Received March 18, 2020, accepted April 14, 2020, date of publication April 27, 2020, date of current version May 11, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2990418

When Parallel Speedups Hit the Memory Wall

ALEX F. A. FURTUNATO[1], KYRIAKOS GEORGIOU[2], KERSTIN EDER[2], AND SAMUEL XAVIER-DE-SOUZA[3], (Senior Member, IEEE)

[1]Diretoria Acadêmica de Informática, Instituto Federal do RN, Natal 59015-000, Brazil
[2]Department of Computer Science, University of Bristol, Bristol BS8 1TH, U.K.
[3]Department of Computer Engineering and Automation, Universidade Federal do Rio Grande do Norte, Natal 59000-000, Brazil

Corresponding author: Alex F. A. Furtunato (alex.furtunato@ifrn.edu.br)

This work was supported in part by the High-Performance Computing Center at UFRN (NPAD/UFRN), in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior-Brasil (CAPES) under Finance Code 001, in part by the Royal Society-Newton Advanced Fellowship under Award NA160108, and in part by the European-Union's Horizon 2020 Research and Innovation Programme through the Time, Energy and security Analysis for Multi/Many-core heterogeneous platforms (TeamPlay) under Grant 779882.

ABSTRACT After Amdahl's trailblazing work, many other authors proposed analytical speedup models but none have considered the limiting effect of the memory wall. These models exploited aspects such as problem-size variation, memory size, communication overhead, and synchronization overhead, but data-access delays are assumed to be constant. Nevertheless, such delays can vary, for example, according to the number of cores used and the ratio between processor and memory frequencies. Given the large number of possible configurations of operating frequency and number of cores that current architectures can offer, suitable speedup models to describe such variations among these configurations are quite desirable for off-line or on-line scheduling decisions. This work proposes a new parallel speedup model that accounts for the variations on the average data-access delay to describe the limiting effect of the memory wall on parallel speedups in homogeneous shared-memory architectures. Analytical results indicate that the proposed modeling can capture the desired behavior while experimental hardware results validate the former. Additionally, we show that when accounting for parameters that reflect the intrinsic characteristics of the applications, such as the degree of parallelism and susceptibility to the memory wall, our proposal has significant advantages over machine-learning-based modeling. Moreover, our experiments show that conventional machine-learning modeling, besides being black-boxed, needs about one order of magnitude more measurements to reach the same level of accuracy achieved by the proposed model.

INDEX TERMS Parallel systems, data access delay, performance modeling, speedup, memory wall.

I. INTRODUCTION

Amdahl's Law [1] has driven the chase for single-processor performance improvements for decades, but the end of frequency-upscaling and the stagnation of instruction level parallelism altogether led to the dawn of a new computational era: the multi-core and many-core era.

In this new era, parallel computing has become the conventional approach to achieve ever-increasing computational performance. Although parallelism is not new in computational systems, its real potential has been obfuscated for many decades by two main factors: Amdahl's skepticism on the ability of parallel systems to scale performance, and the exponential speed growth of single processor systems. It is now a consensus that Amdahl had a limited view on parallelism, and thus numerous works have been emerging towards expressing and exploiting the advantages that parallel computing can offer [2]–[6]. Continuing to broaden and explore different views on parallelism remains of vital importance in maximizing the potentials that parallel computing can offer.

This paper widens the views on parallelism by exploring the effects of the number of cores and their operating frequency on the data-access delay for parallel applications that make extensive use of the main memory. Memory-bound programs are hard to model because their behavior is volatile across runs with different inputs and system configurations due to the variability of how such applications exploit the memory hierarchy. We dedicate the following paragraphs to describe the existing views on parallelism, which we argue do not consider these aspects.

Amdahl showed that even a tiny not parallelized code fraction of an application could compromise the applicability

The associate editor coordinating the review of this manuscript and approving it for publication was Shadi Alawneh.

of multiple processors to scale the application's performance [1]. Long after Amdahl's work on the inability of using multiple processors to scale performance, Gustafson's ''fixed-time speedup'' approach to parallelism has shown that larger programs can benefit from more processors [2]. Amdahl's ''fixed-size speedup'' had a limited view on the potential of parallelism. Gustafson's scaling model, known as Gustafson's Law, opened the path to the multi-core and many-core era. In [4], the author unifies Amdahl and Gustafson's works and concludes that using the execution times instead of the serial and parallel fractions of the code could have avoided decades of unconstructive criticism against the advantages of using parallel processing. Sun and Ni [3] coined another prevalent model shortly after Gustafson's seminal work. The authors present a memory-bounded speedup model, known as Sun and Ni's Law. Their modeling demonstrates that the memory size is a limiting factor for parallel scalability.

More recently, other models extend these analyses to multi-core architectures, showing that they scale better for asymmetric and dynamic multi-core chips [5]. In [6], the authors summarize the contributions of three main speedup models (fixed-size, fixed-time, and memory-bounded speedups models) to the multi-core era, presenting a very optimistic view. However, their view assumes that the data-access delay is fixed and independent of the number of cores and problem sizes. This assumption is often unrealistic because of the memory wall [7], caused by the increasing data-access delay as the number of cores increases. In the following, we discuss three of the significant factors that can affect the data-access delay of an application running in a homogeneous shared-memory architecture: the application's problem/input size, the number of cores utilized, and the ratio of the processor's and memory's frequencies.

While the scaling of the problem size may affect the data-access delay, whether this effect is negative or positive for performance depends on the application's nature and on how the application is utilizing the targeted architecture. In general, increasing the input size can trigger a higher activity in the memory hierarchy, causing more cache misses, which subsequently generates more main memory accesses per cycle. Often, cache-blocking techniques can be applied to avoid or reduce this effect. The modeling presented in this paper does not consider variations in the problem/input size.

Increasing the number of cores can have an even more significant effect on the data-access delay depending on the architecture's characteristics. For instance, even with the problem size kept constant, using more processing cores can cause an increasing data-access delay because the rate of access-requests per cycle can increase due to more cores making simultaneous requests to the same memory. When the demand for accesses reaches the memory's nominal rate of attended requests per cycle, the average data-access delay starts to increase, stagnating the performance scaling in the number of cores, even for codes that are entirely parallel or that have a tiny serial fraction. Hence, for these cases, increasing the number of cores can indeed increase the data-access delay, which will undesirably generate an adverse effect on speedup in a form that resembles an increase in the serial fraction of the application. On the other hand, in the case of private-caches, increasing the number of cores can lead to more available caches, and thus, to fewer memory accesses that, up to a degree, will have a positive effect on the data-access delay and thus will possibly allow further performance gains through parallelization.

A third factor to consider is the ratio of the processor's and memory's frequencies. If the processor is running significantly faster than the memory, the data-access delay relative to the processor speed may also increase. Considering all these factors and their interactions is crucial both for developing parallel programs that do not become bounded by the memory and for finding the optimal configuration of the number of cores and the processor's frequency that achieves maximum speedup for an application. Currently, there is no analytical model to capture these effects altogether. Some authors have used hardware performance counters to build models [8]–[10]. However, since those are processor-specific and not standardised, their use limits the portability of the models.

In this paper, we present a new analytical speedup-model for multi-core architectures that captures the adverse and the favorable effects on performance due to variations in the data-access delay caused by increasing the number of cores (see II). The proposed model does not use performance counters and therefore is arguably more portable and less complex than those that do.

The proposed model has many practical uses, including finding suitable configurations [11]–[13] that, coupled to a power model, could achieve better energy efficiency while meeting the application's performance constraint. It could also be used by operating systems to estimate relative performance of multiple applications and to implement resource-optimal scheduling. Estimating wall time for high performance computing jobs in unseen configurations is another possible practical use for the proposed model.

We initially investigate the potential abilities of the proposed model to capture the above effects analytically (III). The analytical results indicated that the speedup is dependent on the ratio between the frequencies of the processor and the main memory, both for memory-bound applications and for processor-bound applications that became memory-bounded after an increase in the number of cores. The analysis indicated that the larger this ratio, the higher its limiting effect can be on the speedup and that this limitation grows with the degree of parallelism of the code.

The proposed modeling was then fitted with actual hardware measurements to validate our analytical findings (IV). Furthermore, we demonstrate that our approach has higher accuracy and lower variance than Amdahl's model (IV-B). Comparisons to other analytical speedup models would not be more relevant since the other models differ from Amdahl's model by aspects that were not considered in our experiments,

such as the problem size and architectural features like memory hierarchy and the amount of memory available. V presents more details. Therefore, to the best of our knowledge, the features modeled by other models are orthogonal to the memory-wall effect modeled in this work. Thus, those models, and their features, are complementary to the proposed model.

We compare the proposed model to non-linear machine learning approaches (IV-C), which are considered more flexible than any analytical model. In this comparison, the proposed model is demonstrated to exhibit a higher accuracy while using fewer hardware measurements.

Finally, based on the presented modeling and experimental results, we then discuss the implications that the contributions of this paper can have in application-specific multi-core design and towards more energy-efficient parallel software.

The paper is organized as follows. In Section II we present our modeling for speedup as a function of the ratio between processor and memory frequencies. In Section III we analyze the model behavior. In Section IV, we detail the methodology used to validate the proposed models and provide results of experiments in real hardware. In Section V we put our contributions in perspective with the existing literature and, finally, in Section VI, we draw conclusions and suggest future work.

