Hudson Jorge, Emilio C. G. Wille

Federal University of Technology - Paraná (UTFPR) - Brazil

doi:10.15199/48.2021.04.25

Improving the Accuracy of Multi-Core Systems Simulations using Behavioral Modeling

Abstract. *Many studies undertaken to date realize that the CPU performance does not scale linearly as the number of CPU cores increases. Several models seek to quantify the performance of multi-core systems. However, these models have a set of limitations and cannot accurately predict the real speedup of a given system. Therefore, other kinds of models can be helpful. In the paper we first set up and validate a simulation environment in order to evaluate multi-core systems, and second we propose and test a refined model that significantly improves performance predictions.*

Streszczenie. *Wiele dotychczasowych badań wykazuje, że wydajność procesora nie skaluje się liniowo wraz ze wzrostem liczby rdzeni procesora. Przedstawiono kilka modeli umożliwiających ilościowe określenie wydajności systemć w wielordzeniowych. Jednak modele te mają szereg ograniczeń i nie mogą dokładnie przewidzieć rzeczywistości przyspieszenie danego systemu. Dlatego pomocne mogą być inne rodzaje modeli. W artykule najpierw skonfigurowaliśmy i sprawdziliśmy kolejno środowisko symulacji do oceny systemć w wielordzeniowych, a następnie proponujemy i testujemy udoskonalony model, ktć ry znacznie poprawia prognozy wydajności.* ***(Poprawa dokładności symulacji systemów wielordzeniowych za pomocą modelowania behawioralnego)***

Keywords: Multi-core systems, parallel architectures, hardware performance counters, CPU speedup, simulation
Słowa kluczowe: System wielordzeniowy, ocena szybkości CPU, symulacja i model systemu wielordzeniowego

Introduction

One well known empiric result in literature is that a n-core system does not deliver n times the speed of a single-core system, independent of the architecture used to interconnect them. There are many reasons for that, for example, memory and cache availability, I/O interruptions and disk wait time. This measurable behavior indicates that no multi-core architecture can give exclusive resources to each individual core, making the overall system performance decreases, due to shared resources competition. This resource competition has an impact on the overall performance of applications and consequently on the scalability of multi-core systems. In addition to these problems, the software also plays an important role since it should be designed to make use of this architecture, implementing optimized parallel algorithms, e.g., [1, 2].

Several studies seek to quantify the *speedup* of multi-core systems, i.e., the gain (considering execution times) of a multi-core system over a single-core system. Statistical models, like Amdahl's and others [3, 4, 5, 6, 7], are fundamental to guide the efforts of researches and industry. However, these models cannot accurately predict the actual speedup of a system based on the architecture and the software currently running on top of it. Therefore, other kinds of models can be helpful. For example, Kandalintsev and Lo Cigno [8] approach this problem in a very pragmatic way, proposing a model – called behavioral first order (BFO) model – based on direct measurements on hardware performance counters (HPCs) available in multi-core processors.

The contribution of our work is thus twofold. First of all, we set up and validate a simulation environment in order to evaluate multi-core systems. Thus we modify an existing simulator to implement the BFO model, giving a more accurate simulator platform to researchers and developers. We choose the CloudSim (release 3.2) tool [9] because it is flexible enough to be extended, it is free software, well documented and most of all (besides being created to investigate cloud systems) it could be adapted to our purpose, which is to predict the CPU scaling in multi-core computational systems. Second, our simulations show that the BFO model rapidly loses its prediction capability as the number of cores increases. Therefore, we propose and test a refined model (introducing a logarithmic correction) that explicitly considers the number of active parallel cores. Our proposed model proves to significantly improve the performance predictions.

Related Work

The seminal paper of Amdahl [3] claims that, statistically, a software execution in a multi-core CPU environment can be divided in two parts, the parallel one that represents the fraction of the execution time where the cores have all the resources (physical and logical) available to execute their jobs and the serial one that represents the fraction of the execution time where the cores have to wait for a resource to be free or available. In the time of Amdahl's observation the serial fraction was estimated to be as big as 30%.

Gustafson [4] noticed that the Amdahl's law was not predicting well the measured data. He stated that the Amdahl's law was too restrictive because assumed that the serial fraction remained the same size independent of the problem size to be computationally solved (this is known as *fixed size* law). According to Gustafson, the software designers make use of the new multi-core architecture resources and developed new algorithms that escalate down the serial fraction in the same proportion of the problem size. Thus if the problem is doubled in size, the serial fraction is halved. This is known as *fixed time* law, because the serial fraction has the same absolute time independent of the problem size. The fixed time law assured that the multi-core architectures could perform very well if the software (operational system and algorithms) was designed to extract the best of these systems. However, in the servers market, very limited multi-core architectures were available for use, mainly because of the costs to implement these chips. Then, mainly because of the Moore's law failings [10], the multi-core approach had become the only viable solution to continue the exponential growth of the power computing.

