Agile Frequency Scaling for Adaptive Power Allocation in Many-core Systems Powered by Renewable Energy Sources

Xiaohang Wang*, Zhiming Li*, Mei Yang†, Yingtao Jiang†, Masoud Daneshtalab‡, Terrence Mak§*
*Guangzhou Institute of Advanced Technology, CAS, China
Email: {xh.wang, zm.li}@giat.ac.cn
†University of Nevada, Las Vegas, USA
Email: mei.yang@unlv.edu, yingtao@egr.unlv.edu
‡University of Turku, Finland
Email: masdan@utu.fi
§The Chinese University of Hong Kong, China
Email: stmak@cse.cuhk.edu.hk

Abstract— As low-power electronics and miniaturization conspire to populate the world with emerging devices, one appealing approach is to power these multi-core/many-core-based devices with energy harvested from various environments. Of the most important issues concerning these devices is how to effectively allocate power budget among the cores competing for power, which is formulated as one specific type of power-performance optimization problem in this paper. We attempt to solve this problem by proposing an Adaptive Power Allocation Technique (APAT) that explores a dynamic programming network. Our goal here is to maximize the overall system performance, taking into account a unique yet challenging fact that, available power budget might have to undergo a significant change when a renewable energy source is scavenging. APAT has a linear time complexity and low hardware overhead. Experiments have confirmed that APAT can reduce $20 \sim 30\%$ of execution time compared to other state-of-the-art power allocation algorithms. In addition, as APAT is quite insensitive to the changing rate of the power, lending itself well for power management in many-core systems powered by energy-harvesting sources.

I. INTRODUCTION

There is a general consensus that by the year 2020, the chip power density will increase as much as $10\times$ over the year 2012 [1]. A lot of these power hungry chips then will have to be powered by harvesting renewable energy. Renewable energy like solar energy is abundant to power electronic systems from low-power devices [2] to data centers [3]. One of the technical challenges to improve the energy efficiency of an energy harvesting based multi-/many-core chip is power budget allocation, which is to maximize the chip performance under a limited power or energy budget, *i.e.*, the power budgeting problem [4]. Frequency scaling can be used to solve the problem. However, this power budget problem is complicated by the following issues in an energy harvesting based many-core system.

1) In a many-core chip, the operating frequencies of the many resources are likely tunable. Consider an application mapped to a 16-core region, where the frequency of each core can take one of the four allowed values. The total frequency combination is as large as 4^{16}, which is about 4×10^9. Heuristic-based approaches [5], [6] cannot find optimal solutions given such a large solution space.
2) The input power budget from the renewable energy might have high and rapid variations due to uncertainty in the environmental situations. Existing approaches [5], [6] need long run time, leading to unknown behavior or poor performance. Using energy buffers like supercapacitors or batteries can suppress the variation, which however, incurs significant energy loss and suffers from issues like battery aging, self-discharge, etc [7]. Thus, it would be beneficial to minimize the usage of energy buffers and use the renewable energy directly as much as possible.

In this paper, a novel frequency scaling scheme for power allocation is proposed, Adaptive Power Allocation Technique (APAT) using dynamic programming with renewable energy awareness to address the above aforementioned problems. The power budgeting problem is solved using dynamic programming network with linear time complexity. APAT has the following key features:

1) APAT can generate globally optimal power allocation solution under a given power budget from the harvested energy.
2) The run-time of APAT is much lower (*e.g.*, a few to dozens of cycles) than that of any other known power allocation algorithms [5], [6].
3) APAT can be used for many-core systems to control the power consumption of various on-chip resources at a finer grain.

Experimental results reveal that APAT can reduce the expectation time significantly compared with other three competing power allocation methods [5], [6] with much lower hardware and run-time overhead.

The paper is organized as follows. Section II reviews the related work. Section III introduces the performance-power model and problem definition, followed by the description of the dynamic programming based power allocation scheme in Section IV. Section V provides the experimental results and analysis of the proposed approach. Finally, Section VI concludes the paper.

II. RELATED WORK

Renewable energy could be used to power electronic systems, ranging from datacenters [8] to embedded system [9].

This research program is supported by the Natural Science Foundation of China No. 61376024 and 61306024, Natural Science Foundation of Guangdong Province No. S2013040014366.

