Modeling Irregular Kernels of Task-based codes: Illustration with the Fast Multipole Method

Emmanuel Agullo, Bérenger Bramas, Olivier Coulaud, Luka Stanisic, Samuel Thibault

Modeling Irregular Kernels of Task-based codes: Illustration with the Fast Multipole Method

Emmanuel Agullo, Bérenger Bramas, Olivier Coulaud, Luka Stanisic, Samuel Thibault

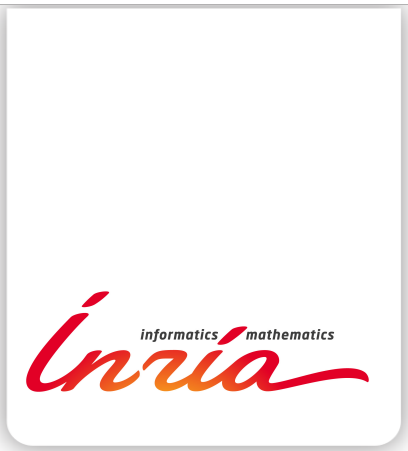
RESEARCH REPORT

N° 9036

February 2017

Project-Teams HiePACS and STORM

ISSN 0249-6399 ISRN INRIA/RR--9036--FR+ENG



Modeling Irregular Kernels of Task-based codes: Illustration with the Fast Multipole Method

Emmanuel Agullo *, Bérenger Bramas†, Olivier Coulaud*, Luka Stanisic*, Samuel Thibault*

Project-Teams HiePACS and STORM

Research Report n° 9036 — February 2017 — 35 pages

Abstract: The significant increase of the hardware complexity that occurred in the last few years led the high performance community to design many scientific libraries according to a task-based parallelization. The modeling of the performance of the individual tasks (or kernels) they are composed of is crucial for facing multiple challenges as diverse as performing accurate performance predictions, designing robust scheduling algorithms, tuning the applications, etc. Fine-grain modeling such as emulation and cycle-accurate simulation may lead to very accurate results. However, not only their high cost may be prohibitive but they furthermore require a high fidelity modeling of the processor, which makes them hard to deploy in practice. In this paper, we propose an alternative coarse-grain, empirical methodology oblivious to both the target code and the hardware architecture, which leads to robust and accurate timing predictions. We illustrate our approach with a task-based Fast Multipole Method (FMM) algorithm, whose kernels are highly irregular, implemented in the ScalFMM library on top of the StarPU task-based runtime system and the SimGrid simulator.

Key-words: Mathematical Software, Modeling and simulation, Parallel computing methodologies, fast multipole method, task-based programming, runtime system

* Inria, Hiepacs Project, 350 cours de la Libration, 33400 Talence, France. Email: Surname.Name@Inria.fr

† Max Planck Computing and Data Facility, Garching

RESEARCH CENTRE BORDEAUX – SUD-OUEST

200 avenue de la Vieille Tour
33405 Talence Cedex

Modélisation de noyaux irréguliers pour des codes à base de tâches : illustration avec la méthode des multipôles rapide

Résumé : L'augmentation significative de la complexité matérielle qui s'est produite ces quelques dernières années a amené la communauté de calcul haute performance à mettre au point de nombreuses bibliothèques scientifiques sur le principe d'une parallélisation à base de tâches. La modélisation de la performance des tâches individuelles (ou noyaux) qui les composent est cruciale pour faire face aux multiples challenges aussi variés que la réalisation de prédictions de performance précises, la mise au point d'algorithmes d'ordonnancement robustes, l'optimisation des applications, etc. La modélisation à grain fin telle que l'émulation et la simulation à la précision du cycle peut donner des résultats très précis. Toutefois, non seulement leur coût élevé peut être prohibitif mais elles requièrent de surcroît une modélisation très fidèle du processeur, ce qui les rend difficiles à déployer en pratique. Dans ce papier, nous proposons une méthodologie alternative, à plus gros grain, empirique, transparente à la fois pour le code et l'architecture cibles, ce qui permet des prédictions robustes et précises. Nous illustrons notre approche avec une méthode multipolaire rapide (FMM) à base de tâches, dont les noyaux sont hautement irréguliers, implémentée dans la librairies ScalFMM au-dessus du moteur d'exécution StarPU et du simulateur SimGrid.

Mots-clés : Logiciels mathématiques, simulation et modélisation, méthodologie pour le calcul parallèle, méthode des multipôles rapide, programmation à base de tâches, moteur d'exécution

1 Introduction

The emergence of more and more complex and versatile hardware architectures led the High Performance Computing (HPC) community to use relatively high level programming paradigms for designing scientific libraries. Task-based programming is certainly one of the most popular of these approaches. Not only many robust fully-featured task-based runtime systems [15, 6, 34] are now available to support it but the introduction of the *depend* clause in the revision 4.0 of the OpenMP standard now allows for its adoption by a broader audience.

Task-based programming requires an algorithm to be cast into individual, well identified pieces, called tasks (or kernels). This requirement may be viewed as a constraint when programming a complex numerical method. On the other hand, it ensures a clear separation of concerns, which can be exploited to allow the programmer to focus on designing advanced numerical algorithms while delegating the orchestration of the execution of the tasks to a runtime system relying on generic but sophisticated scheduling policies [32]. This separation of concerns is also a key factor for ensuring fast and accurate performance predictions. While timing of the regular algorithms such as dense linear algebra factorizations can be predicted transparently for the developer of the numerical algorithm [30], it may be much harder in the case of irregular algorithms. Indeed, such algorithms may rely on tasks with highly irregular workloads whose modeling is extremely complex. For instance, [29] showed that a sparse direct method requires a deep understanding of the parameterisation of the numerical algorithm to achieve faithful timing predictions. The employed methodology for successfully achieving this objective required the programmer to be an expert of both the considered numerical methods (a special sparse direct factorization, namely the multifrontal QR factorization) and statistical analysis.

In the present article, on the contrary, we propose to extend the separation of concerns enabled with task-based programming so that the modeling of the individual tasks becomes an assignment well distinct from the design of the numerical algorithm itself. While this exercise cannot be fully transparent, we propose a procedure that aims at being as much automatic as possible, yet requiring the programmer of the numerical algorithm to explicitly provide all parameters impacting the behavior of the task. From this list of parameters, a model of the task is then computed, relying on standard, multiple linear regression techniques. Anticipating the full list of relevant parameters is often a non trivial duty when dealing with irregular kernels, even for specialists of a numerical method. This is why the procedure we propose not only automatically models the behavior of the tasks but also provides an assessment of the relevance of the provided parameterisation. The programmer may then decide to enrich the list of parameters and trigger a new statistical analysis until the model is considered accurate enough. We illustrate our discussion with a highly irregular kernel, namely the Multipole to Local (M2L) operator arising from the Fast Multipole Methods (FMM) algorithm.

The rest of the paper is organised as follows. Section2 presents related work on modeling irregular kernels, both in a general context and in the special

context of task-based programming considered in the present study where a kernel can be canonically associated with a task. Section3 presents the FMM and more particularly the irregular M2L kernel used to illustrate our discussion. Section4 presents the methodology we propose to model such irregular kernels. We assess our methodology with the overall performance prediction of the FMM and all of its kernels in Section5 before concluding (Section6).

2 Related work

Modeling computational kernels is one of the major challenges of HPC. Not only is it critical for understanding the behavior of the developed numerical algorithms on current machines, but it is also necessary for anticipating the trends on future machines (and possibly for adapting their design according to these anticipations). It furthermore plays a major practical role in the design of modern software stacks, for instance for the design of generic advanced scheduling policies or robust auto-tuning schemes. The notion of computational kernel is itself very much context-dependent, the separation between two kernels being arbitrary in general. When the bounds of a kernel are not explicitly specified by the programmer within his code, some default strategies may be employed. For example, in a distributed memory code relying on the Message Passing Interface (MPI) [17], a kernel may be implicitly defined as a portion of code between two consecutive MPI calls. On the contrary, in the specific context of task-based programming, a kernel can be canonically associated with a task. We present an overview of general techniques employed for modeling kernels in general and in the specific case of task-based programming in sections2.1 and2.2, respectively.

2.1 Modeling kernels in a generic context

Authors of scientific HPC libraries and applications usually have a deep understanding of the algorithms they design and of the kernels they are composed of. When optimizing their code, they often first derive basic metrics such as the rough number of operations and volume of communication induced by the variants they consider. Such an analytical modeling is thus certainly the most common (and actually almost systematic) approach used in HPC domain. Different levels of analytical models may be considered depending on the objective. In some cases, relatively trivial models may be sufficient. For instance, the very basic roofline model has been used for predicting the respective behavior of FMM and other prominent algorithms on future machines in [7]. [13] proposed more advanced analytical modeling of the FMM for tuning their performance on heterogeneous architectures. While this analytical procedure may be extremely relevant in some contexts, it may lack of accuracy when aiming at predicting the performance of non trivial kernels on modern, complex hardware architectures.

For this reason, coarse-grain simulators such as BigSim [35] or SimGrid [12] have emerged. The complex kernels are abstracted with simple models based purely on the analysis of the benchmarks or of the full application executions,

on the real machine. The derived models are then possibly fine-tuned with few empirically measured coefficients. Even though much better control of the bias is provided, the accuracy loss due to modeling approximations is still hard to evaluate. Additionally, finding the right level of abstraction is often not trivial.

To obtain even more realistic models, many researchers use emulators and cycle-accurate simulators [8, 23, 25]. This method is considered to provide the most faithful predictions as it simulates the execution of every instruction. However, these executions can be up to 1 million times slower than the original running time [20] of the code, which makes this approach limited, especially for the large irregular kernels. In such a case, due to the prohibitive costs of the required exhaustive studies, researchers tend to rely on various interpolation and extrapolation techniques to construct their models, which may bring unwanted bias [33].

