Furtunato, A. F. A., Georgiou, K., Eder, K., & Xavier-de-Souza, S. (2019). When parallel speedups hit the memory wall. *arXiv*.

Early version, also known as pre-print

When parallel speedups hit the memory wall

Alex F. A. Furtunato[1], Kyriakos Georgiou[2], Kerstin Eder[2], Samuel Xavier-de-Souza[1]

[1] Universidade Federal do Rio Grande do Norte, Brazil
[2] University of Bristol, UK

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 new parallel speedup models that account for variations of the average data-access delay to describe the limiting effect of the memory wall on parallel speedups. 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 degree of parallelism and susceptibility to the memory wall, our proposal has significant advantages over machine-learning-based modeling. Moreover, besides being black-box modeling, our experiments show that conventional machine-learning modeling needs about one order of magnitude more measurements to reach the same level of accuracy achieved in our modeling.

1 Introduction

Amdahl's Law [Amd67] 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 [Gus88, SN93, Shi96, HM08, SC10]. 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 of multiple processors to scale the application's performance [Amd67]. 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 [Gus88]. 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 [Shi96], 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 [SN93] 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 [HM08]. In [SC10], 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 [WM95], 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: 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 model to capture these effects altogether.

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 Section 2).

We initially investigate the potential abilities of our model to capture the above effects analytically (Section 3). 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 (Section 4). Furthermore, we demonstrate that our approach has higher accuracy and lower variance than Amdahl's model (Section 4.2). Comparisons to other analytical speedup models would not be more relevant since the other existing models differ from Amdahl's model by aspects that were kept constant in our experiments, such as the problem size and architectural features like memory hierarchy and the amount of memory available.

These behavioral aspects are orthogonal to the memory wall aspect and complement our work.

We compare our model proposition to a non-linear machine learning regression approach (Section 4.3), which is arguably 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 2 we present our modeling for speedup as a function of the ratio between processor and memory frequencies. In Section 3 we analyze the model behavior. In Section 4, we detail the methodology used to validate the proposed models and provide results of experiments in real hardware. In Section 5 we put our contributions in perspective with the existing literature and, finally, in Section 6, we draw conclusions and suggest future work.

2 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:

$$S_p = T_\mathrm{s}/T_\mathrm{p}, \tag{1}$$

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 assumptions, necessary to devise the proposed model. These are later proved to be satisfactorily sustained by the model validation presented in Section 4:

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. 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 memory system is a centralized entity and serves all the processing cores uniformly, which reassembles most of current multi-core architectures.

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

$$T_\mathrm{s} = t_c C + t_m M. \tag{3}$$

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 [Amd67] 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

$$T_\mathrm{p} = (1 - f)T_\mathrm{s} + f\frac{T_\mathrm{s}}{p}. \tag{4}$$

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.

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

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.

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

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,

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

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 Section 1. 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.

3 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 Section 4, 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.

3.1 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.

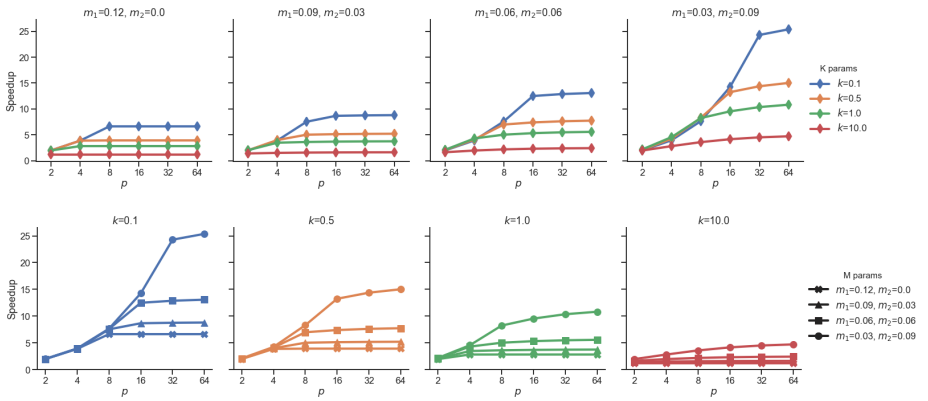


Fig. 1. Speedup plots for a computational task with parallel fraction $f = 0.99$, frequencies ratio $\phi = 3.0$ and a varying number of cores $p = \{2, 4, 8, 12, 16, 32, 64\}$. Each plot and curves refers to combinations of k and m parameters. For k plots, the curves represent different m parameters,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

As Fig. 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.