One of the challenge appears as how to allocate power to resources since the renewable energy source is not stable and might have high variation. Power management approaches for energy harvesting based multi-/many-core can be achieved by task scheduling [9], frequency/voltage scaling [8], [10], etc. Similar approaches can be found to solve the power budgeting problem [4] in many-core systems. Frequency/voltage scaling can be performed for power allocation [4]–[6]. For example, [6] proposed a method where the number of cores that can be powered on is coarsely determined, after which fine-grained frequency adjustment is followed. Most of these approaches are heuristic-based [4]–[6], or linear programming-based [11]. Heuristic-based approaches cannot find the optimal solutions when the number of resources to be controlled is increased or when the applications' behavior becomes complicated. Linear programming-based approaches might lead to high run-time and power consumption overhead to find a good solution, which might waste the harvested energy.

III. Models and Problem Formulation

A. Performance model

Frequency scaling is used to balance between power consumption and performance. Suppose each application q occupies an N_q-tile region where the frequencies of the tiles are $f_1, \ldots, f_{Nq}$. Note that some of the regions can be overlapped, *e.g.*, a cache bank might be shared by several applications. An application is assumed to be mapped to only one region. For a total of Q applications running simultaneously, we have $\sum_{q=1}^{Q} N_q = N$, where N is the total number of tiles. Performance of each application measured in execution cycles is modeled in terms of the frequencies of its region/tiles, as follows,

$$Cycle = g_{cycle}\left(f_1, \ldots, f_{Nq}\right) \tag{1}$$

Regression models [12] are used to find the g_{cycle} in Eqn. 1 through curve fitting. We have proposed the following model in Eqn. 2 experimentally, which can result in lower regression error, and this model will be validated in the experimental part in Section V.B.

$$\ln Cycle = \sum_{i=1}^{N_q} a_i \cdot \sqrt{f_i} \tag{2}$$

where a_i is the regression coefficient w.r.t. f_i. At the k-th time interval, the frequencies of an application's region are set randomly with a vector $\overrightarrow{f_k} =< f_1, ..., f_N >$ and measure the cycles $Cycle_k$. With K time intervals, the training data are collected. A linear regression model with the maximum likelihood estimator [12] will find the coefficients (a_i's) with the training data set. When some regions are overlapped, there are some resources shared by multiple applications, which implies resource contention. Thus, the performance models are trained for each application in parallel with possible contentions. These models remain accurate after the training.

B. Dynamic power model

Assuming all the cores are operating at the same voltage level, the dynamic power of an application with N_q tiles can be determined as follows:

$$P_q = \sum_{i=1}^{N_q} \alpha_i \cdot C_i \cdot f_i \cdot V^2 = \sum_{i=1}^{N_q} b_i \cdot f_i \tag{3}$$

where α_i is the switching activity, C_i is the effective capacitance, V is the voltage, $b_i \equiv \alpha_i \cdot C_i \cdot V^2$.

If, besides the core frequencies, the core voltages can also be adjusted, the dynamic power can be calculated as

$$P_q = \sum_{i=1}^{N_q} \alpha_i \cdot C_i \cdot {f_i}^3 / K^2 = \sum_{i=1}^{N_q} d_i {f_i}^3 \tag{4}$$

where K is a constant. Similarly, $d_i \equiv \alpha_i \cdot C_i / K^2$.

C. Problem Definition

With the above models, the power budgeting problem aims to minimize the overall execution time under the input power budget. With Q applications each occupying an N_q-tile region, we have

$$\sum_{q=1}^{Q} w_q \cdot P_q = P \tag{5}$$

where each P_q is the power budget for application q at a given time t, and w_q are user defined priority weights for each application.

The power budgeting problem for each application q can be formulated as,

$$\min Cycle = g_{cycle}\left(f_1, \ldots, f_{N_q}\right) \tag{6}$$

subject to

$$\sum_{i=1}^{N_q} b_i \cdot f_i \leq P_q \tag{7}$$

for each

$$f_i \in \{F_1, \ldots, F_M\} \tag{8}$$

The power P_q in Eqn. 7 is determined by applying either Eqn. 3 or 4. Eqn. 8 specifies a discrete set of frequency values that each frequency variable f_i can take.

Dropping the ln notation in Eqn. 2 and converting the min operator to max in the objective equation, the above problem definition has the same form as a bounded knapsack problem, which is NP-hard. On the other hand, the power budget might have rapid variations, which requires the problem be solved with low run time and hardware overhead.

