Al-Hayanni M, Shafik R, Rafiev A, Xia F, Yakovlev A.

Speedup and Parallelization Models for Energy-Efficient Many-Core Systems Using Performance Counters.

***In: 2017 International Conference on High Performance Computing & Simulation*. 2017, Genoa, Italy: IEEE.**

DOI link to article:

https://doi.org/10.1109/HPCS.2017.68

Date deposited:

18/07/2017

Speedup and Parallelization Models for Energy-Efficient Many-Core Systems Using Performance Counters

Mohammed A. N. Al-hayanni[1], Rishad Shafik[1], Ashur Rafiev[2], Fei Xia[1], Alex Yakovlev[1]
School of EEE[1] and CS[2], Newcastle University
Newcastle Upon Tyne, NE1 7RU, UK
E-mail: m.a.n.al-hayanni, rishad.shafik, ashur.rafiev, fei.xia, alex.yakovlev@ncl.ac.uk

***Abstract*—Traditional speedup models, such as Amdahls, facilitate the study of the impact of running parallel workloads on manycore systems. However, these models are typically based on software characteristics, assuming ideal hardware behaviors. As such, the applicability of these models for energy and/or performance-driven system optimization is limited by two factors. Firstly, speedup cannot be measured without instrumenting the original software codes, and secondly, the parallelization factor of an application running on specific hardware is generally unknown.**

In this paper, we propose a novel method, whereby standard performance counters found in modern many-core platforms can be used to derive speedup without instrumenting applications for time measurements. We postulate that speedup can be accurately estimated as a ratio of instructions per cycle for a parallel manycore system to the instructions per cycle of a single core system. By studying the application instructions and system instructions for the first time, our method leads to the determination of the parallelization factor and the optimal system configuration for energy and/or performance. The method is extensively demonstrated through experiments on three different platforms with core numbers ranging from 4 to 61, running parallel benchmark applications (including synthetic and PARSEC benchmarks) on Linux operating system. Speedup and parallelization estimations using our method and their extensive cross-validations show negligible errors (up to 8%) in these systems. Additionally, we demonstrate the effectiveness of our method to explore parallelization-aware energy-efficient system configurations for many-core systems using energy-delay-product based formulations.

***Index Terms*—Many-core processors; speedup; performance counter, power normalized performance, energy-delay-product.**

I. Introduction

Aggressive technology scaling has facilitated significant reductions in device geometries and hence circuit delay, leading to performance improvement [1]. According to Moore's and Koomey's laws the trend of scaling has led to doubling of performance per watt every 1.5 years [2], [3].

Many studies have been undertaken to date realize the trend of performance growth with many CPU cores. For instance Pollack's rule suggests that performance is increasing approximately proportional to the square root of the complexity defined by the power density per unit area [4]. According to this rule, doubling the number of processors also doubles the performance [1]. As such, multi- or many-core systems are expected to deliver further improvement in throughput and latency for the same die area. However, such performance growth is also being inhibited by high performance densities typically seen in modern technology nodes, leading to the concept of dark silicon [5]. Hence, energy-efficient resource allocation is of crucial importance for many-core systems with high-performance requirements [3].

Speedup models are popular methods to reason technology-independent normalized performance improvements with many-cores. In 1967 the first speedup model was proposed by Amdahl [6], which described how the performance of fixed workloads can be estimated when executed on N processors. Many studies followed the idea of Amdahl's model to extend performance reasoning for parallel workloads executed on multi- or many-cores systems. In [7] Hill and Marty complemented Amdahl's model to define performance expectations of systems with heterogeneous and dynamic configurations via Pollack rules [4]. In [8] these performance models were generalized for homogeneous system configurations. Both of these models were elaborated further by [9] to investigate the theoretical multi-core scalability and to determine the optimal multi-core performance.

In [10] performance models were defined as a function of the system architecture. Detailed account for the additional uncore theoretical components was proposed, including contributions from on-chip interconnect, pipeline and cache memory subsystems. The effect of memory was also highlighted by [11], which showed how performance speedup can be capped by shared memories in a homogeneous many-core system. The power contributions of different architectural components for various system configurations were elaborated by [12]. The power models demonstrated the trade-offs between speedup and relevant power consumptions.

Table I shows summary of the existing studies relevant to speedup models. As can be seen, these models have the following limitations. Firstly, the ideal assumptions on both the workload and the system platform limit the applicability of the models for real systems. For instance, they do not consider real system overheads, such as scheduling and hardware synchronizations. Secondly, they do not use real benchmark applications to estimate the system performance. Thirdly, they do not highlight determination of application parallelization factors and their impacts on the speedup models; instead they consider theoretical limits of parallelization factors. As such, using speedup and parallelization models to identify suitable core allocations remains challenging for energy efficiency considerations. To address these limitations, this paper makes the following ***contributions***:

- Extend Amdahl's speedup model considering applications and system software related overhead separately.
- Propose a new method to model parallelization and speedup via performance counters to avoid the need for instrumenting applications. We show that speedup can be accurately estimated as a ratio of instructions retired/executed per cycle of

TABLE I: Existing Speedup Models and the proposed model.

Previous study	Amdahl's Model	System overhead	Performance counter	Real Benchmark	Parallelization factor	Power
[6]–[11]	Yes	No	No	No	user control	No
[12]	Yes	No	No	No	user control	Yes
[13], [14]	No	Yes	No	Yes	No	No
[15]	Yes	Yes	No	No	No	No
[16]	similar principle	Yes	No	Yes	No	Yes
[17]	related	Yes	No	Yes	No	No
[18]	similar principle	Yes	No	Yes	estimated	No
[19]	Yes	Yes	No	Yes	estimated	Yes
Proposed Model	Yes	Yes	Yes	Yes	user control and estimated	Yes

parallel many-core system to that of a single core system.