II. VARIABLE-DELAY SPEEDUP MODEL

In this section, we devise a new parallel speedup model that accounts for the effect of the variation in the number of cores on the data-access delay. Furthermore, the model allows us to describe the effect that variations of the ratio between processor and memory frequencies have on the speedup.

Let us first restate the equation for the speedup of an application running in parallel with p cores as follows:

$$S_p = T_s/T_p, \tag{1}$$

where T_s is the sequential time, measured when running the application on a single core processor, and T_p is the time for running the same application in parallel with p cores.

We now make some simplifying assumptions, desirable and necessary to achieve a good trade-off between accuracy and complexity of the proposed model. These are later proved to be satisfactorily sustained by the model validation presented in IV:

Assumption 1: The computations of a given application can be divided into two types of instructions: memory instructions and processor instructions. The former representing the loads and stores that generate accesses to the main memory and the latter representing those instructions that are carried out without data transfer and those loads and stores that are captured by the cache hierarchy. This is an abstraction similar to Amdahl's assumption that the parallel and sequential parts of the code never overlap, which is often and generally not the case, but allows for model simplification. The total number of instructions is then given by

$$W = C + M, \tag{2}$$

where C is the number of processor instructions, and M is the number of memory instructions.

Assumption 2: The main memory can only attend requests at a given maximum rate. And, for a given parallel application, the access time is approximated by an average access time.

Assumption 3: For a specific processor frequency, the execution time of processor instructions can be approximated by an average value t_c, which is inversely proportional to the processor operating frequency.

Assumption 4: For a specific processor frequency and memory frequency, the time necessary to execute a memory instruction, as defined in Assumption 1, can be approximated by an average value t_m.

Then, the sequential execution time for the computation of all W instructions can be given by

$$T_s = t_c C + t_m M. \tag{3}$$

Accordingly, the formulation of an equation for the parallel execution time for the computation of the same W instructions depends on how these instructions are distributed and carried out by multiple processing elements. We use a simplistic model first coined by Amdahl in [1] to model parallel software. The computation is modeled by a parallel fraction f, representing the instructions that have no dependencies among them and that could be executed in parallel with no performance penalty, and its complement $(1-f)$, which correspond to the serial fraction or the fraction of code that cannot be parallelized. The parallel execution time for p processing cores would then be given by

$$T_p = (1-f)T_s + f\frac{T_s}{p}. \tag{4}$$

Amdahl's model arises from combining (1) and (4), such that

$$S_p = \frac{1}{(1-f) + \frac{f}{p}}. \tag{5}$$

However, with Assumption 2, we must consider that the memory system can only attend requests at a given maximum rate. Therefore, the term that is divided by p in (4) cannot decrease indefinitely. In fact, the execution time of the whole parallel computation cannot be accelerated beyond $t_m M$ by increasing p, which leads us to the following equation for the parallel execution time of the W instructions with p processing cores.

$$T_p = \max\left((1-f)T_s + f\frac{T_s}{p}, t_m M\right). \tag{6}$$

Next, we devise a model that accounts for the variation in the number of memory accesses, dependent on the number

of cores used, and the variation in the average duration of a memory instruction, dependent on the processor and memory frequencies ratio.

By combining (1), (3) and (6), we derive the first form of our speedup model:

$$S_p = \frac{t_c C + t_m M}{\max\left((t_c C + t_m M)\left((1-f)+\frac{f}{p}\right), t_m M\right)} \tag{7}$$

In terms of the ratio between the time to complete a memory instruction and the time to complete a processor instruction, by dividing everything by t_c, we can rewrite (7) as

$$S_p = \frac{C + \rho M}{\max\left((C + \rho M)\left((1-f)+\frac{f}{p}\right), \rho M\right)}, \tag{8}$$

where ρ denotes the ratio between t_m and t_c.

The average duration of a memory instruction should depend on the processor instruction execution time and memory access frequency according to Assumption 4, which we model as follows.

$$t_m = t_c + \frac{k}{F_{Mem}}, \tag{9}$$

where k is an application model parameter that models how the computation of memory instructions is affected by the frequency of the main memory. The effect of k is stronger for memory-bound applications and weaker for those that are CPU-bound.

So, considering (9) and Assumption 3, the ratio ρ can be expressed as

$$\rho = \frac{t_m}{t_c} = 1 + k\phi, \tag{10}$$

where ϕ is the ratio between processor and memory frequencies,

$$\phi = \frac{F_{CPU}}{F_{Mem}}. \tag{11}$$

with F_{CPU} and F_{Mem} denoting the processor and memory frequencies, respectively.

Finally, to remove the absolute values of M and C from (8), we can rewrite it in terms of the fraction of memory instructions over the total number of instructions, μ, as follows.

$$S_p = \frac{(1-\mu) + \rho\mu}{\max\left(((1-\mu) + \rho\mu)\left((1-f)+\frac{f}{p}\right), \rho\mu\right)}, \tag{12}$$

where

$$\mu = \frac{M}{W}. \tag{13}$$

Consequently,

$$1 - \mu = \frac{W - M}{W} = \frac{C}{W} \tag{14}$$

is the fraction of processor instructions over the total number of instructions involved in the computation. The ratio μ, however, is not fixed due to Assumption 1. When we vary the number of cores, the value of μ may also change due to the addition of more private caches, as discussed in I. To account for variations in the number of memory instructions caused by variations in the number of cores, we rewrite (12) to express the final form of our proposed variable-delay speedup model as follows.

$$S_p = \frac{(1-\mu_1) + \rho\mu_1}{\max\left(((1-\mu_p) + \rho\mu_p)\left((1-f)+\frac{f}{p}\right), \rho\mu_p\right)}, \tag{15}$$

for μ_p being the fraction of memory instructions observed when using p cores, defined by

$$\mu_p = \min\left(m_1 + \frac{m_2}{p}, 1\right), \tag{16}$$

with m_1 and m_2 denoting application model parameters and μ_1 representing the serial case of μ_p, with $p = 1$. The minimum function $\min(\cdot, 1)$ limits the upper value of μ_p to 1, which represents an application that is 100% dependent on memory instructions. The term m_1 accounts for the portion of accesses that are not affected by changes in the number of cores. The term m_2 accounts for the portion of accesses that vary with changes in the number of cores, which for example would vary μ due to the addition of more private caches. With more caches, the main memory receives fewer accesses, and μ should decrease.

III. MODEL ANALYSIS

In this section, we perform two parametric analyses with the model proposed in (15) to investigate the model's behavior. What we intend is to present the model's ability to capture the performance-limiting behavior caused by a change in the data-access delay. Then, in IV, this ability is validated by fitting the model in (15) to hardware measurements.

Firstly, we investigate the dependency between the number of cores and the data-access delay which causes the memory performance to decrease with an increase in the number of active cores. Secondly, we investigate the performance predictions for variations on the ratio between processor frequency and memory frequency.

Because exhaustive analyzes with seven parameters (f, k, m_1, m_2, f, ϕ, and p) would be impractical, we propose a set of parameter-value combinations whose variations can better expose the behavior expected to be modeled.

A. NUMBER OF CORES VERSUS DATA-ACCESS DELAY

We analyzed the behavior of the proposed speedup model for systems with 2, 4, 8, 16, 32 and 64 processing cores. We assumed a parallel fraction $f = 0.99$, representing a highly parallel code, and a processor and memory frequencies ratio $\phi = 3.0$, which would denote, e.g. the memory functioning at 1.0 Ghz and the processor at 3.0 GHz. Fig. 1 presents the speedup plots of these configurations for different values of k, m_1, and m_2.

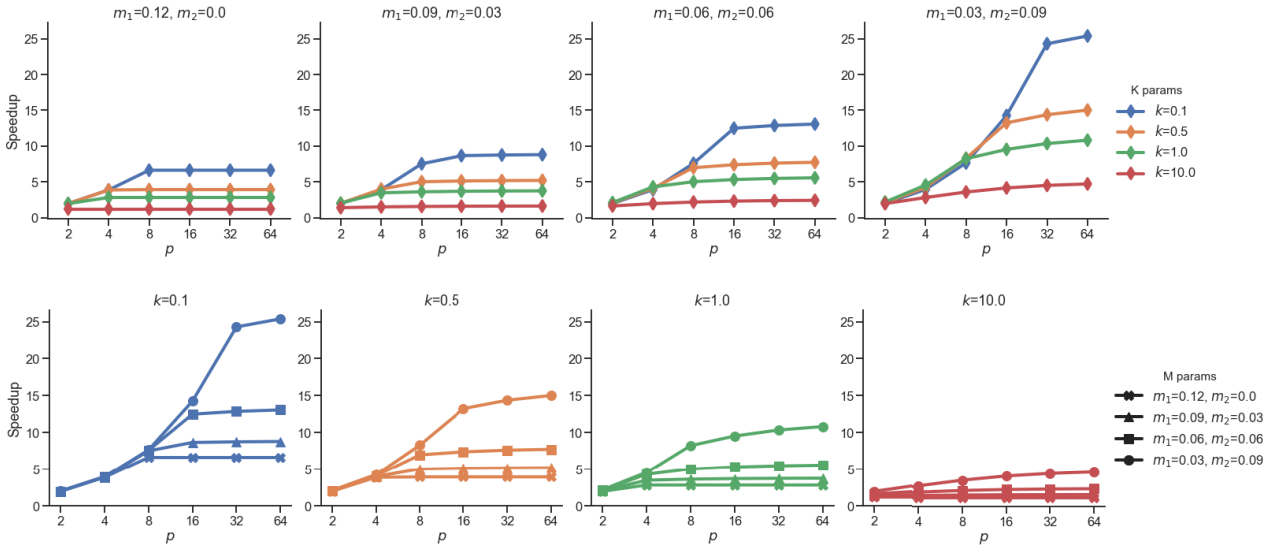


FIGURE 1. Speedup plots for a computational task with parallel fraction $f = 0.99$, frequencies ratio $\phi = 3.0$ and a varying number of cores $p = \{2, 4, 8, 12, 16, 32, 64\}$. Each plot and curves refers to combinations of k and m parameters. For k plots, the curves represent different m parameters, and vice versa.