Since then the discussion of CPU performance scaling reopened. The main reason is because, even if the multi-core CPU implementations are based on high performance computing architectures, they are more limited in a silicon chip because of restricted available space and limitations of the interconnections possible between the processors. Based on these limitations Marty and Hill [5] developed a corollary of the Amdahl's law, based on the concept of BCE (Base Core Equivalent). The Amdahl's law served as a baseline for Sun and Ni's work [6]. They created a model adherent to multi-core processor architectures, showing that it is possible to perform close to Gustafson's law if there is enough cache memory available inside the chip. Juurlink and Meenderinck [11] generalized Amdahl's and Gustafson's laws by assuming that the parallel fraction does not stay constant as in Amdahl's law, nor that it grows linearly with the number of cores as in Gustafson's law, but something in between (i.e., sublinear to n, where n is the number of cores). They refer to this equation as GSSE (Generalized Scaled Speedup Equation). Moreover, Sun and Chen [7] showed that it is possible to perform reasonably well even in presence of the *memory wall* problem (that is the disparity between how fast a CPU can operate on data and how fast it can get/put data on memory).

These statistical models are fundamental to guide efforts of researches and industry concerning the development of new generations of multi-core processors, as well as guide the software research to improve the use of parallel architectures. However, they cannot accurately predict the actual speedup of a system based on the architecture and the software currently running on top of it.

Therefore, other kinds of models can be helpful, those capable to predict the performance in the systems based on observable statistics collected during runtime. There are several works in this area, usually they are either too complex for implementation in real systems or are based on "hard to measure" parameters. For example, Barroso et al. [12] created the RPM (Rapid Prototyping for Multiprocessor), an engine based on queue analysis. It presents a very good precision on the predictions, but it is slow to run since it executes the pipe of instructions in dedicated hardware (like cache and cache controller) to access the needed cache statistics. Nanda et al. [13] developed MemorIES that substitute physically the L1, L2 and L3 caches by a dedicated hardware capable of collecting the cache statistics and calculating the processor's performance. This model is used to orient the design of servers and, besides being faster than other models, it uses parameters inaccessible in commercial architectures. Bergamaschi et al. [14] proposed SLATE (System-Level Analysis Tool for Early Exploration), a tool that models the CPU internals, based on execution and wait times, in a queue driven analysis. It is based on SystemC performance modeling, which consists on the description of the model to be simulated in terms of components (caches, queues, cores and interconnections between them). Cores are modeled in high-level in terms of delays and execution times, and an engine calculates the state of the system in a cycle-by-cycle fashion. Then the interconnections between the components are modeled in order to have a multi-core environment.

Searching for a model capable of delivering good and fast results, Kandalintsev and Lo Cigno [8] proposed a simple model based on the measured behavior an application presents when running on multi-core systems. This approach proposes a CPU centric model, based only on measurable parameters (collected using the hardware performance counters), capable of producing better predictions than the linear CPU scaling paradigm usually present in the literature, e. g., [15, 16, 17, 18]. This model disregards CPU internals, what makes it computationally very fast and suitable to be implemented in simulators, since it is possible to model the CPU performance individually. This model will be hereafter called behavioral first order (BFO) performance model. The BFO model was further developed in Kandalintsev's PhD thesis *"Application Interference in Multi-Core Architectures: Analysis and Effects"* [19] where a methodology to collect the parameters of the model is described as well as the methodology to test the model against real world systems.

The Behavioral First Order (BFO) Performance Model

Based on an inter-core interference interpretation, Kandalintsev and Lo Cigno proposed a model that studies the decrease in performance of a CPU core, due to concurrency with others CPU cores for common shared resources. The BFO models the observable behavior, the increase in execution time when the CPU core is competing with others for shared resources, comparing with the situation where the CPU core presents no concurrency. The model defines as *overhead* (OH) the extra time to complete a job when concurrency is present. The BFO model proposes that the overhead will be measured when the CPU cores under test are fully occupied (100% load), but admits that the actual overhead is a function of the actual load in the CPU cores. In other words, if the CPU cores are using less than 100% of capacity, they will have less probability to find some resource already occupied. This assumption is compatible with the Amdahl's corollary, where the speedup of multi-core systems are better than the Amdahl's law predicts (in terms of BCEs) if the actual utilization is lower than what is available in the processor [5].

The BFO model claims that the overhead is present when there are tasks running in parallel, otherwise there is no overhead. The model considers steady state performance (meaning tasks that run for long time), examples are video encoding, scientific simulations and data streaming. It is assumed that the dominant interaction between tasks is pairwise, i.e., the total overhead is the sum of the overhead running in pairs. The expectation here is to have the overhead overestimated, but still a good approximation of the real system.

Considering the model, the *task load*, $L_i = L(T_i)$ with $L_i \in [0, 1]$, is defined as the time required by task i in a particular CPU core, assuming that the CPU core can be running several tasks in a time shared way. The total load per-core is the sum of the task loads and sum up to a maximum of 1. The *normalized system load*, $L_{sys} \in [0, n]$, where n is the number of CPU cores, is defined as the sum of total loads of all CPU cores in the system. The pipe notation $T_i|T_j$ denotes that the tasks T_i and T_j are running in different CPU cores. Using pipe notation we have $L_{sys}(A|B)$ being the normalized system load when task A is running in one CPU core and task B in another CPU core,

(1) $$\mathsf{L}_{sys}(A|B) = L_A + L_B + L^{oh}_{A|B},$$

where $L^{oh}_{A|B}$ is the *parasitic* or *overhead load*, which is present only when there are tasks running in parallel on both CPU cores. This is the most essential part of the model and reflects the waiting time where one CPU core has to stop the execution of a task because of a busy common resource, extending the execution time. The idea is that this extended execution time is enough to model the interaction and make dependable predictions. The formal definition of $L^{oh}_{A|B}$ is given by:

(2) $$\mathsf{L}^{oh}_{A|B} = \beta_{A|B} L_A L_B,$$

where $\beta_{A|B}$ expresses the *level of interference* (or coupling) that task A and task B exert on each other. This interference is assumed to be constant and dependent only on the nature of the tasks itself. If two tasks have affinity to the same resources, it is expected to be higher than if the tasks have different affinities. The overhead load also captures the amount of time where the interference actually works as the product of the individual loads.

