Is the PCM Ready for ACTORs and Multicore CPUs?—A Use Case-based Evaluation

Markus Frank
markus.frank@informatik.uni-stuttgart.de
Uni Stuttgart, Stuttgart, DE

Stefan Staude and Marcus Hilbrich
name.lastname@informatik.tu-chemnitz.de
TU Chemnitz, Chemnitz, DE

Abstract

Multicore CPUs have been common for years. However, developing parallel software is still an issue. To ease the development, software developers can use a range of frameworks and approaches, e.g., OpenMP, MPI or ACTOR. These approaches have an enormous impact on the performance of the software. Thus, **S**oftware **P**erformance **E**ngineering (SPE) needs to consider the impact of the parallelization approaches to deliver reliable results. In this paper, we evaluate the capability of the **P**alladio **C**omponent **M**odel[1] (PCM) based on the use case of a bank transaction example with a realization following the ACTOR approach. We observed that the accuracy of the performance predictions is unsatisfying, the modeling is challenging, and the characteristics of the ACTOR approach cannot be modeled. In future we need to consider additional attributes or properties to enrich the PCM as well to include concepts like active resources, message passing, and automatization concepts.

1 Introduction

Developing software that can be executed in parallel is challenging. In the past couple of years, new approaches like ACTOR or shared memory programming became popular. These approaches have in common that they abstract the parallelization to a level that is easy to use and enables the software developer to focus on the actual problem instead of handling low level parallelization challenges. Example challenges are synchronization, dead locks, life locks and so on. Based on the performance impact of the parallelization approaches, SPE can no longer ignore their effect or abstract them as implementation details. In order to continue to deliver reliable performance predictions SPE needs to consider such approaches in their performance prediction models.

In previous work [2] we performed a experiment that evaluated the capabilities of a state of the art performance prediction tool based on a matrix multiplication use case and shard memory approach realized by the OpenMP freamwork. We were able to show that the accuracy of the prediction decreases, once the number of used CPU cores increases.

[1]`http://www.palladio-simulator.com/`

In this paper, we extend the knowledge about modeling and performance prediction by evaluating an additional use case based on the ACTORs approach. When we talk about the ACTORs approach, we mean the ACTORs model and not a concert implementation. Thereby, we identified limitations during modeling the use case and also regarding the accuracy of the performance prediction results. On the modeling side, it was not possible to represent the ACTOR principle in the model. This results in a high abstraction and a lot of manual modeling overhead. In the future, we need active resources, message passing mechanisms, and enhanced tool support for the ACTOR approach. On the performance prediction side, we could show that the accuracy of the predictions drop to 40 %. Here we need to consider additional metrics in the prediction models like thread numbers, cache sizes, and memory hierarchies.

To present our results in a structured way, we first present the experiment setup (Sec. 2). Followed by the description of the conduction (Sec. 3), where a description of the implementation and modeling is given. In the last part, we discuss the obstacles we had to overcome and evaluate the capabilities of current tools to predict the performance of the ACTOR approach on multicore systems. We finish with lessons learned (Sec. 5) and an outlook (Sec. 6).

2 Experiment Setup

Research Questions: Based on the use case we answer, whether the PCM is suited for modeling a ACTOR. Therefore, we investigate the following research questions: (Q_1) Is it possible to model a parallel system following the ACTORS approach with the PCM? (Q_2) How accurate are the simulated predictions compared to the real execution?

Use Case Description: One of the most common use cases for ACTORs is a bank transaction system. This scenario considers a bank with multiple accounts, where each account has a balance and belongs to a customer, is considers. The customers issue transactions to transfer money. Each of the transactions needs a source account, a target account, as well as an amount of money to be transferred. All transactions must be logged and we assume that all balances

Measurement Results

Worker Threads	Ordered Set Time*	Ordered Set Speedup	Randomized Set Time*	Randomized Set Speedup
1	33.61 s	1.00	33.88 s	1.00
2	14.63 s	2.30	15.99 s	2.12
4	7.21 s	4.66	7.48 s	4.53
8	4.23 s	7.94	6.01 s	5.63
16	3.67 s	9.16	5.89 s	5.75

*mean execution time over all runs

Table 1: Measurement summary for both input sets

Simulation Results

Worker Threads	Ordered Set Time*	Ordered Set Accuracy	Randomized Set Time*	Randomized Set Accuracy
1	33.22 s	0.99	33.22 s	0.98
2	16.71 s	0.88	16.71 s	0.96
4	8.41 s	0.86	8.41 s	0.89
8	4.25 s	1.00	4.25 s	0.70
16	2.18 s	0.59	2.18 s	0.37

*model-based response time prediction

Table 2: Simulation result summary

must be positive. Thus, transactions can also fail.

