PARMA: Parallelization-Aware Run-time Management for Energy-Efficient Many-Core Systems

Mohammed A. Noaman Al-hayanni, *Member, IEEE,* Ashur Rafiev, Fei Xia, Rishad Shafik, *Senior Member, IEEE,* Alexander Romanovsky, Alex Yakovlev, *Fellow, IEEE*

Abstract—Performance and energy efficiency considerations have shifted computing paradigms from single-core to many-core architectures. At the same time, traditional speedup models such as Amdahl's Law face challenges in the run-time reasoning for system performance and energy efficiency, because these models typically assume limited variations of the parallel fraction. Moreover, the parallel fraction, which varies dynamically in workloads, is generally unknown at run-time without application-level instrumentation. This paper describes novel performance/energy trade-off models based on realistic architectural considerations, which describe the parallel fraction and speedup as functions of performance counter values available in modern processors, removing the need for application-level instrumentation. These are then used to develop a Parallelization-Aware Run-time Management (PARMA) approach. PARMA aims at controlling core allocations and operating voltage/frequency points for energy efficiency, according to the varying workload parallel fractions. The efficacy of our models and the PARMA approach is extensively validated using a number of PARSEC benchmark applications, involving two performance/energy trade-off metrics: energy-delay-product (EDP), typically used in high-performance applications and energy per instruction (EPI), suitable for energy-aware applications. Up to 48 and 68 per-cent improvements in EDP and EPI have been observed using the PARMA approach compared with parallelization-agnostic methods.

Index Terms—run-time management; many-core; speedup; power modelling; energy-delay-product; energy per instruction.

1 Introduction

Multiple processing cores on the same die [1] enable higher computational performance by distributing workloads among parallel cores. Speedup measures this improvement, by comparing the performance of a multi-/many-core implementation to its single-core equivalent.

Amdahl's Law defines speedup as a function of parallel and sequential elements and resources in a system [2]. According to this law, speedup is related to the parallelizability of a workload, defined by the parallel fraction (p), and the number of parallel resources. Although the original Law describes p as a function of ideal parallel cores only, in reality it defines how effectively parallel the workload is under architectural constraints, such as shared memory [3].

In recent years, application programming models and hardware systems have acquired powerful parallelization features, including micro-architectural support, e.g. instruction-level parallelism (ILP) [4], and macro-architectural support, e.g. thread-/core-level parallelism [5]. These features are often intertwined internally by compilers to provide better computational performance for the parallel part of the workload [6]. However, when parallel parts are interleaved with sequential parts with shared resources between them, it can make effective parallel fraction per workload vary dynamically and unpredictably during execution (See Figure. 1 and Section 6.2).

- *M. Al-hayanni, A. Rafiev, F. Xia, R. Shafik, A. Romanovsky, and A. Yakovlev are with Newcastle University, UK*
  E-mail: {m.a.n.al-hayanni, ashur.rafiev, fei.xia, rishad.shafik, alexander.romanovsky, alex.yakovlev}@ncl.ac.uk
- *M. Al-hayanni is also with University of Technology-Iraq*
  E-mail: {110093@uotechnology.edu.iq}
- *This work is supported by EPSRC/UK as a part of PRiME EP/K034448/1 and STRATA EP/N023641/1 projects.*

Run-time management is a set of methods for managing hardware/software knobs and monitors under variable workloads to improve system operation, for example by optimizing some chosen metric in the performance/energy trade-off [7]. Existing run-time algorithms react to workload change by dynamically scaling voltage/frequency (DVFS) [8], in combination with task mapping and core allocations [9]. However, so far the natural attribute of applications that lends itself to distribution into parallel cores, the p fraction, has not been taken into account for run-time control, as discussed in Section 2 in detail.

Workload parallelizability can be a significant indicator for the run-time allocation of parallel resources. Run-time decisions may be tuned to the value of p. For instance, if the p fraction is low, there is little reason to use too many cores.

A crucial requirement of such parallelization-aware run-time controls is to know p value closely during a workload's execution (Figure 1). However, the p fraction can vary dynamically depending on application workload exercised on a specific platform, and may require extensive software instrumentation [5]. Our previous work in [10] has shown the possibility of determining the p fraction by statically analysing the system performance counter readings. In this work, we extend the method to enable run-time tracking of the instantaneous value of a variable p fraction for Parallelization-Aware Run-time MAnagement (PARMA). To the best of our knowledge, PARMA is the first approach in run-time modeling and management using the accurate measurement of p fraction of tasks/threads. We demonstrate the strength of this method with practical implementation and validation on real off-the-shelf systems.

This paper makes the following specific *contributions*:

- Develop a new practical method for determining

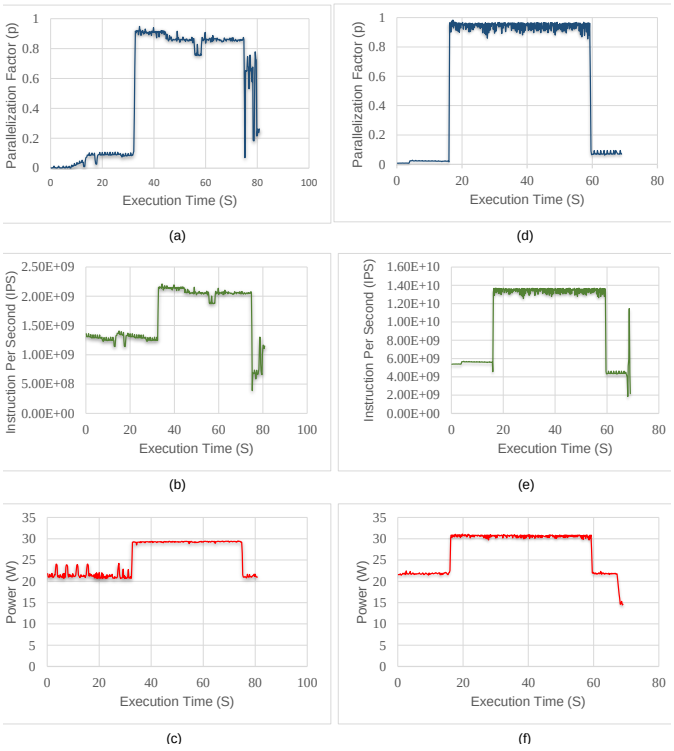


Fig. 1. The same workload may display different p values in different phases of execution. High p-variation benchmarks: a), b), c) `canneal`; d), e), f) `blackscholes`.

workload parallel fraction p at run-time using hardware performance counters and Amdahl's Law. The method improves on our previous work aimed at offline analysis [10].

- Generate parallelization-aware power models for many-core processors taking into account the full-domain and per-core DVFS assumptions, such that the optimal operating points for a given platform can be calculated with regard to the following optimization metrics: Energy per Instruction (EPI), Energy-Delay Product (EDP), or the general energy-delay trade-off metric ED^{x} defined in Section 4.2.
- Implement a PARMA run-time and validate its efficacy using a number of benchmark applications on an off-the-shelf multi-core system.

