ParaMASK: a Multi-Agent System for the Efficient and Dynamic Adaptation of HPC Workloads

Mateusz Guzek†, Xavier Besseron‡, Sébastien Varrette*, Grégoire Danoy* and Pascal Bouvry*
†Interdisciplinary Centre for Security Reliability and Trust
‡Research Unit in Engineering Science
*Computer Science and Communications (CSC) Research Unit
6, rue Richard Coudenhove-Kalergi, L-1359 Luxembourg, Luxembourg
Firstname.Lastname@uni.lu

Abstract—The growing parallelism and heterogeneity of modern computing infrastructures such as High Performance Computing (HPC) platforms raises new challenges to their programmers and users. Additional requirements have emerged nowadays, such as minimizing the consumed energy, reducing the utilized system resources, or providing built-in reliability mechanisms. Therefore, High Performance Computing (HPC) applications require adaptation mechanisms and then must avoid traditional monolithic centralized approaches in favor of novel autonomous, flexible and decentralized decision systems. In this context, we describe here a dynamic and flexible adaptation scheme based on a Multi-Agent System (MAS) to handle parallel or distributed executions in an HPC environment. More precisely, we model and extend the existing HPC middleware KAAPI to offer the power of the ParaMoise multi-agent organizational framework. Our proposed solution, named ParaMASK, relies on the similarities between ParaMoise workflow-based functional specifications and the Direct Acyclic Graph (DAG) representation of the distributed execution within KAAPI. As a result, Parmask permits to analyze and reorganize the scheduling of tasks that compose a program in an autonomous and decentralized way, while additionally handling dynamic adaptations (using task migration to fulfill energy consumption goals for example). The proposed solution was implemented on top of the existing KAAPI middleware and includes an optimized algorithm for the agent coordination. ParaMask has been validated with a series of experiments on a real computational grid. Experimental results show a good scalability and an exceptional low overhead induced by the approach: less than 1.5% execution time increase with periodic coordinations every 15 seconds on 2662 cores.

Keywords—Multi-Agent System (MAS); High Performance Computing (HPC); DataFlow Graph (DFG); KAAPI; ParaMoise;

I. INTRODUCTION

With a growing concern on the considerable energy consumed by High Performance Computing (HPC) systems and data centers, research efforts are targeting toward green approaches with higher energy efficiency. In this context our objective is to investigate novel ways to make the executed application aware of the underlying computing platform so as to take the best decisions as regard process migration and scheduling, i.e., more generally, runtime adaptation. Indeed, the quest for highly adaptable applications, where the reconfiguration is operated using heterogeneous criteria, reveals the need to raise the level of abstraction within the application and in result to allow to reason with coarse-grained concepts.

Common HPC applications and middleware usually provide an highly optimized code and focus on computation efficiency. Then one drawback appears to be the lack of flexibility and malleability of such software. On the other hand, the Multi-Agent System (MAS) paradigm brings new capabilities to computing systems, by explicitly modeling the high-level characteristics of the relations and interactions between system components. One important aspect of MAS is their ability to perform distributed decision making. We propose to combine these two paradigms to offer in the same middleware high performance execution together with expressiveness and flexibility. It is of special interest for new HPC platforms, where runtime fine-tuning is dependent on multiple changing parameters, to optimize the system dynamically.

Our approach relies on extending an existing HPC middleware KAAPI which has proved to be efficient and scalable [1]. Additionally, KAAPI represents a parallel computation as a Direct Acyclic Graph (DAG) which serves as a portable representation for the distributed execution of a parallel program on a fixed input. This concept of DAG is then coupled with the ParaMoise MAS organizational model introduced in [2], itself based on Moise [3] and $\mu$Moise [4]. The DAG in ParaMoise is the core of Functional Specification, used by agents to plan and reason about their goals and missions. At the end, our solution called ParaMASK, unifies KAAPI and ParaMoise and offers a flexible MAS-based middleware with performance close to the original KAAPI even for large scale execution.

The main contributions of this paper are: 1. The model of a distributed HPC execution as an ParaMoise-based organizational Multi-Agent System (MAS), capable of dynamic reorganization. 2. Its implementation named ParaMASK on top of the existing KAAPI middleware, including an optimized agent-coordination algorithm. 3. An experimental validation of ParaMask in a large scale, highly heterogeneous and geographically distributed environment.