3.2 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.

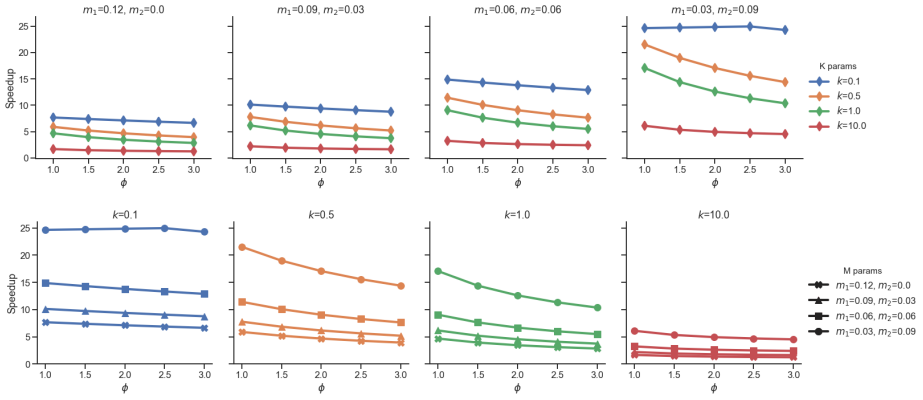


Fig. 2. Speedup plots for computational tasks varying the ratio between the processor and memory frequencies $\phi = [1.0,\ 1.4,\ 1.8,\ 2.2,\ 2.4,\ 3.0]$, with number of cores $p = 32$ and parallel fraction $f = 0.99$. Each plot and curves refers to combinations of k and m parameters. For plots by k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

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.

4 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.

4.1 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. For our validation, applications from the PARSEC [Bie11] and SPLASH-2 [WOT+95] parallel benchmark suites have been used. 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.

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 Particle Swarm Optimization (PSO) [KE95] global optimization method to minimize the Mean Squared Error (MSE) between the measured application speedups and their models. The PSO algorithm used was the version with the coefficient of constriction [CK02].

To vary the ratio between processor frequency and memory frequency, we changed the processor's frequency while the operating frequency of the memory system was kept at a fixed value.

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 with hardware multi-threading disabled, 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.

A Python version 3 library was developed[3] to implement the PSO 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 Section 4.2, we assessed 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 4.3, we investigate how the accuracy of these models and the accuracy of an unstructured machine learning model vary according to the amount of information used to construct them.

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

Table 1. Model parameters and MSE for Amdahl's model and for the proposed model for the PARSEC and the SPLASH2 benchmark 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	0.9239	10.0000	0.0003	0.8761	0.0021	49.43 %
parsec-bodytrack	322	0.8934	0.1417	0.6964	10.0000	0.0373	0.2454	0.0945	33.31 %
parsec-canneal	322	0.9985	0.2325	0.9982	0.4376	0.0057	0.8556	0.1124	51.66 %
parsec-dedup	322	0.6745	0.1969	0.7387	0.1561	0.3205	0.0000	0.1481	24.82 %
parsec-facesim	84	0.9731	0.1443	0.9734	0.0852	0.0447	0.6742	0.0220	84.74 %
parsec-ferret	322	0.9912	1.3371	0.9949	0.2422	0.0363	0.1681	0.1114	91.67 %
parsec-fluidanimate	56	0.9834	0.0036	0.9984	0.0000	0.0173	0.9728	0.0029	19.26 %
parsec-freqmine	322	0.9791	0.1316	0.9915	0.0000	0.0294	0.8215	0.0097	92.64 %
parsec-raytrace	322	0.9959	0.0675	0.9016	10.0000	0.0039	0.7799	0.0623	7.71 %
parsec-streamcluster	322	0.9860	0.3274	0.8188	9.5443	0.0077	0.3525	0.1788	45.40 %
parsec-x264	322	1.0000	4.6452	1.0000	0.4837	0.0152	0.4200	0.4944	89.36 %
splash2x-barnes	322	0.9969	0.0290	0.9360	5.8737	0.0029	1.0000	0.0268	7.80 %
splash2x-cholesky	322	0.8978	1.8236	0.9274	0.1330	0.1293	0.0000	1.2996	28.73 %
splash2x-fft	56	0.9999	0.0436	0.6850	10.0000	0.0013	0.7152	0.0377	13.61 %
splash2x-fmm	322	0.9629	0.0326	0.7545	10.0000	0.0262	0.6271	0.0253	22.38 %
splash2x-lu-cb	322	0.9950	0.0668	0.8822	10.0000	0.0049	0.9270	0.0664	0.53 %
splash2x-lu-ncb	322	0.9538	3.0182	0.8492	10.0000	0.0124	0.1426	2.5273	16.27 %
splash2x-ocean-cp	56	0.9769	0.6297	0.8710	10.0000	0.0093	0.2049	0.3457	45.10 %
splash2x-ocean-ncp	56	1.0000	0.3854	0.9011	10.0000	0.0026	0.2958	0.1787	53.64 %
splash2x-radiosity	322	0.9408	0.8001	0.9673	0.1187	0.0942	0.0404	0.0844	89.46 %
splash2x-radix	56	0.9961	0.0172	0.8842	4.9551	0.0036	1.0000	0.0156	9.21 %
splash2x-raytrace	322	0.9973	0.0493	1.0000	0.0000	0.0056	0.9403	0.0281	42.98 %
splash2x-volrend	308	0.7060	0.1251	0.2340	8.0727	0.2719	1.0000	0.0815	34.79 %
splash2x-water-nsquared	322	0.9892	0.1468	0.7221	9.6431	0.0103	1.0000	0.1243	15.34 %
splash2x-water-spatial	322	1.0000	41.7510	0.9943	1.7865	0.0021	0.2746	4.0742	90.24 %