The rest of the paper is organized as follows. Section 2 describes related work. Section 3 gives the background on our proposed extended speedup and parallelization fraction models. Section 4 proposes a new paradigm for parallelization-aware energy-efficient computing, starting with many-core power modeling, EPI and EDP modeling and optimization. Section 5 explains the parallelization-aware run-time management algorithm. Section 6 describes the experimental set-up, hardware performance counters and the benchmark applications used in this study, and presents a cross-validation of the models with measured speedups and considers the applicability of our method. Finally, Section 7 concludes the work in this paper.

2 Related Work

This section provides a summary of the research relevant to parallelization-aware run-time management.

A framework called *Thread Reinforcer* was developed [17] in order to dynamically monitor multi-thread execution. It can determine the number of threads that can produce the best speedup. Hardware performance counters, a favored method of on-chip monitoring [18], can also be useful as monitors for this purpose [15], [19]. Continuous monitoring can provide timely state control by combining information about speedup and power [16], [19]. Workload and instantaneous hardware capabilities can be matched to improve speedup while satisfying efficiency metrics [6].

An efficient run-time system for many-core systems [20] controls both types of on-chip actuators, thread-to-core mapping (also known as context switching) and DVFS, together rather than only using one of them. A heuristic algorithm uses hardware performance counters to predict run-time power and performance.

Per-core DVFS, which provide DVFS controls at the granularity of individual cores [21], [8] provides enhanced controllability. With per-core DVFS, the number of active cores may be associated with the number of active threads [22]. In addition, a wide range of DVFS points with run-time DVFS optimization was shown to improve speedup, power and energy in real time systems [23], [24].

As these examples show, the number of threads has been a favorite handle for optimization with a substantial body of work in the literature. However, the Amdahl p fraction has not been investigated to date in the same context. Table 1 summarizes existing studies relevant to this work. The run-time determination of the p fraction and per-core DVFS do not feature in existing work. As such, using speedup and parallelization models to identify suitable core allocations and operating frequencies remains challenging These are the gaps that this paper addresses.

3 Performance and Speedup Models

Amdahl's Law is based on an idealistic approximation that divides a workload into completely sequential and parallel parts, executed on a given number of processors. The ratio between the parallel part and the entire workload, 0 p 1, is called *parallel fraction*. In reality p is not the property of a workload alone. It implicitly incorporates all effects including hardware bottlenecks, such as memory and interconnects, and software properties, such as programming models. It is a property of the entire hardware-software system.

In a run-time system, control actions require time and introduce additional overheads [19]; therefore an efficient real-world run-time management should use control cycles with reasonable durations. The core idea behind PARMA is that the system behavior within such a control cycle can be described using parallel fraction as a single dynamic parameter. It also assumes that similar p values correspond to a given set of optimal operating configurations. In this paper we successfully demonstrate that this approximation can be practically used in an efficient run-time environment. Section 4 presents the methodology of finding the optimal operating points, while this section focuses on the

TABLE 1
Existing Optimization Methods and the Proposed Study

Previous studies	Run-time	Parallelization	speedup based	Validation	Application	DVFS	Optimization
[11], [9]	Yes	No	No	Simulation	Single	Full-domain	Off-line
[12]	Yes	No	No	Simulation	Single	Full-domain	On-line
[13]	Yes	Instr. level	No	Simulation	Single	Full-domain	Off-line
[14]	Yes	No	No	Experimental	Single	Full-domain	On-line
[15]	Yes	No	Yes	Experimental	Single	No	On-line
[16]	Yes	No	Yes	Experimental	Single	Full-domain	On-line
[6]	Yes	Calculated	Yes	Experimental	Single	No	Off-line
Proposed	**Yes**	**Measured**	**Yes**	**Experimental**	**Single**	**Full and Per-core**	**On-line**

theoretical foundation of calculating p from the system performance.

3.1 System Performance

We consider system performance as instructions per second (IPS) and denote the performance of the n-core execution as $IPS\ (n)$. In this work, we assume that the average number of instructions per clock cycle (IPC) of a single core execution is constant for a given workload:

$$IPC\ (1) = \text{const}. \tag{1}$$

Therefore the system performance can be modeled by the system operating frequency F using a linear relationship:

$$IPS\ (n) = IPC\ (n) \quad F. \tag{2}$$

Under this definition, IPC includes both halted and unhalted clock cycles to ensure that the performance is represented in relation to the system (wall clock) time rather than CPU time. Halted cycles happen during a non-parallelizable part of the workload execution and are the main source of performance reduction in this model.

The speedup of n cores compared to m cores is defined as a ratio of performances:

$$S\ (n, m) = \frac{IPS\ (n)}{IPS\ (m)}. \tag{3}$$

Typically, speedup models consider $m = 1$, but in this work we define the speedup between any two numbers of cores. The numbers m and n represent cores available to the application. The fundamental assumption behind PARMA is that this number can change during the execution, for example, because of the run-time management.

3.2 Amdahl's Speedup Model

Amdahl's model assumes that the total workload constitutes I instructions. The workload I is also split into Ip parallelizable instructions and $I\ (1 \quad p)$ instructions that must be executed sequentially. Assuming that time $t\ (1)$ is needed to execute both the sequential and parallel parts of workload I on a single core and the time $t\ (n)$ is needed to execute the sequential part of workload I on a single core and the parallel part on n cores, we have:

$$t\ (1) = \frac{I}{IPS\ (1)}, \text{ and} \tag{4}$$

$$t\ (n) = \frac{(1 \quad p)\ \ I}{IPS\ (1)} + \frac{p \quad I}{n \quad IPS\ (1)}, \tag{5}$$

where $IPS\ (1)$ is single-core performance. The IPS-based classical Amdahl's speedup model is thus [2]:

$$S\ (n, 1) = \frac{I \quad t(1)}{I \quad t(n)} = \frac{1}{(1 \quad p) + \frac{p}{n}}. \tag{6}$$

It is also possible to compare the speedup between n cores and m cores, where $n \quad 1, m \quad 1$. From (3) we have:

$$S\ (n, m) = \frac{I \quad t(m)}{I \quad t(n)} = \frac{(1 \quad p) + \frac{p}{m}}{(1 \quad p) + \frac{p}{n}}. \tag{7}$$

This law shows an indirect relation between p and performance counters via the speedup. By knowing the initial m-core performance $IPS\ (m) = I/t\ (m)$ and p, it is possible to calculate the performance on n cores. This is often used to solve an optimization problem, i.e. which value of n gives the highest performance. The major challenge here is that p is usually an unknown parameter. However, this can be addressed if p can be measured directly from the architectural performance counters at run-time, which is demonstrated in the next section.

3.3 Estimation of Parallel Fraction

Finding p can be done by measuring the speedup $S\ (n, m)$ as a ratio of performances between two executions on different numbers of cores and then solving (7) for an unknown p:

$$p = \frac{S\ (n, m) \quad 1}{S\ (n, m) \quad \frac{(n-1)}{n} \quad \frac{(m-1)}{m}}. \tag{8}$$

$S\ (n, m)$ is calculated according to (3), with IPS values directly obtained from performance counters. The average p of an entire workload can be obtained from offline analysis of execution traces [10], but the method has limited use as it requires complete workload executions. In this paper, we modify the method in two important ways: 1) it can be used in run-time while the workload is being executed and 2) it provides an instantaneous estimate of a workload's p within a given time window. Given that p has distinct phases during a workload's execution, similar to other properties [18], [25], the average p of an entire workload is less useful than its instantaneous p for run-time use.

