonomy is enforced by a dedicated *autonomous agent*. As key component to implement PowerTrader, it is responsible for monitoring core runtime power demand in different executing phases, acquiring desirable power quota and eventually actuating power control. Autonomous agents negotiate with each other via on-chip interconnection network (NoC), broadcasting power requests to the vicinity and then receiving, from vicinity agents, the responses containing the "about-to-provide" power quota. Core power state could only be upgraded if one or multiple agents in the vicinity are able to fully provide the requested power quota, or in other words, if chip power headroom allows.

The major advantage of the proposed PowerTrader is that it augments many-core power management with superior scalability and extremely low cost. The overheads as for the statistics collection and algorithm complexity suffering centralized power manager no longer exists, replaced with local inter-agent communications enclosed within certain physical ranges, and most importantly, agnostic of the actual scale of manycores.

2 Background and Motivation

2.1 Scalability Limitation in Many-Core Power Management

Previous studies mostly employ the so-called *centralized* power manager, possibly a co-processor or firmware, for capping peak power at chip-level or achieving power reduction for as much as possible. Hardware runtime statistics (always of all the cores in processor chip) must be transmitted back to the power manager for power control decision-making, usually at a huge temporal overhead of millions of cycles. The classic steps involve: (1) statistics monitoring and collection, (2) statistics transmission back to the manager, (3) power budget allocation for each core in the manager, and (4) decision delivery to the actuator. Even though we can use sophisticated hardware design to minimize the overhead of (2) and (4), and neglect the overhead of (1), *power allocation*, however, is still a huge barrier for many-core power management due to poor scalability.

As evidence, Fig. 1 plots the complexity of various commonly used power allocation algorithms introduced in [6] under 2 GHz processor frequency. It shows that even for a $O(n^2)$ algorithm (n is number of cores) is hard to deploy in reality. If we consider 1,000-core scale, the time penalty could reach 500 ms, and even "hundreds of seconds" for $O(n^3)$ algorithm. Unfortunately, most of the existing algorithms in recently proposed centralized approaches have worse complexity than $O(n^3)$. As one of the representatives, Isci et al. [11] proposes MaxBIPS that uses exhaustive searching ($O(N!)$ complexity) to find appropriate combination of voltage/frequency (V/F settings) for each core. The temporal overhead grows roughly more-than-exponentially after the algorithm has finished traversing each core, so that it is only workable in "multi-core" systems. Meng et al. [12] introduces a global power saving strategy for multi-core processor through calculation of power-performance estimates for different power modes. The evaluation interval for only 4 cores may exceed 600 μs as reported.

The increased time penalty inevitably results in increased power control interval, as a way to cover the overhead of power allocation; for example, if the allocation algorithm costs 500 ms to accomplish for all the cores, the total control interval would be 25 s, if we require the time penalty could only occupy 2 percent of the total control interval. It means power management can hardly work at fine granularity, risking opportunities to reach optimal power efficiency for the running application.

2.2 Autonomous Power Management

To tackle the scalability limitation, we employ an *autonomous* approach. In fact, as a general concept, a modeled instance with "autonomous" property implies that it can do self-adaptations without imposing restrictions to others. There are many autonomous-style management routines designed for various purposes. Some prior researches resort to self-adaptive power management for kilo-core processors; for example, [7] proposes using software-level agent for each malleable application as a way to autonomously manage the power states of its resources (i.e., cores). Similar technique is also shown in [13] which dynamic changes runtime application parallelism by borrowing or lending idle cores between each other to optimize overall system power efficiency. These techniques work at software-level by allocating each malleable application an autonomous agent, responsible for collecting statistics from hardware monitoring infrastructure and implement core expanding or shrinking according to a game-theoretic approach. Sartori and Kumar [14] also provides three techniques in operation system level, by employing efficient task mapping and

1. "Iteration" means basic operation that algorithm uses to calculate power allocation for "one core". In [6], one iteration is assumed to be 1,000 cycles under 2 GHz frequency, so allocating 1,000 cores under $O(n)$ complexity will hence cost $1{,}000\ cores \times \frac{1{,}000\ cycles}{2\ \mathrm{GHz}} = 500\ \mu s$. We also follow this assumption in plotting this figure.

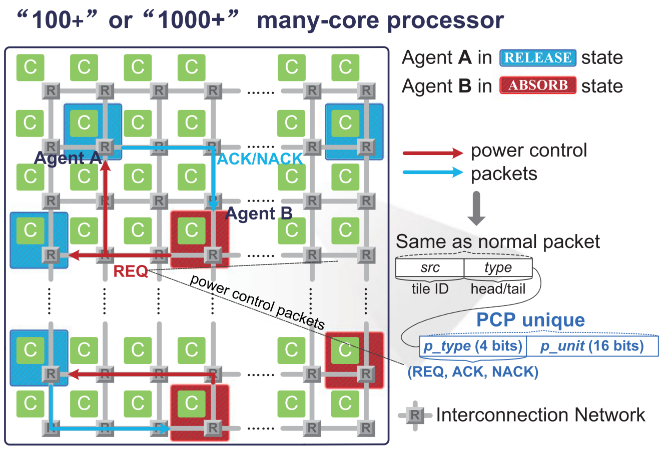


Fig. 2. Autonomous power management outline. In this figure, we present an example of two autonomous agents trading power, through dedicated power control packets (PCPs). The legend instructs the typical format of PCP and possible state an agent might be.

minimizing search space of global power management, to optimize performance under a fixed power budget. In [15], an multi-agent auction model is proposed to serve the same goal. It is a hierarchical globalized power management dividing many-core chip into several clusters governed by a global auctioneer, aiming at reducing execution time but taxed with an unavoidable communication overhead and complex agent state calculation, as stated in the paper.

Besides, researchers also employ autonomous management style to address the on-chip thermal alerts (dynamic thermal management, DTM). For example, [16] and [17] present two verification methodologies to validate the efficacy of distributed DTM schemes. [18] proposed a fully distributed agent-based DTM approach, focusing on evenly distributing power and the resulting heat across a many-core architecture. The approach leverages a classical supply/demand model to proactively deal with potentially developing thermal hotspots. State-of-the-art researches also propose to use scalable ways to control power density and thermal dissipation in dark silicon era, emerging in future large-scale manycores. Shafique and Garg [19] lists the current trends and research challenges in TDP-restrained computing system. [20] and [21] both propose to use power density/thermal-aware resource management for NoC-based heterogeneous architectures in order to maintain peak performance under fixed power/thermal budget.

While in this article, on top of the concept of autonomy, we develop a novel hardware based, autonomous power management approach for manycores and thoroughly validate its superior scalability. Apart from [18] that power trading is achieved only between neighbors, our approach allows regional agent interaction to harvest, in a wide range, possible power quota to upgrade to a certain new power state, with ignorable communication overhead. For the most recent work, [22] proposes a scalable method to calculate the desired power budget of each core, shortening the temporal overhead dramatically. Our work does not calculate power budget, but resorts to autonomous operation to automatically "maintain" power budget. Compared with [7] and [13], the instanced agent in our approach is purely hardware entities, responsible for monitoring/collecting runtime statistics locally, calculating desirable power state and finally controlling V/F regulators. Compared with software agent, hardware based agent is not impacted by software execution phases, i.e., thread scheduling, task mapping and synchronization, which means it can more conveniently observe core operations via various performance counters, and more timely coordinate its power state at a finer grain to effectively accommodate application power demands.

3 Enforce Autonomous Power Management—PowerTrader

3.1 General Concept

To illustrate the concept of PowerTrader, we take a tile-organized manycore as an example, as shown in Fig. 2. The processor is partitioned into multiple "*autonomous agents*" in colored blocks, which could be an arbitrary combination of on-chip hardware; for example, core with first and last level caches, on-chip routers, etc. as long as the power consumption could be individually tuned, but for simplicity, we focus on core power only because it is the most power consuming hardware in manycores, and each core serves as an autonomous agent. In the rest of this article, we use "core" and "agent" interchangeably.

Supposing core B is currently looking for available power quota, it broadcasts the power request (`REQ` in the figure) to its neighboring cores at certain timestamp. If any nearby agent, i.e., core A, just has excessive power quota that could satisfy the request imposed by core B, it acknowledges back (`ACK/NACK`) to core B. Then, if core B accepts the offer from A, its power state could be safely raised. As expected, the total power consumption of the chip remains identical, but the power state for A and B is reversed without the help of any centralized manager. It's just like power is directly shifted to the target core autonomously, and the operation is viable for each core across the entire chip. Hence, power management is represented as some sort of handshake communication between A and B, and that is also why we term it "PowerTrader". It is similar to commercial trade using currencies to exchange products, but the difference is that the "currency" here is actually the power tokens that travel amongst agent instances, used for probing available power quota and establishing power trading.