As 1 shows, the model indicates that the ratio ρ, affected by k, has a significant effect on the speedups. The higher the k, the higher the limiting effect on speedups as the number of cores increases, which resembles the effect of a reduction of the parallel fraction of the code. So, the k parameter controls the memory access behavior of applications that depend on the variations of CPU and memory frequency ratio. For lower values of k and m_2, the speedups saturate faster with the increase in the number of cores, indicating that the application transitions from a processor-bound mode to a memory-bound one.

Fig. 1 also indicates the positive effects on the speedups caused by varying the number of cores with private caches. For larger values of m_2, which drives the number of memory instructions down with the use of more cores, the speedups are considerably larger. Higher values of m_2 allow the transition to a memory-bound mode behavior to happen at a larger number of cores with higher speedups whereas lower values force this to happen at smaller numbers of cores with lower speedups.

Considering that the frequencies of processor and memory are constant, larger values of the k parameter may represent applications with larger average memory-access time. So, in this case, a larger number of cores trend to saturate the speedup more quickly. On the other hand, the m_1 and m_2 parameters model the percentage of memory instructions of a particular application. The larger m_2 compared to m_1, the more susceptible the application behavior is to larger memory delay caused by an increase in the number of cores.

B. FREQUENCY RATIO VERSUS DATA-ACCESS DELAY

The analytical results of the previous subsection indicate that memory-bounded applications lose the apparent advantages of using more cores to achieve more considerable speedups at some point. The capacity of the memory to hold down the average data-access delay limits the speedup. Nonetheless, the effects of varying the ratio between the processor and memory frequencies remain to be analyzed.

With the following analysis, we intend to show that, according to the proposed model, a memory-bounded application can become processor bounded with a suitable adjustment of the ratio ϕ in order to make the processor work more symbiotically with the memory and, thus, could avoid processor idling, increase efficiency and decrease energy consumption.

We analyzed the behavior of our speedup model for computational tasks with parallel fractions $f = 0.99$ running with 32 processing cores. Processor and memory frequency ratios varied according to $\phi = \{1.0, 1.5, 2.0, 2.5, 3.0\}$, for which the plots are depicted in Fig. 2.

As expected, the proposed model reproduces the effect caused by varying the ratio of memory and processor frequencies. When the ϕ parameter increases—caused by an increase in the processor frequency, for example—the speedup decreases. However, this effect is more or less intense depending on the parameters that model the application. Thus, for the same number of cores, an increase in k makes this negative effect more evident. On the other hand, the parameters m_1 and m_2 are related to the number of memory instructions and, therefore, an increase in these also increases the sensitivity of the application to variations in the processor frequency.

Note, in Fig. 2, that larger speedups can be achieved by reducing the ratio ϕ in almost all analyzed configurations. This shows that the decay in memory performance could be avoided by a suitable reduction of the processor's operating frequency.

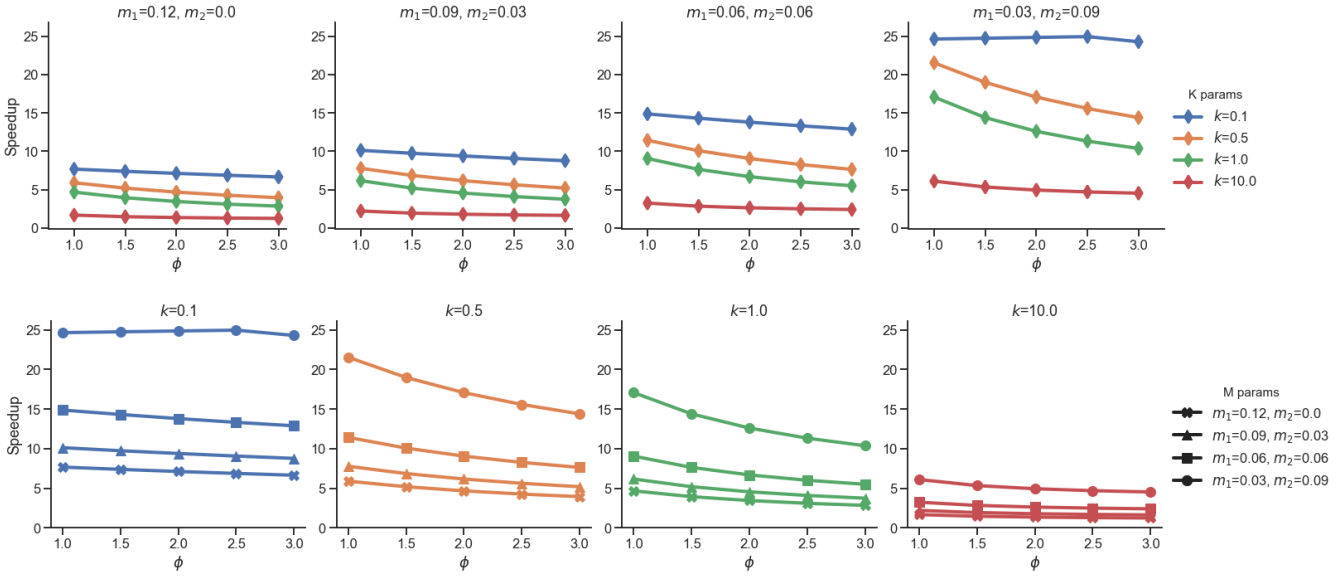


FIGURE 2. Speedup plots for computational tasks varying the ratio between the processor and memory frequencies $\phi = [1.0, 1.4, 1.8, 2.2, 2.4, 3.0]$, with number of cores $p = 32$ and parallel fraction $f = 0.99$. Each plot and curves refers to combinations of k and m parameters. For plots by k parameter, the curves represent different m parameters, or vice versa.

IV. MODEL VALIDATION

In this section, we present the results of several modeling experiments in order to validate the proposed model with real applications running on multi-core processors in a shared-memory architecture.

A. EXPERIMENTAL SETUP

We have measured the execution times for a set of applications varying the number of cores and their operating frequency in order to calculate their speedups for each frequency value. We validate the proposed model using the PARSEC [14] and SPLASH-2 [15] parallel benchmark suites. They comprise a large and diverse set of applications, covering several different application domains, such as computational finance, computer vision, real-time animation or media processing. In total there were 25 programs, 11 from the PARSEC suite and another 14 from the SPLASH-2 suite. We used the number of threads to control the number of cores active during the execution of each benchmark application. This way, besides effectively controlling the number of cores available, we also isolate from the measurements the effect on speedup arising from using multiple threads per core, which is not the target of our validation.

The measured execution times were used to fit the proposed model and Amdahl's model for each application. All model variables were fitted using the Coupled Simulated Annealing (CSA) [16] global optimization method to minimize the Mean Squared Error (MSE) between the measured application speedups and their models. The CSA method used was the CSA modified (CSA-M).

The ratio between memory and processor instructions is modeled by the m_1 and m_2 parameters that make up the μ_p instruction ratio. These parameters are fitted, using the CSA optimizer, based on the execution time measurements of the whole application.

To vary the ratio between processor frequency and memory frequency, we changed the processor's frequency for each execution round using the 'user-mode' governor from the "Advanced Configuration and Power Interface" (ACPI) driver. In contrast, the frequency of the memory system was fixed and known.

The measurements were taken on a dual-socket shared memory platform with 2× Intel(R) Xeon(R) CPU E5-2680 v3, 12 cores at 2.50 GHz, and 30 MB shared L3 cache. The L1 and L2 private caches have 64 KB and 256 KB, respectively. The operating processor core frequencies ranged from 1.2 GHz to 2.5 GHz, with steps of 100 Mhz. The number of cores ranged from 1 to 24, with unity steps, except for some applications that have the number of cores limited to a power of two. Hardware multi-threading was disabled to simplify modeling and to emphasize the effect of the memory wall. This way, cores were always running a single thread.

A Python version 3 library was developed[1] to implement the CSA algorithm and the utility methods to fit the models, to store the collected data, and to plot the graphs of the experiments performed in this paper. The repository also contains text files with information on measurements, execution metadata, the model parameters and the respective modeling errors for all experiments.

In IV-B, we will assess Amdahl's and the proposed model's accuracy by fitting them to each application using all measurements available to compute the MSE values.

In Section IV-C, we will investigate how the accuracy of these models and the accuracy of an unstructured machine

[1] https://gitlab.com/lappsufrn/parsecpy.git

TABLE 1. Models parameters and MSE for Amdahl's model and for the proposed model for the PARSEC and the SPLASH2 benchmarks applications using all available execution time measurements.

