Application Topology Definition and Tasks Mapping for Efficient Use of Heterogeneous Resources

Kods Trabelsi, Loïc Cudennec, Rihab Bennour

Application Topology Definition and Tasks Mapping for Efficient Use of Heterogeneous Resources

Kods Trabelsi, Loïc Cudennec, and Rihab Bennour

CEA, LIST
Computing and Design Environment Laboratory
F-91191 Gif-sur-Yvette, France
{kods.trabelsi, loic.cudennec, rihab.bennour}@cea.fr

Abstract. Nowadays, high-performance computing (HPC) not only faces challenges to reach computing performance, it also has to take in consideration the energy consumption. In this context, heterogeneous architectures are expected to tackle this challenge by proposing a mix of HPC and low-power nodes. There is a significant research effort to define methods for exploiting such computing platforms and find a trade-off between computing performance and energy consumption. To this purpose, the topology of the application and the mapping of tasks onto physical resources are of major importance. In this paper we propose an iterative approach based on the exploration of logical topologies and mappings. These solutions are executed onto the heterogeneous platform and evaluated. Based on these results a Pareto front is built, allowing users to select the most relevant configurations of the application according to the current goals and constraints. Experiments have been conducted on a heterogeneous micro-server using a video processing application running on top of a software-distributed shared memory and deployed over a mix of Intel i7 and Arm Cortex A15 processors. Results show that some counterintuitive solutions found by the exploration approach perform better than classical configurations.

Keywords: heterogeneous architectures · tasks mapping · solutions space exploration.

1 Introduction

Numerical simulation requires the efficient use of computing resources and leads to a growing demand in performance to provide more accurate results or to decrease the computing time. High-performance computing centers usually scale up to offer more computing power and, despite significant R&D efforts on the hardware side to limit the energy consumption, the power efficiency has become an important constraint in the design and management of such centers. Heterogeneous computing platforms combines high-performance and low-power computation nodes and are not only intended to be deployed in HPC but also in embedded HPC as in autonomous vehicles, IoT and smart manufacturing. The efficient use of heterogeneous platforms is a complex task since it is the result of several intricated sub-problems including application sizing, task mapping and scheduling. The design of high-level tools to help users and platform managers has become an important field of research in the heterogeneous computing community.

One of the issues in such architectures is the deployment of distributed applications in respect of performance constraints and goals. Distributed applications can usually be configured prior to deployment by setting the number of tasks and the placement of tasks onto computing resources. The combination of application sizing and task mapping provides different computing performance (eg. computing time, latency, bandwidth..) and energy consumption (eg. instantaneous power in W or total consumption in kJ) for the same functionality. In this work we propose an exploratory approach to automatically evaluate different application configurations and relieves the user from manually configuring the deployment of applications. Configurations are evaluated on the heterogeneous platform when needed and a Pareto front is built according to constraints and objectives of interest. This representation is given as a decision tool for the user, from which it is possible to pick a particular configuration that meets at best the current requirements.

As a motivating example, we consider applications running on top of a software-distributed shared memory (S-DSM) and deployed over a heterogeneous computing platform. S-DSM is basically a runtime that aggregates distributed physical memories into a shared logical space. It is inherently a distributed system with different roles: S-DSM servers for managing data and metadata and application clients to run the user code. These roles can be instantiated, organized into topologies and mapped onto physical resources, hence leading to performance and energy consumption trade-off when deploying onto the heterogeneous platform. We use a video processing application on top of the S-DSM and evaluate the exploratory approach to build a Pareto front using an heterogeneous Christmann RECS|Box Antares Microserver as for testbed.

The paper is organized as follows: section 2 describes the S-DSM model and deployment context. Section 3 introduces the S-DSM topology definition problem, the resolution approach and the results of the deployment on heterogeneous architectures. Section 4 defines the mapping problem, the developed strategies and the deployment on heterogeneous architectures results. Section 5 gives some references on previous works. Finally, section 6 concludes this paper and gives new perspectives.

2 Topologies and Mappings for DSM

Shared memory is a convenient programming model in which a set of tasks can concurrently allocate and access data in a global memory space. While the implementation is quite straightforward in a single memory system, shared memory requires a tight design to be deployed on a complex architecture with physically distributed memories.