IV. The APAT Algorithm

A. Overview of the APAT algorithm

Inspired by the dynamic programming approach to solve the knapsack problem, which is pseudo-polynomial time complexity, the problem in Section II.C can also be solved optimally by the following dynamic programming approach with polynomial time. Since the logarithmic function in the objective (Eqn. 2) is monotonic, minimizing Eqn. 2 is equivalent to minimizing $\sum_{i=1}^{N_q} a_i \cdot \sqrt{f_i}$. Let $C_{i,p}$ denote the minimum cycles of assigning $\{f_1, ..., f_i\}$ with $\sum_{j=1}^{i} b_j \cdot f_j \leq p$ and $0 \leq p \leq P_q$. Thus, for each f_i,

- if $\sum_{j=1}^{i} b_j \cdot f_j > p$, $C_{i,p} = C_{i-1,p}$.
- Otherwise, $C_{i,p} = \min\{C_{i-1,p}, C_{i-1,p-b_i f_i} + a_i\sqrt{f_i}\}|_{f_i = F_j}$, where $F_j \in \{F_1, ..., F_M\}$ as in Eqn 8.

The above steps can be called recursively until the solution is found. The dynamic programming based algorithm can be summarized in Algorithm 1.

Algorithm 1: Dynamic Programming Based Frequency Assignment

```
Input: $a_i$, $b_i$: the coefficients in Eqns. 2 and 3
Output: $C_{i,p}$: the minimum cycles given the power
        budget $\leq p$.
Function: Find the minimum cycles.
begin
    Initialize all the $C_{i,p}$ to be 0;
    for each $f_i$ do                      /* $i = 1, ..., N_q$ */
        for each $F_j$, do                 /* $\{F_1, \ldots, F_M\}$ */
            for each $p \leq P_q$ do
                if $\sum_{j=1}^{i} b_j \cdot f_j > p$ then
                    $C_{i,p} = C_{i-1,p}$;
                else
                    $C_{i,p} = \min\{C_{i-1,p}, C_{i-1,p-b_i f_i} + a_i\sqrt{f_i}\}|_{f_i=F_j}$;
            end
        end
    end
end
```

The time complexity of the above algorithm can be further reduced to be linear. To accelerate the computation, a multi-stage dynamic programming network (DPN) is designed to solve the problem for each application q with linear time complexity. It is first constructed by mapping the terms in the constraint (Eqn. 3) and objective (Eqn. 2) equations to the weights of the vertices or edges. Then the DPN is traversed to find a minimum weight path corresponding to the optimal solution.

In the DPN, each vertex represents a different power budget. An edge exists between two vertices in adjacent stages if the power consumption of assigning the frequency equals to the difference in the power budgets of the two vertices. Each edge is assigned a weight by the term in Eqn. 2, *i.e.*, minimum cycles by assigning frequency to a variable. If a stage (corresponding to a tile) is not in the application's region, the stage will be bypassed.

With the DPN, finding the minimum weight path is equivalent to an optimal frequency assignment. The traversal can be done in parallel. Each vertex at current stage selects the edge such that, the sum of the edge weight and the minimum cycles achieved by the later stage is minimum. This sum is transmitted back to vertices in the previous stage. In this manner, after reaching the source, the minimum weight path is found in linear time.

Fig. 1 shows the block diagram of APAT. The performance model in Section III.A for each application will be set up first with a regression model either online or offline. The DPN will be constructed based on the performance-power model and input power budget. Then the DPN is traversed to find the optimal solution.

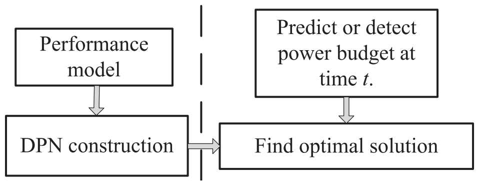


Fig. 1. The APAT system block diagram.

B. The dynamic programming network

The dynamic programming approach in above sub-section can be run in a dynamic programming network. Fig. 2 shows the transformation of the problem into a dynamic programming network. Once the DPN is constructed, it only needs minor update to remove some vertices and edges according to current power budget.

Definition 1. *Dynamic programming network. A dynamic programming network is denoted as a graph $DPN(V, E)$, with V and E represent the sets of vertices and edges, respectively.*

- Each vertex is assigned with two properties, $Cycle$ and P, where $Cycle\left(v_{i,j}\right)$ denotes the minimum cycles given the power budget $P\left(v_{i,j}\right)$ equal to j.
- An edge $e_{i,j,k}$ is added between vertices $\left(v_{i,j}, v_{i+1,k}\right)$, if $j + b_i \cdot f_i|_{f_i=F_l} = k$, for $l = 1, \ldots, M$.
- The weight of an edge $e_{i,j,k} = \left(v_{i,j}, v_{i+1,k}\right)$ is represented as $w_{i,j,k}$. $w_{i,j,k} = a_i\sqrt{f_i}|_{f_i=(j-k)/b_i}$ equals to the corresponding term in Eqn. 2.