We can avoid running the application fully twice if we change the number of available cores in the middle of execution. Let's split the workload arbitrarily in two parts, $I_1 + I_2 = I$ and run each part on a different number of cores (m and n respectively), as shown in Figure 2. For the moment, we assume that the parallel fraction remains constant throughout the execution: $p_1 = p_2 = p$. Since the number of cores change, the sub-parts I_1 and I_2 must

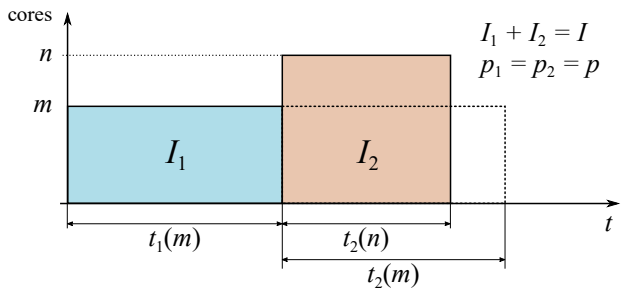


Fig. 2. Determining p by splitting the workload in two arbitrary parts I_1 and I_2.

show different speedup values. However, since p is assumed to be constant in this scenario, we can use the speedup of any of the parts in order to calculate (8); let's use $S_2(n,m) = (I_2/t_2(n)) / (I_2/t_2(m))$. The value of $t_2(m)$ is unknown but relates to the measurable $t_1(m)$ as follows:

$$t_2(m) = \frac{I_2}{I_1} \; t_1(m), \tag{9}$$

which can be found from (5). This formally derives that the speedup substitute for (8) can be calculated from the IPS measurements as follows:

$$S_2(n,m) = \frac{I_2 \;\; t_1(m)}{I_1 \;\; t_2(n)} = \frac{IPS_2(n)}{IPS_1(m)}. \tag{10}$$

Under the PARMA assumptions of the dynamic p, we cannot apply $p_1 = p_2 = p$ to the entire workload; however, we can still approximate p to a constant within a small sliding window. The scenario shown in Figure 2 represents two most recent PARMA control cycles: $t_1(m)$ time and I_1 instructions for the previous cycle, and $t_2(n)$ time and I_2 instructions for the current cycle. The control then assumes that the current value for p, determined from these two cycles, will be valid for at least the next control cycle and can be used to find the current optimal system configuration. In the worst case, if this is not true, we lose optimal decision in only one control cycle, hence for sufficiently small control cycles and sufficiently long periods of stable workload p, this approach should be acceptable.

Another problem remains: (8) requires $m \neq n$, i.e. if we want to always use the two most recent control cycles, we need to change the number of cores on every control cycle, leading to inefficiency. We need to update the value of p based on its previously determined value without changing the number of cores. Let us assume that the base core performance $IPS(1)$ remains the same between two recent cycles, i.e. the core frequency does not change. According to Amdahl's Law, if p and n remain the same, we should expect $IPS_1(n) = IPS_2(n)$. On the other hand, a change in performance indicates a p change if n stays the same. In this case, we have $S_1(n,1) = IPS_1(n)/IPS(1)$ based on the previous p_1, which is assumed to be known from the previous control cycle, and $S_2(n,1) = IPS_2(n)/IPS(1)$ based on p_2, which we need to calculate. From (6) we can derive the following equation:

$$IPS_1(n) \quad (1 \quad p_1) + \frac{p_1}{n} \quad = IPS_2(n) \quad (1 \quad p_2) + \frac{p_2}{n}, \tag{11}$$

from which, in the case of $n > 1$, we can find p_2:

$$p_2 = \frac{IPS_1(n)}{IPS_2(n)} \; p_1 \quad \frac{IPS_1(n)}{IPS_2(n)} \quad 1 \quad \frac{n}{n \quad 1}. \tag{12}$$

The minimum number of cores we can run to use this equation is 2, even if the workload is purely sequential. It is a necessary overhead for capturing the sequential to parallel transition as workload dynamically varies.

So far, all models in this section assume constant IPS (1). It is possible to extend the models to variable frequency by using performance assumptions (1) and (2). When the frequency changes between control cycles from F_1 to F_2, we can recalculate the models for one control cycle in the time domain of another control cycle by using the ratio $IPS_1(1)/F_1 = IPS_2(1)/F_2$.

However, in real applications, IPC (1) can also change over time in the same way as p. The method still works if IPC (1) stays approximately the same within two consecutive measurement cycles.

4 Proposed Parallelization-Aware Models

This section describes the power modelling methodology aimed at determining the optimal operating points for a given platform. The method considers two types of DVFS control: full-domain and per-core. The presented models are parametric, and the proposed characterization method is based on a series of experiments with a synthetic benchmark. Any given platform needs to be characterized only once, therefore the calculations in this section do not contribute to the run-time overheads.

4.1 Many-Core Power Modelling

The total power W_{total}, includes background switching power W_0, effective power W_n, and leakage power W_L [26]:

$$W_{\text{total}} = W_0 + W_n + W_L. \tag{13}$$

Background switching power is the dynamic power always consumed by the cores unrelated to the workload. Effective power is the switching power consumed by the workload. This work considers leakage power W_L to be constant and does not consider thermal effects.

W_0 and W_n can be expressed as functions of the supply voltage V, frequency F, and switching capacitance [27]:

$$W_0 = \frac{1}{2} \; A_0 \; V^2 \; F, \quad W_n = \frac{1}{2} \; A_n \; V^2 \; F, \tag{14}$$

where A_0 is the capacitance of components contributing to the background switching power, and A_n is the capacitance of components contributing to the effective power, which depends on the activity factor and can be calculated as the effective capacitance of a single core A_1 multiplied by the speedup (6). The total power of executing a given workload on n cores can be calculated as follows:

$$W_{\text{total}} = \frac{1}{2} \; (A_0 + A_1 \; S(n,1)) \; V^2 \; F + W_L. \tag{15}$$

Equation (15) can be applied to full-domain DVFS platforms, where all cores operate on the same frequency and voltage, and idle cores remain online contributing to the background and leakage powers. Per-core DVFS control

provides much greater flexibility in controlling system power by tuning down or switching off individual cores. Although at present this feature is often unavailable in off-the-shelf consumer devices, we extend our models to cover such future systems because of their advantages.

Amdahl's Law for homogeneous systems implies even distribution of the workload between the cores, hence it does not provide a theoretical basis for individually tuning DVFS of active cores. However, for the execution on n out of $n_{\max}$ cores, it is still possible to distinguish ($n_{\max}$ n) unused cores and reduce their contribution to the total power. If we set the voltage and frequency of unused cores to V_{un} and F_{un} while running the workload on n cores at V and F, the total power becomes:

$$W_{\text{total}} = \frac{1}{2} \frac{n}{n_{\max}} A_0 + A_1 S(n,1) V^2 F + \\ + \frac{1}{2} \frac{n_{\max} n}{n_{\max}} A_0 V_{\text{un}}^2 F_{\text{un}} + W_{\text{L}}. \quad (16)$$

It is possible to further reduce the power if we allow switching off the unused cores, so they no longer contribute to the background power and also the leakage power:

$$W_{\text{total}} = \frac{1}{2} \frac{n}{n_{\max}} A_0 + A_1 S(n,1) V^2 F + \frac{n}{n_{\max}} W_{\text{L}}. \quad (17)$$

In this paper we use (17) to model the platforms with per-core DVFS capabilities.