This paper is organized as follows. Section II reviews the related work as regards state-of-the-art approaches to cover runtime adaptation in the literature. Then, Section III describes the background knowledge, while the modeling and implementation of a ParaMASK system is presented in Section IV. The validation of the approach on concrete applications is explained in Section V which details and discusses the experimental results obtained. Finally, Section VI concludes the paper and provides some future directions and perspectives opened by this study.
II. STATE OF THE ART

The problematic of dynamic adaptation in HPC workloads is obviously not a new topic as it has already been tackled in many middleware. For instance, ACFPAC [5] and ASSIST [6] are based on the parallel component model and provide a framework for generic adaptation. However, these are limited by the capabilities of the underlying application they control. ASSIST applies only to SPMD (Single Program, Multiple Data) programs and master-slave models, and ACFPAC is limited to SPMD applications. Regarding the MPI implementations, AMPI [7], Starfish MPI [8], GrADS [9] propose a dynamic adaptation of the application but only consider adding, removing or migrating processes. However, they use an ad-hoc method which is strongly tied to their internal implementation. It means that their approach is not generic enough and thus they can hardly be extended to other kinds of adaptations.

The agent paradigm, and more precisely organized multi-agent systems, have thus been considered to address these limitations and bring the expressiveness and power of organizations to coordinate and control autonomous agents. A large number of organization modeling languages and “organization-oriented” frameworks have been proposed in the MAS literature, but only few of them actually permit to model and/or implement highly parallel HPC programs in an adaptive way. For instance, the JaCaMo multi-agent programming framework [10], brings the power of Moise [3] with its three specifications (structural, functional and deontic) for programming MAS organizations and reorganizations. However, its usage for executing HPC programs is limited because of the single agent responsible for reorganization that results in a sequentially executed implementation plan. Moreover, the organization needs to be stopped for the reorganization, which may be unfeasible in a real scenario. Some other organizational MAS frameworks have thus been developed specifically to manage executions in such distributed environments. For instance, Al-Jaroodi et al. [11] propose a Java-based middleware that permits to run user applications implemented in Java on a distributed environment. The core of the concept is a hierarchical structure of agents. Leader agents are superior over the agents in lower layers, while being equal among other leaders in the same layer. The system is presented to the end user as an infrastructure and offers functionalities such as resource, request, and security management, class loading or scheduling. The available reorganizations are only on the Organization Entity (OE) level, meaning that the structure of the system is fixed. Additionally, the organization and its functioning are problem specific, which limits the flexibility and generality of this system. Martin and Barber [12] empirically prove the usability of an adaptive decision scheme and its superiority in a dynamic MAS setting over static organizations. Kota et al. [13] propose a MAS with decentralized decision making, designed as a framework that runs service instances in a distributed computing environment. The solution enables to define the relations between agents either as superior-subordinates, peers or acquaintances. Because of that, the possible reorganizations are limited to transitions between these relations and less focus is put on their implementation. Moreover, the possible changes in the system are only creating/dissolving the relations or load management functions. As a result, this approach does not introduce fine-grain, general model that could be further extended and fully explain its functionality (e.g., load management functions are separate from the organization management functions). The experiments prove the applicability and the benefit of the approach, concluding that its performance is close to the centralized solution or an oracle.

Wang et al. propose [14] a trust-based system for plan management in multi-agent systems. The objective of the work is to maximize the probability of success of the execution. The methodology is based on the additional control loop that adapts the schedule, directed by each agent’s levels of trust, resulting in increased robustness and predictability. The proposed solution is problem specific and does not propose a generalized organizational framework.

To summarize, these dedicated MAS frameworks do not fully exploit the organizational modeling power, i.e., not all dimensions in the organization and reorganization. Moreover, the reorganization process is done offline or its performance is neglected. On the other hand, stat-of-the-art HPC middlewares can exploit the resources of computing platforms, however they lack dynamic adaptation and reorganization capabilities. In this work we therefore propose ParaMASK, which demonstrates how the ParaMoise model can bring explicit, high-performance organization and reorganization capabilities at runtime to a parallel execution middleware, i.e., KAAPI.