2.1 Distributed Shared Memory

The distributed shared memory (DSM) provides such a completely hardware-independent layer, at the price of hiding complexity into the runtime. The runtime is in charge of transparently managing local and remote data while minimizing the processing and communication costs. DSM have been studied since the late eighties with systems such as Ivy [9] and later adapted to new computation contexts such as clusters [1], grids [2] and many-core processors [11]. There is a price for offering hardware abstraction and code portability: most of DSM systems come with a significant overhead compared to distributed applications that use explicit communications. The contribution proposed in this paper, while based on a generic approach, is applied to the DSM context and aims at finding efficient configurations for the deployment of distributed shared memory applications.

2.2 Topology for DSM

In this work [3], a Software-DSM (S-DSM) is proposed to federate memories over heterogeneous architectures. The system can be seen as a regular distributed application with state machines to implement data coherence protocols. The S-DSM is organized as a semi-structured super-peer network as presented in Figure 1. A set of clients are connected to a peer-to-peer network of servers, mixing both client-server and peer-to-peer topology types. Servers are in charge of the shared data and metadata management while clients stand as the interface between the application user code and the S-DSM API. Building constraints for topologies include: (1) a minimal topology is made of one server, (2) there is a fully connected graph between servers, (3) each client is connected to one and only one server and (4) connections are not allowed between clients.

2.3 Application Model and Description

Applications running on this S-DSM are defined as a set of roles. Roles can be instantiated into clients using a given implementation. For each role, the application description defines the following constraints: the minimum and maximum numbers of instances (clients) and the available implementations. A description example is given in Figure 2. This application requires one client to decode the input video stream, at least one client to process the stream and one client to encode the output. From this description it is possible to build different functionally-equivalent S-DSM topologies by setting the number of S-DSM servers, the number of processing clients and the way it is connected.

In this paper we consider a video processing application as presented in Figure 2. Video frames are decoded by the input role, assigned to one of the

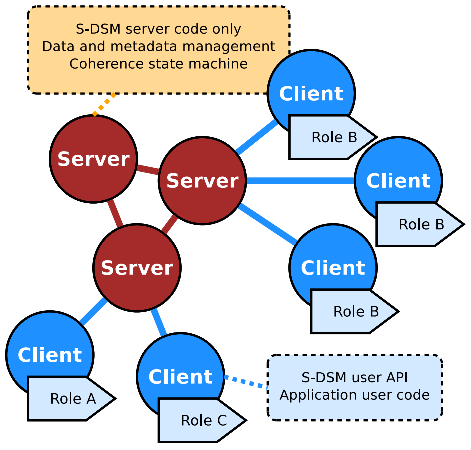


Fig. 1: S-DSM semi-structured super-peer topology.

ROLE	MIN	MAX	IMPLEM
sdsm_server	1	∞	C, Pthread
video_input	1	1	OpenCV
video_process	1	∞	C, Pthread, OpenMP
video_output	1	1	OpenCV

Fig. 2: Video processing application description.

process role using an eager scheduling strategy and encoded by the output role. Frames are stored into shared buffers within the Distributed Shared Memory: one input buffer and one output buffer for each processing task. The processing task applies an edge detection algorithm (a convolution using a 3x3 kernel) and a line detection algorithm (a Hough transform implemented in double precision). For technical reasons, the input and output roles are implemented using the OpenCV library and always deployed on the Core i7 processors. The processing role can be instantiated in C, Pthread (4 threads) and OpenMP. The input is a 1-minute video file, with a total of 1730 frames and a resolution of 1280x720 pixels.