Obviously, PowerTrader has three major merits:

- *superior scalability*. Due to per-core autonomy, power management is transformed into localized operations, agnostic of the actual scale of manycores;
- *peak power guarantee*. Autonomous power management could work at power constrained scenario. Boosting power state for an agent is only allowable when chip power headroom allows, so the chip power budget will not be violated;
- *dynamic power adaptation*. It can not only adapt to "Turbo mode" that the overall power consumption approaches the preset power budget, but also accommodate "low-power mode" in which most of the cores are not craving for power. Most importantly, no centralized manager is necessary at all.

Implementing such autonomous power management, however, is challenging because an agent may encounter unpredictable scenarios during interacting with vicinity agents. For example, receiving redundant requests, frictions between power quota provided and requested, and unpractical "scope" configuration all have non-negligible impacts. In rest of this section, we elaborate the details of PowerTrader and how it works to enforce autonomous power management.

3.2 Inter-Agent PowerTrading

3.3 Agent States

Agent power state is used to depict the power demand under different execution phases. Intuitively speaking, it must involve the power requesting/providing phases, as well as the state that an agent has no particular power demand. Therefore, we define three states in PowerTrader framework as defined in Fig. 2:

- *Stable*: this is the normal state for the agent, which implies that the core is satisfied with the current power state configuration and does not exhibit any power control intention. The core neither requests power nor has extra power quota that can provide to other cores.
- *Absorb*: when a core requests more power, we call it is in "absorb" state, which will go back to "stable" as long as all the required power quota is provided by other cores.
- *Release*: when the power state of a core turns out to be over-provisioned, this state arises, and it will return to "stable" as long as some other cores in "absorb" state shift away all of its over-provisioned power quota.

3.3.1 Agent Communication

Major power control information is, unlike centralized approach, communicated directly between agents. PowerTrader employs dedicated *power control packets* (PCPs) to aid each agent in advertising desired power quota and receiving corresponding acknowledgements during trading. PCPs, like normal data packets, have identical payloads like `SRC` (source agent sending this PCP), `type` (head/tail flit label) etc., as shown in Fig. 2. Besides, it also includes two unique payloads within PowerTrader context: `p_type` indicates the control type of this PCP, which could be either `REQ`, `ACK` or `NACK`; `p_unit` indicates the desired power quota requested or to be provided. This field does not represent power quota in actual Watts; instead, it uses unified power quota; for instance, if an agent requests 50 watts of power, `p_unit` would be 5 units if we use 10 watts as one power unit, simplifying the subsequent decoding in the agent. Hence, a one-flit packet is adequate to convey all power control information.

Although PowerTrader enrolls additional PCP traffic, it is unnecessary to dedicate additional channels to deliver PCPs, considering the implementation overhead with respect to chip area and power. The already-existing interconnect infrastructure, such as the Networks-on-Chip (NoC) in modern many-core processors like Intel SCC [23] or Tilera Gx/Mx Families [8], could be directly exploited for the delivery of PCPs. In particular, we observe that normal on-chip traffic, i.e., memory access or cache coherence, are not burdened with PCP overlaid; their co-existence does not degrade the responsiveness of PowerTrader either, so new data channels for PCPs are not desirable at all. In Section 6.6, we will thoroughly investigate the impact of PCPs on the worst-case latency of normal data packets.

Algorithm 1. Power Trading Mechnism

```
Input: Received PCP: $p$; Agent current state: $cur\_state$;
       Power unit: $cur\_unit$
Output: Agent final state: $state$; Output PCP: $pcp\_out$;
/* Step 1: Interpret the received PCP */
1  $type = p->type$;
2  $p\_unit = p->unit$; // power quota requested/provided
   /* Step 2: determine state transition */
3  switch $cur\_state + type$ do
4      case $ABSORB + ACK$:
5          $cur\_unit -= p\_unit$; // reduce the required quota
6          if $cur\_unit == 0$ then
7              $state = STABLE$; // having obtained all quota
8              $set\_freq()$;
9          end
10         $pcp\_out = ACK(p\_unit)$; break;
11     case $STABLE + ACK$:
12         $pcp\_out = NACK(p\_unit)$; break;
13     case $ACK\_SENT + NACK$:
14         $state = RELEASE$; break; // continue release power
15     case $ACK\_SENT + ACK$:
16         $cur\_unit -= p\_unit$;
17         if $cur\_unit == 0$ then
18             $state = STABLE$; // having released all quota
19         else $state = RELEASE$; break;
20     case $RELEASE + REQ$:
21         $handle\_req()$; break;
22     otherwise break;// meaningless combinations, safely ignore if encountered
23 endsw
```

3.3.2 Power Trading Mechanisms

Multiple agents involved in power trading may act asynchronously in dispersing different type of PCPs, maybe REQs or (N)ACKs, across all agents in the vicinity, especially during REQ broadcast from "absorb" agents, who may have no prior knowledge of the locations of potential "release" agents. Therefore, scenarios that a "release" agent receives multiple REQs sourced in different "absorb" agents, or an "absorb" agent receives multiple ACKs from different "release" agents, may be frequently encountered. Each agent must be able to react to these unpredictable scenarios autonomously, and in PowerTrader, dedicated power trading mechanisms are introduced to fulfill this purpose.

Algorithm 1, as the key mechanism of PowerTrader, handles various combinations of local agent state and received PCPs. It first interprets `type` and `unit` information in line 1, followed the by the second state transition phase in line 3. cur_unit indicates the total power quota an agent intends to request or provide, contingent to the current state of the agent, whether "absorb" or "release". In particular, if an "absorb" agent receives an ACK, cur_unit will be subtracted

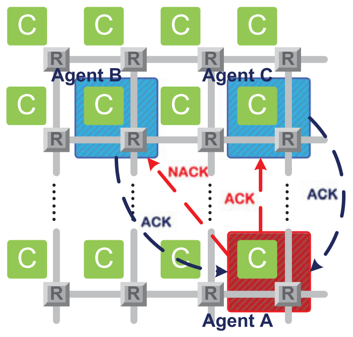
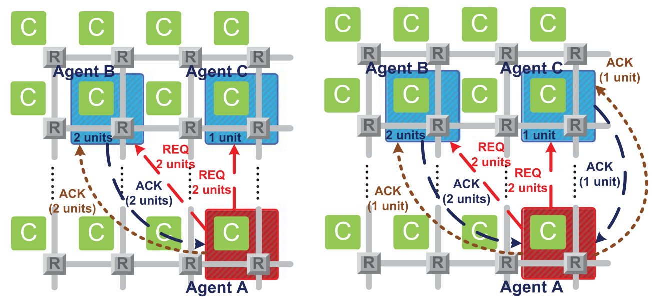


Fig. 3. Scenario when receiving redundant ACKs from different "release" agents.



(a) Bilateral Power Trade (BPT) (b) Multilateral Power Trade (MPT)

Fig. 4. Two power trading mechanisms in PowerTrader.

by the $p->unit$ in this ACK, and the agent will return to "stable" as long as cur_unit is reduced to 0, indicating the requested power quota is successfully "absorbed" by vicinity agents (line $4 \rightarrow$ line 10).

Handling Redundant Acknowledgements. For an "absorb" agent receiving multiple ACKs after REQ broadcasting, the "absorb" agent will only establish power trading with the "release" agent indicated by the first-received ACK, while send NACKs to the agents indicated by redundant ACKs, as shown in Fig. 3. That is based on the consideration of prompt power control responsiveness. The "release" agent could behold and serve other "absorb" agents upon receiving NACK, as shown from line $11 \rightarrow$ line 12. "ACK_SENT", a transient state for the "release" agent after sending ACK, could either turn to "stable" upon receiving ACK, or back to "release" again if NACK is received (line $13 \rightarrow 19$).

Bilateral/Multilateral Power Trading. The REQ broadcasted, on the other hand, may encounter "release" agents providing different amounts of power quota, maybe smaller than the amount REQ asks. "Release" agent may thereby have two choices: (1) just ignore this REQ, since other agents may have "*equal or larger*" power quota serving this REQ; (2) acknowledge this REQ, making the source agent, to some extent, approach stable state. In PowerTrader, we render this parameter configurable and implement two canonical mechanisms, namely *Bilateral Power Trading (BPT)* and *Multilateral Power Trading (MPT)* to accommodate two circumstances respectively.

Fig. 4 exemplifies BPT and MPT. In Figs. 4a and 4b, agent A is at "absorb", both agent B and C are at "release", holding available power quota. Following BPT strategy, only agent B will respond to A because of the matched power unit: agent A requires 2 units of power, while agent B can provide exactly 2 units. Whereas in MPT, both B and C will respond to agent A, so agent A will absorb 1 unit apiece, from B and C, and leave B with 1 unit. $handle\_req()$ subroutine (Algorithm 2) formalizes the operation of BPT and MPT, which is extremely light-weight, only including one major judgement to decide if ACK will be sent or not, according to the pre-set mode option.