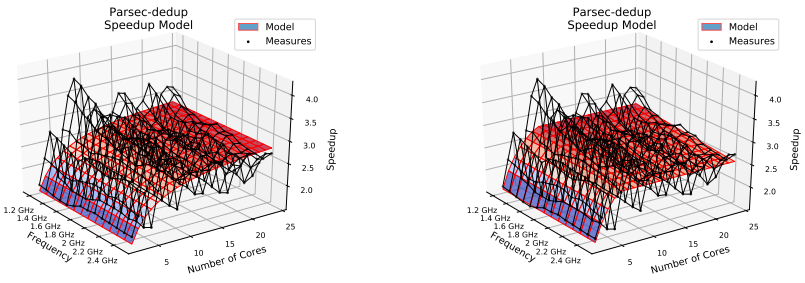
4.2 Model accuracy

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 0.5% 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 42.40% 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 [BKSL08], 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 [SR16]. 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.

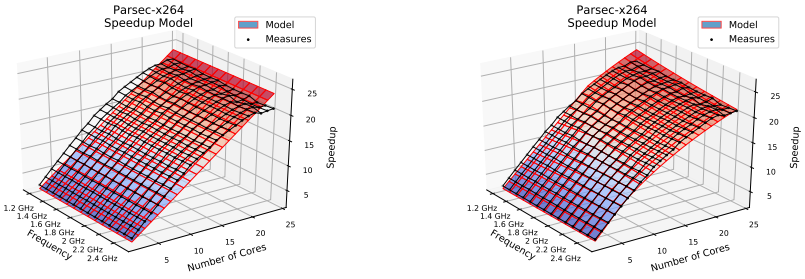


(a) Modeled by Amdahl (5). (b) Proposed model (15).

Fig. 3. Amdahl's and proposed models for the PARSEC Dedup application.

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.

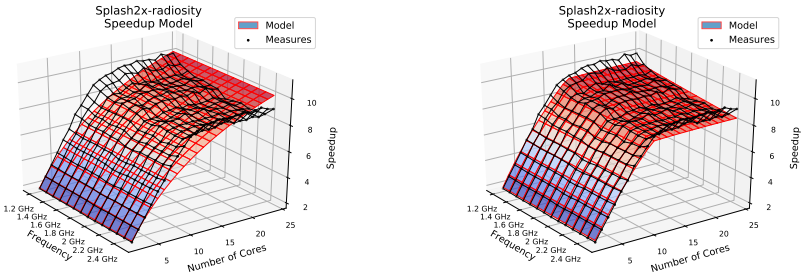
Fig. 5 presents the models for the SPLASH-2 Radiosity application. It computes the equilibrium distribution of light in a scene [WOT+95]. One of the computational characteristics of this algorithm is a large number of memory instructions and, therefore, it is an appropriate case study to prove the proposed 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 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.



(a) Modeled by Amdahl (5).

(b) Proposed model (15).

Fig. 4. Amdahl's and proposed model for the PARSEC X264 application.



(a) Modeled by Amdahl (5).