2.4 Heterogeneous Platform

Previous results in [3] have shown that building relevant topologies and mappings are of major importance when it comes to efficiently use computing resources. This is particularly true when considering heterogeneous resources. The platform used in [3] is close to the one that is used in this work. It is a Christmann RECS|Box micro-server with heterogeneous processing elements. This server is a 1U rack composed by a backplane that provides power supply and networking capabilities to a set of slots. Each slot can host a computing node such as high-performance processors, low-power processors, accelerators, GPGPUs and FP-

GAs. Processing elements are different in terms of computing power and energy consumption. In this configuration, and for our own applications, a Cortex A15 is nearly 4 times slower than a Core i7. Instantaneous power consumption is around 7W for A15 and 30W for i7 at full load. The network also presents disparities in terms of bandwidth and latency due to different mediums and network architectures. For example, the Ethernet over USB is common for embedded devices and the Cortex A15 processors that rely on this interface are loosely connected compared to the i7 processors. In this work, we limited resources to a subset of the computations nodes available on the RECS|Box Antares micro-server. Figure 3 gives details of the nodes used in this paper as well as the number of processing units and supported implementations.

Node	PU	IMPLEM
Intel I7	8	C, OpenMP, Pthread, OpenCV
Cortex A15	2	C, OpenMP, Pthread
Cortex A15	2	C, OpenMP, Pthread
Cortex A15	2	C, OpenMP, Pthread
Cortex A15	2	C, OpenMP, Pthread

Fig. 3: Computing nodes used in the experiments.

Consequences on Heterogeneous Resources. In Figure 4, processing times are given for an image processing application with different topologies and mappings. S-DSM servers are represented with green cylinders, image input and output clients with orange arrows and processing clients with blue arrows. For each client, an horizontal segment indicates to which server it is connected. Topologies and mappings lead to very different results, even when comparing similar configurations. Even with a tight knowledge of the application, the S-DSM runtime and the hardware, it is difficult to find efficient hand-made solutions based on the sole expression of intuition.

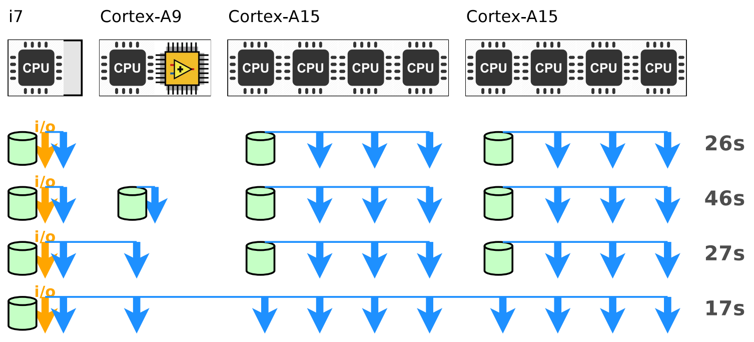


Fig. 4: S-DSM performance results for different topologies.

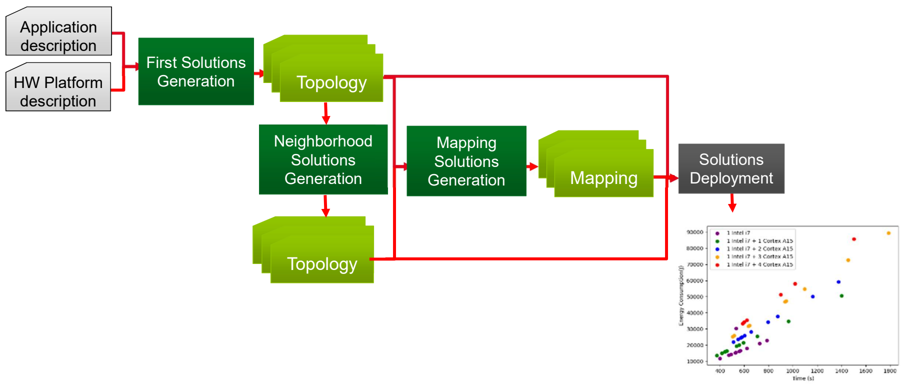

Application description
HW Platform description
First Solutions Generation
Topology
Neighborhood Solutions Generation
Topology
Mapping Solutions Generation
Mapping
Solutions Deployment
1 Intel i7
1 Intel i7 + 1 Cortex A15
1 Intel i7 + 2 Cortex A15
1 Intel i7 + 3 Cortex A15
1 Intel i7 + 4 Cortex A15
Energy Consumption(J)
Time (s)

Fig. 5: Automatic design space exploration flow for efficient use of heterogeneous resources

In this context, an automatic design space exploration should be used to build application configurations. This is particularly important when considering adversarial metrics such as computing time and power consumption, while targeting heterogeneous resources. In that case, the exploration system should propose different trade-off solutions and help the user to take an appropriate decision. Furthermore, this has to be done quite transparently for the application, without any code modification such as *pragmas*.