- Extensive analysis of synthetic and real (PARSEC) benchmarks to validate the speedup and parallelization factors based on our proposed model.
- Demonstrate the effectiveness of our method for identifying parallelization-aware energy-efficient system configurations using power normalized performance and energy-delay-product metrics.

The rest of the paper is organized as follows. Section II gives the background on our proposed models. Section III describes the proposed speedup model, together with power and energy related metrics. Section IV describes our experimental set up, the performance counters and the benchmark application used. Section V presents the cross-validation of the models with measured speedups and applicability of our method. Section VI proposes the new paradigm of parallelization-aware energy-efficient computing. Finally, Section VII concludes the paper.

II. Background

We consider a system consisting of N cores, and a workload with a parallel part and a sequential part. The fraction of parallel workload is P; hence, the sequential part of workload is $(1-P)$. The P value known as the parallelization factor varies from 0 to 1; 0 indicates all sequential workload and 1 indicates fully parallelizable workload. In our study we use a synthetic benchmark application [20], whereby the parallelization P and core allocations (from 1 to N) can be controlled.

Amdahl's speedup model calculations are based on a comparison of execution time for a fixed workload I_0 on a single core with the execution time for the same workload executed on the entire N-core system. Time $T(1)$ is needed to execute both the sequential and parallel parts of workload I_0 on a single core, and $T(N)$ is needed to execute the sequential part of workload I_0 on a single core and the parallel part on N cores [21], [22].

$$T(1) = \frac{I_0}{IPS_{I_0}} \quad \text{, and} \tag{1}$$

$$T(N) = \frac{(1-P)\cdot I_0}{IPS_{I_0}} + \frac{P\cdot I_0}{N\cdot IPS_{I_0}} \quad , \tag{2}$$

where IPS_{I_0} is the throughput measured in instructions per second for the workload I_0 on a single core. Thus IPS based Amdahl's speedup model can be derived as follows [6].

$$SP(N) = \frac{T(1)}{T(N)} = \frac{1}{(1-P)+\frac{P}{N}} \quad . \tag{3}$$

According to [22], this speedup can also be define as:

$$SP = \frac{IPS_{I_0(N)}}{IPS_{I_0}} \quad . \tag{4}$$

In other words, speedup is also the ratio of the throughput achieved by executing on N cores to the throughput achieved by executing on a single core [21].

III. Proposed Speedup Models

Amdahl's speedup model assumes that the workload has a single P. In real systems, the overall workload usually consists of multiple tasks, including system software (e.g. OS), and each task may exhibit a different parallelizability and therefore different P. One potential method of run-time management is the parallelizability-aware optimization of performance and/or energy. For that it is important to treat individual applications and the system software separately. In this section we develop a new speedup model that calculates application speedup and consider realistic system software overhead separately.

In the rest of the paper, we deal with the case of a single application running on a real system with system software at the same time. Expanding to multiple concurrently running applications will be a future task.

A. Modeling Basics

We consider that the overall workload is the number of total instructions executed by the system during the execution of any specific application. The total number of instructions I when a specific application is executed includes the application instructions I_0 plus the system software instructions and halt cycles caused due to resource sharing, ΔI, given as:

$$I = I_0 + \Delta I \quad . \tag{5}$$

If the number of instructions I_0 in (5) can be obtained, in order to calculate speedup we need to find out IPS. In other words, in addition to the number of instructions, we need to know the time spent on executing these instructions, which usually implies instrumenting both applications and system software for time monitoring. On the other hand, instructions per clock IPC does not need the monitoring of time and only requires counting the number of clock cycles spent in the execution. In the next section we will explain how to obtain the relevant clock cycle, with which the IPC can be calculated as follows:

$$IPC = IPC_{I_0} + IPC_{\Delta I} = \frac{I_0}{\mathcal{C}} + \frac{\Delta I}{\mathcal{C}} \quad , \tag{6}$$

where $\mathcal{C}$ is the number of clock cycles spent on the execution. In a many-core system the estimation of effective IPC for a parallel workload given by (6) can be challenging as the instructions retired per core against their corresponding execution cycles cannot be used to estimate an overall average IPC. This is because some cores execute parallel workloads independent of of the other cores, while the core that is in charge of spawning threads

executes mostly sequential, but also some parallel workloads. The execution of a workload therefore causes participating cores to record different numbers of clock cycles. We hypothesize that $\mathcal{C}_{max}$ (recorded from the core with the highest unhalted clock $\mathcal{C}$ value among all cores), generally gives a good indication of the overlapped parallel execution times, measured by the time-stamp counter. As such, the effective IPC in (6) can be defined as:

$$IPC = \frac{I_0}{\mathcal{C}_{Max}} + \frac{\Delta I}{\mathcal{C}_{Max}} \quad , \tag{7}$$

Our experiments in Sections (IV) and (V) will show that (7) can be used with confidence to model speedup. The resulting throughput expressed by IPS as follows:

$$IPS_{I_0} = IPC_{I_0} \cdot F \quad , \tag{8}$$

where F is the system operating frequency. This supports the calculations of sequential and parallel execution time in (1) and (2).

B. Speedup Calculations

For speedup, we can substitute IPC for IPS as the system frequency is eliminated from the equation in case of the system running at the same frequency, i.e.

$$SP = \frac{IPS_{I_0(N)}}{IPS_{I_0}} = \frac{IPC_{I_0(N)}}{IPC_{I_0}} \quad , \tag{9}$$

where IPC_{I_0} and IPS_{I_0} are the instruction per clock and instruction per second, respectively in single core with full sequential workload, and $IPC_{I_0(N)}$ and $IPS_{I_0(N)}$ are the instruction per clock and instruction per second, respectively for given P and N configurations of a parallel application on a many-core system.

C. Estimation of Parallelization Factor

Once the speedup of an application is known through (9) it can be used to calculate the parallelization factor P from equation (3) as:

$$P = \frac{N \cdot (1 - SP)}{SP \cdot (1 - N)} \quad . \tag{10}$$

This expression is used in Section (V-C) to calculate parallelization using performance counters. Note that the calculation for (N $>$ 1) give negative numerator and denominator, thus it gives positive parallelization value.