	Number of	Amdahl's Model (5)		Proposed model (15)					Accuracy
Benchmark Program	Measurements	f	MSE	f	k	m_1	m_2	MSE	Gain
parsec-blackscholes	322	1.0000	0.0042	0.7642	9.9264	0.0003	0.8761	0.0021	49.42 %
parsec-bodytrack	322	0.8934	0.1417	0.8984	9.7185	0.0090	0.0000	0.0931	34.29 %
parsec-canneal	322	0.9985	0.2325	0.9946	0.4341	0.0057	0.8562	0.1124	51.66 %
parsec-dedup	322	0.6745	0.1969	0.7387	0.1545	0.3210	0.0000	0.1481	24.82 %
parsec-facesim	84	0.9731	0.1443	0.9745	0.0950	0.0507	0.5482	0.0217	84.98 %
parsec-ferret	322	0.9912	1.3371	0.9952	0.2348	0.0368	0.1610	0.1104	91.74 %
parsec-fluidanimate	56	0.9834	0.0036	0.9954	0.0064	0.0174	0.9712	0.0029	19.49 %
parsec-freqmine	322	0.9791	0.1316	0.9907	0.0050	0.0294	0.8209	0.0096	92.69 %
parsec-raytrace	322	0.9959	0.0675	0.9155	9.9798	0.0039	0.7814	0.0623	7.70 %
parsec-streamcluster	322	0.9860	0.3274	0.9864	4.7217	0.0061	0.0024	0.1766	46.06 %
parsec-x264	322	1.0000	4.6452	0.9771	1.6662	0.0087	0.2638	0.6169	86.72 %
splash2x-barnes	322	0.9969	0.0290	0.8320	4.6578	0.0029	1.0000	0.0268	7.74 %
splash2x-cholesky	322	0.8978	1.8236	0.9273	0.1274	0.1301	0.0000	1.2997	28.73 %
splash2x-fft	56	0.9999	0.0436	0.9755	9.9984	0.0013	0.7153	0.0377	13.61 %
splash2x-fmm	322	0.9629	0.0326	0.8785	9.9976	0.0261	0.6269	0.0253	22.38 %
splash2x-lu-cb	322	0.9950	0.0668	0.7302	9.9672	0.0049	0.9257	0.0664	0.53 %
splash2x-lu-ncb	322	0.9538	3.0182	0.9657	3.3786	0.0154	0.0001	2.1160	29.89 %
splash2x-ocean-cp	56	0.9769	0.6297	0.9256	9.9994	0.0093	0.2050	0.3457	45.10 %
splash2x-ocean-ncp	56	1.0000	0.3854	0.9244	9.1902	0.0027	0.3123	0.1793	53.48 %
splash2x-radiosity	322	0.9408	0.8001	0.9674	0.1199	0.0940	0.0429	0.0844	89.45 %
splash2x-radix	56	0.9961	0.0172	0.9965	0.0321	0.0591	0.0609	0.0152	11.71 %
splash2x-raytrace	322	0.9973	0.0493	0.9520	1.1918	0.0040	0.9179	0.0356	27.81 %
splash2x-volrend	322	0.8037	0.1327	0.7901	8.8634	0.1827	1.0000	0.1028	22.51 %
splash2x-water-nsquared	322	0.9892	0.1468	0.8800	3.9348	0.0103	1.0000	0.1243	15.33 %
splash2x-water-spatial	322	1.0000	41.7510	0.9947	1.7165	0.0022	0.2811	4.0755	90.24 %

learning model vary according to the amount of information used to construct them.

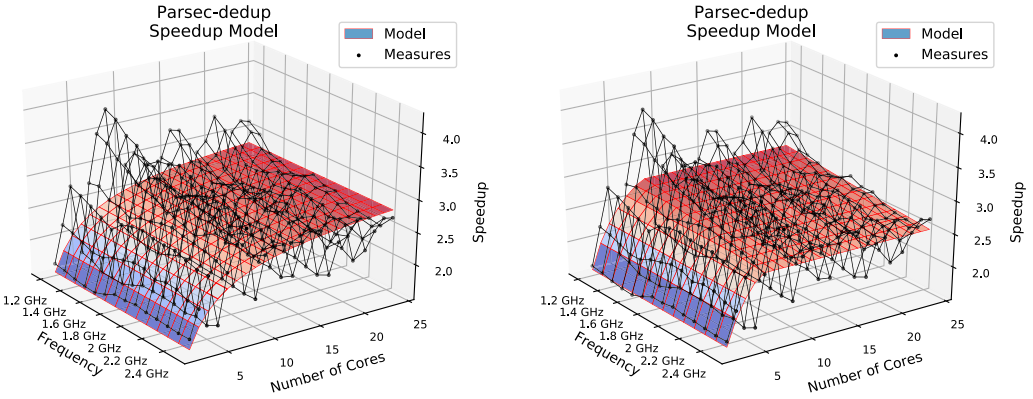
B. MODEL ACCURACY

The accuracy for Amdahl's model and the proposed model is summarized in Table 1 for all applications in terms of MSE. The table also shows the number of measurement points available for each application. Each measurement point represents a configuration of frequency and number of cores. These points are relative to the median of 10 runs of an application.

The MSE columns in Table 1 show that the results of the proposed model are considerably better than Amdahl's model, with the proposed model scoring always better or the same. The application with the most similar MSE value is "splash2x-lu-cb", whose accuracy was only 0.53% better than with Amdahl's model. On the other hand, "splash2x-water-spatial" was the application whose difference in MSE value was 90.24% better for the proposed model. On average, the proposed model was 41.92% more accurate than Amdahl's model considering all modeled applications.

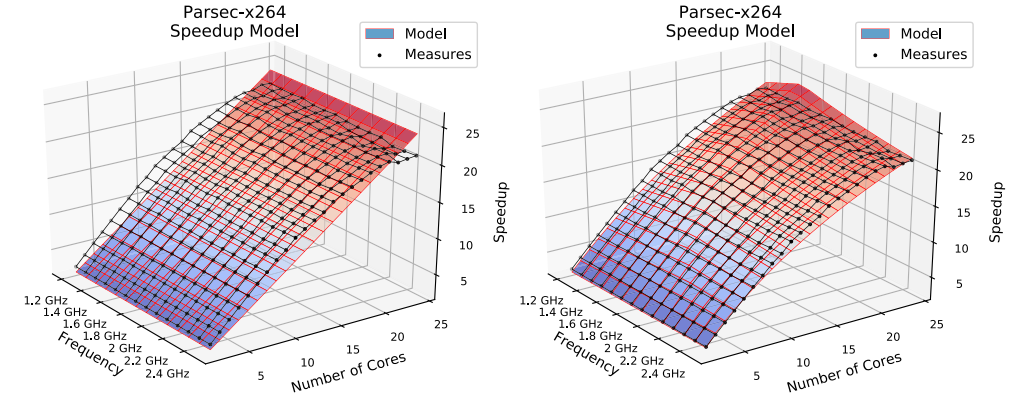
To better present the ability of the proposed model to describe the speedup features of parallel applications correctly, we have selected a few applications for a more detailed analysis. For example, the PARSEC Dedup, a workload that uses "deduplication" to compress a data stream [17], presents small differences in the MSE values of the two models. This application is hard to model because of abrupt speedup variation due to workload imbalance among threads [18]. Nevertheless, the proposed model improves Amdahl's accuracy and accomplishes its task of modeling access-delay limitations by tilting speedups down for more substantial amounts of cores and larger ϕ ratios, as shown in Fig. 3b. The model manages to capture the angle of the speedups along the frequency axis which represents the ϕ ratio. The proposed model also presents a better fit for a smaller number of cores with a steeper slope enabled by the variable number of memory instructions in (16) that allows the modeling of the effect of overcoming cache size limitations.

For the PARSEC x264 application, an H.264/AVC video encoder, the proposed model reduces the MSE error by one order of magnitude. Fig. 4b shows how the proposed model surface is very close to the scatter plot of the measurements. It captures the super-linear speedup that occurs with this application because of the m_2 term in (16) that allows the number of memory instructions μ_p to decay with increase of the number of cores.



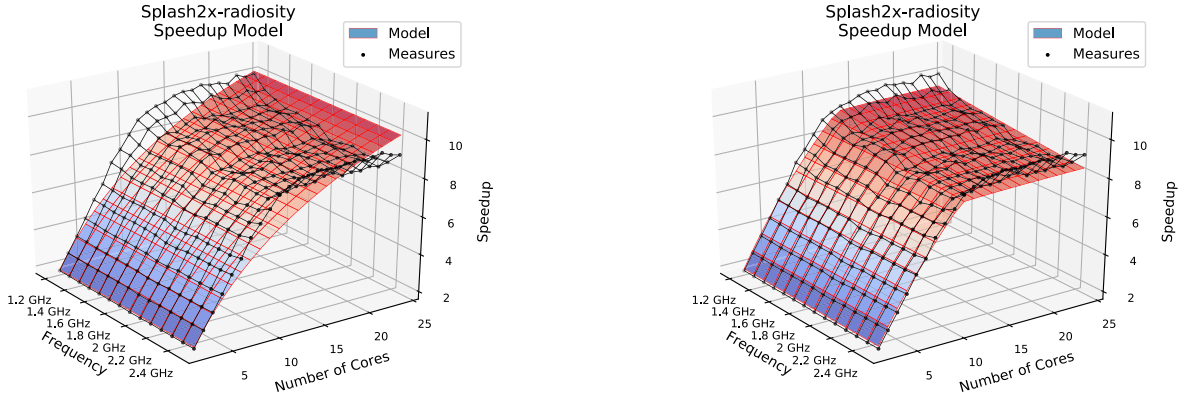
(a) Modeled by Amdahl (5). (b) Proposed model (15).