In commercial platforms, each possible operating frequency value F is typically coupled with a supply voltage V, forming a VF-pair. The set of VF-pairs is predefined by the manufacturer in a rather conservative way and forms the platform's DVFS table. Since we aim to solve the power optimization problem analytically, we require the voltage V to be expressed as a differentiable function of frequency F. This function can be obtained by curve-fitting the data from a given DVFS table. The results for an actual platform can be found in Section 6.1.

Thus, (15) and (17) define the total platform power as functions of p, n, and F, where A_0, A_1, and W_{L} are platform-specific constants that can be obtained from the characterization experiments as described in Section 4.3.

4.2 EPI and EDP Modelling and Optimization

Power-normalized performance (PNP) and its reciprocal, energy per instruction (EPI), are popular metrics for optimizing system power consumption [28]. EPI is calculated as W_{total} t_{ins}, where t_{ins} is the instruction delay that can be determined from the execution time and the workload. From (2), (4), and (6) we have:

$$t_{\text{ins}} = \frac{t(n)}{I} = \frac{1}{S(n,1) \; IPC(1) \; F}. \quad (18)$$

Hence, we can express EPI as a function of p, n, and F:

$$EPI = \frac{W_{\text{total}}}{S(n,1) \; IPC(1) \; F}, \quad (19)$$

where W_{total} can be calculated from (15), (16) or (17), depending on the platform, and $IPC(1)$ is an application-dependent constant as assumed in (1).

One argument against PNP or EPI is that it only optimizes energy, and the system performance is not considered. A commonly used technique to balance multi-variable optimization is to add a trade-off exponent x to the equation to create a general energy-delay trade-off metric:

$$ED(x) = W_{\text{total}} \; t_{\text{ins}}^x = \frac{W_{\text{total}}}{(S(n,1) \; IPC(1) \; F)^x}. \quad (20)$$

For $x = 1$ the metric transforms into EPI, and $x = 2$ gives another popular optimization metric known as energy-delay product (EDP) [29]. In our validation experiments in Section 6 we focus on these two trade-off metrics; however, the method accepts any x 0, including non-integer values.

> The goal of PARMA run-time is to determine the optimal operating tuple (n, F) for the estimated instantaneous parallel fraction p (Section 3.3), in order to minimize $ED(x)$ for any given x.

Minimization can be solved using gradient descent:

$$n_{i+1} = n_i \; \lambda \frac{\partial ED(x)}{\partial n}, \quad (21)$$

$$F_{i+1} = F_i \; \lambda \frac{\partial ED(x)}{\partial F}, \quad (22)$$

where λ is the learning rate. Starting from an arbitrary point (n_0, F_0), partial derivatives $\partial ED(x)/\partial n$ and $\partial ED(x)/\partial F$ are used to gradually approach the local minimum. Note that p and x are constant during this calculation.

The method is not aimed at calculating the actual values of $ED(x)$ but aims at locating its minima. The application-dependent constant $IPC(1)$ scales the calculation by a constant factor, but does not move the optimal points, which means that the optimization results are system-wide and independent of the workload-specific $IPC(1)$. The only determinant variable for the proposed method is p.

Solving (21) and (22) can be computationally expensive and doing it in every run-time control cycle may not be advisable. However, it is possible to use offline pre-calculation for a finite set of discrete input (p) values. We subdivide the range of 0 p 1 into h evenly spaced bins, such that each i-th bin represents p_i $\frac{1}{h}$ parallel fractions, where 1 i h. In our experiments, we chose $h = 100$ to obtain a 1% precision on p. The number of optimization modes supported by the run-time (selected by the parameter x) is also finite. Solutions for (21) and (22) in each combination of (x, p_i) form a lookup table for run-time use to determine optimal n and F with minimal computation overhead. The table is constructed only once for any given platform.

Our approach uses p as the sole parameter to describe the instantaneous behavior of a workload. Under this assumption, the calculated lookup table applies to all applications that can be executed on the platform without needing prior knowledge before run-time. This is similar to the approach adopted in [25] where multiple workloads are classified during run-time for their CPU memory usage characteristics without prior knowledge.

Compared to [25], this work only studies single application scenarios. We consider the challenges posed by multiple concurrent applications as a part of future work.

Two methods show promising potential. One is to adapt the method employed in [25], and classify multiple running and incoming applications according to their p values and make run-time control decisions based on the classification results. Alternatively, it may also be possible to view concurrently running applications as a single combined workload with an overall dynamically changing p. Both methods require further investigations.

4.3 Methodology for Model Characterization

This section gives a platform-independent description of the methodology for experimentally obtaining model parameters for any given platform.

System power measurements are needed for characterizing the system power models. These measurements are performed only once at the stage of model characterization and are not required during the PARMA run-time control. This means that, even if the platform does not provide online power sensors, it is still possible to use the method with a specifically instrumented experimental rig.

The method of finding A_0, A_1, and W_L is based on collecting power readings for the range of values for n, p, and F and curve-fitting the measurements to (15). It is difficult to assemble a set of standard benchmarks that would provide a good coverage for the entire range of p values and at the same time show stable parallel behavior. Instead, we built a synthetic benchmark (available in open source[1]), which implements a controllable parallel fraction, such that p can be set to a given value during the experiments. Figure 3 illustrates the flowchart of the benchmark, which has distinct sequential and parallel sections. It is based on a looped arithmetic calculation (square root), which ensures CPU-heavy workload with minimal memory access. The relative number of loop cycles between the parallel and sequential parts is determined by the input p value. Each parallel thread is pinned to a dedicated core using Linux core-affinity. The benchmark can accept $p = 1$ and run only the parallel section. In this case, the actual p value of the execution may not meet the requirement, but the benchmark will still display the highest parallelization achievable on the platform, which is not expected to be exceeded by any real application.

5 Parallelization-Aware Run-Time

Based on Sections 3 and 4, this section constructs an algorithm for parallelization-aware run-time management. Figure 4 illustrates the control cycle, which consists of the following steps:

- reading hardware performance counters and calculating current IPS value;
- determining the instantaneous value of p according to Section 3.3;
- finding the optimal operating point (n, F) for a given the optimization mode (EPI or EDP) using the lookup table as proposed in Section 4.2.

In order to calculate IPS, the method uses the number of instructions retired and the number of unhalted cycles,

1. https://github.com/ashurrafiev/PThreads

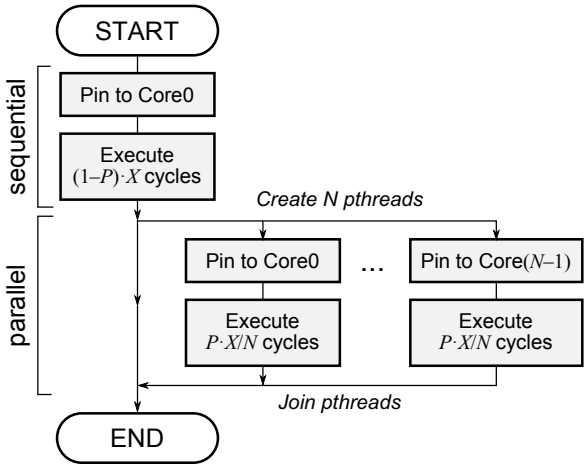


Fig. 3. Flowchart of the synthetic benchmark with programmable p, considering a total workload of X computation cycles.