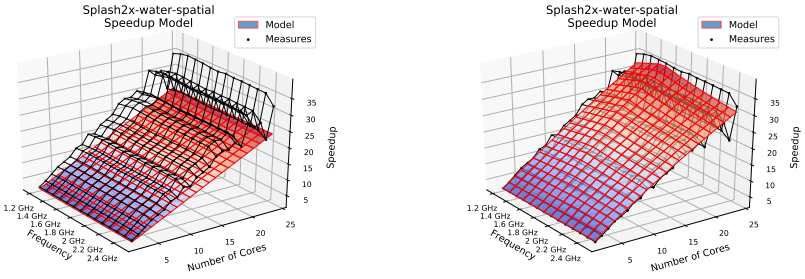
(b) Proposed model (15).

Fig. 5. Amdahl's and proposed model for the SPLASH-2 Radiosity application.

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.

4.3 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,



(a) Modeled by Amdahl (5).

(b) Proposed model (15).

Fig. 6. Amdahl's and proposed model for the SPLASH-2 Water Spatial application.

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 a machine learning technique called Support Vector Machine Regression (SVR) [SS04] to model the applications using the same inputs used to fit the analytical models. Full details of the experiments can be found in the open-source repository mentioned earlier. In the following, we describe the methodology used to evaluate accuracy and variance for the three models under analysis: Amdahl's model (1) fitted with PSO; the proposed variable-delay model as given in (15) fitted with PSO; and the SVR model. 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 PSO method used 200 particles limited to 100 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 SVR model we used the implementation of the Scikit-learn Python module [PVG+11a]. The hyper-parameters of the Radial Base Function (RBF) kernel [PVG+11b] used in the 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 parameter C and the kernel coefficient γ were $C = \{100, 1000\}$ and $\gamma = \{10^{-05}, 10^{-04}, 10^{-03}, 10^{-02}, 10^{-01}, 1.0\}$.

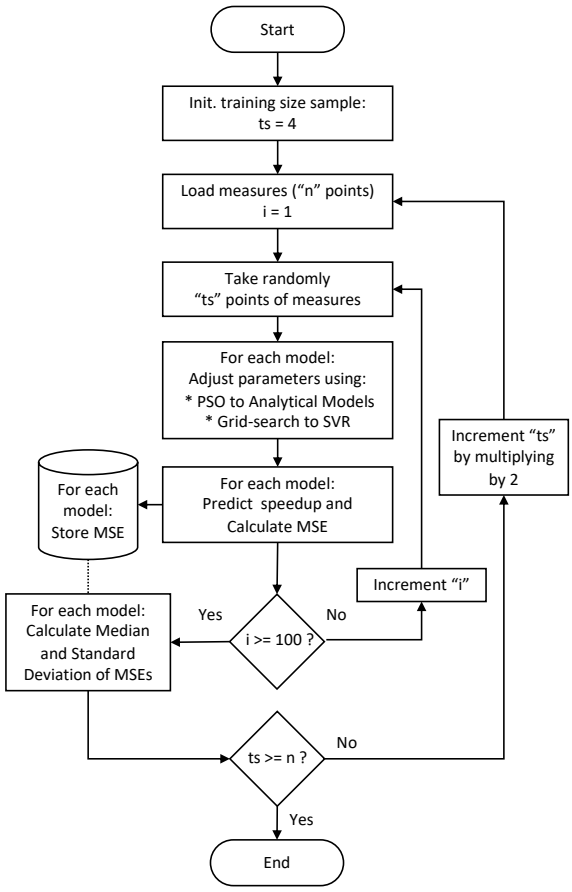


Fig. 7. Flow chart of procedure used to compute the median and the standard deviation of the MSE for each model using different sizes of the training or fitting data.

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.