FIGURE 3. Amdahl's and proposed models for the PARSEC Dedup application. Dedup was developed by Princeton University. It compresses a data stream with a combination of global and local compression that is called 'deduplication'.



(a) Modeled by Amdahl (5). (b) Proposed model (15).

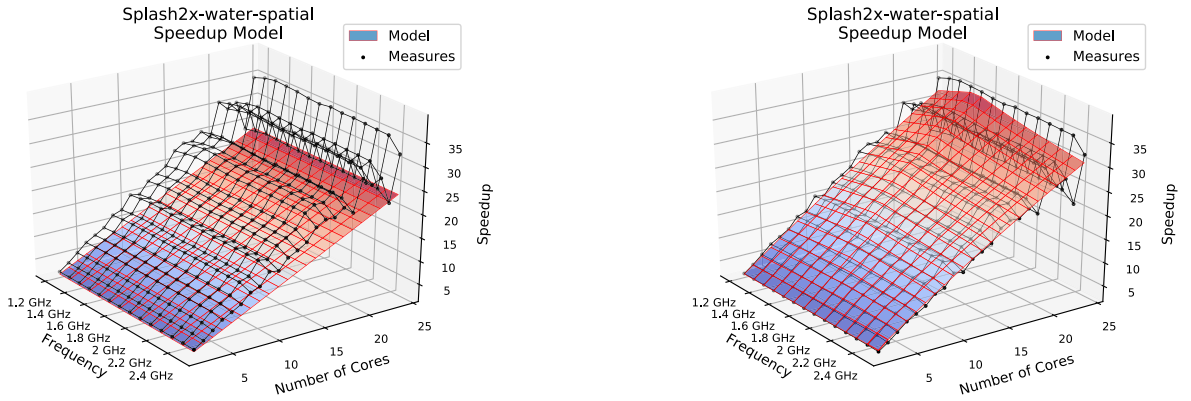
FIGURE 4. Amdahl's and proposed model for the PARSEC X264 application. X264 is an H.264/AVC (Advanced Video Coding) video encoder. H.264 describes the lossy compression of a video stream.



(a) Modeled by Amdahl (5). (b) Proposed model (15).

FIGURE 5. Amdahl's and proposed model for the SPLASH-2 Radiosity application. Radiosity computes the equilibrium distribution of light in a scene using the hierarchical diffuse radiosity method.

Fig. 5 presents the models for the SPLASH-2 Radiosity application. It computes the equilibrium distribution of light in a scene [15]. One of the computational characteristics of this algorithm is a large number of memory instructions and, therefore, it is an appropriate case study to prove the proposed model's ability to capture the memory-wall effect on speedups. As in the previous applications, the proposed model presents a much better fit than the fit of Amdahl's



(a) Modeled by Amdahl (5).

(b) Proposed model (15).

FIGURE 6. Amdahl's and proposed model for the SPLASH-2 Water Spatial application. This application evaluates forces and potentials that occur over time in a system of water molecules.

model. Fig. 5b shows how the proposed model captures the speedup's slope that increases as processor frequency decreases. The model also captures the abrupt saturation that occurs when speedups hit the memory wall.

For SPLASH-2 Water Spatial application, which computes the forces that occur over time on a system with water molecules, Amdahl's model failed to capture the super-linear speedup behavior, achieving the worst MSE errors among the other applications, as Fig. 6 illustrates. The proposed model presents a better fit, despite it underestimating speedups at lower frequencies. Nevertheless, its accuracy is more than 90% better.

C. ACCURACY VERSUS THE NUMBER OF MEASUREMENTS

The results of the previous section were obtained using all available measurements for all configurations of processor frequency and the number of cores. In most cases, each application was executed on 336 different configurations—14 different frequencies and 24 different numbers of cores. For practical scenarios, using as few measurements as possible is desirable to reduce the modeling overhead in terms of the use of computational resources and energy consumption.

In this section we study how the use of fewer sampling points affects model accuracy. With that we intend to support two claims:

- the proposed model can achieve reasonable accuracy even for a small number of measurements; and
- the number of measurements required for reasonable accuracy is much smaller than that required for unstructured models, such as those based on machine learning.

To support the former claim, we observed the accuracy of the models when fitted using various different numbers of measurements, starting from only 4 measurements and then doubling this number several times until reaching the closest power of two below the total number of available measurements for each application. To support the latter claim, we used machine learning techniques to model the applications using the same inputs as were used to fit the analytical

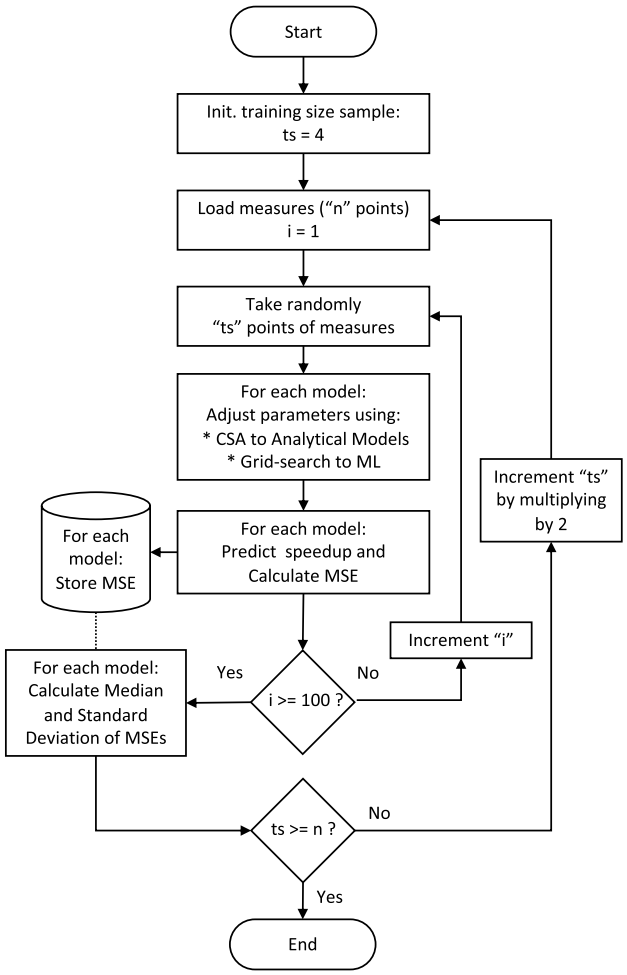


FIGURE 7. Flow chart of procedure used to compute the median and the standard deviation of the MSE for each model using different sizes of the training or fitting data.

models. The machine learning algorithms used for these experiments were: Kernel Ridge Regression (KRR), Decision Tree Regression (TREE) and Support Vector Machine Regression (SVR). Full details of the experiments can be

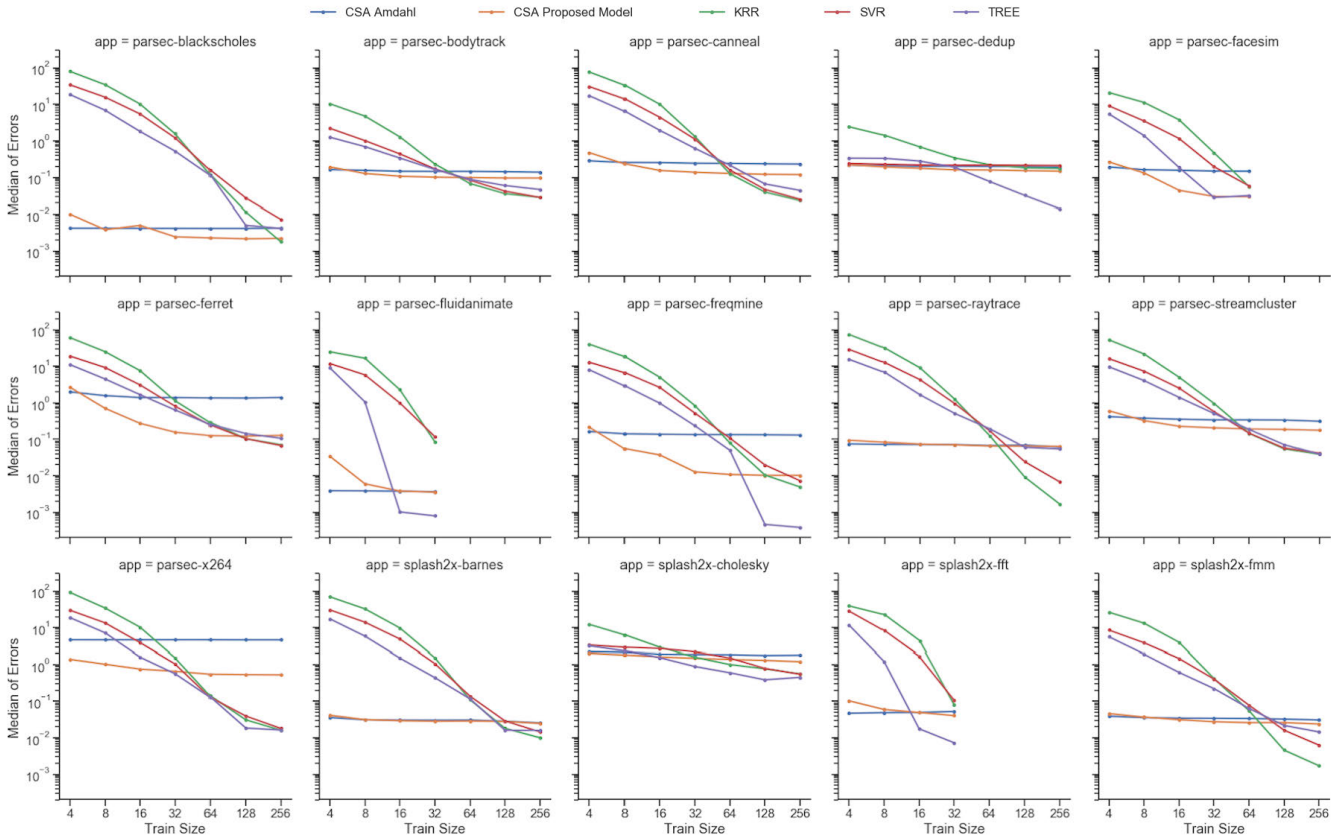


FIGURE 8. Median of the MSE, of the first 15 applications listed in Tab. 1, for 100 different model fittings using different sets of random measurements. KRR - Kernel Ridge Regression, SVR - Support Vector Machine Regression, TREE - Decision Tree Regression.