BPT and MPT have respective pros and cons. Intuitively, MPT is more positive in accepting any quota of power, and thus more prone to obtain desired power quota, but it has to maintain a dynamic power unit countdown and issue handshakes with every contributive "release" agent, at the cost of generating more PCPs. BPT, however, is relatively simple to implement and results in less PCP traffic, but its "picky" nature may undermine the opportunity to serve more "absorb" agents. In Section 6, we will illustrate the impact of BPT and MPT on both application performance and power using multi-program workloads.

Algorithm 2. $handle\_req()$ Subroutine

```
  Input: Mode: $\_mode$; // BPT or MPT
  Output: Agent final state: $state$; Output PCP: $pcp\_out$;
1 if $cur\_unit \geq p->unit$ then
2   $state = ACK\_SENT$; $pcp\_out = ACK(p\_unit)$; //
    respond this REQ
3 else
4   if $\_mode == BPT$ then
      // just do nothing, with low implementation
      cost
5   else if $\_mode == MPT$ then
6     $state = ACK\_SENT$; $pcp\_out = ACK(p\_unit)$;
7   end
8 end
```

3.4 Key Design Tradeoff

Power trading, as regional operations across many-core topology, cannot always be accomplished for an "absorb" agent, as REQs are only broadcasted within its vicinity where "release" agents cannot always be guaranteed to exist. We call such broadcast range as the "*scope*" of PCPs, defined as a region centred with an "absorb" agent, in which the Manhattan Distance between any other agent and the centred agent remains identical, as shown in Fig. 5. Obviously, larger scope leads to larger opportunity to successfully accomplish power trading for the centred "absorb" agent; for example, "scope 4" is more beneficial than "scope 2" in Fig. 5. However, naively setting the scope excessively large would be detrimental, because large scope inevitably results in increased latency of power trading, and thus degrades the responsiveness of power management. Thus, we define a new metric, namely *"Mean-Time-to-Stable (MTTS)"*, to represent such responsiveness, and quantitatively it exhibits the latency overhead to accomplish power trading, that is, from "non-stable" state back to "stable" for an "absorb" agent. MTTS entails the application power efficiency, and usually we want MTTS as minimum as possible by narrowing the broadcast scope. For example in Fig. 5, "scope 2" apparently has a less MTTS than "scope 4", but it also sacrifices possibilities to meet more "release" agents as mentioned above, so selecting a proper broadcast scope

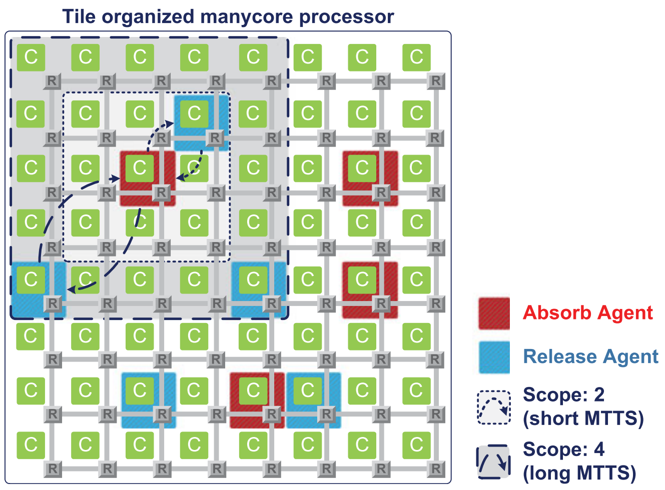


Fig. 5. Tradeoff of "broadcast scope" versus "MTTS". Obviously, the larger the broadcast scope, the more opportunities that "Absorb Agent" will meet "Release Agent"; however, it inevitably enlarges the latency (round trip cycles of PCP) to return to "Stable" state.

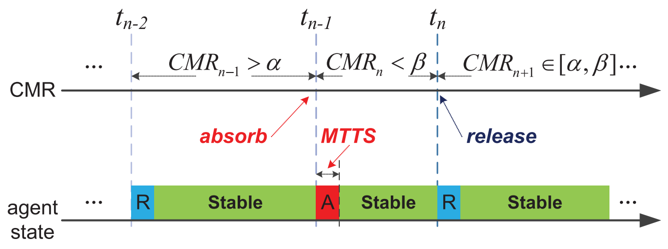


Fig. 6. Triggering a state transition, maybe from "stable" to "absorb" or "release" for an agent.

configuration is very important to the efficacy of power trading mechanism, for both BPT and MPT mechanisms.

System Convergence & Stability. Even if we could provide enlarged opportunity for "absorb" agents to attain "stable" state, through configuring a proper broadcast scope, we still cannot, however, guarantee *every* "absorb" agent could be served, especially when workload demonstrates oscillated behavior in different executing phases, manifested in frequent and uniform oscillation of agent power demand. When this case happens, the broadcast scope of an "absorb" agent may mostly involve "absorb" agents and barely has matched "release" agents for establishing power trading, so the "absorb" agents may keep waiting, staying starving to be served. Similarly, it is possible that "release" agents cannot meet any REQ requesting power quota. Such oscillation may cause autonomous power management to sink into instability and fail to converge. To tackle this worst case, PowerTrader enrolls a pre-set timer, associated with each agent, which starts counting when the agent transforms from "stable" state to "absorb" or "release". When timer expires and the agent still does not meet matched agents, it will be compulsorily turned to "stable", so that it has opportunity to proceed subsequent power state adaptation operations. In Section 6.5, we will demonstrate that this corner case barely happens under various timer settings. Many-core system will not incur instability or divergence, but even if it happens, PowerTrader manages to maintain stability as well. Besides, we will thoroughly study the MTTS tradeoff and conclude that a 6-hop scope is even adequate for achieving both high trading possibilities and a converged-and-stable many-core system.

4 Implementation in Manycores

4.1 Application Runtime Demand

As mentioned in Section 3, each agent may exhibit three possible states. Upon system initialization, the state is set to "stable" for each agent. The transition from "stable" to "absorb", or "release", depends highly on application phase changes. In order to satisfy each core on the fly, we allow the power state transition being triggered based on its own power demand, which is reflected by the variation of hardware events. Therefore, we enroll a classic metric, called "*CMI*" (Computation versus Memory Intensity, extended from [24]) to denote application (or more specifically, thread) affinity to computation or memory during execution, as defined in Eq. (1). $Cycles_{computation}$ is the actual time (in cycles) that the target core spent on executing instructions, while $Cycles_{memory}$ is the time that the core pipeline is stalled due to the accesses of the entire memory system, including last level cache and main memory. The final calculated CMI_n hence reflects the computation or memory intensity of the running thread at interval n

$$CMI_n = \frac{Cycles_{computation}}{Cycles_{memory}} \quad (interval\ n) \tag{1}$$

$$CMR_n = \frac{CMI_n}{CMI_{n-1}} \quad (interval\ n) \tag{2}$$

$$agent\ state = \begin{cases} stable \rightarrow absorb : CMR_n > \alpha \\ stable \rightarrow release : CMR_n < \beta \\ remain\ stable : CMR_n \in [\beta, \alpha]. \end{cases} \tag{3}$$

The quotient of two consecutive CMIs, namely CMR as defined in Eq. (2), thus represents the affinity variation or application phase changes at runtime, and we take CMR as the trigger of agent state transition: either switched from "stable" to "absorb" or "release". This metric denotes the variability of computation intensity versus memory intensity; for instance, CMR larger than 1 indicates an increasing trend towards computation intensive phase, and vice versa. As shown in Fig. 6, each agent will frequently access core-level performance counters to obtain pipeline status and calculate CMR, but note that it is different from centralized approaches using fixed control interval [3], [4], [25]; in PowerTrader, state switching is only necessary when CMR increases or decreases to certain thresholds (α, β in Eq. (3)), so each control interval possibly does not have the same length. For example at time t_{n-1}, CMR has increased beyond the threshold, so the current power state could be upgraded to "absorb"; otherwise, the power state will be degraded to "release".

