Building a Fine-Grained Analytical Performance Model for Complex Scientific Simulations

Jelle van Dijk[0000−0003−3005−9890], Gabor Zavodszky[0000−0003−0150−0229], Ana-Lucia Varbanescu[0000−0002−4932−1900], Andy D. Pimentel[0000−0002−2043−4469], and Alfons Hoekstra[0000−0002−3955−2449]

Institute for Informatics, Faculty of Science, University of Amsterdam, Amsterdam,The Netherlands

Abstract. Analytical performance models are powerful for understanding and predicting the performance of large-scale simulations. As such, they can help identify performance bottlenecks, assess the effect of load imbalance, or indicate performance behavior expectations when migrating to larger systems. Existing automated methods either focus on broad metrics and/or problems - e.g., application scalability behavior on large scale systems and inputs - or use black-box models that are more difficult to interpret e.g., machine-learning models.
In this work we propose a methodology for building per-process analytical performance models relying on code analysis to derive a simple, high-level symbolic application model, and using empirical data to further calibrate and validate the model for accurate predictions.
We demonstrate our model-building methodology on HemoCell, a high-performance framework for cell-based bloodflow simulations. We calibrate the model for two large-scale systems, with different arichtectures. Our results show good prediction accuracy for four different scenarios, including load-balanced configurations (average error of 3.6%, and a maximum error below 13%), and load-imbalanced ones (with an average prediction error of 10% and a maximum error below 16%).

Keywords: Performance modeling, workload imbalance, performance prediction, coupled simulations

1 Introduction

Analytical performance models are powerful for understanding and predicting the performance of large-scale simulations. An *analytical performance model* is a closed-form expression that describes application performance, expressed in a metric of choice, as a combination of *application components*, *application specific parameters* and *hardware parameters*. Analytical models are human-readable and cost little to no resources to use. Furthermore, they can provide many insights that are otherwise expensive to obtain, e.g., locating the performance bottlenecks [10], or are not obtainable at all, e.g., predicting how an application will perform on a next generation of supercomputers [7].

Analytical performance models are also useful in determining the impact of load-imbalance in large-scale parallel applications running on current parallel (distributed) systems. Load-imbalance occurs when the parallel processes of an application are not assigned an equal amount of work. This can cause significant inefficiency during the parallel execution of applications on large-scale systems. Several large-scale applications and libraries [5, 9, 15, 21] already use different simple analytical performance models to predict load imbalance. Some of these analytical models have a per-process view of the application, thus allowing for detailed load imbalance predictions[9].

Despite its advantages, analytical modeling remains challenging, as it requires both performance-modeling and application-specific expertise. Furthermore, because the resulting models are application-specific, most of the work needs to be redone when modeling a different application, or even the next version of the same application. While work on generalizing the *process* of building analytical performance models already exists [10, 14], most of these approaches aim to provide performance models that predict scalability and extrapolate to new, larger systems, and/or lack the fine granularity needed to support a better understanding of application inefficiency.

To address such limitations, we propose in this work a detailed methodology for building fine-grained, per-process analytical performance models for scientific simulations. By design, the per-process modeling gives us more detailed insights into the performance and load-balance of the modeled application. Our four-step modeling process is as follows: (1) we identify the code-sections and input parameters with a relevant performance impact (2) we build a symbolic analytical performance model that describes the application performance at process level, (3) we calibrate the models for a specific machine with the help of empirical performance data, and we aggregate the per-process models for an application-wide performance prediction. (4) We validate the model performance. The resulting model is outlined in Figure 1. We note that the symbolic model remains constant for the target application - that is, migrating the model to a different system only requires re-calibration, i.e., collecting empirical performance data.

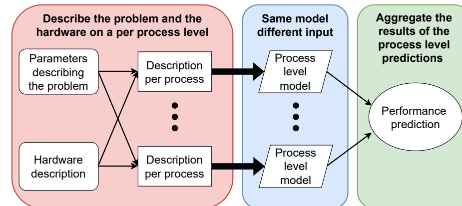


Fig. 1: Overview of the per-process analytical model.

We demonstrate the feasibility of our approach on HemoCell, a high-performance framework for dense cellular suspension flows [19, 20]. Previous work, starting from L. Axner, et. al. [2], developed a model to predict runtime performance from fractional overheads and showed that, when accurate estimation of these overheads is available, the model is an accurate tool (at most 5% error) for analyzing code execution, even in load-imbalanced scenarios. S. Allowayyad, et. al.