One could think that the coupling factors could make more explicit the internals of the processors and peripherals. It could be done considering it as a vector, and each component would refer to one common resource, like a FPU (Floating Point Unit) that is shared with two cores or because the cores are hyper-threaded, same for ALU (Arithmetic and Logic Unit) and this analysis would be extended to caches, memory, disk, I/O, etc. However, this vector would be like 40 or even 50 components long [17], increasing the complexity of the model. In order to overcome this problem the overhead load can be decomposed in individual performance penalties related to each task,

$$\mathsf{L}^{oh}_{A|B} = \beta_{A\to B} L_A L_B + \beta_{B\to A} L_A L_B, \tag{3}$$

where $\beta_{A\to B}$ represents the interference of task A over task B and $\beta_{B\to A}$ represents the interference of task B over task A. Equation 3 can be transformed into a matrix form if we consider that the tasks running on a core k are represented by the vector T_k of loads they generate. For two-core architectures with two tasks we have:

$$\beta_{A\to B} L_A L_B = \begin{bmatrix} L_A & 0 \end{bmatrix} \begin{pmatrix} \beta_{A\to A} & \beta_{A\to B} \\ \beta_{B\to A} & \beta_{B\to B} \end{pmatrix} \begin{bmatrix} 0 \\ L_B \end{bmatrix}, \tag{4}$$

$$\beta_{B\to A} L_B L_A = \begin{bmatrix} 0 & L_B \end{bmatrix} \begin{pmatrix} \beta_{A\to A} & \beta_{A\to B} \\ \beta_{B\to A} & \beta_{B\to B} \end{pmatrix} \begin{bmatrix} L_A \\ 0 \end{bmatrix}. \tag{5}$$

In a compact and more general form:

$$L^{oh}_{1\to 2} = \mathbf{T}_1^T \mathbf{B} \mathbf{T}_2, \tag{6}$$

where $\mathbf{B}$ is the matrix of coupling factors, $\mathbf{T}_1$ and $\mathbf{T}_2$ are vectors that represent the load of a task running on CPU core 1 and 2, respectively, and $L^{oh}_{1\to 2}$ is the overhead that all tasks running on core 1 impose to the tasks running on core 2. The diagonal elements correspond to the interference effect of a process over the same processes but running in different CPU cores. A further generalization permits the calculation of the total overhead in an n-core CPU. It is given by the sum of all the overheads imposed by tasks of every core on every other core:

$$L^{oh}_{sys} = \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j\neq i}}^{n} \mathbf{T}_i^T \mathbf{B} \mathbf{T}_j. \tag{7}$$

Finally the estimated performance of the system (meaning a particular CPU core c_i) can be seen as its effective load L_{sys}, that is calculated by subtracting the overhead from the ideal system load (the load when there is no concurrency). The overhead represents a "penalty" on the *ideal system load* caused by the interference due to the contention for shared resources. Equation 8 shows the system load computation:

$$L_{sys} = \sum_{i=1}^{n} \mathbf{T}_i - \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j\neq i}}^{n} \mathbf{T}_i^T \mathbf{B} \mathbf{T}_j. \tag{8}$$

Concluding this section, it is clear that the BFO model can be advantageously used to increase performance prediction calculations. This model disregards CPU internals, making it computationally very fast and suitable to be implemented in simulators.

Performance Measurements

The objective of this section is to present the performance measurements, testing programs and hardware configurations used in our experiments. A methodology to obtain the pairwise coupling factors needed for the simulations is also proposed.

To obtain the data needed, we resort to the *hardware performance counters* (HPCs). The HPCs are special registers that collect hardware statistics during run-time and have almost no overhead associated. As any measure, HPC statistics are not exact and have some degree of uncertainty. The main reason is that these counters have to be collected on-line, so they use resources to be processed and stored. According to Zaparanuks et al. [20] this accuracy is dependent on the number of enabled counters being collected and managed. However, this overhead in collecting and treating the counters is not the only source of error. Another one is due to the interruptions that CPU core treat during runtime, the result is a possible duplication in the count of the interrupted instruction. This is always a over-count leading to overestimation of performance, but luckily it is small and sometimes partially compensate the underestimation caused by the overhead in collecting the data. Waver et al. [21] estimates this over-count to be less than 1%, what is good for most of the practical purposes.

The hardware used in the experiments in order to get measured dada are:

1. System 1: Notebook with one processor Intel i7-6500U (2 individual cores, or 4 cores with hyper-thread activated), 16GB of RAM, 256 GB SSD and 1GBps Ethernet network adapter.
2. System 2: Server Dell with two processors AMD Opteron (2 individual cores, no hyper-thread), 32GB of RAM, 2TB HD and 4 1GBps Ethernet network adapter.
3. System 3: Server AirFrame Nokia with two processors Xeon (14 individual cores, or 28 cores with hyper-thread activated), 256BG of DDR4 RAM, 512 GB SSD and 2x 6TB HD Raid 2.