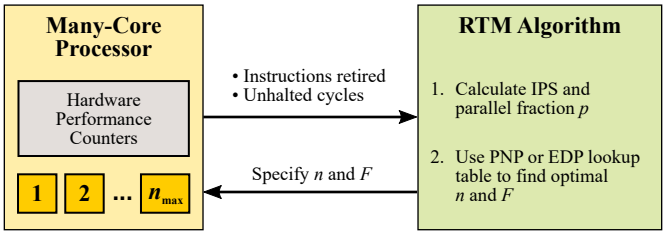


Fig. 4. The run-time management system.

which are usually available as performance counters in modern CPU's [10], [30], [31]. However, as noted in Section 3.1, the instantaneous value of IPC used by the models for IPS calculation includes both halted and unhalted cycles and should correspond to the wall-clock duration of the measurement window. The number of halted cycles is often not available as a performance counter and requires the following workaround. According to the model, the halted cycles appear when the parallel cores are waiting for the sequential execution to finish on one of the cores. This also means that at least one core is always busy executing both parallel and sequential parts and has no halted cycles. Hence, we can indirectly measure the correct total number of halted and unhalted cycles by using the maximum number of unhalted cycles across all cores running on the same frequency [10].

Section 3.3 proposed two methods for determining parallel fraction p, both having pros and cons:

Method 1: Equations (8) and (10) require the number of cores to differ between control cycles. The benefit of using this approach is that it accepts $n = 1$ and does not accumulate errors over time.

Method 2: Equation (12) can determine p over the periods of constant n, but it requires n 2 and can potentially accumulate errors because p is calculated in relation to its previous value. Any measurement uncertainty adds up to the subsequent measurements. Hence, even if the individual uncertainties are small, they may cause the estimated p to gradually drift away from its true value.

Consequently, each of the methods cannot be reliably

used on its own. Therefore, PARMA run-time employs a combination of both methods in a way that maximizes the advantages of both methods while cancelling their drawbacks. Here, Method 2 is used during the periods of stable p, when the optimal number of cores stays the same. Method 1 is used when there is a significant change in behavior leading to a different n, at which point any potential errors accumulated by Method 2 get reset.

A noteworthy constraint of the PARMA algorithm is that $n = 1$ cannot be used even if it is the optimal operating point (for example, during a purely sequential execution). This makes sense since the behavior of an application constrained to a single core does not exhibit parallelization properties that could be monitored by the run-time. This potential overhead is taken into consideration during the experimental validation of the run-time.

Stability and convergence are important properties for any control scheme [32]. Here we need to explore the conditions under which PARMA may display oscillatory behavior, similar to what was done in [25].

PARMA assumes that the determined p is valid for the next control cycle. The same assumption is also true for IPC. In the worst case when the workload changes its p faster than the control cycle, the PARMA approach may produce suboptimal results. Specifically, if the workload's p oscillates between a high value and a low value in step with the control cycle, oscillations may occur because a high p and a low p require very different control actions. This type of oscillatory behavior is easy to detect at run-time and may be remedied by reducing the control cycle's length away from its current value.

In theory, the control cycle cannot be infinitely reduced because the control actuators, thread-to-core mapping or context switching (tuning n) and DVFS (tuning F), are not cost-free and have inherent minimum latencies. It also needs to include enough time for the control decision computation. This means that PARMA's usefulness is limited by the rate of change of a workload's p in relation to these latencies. An appropriate control cycle length needs to be much greater than the actuator latencies in order for the latter to be relatively negligible and long enough to allow control computation to complete, but also needs to be small enough to satisfy the Nyquist-Shannon sampling theorem [33] with regard to the rate of p change in the workload (i.e. cycling at least twice as fast) for correct control.

DVFS transitions take no more than tens of μs [34] and context switching requires less time [35] in modern systems. On the other hand, empirical evidence shows that even high-variance workloads (Figure 1) consist of relatively long phases of stable p, with any full cycling between high and low values taking multiple seconds. The difference between actuator latencies and workload p cycle times is as much as 5 orders of magnitude. In our experiments the control cycle is set to $100ms$ which is 3 orders of magnitude greater than the context switching and DVFS transition latencies and more than 1 order of magnitude smaller than workload p change latencies. This means that the actuator time overheads can be safely ignored on the one hand and the Nyquist-Shannon requirement is more than satisfied on the other. The conservative range of appropriate control cycle durations spans more than 2 orders of magnitude, between several ms (more than long enough for the low-overhead lookup-table control decision making) and hundreds of ms. This range may be used for trading off the energy overhead of PARMA with its control precision and responsiveness.

Even if PARMA runs out of space for reducing its control cycle, sympathetic oscillations can still be avoided by increasing the control cycle from its current value. The resulting control will not be optimal in that case as the control decision will be based on an average p whose relevance is unknown in the next control cycle. This is a general problem for any control scheme if actuators and control decisions cannot change as fast as the plant being controlled, making Nyquist-Shannon impossible to satisfy. This, however, rarely happens when the objectives of control optimization consist of non-functional properties such as performance and energy [32], [25].

The proposed algorithm has been implemented and Section 6 validates it in an experimental setup.

6 Case Study

This section presents the experimental validation of the proposed method.

6.1 Experimental Platform and Characterization

The experiments are based on Core-i7-4820K Intel quad-core platform with 64 KB L1 cache, 256 KB L2 cache, and 10 MB L3 shared cache. In our experiments we disable hyper-threading and allocate tasks to physical (not logical) cores.

The platform supports full-domain DVFS, i.e. all cores operate on the same frequency. Table 2 presents the available voltage-frequency pairs. In the characterization experiments we only make use of 4 test points (as marked 1 to 4 in Table 2). In order to validate per-core DVFS, we build a simulation in Matlab based on the characteristics of this Intel platform. The simulation does not take into account the time overheads of switching the cores on and off, hence the simulation results are optimistic as discussed in Section 6.3.

Because of the requirement in Section 4.1, we convert voltage-frequency pairs into a differentiable function by building a linear regression model using the following hypothesis:

$$V = \gamma_0 + \gamma_1 \quad F, \tag{23}$$

where γ_0 is found to be 0.6619 and γ_1 equals to 0.1178. The models give R-squared 0.98 and SSE 0.004232.

The platform provides performance counters via the Linux Model-Specific Register (MSR) module. In this paper, we use the `likwid` tool to collect this information; `likwid` is a lightweight performance oriented tool suite for x86 multi-core processors [36]. The following `likwid` events are used in this work.

- `INSTR_RETIRED_ANY` counts the instructions leaving the retirement unit: these are the instructions that have been executed and their results are correct [31].
- `CPU_CLK_UNHALTED_CORE` counts the number of unhalted clocks, i.e. the number of recorded clock cycles while the core is not in a halt state. This number is not proportional to the elapsed time in the case of core halting and changing core frequency [30].