There is a whole spectrum of solutions in between these major approaches. Structural Simulation Toolkit [27] is a highly modular framework that provides both coarse-grain and fine-grain models, allowing users to find a good trace-off. An interesting approach is proposed in the PMaC framework [28], where kernels are characterized with certain performance parameters such as the number of floating-point instructions, number of memory accesses, cache miss rates, etc. This code signature is later convoluted with the description of the machine to obtain the overall kernel performance prediction.

2.2 Modeling kernels (or tasks) in a task-based context

The high degree of separation of concerns brought by task-based paradigms can be exploited for refining kernel modeling. The main reason is that kernels may be canonically associated with tasks, which are explicitly defined and parameterized by the programmer. This information can help improving the procedures discussed in Section2.1. Not only the (hard) problem of defining relevant boundaries of a kernel has necessarily been thoroughly considered by the programmer when he turned his algorithms into a task-based scheme, but a list of relevant parameters must have also been defined and exposed.

In the case of regular algorithms, the list of the parameters directly exposed by the programmer is generally sufficient for characterizing the behavior of the algorithm. For instance, in the case of dense linear algebra, modern tile algorithms [11] rely on kernels that depend solely on a single parameter, the tile size. They can then be simply modeled with a coarse-grain approach as follows. The task associated with the considered kernel is benchmarked with a fixed tile size, followed by an extraction of the distribution or of a simple mean value from all the observations. Since the optimal tile size is often well known for every type of processing unit, there is a need for only few different models. This simple *history-based* approach is often robust enough to capture the behavior of regular kernels. Still, in a multicore context, concurrency may remain very challenging to handle as significant performance degradation can occur if multiple kernels are executed in parallel (e.g., for CPUs with shared cache memories). There have been several independent works on modeling kernels used in tile al-

gorithms in that context, namely [26] on top of the OmpSs [15] runtime system, [30] for StarPU [6] and [18] for the OmpSs, StarPU and QUARK [34] runtime systems.

When kernels have more irregular workloads, such as in sparse linear solvers or FMM, the amount of work that has to be performed by a kernel may strongly vary from one kernel execution to another. Furthermore, this amount may be dependent on symbolic structures (structure of a sparse matrix, distribution of particles, etc.) that are often not directly exposed by the programmer as quantifiable data. Therefore, there is a two-fold challenge for capturing the irregularity of the kernel in its modeling. First, a strategy must be decided for dealing with the variability of the data the kernel operates on. Second, it may be necessary to provide extra quantifiable parameters to better handle the impact of the symbolic structure of the data on performance. [14] modeled the SuperLU [19] sparse direct solver. It implements a supernodal method, which can be cast into a sequence of dense operations with variable input sizes. Although SuperLU does not rely on a generic runtime system, it is implemented as a sequence of tasks consisting of dense operations and handled with an internal scheduler. To deal with the variability of the input parameters, the authors proposed a hybrid approach consisting of a baseline history-based scheme extended with a simple interpolation mechanism. Some data are accumulated during preliminary benchmarks on a collection of matrices for setting up the history. When a new matrix is processed and a kernel with new input values is called, the estimation is computed via a linear interpolation. Although modeling a sparse linear solver, the supernodal method considered in [14] was composed of dense kernels. When the kernels furthermore depend themselves on symbolic data, their modeling may be much harder. For instance, the kernels of the multifrontal QR factorization of the task-based qr_mumps solver [10, 5] deal with irregular staircase structures. In [29], the authors had to expose extra-parameters to accurately account for the number of operations involved in those complex, internal data structures.

3 Task-based FMM with an emphasis on the M2L operator

Originally introduced in [16], the FMM is considered as one of the top ten algorithms of the twentieth century [31]. A wide range of applications, such as molecular dynamic, astrophysics, vertex method, boundary element method (BEM) or radial basis functions, is now accelerated by the FMM. The main idea consists of reducing the quadratic complexity of pair-wise interactions to a linear or a linearithmic one by performing a hierarchical decomposition of the space into so-called cells (hierarchical subdivisions of the space) and applying a tree-based algorithm on that hierarchical space decomposition. Each node in that tree represents a cell of the space, the root node of the tree being the whole space while children nodes recursively represent spatial subdivisions. Recent

studies [21, 4] have proposed to design the FMM as a task-based algorithm in order to exploit modern architectures.

The goal of this section is to illustrate how task parameters impact the behavior of the corresponding FMM tasks due to the inherent irregular nature of the algorithm. Parameters may either be *direct* when they are immediately known or *indirect* when assessing them requires extra processing. In this section we show that irregular algorithms such as the FMM are inherently composed of irregular tasks that direct parameters may not be sufficient to fully characterize. We illustrate our discussion with a key task of the FMM involved for computing long distance interactions (namely, the M2L task), showing that all three direct parameters of that task (the level $TreeLevel$ in the tree, the number of cells $NbCells$ involved in the task and the $IntervalSize$ grouping parameter) fail to model its computational complexity when considering an unstructured test case. We introduce an extra indirect parameter, the number of interactions $NbInteractions$ between cells involved the task, and show that the evaluation of this parameter may be required to obtain a fine modeling of the task. **Readers only interested in the modeling of irregular kernels of task-based codes (and not particularly in FMM) may proceed to Section 4 with this background in mind.** The rest of this section is dedicated to illustrate that on a particular example. For that purpose, we further introduce the FMM algorithm (Section 3.1) and M2L operator (Section 3.2). After a brief review of task-based FMM (Section 3.3), we then explain how M2L operator can be designed as a task (Section 3.4), and finally we show how the aforementioned direct and indirect parameters impact the computation of the task.

3.1 FMM Algorithm

We consider the use of the FMM to compute the electrostatic field

$$\vec{E_i} = -\sum_{j \neq i} \frac{q_i q_j}{4\pi\epsilon_0} \frac{(\vec{x_i} - \vec{x_j})}{|\vec{x_i} - \vec{x_j}|^3} \tag{1}$$

of N particles $\vec{x_i}$ with charge q_i.

The key-point of the FMM algorithm is to approximate the far-field – the interactions between far particles – while maintaining the desired accuracy, exploiting the property that the underlying mathematical kernel decays with the distance between particles. The interactions between close particles are still computed with a direct particle-to-particle (P2P) method, but the far-field is processed with the following algorithm. A recursive subdivision of the space is performed in a pre-processing symbolic step (see Figure 1). This recursive subdivision is usually represented with a hierarchical tree data structure, and we call the height h of the tree the number of recursions. The name of the tree is related to the dimension of the problem, but in the current study we use the term *octree* to refer to the FMM tree for any dimension. Figure 1 is an example of an octree with an explicit correspondence between the spatial decomposition and the data structure and we see that each cell represents its

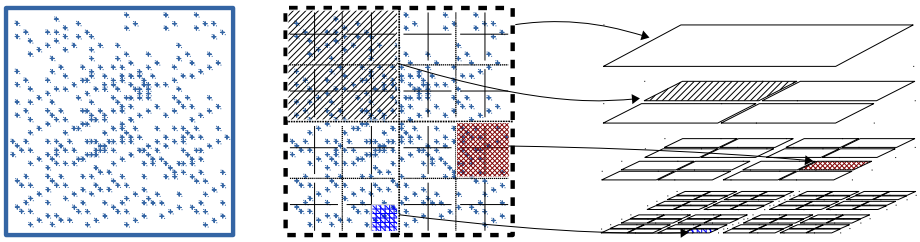


Figure 1: 2D space decomposition (Quadtree). Grid view and hierarchical view.

descendants composed of its children and sub-children. The multipole (M) of a given cell represents an approximation of the charge of its descendants. On the other hand, the local part (L) of a cell c represents the potential field due to all cells well-separated of cell c that will be applied to the descendants of c. Relying on those recursive data structures, the FMM algorithm proceeds as follows (see Figure 2).

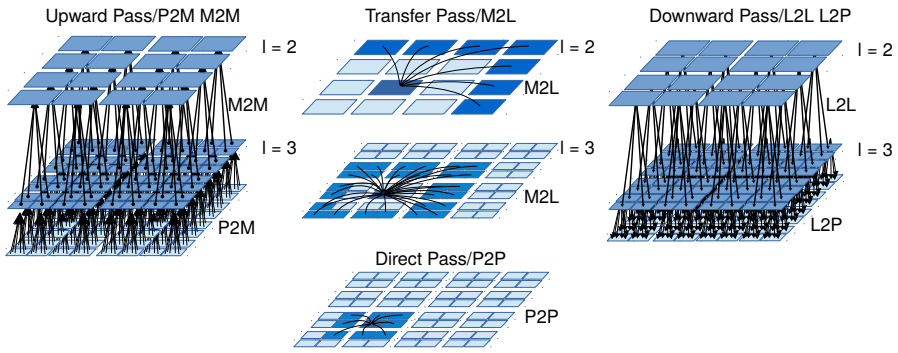


Figure 2: Different steps of the FMM algorithm; upward pass (left), transfer pass and direct step (center), and downward pass (right).

In an upward pass of the FMM, the physical values of the particles are aggregated from bottom to top using the Particle-to-Multipole (P2M) and Multipole-to-Multipole (M2M) operators. After this operation, each cell hosts the contributions of its descendants. In the transfer pass, the Multipole-to-Local (M2L) operator is applied between each cell and its corresponding interaction list at all levels. After the transfer pass, the local part of all the cells are filled with contributions. The downward pass aims to apply these contributions to the particles. In this pass, the local contributions are propagated from top to bottom with the Local-to-Local (L2L) operator and applied to the particles with the Local-to-Particle (L2P). Consequently to these far-field operations, the particles

have received their respective far contributions.

3.2 M2L operator

The computation occurring within all these operators depend on the particle distribution, which is unstructured in most practical applications. As a consequence, the number of operations and the time required to apply a given operator may be highly irregular, even during the course of a single FMM computation. Since the goal of the present study is to discuss the modeling of irregular kernels rather than the FMM itself, we focus only on one particular operator, M2L. In many physics, M2L is indeed the dominant operator in the far field, hence the most critical kernel to model accurately.