The basic metric to be considered is the *number of instructions per clock cycle* (I_c). This metric shows naturally the problem of the interference when a CPU core is waiting to use an already busy resource, or if it is fetching data that is not in the local cache. This last reason is a really important one, because caches are much faster than accessing the local memory that in its turn is faster than accessing remote memory (i.e., in other NUMA zone). The effects of these factors are a stalling of the CPU core, which loses CPU clocks to wait for the busy resource or the data to be available. The data needed is easily measurable in a computational system by using performance tools like Linux *perf*. We use the *stress-ng* tool to generate the CPU activity of different profile. This choice is because of the fact that *stress-ng* can generate controlled loads in various dimensions of the system, generating CPU load, RAM load, I/O load in a controlled way. Also the *taskset* tool is used to pin a job to a particular core. The native load profiles, also known as *stressors*, chosen for this work are:

1. INT128: This stressor generates 1000 iterations of a mix of 128 bit integer operations. It is good for benchmarking the CPU performance for operations over integers.
2. FFT: This stressor generates blocks of 4096 samples for FFT calculations. It is good for benchmarking the CPU performance for DSP applications.
3. MPROD: This stressor generates matrix product of two 128 by 128 matrices of double floats. Testing on 64 bit

x86 hardware shows that this provides a good mix of memory, cache and floating point operations.

4. CALLF: This stressor generates recursively call 8 argument C function to a depth of 1024 calls. It is good to benchmarking the memory scaling capabilities on the processor cache.

Finally, when analyzing a given system, we considered the *root mean square error* (RMSE) on I_c values (following [8]) to evaluate the quality of the predictions. The RMSE is given by:

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(\frac{I_c^{pred}(i) - I_c^{meas}(i)}{I_c^{meas}(i)}\right)^2}, \tag{9}$$

where N is the number of runs, $I_c^{pred}(i)$ is the i^{th} value of I_c predicted by the simulator and $I_c^{meas}(i)$ is the i^{th} value of the measured I_c. A low RMSE means better prediction.

Empirical Determination of the Coupling Factors

In order to obtain the pairwise coupling factors (β) needed for the simulations, we consider two distinct moments of measurements. The first one (*step 1*) is when a stressor is issued alone in one CPU core and no other activity is issued any other core. The second moment (*step 2*) is when stressors are issued synchronously in a pair of cores, these stressors can be distinct ones or not. These scenarios can also be called *ideal* and *pairwise* scenarios.

Step 1: In this step we use *taskset* to pin the stressor to one core, and use *perf* to read the number of instructions, named here X_j^i, with $i \in \{0, 1, 2, 3, ...\}$ being the number of the core in the processor, and $j \in \{A, B, C, ...\}$ being the stressor available to test. As the stressors are issued via *stress-ng* for a pre-defined amount of time Δt (we used $\Delta t = 320s$), we define $x_j^i = X_j^i/\Delta t$, in $[MIPS/s]$, as the capacity of core i to consume the stressor j. As the stressor is issued for 100% CPU power, by definition of BFO model, the system load is $L_{sys} = 1$, and in this case the task load is $L_j^{i,full} = 1$, meaning that for core i running stressor j the *full utilization* is reached. This test is done in all cores and with all stressors.

Next we compare these measurements pairwise, in other words, what should be the behavior of the system if no interference exist pair-wisely. In this case $L_{sys} = 2$, by definition, and we can say, without loss of generality, that for cores 0 and 1 and stressors A and B,

$$L_{sys} = L_A^{0,full} + L_B^{1,full} = 2. \tag{10}$$

However, in general, $X_A^{0,step1} \neq X_B^{1,step1}$ and $L_A^{0,full} = L_B^{1,full} = 1$, what means that the total number of instructions $X_A^{0,step1} + X_B^{1,step1}$ have different weights for stressor A and B. We define ϵ as the instruction ratio, as a representation of the weights that $X_A^{0,step1}$ and $X_B^{1,step1}$ have in the total of instructions available with no interference considered,

$$\epsilon_A = \frac{X_A^{0,step1}}{X_A^{0,step1} + X_B^{1,step1}}, \tag{11}$$

As the time of test is fixed, ϵ can also be calculated in terms of the core capacity x,

$$\epsilon_A = \frac{x_A^{0,step1}}{x_A^{0,step1} + x_B^{1,step1}}, \tag{12}$$

Step 2: The *perf* tool present the total instructions that stressor A and B consumed to be completed. Because of the definition of BFO model, we need the instructions consumed in core 0 and core 1 individualized. To calculate it we assumed that ϵ is constant if the task load is unchanged, that is our case because *stress-ng* still consumes 100% of the core processing power. Defining $Z_{sys}^{meas.}$ as the number of instructions read in *perf* for *step 2*, we have:

$$X_A^{0,step2} = \epsilon_A.Z_{sys}^{meas.},\ \ X_B^{1,step2} = \epsilon_B.Z_{sys}^{meas.}, \tag{13}$$

symilarly,

$$x_A^{0,step2} = \frac{\epsilon_A.Z_{sys}^{meas.}}{\Delta t},\ \ x_B^{1,step2} = \frac{\epsilon_B.Z_{sys}^{meas.}}{\Delta t}. \tag{14}$$