[1] applied this model for HemoCell, where the fractional overhead caused by load imbalance was estimated under the assumption that it is entirely dependent on local red blood cell count. In this work, we propose a novel methodology to build a model for the function level performance, which, after calibration, provides *an estimation* for the main sources of computational load (i.e., for the fractional loads). This method is demonstrated using HemoCell, similarly to [1]. However, the calibrated performance functions are defined using natural units of the simulation (red blood cell count and fluid node count). Specifically, we build a per-process function-level symbolic model for HemoCell, and calibrate it for two different HPC platforms: Snellius (SURF, Netherlands) and DAS6 (ASCI, Netherlands) [3]. The model accuracy is evaluated on balanced and non-balanced simulations, using three scenarios that showcase different types of execution imbalance[1]. Our results demonstrate good *prediction accuracy* for our models, indicating they can be useful tools for assessing load-imbalance impact in scientific simulations.

The remainder of this paper is structured as follows. We present our modeling approach in Section 2 and further show, in Section 3, how it is applied to build and calibrate an analytical performance model for HemoCell. We further evaluate the accuracy of the model in Section 4 on four different scenarios with different degrees and types of load-imbalance. Finally, we provide a brief overview of related work in Section 5, and conclude the paper in Section 6.

2 Performance Modeling Methodology

In this section we present our methodology for building per-process analytical performance models for large-scale simulations. We assume a Single-Program, Multiple-Data model, where processes with different ranks and are executed concurrently on different processing units (e.g., cores or nodes). Throughout the modeling process we also assume that at least function level performance measurements of the application are available. The collected data depends on the desired model output, e.g., time (s), execution rate (Mflop/s), or energy (J).

Our methodology has four steps: (1) identifying relevant code sections and parameters, (2) building the model, (3) calibrating the model, and (4) validating the model. In this section we elaborate on each of these steps.

(1) Identify performance relevant code sections and parameters. A code section can be any part of the application code which is monitored individually. Usually, in practice, such code sections are *functions*. The *relevant code sections* are those code sections that are significant in the performance breakdown. The performance of a code section will change based on external parameters, e.g., size of the simulated domain. For each code section, we identify the relevant parameters, which are then selected as inputs for the model.

(2) Build the symbolic model: The model is built in a top-down manner: we start from a coarse symbolic model, and refine parts as needed, which allows

[1] The code for data processing and the raw data used in this paper are available at DOI:10.5281/zenodo.6570501.

for control over the level of detail incorporated into the model. The results, is a symbolic analytical performance model that describes the performance of a singe process in terms of the code-sections and parameters selected in step (1).

The output of the per-process model is aggregated into a final prediction using operators that are application- and metric-specific (see Figure 1). For example, when predicting execution time for fully concurrent applications, the performance is dominated by the longest process; however, when processes run sequentially, the aggregated execution time is the sum of the execution time of all processes.

(3) Calibrate the model: To calibrate the model we replace the symbolic terms describing code section performance with predictive functions. Firstly, empirical data of code section performance is collected. This data is used to fit a function for each individual code section, the degree of the function depends on the relationship between the code section and the input parameters e.g., the output can scale linearly or exponentially in relation to the parameters.

(4) Validate the model: To validate the model, we measure performance on relevant (unseen) datasets, and report prediction error, calculated as $e = \text{abs}(\text{predicted} - \text{measured}) * 100/\text{measured}$. If needed, to increase prediction accuracy, the model can be further refined (i.e., functions can be further split into smaller units). This, however, also increases the model complexity.

3 Modeling Hemocell

In this section, we build an analytical performance model describing the execution time of Hemocell. This model is calibrated on two different machines, Snellius (SURF, Netherlands) and DAS6 (ASCI, Netherlands) [3].

3.1 Hemocell

Hemocell is a coupled multi-scale simulation code used for modeling blood flow. The application simulates blood as a dense cellular suspension flow, modeling the evolution of particles, i.e., red blood cells (RBCs) and platelets, suspended in a solvent, i.e., the blood plasma, over multiple discrete time steps [19, 20]. The solvent is modeled as a fluid using the lattice Boltzman method (LBM). LBM calculations are handled by the Palabos library [12]. The movement, deformation, and interaction of particles is modeld separately from the LBM calculation. Both models are coupled together intermittently to simulate the full blood flow system.

For parallelization, Hemocell uses multi-processing: each process receives a section of the *simulated domain*, i.e., a *subdomain*, and is responsible for computing the fluid and particles within that subdomain. During the simulation, the processes communicate with each other using MPI. The edges of the fluid field, as well as parts of the particles which may span multiple subdomains, must be communicated to ensure correct results.