TABLE 2
Voltage Frequency Scaling Readings

Test point	F, GHz	V, V	W_{total}, W
1	3.7	1.1	49.683
	3.5	1.07	45.785
	3.3	1.05	42.263
	3.2	1.04	40.668
2	3.0	1.02	37.111
	2.8	0.99	34.209
	2.6	0.97	31.490
	2.4	0.94	29.075
	2.3	0.93	27.754
3	2.1	0.91	25.234
	1.9	0.89	23.198
	1.7	0.86	21.157
	1.6	0.85	20.492
	1.4	0.82	18.886
4	1.2	0.81	17.181

The number of halted cycles and the total number of cycles are not available directly in Intel platforms, as Intel focuses on unhalted clock for IPC calculations [30]. Section 5 describes the method of calculating the total number of cycles from the number of unhalted cycles on each core.

The characterization experiments are aimed at building the power profile of the platform and follow the methodology presented in Section 4.3. The synthetic benchmark (Figure 3) is used to produce the workload with a specific value of p. The monitoring of CPU energy has been achieved using `PWR_PKG_ENERGY` counter; [37] shows that this performance counter produces reliable results validated through direct measurements such as DC instrumentation.

We ran a total of 48 experiments for the range of $n \in \{1, 2, 3, 4\}$, DVFS test points 1 to 4 in Table 2, and $p \in \{0.1, 0.5, 0.9\}$. In each experiment, we measured the total power dissipation W_{total} and fitted the measurements to (15) to find platform constants A_0, A_1, and W_L. The average W_L is 10.349 W with standard deviation of 0.113 and R-squared 0.994. The background capacitance A_0 = 7.012 nF and the effective capacitance A_n = 2.406 nF.

The resulting platform models and optimal operating points are discussed in Section 6.3.

6.2 Benchmark Applications and Motivational Findings

The PARMA implementation has been validated using the PARSEC benchmark suite [38]. The PARSEC suite contains a set of reference applications from many fields, including industry and academia, aimed at studying concurrent applications on parallel hardware. Some of them are parallelized using OpenMP and the others are parallelized using POSIX threads.

In our initial experiments for determining the instantaneous values of p and IPS, we observed that some benchmarks display a very stable behavior with low variation of p value, while others have distinct phases with drastically different parallel fraction values. In this paper we present typical cases of each behavior class. Other benchmarks fall somewhere in between these two extremes and behave accordingly[2].

2. http://async.org.uk/data/PARMA2019/

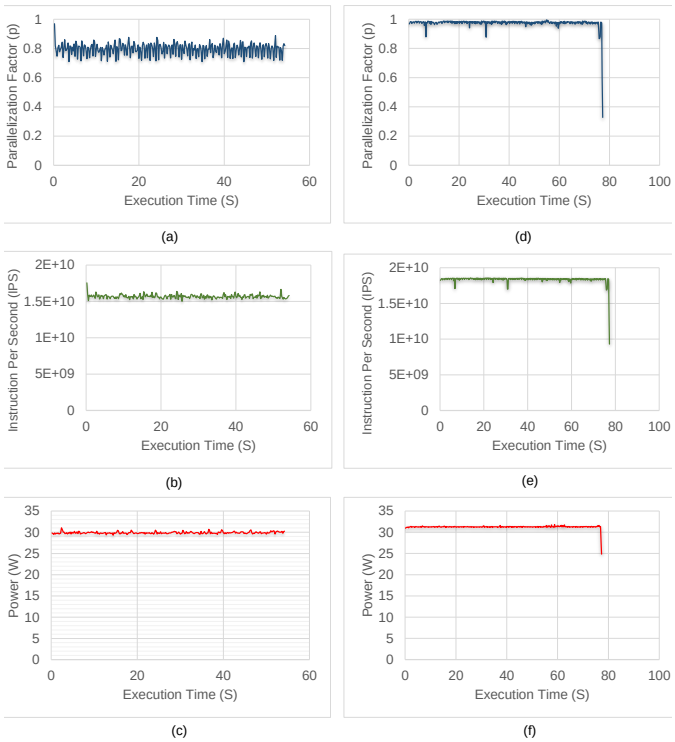


Fig. 5. Low p-variation benchmarks: a), b), c) `bodytrack`; d), e), f) `swaptions`.

For the low-variation cases, the selected benchmarks are `bodytrack` and `swaptions`. Figure 5(a),(b) and (c) illustrate the low changes in p, IPS calculation and power consumption respectively during execution time for `bodytrack`. Figure 5(d), (e) and (f) show the low variation in p, IPS and power consumption during execution time respectively for both `swaptions` benchmark.

For the high-variation cases, the selected benchmarks are `canneal` and `blackscholes`. Figure 1(a),(b) and (c) illustrate the low changes in p, IPS calculation and power consumption respectively during execution time for `canneal`. Figure 1(d), (e) and (f) show the low variation in p, IPS and power consumption during execution time respectively for both `blackscholes` benchmark.

6.3 Results and Discussions

This section provides offline analysis of the obtained platform models and presents the results of using these models in run-time optimization.

6.3.1 *Modelling and Optimization Results*

The experimentally obtained platform characteristics are applied to (20) to obtain the EPI and EDP models shown in Figures 6 and 7. The presented values are calculated for $IPC(1) = 0.411$ corresponding to the synthetic benchmark. Throughout the experiments we observe $IPC(1)$ in the range of 0.2 to 2.0 across various applications. The value of $IPC(1)$ is shown to scale the EPI and EDP graphs but not affect the locations of minima, as predicted by the models.

Figure 8 shows the optimal operating points (n, F). These points are used in the PARMA control, and the

Fig. 6. EPI for synthetic application: (a) with high parallelization p = 0.9 and (b) low p = 0 :1.

Fig. 7. EDP for synthetic application: (a) with high parallelization p = 0.9 and (b) low p = 0 :1.

respective run-time results are discussed in Section 6.3.2. To investigate further trends, we increase n_{max} to 64 cores and show the results of the same optimization in Figure 9 under the assumption that A_0, A_1, and W_L stay the same.

For the full-domain DVFS models, the optimal operation is always at the maximum number of cores. An interesting point is $p = 0$ where the suggested n is 1. For $p = 0$, this is not a unique optimal point: the minimum values of EPI and EDP are the same for all n for any given frequency, hence the maximum number of cores can also be used for $p = 0$. This choice can be explained by not being able to shut down individual cores in full-domain platforms, hence the total power is dominated by background and leakage powers regardless of the number of active cores.

The optimal F for full-domain DVFS depends on the optimization metric. EDP requires maximum F, but EPI results depend on p. As can be observed from Figures 6(a) and 7(a), with large p and maximum n, the metric graphs are almost flat with regard to F. The optimal frequency point does not lead to large improvements in the metrics.

In the case of per-core DVFS, EPI control is uncomplicated and always requires minimum n and minimum F regardless of p. Since EPI is heavily biased towards energy optimization, it is reasonable to choose the smallest number of cores and the lowest frequency. On the other hand, EDP optimization with per-core DVFS is equally concerned about performance and energy; it requires different n for different p values, but the frequency is always set to the maximum.