By definition, OH is the part of the system processing that is spent with the interference, in our case it is $OH = (X_A^{0,step1} + X_B^{1,step1}) - Z_{sys}^{meas.}$. In terms of task load,

$$L_{sys}^{oh} = \frac{OH}{Z_{sys}^{meas.}}, \tag{15}$$

and by definition of overhead,

$$L_{sys}^{oh} = \beta L_A^{0,step2} L_B^{1,step2}. \tag{16}$$

We know that $L_A^{0,step2} = L_B^{1,step2} = 1$. That lead us to Equation 17 with the final form of β,

$$\beta = L_{sys}^{oh} = \frac{(X_A^{0,step1} + X_B^{1,step1}) - Z_{sys}^{meas.}}{Z_{sys}^{meas.}}. \tag{17}$$

Finally, the individual contributions $\beta_{A\to B}$ and $\beta_{B\to A}$ can easily be calculated as:

$$\beta_{A\to B} = \epsilon_A.\beta \text{ and } \beta_{B\to A} = \epsilon_B.\beta. \tag{18}$$

Framework for Multi-Core Systems Evaluation

Our framework for multi-core systems evaluation was implemented as an extension of the CloudSim tool. The framework can be divided in three main phases: *testing phase*, *training phase* and *simulation phase* which are described in the following.

Testing Phase: Results of the Experimental System

In the testing phase the objective is to generate enough statistics in an experimental system. For that we use the data from HPC considering the combinations of hardware and stressors described earlier. Two scenarios can be considered. The first one (*ideal*) is when a stressor is issued alone in one CPU core and no other activity is issued any other core, and the second one (*pairwise*) is when stressors are issued synchronously in a pair of cores. Statistical data are stored for use in the next phase.

Training Phase: Results of the Calibration Tests

In this phase the coupling factors and the adjusts of the real CPU power (in MIPS) are find out. In Table 1 it is shown the number of instructions per second that each stressor generates for the ideal case. This data is calculated as the average over 10 interactions on each CPU core for each run of the script. Considering a case where a processor have $n = 2$ cores and the script will run for $k = 4$ stressors, the number of times the same statistic for instructions per second is collected is $m = A_n^2.k^2.10 = 160$.

It is important to note that the value of MIPS measured varies in a fixed CPU type depending on each stressor. For

Table 1. Instructions per second (MIPS) calculated for each CPU type in the ideal case.

CPU / Stressor	INT128	FFT	MPROD	CALLF
i7-6500U (System 1)	1550.14	3204.71	4399.28	5289.63
Opteron 2214HE (System 2)	1927.93	2163.79	1063.88	2190.07
XEON E5-2680v4 (System 3)	1873.95	3990.38	5238.49	6168.87
Number of tests	160	160	160	160

example, consider the Opteron CPU (System 2), for the stressor INT128 the value measured is 1927.93 MIPS, this contrasts with the FFT stressor that presented a value of 2163.79 MIPS. This result is due to each stressor to use the resources in a particular way. The measured values are also dependent to the CPU architecture and the pattern of the use of the resources. This information is very important because in CloudSim the capacity of a CPU (vCPU) is defined in MIPS, and it is not dependent on the task to be simulated (the *cloudlet*), but if we are to simulate different kinds of task/applications, it is necessary to correct the CPU capacity. These correction factors are derived from Table 1.

Basically the correction is done by adjusting the size of the *cloudlet* before the start of a simulation to maintain the execution time coherent with that observed in the experimental system. In our approach the original *Cloudlet* class is extended to have the attributes *applicationType*, that stores the name of the application a *cloudlet* object is carrying, and *correctionFactor*, that will be used to correct the *cloudlet* object size to be consumed by the CPUs.

For example, if a 2000 MIPS CPU is defined in CloudSim for a Host and VM objects, that represent a specific market CPU core, and this CPU has to consume a 10,000M instructions from INT128 and another 10,000M from CALLF, if not corrected both tasks would finish in the same execution time of 5s. But, as we can see, the CPU would present a different execution time in the experimental world, it would finish CALLF earlier then INT128. To illustrate that, imagine we were representing an i7-6500U CPU core. The task INT128 will be corrected to $\text{INT128}_{corrected}$ = 10,000M $\times$ 2000/1550.14 = 12,902M instructions, and CALLF to $\text{CALLF}_{corrected}$ = 10,000M $\times$ 2000/5289.63 = 3,781M instructions. So, the INT128 will finish in 6.45s and CALLF in 1.89s.

The next step corresponds to calculate the coupling factors β. The results are shown in Table 2.

Simulation Phase: Validation of Extensions made on CloudSim

After collecting and processing the data in the previous phases, the next step is to validate the extended CloudSim to guarantee it reproduces the results observed in the experimental system. This phase has two parts, the first one is to ensure that CloudSim reproduces the ideal results. In the second part we apply the concept of interference overhead, to simulate the tests with two CPU cores interfering with one another. These two parts will ensure that CloudSim is capable to reproduce the BFO model validating the model and the implementation in software.