Previous research focused on improving Hemocell's overall performance [16], as well as improving the scaling performance through better load balancing [1].

3.2 Performance-relevant Functions and Parameters

A Hemocell simulation consists of three phases: (1) setup, (2) computation, and (3) data output. Our work focuses on the most expensive of these phases, the computation. In turn, the Hemocell computation phase has three components: (i) fluid computation, (ii) particle computation, and (iii) model coupling.

We define *performance-relevant functions* as those functions that have a non-negligible performance impact. Similarly, we define *performance-relevant parameters* as function parameters that have a non-negligible performance impact. The process of identifying the performance-relevant functions and parameters is based on both expert application knowledge, code inspection, and investigation of any available fine-grained performance measurements. Table 1 shows the performance-relevant functions and parameters for Hemocell.

Table 1: Performance-relevant functions and parameters for Hemocell.

Name	Component	Description	Parameters
CollideAndStream	Fluid field	Lattice-Boltzmann calculations.	(xs, ys, zs)
CollideAndStream_comm	Fluid field	Lattice-Boltzmann communication.	(xs, ys, zs)
spreadParticleForce	Model coupling	Apply particle forces to the fluid field.	RBCs
interpolateFluidVelocity	Model coupling	Apply fluid forces to the particles.	RBCs
syncEnvelopes	Particle field	Setup for particle communication.	RBCs
syncEnvelopes_comm	Particle field	Communicate particle vertices.	RBCs, (xs, ys, zs)
AdvanceParticles	Particle field	Calculate new particle position.	RBCs
applyConstitutiveModel	Particle field	Compute and apply internal particle forces.	RBCs
deleteNonLocalParticles	Particle field	Remove non-local particle information.	RBCs
setExternalVector	Fluid field	Apply external forces to the fluid.	(xs, ys, zs)

3.3 Model-Building

For building the model, we start with the highest-level description of the application: the components.

$$\begin{aligned} T = \text{Iters} \times\ [\ &\text{FluidField}(\text{xs}, \text{ys}, \text{zs}) \\ &+ \text{ParticleField}(\text{xs}, \text{ys}, \text{zs}, \text{RBCs}) \\ &+ \text{ModelCoupling}(\text{RBCs})\ ] \end{aligned} \tag{1}$$

In Equation (1), *Iters* is the number of iterations, (xs, ys, zs) are the dimenions of the domain, RBCs is the number of red blood cells within the domain, and *FluidField*, *ParticleField* and *ModelCoupling* are the functions that describe the execution time per iteration for each component[2]. We improve on this initial model by expanding the component terms. Each component term is made up of the summation of the time spent in the respective relevant functions, see Table 1.

To simplify the calibration step we derive two new parameters: V and SA, representing the subdomain volume and surface area, respectively. For rectangular domains they are defined as $V = xs \times xy \times xz$ and $SA = 2 \times (xs \times ys + xs \times zs + ys \times zs)$. Expanding on the initial model, replacing *xs, ys, zs* with either V

[2] Please note: for readability purposes, when using the name of a function in a model, we denote its performance, in most cases, execution time. In other words, we use `ParticleField` instead of $T_{ParticleField}$.

or SA, gives us the following analytical model:

$$\begin{aligned} T = \text{Iters} \times \; [\; & \text{FluidField(V, SA)} \\ & + \text{ParticleField(SA, RBCs)} \\ & + \text{ModelCoupling(RBCs)} \;] \end{aligned} \tag{2}$$

$$\begin{aligned} \text{FluidField}(V, SA) \; = \; & \text{CollideAndStream}(V) \\ & + \text{CollideAndStream_comm}(SA) \\ & + \text{setExternalVector}(V) \end{aligned} \tag{3}$$

$$\begin{aligned} \text{ParticleField}(SA, RBCs) = \; & \text{syncEnvelopes}(RBCs) \\ & + \text{syncEnvelopes_comm}(RBCs, SA) \\ & + \text{AdvanceParticles}(RBCs) \\ & + \text{applyConstitutiveModel}(RBCs) \\ & + \text{deleteNonLocalParticle}(RBCs) \end{aligned} \tag{4}$$

$$\begin{aligned} \text{ModelCoupling}(RBCs) \; = \; & \text{spreadParticleForce}(RBCs) \\ & + \text{interpolateFluidVelocity}(RBCs) \end{aligned} \tag{5}$$

3.4 Model Calibration

In the calibration step the terms in the model are replaced with *predictors*. The predictors are fitted, using empirical data collected from the two machines, Snellius and DAS6, the machine details are presented in Table 2.