found in the open-source repository mentioned earlier. In the following, we describe the methodology used to evaluate accuracy and variance for the models under analysis: Amdahl's model (1) fitted with CSA; the proposed variable-delay model as given in (15) fitted with CSA; and the Machine Learning (ML) models. For Amdahl's model we fitted the parallel fraction f and for the proposed model we fitted f as well as the other new parameters k, m_1, and m_2.

For each number of samples, all measurement data were divided into a training or fitting set and a test set. The test set was always the remaining set of samples after removing the samples used to train or fit the models. The training or fitting for a given number of samples was repeated 100 times using each time a different set of random samples. All reported Mean Square Errors (MSEs) are the average of the MSE values of all 100 repetitions calculated using only the corresponding test sets. Fig. 7 illustrates the procedure used to compute the median of the MSE values for each set of 100 repetitions. The CSA method used 10 annealers limited to 30.000 iterations to fit the analytical models. The minimum and maximum limits of the model parameters were set to be between 0.0 and 1.0, for f, m_1 and m_2, and between 0.0 and 10.0 for k. For the KRR and SVR models we used the implementation of the Scikit-learn Python module [19]. The hyper-parameters of the Radial Base Function (RBF) kernel used in the KRR and SVR were tuned using a 3-fold cross-validation with a grid search that was repeated for each new set of random measurements. The search range for the error penalty parameters: C (SVR) and α (KRR), and the kernel coefficient γ (SVR and KRR) were $C = \{100, 1000\}$, $\alpha = \{10^0, 10^{-01}, 10^{-02}, 10^{-03}\}$ and $\gamma = \{10^{-05}, 10^{-04}, 10^{-03}, 10^{-02}, 10^{-01}, 10^0\}$.

Fig. 8 and Fig. 9 resumes all MSE results for each application using different numbers of measurements. The horizontal axis is in logarithmic scale and holds the number of sample measurements used to fit or to train the models: 4, 8, 16, 32, 64, 128, and 256 samples. Some applications restrict the number of cores that can be used, and thus, have fewer data points in the plots. For example, PARSEC Fluidanimate is limited to run only with numbers of cores that are a power of two. The last data point in the plot is always the power-of-two number immediately below the total number of measurements available for each application.

The main behavior observed in Fig. 8 and Fig. 9 is that the analytical models obtain lower mean squared errors as they use more measurements for modeling until they reach a plateau. Another important observation is that the analytical models have higher accuracy for smaller training sizes than the Machine Learning models. Although the Decision Tree model is generally more accurate for sets of measurements with more than 128 samples, the proposed model was overall more accurate for the smaller number of measurements,

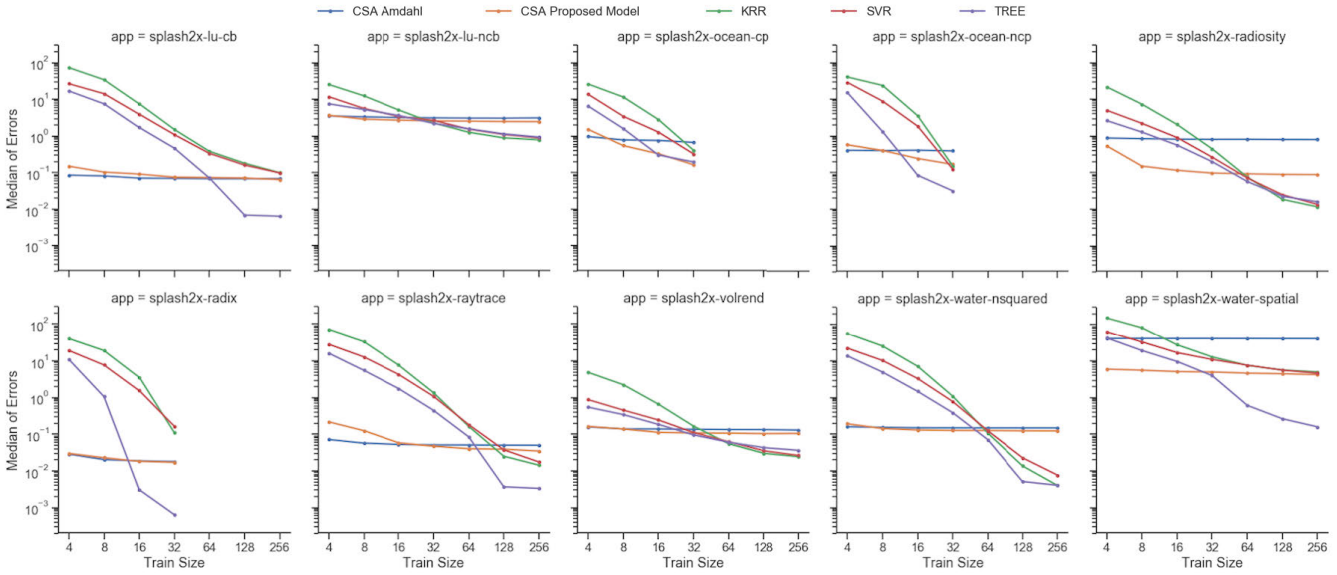


FIGURE 9. Median of the MSE, of the last 10 applications listed in Tab. 1, for 100 different model fittings using different sets of random measurements. KRR - Kernel Ridge Regression, SVR - Support Vector Machine Regression, TREE - Decision Tree Regression.

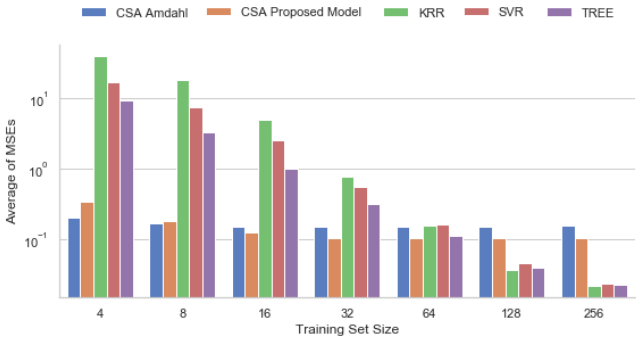


FIGURE 10. Average of all MSE values across all applications as function of the training set size. KRR - Kernel Ridge Regression, SVR - Support Vector Machine Regression, TREE - Decision Tree Regression.

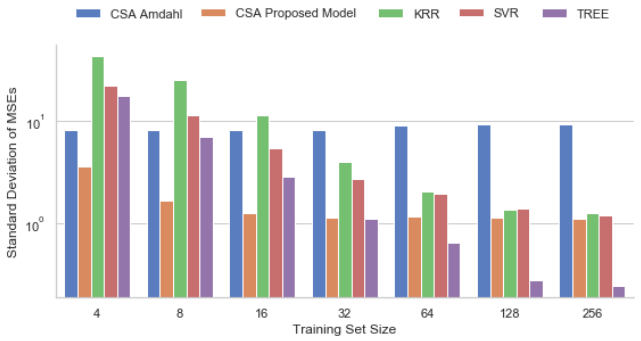


FIGURE 11. Standard deviation of all MSE values across all applications as function of the training set size. KRR - Kernel Ridge Regression, SVR - Support Vector Machine Regression, TREE - Decision Tree Regression.

except for size 4 and 8, for which Amdahl's models scored best in many cases. The reason for Amdahl's model scoring better than the proposed model for very small number of measurements is the same for the proposed model scoring better than Machine Learning models for midsize number of measurements: the more flexible the model is, i.e. the more parameters it has, the more information it requires to fit these parameters to the measured data while being sufficiently general.

The overall mean of the median MSE and standard deviation values of the five models across all applications according to the size of the sample set used in the modeling is depicted in Fig. 10 and Fig. 11.

Table 2 shows the time spent to model the speedups of each application using the proposed and using the Decision Tree model, which achieved the best results among the machine learning models analysed. The values reported for the proposed model refer to the number of points at which the accuracy of the proposed model surpasses the accuracy of Amdahl's model. For example, for the Blackscholes application, the proposed model shows better results when the training set size was at least 8 points. On the other hand, the values reported for the Decision Tree model refer to the number of points at which this Machine Learning model achieves higher accuracy than the proposed model. In this case, for Blackscholes, Decision Tree performs better only when 256 points or more are used for training. The table shows that the difference in time and, proportionally, in energy consumption between both models can often be around one order of magnitude. On average, considering all applications, the Decision Tree needed about three times longer to obtain more accurate results than the proposed model.