Part I: Reproducing the ideal scenarios

In this part, it is tested the concept that a *cloudlet* must be corrected in its size, to reflect that the use of a CPU core from a particular task have differences inherent to it, as explained in training phase section. The methodology here is to create simulation environments mimicking the

Table 2. Coupling Factors ($\beta_{A \to B}$)

row:task A / column:taskB	INT128	FFT	MPROD	CALLF
INT128	0.3966	0.1785	0.1078	0.1265
FFT	0.3691	0.0192	0.2203	0.2101
MPROD	0.3059	0.3025	0.1287	0.2588
CALLF	0.4317	0.3468	0.3112	2.0482

(a) i7-6500U

row:task A / column:taskB	INT128	FFT	MPROD	CALLF
INT128	0.1973	0.553	1.2441	0.0698
FFT	0.6207	0.0487	0.0583	0.0524
MPROD	0.6865	0.0286	0.0536	0.00125
CALLF	0.0792	0.0530	0.0026	0.2161

(b) Opteron 2214HE

row:task A / column:taskB	INT128	FFT	MPROD	CALLF
INT128	0.2847	0.1792	0.0927	0.1140
FFT	0.3816	0.0752	0.2054	0.2228
MPROD	0.2592	0.2696	0.1376	0.2158
CALLF	0.3754	0.3445	0.7054	1.1595

(c) Xeon E5-2680v4

experimental systems, where *cloudlets* of different kind and sizes are created and executed. The quality of the results is based on the root mean square error (RMSE). Also, the simulated execution time is compared with the measured execution time of the experimental system.

Initially we take the ideal test scenarios executed in the training phase and tweak the simulator parameters (MIPS) to calibrate the simulator. After these initial simulations, some *cloudlets* (bigger and smaller) were simulated, and compared with the experimental measurements (over 10 runs). The *cloudlets* sizes are 25%, 50%, 100%, 200% and 400% of the ideal scenarios. Table 3 shows the results of the tests for the stressors INT128 (a), FFT (b), MPROD (c) and CALLF (d), considering the Xeon CPU. Similar RMSE values were obtained for the other CPUs (not shown by space restrictions).

With these results we can see that the *cloudlets* can approximate the experimental scenarios very well. As expected in these scenarios the execution time scales linearly. Maybe it is not a very good approximation if the *cloudlet* is very short, since the model used has the assumption that the tasks/CPU cores are in steady state. Meaning that the caches are already stabilized, since tasks too short and possibly uncoordinated can raise the memory access due to more often cache misses.

Part II: Interference in pairwise scenarios

Now the goal is to put CloudSim to simulate the scenarios that were used to calculate the coupling factors β. In this part the extensions made in all classes to support the overhead calculations are tested. Table 4 synthesizes the RMSE calculated over the simulation results against the pairwise measurements done in the training phase (all tests are calibrated for 320s of runtime).

We notice that, in this phase, the simulator has consistently presented good results, with absolute errors of almost zero. Actually this was expected, since the pairwise model is extracted directly from measurements, so we understand the errors are related with the way the simulator operates internally. As the clock tick is a fixed value, the

Table 3. CloudSim simulating various stressors results with Xeon E5-2680v4 parameters (declared in CloudSim with MIPS=6000)

INT128 Cloudlet	25%	50%	100%	200%	400%
size (Millions Inst.)	84327	168655	337311	674622	1349244
corrected size factor=3.2018	270000	540000	1080000	2160000	4320000
CloudSim exec. time (s)	45.03	90.02	180.01	360.03	719.94
Real System exec. time (s)	47.67	90.99	180.51	359.17	721.19
$\Delta\%$	-5.54	-1.07	-0.28	0.24	-0.17
RMSE	0.0435	0.0225	0.0100	0.0102	0.0091

(a) Simulating Xeon E5-2680v4 for INT128

FFT Cloudlet	25%	50%	100%	200%	400%
size (Millions Inst.)	179586	359172	718344	1436688	2873376
corrected size factor=1.5035	270000	540000	1080000	2160000	4320000
CloudSim exec. time (s)	45.01	90.01	180.00	359.96	720.03
Real System exec. time (s)	47.56	90.67	180.19	358.93	719.31
$\Delta\%$	-5.36	-0.73	-0.11	-0.29	0.10
RMSE	0.0453	0.0186	0.0062	0.0099	0.0062

(b) Simulating Xeon E5-2680v4 for FFT

MPROD Cloudlet	25%	50%	100%	200%	400%
size (Millions Inst.)	235732	471464	942928	1885856	3771712
corrected size factor=1.1454	270000	540000	1080000	2160000	4320000
CloudSim exec. time (s)	44.99	90.02	180.03	359.99	720.08
Real System exec. time (s)	47.77	91.01	179.45	360.21	721.03
$\Delta\%$	-5.82	-1.09	0.32	-0.06	-0.13
RMSE	0.0520	0.0233	0.0106	0.0053	0.0068

(c) Simulating Xeon E5-2680v4 for MPROD

CALLF Cloudlet	25%	50%	100%	200%	400%
size (Millions Inst.)	277599	555198	1110396	2220793	4441586
corrected size factor=0.9726	270000	540000	1080000	2160000	4320000
CloudSim exec. time (s)	44.99	90.02	180.07	360.04	720.08
Real System exec. time (s)	47.83	90.89	179.84	359.23	719.41
$\Delta\%$	-5.94	-0.96	0.13	0.23	0.09
RMSE	0.0477	0.0208	0.0077	0.0088	0.0058

(d) Simulating Xeon E5-2680v4 for CALLF

simulator always will present an execution time equal or greater than the real, this is caused because a *cloudlet* could be totally consumed in the middle of a tick, but the completion time is computed in the end of the slice of the time. To improve the quality of the simulation results the tick was set "small enough". For the tests it was fixed in 10 milliseconds. But for really big simulations, where the time of simulation should be optimized, we found that a tick of one hundredth of the smallest cloudlet size leads to errors less than 0.2% in average.