Table 2: Machine Descriptions

Machine	CPU	Cores	Frequency	Memory
Snellius	AMD Rome 7H12 (x2)	128	3.2 GHz	256 GiB
DAS6	AMD EPYC-2 7402P	24	2.8 GHz	128 GB

To collect the data, we simulate a cuboid-shaped domain of blood for 500 iterations. The size of the domain ranges from $(12.5, 12.5, 12.5)$µm to $(75, 75, 50)$µm. Every domain size is run with 7 differnt volume fractions of RBCs (hematocrit): 0%, 9%, 10%, 12%, 14%, 16%, and 18%. The workload in these experiments is fully balanced, i.e., every process performs the same amount of work. Throughout the modeling and analysis process, we use the Scalasca toolchain for automatic code instrumentation and performance measurements of Hemocell [8, 11].

The predictors are all fitted functions over the respective performance data.The degree of the fit function is dependent on the relationship between the parameters and the output metric. For this model we have chosen the parameters such that all relationships are linear. The calibrated predictors are presented in Table 3.

4 Scenario Analysis

The fine granularity of the model allows for accurate performance predictions in scenarios that differ from the configuration used for calibration. In this section we evaluate the model, as presented in Section 3, and use it to analyze the

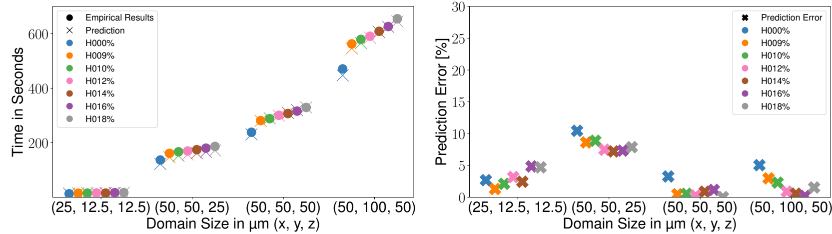
Table 3: Calibrated performance predictors for Snellius and DAS6.

Name	Predictors Snellius [S]	Predictors DAS6 [S]
collideAndStream	$0.0062 + V \times 3.5 \times 10^{-7}$	$0.008 + V \times 2.5 \times 10^{-7}$
setExternalVector	$2.6 \times 10^{-5} + V \times 4.3 \times 10^{-8}$	$-0.000\,22 + V \times 2.1 \times 10^{-8}$
collideAndStream_comm	$-0.000\,47 + SA \times 9.1 \times 10^{-7}$	$0.000\,94 + SA \times 2.2 \times 10^{-7}$
syncEnvelopes_comm	$0.000\,48 + SA \times 1.3 \times 10^{-7}$ $+RBCs \times 3.5 \times 10^{-5}$	$0.000\,46 + SA \times 1.3 \times 10^{-8}$ $+RBCs \times 9.5 \times 10^{-6}$
syncEnvelopes	$-1.4 \times 10^{-5} + RBCs \times 8.3 \times 10^{-5}$	$9.2 \times 10^{-5} + RBCs \times 3.6 \times 10^{-5}$
advanceParticles	$0.000\,49 + RBCs \times 0.000\,14$	$0.000\,59 + RBCs \times 8.2 \times 10^{-5}$
applyConstitutiveModel	$-1.3 \times 10^{-5} + RBCs \times 4.4 \times 10^{-5}$	$-4.4 \times 10^{-5} + RBCs \times 2.8 \times 10^{-5}$
deleteNonLocalParticles	$5.1 \times 10^{-5} + RBCs \times 1.5 \times 10^{-5}$	$2.1 \times 10^{-5} + RBCs \times 7.5 \times 10^{-6}$
spreadParticleForce	$0.000\,81 + RBCs \times 0.0004$	$0.0012 + RBCs \times 0.000\,25$
interpolateFluidVelocity	$0.000\,13 + RBCs \times 7.7 \times 10^{-5}$	$0.000\,31 + RBCs \times 4.2 \times 10^{-5}$

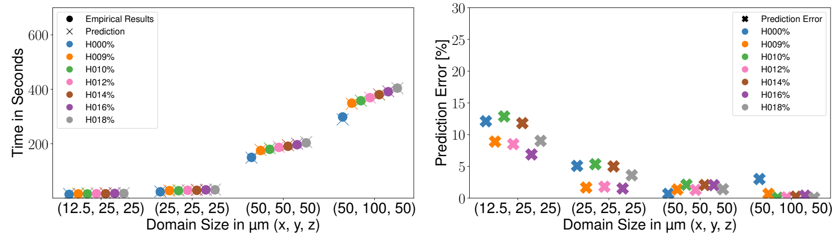
performance of Hemocell in four different scenarios, (1) balanced workload, (2) imbalanced subdomains, (3) imbalanced hematocrit, and (4) imbalanced communication.