In contrast to the machine learning model, the architecturally-inspired models require only a few executions of the application to provide sufficiently good predictions of their speedups in configurations that were not previously assessed. This demonstrates an important advantage of these models, which allow an estimation of application performance for

TABLE 2. Time spend to collect applications measurements on specific number of points for each model. The column Δ% represents the percentage difference of times between the proposed model and decision tree model related to the proposed model.

	Proposed		Decision Tree	
Benchmark Program	points	time (s)	points	Δ%
parsec-blackscholes	8	1.8e+03	256	524.21%
parsec-bodytrack	8	1.97e+03	64	239.79%
parsec-canneal	8	1.96e+03	64	192.90%
parsec-dedup	8	360	64	317.75%
parsec-facesim	8	6.95e+03	64	251.87%
parsec-ferret	8	3.76e+03	128	309.25%
parsec-fluidanimate	32	1.26e+04	16	-33.69%
parsec-freqmine	8	5.67e+03	128	327.66%
parsec-raytrace	32	5.02e+03	128	77.97%
parsec-streamcluster	8	8.58e+03	64	198.05%
parsec-x264	4	826	32	283.11%
splash2x-barnes	16	3e+03	128	154.25%
splash2x-cholesky	4	0.719	16	196.13%
splash2x-fft	16	1.1e+03	16	0.00%
splash2x-fmm	16	2.33e+03	128	174.86%
splash2x-lu-cb	256	1.1e+10	128	-34.52%
splash2x-lu-ncb	16	3.03e+09	32	47.32%
splash2x-ocean-cp	8	2.41e+03	16	63.76%
splash2x-ocean-ncp	16	6.52e+03	16	0.00%
splash2x-radiosity	4	746	128	748.20%
splash2x-radix	16	1.24e+03	16	0.00%
splash2x-raytrace	32	6.51e+03	128	75.29%
splash2x-volrend	8	1.65e+03	64	281.42%
splash2x-water-nsquared	8	6.04e+03	64	198.94%
splash2x-water-spatial	4	1.45e+03	32	303.15%
Mean	22.08		76.80	195.91%

unseen configurations of a given architecture with reduced overheads of time and energy. On the other hand, if more sampling points are available, machine learning models provide better accuracy at the cost of a higher overhead.

The results demonstrate that there is space for the use of analytical models as opposed to the use of traditional Machine Learning-based models. Machine learning models do achieve higher accuracy when using a more representative data set for training. However, they fail to explain the behavior and features of the applications and their relation to hardware characteristics. In turn, analytical models require fewer data points to achieve accuracy similar to what Machine Learning can only achieve when using far more data points for training. Moreover, analytical models facilitate the understanding of the interplay between the hardware properties and the application behavior, which makes their use important for software and hardware development.

V. RELATED WORK

Inspired by earlier analytical models, such as [1]–[3], many more recent models attempt to capture better the behavior of application and architecture features that describe parallel speedups more precisely. None of them, however, consider the effect of the memory wall [7] on parallel speedups as considered in this work.

Analytical speedup models for multi-core processors were devised to describe communication [20] and synchronization [21] overhead separately. Communication and synchronization overheads were modeled together in [22] providing a more general description of both behaviors. Apart from not considering the effect of the memory wall on the modeled speedups, no hardware or simulation validation was presented to confirm their results.

Other analytical models for multi-core architectures consider the variations in parallel speedups caused by variations in the problem or input size, including the modeling of the parallel overhead [23] or not [24]. The parallel overhead was also modeled together with the parallel speedup for distributed parallelism in [25]. Similar to our work, these studies also validated the models using execution time measurements, but no feature was associated with the effect of the memory wall.

The work of Liu and Sun [26] combines the limitations related to the finite size of the memory [3] with memory access concurrency [27] to provide a speedup model that can be used for multi-core design space exploration. Although this model contains elements that relate to our data-access delay speedup model, the authors focus on chip design and perhaps, for this reason, do not explore the effects of frequency variations on speedups.

The roofline model [28] introduced a simple model for visualization of actual and attainable performance in the compute- and memory-bounded regions. The model uses the number of operations per byte of DRAM traffic as a metric. It considers only the bandwidth between main memory and Last Level Cache (LLC). More recently, the cache-aware roofline model [29] extended the roofline model to include byte traffic between the cores, the various cache levels, and the main memory, which in fact is a generalization of the original roofline model. Both models are very useful to help finding architectural bottlenecks and which code optimizations should be applied to achieve better performance on specific hardware architecture. However, these models did not analyze the relationships between the operating frequency and the speedup of applications.

Therefore, to the best of our knowledge, this work is the first to explore the effect of operating frequency on the the speedup of parallel applications running on shared memory platforms. For this reason, the only model mentioned in this section that we used for comparison was the original Amdahl's model, as many of the other works did. Moreover, since those models differ from Amdahl's by aspects that were kept fixed in our experiments, such as the problem size and architectural features like memory hierarchy and the amount of available memory, other comparisons would not be relevant to this study.

VI. CONCLUSION

We have presented a new modeling approach for estimating speedups of parallel applications that are subject to the limitations of the memory wall. The proposed modeling considers variations in the data-access delay of the main memory when the number of cores increases and when the processor's or memory's operating frequency change; capturing the effect of changing the ratio between the processor's and the memory's frequencies. To the best of our knowledge, this behavior was not described by previous analytical speedup models.

Several hardware experiments presented in this paper validate the ability of the proposed models to describe the memory wall behavior for many different applications.

Our analysis shows that reducing processor frequency reduces the adverse effect of the memory wall on parallel speedups, suggesting that there could be an optimal processor frequency for each number of cores used to run a given application. Therefore, we argue that this work is not a pessimistic view of multi-core scalability. Instead, it shows that the race toward single-core performance under the influence of Amdahl's Law has perhaps obfuscated a more efficient way to match processor and memory frequencies for parallel applications. That is undoubtedly true if the focus is energy efficiency; as such models could be applied, for example, to devise better Dynamic Voltage and Frequency Scaling (DVFS) schemes for the Internet of Things [30], data centers [31], and high-performance computing [32].

Ideally, these new DVFS schemes may also consider the number of cores used by the application, such as in [33], [34]. To be practical for this, the speedup models need to be able to predict performance at non-visited configurations with the smallest possible number of measurements. In this sense, we showed that, based on only about a dozen measurements, the proposed model can produce predictions that are as accurate as those obtained from three Machine Learning regression algorithms after training with at least a hundred measurements. On average, our model achieved higher accuracy than Amdahl's model when using more than eight random measurements and also achieved higher accuracy than Decision Tree regression when using 64 random measurements or less. The standard deviation of our modeling was lower than Amdahl's model for all measurements, and was lower than Decision Tree regression for 32 random measurements or less.

In contrast with Machine Learning speedup models, the proposed model holds an inherent mapping of the application features, such as rate of memory versus processor instructions and the value of the parallel and serial fractions of the code, which is often relevant to software and hardware development. In its turn, machine learning schemes, such as Decision Tree Regression, work as black boxes with relations between model parameters and applications behavior that are hard to infer. Additionally, evaluating analytical models is also faster, which makes it suitable for use in on-line performance and/or energy optimization schemes.

Despite the many different existing models for parallel speedups, the practical use of these models requires both better generalization and a lower fitting overhead. In this work, we have made contributions to both aspects, but there is still room for further improvements. For example, to make the model more general, the modeling of problem size could be included. For reducing fitting overhead, devising a heuristic to choose the initial measurements might work better than random sampling, as it has been observed in [35]. For on-line fitting, increasing the complexity of the models as the number of measurements increases might also reduce fitting overhead. Extending this approach to speedup models for heterogeneous systems [11] is also promising, as the use of these systems has grown substantially in recent years.

ACKNOWLEDGMENT

The authors would like to thank the Center for Information Services and High Performance Computing (ZIH) at TU Dresden for generous allocations of computer time.

REFERENCES

[1] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *Proc. Spring Joint Comput. Conf.*, Atlantic City, NJ, USA, 1967, pp. 483–485.

[2] J. L. Gustafson, "Reevaluating Amdahl's law," *Commun. ACM*, vol. 31, no. 5, pp. 532–533, May 1988.

[3] X. H. Sun and L. M. Ni, "Scalable problems and memory-bounded speedup," *J. Parallel Distrib. Comput.*, vol. 19, no. 1, pp. 27–37, Sep. 1993. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0743731583710877

[4] Y. Shi. (1996). *Reevaluating Amdahl's Law and Gustafson's Law*. [Online]. Available: https://www.researchgate.net/profile/Yuan_Shi12/publication/228367369_Reevaluating_Amdahl's_law_and_Gustafson's_law/links/562f9dd408ae8e1256876a0a.pdf

[5] M. D. Hill and M. R. Marty, "Amdahl's law in the multicore era," *Computer*, vol. 41, no. 7, pp. 33–38, Jul. 2008.

[6] 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. [Online]. Available: http://www.mendeley.com/catalog/reevaluating-amdahls-law-multicore-era/

[7] W. A. Wulf and S. A. McKee, "Hitting the memory wall: Implications of the obvious," *ACM SIGARCH Comput. Archit. News*, vol. 23, no. 1, pp. 20–24, Mar. 1995, doi: 10.1145/216585.216588.

[8] X. Wu and V. Taylor, "Utilizing hardware performance counters to model and optimize the energy and performance of large scale scientific applications on power-aware supercomputers," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshops (IPDPSW)*, May 2016, pp. 1180–1189.