6.3.2 Run-Time Optimization Results

The PARMA algorithm presented in Section 5 is imlemented on the platform and evaluated with PARSEC benchmarks: *bodytrack* and *swaptions* representing low p-variation, and *blacksholes* and *canneal* representing high p-variation workloads. These experiments provide results for full-domain DVFS control. Since per-core DVFS is not available on the platform, we evaluated it using Matlab simulations. The simulations use the traces of actual benchmark executions to feed the performance counter data, but also consider the effect of switching off individual cores.

The baseline for comparison is the default Linux *ondemand* governor. The results are shown in Figures 10 and 11. The graphs are normalized to *ondemand*, and the tables show actual metric values.

The EPI results show that PARMA run-time is better suitable for optimizing this metric than *ondemand*. For full-domain DVFS, PARMA shows up to 35% improvement; for per-core DVFS, the improvement is up to 48%.

When optimizing EDP under full-domain DVFS for *bodytrack* and *swaptions* benchmarks, their high and stable paralellizability requires high numbers of cores and high frequency, which are provided by both *ondemand* and PARMA controls. Hence both run-times show equally good results. However, PARMA shows better reaction to variability in parallel behavior and improved over *ondemand* for *blacksholes* and *canneal* benchmarks.

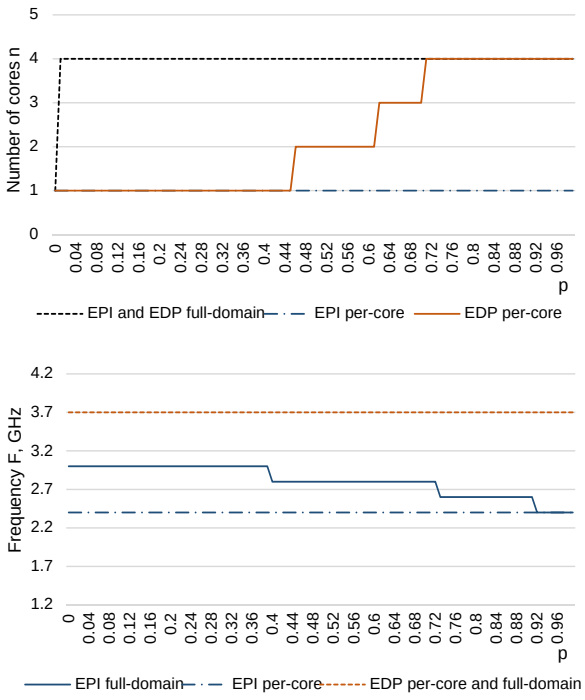


Fig. 8. Optimal number of cores n and frequency F for the experimental platform in the range of parallel fractions $0 \leq p < 1$. Different lines represent different optimization targets.

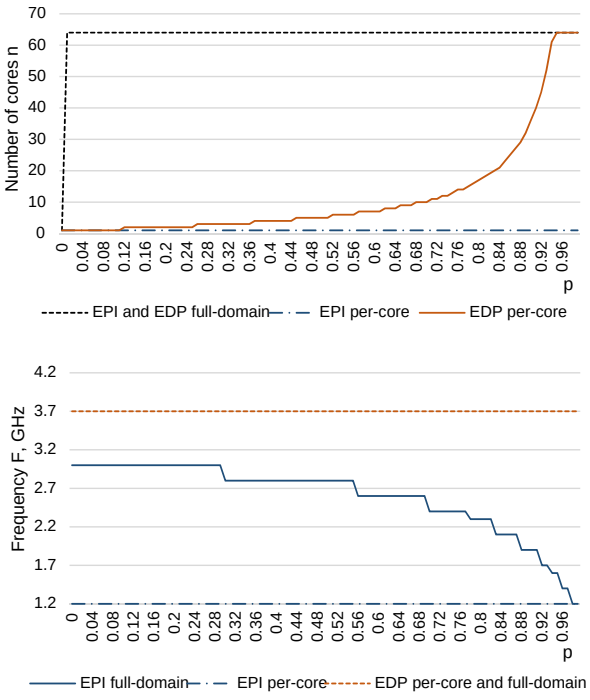
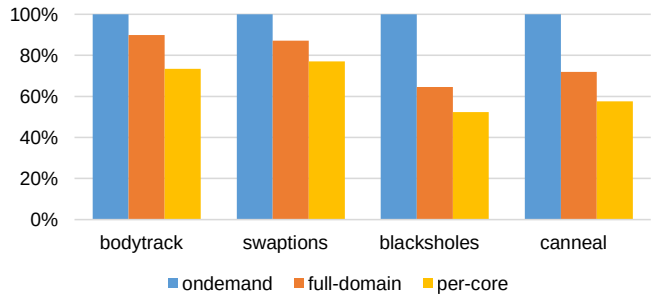
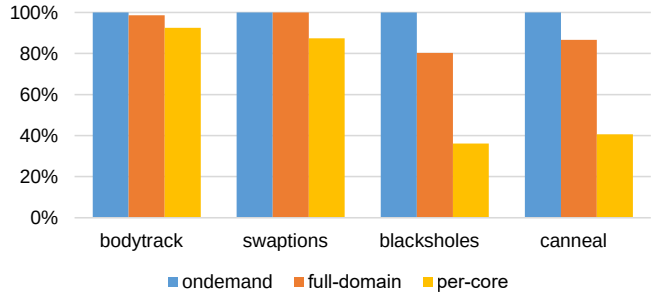


Fig. 9. Optimal number of cores n and frequency F for the case of $n_{max} = 64$ in the range of parallel fractions $0 \leq p < 1$. Different lines represent different optimization targets.



application	EPI, J		
	ondemand	full-domain	per-core(*)
bodytrack	2.12E-09	1.90E-09	1.56E-09
swaptions	1.95E-09	1.69E-09	1.50E-09
blackscholes	3.13E-09	2.02E-09	1.63E-09
canneal	2.03E-08	1.46E-08	1.17E-08

Fig. 10. Experimental and simulation(*) results for EPI minimization in run-time.



application	EDP, J s		
	ondemand	full-domain	per-core(*)
bodytrack	9.67E-20	9.54E-20	8.95E-20
swaptions	6.85E-20	6.85E-20	5.98E-20
blackscholes	2.42E-19	1.94E-19	8.74E-20
canneal	1.01E-17	8.74E-18	4.10E-18

Fig. 11. Experimental and simulation(*) results for EDP minimization in run-time.

Overall, EDP optimization is improved by up to 20% in full-domain and up to 64% in per-core DVFS modes.

It is important to note that our per-core DVFS simulations do not take into account the overheads of switching the cores on and off, hence the results are optimistic. On the other hand, the PARMA control requires the minimum of two cores to be active while some optimal points indicate $n = 1$. This means that the system is sometimes running sub-optimally, and this effect is taken into account in the results.

6.4 Future Work

This paper presents a run-time algorithm based on monitoring the parallel behavior of an application. It is strongly tied to Amdahl's Law and uses the parallel fraction p as the key determinant of the application properties.