D. Power and Energy Normalized Performance

Power normalized performance is an established metric related to the power efficiency of systems. It is simple to model the performance achievable at the same cooling capacity by calculating performance per watt ($Perf/Watt$) [12], [21]. Power normalized performance can be calculated from dividing the system performance from (9) by the total power (W_{total}):

$$Perf/Watt = \frac{IPS_{I_0(N)}}{W_{total}} \quad . \tag{11}$$

Power normalized performance model is the reciprocal of energy per instruction (EPI_N) because performance is the reciprocal of execution time [12], [21]. Thus, EPI_N can be calculated from dividing the total power (W_{total}) by the system's performance (9):

$$EPI_N = \frac{W_{total}}{IPS_{I_0(N)}} \quad . \tag{12}$$

As metrics, EPI and power normalized performance can be limiting. For instance, if an execution progresses extremely slowly but consumes very little energy, it can result in good EPI and power normalized performance numbers because it consumes almost zero power. On the other hand, it may not get anything useful done. In effect, EPI and power normalized performance promote the minimization of energy but does not care much about performance. To capture this concern the metric known as energy-delay-product (EDP) [23] puts more emphasis on the completion of tasks by explicitly incorporating delay. In our method, EDP can be obtained from (11) and (2) as follows:

$$EDP = W_{total} \cdot (\frac{(1-P) \cdot I_0}{IPS_{I_0}} + \frac{P \cdot I_0}{N \cdot IPS_{I_0}})^2. \tag{13}$$

IV. Experimantal studies

The models on speedup, parallelization, power and energy metrics are demonstrated in this section through experiments.

A. Experimental Platforms

In this work we make use of three different Intel platforms. Table II explains the general architecture details of all platforms. All of these systems additionally allow hyper-threading. In all our experiments we disabled hyper-threading by allocating tasks to physical (not logical) cores. Although these systems are from Intel, other platforms such as those from ARM also provide similar performance counters which supports the generality of this work. Extending this investigation to other platforms will be part of our future work.

TABLE II: Experimental platforms used in this work.

Parameters	Intel CPU Type		
Processor Name	Core i7	Xeon	Xeon Phi
Processor No.	i7-4820k	E5-2630V2	7120X
Lithography	22 nm	22 nm	22 nm
No. of Sockets	1	2	1
Cores per Socket	4	6	61
No. of Cores	4	12	61
L1D Unified Cache	32 KB	32 KB	32 KB
L1I Unified Cache	32 KB	32 KB	32 KB
L2 Unified Cache	256 KB	256 KB	512 KB
L3 Shared Cache	10 MB	15 MB	-
Base Frequency	3.7 MHz	2.60 MHz	1.24 MHz

B. Performance Counters

Hardware performance counters are a set of special purpose regisers built into CPUs to store the counts of hardware activites in a specific system. Users depend on those counters to collect low-level performance analysis. This performance data varies depending on the performance monitoring hardware and system software configuration. An interface to access model specific registers from user space is provided via the Linux Model-Specific Register (MSR) module. This allows the user to extract hardware performance counter events with an unmodified Linux kernel. $Likwid$, used in this paper, is a lightweight performance oriented tool suite for x86 multi-core processors [24], [25].

The following performance counters are used in this work.

$INSTR\_RETIRED\_ANY$ counts the instruction retired which leave the retirement unit. Such instructions have been executed and their results are correct [26].

$CPU\_CLK\_UNHALTED\_CORE$ counts the number of unhalted clocks while the core is not in a halt state. This

performance counter is obtained through clock cycle recording. If the clock frequency changes the number of cycles will not be proportional to time. And halted states also affect the accuracy of using this performance counter to represent time [27].

$CPU\_CLK\_UNHALTED\_REF$ counts the number of reference clocks at the Time Step Counter (TSC) rate, while the core is not in a halt state. This event is not affected by core frequency changes. It counts at the same frequency as the TSC [27].

In our model, we need both the number of instructions and the number of clock cycles for calculating IPC. For the number of instructions, we use $INSTR\_RETIRED\_ANY$ as application workload I_0 in equation (5). For the number of clock cycles we use $CPU\_CLK\_UNHALTED\_CORE$ as IPC_{I0} in equation (7) which represent the accurate perfromance counter to calculate IPC [27].

In Section (III) we showed that the number of cycles represents time. $CPU\_CLK\_UNHALTED\_REF$ shows the number of cycles which includes halted cycles, hence is closer to representing real execution time. In the real world, halted cycles occur when the system has nothing to run. This occurs when threads waite for interrupt thus the counting includes halted cycles in real execution time [28], [29]. However it is not always available in Intel, as Intel focuses on unhalted clock for IPC calculations [27]. In our experiments we explore the use of unhalted clock to calculate speedup, and the outcome is presented in Section (V).

In addition, hardware performance counters exist that provide power and energy information. For instance, PWR_PKG_ENERGY counts the CPU energy consumption [27]. In previous work, it has been shown that this performance counter produces reliable results validated through direct measurements such as DC instrumntation [30]. It is used in conjunction with the execution time information inferred from unhalted clock performance counters in Section (VI-A) to derive all power and energy information. In this paper we focus on CPU energy which changes with the number of utilized cores and the parallelization factor P and disregard other energy consumption which have weak correlations with these factors (e.g. memory energy).

From (3), it is possible to calculate speedup if T(1) and T(N) can be obtained. However, this requires running a workload at least twice, with different core configurations. To avoid having to run a workload more than once, time-based calculations of P require the knowledge of sequential and parallel time, which requires instrumenting the code of a workload. By using the performance counters listed above, however, it may be possible to obtain the same functionality as instrumenting separate parallel and sequential time monitoring, whilst only needing to monitor the start and end of a workload. This means that there is no need to modify workloads in any way.