The idea behind selecting this metric stems from the adaptation of application runtime demand, which could be conveyed if observed in a finer granularity, but we also believe that other methods (i.e., [26]) also fit in PowerTrader framework as long as it could precisely reflect compute/memory

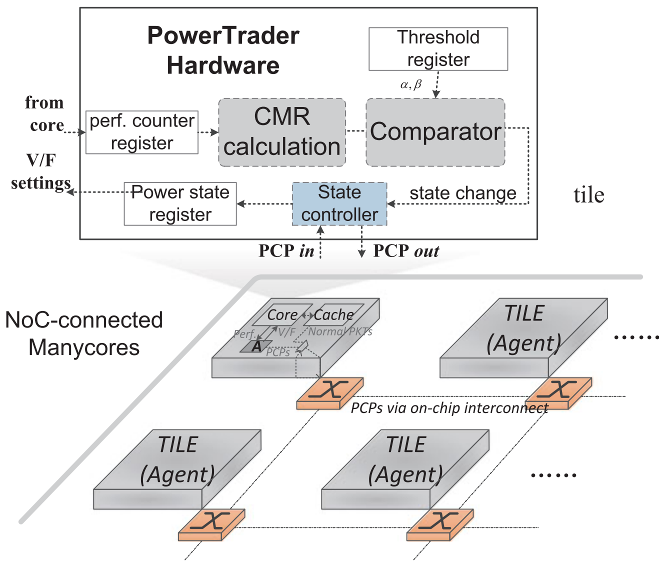


Fig. 7. Hardware required to implement PowerTrader.

intensity and make right decisions to control agent state transition.

4.2 Hardware Agent

PowerTrader also requires dedicated on-chip hardware in charge of power state transition and PCP interaction. However, the implementation is much simpler compared with centralized approaches. As shown in Fig. 7, several registers and three modules are just enough to achieve full functionalities of PowerTrader: “state controller” is responsible for the PCP interaction and the execution of Algorithm 1; core-level performance counters, stored in the counter register, is used to calculate *CMR*; the final state decision is made in the “comparator” based on the current *CMR* and the preset thresholds α and β, and state controller will be then mandated to change from “stable” to “release” or “absorb” upon receiving the decision from the comparator. The REQs will be broadcasted as soon as the agent state is shifted to “absorb”.

In order to estimate the area overhead of PowerTrader hardware, we use Synopsis Design Compiler [27] under SMIC90 technology library. The registers are all configured to 32 bits, capable of representing gigabytes magnitude of data. The synthesis reports only $0.0023\ \mathrm{mm}^2$ area overhead in total. To approximate the power consumption, we assume these registers similar to the pipeline registers in NoC routers, which only consumes 0.0015 W in total when fully utilized (0.86 percent of the whole NoC power consumption), and could be safely ignored since the tiny power consumption will not outweigh total power savings. These facts also render PowerTrader a light-weighted power management approach for on-chip implementation in manycore processors.

4.3 Tackling Thermal Alerts

On-chip thermal management is also a widely-used technique, designed to keep system below a safe operating temperature. PowerTrader is designed as a power management technique, but that does not mean it cripples serving the purpose of on-chip thermal management. Quite on the contrary, PowerTrader could be an effective addon to solve thermal hotspots in manycores. That is because, as a regional scheme, it intrinsically considers the heat-specific factors such as the spatial distribution of power across the chip, and is capable of transferring power from thermal hotspots to other cooler areas. For example, if one thermal hotspot is detected, hardware agent could compulsorily turns its power state immediately into “release” and let other agents fetch its excessive power quota. Such Power shifting can maintain the nearly uniform distribution of chip heat, and at the same time preserve time/performance constraints of critical applications. While in this article, we only focus on its efficacy in managing power.

5 Full System Simulation Setup

5.1 Platform

We use Graphite [28], a full system simulator for manycores as our basic simulation framework. We use a Tiled-like manycore architecture with 64 single issue, in-order cores. L1 I/D cache is privately occupied and last level cache is shared by all the cores. In particular, we augment the simulator with Algorithm 1 as well as the associated trading mechanisms *BPT* and *MPT*. We run multiprogrammed workloads selected from Parsec-3.0 [29] and Splash-2 [30] benchmark suite. We categorize 17 benchmarks based on their respective cache miss rate, obtained using prior profiling knowledge, classify them into `Computation intensive` and `Memory intensive`, and then create “bundles” that include a variety of mixed workloads. The multiprogram workload mixes are organized from *high* computation intensive to *high* memory intensive, so each bundle should exhibit distinct `CMR` behavior. Since we use 4 benchmarks as a bundle, we allocate 16 threads for each of them and consider one-to-one mapping, which means one core only executes one thread and the threads in total occupy all the cores in a 64-core processor.

5.2 Baselines

We use four baselines to prove the efficacy of PowerTrader: (1) the first one is a traditional manycore configuration with no power management involved, referred to as “No_PM”; the results are normalized to this baseline for comparison; (2) the second one is a state-of-the-art approach, referred to as “FreqPar” [3]. This technique is a *hierarchical* power management approach, which aims at allocating pre-set power budget to each core “level-by-level” based on the estimated power and performance. We implement the detailed frequency partitioning method and various prediction models associated with this baseline, and prove the benefit of our autonomous PowerTrader in both performance and power; (3) the third baseline is an ideal case, referred to as “PT-Ideal”, implemented within PowerTrader context, but the difference is that it does not involve PCP interaction between agents. An “absorb” agent is “assumed” to have already located “release” agents within its broadcast scope, so power trading could start up directly, excluding MTTS overhead. This baseline could be regarded as the theoretical upper bound of the benefits PowerTrader might achieve; (4) we employ and modify LinOpt [31] as the last baseline, referred to as “Global-opt.”, because it uses linear programming in an attempt to attain global optimum in a centralized manner, but differed from its raw technique, that is, using Simplex method to solve linear programming problem at a per-10 ms basis so as to cover polynomial time complexity, usually estimated at $O(n^4)$ level [6]. We ideally and intentionally eliminate this overhead to make it work at finer granularity that is on par with PowerTrader. But note that we do not take process variation issues that LinOpt initially aims to tackle into account, because our evaluation platform is not able to model precise variations in frequency and voltage. We herein only intend to explore how much design space PowerTrader leaves compared with the global optimum in theory.

5.3 Power Control Actuation

To finalize power control, we resort to existing dynamic voltage and frequency scaling (DVFS) infrastructure, already provided in our Graphite platform. We assume that each core can be independently voltage-and-frequency scaled (*per-core* DVFS) via *on-chip voltage regulators* (VR), which complies with many prior commercial products [23], [32] and research studies [3], [33]. Observed from our evaluation platform, total chip power consumption is around 11 watts when each core is configured at 1 GHz and no DVFS is applied. So we use the value as our chip-level power budget. We then select 4 frequency levels under 32 nm technology node: 0.6, 0.79, 1, 1.35 GHz by referring to McPAT with CACTI [34]. This configuration is also similar to many commercial many-core processors like Intel SCC [23] (125 MHz $\rightarrow$ 1.3 GHz) and Tile64 [35] (500 MHz $\rightarrow$ 866 MHz). Furthermore, we observed that the power quota needed to upgrade a V/F level is nearly equal at each DVFS operating point (slightly different for our evaluated benchmarks), so we can simply take this value as the unit power quota (1 unit) in PCP transmission. For example, if an agent intends to vary its frequency level from 0.6 to 1 GHz, the power that needs to be absorbed is hence 2 units. But also note that in today's technology, more frequency steps are also possible. Power impact of a frequency change differs per core (i.e., due to process variability), and power consumption is application-dependent due to extensive clock- and power-gating. PowerTrader embraces these fine grained frequency levels, we only make this conservative assumption in this article, uninfluential to demonstrate key functionalities of PowerTrader.

DVFS Actuation Overhead. Since we assume on-chip VR as the DVFS actuator for each core, DVFS actuation overhead, as reported in recent studies, could be decreased to tens of nanoseconds [36], so we take 250 ns as the DVFS actuation overhead in evaluating PowerTrader [3]; this value is also assumed in our "FreqPar" baseline for the purpose of fairness in comparison. Graphite does not model this overhead, so we manually added it after each "absorb" agent has obtained the power quota, but before the power state, represented by V/F level, is finally altered.