In this work, we propose to automatically explore application configurations and mappings over heterogeneous resources. Figure 5 illustrates the proposed design space exploration flow. Topologies and mappings are generated from given application and hardware descriptions. Solutions are evaluated by deploying and monitoring the application on the targeted computing hardware platform. The results are then used to build a Pareto front allowing a user to select relevant configurations corresponding to his objectives and constraints.

3 Space exploration for topologies

Generating all possible configurations is not acceptable because it is a time consuming operation. However, in order to generate a relevant set of topologies, we have been inspired by approximate methods. This class of methods, called also heuristics, gives a trade-off between the computation time and the quality of solutions. Neighborhood search (local search), is a meta-heuristic method for solving computationally hard optimization problems. This type of algorithms is widely applied to various computational problems in several fields. This algorithms move from a solution to another in the space of candidate solutions (the search space) by applying local changes, until a solution deemed as optimal is found or a time bound is elapsed.

In this work, we instrument a multi-starts local search to investigate the search space. This approach involves starting from several solutions and performing as much parallel local searches in order to generate a set of new solutions. The key point of this approach is the generation of starting solutions. The starting solutions have to be sufficiently scattered in the search space to explore it at best.

We chose to implement this approach among others because of its simplicity. Moreover, it can be a good starting point for building more sophisticated approaches such as simulated annealing algorithm. This method involves two steps. The first step is the generation of initial solutions. The second one corresponds to the neighborhood exploration. Initial solutions and those generated using local search were deployed on the RECS|Box Antares micro-server for evaluating their execution times and their energetic costs. For the rest of the document, "solution" designates a topology.

3.1 Initial solutions generation

Initial solutions are built using a greedy approach. To build a solution we have to set the number of servers, the number of tasks for each role and the connections between servers and clients. To obtain various starting solutions, we varied the number of servers and the number of tasks for each role. The server number has been varied from one to the number of nodes available on the targeted computing platform (5 in our example), to obtain a set of partial solutions. Then for each partial solution, we varied the number of the processing role instances to obtain a new set of partial solutions. Once the number of servers and the number of tasks for each role are set, a function is in charge of randomly establishing connections between servers and tasks preserving the uniqueness constraint. This last step leads to the completion of all the solutions. The generated topologies are not necessarily valid solutions: at this stage we can not guarantee that each topology will have at least one possible mapping on the target computing platform.

Deployment of initial solutions on heterogeneous platform.

Figure 3 gives details on the resources used while the application is described on figure 2. Figure 6 shows the performance and energetic costs of initial solutions. First, the energy consumption increases according to the number of nodes. Second, the execution time does not necessarily decrease if we use more computing nodes, hence falling beyond speedup. Figure 7 gives details of the solutions used to build the Pareto front (Solutions A and B). Solution B takes less time to complete its execution thanks to the extra processing task and the load distribution between the two S-DSM servers. However this has an additional cost for energy consumption and solution B is not as efficient as solution A when comparing frames per second per KJ (FPS/KJ).

Solution A' (Figure 7) is obtained by adding to solution A a processing instance mapped on the Intel processor. Adding this processing task should intuitively decrease the application execution time, but that is not what happens. The Open MPI runtime implementation is intended to be deployed on HPC