The above edge connection means, there is a path between two adjacent vertices if the difference in the power budget corresponding to them equals to the power consumption by assigning frequency to the vertex between them.

Of the total of $P \times (N+1)$ vertices in a DPN, the network is organized as $N+1$ stages (as there are N frequency variables) with each group consists of P vertices (corresponding to the available power budget levels). Two dummy vertices, S and D, are added before stage 1 and after stage $N+1$, respectively, as in Fig. 2.

Note that if a stage i (corresponding to a tile) is not inside an application's region, it is bypassed, *i.e.*, connecting vertices directly to those in the previous stage with the edge weight set to be 0. This can be selected by a MUX as in Fig. 2.

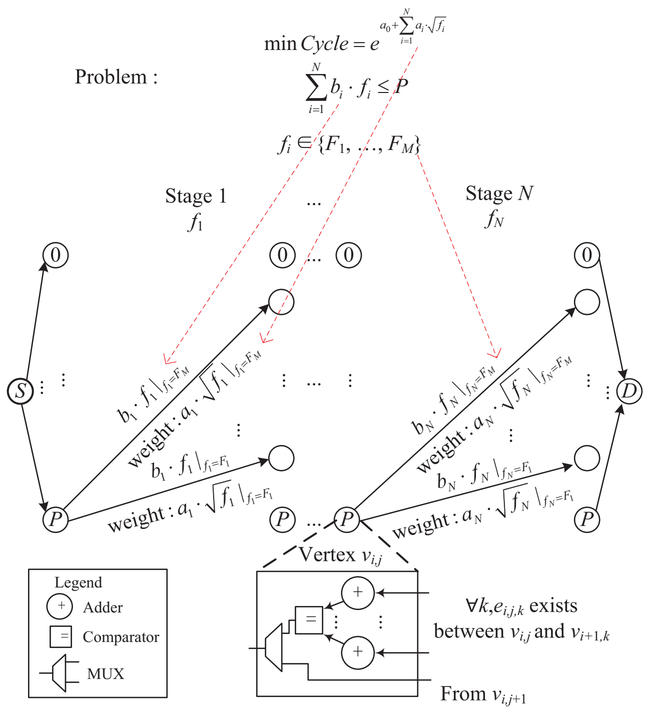
C. Find the optimum solution

The DPN can be traversed to find an optimal solution as follows.

- The edge $e_{i,j,k}$ is assigned with weight
$$w_{i,j,k} = a_i\sqrt{f_i}|_{f_i=(j-k)/b_i} \tag{9}$$
where a_i and b_i are defined in Eqns. 2 and 3.
- Find a minimum weight path from S to D, which corresponds to the optimal solution of the problem.

Note that f_m with $1 \leq m < i$ (of the stages proceeding stage i) are assigned a value at stage i, while f_n with $i < n \leq N$ (stages after stage i) are yet to be assigned.

The weight of each edge equals to the corresponding term in Eqn. 2 as shown in Fig. 2. When the power budget P_q changes,



Problem :
$\min Cycle = e^{a_0+\sum_{i=1}^{N} a_i \cdot \sqrt{f_i}}$
$\sum_{i=1}^{N} b_i \cdot f_i \le P$
$f_i \in \{F_1, \ldots, F_M\}$
Stage 1
f_1
Stage N
f_N
$b_1 \cdot f_1 |_{f_1=F_M}$
weight : $a_1 \cdot \sqrt{f_1}|_{f_1=F_M}$
$b_1 \cdot f_1 |_{f_1=F_1}$
weight : $a_1 \cdot \sqrt{f_1}|_{f_1=F_1}$
$b_N \cdot f_N |_{f_N=F_M}$
weight : $a_N \cdot \sqrt{f_N}|_{f_N=F_M}$
$b_N \cdot f_N |_{f_N=F_1}$
weight : $a_N \cdot \sqrt{f_N}|_{f_N=F_1}$
Vertex $v_{i,j}$
Legend
Adder
Comparator
MUX
$\forall k, e_{i,j,k}$ exists between $v_{i,j}$ and $v_{i+1,k}$
From $v_{i,j+1}$

Fig. 2. Transformation of the optimization problem to a dynamic programming network. In total, there are $P \times (N+1)$ vertices.