Control Interval Setting. Effective power management requires that DVFS actuation overhead, though unavoidable, is minimized as much as possible, possibly covered by appropriate setting of control interval. PowerTrader, however, is an autonomous power management approach, whose control interval or the duration between two

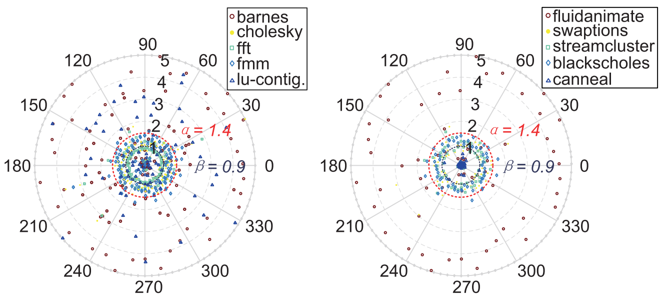


Fig. 8. CMR distributions for selected parsec-3.0 and splash2 benchmarks. The figure is plotted using "polar" scale because it could clearly demonstrate the distribution of CMRs. For example, values smaller than 2 are all enclosed by circle "2". The ticks on the outmost circle indicate CMR evaluation intervals. We plot each CMR value every 50 μs.

consecutive state transitions is never fixed, but is determined by CMR threshold settings: α and β. Therefore, we evaluate CMR distributions for our selected benchmarks, as shown in Fig. 8. We found that CMR exhibits obvious distributions ($0 \sim 5$) at a granularity of tens of microseconds, so it indicates that the DVFS actuation overhead (250 ns as assumed) only occupies less than 1 percent of the duration between two continuous state transitions. Besides, we found that most CMRs scatter around the innermost circles ($0.5 \sim 1.5$), so without losing generality, we choose $\alpha = 1.4$ (marked with red circle) and $\beta = 0.9$ (marked with blue circle) in our subsequent evaluations, just enough to cover the DVFS plus MTTS overhead, but note that the chosen thresholds are both empirical values; other settings are also possible, contingent to the actual DVFS actuator overhead and CMI variation granularity.

For our centralized baseline "FreqPar", control interval setting also needs to consider the overhead of power allocation algorithm, which has an $O(n)$ complexity as reported in the paper [3]. As expected, for our 64-core platform, it costs $64\ cores \times \frac{1{,}000\ cycles}{1\ \mathrm{GHz}} = 64\ \mu\mathrm{s}$, for the algorithm to accomplish power allocation for all 64 cores, if we assume 1 GHz operating frequency in the power manager. If we also require that the total overheads (DVFS+power allocation) can only occupy up to 1 percent of the total control interval (to be in line with PowerTrader), the control interval should be set to $(64\ \mu\mathrm{s} + 250\ \mathrm{ns})/1\% \approx 6.4\ \mathrm{ms}$ as reported in literature [3]. While for our idealized baseline "Global-opt.", we do not set such large control interval; instead, we want the control interval to be comparable with PowerTrader as a way to evaluate its headroom below global optimum, so we set 1 μs interval (average MTTS +DVFS overhead in PowerTrader) to invoke this scheme each time. In realistic operating scenario, it is impossible to implement power control in such tiny control interval, because linear programming in LinOpt occupies nearly as large as 13.3 percent runtime of total control interval for 64-core platform as reported in literature [6] due to its high-order polynomial complexity, so its normal control interval is usually set larger than 10 ms [31]. That is also why we call it *idealized* baseline within our evaluation environment. Besides, control interval setting for these two baselines do not include the statistics transmission overhead that we are not able to estimate in our platform (though it actually exists); however, these overheads are not necessary at all in PowerTrader.

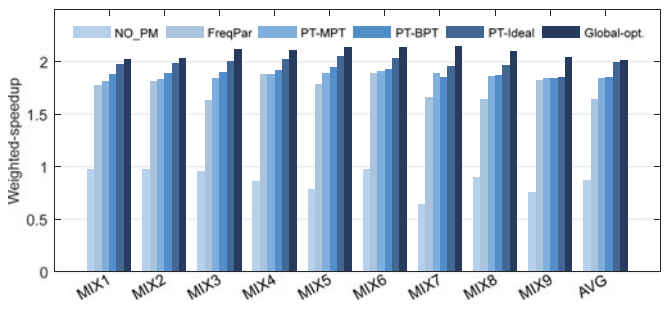


Fig. 9. Instruction throughput evaluation, represented by weighted-speedup metric.

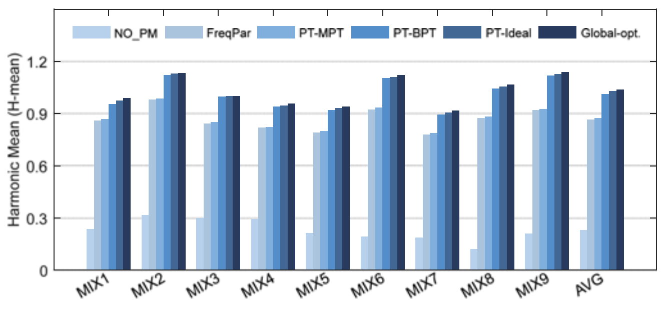


Fig. 10. Multiprogram fairness, represented by Harmonic Mean (H-mean).

6 Results and Analysis

6.1 Application Performance

In this set of experiment, we evaluate PowerTrader in terms of the performance of all workload mixes, under fixed power budget. We use the metric "Weight-speedup (WS)" to evaluate instruction throughput [4], [37], [38], [39], as representative of the performance of various mixes. The weighted-speedup for n applications is defined as: $Weighted\_speedup = \sum_{i=1}^{n} \frac{IPS^i_{scheme}}{IPS^i_{alone}})$, where IPS^i_{scheme} is the "Instruction per Second" achieved by the employed scheme (i.e., PT-BPT or PT-MPT) for application i, and IPS^i_{alone} is the measured IPS when the application i is executed alone without any power management schemes. We use this metric instead of IPS alone because researchers in recent literatures claim that weighted speedup more accurately captures the multi-program throughput performance by equalizing the contribution of each program in the workload mix through normalizing its multiprogram IPS with its single-program IPS [38], [40], [41].

Discussion. As shown in Fig. 9, two items of PowerTrader show higher weighted-speedup than FreqPar by 12.1 and 12.2 percent apiece (Note that weighted speedup are all raw values). The benefit stems from the finer-grained power adaptation of PowerTrader that could timely respond the application runtime demand, rather than waiting for the arrival of each control interval. The "FreqPar", on the other hand, must go through the statistics collection, power quota allocation to finalize power state adaptation. This tedious centralized style makes it fail to capture application phase changes in time, and hence undermines power management efficiency.

The figure also demonstrates that the performance of BPT is on par with that of MPT: for some mixes like `MIX6`, `MIX7`, `MIX8` and `MIX9`, MPT exhibits close or even higher WS than BPT; on the contrary, some mixes like `MIX1` and `MIX5` etc., exhibit a higher WS for BPT than MPT. This behavior abides by the previous analysis of two trading mechanisms in Section 3, that is, BPT and MPT have their respective pros and cons: some mixes benefit from faster trading establishment, while others may prefer less PCP traffic and short packet latency thereof. The difference of average WS improvement is hardly recognizable for BPT and MPT as shown in the figure, so the two trading mechanisms could be dynamically selected according to workload real-time necessities.

Comparison with Theoretical and Global Optimum. PowerTrader is still suboptimal, 7.5 percent lower WS for BPT, compared with its theoretical upper bound ("PT-Ideal") because of its PCP interaction overhead (MTTS), and in the following sections, we will show how the tradeoff between PCP broadcast scope and MTTS affects workload power and performance efficiency, by thoroughly investigating MTTS curve dynamically changing with broadcast scope. Furthermore, two instances of PowerTrader also demonstrate 13.2 percent lower WS compared with our idealized centralized approach achieving global optimum. This set of results instruct that PowerTrader, although to some extent improves scalability and performance, still lacks of sufficient superiority to attain global optimality. As a localized operation, limitation of regional PCP interactions will not cover power requests imposed by all agents and handle them as a whole. However in realistic power management, no centralized approach is able to attain such optimum either, if taken the costly allocation overhead into consideration, PowerTrader could provide both satisfied performance and scalability, at the same time.

6.2 Fairness

In addition to the weighted-speedup metric, as a direct measure of application performance, we use another metric "Harmonic Mean (H-mean)" to further evaluate the balance between fairness and instruction throughput for various workload mixes [38]. The definition of H-mean is as follows: $H\_mean = \frac{n}{\sum_{i=1}^{n} \frac{IPS^i_{alone}}{IPS^i_{scheme}})}$. From Fig. 10, we observe that: (1) BPT exhibits obvious improvement compared with FreqPar (16.8 percent more); (2) MPT is hardly fairer than FreqPar (0.8654 0.8749 respectively).