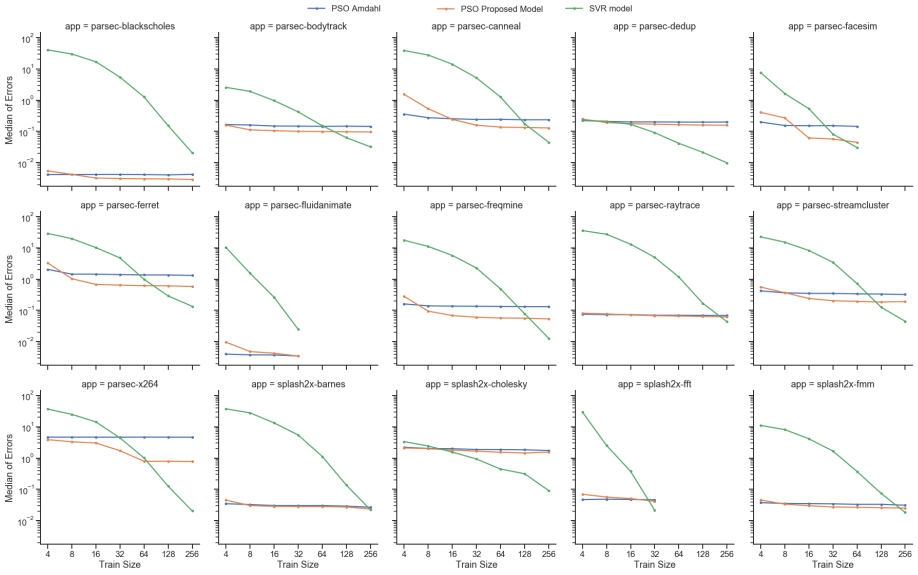


Fig. 8. Median of the MSE, of the first 15 applications listed in Tab. 1, for 100 different model fittings using different sets of random measurements.

Table 2 shows the time spent to model the speedups of each application using the proposed and the SVR models. 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 Canneal application, the proposed model shows better results when the training set size was at least 16 points. On the other hand, the values reported for the SVR Model refer to the number of points at which the the SVR model achieves higher accuracy than the proposed model. In this case, for Canneal, SVR performs better only after 256 points are being 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

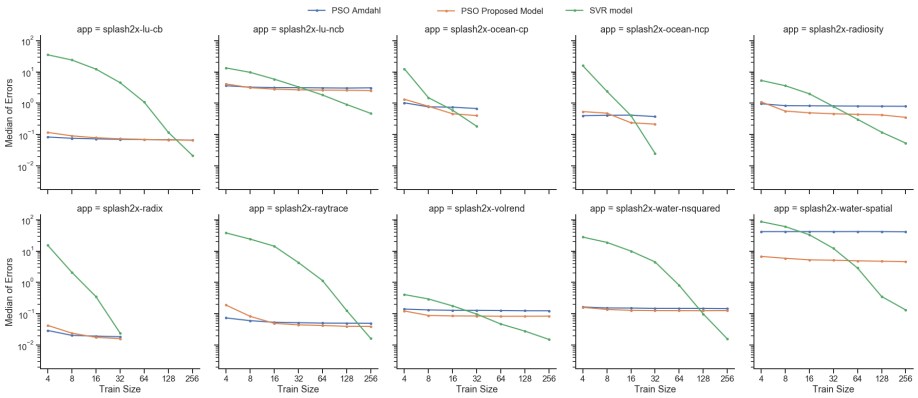


Fig. 9. Median of the MSE, of the last 10 applications listed in Tab. 1, for 100 different model fittings using different sets of random measurements.

applications, the SVR needed 293.27% more time to obtain better results than the proposed model.

The main behavior observed in Fig. 8 and Fig. 9 is that the analytical models obtain better results as they use more measurements for modeling until they reach a plateau. Another important observation is that all analytical models have better accuracy for smaller training sizes than the SVR model. Although the SVR model is generally more accurate for sets of measurements with more than 128 samples, the proposed model was overall better for the smaller-number sets except for size 4 and 8, for which Amdahl's models scored best in many cases.

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

In contrast to the machine learning model, the architecturally-inspired models require only a few executions of the application to provide reasonably good predictions of their speedups in configurations that were not previously assessed. This demonstrates an important advantage of these models, which allows an estimation of application performance for unseen configurations of a given architecture with reduced overheads of time and energy. On the other hand, if more sampling points are available, SVR provides better accuracy at the cost of a higher overhead.

5 Related Work

Inspired by earlier analytical models, such as [Amd67, Gus88, SN93], many more recent models attempt to capture better the behavior of application and architecture features that describe parallel speedups more precisely. None of them,

Table 2. Time spend to collect applications measurements on specific number of points for each of Proposed and SVR model.