some vertices $v_{i,j}$ and the corresponding edges are removed if the power budget of $v_{i,j}$ (*i.e.*, j) exceeds P_q. To find the minimum weight path under the power constraint in linear time, the following dynamic programming equations can be calculated within N iterations [13] following a backward moving procedure (from D back to S). At each stage, each vertex calculates the following value, a modified version of the Bellman equations [13].

$$Cycle_{MIN}(v_{i,j}) = \min_{\forall v_{i+1,k}, \exists \text{ an edge } e_{i,j,k} \text{ between}(v_{i,j}, v_{i+1,k})} \{w_{i,j,k} + Cycle_{MIN}(v_{i+1,k})\} \tag{10}$$

where $Cycle_{MIN}(D) = 0$, and $w_{1,S,k} = w_{N,D,k} = 0$, with $k \in [1, P]$, *i.e.*, the edges connecting S to the vertices in the first stage and the vertices in stage N to D have a weight of 0.

Thus, the calculation of Eqn. 10 will become to find the minimum weight path (optimal solution) under the power constraint in N steps as follows,

$$\begin{aligned} Cycle^*_{MIN}(PATH_{S,D}) &= \min_{e_{i,j,k} \in PATH_{S,D}} \{ \sum_{i=1,j=1,k=1}^{i=N_q, j=P_q, k=P_q} w_{i,j,k} \} \\ &= \min_{f_i \in \{F_1, \ldots, F_M\}} \sum_{i=1}^{N_q} a_i \sqrt{f_i} \end{aligned} \tag{11}$$

where $PATH_{S,D}$ is the set of all the paths from S to D. The optimal path from $v_{i,j}$ to $v_{i+1,\mu}$ at each vertex $v_{i,j}$ (corresponding to the assignment of each frequency variable f_i by $(j-\mu)/b_i$) along the minimum weight path can be obtained as follows:

$$v_{i+1,\mu} = \arg\min_{\forall v_{i+1,k}, \exists \text{ an edge } e_{i,j,k} \text{ between}(v_{i,j}, v_{i+1,k})} \{w_{i,j,k} + Cycle^*_{MIN}(v_{i+1,k})\} \tag{12}$$

where the optimal assignment of f_i is $F_l = (j-\mu)/b_i$. In this way, the optimal assignment of each frequency variable is found and the system can be tuned to run under this frequency configuration.

The pseudo-code in Algorithm 2 shows the traversal of the DPN to find the minimum weight path from S to D. Both DPN construction and traversal take N_q iterations for each application. Each iteration only involves add and compare operations which could be done in one cycle. Thus, the worst case run time is $2N_q$ cycles.

Algorithm 2: FindMinWeightPath

```
Input: $e_{i,j,k}$: the weight of each edge, for
       $i, j \in [1, N+1]$ and $k \in [1, P]$.
Output: $Cycle(v_{i,j})$:the minimum cycles of each
        vertex after assigning $f_i$.
Function: Find the minimum weight path &
corresponding to the optimal solution.
begin
    Initialize all the $Cycle(v_{i,j})$ to be $\infty$, except
    $Cycle(D) = 0$;
    for stages $i$ from $N-1$ to $0$ do
        for each vertex $v_{i,j}$ do
            for adjacent vertex $v_{i+1,k}$ do    /* an
            edge $e_{i,j,k}$ connecting $v_{i,j}$ and
            $v_{i+1,k}$ */
                if
                $Cycle\left(v_{i+1,j}\right) + w_{i,j,k} < Cycle(v_{i,j})$
                then
                    $Cycle(v_{i,j}) =$
                    $Cycle\left(v_{i+1,j}\right) + w_{i,j,k}$;
                    $f_i = (j-k)/b_i$;
            end
        end
    end
end
```

V. EXPERIMENTAL RESULTS

A. Experimental setup

Experiments are performed using an in-house developed event-driven many-core simulator [14]. Table I lists the configuration of the many-core simulator and the mixes of the benchmarks selected from PARSEC and SPLASH-2. Each of the applications occupies a 4×4 mesh region in the many-core system. The weights of each application (Eqn. 5) is randomly selected whose sum equals to 1.