Table 4. RMSE between CloudSim simulations over stressors and real world measurements

core1/core2	INT128	FFT	MPROD	CALLF
INT128	0.0243	0.0110	0.0151	0.0082
FFT	0.0110	0.0275	0.0086	0.0077
MPROD	0.0151	0.0086	0.0026	0.0096
CALLF	0.0082	0.0077	0.0096	0.0287

(a) i7-6500U

core1/core2	INT128	FFT	MPROD	CALLF
INT128	0.0206	0.0174	0.0292	0.0075
FFT	0.0174	0.0142	0.0123	0.0368
MPROD	0.0292	0.0123	0.0355	0.0277
CALLF	0.0075	0.0368	0.0277	0.0221

(b) Opteron 2214HE

core1/core2	INT128	FFT	MPROD	CALLF
INT128	0.0305	0.0343	0.0022	0.0174
FFT	0.0343	0.0356	0.0166	0.0143
MPROD	0.0022	0.0166	0.0320	0.0192
CALLF	0.0174	0.0143	0.0192	0.0381

(c) Xeon E5-2680v4

Predicting the Behavior of Different Loads

In this section we test the extended CloudSim in different load scenarios, and compare the results with experimental measurements (i.e., a *generalization phase*). In addition, the same scenario is set in the original CloudSim (only correcting the MIPS for each stressor). This give us results as if the multi-core CPUs scale linearly, because that is what is inherently assumed in the literature (and in the original CloudSim implementation). Of course the number of possible combinations that can be tested is too big, so we considered some representative scenarios in order to obtain conclusions on the effectiveness of the method and the simulator capacity. The natural scenario is to increase the number of cores, and this was done considering Xeon E5-2680v4.

The tests in Table 5 were performed over all stressors, and represents the overall results we got (tests calibrated for 320s of runtime). Considering the line "Ext.CloudSim vs. Xeon", that represents the model proposed against the experimental measurements, the RMSE enlarges with respect to the number of cores. With 4 or 6 cores the error is acceptable, with absolute values always less than 10%. When the number of cores raise up the model show less accuracy, for 8 or more cores we cannot guarantee a good approximation since the absolute error increased. This is an indicator that for a large number of CPUs a higher order model is needed. Finally, considering the line "Ext. CloudSim vs. Orig. CloudSim" that represents the linear scaling model, we see that the RMSE is always worst when compared with the previous line. This means that the model gives a better prediction on CPU scaling than the simple approach of linear scaling model.

Table 5. RMSE between Extended CloudSim simulations and Xeon E5-2680v4. In addition the RMSE between Extended CloudSim and the Original CloudSim (that uses the linear CPU scaling model).

# CORES	4	6	8	10	12	14
Ext.CloudSim vs. Xeon	0.0287	0.0983	0.1356	0.2334	0.4521	0.8034
Ext.CloudSim vs. Orig.CloudSim	0.1498	0.2345	0.2847	0.4334	0.8521	1.5034

A Refined Model with Increased Number of Cores

Previous findings present in the literature ([11], [19]) and in our tests indicate that the parallel fraction grows with the number of cores following a sublinear law. Consequently, the coupling factor, being a measure of interference, will also follow this law. Then, in order to improve the accuracy of the model, a log factor that depends on the number of cores is considered. Equation 19 shows the proposed correction,

$$\beta' = \beta + \gamma.\beta.\log_2 n, \qquad (19)$$

where $n \geq 2$ is the number of cores, $\gamma \in [0, 1]$ is a non-negative constant, and β' is the new coupling factor in our proposed refined model.

We found that, for our log correction, there is a tradeoff: to make good predictions with n big we sacrifice the error when n is small. However, as the error introduced for small n grow slower than the reduction of the error for n large, we found that $\gamma = 0.1$ can produce good results in the range of Xeon with 14 cores. Table 6 shows the extended CloudSim RMSE (repeating the first line of Tabel 5), and comparing with the RMSE with extended CloudSim with log correction.

Table 6. RMSE between Extended CloudSim simulations and Xeon E5-2680v4 (considering $\gamma = 0$ and $\gamma = 0.1$).

# CORES	4	6	8	10	12	14
Ext.CloudSim ($\gamma = 0$) vs. Xeon	0.0287	0.0983	0.1356	0.2334	0.4521	0.8034
Ext.CloudSim ($\gamma = 0.1$) vs. Xeon	0.0293	0.0352	0.0456	0.0614	0.0815	0.1086

Conclusion

In the paper we first set up and validate a simulation environment in order to evaluate multi-core systems, and second we propose and test a refined model that significantly improves performance predictions.

The simulator has consistently presented good results in the training phase, with absolute errors of almost zero. During the simulation phase, again the results of the simulator were adherent to the measured during the training phase. The RMSE errors are quite small and should be zero. This was expected because the model is a first order model, and the coupling factors are a way to express the scaling behavior using the overhead concept. The generalization phase of the simulation presented some interesting results. The error increases rapidly with the number of CPU cores, for example with 4 cores it is about 2% to 3%, for 6 cores it jumps to 9% and sometimes 10%, and it continues growing when 14 cores are used, and the error is about 80%. It indicated that the first order model was not enough to deal with a large number of cores. To improve results of the model we introduced a logarithmic correction that worked well putting the error for 14 cores to about 11%, widening the range where the model could be used.