III. BACKGROUND

A. ParaMoise

ParaMoise [2] is an organizational model based on the Moise+ [4] framework, from which it inherits the division of an organization into three specifications: Structural (roles, groups, and relations between them), Functional (goals, missions, and corresponding relations) and Deontic (obligations and permissions in MAS). The three specifications form the Organizational Specification (OS), which is a meta-description of a MAS, and Organizational Entity (OE), which is a description of an instance of a MAS following a specific OS. The main objective of ParaMoise is the efficient execution in distributed systems, introducing the concept of Workflow Specification (WFS) into the Functional Specification. A WFS is a Directed Acyclic Graph (DAG), with nodes representing agents’ goals and edges representing precedence constraints between goals. It is possible to nest WFS, i.e., encapsulate a

![Figure 1. An example of Workflow during execution [2].](image-url)
WFS in another WFS as a goal. An instantiation of WFS is called Workflow (WF). A WF includes an additional labeling of links and nodes, which is used to track and synchronize operations of the MAS. Each goal is associated with a mission. A mission can be therefore seen as a subset of goals. Missions are bound with roles from the Structural Specification in the Deontic Specification that includes permissions and obligations of the roles in a system. Figure 1 presents an example of a ParaMoise WF during its execution. WFS and WF are explicitly defined as a part of the OE, thus they can be modified during a reorganization process. The properties of a WF ensure progress and basic coordination, while additional read and write locks ensure non-destructive modifications of an organization at runtime. The reorganization is performed by a specialized group of agents (ReorgGr), governed by multiple Organization Managers (OrgManagers). As a result, a ParaMoise-based system is able to concurrently execute multiple reorganizations, if they are not conflicting. The reorganization process performance and reliability is enhanced by the multiplicity of OrgManagers.

The ParaMoise is a theoretical model, but its possible implementation decisions were also discussed [15], indicating that an efficient implementation of the ParaMoise-based system is dependent on the properties of the distributed system, characteristics of its dynamics and its environment. The first introduction of the concept of a joint usage of ParaMoise and KAAPI is related to an energy-aware computing environment [16], however the general execution framework, its implementation details, and its validation have not yet been presented and remain the object of this paper.

B. Data Flow Modeling & the KAAPI Runtime Middleware

KAAPI [1] is a high performance middleware that allows to execute parallel applications on a distributed platform, e.g. a grid or a cluster. It offers a high-level programming interface called Athapascan [17] that allows to describe an application as a DataFlow Graph (DFG) independently of the execution platform. The DFG of the application is based on two simple concepts: shared data and tasks. A shared data is a data in global memory that a task can produce or consume. A task is an instruction set that declares an access mode to a shared data (read or write). DFG is formally defined as a directed graph \( G = (V, E) \), where \( V \) is a finite vertex set (tasks and shared data) and \( E \) is an edge set (precedence constraints) between the vertices. With this description, KAAPI can execute the tasks of the application according to the precedence constraints which are dynamically detected. The DFG is called the abstract representation of the application. This representation is causally connected to the (execution of the) application: any new execution of an API instruction is reported by the creation of new vertices in the DFG; and any modification in the DFG is rendered in a modification in the application execution. Practically, it means that the state of the application can be dynamically inspected or modified at runtime to adapt the execution of the application (to take scheduling decisions for example).

Figure 2 shows an example of the data flow representation for the simple recursive Fibonacci application. The Fibo tasks are recursive, so the execution of Fibo(3) actually creates two additional Fibo tasks and one Sum task. This operation creates parallelism in the application and the concurrent branches in the graph can be distributed among processors to perform a parallel execution. An approach to distribute the parallelism on different processors and schedule the tasks is the work-stealing. When a processor is idle, i.e. when it has no work assigned, it will try to steal work from another busy processor. The DFG on the victim processor is then split into two independent DFGs. A communication task (Signal) is created to maintain the precedence constraint and sends the data to the other DFG. This distribution of the work on two DFG is illustrated on Figure 3. The initial DFG, before the work-stealing operation, is showed on the left; the split DFG, after work-stealing is on the right. This last notion is of importance for the migration process implemented in the ParaMASK framework.