In the transfer pass, the M2L operator is applied between each cell and its corresponding interaction list at all levels. The interaction list for a given cell c at level l is composed of the children of the neighbors of c's parent that are not direct neighbors/adjacent to c. As illustrated in Figure 3, the interaction list of cell 26 (red cell) is given by the 27 cells represented in orange. This is actually the maximum number of interactions $m2l_{max}$ a cell may have (the bound being $m2l_{max} = 6^d - 3^d$ where d is the space dimension in general, hence $m2l_{max} = 27$ with $d = 2$ and $m2l_{max} = 189$ with $d = 3$). On the other hand, some cells may have a lower level of interactions. For instance, the cell 22 has only 7 cells in its interaction list. On irregular distributions, cells may have much fewer interactions in low density areas as further illustrated in Section 3.4 . All in all, a particle/multipole may have a number of interactions ranging between 0 and $m2l_{max}$.

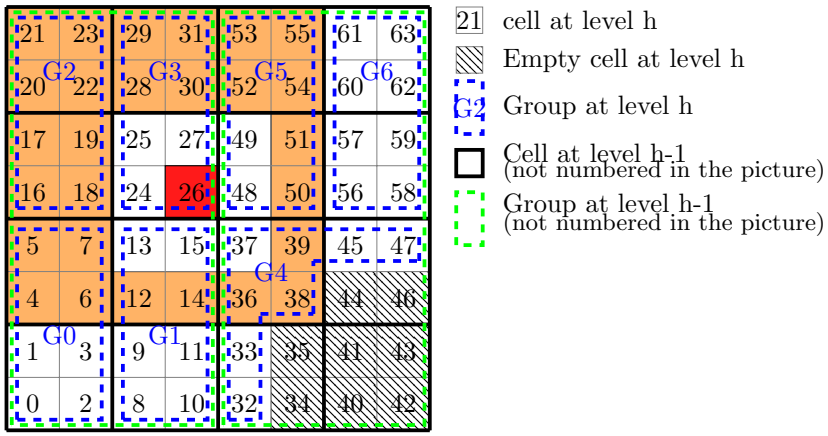


Figure 3: The octree at level 3, the cells are ordered by their Morton index and the cells numbered $34, 35, 40 - 44, 46$ are empty. The 56 non empty cells are grouped in 8 groups (blue) of size $ng = 8$. At level 2, we only have 16 cells and two groups (green). The group $G4$ starts at Morton index 32 and finishes at index 47.

3.3 Task-based FMM

Like many important numerical methods, the FMM has been turned into task-based algorithms for exploiting modern architectures (see [21, 4, 2] for instance). A key point for achieving high performance with a task-based model is to design tasks of well-chosen granularity. On the one hand, tasks must be small enough so that multiple tasks can be executed concurrently, ideally providing work for each CPU core. On the other hand, they must be large enough so that the overhead of the task management performed by the runtime system remains low in comparison with the time spent for performing numerical computation. [4] introduced a blocked data structure for the octree, called group-tree, which increases the granularity of the FMM operators and yields higher performance than operating at the granularity of a single cell/leaf. In a so-called group-tree, we aggregate the elements – cells or leaves – into blocks of size n_g. This granularity parameter can then be tuned to efficiently trade-off the overhead of the runtime with the level of concurrency. Figure 3 illustrates this grouping scheme by representing groups at level h and $h-1$ by blue and green dashed rectangles, respectively.

3.4 M2L task (or M2L kernel)

Tasks may then be defined as an aggregation of applications of an operator. We again illustrate our discussion with the M2L case. Cell 26 (red cell in Figure 3) has interactions with four cells [28-31] within its own group G3 (which contains the eight cells numbered 24 up to 31) and other interactions with 23 other cells scattered within five other groups (G0, G1, G2, G4, G5). All in all, the eight cells [24-31] composing group G3 have 24 interactions with one another and 150 interactions in total with five other groups (G0, G1, G2, G4, G5). We name M2L_in and M2L_out tasks operating on a single group and on two different groups, respectively. All the 174 interactions related to the eight cells composing group G3 are thus split into one M2L_in task (corresponding to 24 interactions) and five M2L_out tasks (corresponding to the remaining 150 interactions).

However, the irregularity of the distribution may lead to varying numbers of interactions. For instance, G4 has an irregular shape (cells 32-33, 36-39, 45, 47) due to empty cells (34-35, 40-44, 46). Yet composed of eight cells too (the group size), the irregular shape of G4 M2L_in induces more (32) interactions than in the case of G3 (24). Moreover, in the case of irregular particle distributions, the octree is pruned so that no computation occurs on regions of the space where there are no particles. As a consequence, the sparsity of the distribution and the shape of the group lead to workloads ranging from 0 to $n_g \times m2l_{max}$ interactions. Note that in actual numerical simulations, the group size n_g being typically much larger than eight, the number of interactions within a group tends to be higher than the number of interactions with external groups. In the rest of the paper, we focus solely on M2L_in tasks operating within a single group. We furthermore indifferently name it "M2L task" or "M2L kernel" as we canonically associate a kernel with a task in a task-based context.

Hence, the number of interactions $NbInteractions$ within a M2L_in task is an important parameter impacting the execution time of the task. However, in an actual production code this parameter is typically not exposed as a quantifiable data prior to the task execution, as its computing can take a significant amount of the valuable processor time. We name *indirect parameter* such type of parameters in the rest of the paper. It is opposed to *direct parameters* which are immediate to evaluate in the baseline code. For the M2L_in task, these direct parameters are the level of the octree ($TreeLevel$), the number of cells in the group, ($NbCells$, bounded by n_g) and the size of the Morton interval covered by the first and last cells in the group ($IntervalSize$). Depending on the expected robustness and accuracy of the kernel modeling, we show in the following section that the instrumentation of the code with some indirect parameters although costly may yet be necessary.

4 Modeling irregular computation kernels

We now propose a coarse-grain, empirical methodology for modeling irregular tasks, illustrating the required effort through the M2L_in kernel described in the previous section. This kernel is a good representative of tasks arising in the modern HPC applications, which are extremely challenging to model. First, the kernel duration depends on a combination of multiple parameters, some of which are costly to define, e.g., the number of interactions for the M2L_in task. Second, the value of these parameters is greatly varying, as seen with the impact of the particles distribution or the group granularity for the M2L_in.

To find the good estimation of the kernel duration, accurate regardless of its inputs, we propose to rely on *multiple linear regression models*. Multiple linear regression attempts to model the relationship between multiple explanatory variable vectors $X_{1..k}$ and a response variable vector Y by fitting a linear equation of the form Equation (2) to the observed data:

$$Y = a + bX_1 + cX_2 + ... + zX_k + \epsilon \tag{2}$$

In our approach, we consider that the response variable is the execution time T_{kernel} of a kernel while each explanatory variable X_i is a combination of parameters M, N, K (the direct $TreeLevel$, $NbCells$, $IntervalSize$ and indirect $NbInteractions$ parameters of the M2L_in task in our example). Which parameter combinations are explanatory depends on the kernel algorithm, but also on the machine architecture. Indeed, various platform characteristics, such as cache sizes, can have an influence, which makes a purely theoretical prediction of the parameter combinations unreliable. Our models follow the same assumptions as the standard multiple linear regression (linearity, independence of errors, homoscedasticity, etc.), except for the independence of the explanatory variables. Indeed, we adopt an empirical approach where we do not make *a priori* assumption on the combination of parameters that shall form the explanatory variables. Instead we test a relevant subset (see phase 2 - modelization

- detailed in4.2) of combinations based on products of parameters of the form provided in Equation (3)

$$T_{kernel} = a + b(M^{\alpha_1} \times N^{\beta_1} \times K^{\gamma_1}) + c(M^{\alpha_2} \times N^{\beta_2} \times K^{\gamma_2}) + ... \quad (3)$$

where M, N, K are the direct and indirect parameters of the kernel, α, β, γ their exponents and finally a, b, c are coefficients which mostly depend on the machine speed. Note that it is also possible to assess other combinations including the application of other functions (logarithmic, exponential, etc.) using the same principles. For a matter of simplicity we do not discuss these options further in the present article, but our model (and code) fully support those cases. In practice, for most cases a research among combinations with integer exponents offered by Equation (3) is likely to deliver a sufficiently robust modeling.

Having a faithful kernel model allows to calculate kernel duration prediction online, during another application run. In order to be able to make such estimations, a runtime system, which orchestrates a task-based application execution, needs to have all the elements from Equation (3). The values of the parameters M, N, K should be made visible to the runtime system, and the right equation together with exponents and coefficients have to be computed in advance. All these require modifying the application code and doing the initial thorough offline analysis of the experiment traces. We divided such an effort in three main phases presented in Figure4 an described as following:

1. **Instrumentation**: Manually adding direct and indirect parameters M, N, K to the kernel structure, so their values can be retrieved during the application execution. These parameters are typically provided by the *application developer*, who has a good knowledge of kernel performance and implementation details.

2. **Modelization**: Finding the right parameter combinations together with the exponents α, β, γ. These are provided by the *researcher doing the full statistical analysis* of the experimental traces produced by kernel benchmarks. The offline analysis is based on multiple linear regressions and in our case it is performed using R language [24]. It aims at identifying the optimal model and validating its accuracy.

3. **Calibration**: Computing the coefficients a, b, c for a specific machine. This is performed when an *application user* executes the application for a desired range of input configurations on a target platform. Since it is calculated automatically online by the runtime system, using ordinary least squares method, the user running the experiments may even not be aware of the previous phases or the algorithms used to compute the coefficients. It is also possible to skip this phase if the same person has previously performed the modelization part, as the outputs of statistical analysis already contain the coefficient values. In such case, performance models for the runtime system need to be manually written.

