CHAMELEON: Reactive Load Balancing and Migratable Tasks for Hybrid MPI+OpenMP Applications

Jannis Klinkenberg
Christian Terboven
Matthias S. Müller
j.klinkenberg@itc.rwth-aachen.de
terboven@itc.rwth-aachen.de
mueller@itc.rwth-aachen.de
Chair for High Performance Computing, IT Center, RWTH Aachen University
Aachen, Germany

Philipp Samfaß
Michael Bader
samfass@in.tum.de
bader@in.tum.de
Department of Informatics, Technical University of Munich
Garching, Germany

Karl Fürlinger
karl.fuerlinger@nm.ifi.lmu.de
Department of Informatics, Ludwig-Maximilians-University Munich
Munich, Germany

ABSTRACT

Many HPC applications are designed based on underlying performance and execution models. These models could successfully be employed in the past for balancing load within and between compute nodes. However, the increasing complexity of modern software and hardware makes performance predictability and load balancing much more difficult. Tackling these challenges in search for a generic solution, we present a novel library for fine-granular task-based reactive load balancing in distributed memory based on MPI and OpenMP. Our concept allows creating individual migratable tasks that can be executed on any MPI rank. Migration decisions are performed at run time based on online performance or load data. Two fundamental approaches to balance load and at the same time overlap computation and communication are compared. We evaluate our concept under enforced power caps and clock frequency changes using a synthetic benchmark and demonstrate robustness against work-induced imbalances for an AMR application.

KEYWORDS

Reactivity, Task migration, Hybrid MPI+OpenMP, Load balancing, Tasking

1 INTRODUCTION

Over the past decades, many applications in high performance computing (HPC) have been developed under the assumption of an underlying homogeneous execution environment where all cores or compute nodes of a larger system have the same speed. Consequently, executing the same work on every node should also take the same time. In the past, this model has been proven to be highly accurate and efficient to balance computational load, e.g., for static partitioned problem domains. However, due to the increasing complexity and heterogeneity of hardware and software, these assumptions no longer hold for current and future systems.

In fact, *dynamic variability* and run time variations are already observable for today's architectures due to dynamic voltage and frequency scaling (DVFS), the use of sophisticated memory hierarchies comprising caches, DRAM, HBM and NVRAM or features like Intel's Turbo Boost [1]. In addition, varying CPU power efficiencies that arise from the manufacturing process can result in performance variations under an enforced power cap [2].

Further, *dynamic variability* can also be caused by applications such as particle simulations or iterative codes employing adaptive mesh refinement (AMR) where the computation load changes over time resulting in load imbalances in both shared and distributed memory. The literature describes several approaches to tackle this problem such as global repartitioning of work or data in regular intervals or Charm++. However, to the best of our knowledge they act on a rather coarse-grained level at fixed synchronization points where load balancing is triggered and running exclusively or might induce higher overhead for data transfers. In order to prevent performance declines and imbalances resulting from hardware variability that can arise on a very short time scale, we believe that it is inevitable that the application is able to dynamically react on the changing environment or execution conditions.

To compensate these shortcomings we present *CHAMELEON*, a novel library for fine-grained reactive load balancing in hybrid MPI+OpenMP task-parallel applications that enables load balancing within and across process boundaries at run time. Further, our approach builds on top of well established standards MPI and OpenMP allowing an incremental integration into a large amount of existing applications developed in C, C++ and Fortran.

In our previous work [6], we conducted a feasibility study with a prototype implemented in sam(oa)^2 [4, 5], a PDE framework that employs AMR. In this work, we discuss general requirements and objectives as well as a generic solution that can be used with arbitrary hybrid task-parallel applications [3]. In a systematic evaluation we show effectiveness against both, hardware- and work-induced imbalances.

2 CONCEPT & APPROACHES

Our concept is based on *migratable task* representing basic units of work without any side effects that can be executed locally or migrated and executed on a remote MPI rank. In case an an imminent imbalance is detected at run time our approach attempts to quickly migrate task to underloaded processes.

2.1 Assumptions & Objectives

In order to establish a general solution for fine-granular reactive task-based load balancing we identified three essential components: A *task execution environment* that is able to create, execute and

migrate tasks, a *self introspection component* that constantly monitors load and execution conditions of the current process/rank and an *analysis component* that consolidates all per-rank introspection data in order to detect emerging imbalances and decide whether to migrate tasks. Further, our implementation has the following key objectives [3]:

Reactivity: Since load imbalances can result from dynamically changing execution conditions or computational load on a very short time scale, it is inevitable to detect these changes as quickly as possible.
Smart decision making: Relying on permanently collected introspection data an implementation has to identify an emerging imbalance and decide whether to migrate tasks or not. Further, it has to select adequate victims to migrate tasks to. However, inaccurate or incorrect decisions can result in a performance decline.
Hiding overhead: Compared to work stealing in shared memory, migrating tasks in distributed memory induces additional overhead as task-related information and data needs to be transferred over the network. Consequently, it is desired to migrate tasks as soon as possible to sufficiently overlap communication with computation and hide any migration overhead.
Ease of integration: Augmenting existing applications with task migration should not require extensive programming efforts or code modifications.
Generalization and modularity: Although the objective is to create a generally applicable solution that can be integrated into arbitrary applications, it might be desired and profitable to customize introspection / load specification or migration strategy in order to incorporate domain and application knowledge. Nevertheless, an implementation should provide a default behavior.