	Proposed		SVR	
Benchmark Program	points	time (s)	points	time (s)
parsec-blackscholes	16	2.9e+03	256	1.1e+04
parsec-bodytrack	4	1.1e+03	128	1e+04
parsec-canneal	16	3.1e+03	256	1.2e+04
parsec-dedup	8	3.6e+02	16	6.1e+02
parsec-facesim	16	1.2e+04	64	2.4e+04
parsec-ferret	8	3.7e+03	128	1.5e+04
parsec-fluidanimate	32	1.3e+04	32	1.3e+04
parsec-freqmine	8	5.7e+03	256	3.8e+04
parsec-raytrace	16	3.5e+03	256	1.4e+04
parsec-streamcluster	16	1.4e+04	128	3.5e+04
parsec-x264	4	8e+02	128	5.3e+03
splash2x-barnes	8	1.8e+03	256	1.2e+04
splash2x-cholesky	4	0.71	16	2.1
splash2x-fft	32	1.6e+03	32	1.6e+03
splash2x-fmm	8	1.5e+03	256	1e+04
splash2x-lu-cb	64	5.2e+09	256	1.1e+10
splash2x-lu-ncb	8	1.8e+09	64	6.1e+09
splash2x-ocean-cp	16	4.1e+03	32	5.9e+03
splash2x-ocean-ncp	16	6.4e+03	32	9.6e+03
splash2x-radiosity	8	1.4e+03	64	4.4e+03
splash2x-radix	16	1.2e+03	32	1.9e+03
splash2x-raytrace	16	4.6e+03	256	1.7e+04
splash2x-volrend	4	8.8e+02	64	6e+03
splash2x-water-nsquared	4	3.4e+03	128	2.5e+04
splash2x-water-spatial	4	1.5e+03	64	7.3e+03

however, consider the effect of the memory wall [WM95] on parallel speedups as considered in this work.

Analytical speedup models for multi-core processors were devised to describe communication [HZQ+13] and synchronization [EE10] overhead separately. Communication and synchronization overheads were modeled together in [YMG14] providing a more general description of both behaviors. Apart from

Fig. 10. Average of all MSE values across all applications as function of the training set size.

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 [OFS+18] or not [NSS15]. The parallel overhead was also modeled together with the parallel speedup for distributed parallelism in [HH17]. 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 [LS17] combines the limitations related to the finite size of the memory [SN93] with memory access concurrency [SW14] 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.

Therefore, to the best of our knowledge, this work is the first to explore this effect. 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 memory available, other comparisons would not be relevant to this study.

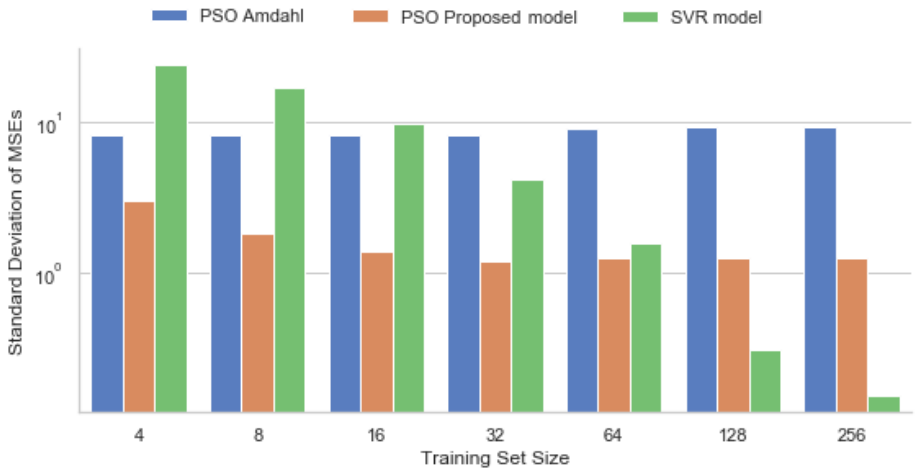


Fig. 11. Standard deviation of all MSE values across all applications as function of the training set size.

6 Conclusions

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 in 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 multicore 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 [GXdSE17], data centers [PPZ+16], and high-performance computing [SFG+18].

Ideally, these new DVFS schemes may also consider the number of cores used by the application, such as in [DSDMD18, LCB16]. 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 the proposed model can reach a level of accuracy with about a dozen of measurements that Support Vector Regression can only reach with hundreds of measurements. On average, our modeling presented higher accuracy than Amdahl's model, when using more than 8 random measurements, and than support vector regression, when using 128 random measurements or less. The standard deviation of our modeling was better than Amdahl's model for all number of measurements, and was better than Support Vector Regression for 64 random measurements or less.

In contrast with ML 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 Support Vector 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 [Sen16]. For on-line fitting, increasing the complexity of the models as the number of measurements increases might also reduce fitting overhead. Extending this approach for speedup models in heterogeneous systems [BSVXdS15] is also promising, as the use of these systems has grown substantially in recent years.

Acknowledgment

This work was supported by High-Performance Computing Center at UFRN (NPAD/UFRN) and financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001, and in part by the Royal Society-Newton Advanced Fellowship award no. NA160108. It is also supported by the European-Union's Horizon 2020 Research and Innovation Programme under grant agreement No. 779882, TeamPlay (Time, Energy and security Analysis for Multi/Many-core heterogeneous PLAtforms). We also thank the Center for Information Services and High Performance Computing (ZIH) at TU Dresden for generous allocations of computer time.