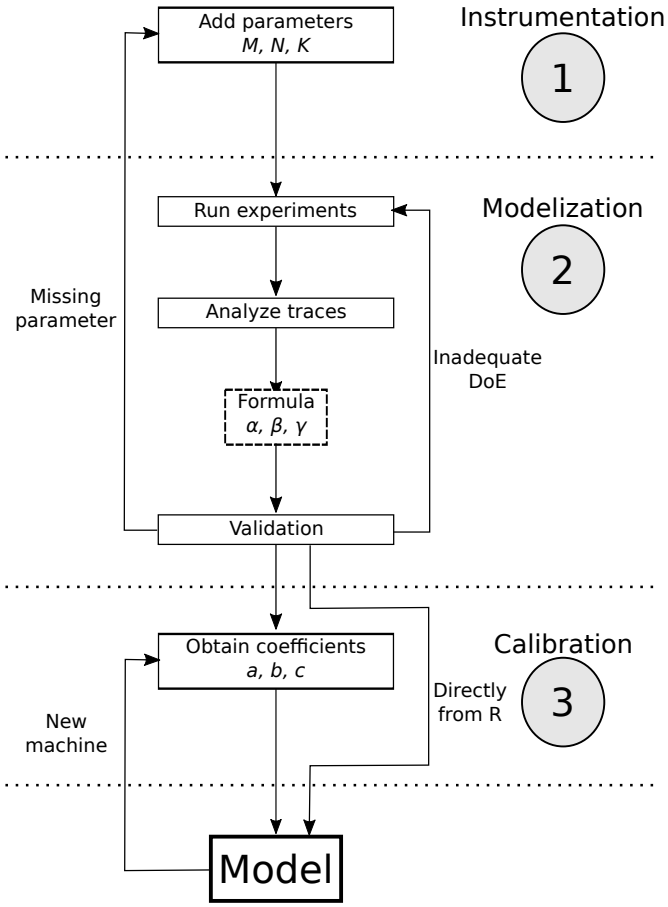


Figure 4: Summary of the kernel modeling workflow. The process is divided into three phases and although it is essentially linear, several loopbacks are often needed due to the missing parameters or initially inadequate design of experiments.

In the rest of this section, we provide more details for each step of these three phases along with the additional explanations of the actions presented in Figure4. **Note that the detailed explanations of those three steps can be read independently from one another.**

4.1 Instrumentation phase

In this stage, the application developer is expected to provide all the important parameters on which each kernel execution time depends. These are annotated as M, N, K in Equation (3) and can represent the height and weight of a matrix, number of elements to process, stride size, etc. For example, in the specific case of the M2L_in kernel, these parameters are the level of the tree, the number of cells of the group and the size of the interval (see Section3.4 for more details).

It is highly advised to provide as many different parameters as possible to ensure that the final model contains the best parameter combination for explaining the kernel duration. Application developers should not hesitate to add a parameter whose utility intuitively seems uncertain, since the analysis will easily ignore it and the performance lost for having one parameter more in the list is negligible. Moreover, even for the application experts, it is hard to foresee the significance of a parameter for a wide range of machines and input configurations.

In certain cases, mostly for highly irregular kernels, the directly accessible input parameters are not sufficient to explain kernel durations. The sequence of the executed code will depend on matrix and application structure as well as the data placement and already-completed kernels. Therefore, the application developer needs to provide a supplementary code for calculating additional, indirect, parameters which will help in estimating kernel duration. This code is executed before the kernel and it provides new parameters, such as the number of children or number of neighbor tasks, the estimated number of flops, etc. In the Section3.4, it is described how the number of interactions parameter is an extremely important indirect parameter for estimating the workload of the M2L_in kernel. Computing these values introduces overhead to the overall application execution time, but it might be mandatory to obtain accurate models.

Program instrumentation demands some effort from the application developer, but the advantage is that this work has to be done only once per kernel.

4.2 Modelization phase

The application developer or the application analyst then needs to find the combinations of parameters and their exponents that allow for the most faithful model for each kernel. Considering Equation (3), this phase consists in finding the necessary number of parameter tuples and the exponents α, β, γ for each parameter of the tuple.

First an exhaustive benchmarking of the kernels needs to be performed. These benchmarks can be simple scripts, specially written for this purpose, that will assess kernel behavior with different input parameter values. Such an

approach is easy to implement for the kernels that are wrappers around standard BLAS/LAPACK functions. Additionally, this provides a lot of flexibility for testing various input scenarios. However, it is very difficult to develop similar benchmarking scripts for more complex kernels, which depend on the previously executed code. In such cases, one would need to construct a whole "mini-application", which is an abstraction of the original program. This would require a tremendous programming effort, thus it is rarely performed. Instead, the original application with carefully chosen input configurations is run.

The described benchmarks are executed on the targeted machines and they generate traces that contain durations of the kernel in addition to its parameter values. It is important to note that the benchmarks should be run separately over distinct processing units, since the algorithm and behavior of the kernels may greatly differ. On the other hand, parameter combinations should be the same for machines with a similar architecture.

Once traces from the benchmarks are collected, they are carefully analyzed offline using statistical tools, in our case R language. Analysis based on multiple linear regression is used to derive the most important parameter combinations and their exponents α, β, γ. In theory, this analysis could be completely automatized, but in practice it is extremely difficult to propose a general analysis script that will work for any kernel and that will take into account all possible phenomena observations may contain. Therefore, we propose to perform a basic initial analysis and then manually zoom on each kernel and examine any specific behavior. Even though such an approach may be slightly repetitive, it ensures that no kernel particularity is skipped.

The next step validates the accuracy of the models, typically evaluating parameters importance, confidence intervals, adjusted R^2 value and visually inspects model characteristics. Figure5 shows the most common analysis of the model, applied for the M2L_in kernel discussed above with all of its direct parameters.

The table at the top shows a summary of the multiple linear regression over the calibrated data. For each parameter combination and the constant in the first column (e.g., $TreeLevel$), an estimation of the corresponding coefficient (for $TreeLevel$ example this is 7.73×10^1) is provided along with the 95% confidence interval (for $TreeLevel$ example this is 7.32×10^1 - 8.14×10^1). The final computation of these coefficients for a given machine will be detailed in the next phase (see Section4.3) and at this point researchers need to inspect only the importance of each parameter combination as they are seeking for the optimal equation for computing the kernel duration. The three stars in the last column indicate that in this case all parameter combinations (except $IntervalSize^2$) appear significant. However, this does not necessarily mean that the model is perfect, on the contrary, it is often a product of overfitting. In fact, adjusted R^2 of the corresponding model is 0.90, which is considered as mediocre accuracy in our context.

The model weakness can be observed even better when inspecting the residuals in the two bottom plots of Figure5. On the left plot, one can clearly observe several structures, which indicates that there is a certain phenomena

$TreeLevel$	7.73×10^{1} $(7.32 \times 10^{1}, 8.14 \times 10^{1})$	***
$NbCells$	1.52×10^{2} $(1.46 \times 10^{2}, 1.59 \times 10^{2})$	***
$IntervalSize$	-5.55×10^{-1} $(-6.65 \times 10^{-1}, -4.45 \times 10^{-1})$	***
$TreeLevel^2$	-6.73 $(-7.10, -6.37)$	***
$NbCells^2$	-6.08×10^{-3} $(-7.94 \times 10^{-3}, -4.21 \times 10^{-3})$	***
$IntervalSize^2$	-3.63×10^{-12} $(-6.36 \times 10^{-12}, -9.04 \times 10^{-13})$	
$TreeLevel \times NbCells$	-4.68 $(-5.88, -3.48)$	***
$TreeLevel \times IntervalSize$	6.24×10^{-2} $(5.02 \times 10^{-2}, 7.46 \times 10^{-2})$	***
$NbCells \times IntervalSize$	1.81×10^{-4} $(1.39 \times 10^{-4}, 2.24 \times 10^{-4})$	***
$TreeLevel \times NbCells \times IntervalSize$	-2.08×10^{-5} $(-2.55 \times 10^{-5}, -1.61 \times 10^{-5})$	***
Constant	-2.27×10^{2} $(-2.40 \times 10^{2}, -2.14 \times 10^{2})$	***
Observations	4,987	
R^2	0.906	
Note:	*p<0.1; **p<0.05; ***p<0.01	

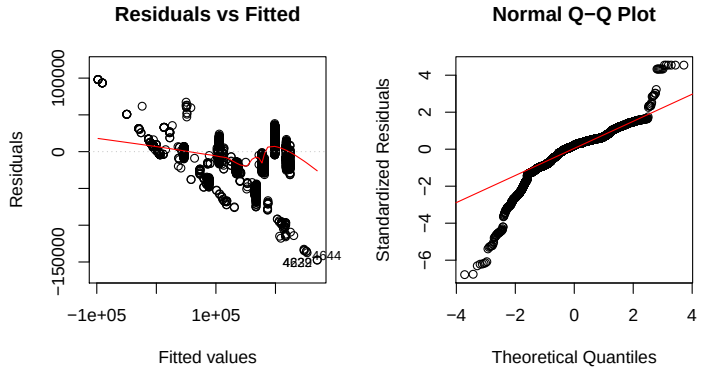


Figure 5: Initial model for M2L_in kernel. All parameter combinations (except $IntervalSize^2$) appear to be important, which is often a sign of overfitting. The residuals are structured and the distribution on $Q-Q$ plot is skewed, which both suggest a missing parameter.

not explained by the model. Many points on the same horizontal line represent repetitive occurrences of the kernel with the same parameter values, which is typical for a single experiment run with homogeneous data. The fact that there is some variability is indeed common, since executing exactly the same code on a real machine always leads to slightly different durations. However, a bad characteristic of this model is that the mean value of many of these groups is far from zero. This implies that the predictions of this model for an application execution that mostly contains kernel instances for these specific parameter values will be constantly overestimating or underestimating kernel duration. This can create a non-negligible cumulative error. Moreover, $Q-Q$ plot on the right in Figure5 shows the curved pattern which suggests that the residuals do not follow the normal distribution, as there are many points far from the red $y = x$ line. Therefore, such a model would in overall have a limited predictive power.