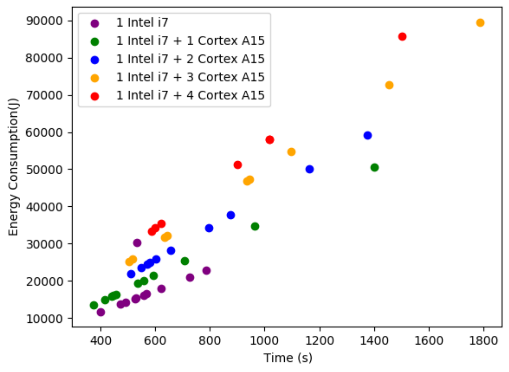


Fig. 6: Initial topologies

Solutions	Nodes	nb_s	nb tasks	Time (s)	FPS	KJ	FPS/KJ
solution A	1 Intel	1	4	398	4.3	11.5	0.38
solution B	1 Intel + 1 Cortex A15	2	5	375	4.6	13.5	0.34
solution A’	1 Intel	1	5	375	4.6	13.5	0.34

Fig. 7: Solutions of the initial Pareto front (A and B) and solution A’ obtained by adding a processing task onto the Intel node. nb_s stands for the number of S-DSM servers. Frames per second (FPS). Energy is given in KJ.

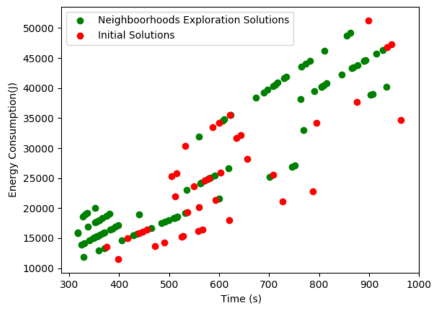
systems. In order to be as responsive as possible, the receive function busy-waits and continuously polls for new messages, the latter being CPU-demanding. When deploying several MPI processes on the same CPU, the local OS scheduler has to cope with legitimate MPI processes running user code and falsely busy MPI processes waiting for messages, the first being slowed down by the second.

3.2 Neighborhoods description

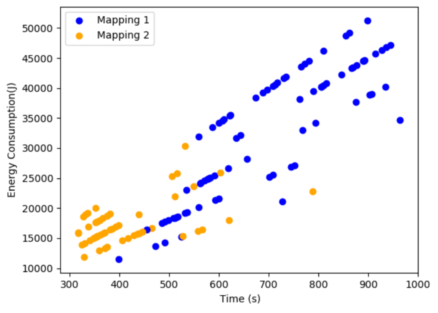
A neighborhood is obtained by applying a given number of modifications such as sub-topology swapping to the original solution. This generates several new solutions. In our context, several modifications are used such as adding or deleting S-DSM servers, adding or deleting a role instance (in respect with the min and max constraints), deleting a connection between a task and a server and establishing a connection with a new server. A first neighborhood is obtained by moving a client from the clients’s list of a server to a clients’s list of another server. The second neighborhood is obtained by merging all servers clients’s lists into a single list, shuffling the clients, splitting the list according to the initial number of servers, and finally randomly assigning new lists to servers.

Deployment of local solutions on the heterogeneous platform.

In Figure 8a the performance and the energy consumption of the initial solutions are compared with the solutions generated by the local search. For these



(a) Initial vs. neighborhoods topologies



(b) 'mapping 1' vs. 'mapping 2' strategies

Fig. 8: Topologies and mapping solutions spaces exploration.

experiments we have discarded solutions that overrun 16 minutes of execution time. This figure reveals that the local search allowed to conquer empty spaces in which solutions are of better quality in terms of both energy cost and execution time, compared to those generated initially. The best solution found using the neighborhood exploration regarding the performance metric is 16% better than the best solution of the initial set. Figure 9 gives details about the solutions used to build the Pareto front with local search.

Solutions	Nodes	nb_s	Time (s)	FPS	KJ	FPS/KJ
solution C	1 Intel + 3 Cortex A15	4	316	5.4	15.8	0.35
solution D	1 Intel + 2 Cortex A15	3	324	5.3	13.9	0.38
solution E	1 Intel + 1 Cortex A15	2	328	5.3	11.8	0.45

Fig. 9: Solutions building the Pareto front using local search.