Hardware: All calculations are performed on the same hardware characterized by 2 CPUs with 20 MB cache each and features like Hyper-Threading disabled. Each CPU contains 8 physical cores, at 2.4 GHz. We used Ubuntu 16.04.1 as operating system. Also, all measurements and simulations are conducted for 1, 2, 4, 8 and 16 worker threads.

3 Conduction

To evaluate the prediction capabilities of the PCM, we first implement the use case and measure it. In the next step, we model it. We use the initial sequential measurements to calibrate the model (all models, measurements, and source code are available in our repository[2]). The following elaborates on the steps:

Implementation: Link [1] already discussed how to implement the use case. We decided for an implementation according to Link, where each account is represented by an actor and transactions are handled by the source account sending a message to the target account. We issue a transaction, by sending a message from the experiment handler to the source account. An additional management actor is not used. The ACTOR approach is realized using the Akka Toolkit implementation [3].

Measurement: To analyze e.g., cache-based effects, two sets of input data are created. Both sets hold the same transections but in different order. The ordered data set performs all transaction of one source account after another, while the random data set uses a randomized order.

Each data sets holds two million transactions and uses five thousand accounts. For each number of used worker threads (1, 2, 4, 8, and 16), each data set is measured multiple times. We measured the time to perform all transactions of a set. Individual transactions are not measured to avoid measurement overhead. Based on multiple measurements, the average time is determined and presented in Tab. 1.

Model: The straightforward approach is to model each actor of the implementation with a deployed component in PCM. Manually deploy five thousand components wastes to much manpower and it was not clear how to handle the transactions. Thus, we decided to model all transactions as one component, called `Transactions`. The usage model calls the services of the component to perform transactions. Based on the behavior description of `Transactions` (the SEFF) it is statistically decided whether the transaction fails or not. Afterwards, the corresponding resource demands for the transactions are utilized. The probability that a transaction fails and the resource demands are calibrated based on the sequentially executed implementation.

To realize the parallel implementation models, the `Transactions` component is duplicated (according to the number of worker threads from 1 to 16). Also, an additional component (`ExperimentHandler`) is introduced. The `ExperimentHandler` distributes requests from the usage model and uses a `Fork` action to restrict the number of parallel processed transactions to the number of worker threads. By that, a message queue, that is not directly present in PCM, is modeled. Finally, the exact Linux O(1) scheduler [4] was used and the number of the CPU replicas has to be adjusted according to the number of available cores.

As result, the ACTOR concept based on actors and transactions more or less vanished in the model.

Simulation: According to the measurements, we run simulations for 1 to 16 worker threads. Based on the fact, that the order of the input sets cannot be modeled, the input sets are abstracted by one usage model, considering only the number of transactions.

Finally, all combinations of data sets and number of worker threads are simulated by using SimuCom. The results are shown by Tab. 2.

4 Evaluation

In the following we answer the research questions:

Evaluation of Q_1: Characteristic for the ACTOR approach is the "everything is an actor" principle. Realizing this approach in the PCM would mean to model every ACTOR by hand, which is a impossible task for our use case, considering that we have more than thousand actors. Further, the ACTOR approach is based on message passing, which is currently not supported by the PCM. To still use the PCM we had to abstract all of the ACTORs characteristics.