Hence, the model of M2L_in clearly needed improvements. Since adding parameter combinations with higher exponents only makes overfitting effect worse, application developers had to search for new, indirect, parameters that could help us explain the observed structures. The number of interactions of M2L_in task, described in Section3.4, came as a perfect solution, especially since the overhead required to obtain it proved to be harmless for the overall application performance.

Indeed, adding such a parameter greatly improved the model, as shown in Figure6. One can observe that there is no more structures and that the residuals are approximately equally distributed, homoscedastic and follow the normality law. Apart from few outliers, the model fits very well the data and the adjusted R^2 value of 0.999 is almost perfect. Furthermore, the obtained model is much simpler and thus better, as it depends solely on a linear combination of three parameters, without any parameter interactions or exponents. It is important to note that many other irregular kernels, including some other FMM kernels, have models and parameter combinations that are much more complex.

This method ensures obtaining the best possible combination of parameters, yet this does not guarantee that the model is completely faithful. Indeed, if the right parameters for describing the kernel duration were not provided by the application developer, the obtained model would only be a rough approximation based on what is available. Moreover, when later doing more extensive benchmarks for very different input configuration, new analysis frequently uncover the need for more complex kernel models. Therefore, if the models are not accurate enough for the future purposes, one needs to loop back to the instrumentation phase and include additional direct and indirect parameters, as we discussed with the M2L_in kernel and the addition of the number of interactions parameter.

Once the final models are obtained, the choice of parameter combinations and the values of exponents should stay the same for any machine with a similar architecture. The application code has to be modified one last time, putting combinations and exponents values into the source code. These small modeling extensions will persist even when the application source code around it evolves, unless the core of the application or its computation kernels change in the future,

TreeLevel	9.10×10^2 (8.46×10^2, 9.74×10^2)	***
NbCells	5.34 (5.17, 5.50)	***
NbInteractions	8.59×10^{-1} (8.58×10^{-1}, 8.60×10^{-1})	***
Constant	-7.09 (-7.50, -6.67)	***
Observations	4,987	
R^2	0.999	
Note:	*p<0.1; **p<0.05; ***p<0.01	

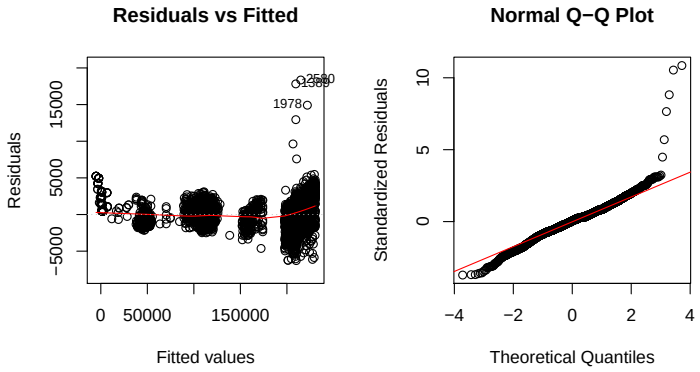


Figure 6: M2L_in model analysis when additional parameter (number of interactions) is added. Simpler model, higher adjusted R^2, smaller and unstructured residuals and a normal distribution all suggest that this model is much more accurate than the initial one in Figure5

which is rarely the case.

Finally, with the completed instrumentation and modelization, the only missing element to compute the kernel duration predictions is to obtain coefficients for each parameters combination, which can be found automatically through machine calibration.

4.3 Calibration phase

To compute the necessary coefficients (a, b, c in Equation (3)), the fully instrumented task-based application is executed on a target machine by the application user. The runtime system in charge of executing the task-based code obtains the parameter combinations from the program, then collects kernel execution times during the application run and at the end computes coefficients for a given machine using a least squares method. These coefficients are stored in performance model files, similar to classical history-based performance models. In the M2L_in example, this means keeping computed coefficients for the constant, $TreeLevel$, $NbCells$ and $NbInteractions$.

Calibration needs to be performed only once for a target platform. When changing the machine, the application should be re-executed in order to obtain new coefficient values for the new machine, but the parameter combinations remain unchanged. The same applies when studying different types or sizes of the problem. Indeed, the application is calibrated for a particular subset of input configurations. If the Design of Experiments (DoE) is correctly performed, these inputs should cover a wide range of possible application runs. However, for some applications this would require an extremely large experimental campaign, which is prohibitive in terms of time and machine resources. Hence, in practice, calibrated coefficients are applicable only for the domain in which calibration was performed.

Finally, it is important to note that collecting the kernel durations with corresponding parameters and computing coefficients is a completely automatic process and thus transparent to the application user.

4.4 Conclusion

When the model accuracy is validated in the phase 2 and the coefficients for a given machine and a subset of problems are obtained from models or carefully calibrated in the phase 3, the model is ready. It should be able to provide faithful predictions of the kernel duration for a wide range of inputs. In the case of M2L_in, the final model is presented in Equation (4).

$$T_{M2L} = -7.09 + 910 \times TreeLevel + 5.34 \times NbCells + 0.85 \times NbInteractions \quad (4)$$

Although the whole process of identifying the right kernel model is essentially linear, in practice even experienced researchers are obliged to go through many iterations. In the beginning, model validation often indicates that additional

parameters are needed to obtain a more accurate model. Moreover, modelization itself has to be repeated multiple times until it is established which DoE can provide the most general models. However, once the formula has been validated, all application users can benefit from it, as they only need to execute the calibration which generates the coefficients and with it the corresponding final model for a given machine. It is worth noting that if the implementation of the kernel is significantly modified, due to algorithm changes or adaptation to a different architecture (e.g., GPU kernel implementation), the model may significantly change as well, thus the whole workflow needs to be redone. Figure4 summarizes the most important parts of this process.

Finally, in order to use these models, one needs to simply call the function which recovers all the necessary elements from the kernel structure and performance files, and put them into an equation (such as Equation (4)) to calculate kernel duration estimation. These predictions can then be used in many different contexts, such as for optimizing scheduling policies or for doing performance simulations.

5 Evaluation of model accuracy through application performance prediction

Thanks to the accurate kernel models, durations of tasks can be estimated during the execution. These estimations can be used for many purposes, and in this paper we present a use case where the models are used by a simulation in order to predict the overall application performance.

To perform such an evaluation, we used a task-based version of ScalFMM library [4], an implementation of task-based FMM algorithm. This code is implemented on top of the StarPU runtime system [6], which is a dynamic task-based runtime system responsible for orchestrating execution of the program. Well identified computation pieces, called kernels (tasks), of the ScalFMM library are scheduled on different processing units by the runtime system, in order to exploit the parallelism. During the native execution, information about the input parameters and duration of each kernel are automatically collected by the runtime system. In the scope of this study, StarPU has been extend to fully support models based on multiple linear regression, and this addition will be publicly available in the next, 1.3, release of StarPU.

It is possible to easily benefit from new performance models using StarPU's simulation extension, based on the SimGrid framework. StarPU-SimGrid is a simulation/emulation tool which simulates the execution of a task-based application on a desired platform. The control part of the runtime system has been modified to inject delays instead of actually executing computation tasks (kernels), performing data transfers, memory allocations, etc. However, in order to obtain faithful simulation predictions, accurate task models are indispensable. For a more detailed explanation of StarPU-SimGrid tool and its accuracy, we refer readers to [30].

Hence, to validate our modeling approach, we followed the phases described in Figure4 and obtained models for 8 extremely irregular ScalFMM kernels. In order to provide a better comparison, we have also computed classical ("naive") models, which are basic kernel estimations based on a single parameter value (e.g., group granularity or number of particles). Finally, we performed an experimental campaign, where models were used to make kernel duration predictions that are injected into simulator and then the simulations are compared to the real machine executions.

5.1 Experimental settings

Performing an exhaustive study on various heterogeneous machines with a huge number of different application inputs would be extremely time demanding, thus we limited our research to only the most common problems studied by ScalFMM developers. Although the principles for doing an extended research would be exactly the same, the required experimentation, analysis and finally generated models would be slightly more complex. Indeed, proposing an optimal DoE that will allow for the most accurate and general models is often not trivial and it is a problematic that belongs to completely different research areas.

We have chosen a pragmatic path and relied on highly optimized input configurations, experimenting solely on a single homogeneous node (2 Dodeca-core Haswell Intel Xeon 2,5 GHz 128GB RAM machine). To calibrate our models and find the coefficients for the target machine, we executed application runs with 12 unique input configurations. 6 of them correspond to the uniform cube (volume) distribution while the other 6 are for a very different non-uniform ellipsoid (surface) distribution. For both cases, the problem size ranges from 1 million to 100 million particles. The tree height and granularity of the group where chosen according to the ScalFMM performance study presented in [1]. Finally, only 23 CPU cores were used for computation, while 1 core was always dedicated to the StarPU main thread, as such resource utilization proved to perform better compared to using all 24 cores for running kernels.

5.2 Description of evaluated models

In order to evaluate the gain provided by our methodology and multiple linear regression models, we have used 12 (6 cube and 6 ellipsoid) previously described input traces to generate five separate groups of kernel models.

First two represent a naive approach, where kernel duration is approximated using a single parameter value. For 4 ScalFMM kernels (L2L, M2L_in, M2L_out, M2M) that mostly depend on the granularity of the groups, this granularity input value was used to generate a simple linear model. For 4 other kernels (L2P, P2M, P2P_in, P2P_out), linear model was based solely on the problem size, i.e., total number of particles. We had to generate two separate models for each type of distribution (cube and ellipsoid), since kernels perform very differently in two cases and an unified model would be extremely inaccurate.