IV. PROPOSED MODEL AND ORGANIZATION

The presented ParaMoise framework and the KAAPI execution environment were designed using similar concepts to address similar issues. This work leverages on the complementarity: KAAPI is used as an engine and foundation, encapsulating the organization and reorganization capabilities of the ParaMoise model. As a result, the ParaMoise Multi...
Agent System in KAAPI (ParaMASK) was designed and implemented. This section presents this solution in details.

A. Modeling Kaapi runtime using ParaMoise.

Table I presents a proposed mapping that unifies varying terminologies. The mapping is not exact, therefore it is necessary to explain it in more details. The basic actors, i.e. agents, are not explicitly defined in ParaMoise, while in KAAPI they are defined POSIX threads. Notions of Goal and Task are quite similar: they are notions of workloads, that should produce results. In both frameworks there are two levels of execution description. WFS and Abstract Representation are used to model a general plan of execution, while their corresponding instantiations are WF and DFG. However, the mapping on both levels is not straightforward. To map an Abstract Representation to a WFS, the shared data nodes must be reduced by transforming them into edges. Reverse mapping involves designing shared data structures required during execution. Analogical transformations can be performed on the WF and data flow graph. An example of the transformation of the data flow graph from Figure 2 is presented in Figure 4.

B. ParaMASK - ParaMoise MAS in KAAPI.

We propose ParaMASK (ParaMoise MAS in KAAPI), a system which uses the ParaMoise organizational framework, adapted to the needs of KAAPI. ParaMASK is able to execute KAAPI-compatible programs and to dynamically reorganize the middleware modeled as a ParaMoise MAS. To reflect the specifics of KAAPI, a novel, specialized Structural Specification is proposed (Figure 5). The ReorgGroup is extended in comparison to standard ParaMoise definitions by the addition of a LocalManager. The main roles for the system are OrgManager, LocalManager, and Worker. A sample presentation of a deployed system is presented in Figure 6.

OrgManagers have an authority over all roles in the organization (inter-group link between OrgManager and soc, the root of all roles), including LocalManagers. OrgManagers are used as the highest level entities that govern the organization and enforce reorganizations. LocalManagers form an intermediary layer in the hierarchy. To optimize the management of the Worker agents, each physical node has an associated group with a single LocalManager and all Workers of the node. The LocalManager of a Worker Group (WorkerGr) has authority over Workers in the group. LocalManagers can communicate between themselves, which is used during the coordination process. Because there is always exactly one LocalManager per KAAPI process, they are mapped as corresponding entities, however technically a LocalManager, being an agent, is still a thread. Workers are the responsible for processing the DFG. To balance the load, they perform work stealing operations. To achieve this, each Worker can communicate with its peers. Because of that, a Worker asks other agents to find work instead of parsing the DFG. In case of successful work stealing, the assignment of the involved agent to the targeted goals is changed.

The Deontic Specification that constitutes the foundation of the KAAPI system is relatively simple. All Workers have the obligation to process tasks. In case no task is assigned to a Worker, the Worker is obliged to steal work. In case it is impossible, the Worker becomes idle, periodically checking if it can contribute. The parts of Deontic Specification concerning managers roles are problem-dependent and thus not described here. However, it is crucial that all agents respect the authority relations.

C. Implementation details & Agent coordination

ParaMASK is implemented in C++ as an extension to KAAPI. Management Layer agents are implemented as dedicated system threads that run concurrently to Workers (i.e. KAAPI threads, which are modified POSIX threads) and the KAAPI communication layer. This specialized thread offers a general agent interface, which can encapsulate an arbitrary code responsible for the behavior of an agent. Technically an agent can be assigned to multiple roles, however because of the low overhead of agents, each agent adopts only a single role in the implemented system, as presented in Figure 6. Agents can be responsible to perform different kinds of operations: Work-Stealing, Checkpoint/Restart, Migration, etc. For example, the Work-Stealing operation involves two Worker
agents and requires a coordination only between these two agents. This operation, although not formalized using the agent concept, was already available in the initial KAAPI [1]. Other operations like Checkpoint/Restart or Migration require stronger coordination between the processes to maintain a consistent state of the application. In ParaMASK, the role responsible for coordination is LocalManager, as it has direct authority over Workers in its WorkerGr. Because there is a unique LocalManager per process, we propose a mapping between these two entities.