Solution C takes less time to complete its execution thanks to an efficient load distribution between 4 S-DSM servers. The Pareto front solutions have the following pattern: a server, the I/O clients and the processing tasks are mapped onto the Intel i7 node and additional servers are mapped on the Cortex A15 nodes. The more servers we have, the lower the processing time is. This rule stops being true for solutions having 5 and more servers. Increasing the number of S-DSM servers balances the load of access requests from the clients, and avoids the centralized bottleneck server issue. However, after reaching a given number, the benefit vanishes because of the increasing probability for a server to forward the request to another one, leading to additional communication delays (multi-hop). Using more Cortex A15 to manage shared data increases the energy consumption and solution C is not efficient considering FPS/KJ.

4 Mapping problem

In this section, we evaluate the impact of the mapping step on the execution time and energy consumption of the generated topologies. The mapping step consists in assigning servers and tasks instances to computing resources, taking into consideration the heterogeneous aspect of the platform and the available implementations (a role can provide different implementations, eg. pthread, OpenMP, OpenCL). A complete mathematical formulation of tasks mapping on heterogeneous system problem is available in [13]. In this work, two straightforward mapping strategies were developed for the experiments. The first strategy `mapping 1` attempts to co-localize the clients with their corresponding servers in order to benefit from data locality. The second mapping strategy `mapping 2` randomly assigns servers and clients to computing nodes. For both strategies we limit the exploration to one server per node at most. Figure 8b shows the impact of the two mapping strategies on performance and energy consumption. Blue dots in the Pareto indicates solutions with the `mapping 1` strategy while the yellow dots are for solutions with `mapping 2`. This figure reveals that the solutions coming from `mapping 2` are better in both execution time and energy consumption. Intuitively, collocating processing tasks together with their attached S-DSM servers sounds to be an efficient strategy to benefit from data locality. This does not appear to be an efficient strategy: processing tasks that are mapped onto Cortex A15 severely slow down the entire computation as this kind of processor is not suited for executing high performance tasks. Conversely, Cortex A15 are better used to host S-DSM servers only, as application helpers, which is quite counterintuitive at first given the poor network communication capabilities. In conclusion, as applications and heterogeneous computing platforms become more complex, the automatic exploration of configurations appear to be a steady approach towards an efficient use of resources.

5 Related Works

The idea of using the most suitable hardware resource for a specific application is not new and has been explored in previous works. However, the two different subjects of exploring the application topology and the task mapping are usually addressed separately. Some works have targeted regular MapReduce-based applications. For instance, the TARA [8] system uses a description of the application to allocate the resources. However, this work is tailored for a very specific class of applications and does not address hardware details. In [6] the authors introduce a new topology-aware resource selection algorithm to determine the best choice among the available processing units of the platform, based on their position within the network and taking into account the applications communication matrix. However this work does not study the methodology impact on energy consumption. In mARGOt [5] the authors propose a design space exploration method leading to the building of a Pareto front. Their method requires code transformations and code variants called *software knobs*. In this work, there is no need to modify the application. The tasks mapping problem has been extensively studied in the last decade and numerous methods have been reported in the literature under various assumptions and objectives. In [4] the authors aim at finding a trade-off between energy consumption and execution time using genetic algorithm heuristic to build a Pareto front. In [12] the authors resolve task assignment problem on heterogeneous platform attempting to minimize the total execution time and the communication cost. In [10] an iterative algorithm is proposed for the mapping problem on heterogeneous computing platforms with load balancing as a goal. In [13] the authors model both task scheduling and mapping in a heterogeneous system as a bi-objective optimization problem between energy consumption and system performance.

Previous works have not established a relationship between the application sizing, the application topology building and the task mapping problems, and their impact on both performance and energy consumption. In our work we propose to combine these problems and explore different configurations without relying on user hints, code modifications, pragmas or a specific dataflow programming model. We evaluate the solutions on the heterogeneous platform and build a Pareto front allowing users to select the most relevant configuration as in a decision system. In the early 2000, a definition of autonomic computing has been introduced by IBM [7] including self-managing attributes. This work contributes to the self-configuring and self-optimizing attributes.