Amdahl's Law is often criticized to be an idealistic over-approximation mainly because it views the execution as either parallel or sequential and also because it assumes that the parallel fraction is infinitely parallelizable. A more

precise approach is to represent the workload with a task dependency graph and, assuming that independent tasks can be executed in parallel, calculate the instantaneous value of *parallelism* [39]. Parallelism is defined as the ratio of the workload execution time on one core to the workload execution time on an infinite number of cores [40] and can be understood as the maximum number of parallel threads.

In theory, the parallelism approach is capable of producing better parallelization-aware run-time control; however, it has associated practical difficulties, for instance, determining the application task graph. [39] describes a practical method for Android devices, but it still implies pre-characterizing each individual application.

The presented version of PARMA improves the classical Amdahl's model by introducing the notion of variable instantaneous p, which brings it closer to the concept of parallelism. The mathematical foundation of the method makes the models application-independent as long as the value of p can be determined during run-time. The experiments confirm the efficacy of the PARMA approach, but we believe that there is still a possibility for further improvement.

The plans for future work include properly linking the notion of parallelism with the run-time monitoring of parallel behavior. This way it may be possible to benefit from non-Amdahlian parallelism-based models without requiring pre-determined application-specific task graphs, which should improve the optimization result while still maintaining the practicality of the method.

7 Conclusions and Discussions

The parallelizability of workloads is an important characteristic in the era of multi-/many-core processing. One of the main methods of quantitatively describing workload parallelizability is the parallel fraction p from Amdahl's Law and related models. Making use of this parameter to derive the optimal run-time control of systems has, however, not been attempted systematically until now. This paper describes the PARMA method and its supporting techniques whereby p is the central parameter in the run-time optimization of energy/performance tradeoffs.

A new practical method of determining p at run-time using hardware performance counters is presented. Based on Amdahl's workload model and our existing offline modelling methods, the run-time acquisition of p values supplies enabling information for run-time optimization.

Parallelization-aware power models for many-core processors are presented. These take into account both domain-wide and per-core DVFS assumptions for current relevance and future usability. The models enable the optimization of any weighted-product energy/performance tradeoff metrics, exemplified by EPI and EDP in the analyses included in this paper.

An implementation of PARMA run-time is presented in this paper and its efficacy is validated through a set of experiments using a number of benchmark applications, on an off-the-shelf multi-core system. The PARMA method and its supporting techniques enable the run-time optimization of any metric which has a relation to workload parallelizability and can be monitored through the use of hardware performance counters.

References

[1] J. Koomey, S. Berard, M. Sanchez, and H. Wong, "Implications of historical trends in the electrical efficiency of computing," *IEEE Annals of the History of Computing*, vol. 33, no. 3, pp. 46–54, March 2011.

[2] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," *IEEE Solid-State Circuits Society Newsletter*, vol. 12, no. 3, pp. 19–20, Summer 2007.

[3] X.-H. Sun and Y. Chen, "Reevaluating amdahl's law in the multicore era," *J. Parallel Distrib. Comput.*, vol. 70, no. 2, pp. 183–188, Feb. 2010.

[4] B. R. Rau and J. A. Fisher, "Instruction-level parallelism," in *Encyclopedia of Computer Science*. John Wiley and Sons Ltd., 2003, pp. 883–887.

[5] J. L. Lo, J. S. Emer, H. M. Levy, R. L. Stamm, D. M. Tullsen, and S. J. Eggers, "Converting thread-level parallelism to instruction-level parallelism via simultaneous multithreading," *ACM Trans. Comput. Syst.*, vol. 15, no. 3, pp. 322–354, Aug. 1997. [Online]. Available: http://doi.acm.org/10.1145/263326.263382

[6] S. Sridharan, G. Gupta, and G. S. Sohi, "Adaptive, efficient, parallel execution of parallel programs," *ACM SIGPLAN Notices*, vol. 49, no. 6, pp. 169–180, 2014.

[7] F. Xia, A. Rafiev, A. Aalsaud, M. Al-Hayanni, J. Davis, J. Levine, A. Mokhov, A. Romanovsky, R. Shafik, A. Yakovlev, and S. Yang, "Voltage, throughput, power, reliability, and multicore scaling," *Computer*, vol. 50, no. 8, pp. 34–45, 2017.

[8] Y.-H. Chen, Y.-L. Tang, Y.-Y. Liu, A. C.-H. Wu, and T. Hwang, "A novel cache-utilization based dynamic voltage frequency scaling (dvfs) mechanism for reliability enhancements," in *Proceedings of the 2016 Conference on Design, Automation & Test in Europe*, ser. DATE '16. San Jose, CA, USA: EDA Consortium, 2016, pp. 79–84.

[9] L. K. Goh, B. Veeravalli, and S. Viswanathan, "Design of fast and efficient energy-aware gradient-based scheduling algorithms heterogeneous embedded multiprocessor systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 20, no. 1, pp. 1–12, Jan 2009.

[10] M. Al-hayanni, R. Shafik, A. Rafiev, F. Xia, and A. Yakovlev, "Speedup and parallelization models for energy-efficient many-core systems using performance counters," in *The 2017 International Conference on High Performance Computing & Simulation*, 2017.

[11] J. Luo and N. K. Jha, "Power-efficient scheduling for heterogeneous distributed real-time embedded systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 6, pp. 1161–1170, June 2007.

[12] M. Goraczko, J. Liu, D. Lymberopoulos, S. Matic, B. Priyantha, and F. Zhao, "Energy-optimal software partitioning in heterogeneous multiprocessor embedded systems," in *Proceedings of the 45th annual design automation conference*. ACM, 2008, pp. 191–196.

[13] H. Wong and T. M. Aamodt, "The performance potential for single application heterogeneous systems," in *8th Workshop on Duplicating, Deconstructing, and Debunking*, 2009.

[14] J. Nunez-Yanez and A. Beldachi, "Run-time power and performance scaling with cpu-fpga hybrids," in *2014 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, July 2014, pp. 55–60.

[15] R. Joseph and M. Martonosi, "Run-time power estimation in high performance microprocessors," in *Proceedings of the 2001 International Symposium on Low Power Electronics and Design*, ser. ISLPED '01. New York, NY, USA: ACM, 2001, pp. 135–140.

[16] K. Rajamani, H. Hanson, J. Rubio, S. Ghiasi, and F. Rawson, "Application-aware power management," in *Workload Characterization, 2006 IEEE International Symposium on*. IEEE, 2006, pp. 39–48.

[17] K. K. Pusukuri, R. Gupta, and L. N. Bhuyan, "Thread reinforcer: Dynamically determining number of threads via os level monitoring," in *2011 IEEE International Symposium on Workload Characterization (IISWC)*, Nov 2011, pp. 116–125.

[18] C. Isci, A. Buyuktosunoglu, and M. Martonosi, "Long-term workload phases: duration predictions and applications to dvfs," *IEEE Micro*, vol. 25, no. 5, pp. 39–51, Sep. 2005.

[19] P. Mercati, R. Ayoub, M. Kishinevsky, E. Samson, M. Beuchat, F. Paterna, and T. . Rosing, "Multi-variable dynamic power management for the gpu subsystem," in *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, June 2017, pp. 1–6.

[20] H. Sasaki, S. Imamura, and K. Inoue, "Coordinated power-performance optimization in manycores," in *Proceedings of the 22nd*