4.1 Scenario: Balanced Workload

In the balanced scenario each process receives the same amount of work. The setup is identical to the simulation configurations used for model calibration, however the domain sizes are of course different. Empirical and predicted results are shown for Snellius and DAS6 in Figures 2a and 2b.The results show that the model can accurately predict the performance in this scenario a maximum error of 12.87% and an average error of 3.6%.



(a) Snellius (128 processes)



(b) DAS6 (24 processes)

Fig. 2: Observed and predicted execution time and prediction error for the load balanced scenario, on DAS6 and Snellius. The standard deviation of the observed results is within 1.5%

4.2 Scenario: Imbalanced Subdomains

In an ideal scenario each process is assigned a subdomain of the same size. However, due to complex simulation domains this is not always achievable. An imbalanced distribution of the domain leads to a loss of performance.

The imbalance in this scenario is generated by assigning half of the processes to 75% of the full domain, and the other half of the process to 25% of the full domain, see Figure 3. This means that half of the process are assigned three times more work than the other half.

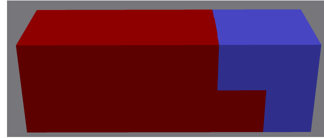


Fig. 3: Imbalanced domain distribution. Both the red and blue parts are assigned to half of the processes.

For each configuration, the results for the imbalanced and balanced configurations are measured and predicted. By comparing the balanced and imbalanced predictions we estimate the overhead introduced by the load imbalance. The results are presented in Figure 4

We observe good accuracy for the imbalanced scenario predictions, with a maximum error of 15.83% and an average of 10.26%. The prediction accuracy on Snellius is lower than on DAS6. This is most likely caused by the difference in memory layout. A node on Snellius is dual-socket, meaning that the L3 cache is not shared between all threads. By assigning most of the domain to half of the processes we are moving most of the data onto a single socket, significantly increasing the amount of data that is accessed by that socket. This combined with the larger overall domain on Snellius, which results in the subdomains of one process being further apart. The result is that the cost of memory operations increases more on Snellius, which is not captured by the model.

We also observe that the load-imbalance overhead is higher for the 18% hematocrit configurations, compared to the equivalent 0% runs. This increase is due to the RBC computation, which worsens the already-present workload imbalance. This increase is correctly captured by the model.

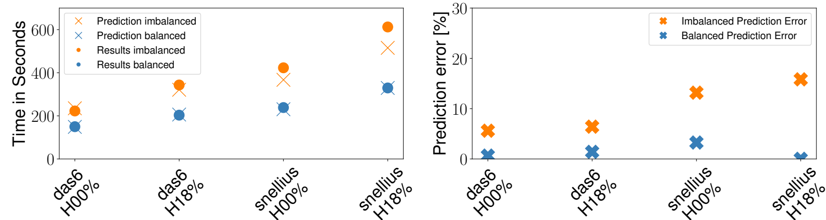
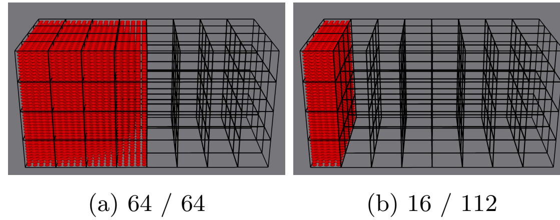


Fig. 4: Observed and predicted execution time and prediction error for the imbalanced domain scenario, on DAS6 and Snellius. The standard deviation of the observed results is within 1.5%

4.3 Scenarios: Imbalanced Hematocrit

The hematocrit value has a significant impact on performance, as can be seen in Figure 2. A higher hematocrit means more of the volume is occupied by RBCs, resulting in more computation and communication. In the configurations up till now, we assumed a homogeneous hematocrit throughout the domain. However, in more realistic scenarios the hematocrit varies throughout the domain.