In all the experiments, we select a 8×8 2D mesh as the underline NoC topology. We compare the performance of the proposed APAT against three other best known schemes: (1) PGCapping [6], where both the number and the frequency of tiles can be adjusted, (2) PEPON [4], where the frequency of each processor and last-level cache bank can be adjusted, and (3) DPPC [11], a linear programming based approach. In what

TABLE I. PARAMETERS USED IN THE SIMULATION

Number of processors	64 (MIPS ISA 32 compatible)
Fetch/Decode/Commit size	4 / 4 / 4
ROB size	64
L1 D cache (private)	16KB, 2-way, 32B line, 2 cycles, 2 ports, dual tags
L1 I cache (private)	32KB, 2-way, 64B line, 2 cycle
L2 cache (shared) MESI protocol	64KB slice/node, 64B line, 6 cycles, 2 ports
Main memory size	2GB
On-chip network parameters	
NoC flit size	72-bit
Data packet size	5 flits
Meta packet size	1 flit
NoC latency	router 2 cycles, link 1 cycle
NoC VC number	4
NoC buffer	5 × 12 flits
Workload mixes used as multiple applications running in a single NoC	
Mix-1	blackscholes, ferret, freqmine, swaptions
Mix-2	streamcluster, dedup, canneal, vips
Mix-3	barnes, raytrace, swaptions, vips
Mix-4	dedup, freqmine, fft, canneal

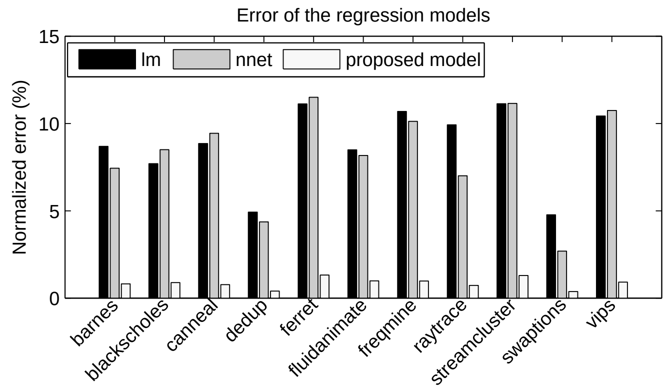


Fig. 3. Comparison of the three models, linear regression (lm), neural network (NNet) and the proposed model as in Eqn. 2. Each of the application is runing on the 8x8 many-core system. All the errors (NRMSE) are normalized to the mean value of the cycles.

follows, we will present the verification of regression model of the performance vs. frequency first. Next the performance of the proposed APAT is compared against that of related approaches. In the end, the hardware cost and overhead of the DP-based optimization are analyzed.

B. Precision of the performance model

Suitable performance model plays an important role in the problem formulation. To justify the accuracy of the performance model in Eqn. 2, we compare it against two other approaches, the linear regression model (lm), and neural network model (nnet). In the lm model, the execution time is a linear combination of the frequencies of the tiles. In the nnet model, the relationship of the execution time and the frequencies of the tiles are found by an artificial neural network. We use the Matlab neural network toolbox to train a three stage neural network by varying the number of neurons in the hidden layer. The metric used here is the normalized root square mean squared error (NRMSE). Fig. 3 shows the errors of the three regression models, where the proposed regression model has shown significantly smaller error than the other two.

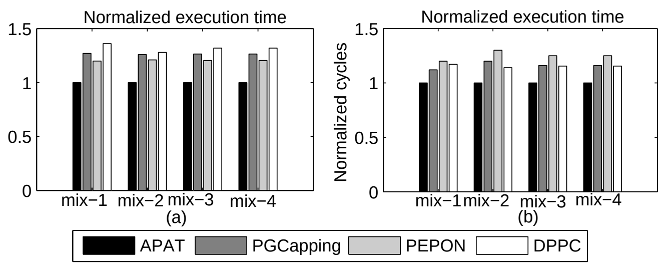


Fig. 4. Execution time of the mixes under power budget of (a) 150W and (b) 70W.

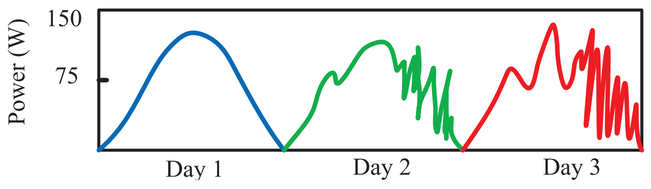


Fig. 5. Solar energy data of three days in 2011 from ORNL website

C. Comparison of the power allocation methods

Fig. 4 shows the performance of the four methods when the power budget is high, 150W as in (a), and low, 70W as in (b). The execution time is measured as the total execution time of all the applications in one mix. From Fig. 4, one can see that APAT has the lowest execution time under both high and low power budgets. When the power budget is high, APAT records an average of 26 %, 20 % and 30 % less time than that of PGCapping, PEPON, and DPPC, respectively. When the power budget becomes low, on average, OPAD needs 19 %, 25 % and 16 % less execution time than that of PGCapping, PEPON, and DPPC, respectively.