References

Amd67. G. M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. *Proc. AFIPS 1967 Spring Joint Computer*

Conf. 30 (April), Atlantic City, N.J., pages 483–485, 1967.

Bie11. Christian Bienia. *Benchmarking Modern Multiprocessors*. Philosophy doctor thesis, Princeton University, 2011.

BKSL08. Christian Bienia, Sanjeev Kumar, Jaswinder Pal Singh, and Kai Li. The parsec benchmark suite: Characterization and architectural implications. In *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, PACT '08, pages 72–81, New York, NY, USA, 2008. ACM. URL: http://doi.acm.org/10.1145/1454115.1454128, doi:10.1145/1454115.1454128.

BSVXdS15. 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 and Microsystems*, 39(6):418–425, aug 2015. URL: http://www.sciencedirect.com/science/article/pii/S0141933115000617, doi:10.1016/j.micpro.2015.05.009.

CK02. M. Clerc and J. Kennedy. The particle swarm - explosion, stability, and convergence in a multidimensional complex space. *IEEE Transactions on Evolutionary Computation*, 6(1):58–73, 2002. doi:10.1109/4235.985692.

DSDMD18. Daniele De Sensi, Tiziano De Matteis, and Marco Danelutto. Simplifying self-adaptive and power-aware computing with nornir. *Future Generation Computer Systems*, pages –, 2018. URL: https://www.sciencedirect.com/science/article/pii/S0167739X17326699, doi:https://doi.org/10.1016/j.future.2018.05.012.

EE10. Stijn Eyerman and Lieven Eeckhout. Modeling critical sections in amdahl's law and its implications for multicore design. *SIGARCH Comput. Archit. News*, 38(3):362–370, June 2010. URL: http://doi.acm.org/10.1145/1816038.1816011, doi:10.1145/1816038.1816011.

Gus88. John L. Gustafson. Reevaluating amdahl's law. *Communications of the ACM*, 31:532–533, 1988.

GXdSE17. Kyriakos Georgiou, Samuel Xavier-de Souza, and Kerstin Eder. The IoT energy challenge: A software perspective. *IEEE Embedded Systems Letters*, pages 1–1, 2017. URL: http://ieeexplore.ieee.org/document/8012513/, doi:10.1109/LES.2017.2741419.

HH17. Siegfried Höfinger and Ernst Haunschmid. Modelling parallel overhead from simple run-time records. *J. Supercomput.*, 73(10):4390–4406, October 2017. URL: https://doi.org/10.1007/s11227-017-2023-9, doi:10.1007/s11227-017-2023-9.

HM08. Mark D. Hill and Michael R. Marty. Amdahl's law in the multicore era. *Computer*, 41(7):33–38, 2008. doi:http://doi.ieeecomputersociety.org/10.1109/MC.2008.209.

HZQ+13. Tian Huang, Yongxin Zhu, Meikang Qiu, Xiaojing Yin, and Xu Wang. Extending amdahl's law and gustafson's law by evaluating interconnections on multi-core processors. *J. Supercomput.*, 66(1):305–319, October 2013. URL: http://dx.doi.org/10.1007/s11227-013-0908-9, doi:10.1007/s11227-013-0908-9.

KE95. J Kennedy and R Eberhart. Particle swarm optimization. *Neural Networks, 1995. Proceedings., IEEE International Conference on*, 4:1942–1948 vol.4, 1995. arXiv:9780201398298, doi:10.1109/ICNN.1995.488968.

LCB16. Arthur Francisco Lorenzon, Márcia Cristina Cera, and Antonio Carlos Schneider Beck. Investigating different general-purpose and embedded multicores to achieve optimal trade-offs between performance and energy. *Journal of Parallel and Distributed Computing*, 95:107–123, sep 2016. URL: https://www.sciencedirect.com/science/article/pii/S0743731516300090, doi:10.1016/J.JPDC.2016.04.003.

LS17. Yu-Hang Liu and Xian-He Sun. Evaluating the combined effect of memory capacity and concurrency for many-core chip design. *ACM Trans. Model. Perform. Eval. Comput. Syst.*, 2(2):9:1–9:25, March 2017. URL: http://doi.acm.org/10.1145/3038915, doi:10.1145/3038915.