The imbalanced hematocrit scenario shows the performance overhead of having a non-homogeneous hematocrit. To create an imbalanced hematocrit each domain is initialized such that part of the domain has a hematocrit of 18%, the other part is either initialized at 9% or 0%. On DAS6 both parts are evenly sized, on Snellius they are either evenly sized, see Figure 5a, or the first 16 threads are assigned the higher hematocrit subdomains, see Figure 5b.



(a) 64 / 64 (b) 16 / 112

Fig. 5: Snellius imbalanced hematocrit (18% and 0%) across 128 processes.

Because the fluid computation is not affected by the hematocrit for the results in this scenario we only show the time spend on the particle and coupling components. The results are presented in Figure 6.

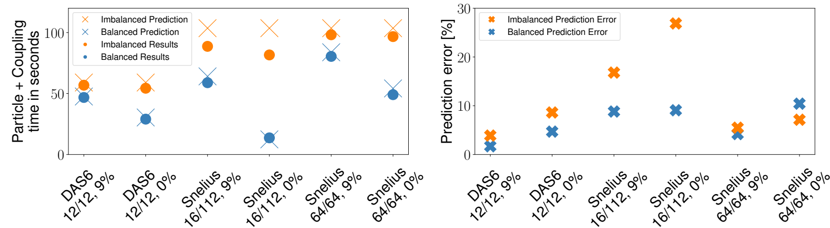


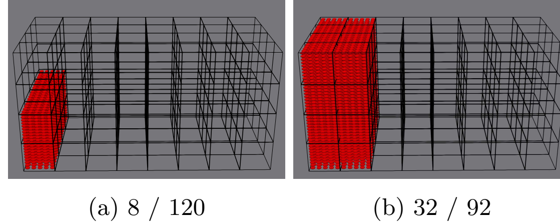
Fig. 6: Observed and predicted execution time and prediction error for the imbalanced hematocirt scenario, on DAS6 and Snellius. The standard deviation of the observed results is within 1.5%

For the imbalanced configurations where the domain is split in half, the highest observed prediction error is acceptable, at 8.61%. However, for the (16/112) configurations, the predictions are less accurate, with errors above 16.8%. The large error is caused by a change in the number of communication neighbors, which is *not captured* in the current model. We address this limitation in Section 4.4.

4.4 Scenario: Imbalanced Communication

In the imbalanced hematocrit results, we observe a lower prediction accuracy on the (16 / 112) distribution configurations. This is partially caused by a change

in the communication costs. The processes that are assigned more work, in the (16/112) configuration, as shown in Figure 5b, are located at the edge of the non-periodic domain. This means that the number of neighbors that need to be communicated with is less than if the subdomain computed by the process is located in the middle of the domain. However, during calibration it is assumed that the processes are fully surrounded by neighbors. To address this we expand the original model to include a term to express how many direct neighbors a process needs to communicate with.



(a) 8 / 120 (b) 32 / 92

Fig. 7: Snellius imbalanced hematocrit (18% and 0%) across 128 processes.

The model is expanded by adding CR_x, which denotes the communication ratio of the fluid or particle component, $CR_x = \frac{\#\text{Neighbors}}{\text{Max Neighbors}}$. This ratio is defined separately for the fluid and particle communication because for fluid communication the maximal number of neighbors that need to be communicated with is 18, as opposed to 26 possible neighbors for the particle communication[3]. In Equations (3) and (4) the functions describing the fluid and particle communications are replaced with Equations (6) and (7). The functions in Equations (6) and (7) multiply the original communication term with the newly introduced $CR_{component}$ ratio.

$$\begin{aligned}&\text{syncEnvelopes_comm}(RBCs, SA, CR_{particle}) = \\ &\qquad CR_{particle} \times \text{syncEnvelopes_comm}(RBCs, SA)\end{aligned} \tag{6}$$

$$\begin{aligned}&\text{collideAndStream_comm}(SA, CR_{fluid}) = \\ &\qquad CR_{fluid} \times \text{collideAndStream_comm}(SA)\end{aligned} \tag{7}$$

To verify that the model expansion improves the accuracy the updated model is applied to the (16 / 112) configuration, as well as a (8 / 120) configuration, shown in Figure 7a, and a (32 / 92) configuration, shown in Figure 7b. These experiments are run on Snellius for 500 iterations with a subdomain size of (50, 50, 50) µm. The results are presented in Figure 8.