2.2 Approaches

This work compares the following two fundamental approaches. In addition to the current *task migration* approach this work introduces our new *task replication* concept and the advantages that come along with it.

Task Migration

With this reactive approach tasks are migrated from an owner rank with higher load to a remote rank as soon as impending imbalances are detected. Migrated tasks are executed on the remote rank and resulting data will be send back to the task owner. Execution of migrated tasks are prioritized to overlap computation (of local tasks) and communication.

Task Replication

This more speculative approach replicates multiple tasks to underloaded remote ranks at the beginning. However, in contrast to *task migration* the owner keeps local copies. In presence of an imbalance remote execution of replicated tasks can be triggered. In case the owner runs idle and replicated tasks have not been executed by remote ranks it can immediately start working on the local copies and cancel the replicates. This approach has the opportunity to better hide migration overhead and features a higher robustness against late result transfers and incorrect migration decisions.

3 EVALUATION

The *task migration* approach is evaluated against hardware variations as well as work-induced imbalances. All tests are conducted on the HPC system of RWTH Aachen University (CLAIX) featuring an Intel Omni-Path interconnect and dual-socket Intel Xeon E5-2650v4 (codename "Broadwell") processor nodes with a TDP of 105 W and 24 cores in total running at 2.2 GHz.

To evaluate robustness against hardware variations we use a synthetic hybrid matrix multiplication benchmark where every rank has to solve 2400 matrix multiplications $C = A * B$ with the same complexity. Thus, all ranks are expected to finish the calculation in the same time. To visualize effects of varying hardware efficiencies and demonstrate robustness against dynamic variability we run tests under enforced power caps and frequency variations where a single node is rendered slower. We show that in both cases *task migration* helps to balance the load and achieve speedups of up to 1.3 X (depicted in Figure 1 and 2 on the poster).

In order to evaluate robustness against unpredictable work-induced imbalances we conduct experiments with the PDE framework sam(oa)^2 simulating 60 min of the Tohoku tsunami in 2011. We show that *task migration* can reduce emerging imbalances leading to a speedup of up to 1.2 X (depicted in Figure 3 and 4 on the poster).

4 CONCLUSION & FUTURE WORK

In this work, we presented *CHAMELEON*, a library for fine-grained reactive load balancing in hybrid MPI+OpenMP task-parallel applications. We demonstrate how continuous introspection and the concept of migratable tasks help to effectively balance load in distributed memory at run time. In our future work, we will evaluate the *task replication* approach that is currently under development. Further, we plan to investigate the support of dependencies between migratable tasks to broaden the class of applications that can benefit from our concept.

ACKNOWLEDGMENTS

Some of the experiments were performed with computing resources granted by JARA-HPC from RWTH Aachen University under projects jara0001 and nova0027. Parts of this work were funded by the German Federal Ministry of Education and Research (BMBF) under grant number 01IH16004 (Project Chameleon).

REFERENCES

[1] Bilge Acun, Phil Miller, and Laxmikant V. Kale. 2016. Variation Among Processors Under Turbo Boost in HPC Systems *(ICS '16)*. ACM.
[2] Yuichi Inadomi, Tapasya Patki, Koji Inoue, Mutsumi Aoyagi, Barry Rountree, Martin Schulz, David Lowenthal, Yasutaka Wada, Keiichiro Fukazawa, Masatsugu Ueda, Masaaki Kondo, and Ikuo Miyoshi. 2015. Analyzing and Mitigating the Impact of Manufacturing Variability in Power-constrained Supercomputing *(SC '15)*.
[3] Jannis Klinkenberg, Philipp Samfass, Michael Bader, Christian Terboven, and Matthias S. Müller. 2019. Reactive Task Migration for Hybrid MPI+OpenMP Applications. In *To appear in: Parallel Processing and Applied Mathematics (PPAM 2019)*.
[4] Oliver Meister. 2010. sam(oa)2 - CHAMELEON version. Retrieved August 7th, 2019 from https://github.com/chameleon-hpc/samoa-chameleon
[5] Oliver Meister, Kaveh Rahnema, and Michael Bader. 2016. Parallel Memory-Efficient Adaptive Mesh Refinement on Structured Triangular Meshes with Billions of Grid Cells. *ACM Trans. Math. Software* 43, 3 (2016), 1–27.
[6] Philipp Samfass, Jannis Klinkenberg, and Michael Bader. 2018. Hybrid MPI+OpenMP Reactive Work Stealing in Distributed Memory in the PDE Framework sam(oa)^2 *(CLUSTER '18)*.