Even though we motivate our work to avoid time measurements, in our experiments we use time-instrumented workload code when possible as well as make pairs of runs with different numbers of cores. This helps demonstrate the validity of our approach of avoiding direct time measurements through comparisons.

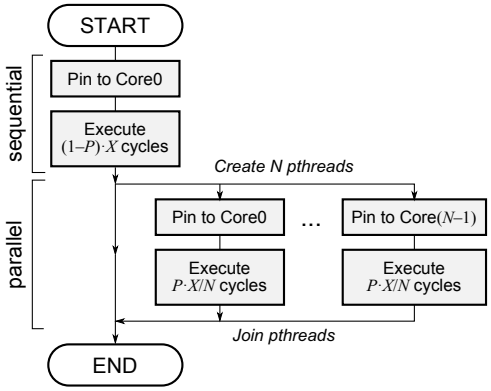

Fig. 1: Flowchart of the benchmark application with programmable P, considering a total workload of X cycles.

C. Benchmark Application

1) Synthetic Benchmark: The synthetic benchmark is executed in all three platforms as shown in Table II at the base frequencies.

The calculation of the theoretical speedup models heavily relies on the knowledge of the parallelization factor P. However, a typical application has an unknown and variable P. Hence, we developed a synthetic benchmark, which allows the control of its P value. The benchmark is available for free usage and research from [20].

The benchmark has distinct sequential and parallel sections, as shown in Fig. 1. Inside, it performs a looped arithmetic calculation (square root), which ensures a CPU-heavy workload with minimal memory access. The number of cycles in parallel and sequential parts is determined by the requested P value. The sequential part is pinned to Core 0, and the parallel code is evenly distributed between cores using core-affinities.

It is important to note that the benchmark can accept P=1 and run only the parallel section. However, the actual parallelization achieved by the platform may not meet the requirement. This will be carefully considered when analyzing experimental results.

2) PARSEC Benchmarks: PARSEC benchmarks are executed at base frequency on the Core-i7 platform only.

PARSEC [31], [32] is a reference application suite used in many fields including industry and academia, for studying concurrent applications on parallel hardware. Some of them parallelized with OpenMP, while the others parallelized with gcc-pthreads.

PARSEC consists of 12 applications representing a diverse set of commercial and emerging workloads [31].

In our study we choose 9 PARSEC benchmarks having different parralelizabilties and memory usage intensities, [31]–[33]. The input set used is "native" and the benchmarks chosen are bodytrack, blackscholes, facesim, fluidanimate, freqmine, swaptions, streamcluster, canneal and dedup.

V. Results and validation

This section describes the model calculations and experimental outcomes. We classify the calculations into fixed workload part and extra workload, demonstrate the validations of execution time and speedup and the estimation of parallelization factor P.

A. ΔI Calculation

In the first stage of our experiments, we use the Core i7 platform to find ΔI and I_0. The synthetic benchmark application was run

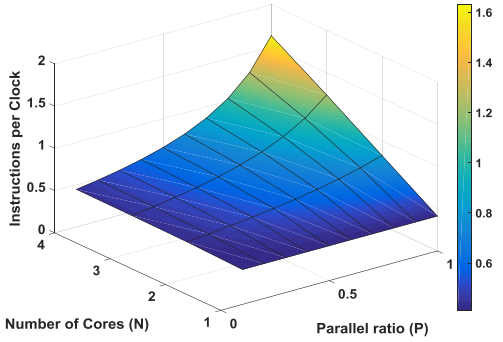

Fig. 2: Application instructions per clock for synthetic benchmark using variable N and P.

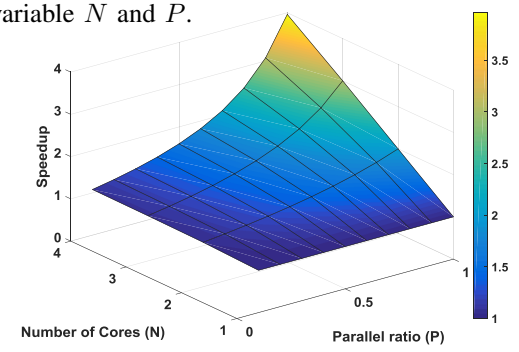

Fig. 3: Performance counter based speedup for synthetic benchmark using variable N and P.

on all core configurations (from N=1 to N=4) and programmed P ranging from (0 to 1). The first observation was that for all N=1 experiments Core 0 showed exactly the same number of instructions retired with no random variation, which is an indication that all system workloads have been scheduled on idle cores (cores that do not have applications running), and Core 0 has been exclusively running the application workload I_0, which is 5.6E+09.

Knowing I_0, we were able to calculate IPC_{I_0} from (6) and the speedup based on IPC_{I_0} from (9). The results are presented in Figures 2 and 3. Fig. 2 shows the throughput, in IPS, that is achieved with the application's programmed P value ranging from 0 to 1 and the number of cores ranging from 1 to 4. The maximum throughput is clearly achieved with P=1 and N=4. It is important to note that with a programmed P of 0 (i.e. non-parallelizable code) increasing the number of cores does not affect the throughput, and with a single core, no matter what the programmed P is the throughput is also constant. Fig. 3 shows the speedup as a function of N and P. It can be seen that the maximum speedup achievable with N=4 and P=1 is close to 4, which shows that the synthetic benchmark dose not have hidden synchronizations and other effects limiting parallelizability, and the hardware platform's impact on IPC-based speedup is small.

The second finding is that ΔI reduces with N and P increasing. We tested the hypothesis that system workload ΔI is proportional to time, and confirmed that $\Delta I/T(N)$ approximates to a constant with the average of 6.58E+04 and the standard deviation of 9.53E+03.

Also the system software workload is very small i.e. 1-2%. However, these extra instructions can cause resource constraints and result in halt cycle. In our experiments we have observed a 1.55% increase of halt cycles for P=0 and N=1.