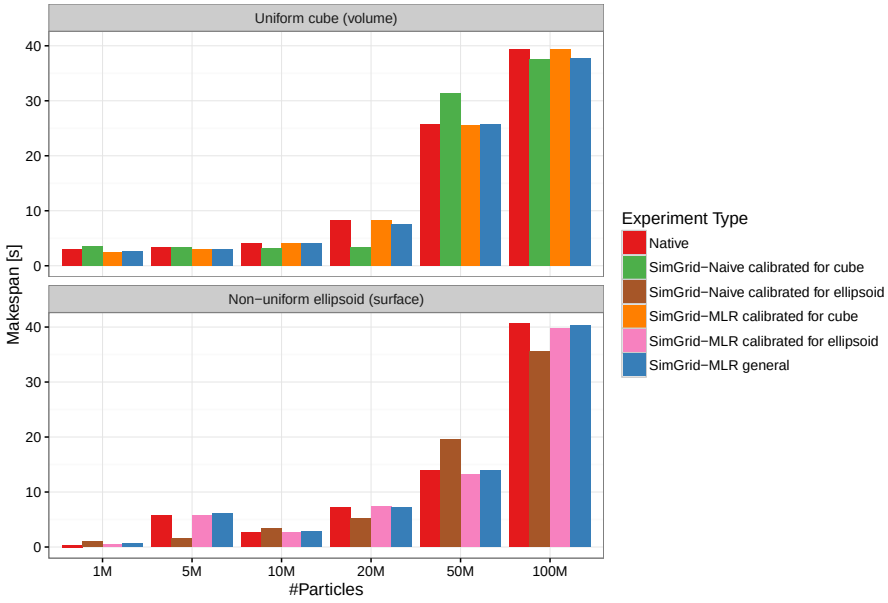


Figure 7: Overall execution time measured on a real machine (Native) and predicted by simulation (SimGrid). Simulation predictions with MLR models are very accurate, regardless of the particle distribution, number of particles, tree level or granularity.

This modeling approach corresponds to the techniques most commonly used in our field.

The rest of the models are based on multiple linear regression (MLR) and they are generated following the approach described in Section4. For MLR cube and MLR ellipsoid models, coefficients are automatically computed, each from 6 executions with the corresponding cube and ellipsoid distribution. Although these models are more accurate than the naive ones, they all have limited predictive power. Indeed, all these approaches contain one big disadvantage, as the models remain specific to the type of distribution they were measured on and cannot be used for any new distribution. This issue is resolved with our last MLR general model, which combines all 12 input traces to obtain the coefficients. Such a model is general, which is a very beneficial characteristic, but it is hard to foresee if its accuracy is matching the other models and represents well the real executions.

5.3 Validating model accuracy

The comparison of the overall application execution times is presented in Figure7. Although naive models are computed separately for each distribution, in most cases their overall prediction is far from reality (Native executions), espe-

cially for more irregular ellipsoid distribution. This illustrates how difficult it is to model complex ScalFMM kernels even for a small and relatively similar subset of input configurations. On the other hand, SimGrid performance predictions with MLR models are very close to the Native execution times, with only few percents of error. SimGrid-MLR general results are typically less accurate than the simulations with distribution specific MLR models, but the difference is negligible. Interestingly, SimGrid-MLR faithfully predicts the performance even for sub-optimal Native executions, such as the 5 million particles ellipsoid experiment where the group granularity was badly chosen.

Furthermore, the StarPU-SimGrid simulation provides not only the final application execution time, but also the trace of the execution. We illustrate this with a single example, 50 million particles with cube distribution, although other cases lead to very similar conclusions. One can observe in Figure8 how Gantt charts of the Native and SimGrid-MLR executions match almost perfectly, while the naive modeling approach resulted in similar scheduling yet with longer overall execution. Each type of ScalFMM kernel is represented with a distinct color, which allows to easily distinguish different phases of the application execution. Even though the scheduling is dynamic, close resemblance of the four traces suggests that the simulation mimics the real execution very faithfully. Some minor differences can be detected at the beginning of the execution, due to the simplistic modeling of the task insertion mechanism used by StarPU-SimGrid. The longer execution of SimGrid-Naive seems to be mostly the product of overestimated duration of the kernels at the final phases of execution.

Visual comparison of the traces is very important, however such a view is macroscopic and thus can hide certain phenomena. Figure9 shows the overall duration of each kernel, computed for the Native and SimGrid traces presented in Figure8. Overestimation of P2P_in and P2P_out kernel duration by naive approach explains why simulation with this model has much longer overall execution. Regarding the MLR models, for all 8 ScalFMM kernels the simulation predictions are close but rarely perfect, as one might have expected from looking only at the Gantt charts. This variance comes from the fact that all the models used in simulations are not solely computed from the Native execution they are now compared to, but from 6 or 12 different executions. The input configuration for each Native experiment run brings certain specificities and thus a model which was calibrated for many other inputs is likely to slightly miscalculate the kernel duration. This is even more enhanced for MLR general model, which also contains measurements for a completely different ellipsoid distribution. This analysis demonstrates how the statistically extremely accurate models, with adjusted R^2 of 0.99, can still be imperfect and have a certain prediction error when applied on a particular use case.

Finally, Figure10 shows the distributions of the kernel durations for the aforementioned traces. As naive models depend only on a single application configuration option, they always predict kernel duration with one constant value. This provides unrealistic executions and can be problematic for the use cases with more idle time, since dynamic scheduling of StarPU runtime system would behave very differently with such a kernel estimations. On the other

Figure 8: Even if runtime scheduler is dynamic, execution traces of Native and SimGrid-MLR match almost perfectly, while SimGrid-Naive execution is much longer.

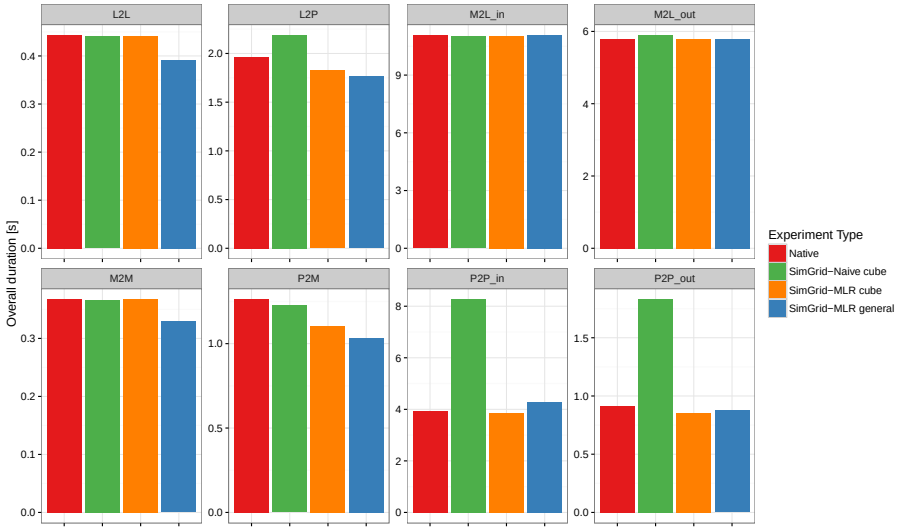


Figure 9: Comparing overall duration for each computation kernel between Native and SimGrid executions. The values are divided by the total number of CPU cores (23) to ease the comparison to the Figure8 and allow better estimation of the overall influence of each kernel.

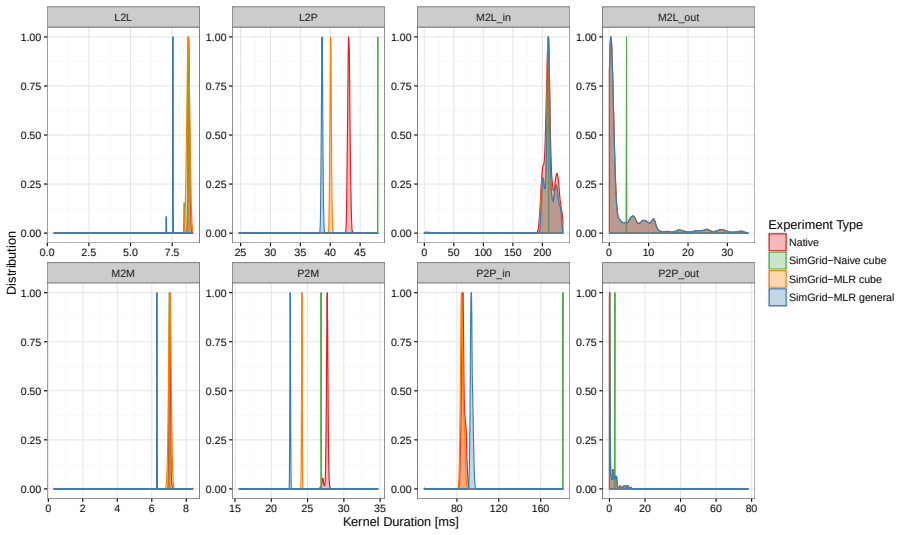


Figure 10: Kernel duration distributions. For many kernels, MLR models are completely overlapped with the Native distribution.

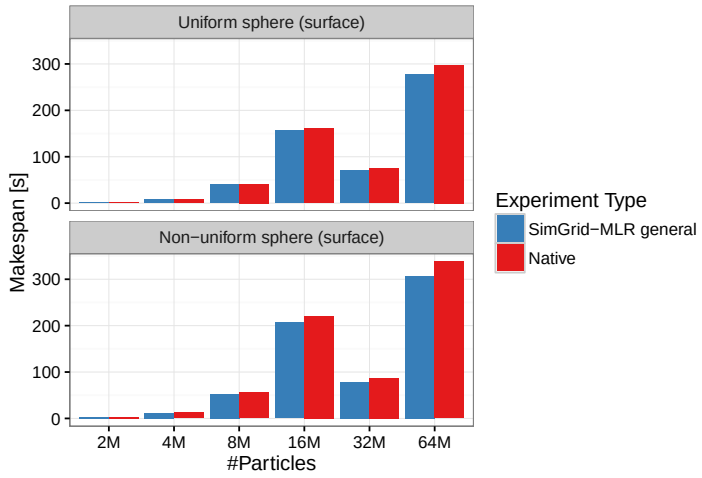


Figure 11: Simulation predictions for the uniform and non-uniform sphere use cases with different configuration options (particles, level and granularity).

hand, MLR models ensure that not only the overall kernel durations are correct, but also that each kernel occurrence is well predicted. Hence, scheduling of such tasks in simulation is fully representative of what is happening in Native execution. In Figure10, for most of the kernels MLR distributions are overlapped with the Native execution, which shows the good predictive power of these models.