6 Conclusion

The new great challenge for today's high-performance computing stands in the energy savings. Innovative heterogeneous computing platforms such as the Christmann RECS|Box offers several computing units with different specifications in order to offer to the users the possibility to optimize the execution of their applications in terms of performance and energy consumption. However, the efficient use of these platforms remains an open topic for both the academic and the industrial worlds. In this work we have presented some experiments using a video processing application on heterogeneous computing machine to analyze the impact of the S-DSM topology definition and mapping steps on the execution time and energetic cost. To achieve this, we have proposed a local search method to generate several topologies that have been evaluated in order to build a Pareto front. This Pareto allows users to choose the solution that matches at best their current goals and constraints in terms of execution time and energy consumption. Thanks to this approach we were able to find counterintuitive solutions that perform surprisingly well for both performance and energy. Future work will include a model for energy and performance estimation to evaluate topology and mapping solutions at a higher level and avoid as much as possible the deployment of the generated solutions onto the hardware.

Acknowledgement

This work received support from the H2020-ICT-2015 European Project M2DC under Grant Agreement number 688201.

References

1. Amza, C., Cox, A.L., Dwarkadas, S., Keleher, P., Lu, H., Rajamony, R., Yu, W., Zwaenepoel, W.: TreadMarks: Shared memory computing on networks of workstations. IEEE Computer **29**(2), 18–28 (Feb 1996)
2. Antoniu, G., Bougé, L., Jan, M.: JuxMem: an adaptive supportive platform for data-sharing on the grid. Scalable Computing: Practice and Experience (SCPE) **6**(3), 45–55 (Nov 2005)
3. Cudennec, L.: Software-distributed shared memory over heterogeneous micro-server architecture. In: Euro-Par 2017: Parallel Processing Workshops. pp. 366–377. Springer International Publishing (2018)
4. Friese, R., Khemka, B., Maciejewski, A.A., Siegel, H.J., Koenig, G.A., Powers, S., Hilton, M., Rambharos, J., Okonski, G., Poole, S.W.: An analysis framework for investigating the trade-offs between system performance and energy consumption in a heterogeneous computing environment (2013)
5. Gadioli, D., Palermo, G., Silvano, C.: Application autotuning to support runtime adaptivity in multicore architectures. In: 2015 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS). pp. 173–180 (July 2015)
6. Georgiou, Y., Jeannot, E., Mercier, G., Villiermet, A.: Topology-aware resource management for hpc applications. In: Proceedings of the 18th International Conference on Distributed Computing and Networking. pp. 17:1–17:10. ICDCN '17, ACM, New York, NY, USA (2017). https://doi.org/10.1145/3007748.3007768
7. Horn, P.: Autonomic computing: Ibm's perspective on the state of information technology **2007** (10 2001)
8. Lee, G., Tolia, N., Ranganathan, P., Katz, R.H.: Topology-aware resource allocation for data-intensive workloads. SIGCOMM Comput. Commun. Rev. **41**(1), 120–124 (Jan 2011). https://doi.org/10.1145/1925861.1925881
9. Li, K.: IVY: a shared virtual memory system for parallel computing. In: Proc. 1988 Intl. Conf. on Parallel Processing. pp. 94–101. University Park, PA, USA (Aug 1988)
10. Renard, H., Vivien, F., Legrand, A., Robert, Y.: Mapping and load-balancing iterative computations. IEEE Transactions on Parallel and Distributed Systems **15**, 546–558 (06 2004). https://doi.org/10.1109/TPDS.2004.10
11. Ross, J.A., Richie, D.A.: Implementing openshmem for the adapteva epiphany risc array processor. Procedia Computer Science **80**, 2353 – 2356 (2016), international Conference on Computational Science 2016, ICCS 2016, 6-8 June 2016, San Diego, California, USA
12. Ucar, B., Aykanat, C., Kaya, K., Ikinci, M.: Task assignment in heterogeneous computing systems. Journal of Parallel and Distributed Computing **66**(1), 32 – 46 (2006). https://doi.org/https://doi.org/10.1016/j.jpdc.2005.06.014
13. Zaourar, L., Aba, M.A., Briand, D., Philippe, J.M.: Modeling of applications and hardware to explore task mapping and scheduling strategies on a heterogeneous micro-server system. IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW) (2017). https://doi.org/http://doi.ieeecomputersociety.org/10.1109/IPDPSW.2017.123