In the second stage, we run PARSEC benchmarks; the applications run in all core configurations (from N = 1 to N = 4), in PARSEC we do not have programmed P. The first observation is that the total instruction retired have fixed values for each application, with small changes (<6%), the total instructions reduced with N increasing and execution time decreasing. Thus, we use linear regression to calculate fixed I_0 and variable ΔI, where ΔI is a function of number of cores N and execution time.

$$\Delta I = \alpha t + \beta N \tag{14}$$

TABLE III: System software workloads $\Delta I/T(N)$ for different PARSEC applications.

Name	Average	Standard Deviation
bodytrack	1.37E+09	4.15E+08
blackscholes	3.92E+08	1.47E+07
facesim	6.78E+08	2.33E+08
fluidanimate	6.66E+08	4.16E+08
freqmine	3.09E+08	4.36E+07
sweptions	3.12E+08	5.86E+07
streamcluster	4.88E+09	1.89E+09
canneal	3.44E+08	1.83E+07
dedup	4.79E+08	1.21E+08

It confirmed that $\Delta I/T(N)$ approximates to a constant in most cases as shown in Table III, Where the standard deviations tend to be much smaller than the averages. The benchmarks bodytrack, facesim, streamcluster, canneal and dedup have a small changes in ΔI rather than blacksholes, freqmine, sweptions and fluidanimate.

B. Time and Speedup Validation

Table IV shows the validation results of synthetic benchmark of the speedup estimated with performance counters using (9), against the traditionally used time measurements. From Table IV, two observations can be made. Firstly we validate the use of performance counters by comparing the measured execution time with the time calculated from (2) and (8) by using the programmed P and the measured IPC and I.

TABLE IV: Cross-validation results for fixed workload I_0 using synthetic benchmark [20].

			Time, ms			Speedup		
	P	N	*Measured*	*Calculated*	*Error %*	*Measured*	*Calculated*	*Error %*
Core-i7	0.1	2	3492	3492	0.01	1.05	1.052	0.05
	0.1	3	3430	3430	0.03	1.07	1.071	0.03
	0.1	4	3400	3400	0.01	1.08	1.081	0.02
	0.9	1	3675	3676	0.03	1	0.999	0.02
	0.9	3	1470	1570	0.03	2.5	2.501	0.09
	0.9	4	1205	1194	0.86	3.05	3.074	0.84
Xeon	0.1	1	521	534.171	2.47	1	1.000	0.00
	0.1	4	483	494.108	2.25	1.078	1.080	0.36
	0.1	12	474	485.205	2.31	1.099	1.100	0.01
	0.9	2	294	293.794	0.07	1.772	1.666	2.62
	0.9	8	115	113.511	1.31	4.530	3.331	3.83
	0.9	12	95	93.4799	1.63	5.484	3.747	4.11
Xeon Phi	0.1	8	29939	30540.26	1.97	1.118	1.118	2.06
	0.1	16	29744	30331.08	1.94	1.126	1.126	2.09
	0.1	61	30182	30176.76	0.02	1.109	1.108	0.05
	0.9	1	32853	33468.78	1.84	1.019	1.019	1.91
	0.9	4	10655	10877.35	2.04	3.143	3.145	2.23
	0.9	32	4372	4288.18	1.95	7.660	7.848	0.55

We then validate the use of performance counters for speedup estimation by comparing the measured speedup, as the execution

time ratio $T(1)/T(N)$, to the IPC-based speedup calculated from the performance counters according to (9).

In these experiments, the errors are generally small; however they increase to nearly 8% when P=1. This is expected as the programmed P value does not correspond to the real parallelization factor in the platforms. The observation is that real platforms cannot keep up with the programmed parallelization, presumably due to extra interactions between components.

However, the speedup based on unhalted clock calculation of IPC matches the theoretical speedup from Amdahl's Law (3) with virtually no error (<0.5%). This result can be found in the full set of data and calculations [20]. It indicates that the discrepancy between the measured speedup and the unhalted clock-based speedup is due to halting of the cores. Additionally, this property can be exploited to estimate the application software P value as discussed in Section (V-C).

For PARSEC benchmarks, we run the nine PARSEC applications on the Intel Core-i7 platform at base frequency (3.7 GHz). We collect the appropriate hardware performance counters in Section (IV-B), thus the speedup can be calculated by (9). Fig. 4 shows the speedup calculations for these benchmarks, the performance counter based speedup calculations show a good cross-validation with general execution time based speedup calculations. The error ratio does not exceed 6.5%. Finally, The parallelization factor P can be calculated as explained in Section (V-C).

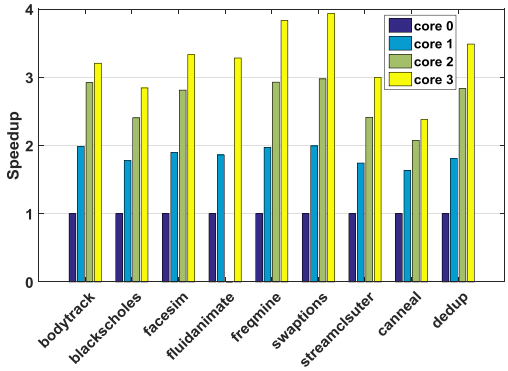

Fig. 4: Performance counter based speedup for PARSEC benchmark applications.

C. Estimating Parallelization Factor P

The effectiveness of scaling to more cores in order to obtain more speedup is related to the value of the parallelization factor P (Section II). In general, from equation (3), scaling to more cores may not improve speedup for a smaller P as much as for a larger P. If it is possible to determine the P value of running any task on any platform, this knowledge may be useful for run-time task to core scheduling. This may be called P-aware run-time management.

P can be estimated if the speedup is known. This can be done for any known N through equation (10). It is also possible to determine P using data from experiments based on multiple N configurations through the method of regression, based on such criteria as least squares [19]. Regression has been used for run-time optimization based on learning for multi-core systems [34] where the models are unknown. Given equation (10), the motivation of using potentially expensive regression during run-time is weaker here. However, we first need to establish that equation (10) provides the same quality as regression-based methods.