[2] `https://gitlab.hrz.tu-chemnitz.de/staus--tu-chemnitz.de/ssp_actors`

[3] `http://akka.io/`

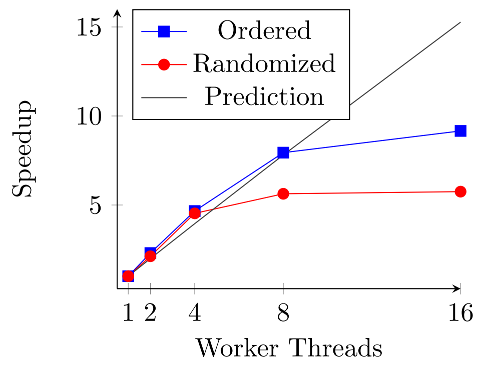

Figure 1: Speedup diagram based on the Measurements from the ordered and randomized input sets and the predicted speedup

We thus deem it is impossible to model the ACTOR approach within the PCM.

Evaluation of Q_2: Fig. 1 shows the speedup in relation to the number of used worker threads. As Sec. 3 mentioned, the execution of our implementation results in differing execution times depending on the input sets and therefore in a different speedup. Further, the figure shows the predicted speedup, which differs more with an increasing number of worker threads. An interesting observation is the achieved super-linear speedup up to 4 threads and the worse scalability starting from around 8 threads. We assume the reasons for the intense difference between the two sets can be found in the provoked caching behavior of the ordered set.

As Fig. 1 shows, the accuracy of the predictions drops with the number of used threads. This can be seen by comparing the measurement lines to the prediction line. Therefor we deduce, that the PCM is missing value metrics and prediction models to estimate the correct response time. These metrics are cache size, memory hierarchies, synchronization overhead produced by the ACTOR's framework. In total, we can say that the predictions are off by 63 % in the worst and by 41 % in the best case for 16 worker threads. That means, that the PCM is not suited to predict the performance of a system realizing the ACTOR approach.

5 Lessons Learned

Finally, we share the lessons we learned during the experiment in a brief and summarized way:

Dynamic threads: The ACTOR approach abstract the parallelization and hides the complexity from the software developer. Therefore, the developer has no influence of the threads created and thus the CPU cores uses. This is decided dynamically based on the available resources. Accordingly features to support dynamic thread allocation on model level are wishful but currently not supported.

Message Parsing: ACTORs use asynchronous message passing to communicate. This principle is not supported by the PCM. Active resources and parameterized events are required.

Manual Modeling: To represent the ACTORs, we would have had to model more than thousand actors by hand. This would have been an error prone and time-consuming task. Therefore, tool support is necessary to automatize the creation process.

Abstraction: We had to abstract the model to a degree, that it lost all the ACTOR specific characteristics. This task was not trivial and requested a lot of experience and manpower.

Individual Model: The presented model is only valid for the ACTOR approach. Other approaches (i.e., OpenMP) result in completely different models.

Caching: By using different input sets, we could provoke caching effects. These effects are not considered by the PCM.

6 Outlook

It is clear that the PCM needs to be extended to be ACTOR aware and aware of parallelization approaches on CPU level in general. To avoid creation of separate models for each number of worker threads and to reduce the needed effort of modeling. Therefore, the concept of Architectural Templets (AT) introduced by Lehrig [3], can be extended with additional ATs for parallel processing. This is currently under development. To respect physically shared resources like caches, that restrict the parallel efficiency, we will introduce according passive, shared resources to PCM. For first experiments, we discuss to reuse network resources and completions. Additionally, we identify a demand for active resources in PCM. Such resources are needed to model input queues, asynchronous communications, and event triggers.

All in all, the PCM and the performance prediction need to be extended to ensure to deliver accurate and useful results in future.

References

[1] J. Link. “Paradigmen für Nebenläufigkeit: Transaktionaler Speicher und Unveränderlichkeit”. In: *JavaSPEKTRUM* (2010).

[2] M. Frank and M. Hilbrich. “Performance Prediction for Multicore Environments—An Experiment Report”. In: *Proceedings of the Symposium on Software Performance 2016, 7-9 November 2016, Kiel, Germany.* 2016.

[3] S. Lehrig, M. Hilbrich, and S. Becker. “The architectural template method: templating architectural knowledge to efficiently conduct quality-of-service analyses”. In: *Software: Practice and Experience* (2017). spe.2517.

[4] J. Happe. “Predicting software performance in symmetric multi-core and multiprocessor environments”. PhD thesis. Karlsruhe: Zugl.: Oldenburg, Univ., Diss., 2008.