Finally, the size of the *cloudlets* have influence in the accuracy of the simulated results. It is due to the fact that, the model used as well as the simulator's CPU model, consider the system in steady state. However, in real systems, there are transitory responses when the caches are adapting to the demand. For the simulator perspective we can always improve the result by implementing small clock tick, but this elevates the time of a simulation. Users have to keep this tradeoff in mind during the simulation setup.

***Authors**: M. Sc. Hudson Jorge, Prof. Emilio Carlos Gomes Wille. Federal University of Technology - Paraná (UTFPR), Av. Sete de Setembro 3165, 80230-901, Curitiba (PR) - Brazil. email: hudson.jorge@nokia.com, ewille@utfpr.edu.br.*

REFERENCES

[1] D. Puchala, M. Yatsymirskyy, B. Szczepaniak, and K. Stokfiszewski, "Effectiveness of Fast Fourier Transform Implementations on GPU and CPU," *Przeglad Elektrotechniczny*, vol. 2016, no. 7, pp. 69–71, 2016.
[2] M. Cegielski, "Parallel computation of transient processes on OpenCL framework," *Przeglad Elektrotechniczny*, vol. 2016, no. 7, pp. 75–78, 2016.
[3] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *AFIPS spring joint computer conference*, 1967.
[4] J. Gustafson, "Reevaluating Amdahl's law," in *Communications ACM*, May 1988, pp. 532–533, doi: 10.1145/42411.42415.
[5] M. D. Hill and M. R. Marty, "Amdahl's law in the multi-core era," *Computer*, vol. 41, no. 7, pp. 33–38, July 2008.
[6] X. H. Sun and L. M. Ni, "Another view on parallel speedup," in *Proceedings Supercomputing '90*, Nov 1990, pp. 324–333.
[7] X.-H. Sun and Y. Chen, "Reevaluating Amdahl's law in the multi-core era," *J. Parallel Distrib. Comput.*, vol. 70, no. 2, pp. 183–188, Feb. 2010, doi: 10.1016/j.jpdc.2009.05.002.
[8] A. Kandalintsev and R. Lo Cigno, "A behavioral first order CPU performance model for clouds' management," in *International Congress on Ultra Modern Telecommunications and Control Systems and Workshops*, 10 2012, pp. 40–48.
[9] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, "CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Softw. Pract. Exper.*, vol. 41, no. 1, pp. 23–50, Jan. 2011, doi: 10.1002/spe.995 .
[10] E. Track, N. Forbes, and G. Strawn, "The end of Moore's law," *Computing in Science Engineering*, vol. 19, no. 2, pp. 4–6, Mar 2017, doi: 10.1109/MCSE.2017.25.
[11] B. Juurlink and C. H. Meenderinck, "Amdahl's Law for Predicting the Future of Multicores Considered Harmful," *SIGARCH Comput. Archit. News*, vol. 40, no. 2, pp. 1–9, May 2012.
[12] L. A. Barroso, S. Iman, M. Dubois, and K. Ramamurthy, "RPM: a rapid prototyping engine for multiprocessor systems," *Computer*, vol. 28, no. 2, pp. 26–34, Feb. 1995.
[13] A. K. Nanda, K.-K. Mak, K. Sugavanam, R. Sahoo, V. Soundararajan, and T. B. Smith, "MemorIES: A programmable, real-time hardware emulation tool for multiprocessor server design," vol. 35, 2000.
[14] R. A. Bergamaschi, I. Nair, G. Dittmann, H. D. Patel, G. Janssen, N. Dhanwada, A. Buyuktosunoglu, E. Acar, G.-J. Nam, D. Kucar, P. Bose, J. A. Darringer, and G. Han, "Performance modeling for early analysis of multi-core systems," in *5th IEEE/ACM/IFIP CODES+ISSS*, 2007.
[15] J. Wenjie Jiang, T. Lan, S. Ha, M. Chen, and M. Chiang, "Joint VM placement and routing for data center traffic engineering," pp. 2876–2880, 3 2012, doi: 10.1109/INFCOM.2012.6195719 .
[16] V. Shrivastava, P. Zerfos, K.-w. Lee, H. Jamjoom, Y.-H. Liu, and S. Banerjee, "Application-aware virtual machine migration in data centers," in *Proceedings - IEEE INFOCOM*, 2011.
[17] C. Tang, M. Steinder, M. Spreitzer, and G. Pacifici, "A scalable application placement controller for enterprise data centers," in *16th International World Wide Web Conference, WWW2007*, 01 2007, pp. 331–340, doi: 10.1145/1242572.1242618 .
[18] M. Feldman, K. Lai, and l. Zhang, "The proportional-share allocation market for computational resources," *IEEE Transactions on Parallel and Distributed Systems*, vol. 20, pp. 1075 – 1088, 09 2009, doi: 10.1109/TPDS.2008.168 .
[19] A. Kandalintsev, "Application interference in multi-core architectures: Analysis and effects," Ph.D. Thesis, Universita degli Studi di Trento, Italy, Department of Information Engineering and Computer Science, Apr. 2016.
[20] D. Zaparanuks, M. Jovic, and M. Hauswirth, "Accuracy of performance counter measurements," in *Proc. of IEEE ISPASS*, Boston, Massachusetts, USA, April 26-28 2009, pp. 23–32.
[21] V. M. Weaver and J. Dongarra, "Can hardware performance counters produce expected, deterministic results," in *3rd Workshop on Functionality of Hardware Performance Monitoring*, Atlanta, GA, 12-2010 2010.