Discussion. This is due to two reasons: (1) FreqPar always assigns highest priority to the critical thread of an application during power quota allocation, so the thread dictates more power each time while others always stay starving. By sharp contrast, PowerTrader is intrinsically a fair mechanism regarding each "absorb" agent equally, does not bias any application or internal thread. Each agent has fair opportunity to procure their share of power quota under the same power budget. (2) the long power control interval also harms fairness. As mentioned above, FreqPar works at a control interval of 6.4 ms [3]. The criticality of a thread varies enormously upon such coarse granularity, rendering FreqPar failed to accommodate all possible critical threads in the control interval. PowerTrader fairly tends to satisfy each "absorb" agent that may exist at each possible spot; hence for PT-BPT, "absorb" agents are given timely IPS speedup so that it yields higher H-mean values [38]. Nonetheless for MPT, not each agent could be fully fed with power quota it requested; thus, one or more workloads will incur lower and

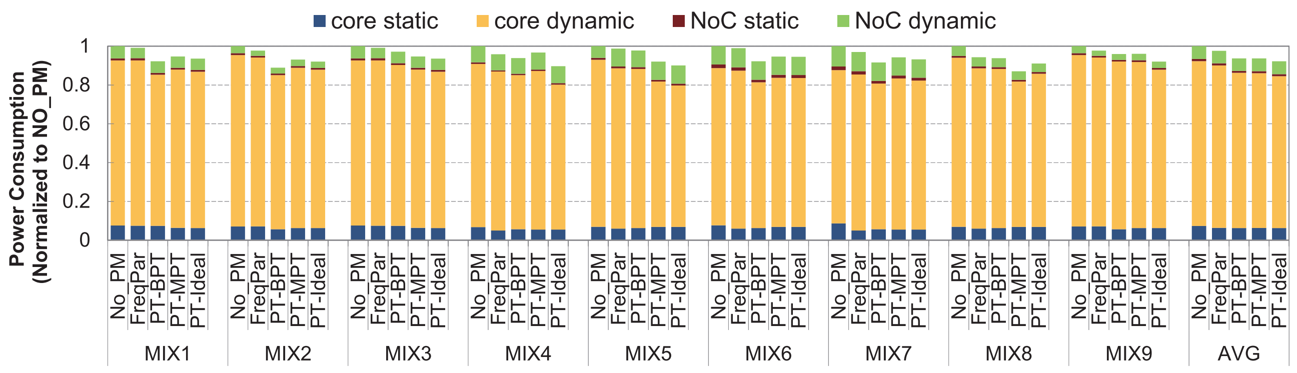
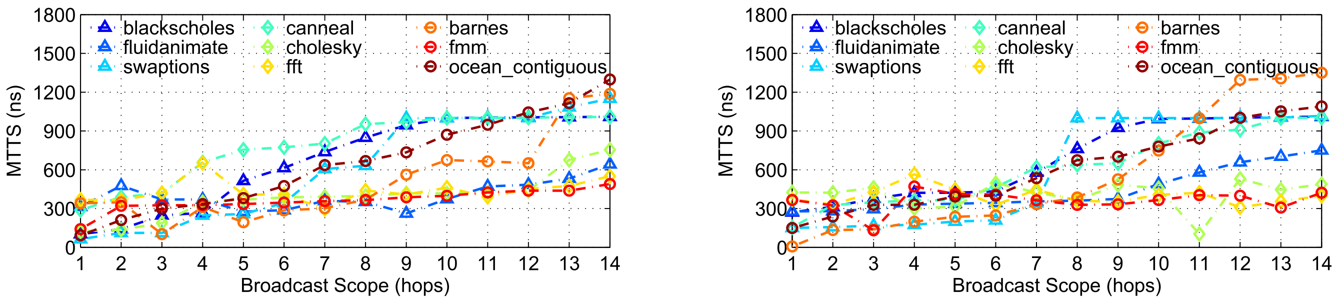


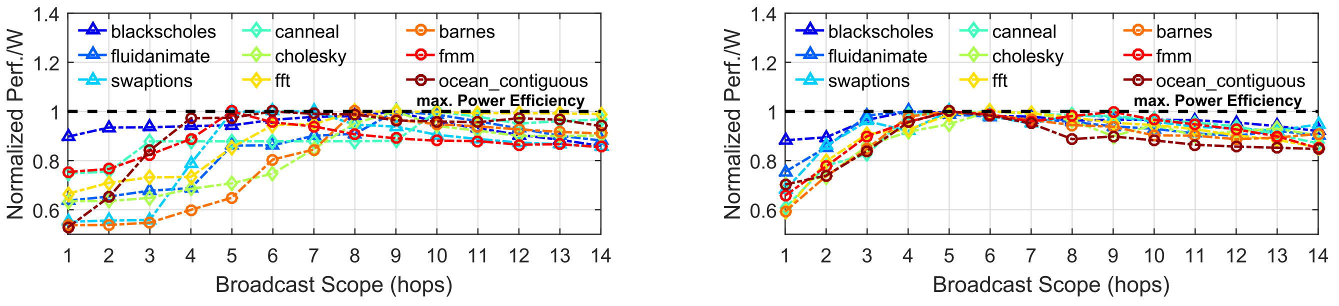
Fig. 11. Total power consumption, decomposed into CORE and NoC dynamic/static portions.



(a) MTTS variation with broadcast scope (**BPT**)

(b) MTTS variation with broadcast scope (**MPT**)

Fig. 12. Tradeoff between "broadcast scope" and "MTTS": Although larger scope enforces larger opportunities to attain stable, MTTS is also enlarged that is harmful to the PowerTrader responsiveness.



(a) Power efficiency varied with broadcast scope for **BPT**

(b) Power efficiency varied with broadcast scope for **MPT**

Fig. 13. The tradeoff impact to application power efficiency: Setting the scope too large or too small is both harmful to application power efficiency.

slower IPS speedup that leads to lower H-mean values, but is also on par with our centralized baseline.

6.3 Overall Power Consumption

In order to explore the power adaptation of PowerTrader, we evaluate power consumption and power/performance efficiency in this set of experiment. First, we show the overall power consumption of our employed application mixes in Fig. 11. We decompose the overall power into several portions as core/NoC dynamic/static power, because we use NoC as the backbone to carry the PCP traffic whose power consumption also needs to be considered. As can be seen, for workload MIX6 and MIX7, NoC power is slightly increased due to the overlayed PCP traffic but the core dynamic power reduction still outweighs the increment of NoC power. In total, PowerTrader could reduce overall power consumption by 6.2 and 6.3 percent compared with FreqPar, for BPT and MPT, respectively.

Discussion. Our baseline FreqPar, only capable of capping peak power, could not work at low-power mode, so even if the running workload mix does not need to much power, it still has to allocate all of the power quota to each core. Whereas for BPT and MPT, two modes could be both detected and dynamically switched, and that's also why PowerTrader could reduce power consumption while FreqPar cannot.

6.4 Tradeoff between Broadcast Scope and MTTS

Although it may happen that as far as 14-hop scope is required in the worst case for an "absorb" agent to finally couple with a "release" agent, it is not necessary to design PowerTrader with 14-hop scope. That is because a broadened scope also enlarges MTTS, which undermines the power control responsiveness. As shown in Figs. 12a and 12b, we evaluate MTTS at different scope configurations. MTTS climbs almost linearly with scope scaled from 0 to 14. MPT is relatively less sensitive that MTTS starts to increase obviously at scope 7 for most of the benchmarks. Intuitively speaking, such increment of MTTS inevitably harms the responsiveness of PowerTrader and further impacts the power efficiency. Fig. 13 proves this notion. It shows the normalized

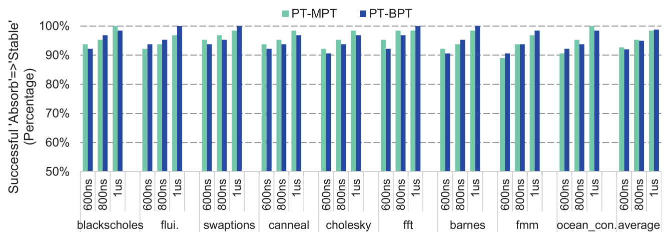


Fig. 14. Convergence and stability analysis. We tested three timer configurations: 600 ns, 800 ns, and 1 us.

application power efficiency (performance per watt) under different scope configurations. All of the applications enjoy the boost in performance with the scope enlarged initially, due to increasing power trading opportunities. However, the efficiency drops significantly when the scope goes beyond some point; for example, $8 \rightarrow 10$ hops for BPT and $5 \rightarrow 7$ hops for MPT. This "hump" behavior is because the large round-trip latency (MTTS) may lead to a hysteresis effect, which means the application behavior may have already changed before stable state is attained. Generally, we find 6-hop is a balanced choice and that is also why we use this value in the previous performance evaluation, but the actual decision could be made at design time considering different many-core configurations and running workloads.