[9] M. A. N. 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 *Proc. Int. Conf. High Perform. Comput. Simul. (HPCS)*, Jul. 2017, pp. 410–417.

[10] X. Zheng, P. Ravikumar, L. K. John, and A. Gerstlauer, "Learning-based analytical cross-platform performance prediction," in *Proc. Int. Conf. Embedded Comput. Systems: Architectures, Model., Simul. (SAMOS)*, Jul. 2015, pp. 52–59.

[11] C. A. Barros, L. F. Q. Silveira, C. A. Valderrama, and S. Xavier-de-Souza, "Optimal processor dynamic-energy reduction for parallel workloads on heterogeneous multi-core architectures," *Microprocessors Microsyst.*, vol. 39, no. 6, pp. 418–425, Aug. 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0141933115000617

[12] S. Xavier-de-Souza, C. A. Barros, M. O. Jales, and L. F. Q. Silveira, "Not faster nor slower tasks, but less energy hungry and parallel: Simulation results," in *Proc. 4th Berkeley Symp. Energy Efficient Electron. Syst. (E3S)*, Oct. 2015, pp. 1–3. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7336814

[13] S. Xavier-de Souza, C. A. Barros, L. F. Q. Silveira, C. A. Valderrama, and R. A. Petta, "Estimating the effects of application speedup on energy saving for lower-voltage and lower-frequency multi-core devices," in *Proc. 3rd Berkeley Symp. Energy Efficient Electron. Syst. (E3S)*, Oct. 2013, pp. 1–8.
[14] C. Bienia, "Benchmarking modern multiprocessors," Ph.D. dissertation, Dept. Comput. Sci., Princeton Univ., Princeton, NJ, USA, 2011.
[15] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, A. Gupta, S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The SPLASH-2 programs," in *Proc. 22nd Annu. Int. Symp. Comput. Archit.*, vol. 23, New York, New York, USA, 1995, pp. 24–36. [Online]. Available: http://portal.acm.org/citation.cfm?doid=223982.223990
[16] S. Xavier-de-Souza, J. A. K. Suykens, J. Vandewalle, and D. Bolle, "Coupled simulated annealing," *IEEE Trans. Syst., Man, Cybern. B. Cybern.*, vol. 40, no. 2, pp. 320–335, Apr. 2010.
[17] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC benchmark suite: Characterization and architectural implications," in *Proc. 17th Int. Conf. Parallel Archit. Techn. (PACT)*, New York, NY, USA, 2008, pp. 72–81, doi: 10.1145/1454115.1454128.
[18] G. Southern and J. Renau, "Analysis of PARSEC workload scalability," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw. (ISPASS)*, Apr. 2016, pp. 133–142.
[19] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, Oct. 2011.
[20] T. Huang, Y. Zhu, M. Qiu, X. Yin, and X. Wang, "Extending Amdahl's law and Gustafson's law by evaluating interconnections on multi-core processors," *J. Supercomput.*, vol. 66, no. 1, pp. 305–319, Oct. 2013, doi: 10.1007/s11227-013-0908-9.
[21] S. Eyerman and L. Eeckhout, "Modeling critical sections in Amdahl's law and its implications for multicore design," *ACM SIGARCH Comput. Archit. News*, vol. 38, no. 3, pp. 362–370, Jun. 2010, doi: 10.1145/1816038.1816011.
[22] L. Yavits, A. Morad, and R. Ginosar, "The effect of communication and synchronization on Amdahl's law in multicore systems," *Parallel Comput.*, vol. 40, no. 1, pp. 1–16, Jan. 2014, doi: 10.1016/j.parco.2013.11.001.
[23] V. H. F. Oliveira, A. F. A. Furtunato, L. F. Silveira, K. Georgiou, K. Eder, and S. Xavier-de-Souza, "Application speedup characterization: Modeling parallelization overhead and variations of problem size and number of Cores.," in *Proc. Companion ACM/SPEC Int. Conf. Perform. Eng.*, New York, NY, USA, Apr. 2018, pp. 43–44, doi: 10.1145/3185768.3185770.
[24] S. Narayanan, B. N. Swamy, and A. Seznec, "An empirical high level performance model for future many-cores," in *Proc. 12th ACM Int. Conf. Comput. Frontiers*, New York, NY, USA, 2015, pp. 1–8, doi: 10.1145/2742854.2742867.
[25] S. Höfinger and E. Haunschmid, "Modelling parallel overhead from simple run-time records," *J. Supercomput.*, vol. 73, no. 10, pp. 4390–4406, Oct. 2017, doi: 10.1007/s11227-017-2023-9.
[26] Y.-H. Liu and X.-H. Sun, "Evaluating the combined effect of memory capacity and concurrency for many-core chip design," *ACM Trans. Model. Perform. Eval. Comput. Syst.*, vol. 2, no. 2, pp. 1–25, May 2017, doi: 10.1145/3038915.
[27] X.-H. Sun and D. Wang, "Concurrent Average Memory Access Time," *Computer*, vol. 47, no. 5, pp. 74–80, May 2014.
[28] S. Williams, A. Waterman, and D. Patterson, "Roofline: An insightful visual performance model for multicore architectures," *Commun. ACM*, vol. 52, no. 4, pp. 65–76, Apr. 2009, doi: 10.1145/1498765.1498785.
[29] A. Ilic, F. Pratas, and L. Sousa, "Cache-aware roofline model: Upgrading the loft," *IEEE Comput. Archit. Lett.*, vol. 13, no. 1, pp. 21–24, Jan. 2014.
[30] K. Georgiou, S. Xavier-de-Souza, and K. Eder, "The IoT energy challenge: A software perspective," *IEEE Embedded Syst. Lett.*, vol. 10, no. 3, pp. 53–56, Sep. 2018. [Online]. Available: http://ieeexplore.ieee.org/document/8012513/
[31] A. Pahlevan, J. Picorel, A. Pourhabibi Zarandi, D. Rossi, M. Zapater, A. Bartolini, P. G. Del Valle, D. Atienza, L. Benini, and B. Falsafi, "Towards near-threshold server processors," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, 2016, pp. 7–12.
[32] V. R. G. Silva, A. Furtunato, K. Georgiou, K. Eder, and S. Xavier-de-Souza, "Energy-optimal configurations for single-node HPC applications," 2018, *arXiv:1805.00998*. [Online]. Available: http://arxiv.org/abs/1805.00998
[33] D. De Sensi, T. De Matteis, and M. Danelutto, "Simplifying self-adaptive and power-aware computing with nornir," *Future Gener. Comput. Syst.*, vol. 87, pp. 136–151, Oct. 2018. . [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X17326699
[34] A. F. Lorenzon, M. C. Cera, and A. C. S. Beck, "Investigating different general-purpose and embedded multicores to achieve optimal trade-offs between performance and energy," *J. Parallel Distrib. Comput.*, vol. 95, pp. 107–123, Sep. 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0743731516300090
[35] D. De Sensi, "Predicting performance and power consumption of parallel applications," in *Proc. 24th Euromicro Int. Conf. Parallel, Distrib., Network-Based Process. (PDP)*, Feb. 2016, pp. 200–207.

ALEX F. A. FURTUNATO received the B.Sc. degree in automation and computer engineering from the University of Rio Grande do Norte, in 1997. His research interests include high-performance computing, security and cryptography, computer networks, systems software, and distributed computing.

KYRIAKOS GEORGIOU received the B.Sc. degree in computer science from the University of Cyprus, and the M.Sc. degree in Internet Technologies with security and the Ph.D. degree from the University of Bristol. He is currently a Senior Research Associate with the Trustworthy System Laboratory, University of Bristol, where he has been researching a broad spectrum of ICT subjects, including energy-aware computing, execution time and energy consumption modeling, compiler auto-tuning, and software engineering. He has previously worked in the industry for two years as a software developer for financial services, and as compiler engineer for three years.

KERSTIN EDER received the M.Eng. degree in informatics from Technical University Dresden, Germany, and the M.Sc. degree in artificial intelligence and the Ph.D. degree in computational logic from the University of Bristol, U.K. She is currently a Professor of computer science and leads the Trustworthy Systems Laboratory, University of Bristol, and the Verification and Validation for Safety in Robots research theme at the Bristol Robotics Laboratory. Her research is focused on specification, verification and analysis techniques to verify or explore a system's behavior in terms of functional correctness, safety, performance, and energy efficiency. Her initiated research into Energy Aware COmputing (EACO) at Bristol during her Royal Academy of Engineering funded Industrial Secondment, in 2010.

SAMUEL XAVIER-DE-SOUZA (Senior Member, IEEE) was born in Natal, Brazil. He received the Computer Engineering degree from the Universidade Federal do Rio Grande do Norte-UFRN, Brazil, in 2000, and the Ph.D. degree in electrical engineering from Katholieke Universiteit Leuven, Belgium, in 2007. He worked for IMEC, Belgium, as a Software/Hardware Engineer, and for the Flemish Supercomputing Center, Belgium, as a High-Performance Computing Consultant. In 2009, he joined the Department of Computer Engineering and Automation of UFRN, where he currently holds the position of an Associate Professor. He is also a Founder and the Chief Coordinator of UFRN's High-Performance Computing Center-NPAD. In 2016, he became a Royal Society-Newton Advanced Fellow. His research interests include software energy, scalable and efficient parallel systems, parallel algorithms, parallel architectures, and their applications.