To support solar energy powered system, a power budget estimator is used based on history to predict the power budget for the next period [9]. The prediction time interval is set to be 1 sec. The renewable energy can be smooth or highly variable, depending on the environment. The power data in Fig. 5 represent three days in different weather: a nice day, a day with nice morning and poor afternoon, and a terrible day. To see the real time power adaptiveness of the four methods, Fig. 6 compares the four methods with a variable power budget to mimic power budget from the renewable energy. The closer the actual power consumption is to the input power budget, the less the energy loss, and thus the better performance. In Fig. 6 the power budget is more smooth, representing a day of good weather. From Fig. 6 the power consumption of PGCapping, PEPON, and DPPC does not match the input power consumption and results in more energy loss. The energy losses (defined as the input power budget minus the power consumption and integrated over the time shown in Fig. 6) of PGCapping, PEPON, and DPPC over that of APAT are 11×, 4×, and 10×, respectively. In Fig. 7, lots of "jitters" or variations are made to mimic a day of poor weather. When the input budget variation is slow in Fig. 7 APAT still achieves better matching than the other three methods. The energy losses of PGCapping, PEPON, and DPPC over that of APAT are 11.3×, 3.5×, and 10×, respectively. Thus, APAT has the least energy loss.

Summary From the above experiments, APAT can have lower execution time given the same input power budget compared to three state-of-the-art methods, PGCapping, PEPON,

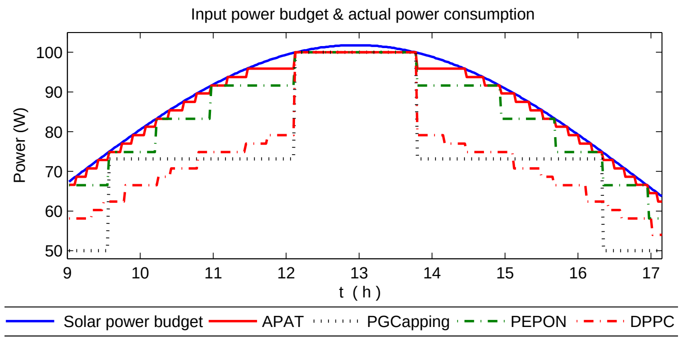


Fig. 6. Input power budget with low variation frequency mimicking a day of good weather and actual power consumption.

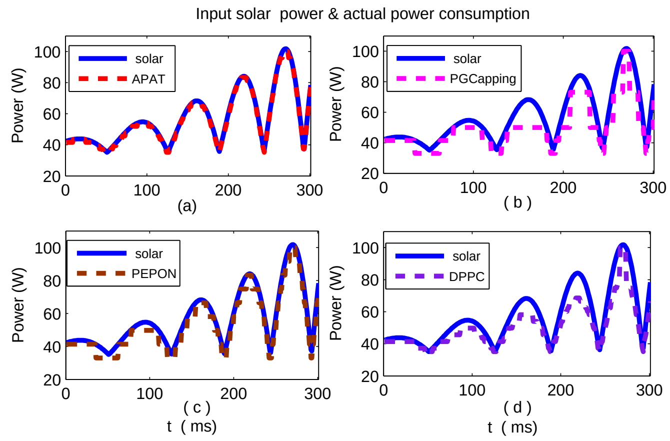


Fig. 7. Input power budget with variation frequency mimicking a day of bad weather and actual power consumption. The differences are shown between the input power budget and (a) APAT, (b) PGCapping, (c) PEPON and (d) DPPC.

and DPPC. On the other hand, as APAT has much lower overhead in power allocation, it is a beneficial method for many-core systems in which the power budget varies rapidly. Power consumption of APAT matches the input budget much better than the other three methods, resulting in less energy loss. Thus, APAT is suitable for online adaptive power allocation.

D. Hardware cost and runtime overhead

The hardware cost of the proposed APAT is mainly due to the nodes in the dynamic programming networks. Each node operation includes a 16-bit comparator and an adder. Each node has an area of 121 μm^2 and consumes 20 μW of power (assuming switching activity of 0.5) using Synopsys Design Compiler under 65nm TSMC library. As there are a total of $P \times (N+1)$ vertices in DPN, for a system with 64 tiles and P in Eqn. 3 normalized to 10, the whole DPN area can be 78650 μm^2 and consumes 13 mW of power. For reference, a single 5×5 router with 2 virtual channels, flit size of 75 bits, and 4 flit-depth FIFO has an area of 145890 μm^2 and consumes 8 mW of power. That is, the total area and power consumption of the DPN is 53 % and 160 %, respectively, of a single router. The power consumption of the whole DPN is about 2% of the power consumed by the 64 routers.