6.5 Convergence Analysis

Intuitively speaking, not all the agents in certain power state could be served in its broadcast scope, which dominates overall stability of power management in many-core chip, because the imbalance of "absorb" and "release" agents will lead to system deadlock, reflected by the infinite waiting to be served for "absorb" agents, and waiting to serve for "release" agents. To explore this scenario, we evaluate the successful rate of power trading for each benchmark, that is, the percentage of agents successfully turned from "absorb" to "stable" state versus overall number of "absorb" agents that emerged during execution. In the experiment, we set a timer ranged from $600\ \mathrm{ns}$ to $1\ \mu\mathrm{s}$, which means if an "absorb" agent still cannot find a matched "release" agent to serve itself when the timer expires, we manually transform it back to stable state. We stat the MTTS values of all "absorb" agents and found, in Fig. 14, that the successful rate could reach more than 92.7 percent within $600\ \mathrm{ns}$ timer setting. In other words, only 7 percent "absorb" agents cannot find matched "release" agents. If we go a step farther by setting the timer to $1\ \mu\mathrm{s}$, more than 98 percent agents could be served, which means the many-core system always maintains stable and barely incurs instability or divergence, but

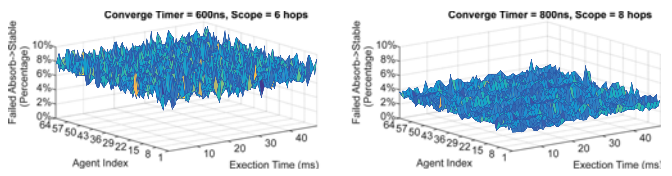


Fig. 15. Thread balance analysis (using workload MIX5 in MPT): We use surfplot to observe dynamic variations of "absorb to stable failures" under different converge timer+scope combinations. Each x-tick is 1 ms time interval, and the percentage is calculated as "timeout absorb requests" over all "absorb to stable requests". We use the data collected within 50 ms right after system warm-up.

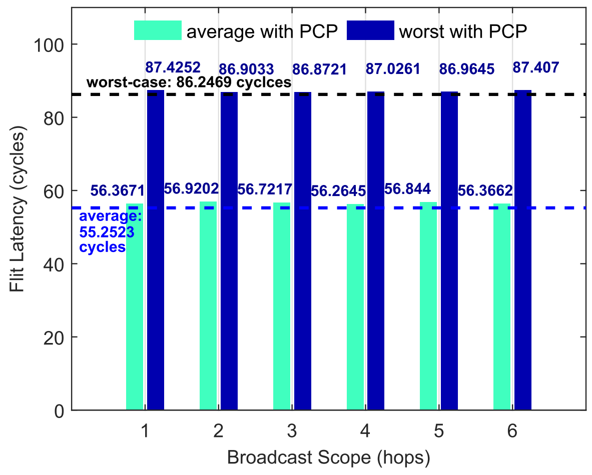


Fig. 16. Worst-case/average flit latency of normal data communication, with/without PCPs overlaid.

even if several agents really starve and keep waiting, we could turn them back to stable state when timer expires, in order to let them proceed subsequent power state adaptations, so as to avoid system instability and make it converge.

Thread Balance Analysis. Additionally, we investigate thread balance under the same context after system warm-up. Thread imbalance happens when certain threads or programs are always got accelerated or starved in power management. To survey this problem on the fly, we observe runtime "absorb to stable" failures as shown in Fig. 15. We use surfplot and the spikes and ebbs in the waveform could conveniently demonstrate if starvation or power monopolization happens. As can be seen, the percentage at each time interval for 64 agents in the mix mildly fluctuates at around 8 to 10 percent for $600\ \mathrm{ns}$ surfplot, while below 6 percent for $800\ \mathrm{ns}$ case. Such failures are nearly uniformly distributed along with executing in both cases, so this result confirms that, in accordance with above conclusion, *threads in our multiprogram mixes are stable and balanced indeed.*

6.6 Overhead Analysis

Communication Overhead. Since PowerTrader uses NoC as the backbone to carry PCPs, the "extra traffic" which never shows up in prior schemes may burden the applications traffic condition. The impact could be reflected by the variation of average and worst-case flit latency of normal data communication, as shown in Fig. 16. The result shows that normal flit latency only increases no more than 2 percent when PCPs overlaid in both the average and worst case. This is because PCP is very light-weighted with limited payload which, compared to normal data packets, is negligible in volume. Also, we found the worst-case flit latency is insensitive to the broadcast scope of PCPs. This experiment proves that PCPs won't degrade the latency of normal data communication, and *hence should not be viewed as a key overhead of PowerTrader*.

Mechanism Overhead. PowerTrader is an autonomous power management scheme, so we compare its mechanism overhead to the centralized algorithm with $O(n)$ complexity (as claimed in our baseline "FreqPar" [3]). We trace the execution time of the $O(n)$ algorithm and PowerTrader under different scale of manycores, as shown in Fig. 17. We found

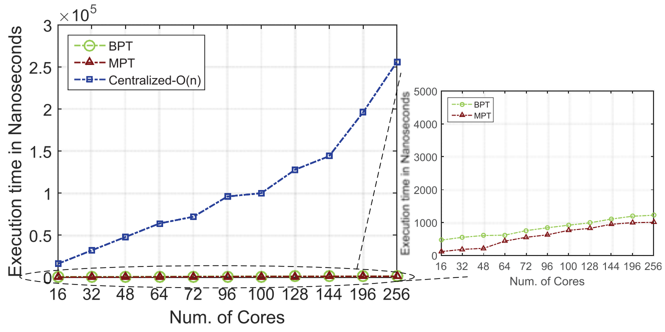


Fig. 17. Scalability evaluation of PowerTrader and a centralized power allocation algorithm with $O(n)$ complexity.

that $O(n)$ algorithm starts to increase dramatically with many-core scaling. By sharp contrast, as regional operations restrained by the scope, the overhead of BPT and MPT both increase mildly, insensitive to the scale of manycores. The maximum execution time is only $1{,}221$ ns for BPT. Note that we do not adopt the statistics collection and transmission overhead inherited in the centralized $O(n)$ baseline, but only account for the overhead of power allocation algorithm, so the *real-world* execution time in Fig. 17 may grow even steeper for the $O(n)$ approach. Therefore, for future ultra large-scale manycore architectures, PowerTrader is undoubtedly applicable which is even beyond reach for existing centralized approaches.

7 Conclusion

This article proposes PowerTrader, an autonomous power management approach targeting manycores. It abandons classic centralized approaches; instead, power could be freely traded between modeled autonomous agents, without violating chip power budget. To govern the trading process, two power trading mechanisms (BPT and MPT) are proposed with different behaviors on power control packet (PCP) interaction. We also analyze the key design tradeoff, that is, the broadcast scope and MTTS, and its impact on application performance and power under BPT and MPT. Without a global centralized manager, PowerTrader exhibits superior scalability and improves application power efficiency substantially compared with state-of-the-art baseline. We thus believe that PowerTrader is a promising power management technique to be deployed in future large-scale manycore processors.

Acknowledgments

This work is supported in part by the National Nature Science Foundation of China (NSFC) under grant No. (61602442, 61572470, 61532017, 61522406, 61432017, 61376043), in part by the Open Project Program of the State Key Laboratory of Mathematical Engineering and Advanced Computing under grant No. 2016A08, in part by the Youth Innovation Promotion Association of CAS under grant No.Y404441000 and in part by the State Key Laboratory of Computer Architecture with the Institute of Computing Technology (ICT) under grant No. CARCH2603.