TABLE V: Parallelization (P) calculations for synthetic benchmark [20].

SoC	P_{SW}	P_{LST}	P_{EQT} (10)	P_{LSC}	P_{EQC} (10)
Core-i7	0.1	0.0994	0.0993	0.0999	0.0998
Core-i7	0.4	0.3990	0.3996	0.3990	0.3999
Core-i7	0.7	0.6840	0.6834	0.6990	0.6999
Core-i7	0.9	0.8970	0.8985	0.8820	0.8999
Xeon	0.1	0.0900	0.0892	0.1002	0.1002
Xeon	0.3	0.2940	0.2912	0.3002	0.3001
Xeon	0.7	0.7000	0.6852	0.7001	0.6999
Xeon	0.9	0.8905	0.8849	0.9000	0.9001
Xeon Phi	0.1	0.1008	0.1086	0.1007	0.1087
Xeon Phi	0.4	0.4003	0.4007	0.4012	0.4015
Xeon Phi	0.5	0.5037	0.5038	0.5038	0.5036
Xeon Phi	0.9	0.8892	0.9020	0.9008	0.9029

TABLE VI: Parallelization factor (P) calculations of PARSEC benchmarks.

Benchmark	P_{LST}	P_{EQT} (10)	P_{LSC}	P_{EQC} (10)
bodytrack	0.981	0.925	0.937	0.965
blackscholes	0.841	0.852	0.868	0.872
facesim	0.900	0.920	0.941	0.948
fluidanimate	0.895	0.902	0.927	0.926
freqmine	0.985	0.984	0.985	0.986
swaptions	0.990	0.993	0.994	0.995
streamcluster	0.859	0.858	0.884	0.873
canneal	0.757	0.762	0.774	0.776
dedup	0.940	0.927	0.953	0.938

The other question we must consider is the avoidance of instrumenting applications for time. Can we replace time measurements with clock-related performance counter data for P estimation? In this section we attempt to estimate P from speedup derived from both clock performance counters and from direct time measurements, using both regression and equation (10) calculations, and compare the results. These again cover both the synthetic as well as PARSEC benchmarks.

Table V shows the results for the synthetic benchmark. Here it is regarded as desirable if the estimated P_{LS} values are obtained with least squares regression and P_{EQ} values obtained with equation (10) are closer to the software-programmed P_{SW} set within the benchmark. It can be observed that the differences between regression and equation (10) is small with P_{EQ} tracking P_{SL} closely in both time measurement and clock derived cases. It can also be observed that P_{LSC} values are very close to P_{LST} values and P_{EQC} values are very close to P_{EQT} values, meaning that using clock performance counters is valid. And finally, all the estimated P values are very close to the programmed P values set in the benchmark. In other words, this shows that 1) estimating P from speedup is a valid approach and 2) using clock performance counter data to replace time instrumentation is a valid approach. Table VI shows the results for PARSEC benchmarks, whose intrinsic P values are unknown and not explicitly set within the programs. They also have more memory access which might introduce unpredictable waiting and synchronization effects making their P values potentially different from run to run. As a result, it is not possible to compare estimated P values to a reference value, and the comparison tries to answer two questions:

Is it a valid approach to use equation (10) to avoid regression and is it a valid approach to make use of clock performance counter data to avoid instrumenting applications for time monitoring. The results show that the answer is yes for both questions with differences between the approaches generally being very small.

As a result, we propose to make use of equation (10) directly to estimate P from speedup estimated from clock performance counters in run-time P-aware scaling management, which is our immediate future work.

VI. Parallelization-aware Energy Efficient Computing

As mentioned in Section V-C, if the P value of an application running on a platform is known, run-time decisions may be made based on this knowledge to improve speedup. Beyond just speedup, Shafique *etal* have shown that in dark silicon operations, the P values of workloads need to be considered to arrive at optimal resource allocations through such techniques as dark silicon patterning [35]–[37].

In this section, we investigate whether it is possible to optimize energy efficiency with a knowledge of P. For this purpose experiments are carried out to relate metrics of energy efficiency to P. We don't make dark silicon assumptions in this study. Parallelization in dark silicon will be a future topic of research.

A. Energy and Power Data

Energy consumption data is collected from experiments described in the preceding sections using the method described in Section IV-B. The energy performance counter PWR_PKG_ENERGY gives total energy consumed by all cores. We calculate the total power consumption W_{total} by dividing energy by the realistic execution time obtained by $CPU\_CLK\_UNHALTED\_REF$. Fig. 5 shows the energy consumption in Intel Core-i7 quad core processor for the synthetic benchmark. Applications that have high parallelization factors consume less energy in high frequency scaling and maximum core allocations as shown in Fig. 5(a) for the P factor equal to 0.9, whereas applications with low parallelization factors consume higher energy as shown in Fig. 5(b) for the P factor equal to 0.1. We observe that for low parallelization factors the best energy consumption is obtained with three cores and highest frequency scaling.

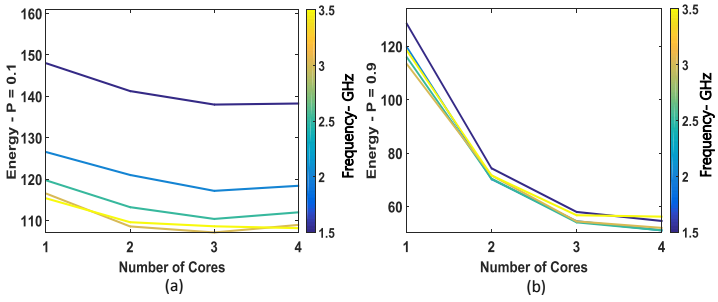

Fig. 5: Energy consumption for synthetic application: a) high parallelization factor P=0.9, b) low parallelization factor P=0.1.