The results show a clear improvement in the prediction accuracy, compared to the previous version without the added imbalance term. The highest prediction error is reduced from 24.53 % to 16.19% when using the updated model. The results in all experiments performed before this did not change, because in those configurations the processes that dominate performance need to communicate

[3] For CR_{fluid} neighbors with no RBCs are not counted, because the communication is overlapped by computation. For $CR_{particle}$ the value of each neighbor is scaled with the relative number of RBCs, i.e., if the neighboring hematocrit is half the neighbors value is 0.5.

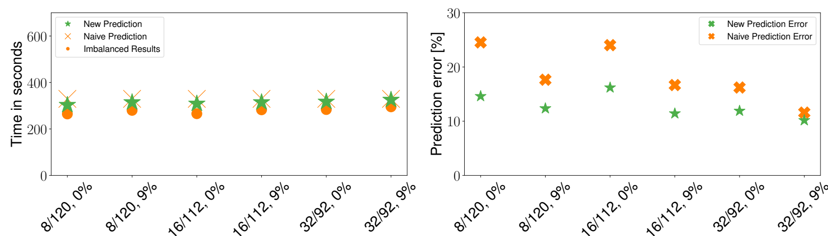


Fig. 8: Observed and predicted results and prediction error for the imbalanced communication scenario, on DAS6 and Snellius. Standard deviation of the observed results is within 1.5%

with the maximum number of neighbors. Not only is the accuracy better, but the results also provide more detailed information about the different configurations. With the old model, the prediction for each configuration is identical. However, in the results we see that the different configurations do not perform the same. The updated model is capable of better highlighting this difference in performance.

5 Related Work

This section provides a brief overview of alternative performance models, and how they differ from our own approach.

Analytical performance models for modeling the performance of large-scale applications have been proposed for many years [14]. However, such models are application-specific, and not generalizable to a wider range of applications. To address this, Hoefler et al. [10] propose a multistep approach for building analytical performance models. Their metric of interest is application scaling behavior; therefore, these models do not capture per-process performance.

Other tools to automate parts of the modeling process, such as *EXTRA-P* [6, 7], target on finding scalability bugs in large-scale applications. EXTRA-P builds a statistical model based on empirical performance results of the application. However, the resulting model is non-trivial to understand and tweak, and cannot predict scalability bugs at process-level.

Beyond analytical models, other types of performance models, such as simulators and machine-learning based models, are typically more accurate, but have interpretability issues. For example, models based on machine-learning require significant training data and resources, and the resulting black-box models provide a lot less insight into the application performance characteristics [13, 17]. Functional and cycle-accurate simulations provide accurate information on how an application behaves, and why, but they take a very long time to build and calibrate, which renders them difficult to use for large-scale applications [4, 18].

6 Conclusion

In this paper we proposed a methodology for building per-process performance models for large-scale, multi-processing simulations. The per-process modeling approach gives portable, fine-grained, accurate, analytical performance models.

We further used the proposed methodology to build an accurate predictive model for Hemocell, a coupled simulation for blood flow. We demonstrated that the resulting model is capable of accurate performance prediction for both balanced workloads, where we see a maximum error of 12.9%, and an average of 3.6 % error, and imbalanced scenarios, with a maximum of 16.2% error, and an average of 10.2% error. These results indicate that, although the model-building and calibration steps are based on simple balanced workloads, the model is able to analyze and predict simulation performance in load-imbalanced scenarios. This is a significant advantage for our per-process approach. Finally, we have shown how to refine the model to address potential inaccuracies, thus showing the advantage of a white-box, analytical approach.

In future work we aim to reduce the amount of manual work required to build the model, by for example incorporating statistical methods for determining performance behavior of code sections. In the near future, we aim to extend the model to work for multi-node computations and both apply our model to more simulations and increasingly dynamic scenarios. Specifically, at runtime, the workload could be dynamically shifting between processes as a result of the simulated phenomena, thus showing different load-imbalance patterns during execution. Thus, we plan to use the model to predict the performance degradation due to such changes in load balancing.