The coordination of the threads is important as it guarantees that a distributed operation, like a Checkpoint or Migration will operate on a global consistent state [19] of the application. Such a coordination usually ensures two aspects [18]. First, the state of the communication must be taken in account (in progress messages must be recorded or delivered). Secondly, the distributed operation has to appear as applied at the same time on all the processes (the operation cannot be applied before the emission of a given message and after the reception of the same message on the other process). Multiple coordination algorithms [19], [20] have been proposed to build a consistent global state between distributed processes and solve these issues. To coordinate agents in ParaMASK, we generalize the coordination protocol proposed in [20], [18] to apply to any kind of operation. Then, we have two coordination schemes available. The Full coordination corresponds to a naive algorithm which involves communications between all pairs of processes. It has been implemented to serve as a reference for our experimental evaluation.

The Optimized coordination algorithm is inspired from the protocol described in [21]. It leverages the knowledge of the application, via its abstract representation (a DFG), to flush communication channels only between processes that can actually communicate. This allows to considerably reduce the number of messages required for the coordination from $O(N^2)$ to $(N)$ with $N$ being the number of involved processes [18]. The overhead induced by the ParaMASK framework during agent coordination is therefore largely reduced.

V. VALIDATION AND EXPERIMENTAL RESULTS

The experimental tests performed as part of the current study have been carried out on the Grid’5000 platform [22] which provides a fully customizable testbed to perform advanced experiments in all areas of computer science related to parallel, large-scale or distributed computing and networking. Grid’5000 is a distributed heterogeneous computing platform featuring clusters located in nine geographical sites – eight in France (Bordeaux, Grenoble, Lille, Lyon, Nancy, Reims, Rennes, Sophia, Toulouse) and one in Luxembourg. It offers therefore a flexible environment of choice to validate the framework presented in this article. We evaluate our ParaMASK framework with the N-Queens application. The application counts the number of solutions to the N-Queens problem and is written as a recursive DFG. Executed with the KAAPI runtime, it is able to scale to many thousand of cores.

A. Scalability of the MAS

We performed a first experiment to evaluate the scalability of our ParaMASK implementation. The scenario is to consider the global coordination of all of the LocalManagers to obtain a global consistent state of the distributed execution of DFGs. For this study, we use an empty reconfiguration, i.e. no operation is performed by the LocalManagers, except synchronizing the state of the subordinate Workers, while their states are coordinated. In order to obtain a larger number of LocalManagers in this experiment, we run a ParaMASK with one LocalManager and one Worker per core. Effectively, they can run concurrently on the same core, as the LocalManager is lightweight in comparison to the Worker. The ParaMASK N-Queens application has been deployed and run up to 1172 cores (154 nodes from 7 sites of Grid’5000). The details of the nodes from Grid’5000 used to run this experiment are given in the table II.

We measured the coordination time of all LocalManagers for an increasing number of agents involved. The coordination time includes the time to synchronize all LocalManagers and to build a consistent state of their subordinate Workers (i.e. computation is stopped and communication channels are flushed). It is measured by the OrgManager that triggers the...
In this work, we present a dynamic Multi-Agent System (MAS) capable of runtime reorganizations, with an explicit organizational model, named ParaMASK. The theoretical aspect of the study includes the mapping between ParaMoise and KAAPI entities. We further explain the implementation of coordination and waits for the termination signal from all the LocalManagers. We compare our Optimized coordination scheme with the Full coordination approach which just flushes naively the communication channels between all pairs of LocalManagers. Our Optimized algorithm extracts dynamically the communication pattern of the application to flush the communication channels only between the pairs of LocalManagers which subordinate Workers can actually communicate. Figure 7 shows the average coordination time for different number of LocalManager agents. The average value is calculated with at least one hundred values. We clearly see the benefit of using the optimized coordination algorithm, especially in the case where agents are distributed over multiple sites. The noticeably longer coordination time between experiments run on one site (less than 120 cores on Figure 7) and multi-site experiments can be explained by a larger network latency of inter-site network links in comparison with intra-site network links. Our optimized coordination algorithm is able to coordinate 1172 agents distributed on 7 sites of Grid’5000 in less than 120 milliseconds.