5.4 Applying general MLR models for different inputs

Unlike all other presented models, MLR general models can be used not only for simulating the same input configurations, but on a much broader range of settings. To demonstrate this, we have performed 12 new simulations with completely new inputs, changing the distribution of the particles to sphere distribution and using different number of particles, tree height and granularity. Only later, to inspect the accuracy of our predictions, we run Native experiments with the same input configurations and compared the makespans. Results presented in Figure11 suggest that the predictions stay fairly close to the real machine performance, even if the models used for simulations were derived from different use cases.

This scenario clearly illustrates the benefits of using a simulation with MLR general model which can quickly provide faithful performance predictions for new inputs. Indeed, once the models have been obtained and the target machine has been calibrated, there is no more need for accessing the experimental platform, as all the simulations can be run on a local commodity machine. Moreover, StarPU-SimGrid can perform such simulations much faster than the real execution, with a lower memory footprint and on a single CPU core. This permits

much larger experimental campaigns, focused on studying different application parameters and optimal configurations (e.g., ideal tree height and granularity for a given particle number and distribution). Using such tool is advantageous for application users and developers, as it reduces the need for accessing large computing clusters. However, this approach has some limitations and the good model accuracy comes only with certain hypothesis.

5.5 Limitations of the approach

The modeling approach described in Section4 provided very accurate kernel duration estimations for extremely complex ScalFMM kernels. Nevertheless, for some of these kernels, such as M2L_in, indirect parameters were needed. Unfortunately, their computation comes with a certain overhead and although it was not the case for ScalFMM, for some other applications this additional computations can perturb the initial Native execution.

Moreover, model accuracy and generality, i.e., applicability to different input configurations, greatly depend on the way coefficients are calibrated. Identifying the appropriate design of experiments needed for obtaining the best models is not trivial. Even for researchers with a good knowledge of the kernels they use or even sometimes develop themselves, this process often requires a lot of empirical tests. Therefore, MLR models presented in this paper should be used with caution and deep understanding of their limitations.

6 Conclusion and future work

In this paper, we described a methodology for accurately modeling durations of irregular kernels using multiple linear regression. We detailed all the necessary steps, from parameterisation and benchmarking of the kernels, through their modeling and validation, to the final coefficient calibrations and utilization. The proposed workflow is simple to use for application and runtime system developers who seek to determine the models, while it is completely automatic and transparent for application users who can benefit from model estimations. This allows for researchers without a deep knowledge in statistical tools to easily obtain accurate kernel duration estimations and exploit them to perform various studies.

We evaluated our approach with a complex task-based application (ScalFMM), implemented on top of the StarPU runtime system. All 8 extremely irregular ScalFMM kernels have been modeled and later successfully used for StarPU-SimGrid performance simulation. Accurate predictions of both the kernel durations and the whole execution suggest that this approach can be very useful. Moreover, the same kernel models can be exploited for predicting application performance on large scale distributed machines, since the task executions stay unchanged. We intend to perform such a study using the recent MPI implementation of task-based ScalFMM [3] and investigate simulations for very big scenarios.

The evaluation presented in this paper focused on the simulation predictions of the application performance. However, this is not the only use case in which kernel estimations can be beneficial. Indeed, these models can be used as additional scheduling heuristics to improve load balancing of the task-based application and we intend to explore this path. We furthermore plan to assess the proposed methodology on task-based OpenMP codes. Regarding hardware architectures, we also aim to study the modeling of task-based codes on heterogeneous machines.

Acknowledgments

We thank Olivier Aumage for his invaluable advice on StarPU and writing of this paper, as well as Samuel Pitoiset for sharing his expertise on task-based ScalFMM configuration and experiment tracing. We also thank Guillaume Sylvand for his suggestions on improving the structure of this paper. Experiments presented in this paper were carried out using the PlaFRIM experimental testbed, being developed under the Inria PlaFRIM development action with support from LABRI and IMB and other entities: Conseil Regional d'Aquitaine, FeDER, Universite de Bordeaux and CNRS (see https://plafrim.bordeaux.inria.fr/).

A Appendix

A.1 Direct parameters limit for estimating the FMM operators

We describe briefly the difficulties of estimating the workload for a task using the direct access parameters.

A P2M task transfers the physical values of the particles to the Multipole value in the leaves. Such task takes three parameters: the symbolic block of the particles group, which includes the physical values, and the symbolic and Multipole blocks of the target cell group. For our FMM kernel, the amount of work is proportional to the number of particles and given by $\zeta \times N \times O$, with ζ a constant, N the number of particles in the group, O the number of Multipole terms in the leaf. However, the distribution of the particles impacts the memory access and the data reuse make the usage of this simple formula potentially inaccurate. If there is one particle per leaf, we need to access N different Multipole arrays, whereas, if the particles are in the same leaf, we reuse N times the same array.

The M2M operator transfers the Multipole values from children to parents. In our case, we transfer a group from level l to a parent group at level $l-1$. This task requires four blocks which are the symbolic and Multipole data for both groups. We can provide a workload bound from the extremities of the Morton interval index of both groups. There are a maximum of $P = Min(parents.g_{end}, parent(children.g_{end})) - Max(parents.g_{start}, parent(children.g_{start}))$ parent cells involved in the M2M, where *parents* and *children* are the groups and $parent(i)$ a function which returns the index of the parent of the cell of index i. Therefore, we can bound the number of M2M interactions to $8 \times P$. We can use a different estimation by considering that the cells are uniformly distributed in space. In this case, there should be around $P/(parents.g_{end} - parents.g_{start}) \times children.nb\_cells$ children involved. Even so such coefficient could be true in some configurations, in non uniform distribution it will not be the case.

The operations in the tree are constant regarding the kernel complexity: no mater the level or the relative position of the cells, any single P2M/M2M/M2L/L2L/L2P has a known cost. But this is not true for the P2P which makes it more complicated to predict. The P2P operator (i.e., near field operator) involves the neighbors of the leaves as it is for the M2L its interaction list. Its complexity is quadratic with respects to the number of particles because we compute all pair-wise interactions between two leaves. Therefore, estimating the cost of the P2P inside a grouped or between two groups is difficult because we need to have the number of neighbors per leaf and the number of particles in each of them. These parameters cannot be easily found from the group's properties and thus non-uniform particles distribution are then difficult to predict.

A.2 Full limitations of the approach

Indirect parameters challenges When the kernel model is not accurately described using solely directly accessible parameters, this can often be improved by adding indirect ones (see Section4.2). However, this introduces an overhead to the application, since the additional computations are needed before the kernel is submitted to the runtime system. If the thread responsible for inserting all these kernels is the performance bottleneck, this can even slow down the whole application execution. Therefore, the influence of adding indirect parameter computations must be carefully monitored.

Indirect parameters often provide good estimators, still there are some specific kernels whose code is too complex to be fully described using any of these parameters. In such cases, the best one can do is to provide some rough approximations based on kernel internals. However, this may also be a sign for application developers to reconsider their kernel implementation, as it is hard for runtime scheduler to handle such a complexity and variability and thus obtaining maximal performance might be extremely challenging.

Influence of design of experiments A key difficulty of our approach relies in obtaining a general model for every kernel that can provide accurate predictions for any value of input parameters. However, due to the high complexity of modern hardware and software stack, it is extremely hard to fully understand kernels behavior. Therefore, the model faithfulness and its applicability domain will greatly depend on the DoE and the initial observations chosen for acquiring the model. In the ScalFMM modelization case, each particles distribution with different granularity has a unique structure and the task graph generated to resolve the problem on a parallel machine also depends on these characteristics. Therefore, the parameters of the kernel greatly vary from one input configurations to another. For example, the cube distribution with a small granularity will have a large number of M2L_in kernels, while the ellipsoid distribution will have a much smaller number of these kernels, but possibly with a longer duration. Consequently, it is very hard to construct a single linear model that is appropriate for both use cases. The inaccuracies caused by such a model imperfection can produce either underestimation or overestimation of the kernel duration.

The simplest solution for this problem would be to use piecewise linear models instead of simple linear ones. This should account for the particular parameter distribution and group the related observations. Constructing such models in practice however proved to be very challenging, as the observations are not following any natural law that divides them into groups and they are also greatly affected by external factors (other threads sharing the CPU cache, operating system noise, etc.). Thus, choosing where to put breakpoints to separate different segments is unclear. We could decide to completely rely on statistical tools that would help us find the best fitting piecewise linear model from the traces. However, such a model, and especially the choosen breakpoints, would be very dependent on the measured values and thus not robust enough for a

more general use. The decision where the breakpoints are made would have a huge influence on the overall model accuracy.

Another important modelization challenge lies in dealing with the outliers. This is a particularly important issue for the ScalFMM kernels P2P_in, P2M, L2P and M2L_in whose few last kernel instances are often operating on the smaller chunks of data and thus have much lower duration. In our current solution, we decided to ignore the existence of such outliers, which has a minor negative influence on the overall accuracy of our models. This can be improved either through the usage of the aforementioned piece-wise linear models or by a finding the additional parameters which can describe such a different kernel behavior.

Model (in)accuracy for sub-optimal executions Computation kernels are highly optimized for the underlying hardware and software. Developers design kernels with a certain typical input configurations in mind. Hence, executing these kernels with a poor choice of input values might not only have a bad performance, but could also exhibit a totally different behavior. Consequently the choice of parameters and parameter combinations for such sub-optimal executions might be very different than for the well-optimized kernels. Therefore, when one targets to obtain accurate general models, the choice of input configuration of the application, and thus corresponding input values for the kernels, needs to be carefully managed.