References

[1] IBM, "Power8 processor user's manual for the single-chip module," version 1.3, Mar. 16, 2016.
[2] S. Damaraju, et al., "A 22nm IA multi-CPU and GPU system-on-chip," in *Proc. IEEE Int. Solid-State Circuits Conf. Dig. Tech. Papers*, 2012, pp. 56–57.
[3] M. Kai, L. Xue, C. Ming, and W. Xiaorui, "Scalable power control for many-core architectures running multi-threaded applications," in *Proc. 38th Annu. Int. Symp. Comput. Archit.*, 2011, pp. 449–460.
[4] A. Sharifi, A. K. Mishra, S. Srikantaiah, M. Kandemir, and C. R. Das, "PEPON: Performance-aware hierarchical power budgeting for NoC based multicores," in *Proc. 21st Int. Conf. Parallel Archit. Compilation Techn.*, 2012, pp. 65–74.
[5] Q. Deng, D. Meisner, A. Bhattacharjee, T. Wenisch, and R. Bianchini, "CoScale: Coordinating CPU and memory system DVFS in server systems," in *Proc. 45th Annu. IEEE/ACM Int. Symp. Microarchitecture*, Dec. 2012, pp. 143–154.
[6] J. A. Winter, D. H. Albonesi, and C. A. Shoemaker, "Scalable thread scheduling and global power management for heterogeneous many-core architectures," in *Proc. 19th Int. Conf. Parallel Archit. Compilation Techn.*, 2010, pp. 29–40.
[7] M. Shafique and J. Henkel, "Agent-based distributed power management for kilo-core processors: Special session: Keeping kilo-core chips cool: New directions and emerging solutions," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, 2013, pp. 153–160.
[8] Tilera, "Tile-Gx series cloud processor," 2013. [Online]. Available: http://www.tilera.com/products/processors/tile-gx_family
[9] Ambric, "Ambric Am2045 many-core processor," 2008. [Online]. Available: https://en.wikipedia.org/wiki/ambric
[10] Adapteva, "Adapteva epiphany-iv series," [2014]. [Online]. Available: http://www.tomshardware.com/news/apateva-epiphany-4096-core-multicore-threaded,15064.html
[11] C. Isci, A. Buyuktosunoglu, C.-Y. Chen, P. Bose, and M. Martonosi, "An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget," in *Proc. 39th Annu. IEEE/ACM Int. Symp. Microarchitecture*, Dec. 2006, pp. 347–358.
[12] E. Rotem, A. Mendelson, R. Ginosar, and U. Weiser, "Multiple clock and voltage domains for chip multi processors," in *Proc. 42nd Annu. IEEE/ACM Int. Symp. Microarchitecture*, 2009, pp. 459–468.
[13] M. Shafique, A. Ivanov, B. Vogel, and J. Henkel, "Scalable power management for on-chip systems with malleable applications," *IEEE Trans. Comput.*, vol. 65, no. 11, pp. 3398–3412, Nov. 2016.
[14] J. Sartori and R. Kumar, "Three scalable approaches to improving many-core throughput for a given peak power budget," in *Proc. Int. Conf. High Perform. Comput.*, 2009, pp. 89–98.
[15] X. Wang, et al., "Adaptive power allocation for many-core systems inspired from multiagent auction model," in *Proc. Des. Autom. Test Eur. Conf. Exhib.*, 2014, pp. 1–4.
[16] M. Ismail, O. Hasan, T. Ebi, M. Shafique, and J. Henkel, "Formal verification of distributed dynamic thermal management," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, 2013, pp. 248–255.
[17] S. Iqtedar, O. Hasan, M. Shafique, and J. Henkel, "Formal probabilistic analysis of distributed dynamic thermal management," in *Proc. Des. Autom. Test Eur. Conf. Exhib.*, 2015, pp. 1221–1224.
[18] T. Ebi, M. Faruque, and J. Henkel, "TAPE: Thermal-aware agent-based power econom multi/many-core architectures," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.-Dig. Tech. Papers*, 2009, pp. 302–309.
[19] M. Shafique and S. Garg, "Computing in the dark silicon era: Current trends and research challenges," *IEEE Des. Test*, vol. 34, no. 2, pp. 8–23, Apr. 2017.
[20] H. Khdr, et al., "Power density-aware resource management for heterogeneous tiled multicores," *IEEE Trans. Comput.*, vol. 66, no. 3, pp. 488–501, Mar. 2017.
[21] G. Kornaros and D. Pnevmatikatos, "Dynamic power and thermal management of NoC-based heterogeneous MPSoCs," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 7, no. 1, pp. 1:1–1:26, Feb. 2014.
[22] A. Pathania, H. Khdr, M. Shafique, T. Mitra, and J. Henkel, "Scalable probabilistic power budgeting for many-cores," in *Proc. IEEE/ACM 20th Des. Autom. Test Eur. Conf.*, 2017, pp. 1–6.
[23] Intel, "Intel single-chip cloud computer," 2012. [Online]. Available: http://www.intel.com/content/www/us/en/research/intel-labs-single-chip-cloud-computer.html
[24] R. Kotla, A. Devgan, S. Ghiasi, T. Keller, and F. Rawson, "Characterizing the impact of different memory-intensity levels," in *Proc. IEEE Int. Workshop Workload Characterization*, 2004, pp. 3–10.

[25] N. Ioannou, M. Kauschke, M. Gries, and M. Cintra, "Phase-based application-driven hierarchical power management on the single-chip cloud computer," in *Proc. Int. Conf. Parallel Archit. Compilation Techn.*, 2011, pp. 131–142.
[26] A. Bhattacharjee and M. Martonosi, "Thread criticality predictors for dynamic performance, power, and resource management in chip multiprocessors," in *Proc. 36th Annu. Int. Symp. Comput. Archit.*, 2009, pp. 290–301.
[27] Synopsis, "Design compiler, version d-2010.03-sp2," 2010.
[28] J. Miller, et al., "Graphite: A distributed parallel simulator for multicores," in *Proc. IEEE 16th Int. Symp. High Perform. Comput. Archit.*, 2010, pp. 1–12.
[29] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The parsec benchmark suite: Characterization and architectural implications," in *Proc. 17th Int. Conf. Parallel Archit. Compilation Techn.*, 2008, pp. 72–81.
[30] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The splash-2 programs: Characterization and methodological considerations," in *Proc. 22nd Annu. Int. Symp. Comput. Archit.*, 1995, pp. 24–36.
[31] R. Teodorescu and J. Torrellas, "Variation-aware application scheduling and power management for chip multiprocessors," in *Proc. Int. Symp. Comput. Archit.*, 2008, pp. 363–374.
[32] AMD, "AMD opteron processor family," 2013. [Online]. Available: http://www.amd.com/en-us/products/server/opteron
[33] D.-C. Juan and D. Marculescu, "Power-aware performance increase via core/uncore reinforcement control for chip-multiprocessors," in *Proc. ACM/IEEE Int. Symp. Low Power Electron. Des.*, 2012, pp. 97–102.
[34] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *Proc. 42nd Annu. IEEE/ACM Int. Symp. Microarchitecture*, 2009, pp. 469–480.
[35] S. Bell, et al., "TILE64-processor: A 64-core SoC with mesh interconnect," in *Proc. IEEE Int. Solid-State Circuits Conf.*, 2008, pp. 88–598.
[36] W. Kim, M. Gupta, G.-Y. Wei, and D. Brooks, "System level analysis of fast, per-core DVFS using on-chip switching regulators," in *Proc. IEEE 14th Int. Symp. High Perform. Comput. Archit.*, 2008, pp. 123–134.
[37] K. Luo, J. Gummaraju, and M. Franklin, "Balancing thoughput and fairness in SMT processors," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw.*, 2001, pp. 164–171.
[38] S. Eyerman and L. Eeckhout, "System-level performance metrics for multiprogram workloads," *IEEE Micro*, vol. 28, no. 3, pp. 42–53, May/Jun. 2008.
[39] S. Eyerman, P. Michaud, and W. Rogiest, "Multiprogram throughput metrics: A systematic approach," *ACM Trans. Archit. Code Optimization*, vol. 11, no. 3, pp. 34:1–34:26, 2014.
[40] A. Snavely and D. M. Tullsen, "Symbiotic jobscheduling for a simultaneous multithreaded processor," in *Proc. 9th Int. Conf. Archit. Support Program. Languages Operating Syst.*, 2000, pp. 234–244.
[41] A. R. Alameldeen and D. A. Wood, "IPC considered harmful for multiprocessor workloads," *IEEE Micro*, vol. 26, no. 4, pp. 8–17, Jul./Aug. 2006.

Hang Lu received the PhD degree from the University of Chinese Academy of Sciences (UCAS), in 2015. He is currently an assistant professor in the State Key Lab. of Computer Architecture, Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS). His research interests include high performance Networks-on-Chip (NoC), power efficient manycore architectures, scale-out processors, etc.

Guihai Yan (M'11) received the BSc degree in electronics and software engineering (dual-degree) from Peking University, Beijing, China, in 2005 and the PhD degree in computer science from the Institute of Computing Technology (ICT), Chinese Academy of Sciences, Beijing, China, in 2011. He is currently an associate professor in the Institute of Computing Technology, Chinese Academy of Sciences. His research interests include computer architecture, domain-specific microsystems, and energy-efficient computing. He is a member of the IEEE.

Yinhe Han (M'06) received the BEng degree from the Nanjing University of Aeronautics and Astronautics, China, in 2001, and the MEng and PhD degrees in computer science from the Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS), China, in 2003 and 2006, respectively. He is currently a professor in the State Key Lab. of Computer Architecture, ICT, CAS. His research interests include computer architecture, especially on fault-tolerant and low power architecture and VLSI design and test. He is a member of the IEEE.

Xiaowei Li received the BEng and MEng degrees in computer science from the Hefei University of Technology (China), in 1985 and 1988, respectively, and the PhD degree in computer science from the Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS), in 1991. Currently, he is a professor and deputy director of the Key Laboratory of Computer System and Architecture, ICT, CAS. His research interests include VLSI testing and design verification, dependable computing, and wireless sensor networks. He is an associate editor-in-chief of the *Journal of Computer Science and Technology*, and a member of the editorial board of the *Journal of Electronic Testing*, and the *Journal of Low Power Electronics*. In addition, he serves on the Technical Program Committees of multiple IEEE and ACM conferences, including VTS, DATE, ASP-DAC, PRDC, etc. He was also the program co-chair of the IEEE Asian Test Symposium (ATS) in 2003, and general co-chair of ATS'07. He is a senior member of IEEE.