NSS15. Surya Narayanan, Bharath N. Swamy, and André Seznec. An empirical high level performance model for future many-cores. In *Proceedings of the 12th ACM International Conference on Computing Frontiers*, CF '15, pages 1:1–1:8, New York, NY, USA, 2015. ACM. URL: http://doi.acm.org/10.1145/2742854.2742867, doi:10.1145/2742854.2742867.

OFS+18. Victor H. F. Oliveira, Alex F. A. Furtunato, Luiz F. Silveira, Kyriakos Georgiou, Kerstin Eder, and Samuel Xavier-de Souza. Application speedup characterization: Modeling parallelization overhead and variations of problem size and number of cores. In *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering*, ICPE '18, pages 43–44, New York, NY, USA, 2018. ACM. URL: http://doi.acm.org/10.1145/3185768.3185770, doi:10.1145/3185768.3185770.

PPZ+16. A. Pahlevan, J. Picorel, A. P. Zarandi, D. Rossi, M. Zapater, A. Bartolini, P. G. Del Valle, D. Atienza, L. Benini, and B. Falsafi. Towards near-threshold server processors. In *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 7–12, March 2016.

PVG+11a. F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

PVG+11b. F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

SC10. Xian-He Sun and Yong Chen. Reevaluating Amdahl's law in the multicore era. *Journal of Parallel and Distributed Computing*, 70(2):183–188, feb 2010. URL: http://www.mendeley.com/catalog/reevaluating-amdahls-law-multicore-era/, doi:10.1016/j.jpdc.2009.05.002.

Sen16. D. De Sensi. Predicting performance and power consumption of parallel applications. In *2016 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP)*, pages 200–207, Feb 2016. doi:10.1109/PDP.2016.41.

SFG+18. Vitor R. G. Silva, Alex F. A. Furtunato, Kyriakos Georgiou, Kerstin Eder, and Samuel Xavier de Souza. Energy-optimal configurations for single-node HPC applications. *CoRR*, abs/1805.00998, 2018. URL: http://arxiv.org/abs/1805.00998, arXiv:1805.00998.

Shi96. Yuan Shi. Reevaluating Amdahl ' s Law and Gustafson ' s Law, 1996. URL: https://www.researchgate.net/profile/Yuan{_}Shi12/publication/228367369{_}Reevaluating{_}Amdahl's{_}law{_}and{_}Gustafson's{_}law/links/562f9dd408ae8e1256876a0a.pdf.

SN93. X.H. Sun and L.M. Ni. Scalable Problems and Memory-Bounded Speedup. *Journal of Parallel and Distributed Computing*, 19(1):27–37, sep 1993. URL: http://linkinghub.elsevier.com/retrieve/pii/S0743731583710877, doi:10.1006/jpdc.1993.1087.

SR16. Gabriel Southern and Jose Renau. Analysis of PARSEC workload scalability. *ISPASS 2016 - International Symposium on Performance Analysis of Systems and Software*, pages 133–142, 2016. doi:10.1109/ISPASS.2016.7482081.

SS04. Alexander J Smola and Bernhard Schölkopf. A Tutorial on Support Vector Regression. *Statistics and Computing*, 14(3):199–222, 2004. doi:Doi10.1023/B:Stco.0000035301.49549.88.

SW14. X. Sun and D. Wang. Concurrent average memory access time. *Computer*, 47(5):74–80, May 2014. doi:10.1109/MC.2013.227.

WM95. W. A. Wulf and Sally A. McKee. Hitting the memory wall: Implications of the obvious. *SIGARCH Comput. Archit. News*, 23(1):20–24, March 1995. URL: http://doi.acm.org/10.1145/216585.216588, doi:10.1145/216585.216588.

WOT+95. Steven Cameron Woo, Moriyoshi Ohara, Evan Torrie, Jaswinder Pal Singh, Anoop Gupta, Steven Cameron Woo, Moriyoshi Ohara, Evan Torrie, Jaswinder Pal Singh, and Anoop Gupta. The SPLASH-2 programs. In *Proceedings of the 22nd annual international symposium on Computer architecture - ISCA '95*, volume 23, pages 24–36, New York, New York, USA, 1995. ACM Press. URL: http://portal.acm.org/citation.cfm?doid=223982.223990, doi:10.1145/223982.223990.

YMG14. L. Yavits, A. Morad, and R. Ginosar. The effect of communication and synchronization on amdahl's law in multicore systems. *Parallel Comput.*, 40(1):1–16, January 2014. URL: http://dx.doi.org/10.1016/j.parco.2013.11.001, doi:10.1016/j.parco.2013.11.001.