Even larger problems may occur if the traces used for modelization (phase 2 in Figure4) are bogus, due to some unexpected hardware or software malfunctioning of the target platform. Indeed, during our study we encountered such an issue, where a group of experiments contained M2M kernels with an overlong duration. The corresponding model generated from these experiments was fitting well the observations and had high adjusted R^2 value. However, such a model was not accurately representing the real kernel behavior in regular circumstances. Consequently, future SimGrid predictions based on these models were largely overestimating new Native executions and it took us a long time to detect the source of the error. We argue that such problems are common in our field, as many researchers are working with prototype hardware and software whose performance is unstable. Therefore, benchmarked observations used for modeling should first be carefully analyzed and verified that they do not contain certain surprising phenomena.

References

[1] Emmanuel Agullo, Olivier Aumage, Berenger Bramas, Olivier Coulaud, and Samuel Pitoiset. Bridging the gap between OpenMP 4.0 and native runtime systems for the fast multipole method. Research Report RR-8953, Inria, March 2016.

[2] Emmanuel Agullo, Berenger Bramas, Olivier Coulaud, Eric Darve, Matthias Messner, and Toru Takahashi. Task-based FMM for heterogeneous architectures. *Concurrency and Computation: Practice and Experience*, 28(9):2608–2629, 2016.

[3] Emmanuel Agullo, Bérenger Bramas, Olivier Coulaud, Martin Khannouz, and Luka Stanisic. Task-based fast multipole method for clusters of multicore processors. Research Report RR-8970, Inria Bordeaux Sud-Ouest, October 2016.

[4] Emmanuel Agullo, Brenger Bramas, Olivier Coulaud, Eric Darve, Matthias Messner, and Toru Takahashi. Task-based fmm for multicore architectures. *SIAM Journal on Scientific Computing*, 36(1):C66–C93, 2014.

[5] Emmanuel Agullo, Alfredo Buttari, Abdou Guermouche, and Florent Lopez. Implementing multifrontal sparse solvers for multicore architectures with sequential task flow runtime systems. Technical Report IRI/RT–2014-03–FR, IRIT, 2014. Accepted in ACM TOMS.

[6] Cedric Augonnet, Samuel Thibault, Raymond Namyst, and Pierre-André Wacrenier. StarPU: A unified platform for task scheduling on heterogeneous multicore architectures. *Concurrency and Computation: Practice and Experience*, 23, February 2011.

[7] Lorena A Barba and Rio Yokota. How will the fast multipole method fare in the exascale era. *SIAM News*, 46(6):1–3, 2013.

[8] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. The Gem5 Simulator. *SIGARCH Comput. Archit. News*, 39(2):1–7, August 2011.

[9] Berenger Bramas. *Optimization and parallelization of the boundary element method for the wave equation in time domain.* Theses, Université de Bordeaux, February 2016.

[10] Alfredo Buttari. Fine-grained multithreading for the multifrontal QR factorization of sparse matrices. *SIAM Journal on Scientific Computing*, 35(4):C323–C345, 2013.

[11] Alfredo Buttari, Julien Langou, Jakub Kurzak, and Jack Dongarra. A class of parallel tiled linear algebra algorithms for multicore architectures. *Parallel Comput.*, 35:38–53, January 2009.

[12] Henri Casanova, Arnaud Giersch, Arnaud Legrand, Martin Quinson, and Frédéric Suter. Versatile, scalable, and accurate simulation of distributed applications and platforms. *Journal of Parallel and Distributed Computing*, 74(10), June 2014.

[13] Jee Choi, Aparna Chandramowlishwaran, Kamesh Madduri, and Richard Vuduc. A cpu: Gpu hybrid implementation and model-driven scheduling of the fast multipole method. In *Proceedings of Workshop on General Purpose Processing Using GPUs*, GPGPU-7, pages 64:64–64:71, New York, NY, USA, 2014. ACM.

[14] Pietro Cicotti, Xiaoye S. Li, and Scott B. Baden. Performance Modeling Tools for Parallel Sparse Linear Algebra Computations. In *Parallel Computing: From Multicores and GPU's to Petascale, Proceedings of the conference ParCo 2009, 1-4 September 2009, Lyon, France*, pages 83–90, 2009.

[15] Alejandro Duran, Eduard Ayguadé, Rosa M. Badia, Jesús Labarta, Luis Martinell, Xavier Martorell, and Judit Planas. Ompss: a Proposal for Programming Heterogeneous Multi-Core Architectures. *Parallel Processing Letters*, 21(2), 2011.

[16] L. Greengard and V. Rokhlin. A fast algorithm for particle simulations. *J. Comput. Phys.*, 73(2):325–348, December 1987.

[17] William Gropp, Ewing Lusk, and Anthony Skjellum. *Using MPI: Portable Parallel Programming with the Message Passing Interface*. Scientific And Engineering Computation Series. MIT Press, 2nd edition, 1999.

[18] B. Haugen, J. Kurzak, A. YarKhan, P. Luszczek, and J. Dongarra. Parallel Simulation of Superscalar Scheduling. In *Parallel Processing (ICPP), 2014 43rd International Conference on*, pages 121–130, Sept 2014.

[19] Xiaoye S. Li and James W. Demmel. Superlu_dist: A scalable distributed-memory sparse direct solver for unsymmetric linear systems. *ACM Trans. Math. Softw.*, 29(2):110–140, June 2003.

[20] Jack L. Lo, Susan J. Eggers, Joel S. Emer, Henry M. Levy, Rebecca L. Stamm, and Dean M. Tullsen. Converting thread-level parallelism to instruction-level parallelism via simultaneous multithreading. *ACM Transactions on Computer Systems*, 15:322–354, 1997.

[21] Hatem Ltaief and Rio Yokota. Data-Driven Execution of Fast Multipole Methods. *CoRR*, abs/1203.0889, 2012.

[22] Todd Mytkowicz, Amer Diwan, Matthias Hauswirth, and Peter F. Sweeney. Producing Wrong Data Without Doing Anything Obviously Wrong! In *Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XIV, pages 265–276. ACM, 2009.

[23] Avadh Patel, Furat Afram, Shunfei Chen, and Kanad Ghose. MARSS: A Full System Simulator for Multicore x86 CPUs. In *Proceedings of the 48th Design Automation Conference*, DAC '11, pages 1050–1055, New York, NY, USA, 2011. ACM.

[24] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2016.

[25] Jose Renau, Basilio Fraguela, James Tuck, Wei Liu, Milos Prvulovic, Luis Ceze, Smruti Sarangi, Paul Sack, Karin Strauss, and Pablo Montesinos. SESC simulator, January 2005.

[26] Alejandro Rico, Alejandro Duran, Felipe Cabarcas, Yoav Etsion, Alex Ramírez, and Mateo Valero. Trace-driven simulation of multithreaded applications. In *IEEE International Symposium on Performance Analysis of Systems and Software, ISPASS 2011, 10-12 April, 2011, Austin, TX, USA*, pages 87–96, 2011.

[27] A. F. Rodrigues, K. S. Hemmert, B. W. Barrett, C. Kersey, R. Oldfield, M. Weston, R. Risen, J. Cook, P. Rosenfeld, E. CooperBalls, and B. Jacob. The Structural Simulation Toolkit. *SIGMETRICS Perform. Eval. Rev.*, 38(4):37–42, March 2011.

[28] Allan Snavely, Laura Carrington, Nicole Wolter, Jesus Labarta, Rosa Badia, and Avi Purkayastha. A framework for performance modeling and prediction. In *Proceedings of the 2002 ACM/IEEE conference on Supercomputing*, Supercomputing '02, pages 1–17, Los Alamitos, CA, USA, 2002. IEEE Computer Society Press.

[29] Luka Stanisic, Emmanuel Agullo, Alfredo Buttari, Abdou Guermouche, Arnaud Legrand, Florent Lopez, and Brice Videau. Fast and Accurate Simulation of Multithreaded Sparse Linear Algebra Solvers. In *The 21st IEEE International Conference on Parallel and Distributed Systems*, The 21st IEEE International Conference on Parallel and Distributed Systems, Melbourne, Australia, December 2015.

[30] Luka Stanisic, Samuel Thibault, Arnaud Legrand, Brice Videau, and Jean-François Méhaut. Faithful Performance Prediction of a Dynamic Task-Based Runtime System for Heterogeneous Multi-Core Architectures. *Concurrency and Computation: Practice and Experience*, page 16, May 2015.

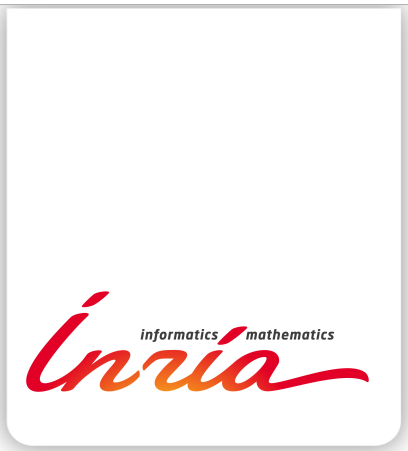
[31] Francis Sullivan and Jack Dongarra. Guest editors' introduction: The top 10 algorithms. *Computing in Science & Engineering*, 2(1):22–23, 2000.

[32] Haluk Topcuouglu, Salim Hariri, and Min-you Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Trans. Parallel Distrib. Syst.*, 13(3):260–274, March 2002.

[33] V.M. Weaver and S.A. McKee. Are Cycle Accurate Simulations a Waste of Time? In *Proc. of the 7th Workshop on Duplicating, Deconstruction and Debunking*, Beijing, China, June 2008.

[34] A. YarKhan, J. Kurzak, and J. Dongarra. QUARK Users' Guide: QUeueing And Runtime for Kernels, 2011.

[35] Gengbin Zheng, Gunavardhan Kakulapati, and Laxmikant Kalé. BigSim: A Parallel Simulator for Performance Prediction of Extremely Large Parallel Machines. In *Proc. of the 18th International Parallel and Distributed Processing Symposium (IPDPS)*, April 2004.



RESEARCH CENTRE
BORDEAUX – SUD-OUEST

200 avenue de la Vieille Tour
33405 Talence Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399