B. Power Normalized Performance and Energy-Delay-Product

Both power normalized performance and EDP are metrics on energy efficiency, with different emphasises, as discussed in Section III-D. Here we calculate power normalized performance from (11) and EDP from (13). Fig. 6 shows power normalized performance of the synthetic benchmark, in high parallelization factor the best performance is obtained from maximum number of cores and maximum frequency as shown in Fig. 6(a), in low parallelization factor the best performance is obtained from high frequency scaling in 3 or 4 cores as shown in Figure 6(b).

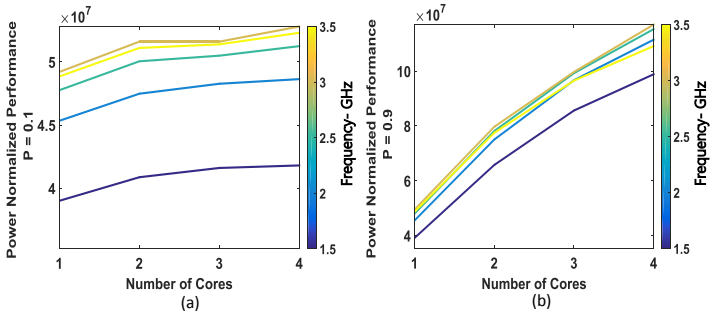

Fig. 6: Power normalized performance for synthetic application: a) high parallelization factor P=0.9, b) low parallelization factor P=0.1.

Fig. 7 shows the calculation of energy-delay-product of synthetic benchmark, in high parallelization factor the best outcome is obtained from high frequency scaling and 3 or 4 number of cores as shown in Fig. 7(a), in low parallelization factor the best outcome is obtained from high frequency scaling in 3 and 4 cores as shown in Fig. 7(b).

The data presented in this section shows that optimal energy efficiency, as measured in either metric, is a function of P. As a result, the idea of parallelization-aware energy efficient computing is valid and we propose to study more examples and develop optimization methods that may be used at run-time as part of our immediate future work.

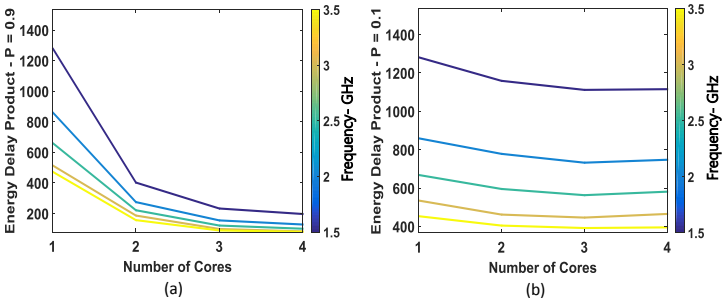

Fig. 7: Energy-delay-product for synthetic application: a) high parallelization factor P=0.9, b) low parallelization factor P=0.1.

VII. Conclusions and Discussions

This paper is the first attempt to address the problem of making use of Amdahl speedup model without knowing the parallelization factor P and without instrumenting applications for time monitoring. Performance counters are proposed as a solution to this problem. Speedup can be indicated by IPS data from before and after parallelization rather than directly from time delays. And by using IPC in place of IPS we make it possible to use instruction and clock performance counters for calculating speedup.

In this paper, we also solve the problem of differentiating application instructions from system software instructions and discover the behavior of typical system software instructions.

Extensive cross-validations have been performed by comparing model-calculated speedup with speedup derived from measured time, with small errors shown. The maximum error, which rarely occurs, is 8%. We followed performance counter speedup model to calculate PARSEC benchmark speedup, the outcomes show a sound error ratio related to outcomes derived from measured time (less than 6.5%).

We also propose a method of determining P once speedup is known, and this is cross-validated by comparing with the programmed P values in our experimental benchmark with very small errors (less than 3.26%). Furthermore, the parallelization factor of PARSEC benchmarks are calculated via the same model.

Based on these parallelization and speedup models we developed models for power, energy, power normalized performance and energy-delay-product explored the energy efficiency of core scaling.

We believe that the speedup, parallelization, and EDP models developed in this paper will give rise to a new method of run-time system control optimizing speedup and/or energy efficiency. This may be called parallelization-aware run-time management for performance and/or efficiency. We will focus on this direction in our future work.

VIII. Acknowledgment

This work is part of the PRiME project (EPSRC grant EP/K034448/1). The first author thanks his sponsor HCED-Iraq and University of Technology - Iraq for funding and supporting.

References

[1] S. Borkar, "Thousand core chips: a technology perspective," in *Proceedings of the 44th annual DAC*. ACM, 2007, pp. 746–749.

[2] G. E. Moore *et al.*, "Cramming more components onto integrated circuits," *Proceedings of the IEEE*, vol. 86, no. 1, pp. 82–85, 1998.

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

[4] F. J. Pollack, "New microarchitecture challenges in the coming generations of CMOS process technologies (keynote address)," in *Proceedings of the 32nd annual ACM/IEEE international symposium on Microarchitecture*. IEEE Computer Society, 1999, p. 2.

[5] J. Henkel, H. Khdr, S. Pagani, and M. Shafique, "New trends in dark silicon," in *Proceedings of the 52Nd Annual Design Automation Conference*, ser. DAC '15. New York, NY, USA: ACM, 2015, pp. 119:1–119:6. [Online]. Available: http://doi.acm.org/10.1145/2744769.2747938

[6] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *Proceedings of the April 18-20, 1967, spring joint computer conference*. ACM, 1967, pp. 483–485.

[7] M. D. Hill and M. R. Marty, "Amdahl's law in the multicore era," *Computer*, no. 7, pp. 33–38, 2008.

[8] N. Ye, Z. Hao, and X. Xie, "The speedup model for manycore processor," in *(ISCC-C), 2013 International Conference on*. IEEE, 2013, pp. 469–474.