### B. Overhead of the MAS

In the second experiment, we measured the overhead of the ParaMASK system against a reference execution with standard KAAPI. We used the N-Queens application and solve the problem of size 21. We performed this experiment on 2662 cores distributed on 331 nodes of Grid’5000 (see Table II). Contrary to the previous experiment, there were deployed one LocalManager per node, and one Worker per core. For the reference time, we use the execution time using the original KAAPI, i.e. with no agent layer, and we noted an average execution time of 44.63 seconds.

During the execution global coordinations of the LocalManagers are triggered periodically by OrgManager in order to build a consistent state of the Workers, which enables the measurement of the overhead of the ParaMASK. No modification of the application state is made while the LocalManagers coordinate. The average execution time is measured and averaged for different frequency of global coordinations. The overhead compared to the reference is reported in Figure 8.

![Figure 8. Overhead of periodic coordination of ParaMASK compared to a standard KAAPI execution.](image)

When no global coordination is performed during the execution (the 'None' bar), there is no significant overhead. This effect is expected because the LocalManagers are started but do not coordinate, which results in negligible overhead. The overhead increases when the global coordinations are more frequent. The highest overhead, 22 % is measured when a global coordination of the processes is triggered every second, which is a high and not realistic frequency in the context of dynamic adaptation of a parallel application, as it would correspond to approximately 55 coordination events. With lower coordination frequency, the overhead is more acceptable. The proposed implementation allows LocalManagers to coordinate all of the 2662 Workers of the application every 20 seconds with less than 1.3 % of overhead. It is important to note that such a global coordination is only required when a change of configuration of the application is triggered by the dynamic adaptation mechanism that would affect the entire organization, which is usually expected to occur at a much lower rate.

### VI. Conclusion

In this work, we present a dynamic Multi-Agent System (MAS) capable of runtime reorganizations, with an explicit organizational model, named ParaMASK. The theoretical aspect of the study includes the mapping between ParaMoise and KAAPI entities. We further explain the implementation of...
the system in a high performance computing application. As a result, the ParaMoise organization is embedded in a distributed computing engine. The resulting ParaMASK system enables the control of the distributed processes via an organizational framework, which simplifies the achievement and formulation of the auxiliary goals of a system.

The system is benchmarked and validated using general, system-wide coordination events during the execution of a standard application. The experimental validation presents the superiority of using the data flow specifics over a naive, flat coordination scheme. An optimized coordination mechanism has lower complexity, which results in a speedup measured in orders of magnitude for large-scale deployments. It allowed to coordinate in less than 120 milliseconds for a system with 1172 LocalManagers. The application execution time overhead of ParaMASK is dependent on the frequency of coordinations and it is acceptable for reasonable range of values, e.g. 1.41 % for coordination every 15s.

The future work induced by this study includes the implementation of the problem specific roles with corresponding Functional Specification, to enable system-wide objectives, such as energy-saving, improved reliability, or dynamic system monitoring and visualization for end users. The scalability of the approach could be more thoroughly validated via large-scale experiments. A practical consequence is an implementation with multiple OrgManagers, which would need in turn to coordinate their actions. The system could exploit more heterogeneity by allowing the support of specialized hardware (e.g. General Purpose GPUs, coprocessors), or low-power systems on a chip. Finally, because of the very promising scalability tests, the ParaMASK systems could be applied to other problems, acting as a universal MAS framework.

Acknowledgments: This work was completed with the support of the INTER/CNRS/11/03/ Green@Cloud Project. M. Guzek acknowledges the support of the National Research Fund of Luxembourg (FNR) and Tri-ICT, with the AFR contract no. 1315254. Experiments presented in this paper were carried out using the Grid’5000 experimental testbed, being developed under the INRIA ALADDIN development action with support from CNRS, RENATER and several Universities as well as other funding bodies (see http://www.grid5000.fr).

REFERENCES