The total run time of the APAT to allocate power online is $2N_q$ cycles, where N_q is the tile number (number of frequency variables) of application q. For an 16-tile application, only 32 cycles is needed. As a comparison, PGCapping, PEPON, and DPPC each typically takes about $10 \sim 100$ M cycles, which is 6 to 7 orders of magnitude higher than that of APAT. In APAT, the regression model takes 10^5 cycles to compute. However, this is infrequently invoked; it only occurs at the initialization or at update interval for model error correction (typically, every 10 million cycles).

VI. Conclusions

In this paper, a power allocation algorithm was proposed to optimize performance with a power budget limit in energy harvesting based many-core systems. A dynamic programming approach is used which can run on a dynamic programming network to solve the problem with linear time complexity. The dynamic programming network is formed based on the performance-power model. The optimal solution can be found by traversing the network. This proposed algorithm can reduce execution time up to $20 \sim 30\%$ on average, compared to three existing approaches, PGCapping, PEPON, and DDPC with both smooth input power budget or power budget with rapid variations. The proposed algorithm also has very low run time, power and area overhead.

References

[1] S. Borkar, "Thousand core chips: a technology perspective," in *Proc. Design Automation Conf*, pp. 746–749, ACM, 2007.

[2] M. Gorlatova, P. Kinget, I. Kymissis, D. Rubenstein, X. Wang, and G. Zussman, "Energy harvesting active networked tags (EnHANTs) for ubiquitous object networking," *IEEE Wireless Communications*, vol. 17, no. 6, pp. 18–25, 2010.

[3] C. Li, W. Zhang, C. Cho, and T. Li, "SolarCore: solar energy driven multi-core architecture power management," in *Proc. IEEE Int'l Symp. High Performance Computer Architecture*, pp. 205–216, IEEE, 2011.

[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. Int'l Conf. Parallel Architectures and Compilation Techniques*, pp. 65–74, ACM, 2012.

[5] S. Imamura, H. Sasaki, N. Fukumoto, K. Inoue, and K. Murakami, "Optimizing power-performance trade-off for parallel applications through dynamic core and frequency scaling," *Proceedings of the RESoLVE*, 2012.

[6] K. Ma and X. Wang, "PGCapping: exploiting power gating for power capping and core lifetime balancing in CMPs," in *Proc. Int'l Conf. Parallel Architectures and Compilation Techniques*, pp. 13–22, 2012.

[7] C. Shen, X. Wang, W. Zhang, and F. Kang, "A high-performance three-dimensional micro supercapacitor based on self-supporting composite materials," *Journal of Power Sources*, vol. 196, no. 23, pp. 10465–10471, 2011.

[8] C. Li, A. Qouneh, and T. Li, "iSwitch: coordinating and optimizing renewable energy powered server clusters," in *Proc. Int'l Symp. Computer Architecture*, pp. 512–523, IEEE, 2012.

[9] J. Lu, S. Liu, Q. Wu, and Q. Qiu, "Accurate modeling and prediction of energy availability in energy harvesting real-time embedded systems," in *Proc. Int'l Green Computing Conf.*, pp. 469–476, 2010.

[10] S. Liu, J. Lu, Q. Wu, and Q. Qiu, "Harvesting-aware power management for real-time systems with renewable energy," *IEEE Trans. Very Large Scale Integration Systems*, vol. 20, no. 8, pp. 1473–1486, 2012.

[11] K. Ma, X. Wang, and Y. Wang, "DPPC: dynamic power partitioning and control for improved chip multiprocessor performance," *IEEE Trans. Computers, in press*, 2013.

[12] C. M. Bishop, *Pattern recognition and machine learning*. Springer New York, 2006.

[13] T. Mak, P. Cheung, K. Lam, and W. Luk, "Adaptive routing in network-on-chips using a dynamic-programming network," *IEEE Trans. Industrial Electronics*, vol. 58, no. 8, pp. 3701–3716, 2011.

[14] J. Xue, A. Garg, B. Ciftcioglu, J. Hu, S. Wang, I. Savidis, M. Jain, R. Berman, P. Liu, M. Huang, H. Wu, E. G. Friedman, G. Wicks, and D. Moore, "An intra-chip free-space optical interconnect," in *Proc. Int'l Symp. Computer architecture*, pp. 94–105, ACM, 2010.