[9] E. Yao, Y. Bao, G. Tan, and M. Chen, "Extending Amdahl's law in the multicore era," *SIGMETRICS Perform. Eval. Rev.*, vol. 37, no. 2, pp. 24–26, Oct. 2009.

[10] G. H. Loh, "The cost of uncore in throughput-oriented many-core processors," in *In Proceeding of Workshop on (ALTA)*, 2008.

[11] X.-H. Sun and Y. Chen, "Reevaluating Amdahl's law in the multicore era," *Journal of Parallel and Distributed Computing*, vol. 70, no. 2, pp. 183–188, 2010.

[12] D. H. Woo and H.-H. S. Lee, "Extending Amdahl's law for energy-efficient computing in the many-core era," *Computer*, no. 12, pp. 24–31, 2008.

[13] P. Ranadive and V. G. Vaidya, "Modeling parallelization overheads for predicting performance," in *2016 (ICISE)*. IEEE, 2016, pp. 62–67.

[14] J. Dou and M. Cintra, "Compiler estimation of load imbalance overhead in speculative parallelization," in *Proceedings of the 13th PACT*, ser. PACT '04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 203–214. [Online]. Available: http://dx.doi.org/10.1109/PACT.2004.12

[15] A. I. Elnashar, "To parallelize or not to parallelize, speed up issue," *CoRR*, vol. abs/1103.5616, 2011. [Online]. Available: http://arxiv.org/abs/1103.5616

[16] 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.

[17] M. Kim, P. Kumar, H. Kim, and B. Brett, "Predicting potential speedup of serial code via lightweight profiling and emulations with memory performance model," in *(IPDPS), 26th International*. IEEE, 2012, pp. 1318–1329.

[18] A. B. Downey, "A model for speedup of parallel programs," Berkeley, CA, USA, Tech. Rep., 1997.

[19] F. Xia, A. Rafiev, A. Aalsaud, M. Al-Hayanni, A. Mokhov, A. Romanovsky, R. Shafik, A. Yakovlev, and S. Yang, "Voltage, throughput, power, reliability and multi-core scaling," 2017.

[20] Data. [Online]. Available: http://async.org.uk/data/speed-up-2016/

[21] M. Al-hayanni, A. Rafiev, , R. Shafik, and F. Xia, "Power and energy normalized speedup models for heterogeneous many core computing," in *(ACSD)*, Torun, Poland., 2016.

[22] M. Al-Hayanni, A. Rafiev, R. Shafik, F. Xia, and A. Yakovlev, "Extended power and energy normalized performance models for many-core systems," Newcastle University, Tech. Rep. NCL-EEE-MICRO-TR-2016-198, 2016. [Online]. Available: http://async.org.uk/tech-reports/NCL-EEE-MICRO-TR-2016-198.pdf

[23] J. H. Laros III, K. Pedretti, S. M. Kelly, W. Shu, K. Ferreira, J. Van Dyke, and C. Vaughan, *Energy-efficient high performance computing: measurement and tuning*. Springer Science & Business Media, 2012.

[24] J. Treibig, G. Hager, and G. Wellein, "Likwid: A lightweight performance-oriented tool suite for x86 multicore environments," in *ICPPW*, ser. ICPPW '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 207–216. [Online]. Available: http://dx.doi.org/10.1109/ICPPW.2010.38

[25] likwid-perfctr: Measuring application using the hardware performance counters. [Online]. Available: https://github.com/RRZE-HPC/likwid/wiki/likwid-perfctr

[26] V. M. Weaver and S. A. McKee, "Can hardware performance counters be trusted?" in *Workload Characterization, 2008. IISWC 2008. IEEE International Symposium on*, Sept 2008, pp. 141–150.

[27] Intel, *Intel 64 and IA-32 Architectures Software Developer's Manual. Volume 3B: System Programming Guide, Part 2*, 2016.

[28] Cycle accounting and Gooda. [Online]. Available: https://github.com/David-Levinthal/gooda/blob/master/gooda-analyzer/docs

[29] A. Nowak, D. Levinthal, and W. Zwaenepoel, "Hierarchical cycle accounting: a new method for application performance tuning," in *(ISPASS)*. IEEE, 2015, pp. 112–123.

[30] D. Hackenberg, T. Ilsche, R. Schöne, D. Molka, M. Schmidt, and W. E. Nagel, "Power measurement techniques on standard compute nodes: A quantitative comparison," in *(ISPASS)*. IEEE, 2013, pp. 194–204.

[31] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC benchmark suite: characterization and architectural implications," in *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*. ACM, 2008, pp. 72–81.

[32] C. Bienia and K. Li, "Parsec 2.0: A new benchmark suite for chip-multiprocessors."

[33] M. Bhadauria, V. weaver, and S. A. Mckee, "A characterization of the PARSEC benchmark suite for CMP design," Computer System Laboratory, Cornell University, Ithaca, NY, Tech. Rep. CSL-TR-2008-1052, 2008.

[34] S. Yang, R. A. Shafik, G. V. Merrett, E. Stott, J. Levine, J. Davis, and B. Al-Hashimi, "Adaptive energy minimization of embedded heterogeneous system using regression-based learning," July 2015. [Online]. Available: http://eprints.soton.ac.uk/379535/

[35] S. Pagani, H. Khdr, J. J. Chen, M. Shafique, M. Li, and J. Henkel, "Thermal safe power (TSP): Efficient power budgeting for heterogeneous manycore systems in dark silicon," *IEEE Transactions on Computers*, vol. 66, no. 1, pp. 147–162, Jan 2017.

[36] M. Shafique, D. Gnad, S. Garg, and J. Henkel, "Variability-aware dark silicon management in on-chip many-core systems," in *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*. EDA Consortium, 2015, pp. 387–392.

[37] H. Khdr, S. Pagani, M. Shafique, and J. Henkel, "Thermal constrained resource management for mixed ILP-TLP workloads in dark silicon chips," in *52nd ACM/EDAC/IEEE (DAC)*, June 2015, pp. 1–6.