References

[1] Alowayyed, S., et al.: Load balancing of parallel cell-based blood flow simulations. J. Comput. Sci. **24**, 1–7 (Jan 2018). https://doi.org/10.1016/j.jocs.2017.11.008
[2] Axner, L., et al.: Performance evaluation of a parallel sparse lattice Boltzmann solver. J. Comput. Phys. **227**(10), 4895–4911 (May 2008). https://doi.org/10.1016/j.jcp.2008.01.013
[3] Bal, H., et al.: A Medium-Scale Distributed System for Computer Science Research: Infrastructure for the Long Term. Computer **49**(5), 54–63 (2016). https://doi.org/10.1109/MC.2016.127
[4] Bohrer, P., et al.: Mambo: A full system simulator for the PowerPC architecture. SIGMETRICS Perform. Eval. Rev. **31**(4), 8–12 (2004). https://doi.org/10.1145/1054907.1054910
[5] Borgdorff, J., et al.: Performance of distributed multiscale simulations. Philos Trans A Math Phys Eng Sci **372**(2021), 20130407 (Aug 2014). https://doi.org/10.1098/rsta.2013.0407
[6] Calotoiu, A., et al.: Using automated performance modeling to find scalability bugs in complex codes. In: SC 13. pp. 1–12. ACM (2013). https://doi.org/10.1145/2503210.2503277
[7] Calotoiu, A., et al.: Lightweight Requirements Engineering for Exascale Co-design. In: IEEE Cluster 2018. pp. 201–211 (2018). https://doi.org/10.1109/CLUSTER.2018.00038
[8] Geimer, M., et al.: The Scalasca performance toolset architecture. Concurrency Computat.: Pract. Exper. pp. n/a–n/a (2010). https://doi.org/10.1002/cpe.1556

[9] Germaschewski, K., et al.: The Plasma Simulation Code: A modern particle-in-cell code with patch-based load-balancing. J. Comput. Phys. **318**, 305–326 (2016). https://doi.org/10.1016/j.jcp.2016.05.013

[10] Hoefler, T., et al.: Performance modeling for systematic performance tuning. In: SC 11. pp. 1–12 (2011). https://doi.org/10.1145/2063348.2063356

[11] Knüpfer, A., et al.: Score-P: A Joint Performance Measurement Run-Time Infrastructure for Periscope,Scalasca, TAU, and Vampir. In: Brunst, H., et al. (eds.) Tools High Perform. Comput. 2011. pp. 79–91. Springer (2012). https://doi.org/10.1007/978-3-642-31476-6_7

[12] Latt, J., et al.: Palabos: Parallel Lattice Boltzmann Solver. Comput. Math. with Appl. (Apr 2020). https://doi.org/10.1016/j.camwa.2020.03.022

[13] Lee, B.C., et al.: Methods of inference and learning for performance modeling of parallel applications. In: Ppopp 07. pp. 249–258. PPoPP '07, Association for Computing Machinery (2007). https://doi.org/10.1145/1229428.1229479

[14] Mathis, M.M., Amato, N.M., Adams, M.L.: A general performance model for parallel sweeps on orthogonal grids for particle transport calculations. In: ISC '00. pp. 255–263. ICS '00, Association for Computing Machinery (2000). https://doi.org/10.1145/335231.335256

[15] Murtaza, S., Hoekstra, A.G., Sloot, P.M.A.: Compute Bound and I/O Bound Cellular Automata Simulations on FPGA Logic. ACM Trans. Reconfigurable Technol. Syst. **1**(4), 23:1–23:21 (Jan 2009). https://doi.org/10.1145/1462586.1462592

[16] Tarksalooyeh, V.A., Závodszky, G., Hoekstra, A.G.: Optimizing Parallel Performance of the Cell Based Blood Flow Simulation Software Hemo-Cell. In: Rodrigues, J.M.F., et al. (eds.) Comput. Sci. – ICCS 2019. pp. 537–547. Lecture Notes in Computer Science, Springer International Publishing (2019). https://doi.org/10.1007/978-3-030-22744-9_42

[17] Witt, C., et al.: Predictive performance modeling for distributed batch processing using black box monitoring and machine learning. Information Systems **82**, 33–52 (2019). https://doi.org/10.1016/j.is.2019.01.006

[18] Xu, G., et al.: Simulation-Based Performance Prediction of HPC Applications: A Case Study of HPL. In: 2020 IEEEACM Int. Workshop HPC User Support Tools HUST Workshop Program. Perform. Vis. Tools ProTools. pp. 81–88 (2020). https://doi.org/10.1109/HUSTProtools51951.2020.00016

[19] Závodszky, G., et al.: Cellular Level In-silico Modeling of Blood Rheology with An Improved Material Model for Red Blood Cells. Front Physiol **8** (2017). https://doi.org/10.3389/fphys.2017.00563

[20] Zavodszky, G., et al.: Hemocell: A high-performance microscopic cellular library. Procedia Computer Science **108**, 159–165 (2017)

[21] Zhu, X., et al.: Gemini: A Computation-Centric Distributed Graph Processing System. In: 12th USENIX Symp. Oper. Syst. Des. Implement. OSDI 16. pp. 301–316 (2016)