On compact mappings for multicore systems

Andrés Goens[0000-0002-0409-1363], Christian Menard[0000-0002-7134-8384], and Jeronimo Castrillon[0000-0002-5007-445X]

TU Dresden, Center for Advancing Electronics Dresden (cfaed),
Chair for Compiler Construction
Dresden, Germany
{andres.goens,christian.menard,jeronimo.castrillon}@tu-dresden.de

Abstract. Application mapping is key for efficient multicore processing, i.e., selecting which resources to allocate to a given application, like computation to cores. Mapping is increasingly difficult in multi-application scenarios, where resource contention might degrade the performance of an application. In order to solve this, a promising avenue is to consider "compact" mappings, those which require a small and (geometrically) compact area within the chip. Compact mappings should decrease contention between applications by providing regional isolation and allowing multiple applications to be mapped simply. Previous work has shown that compact mappings can significantly outperform mappings obtained with a random strategy. In this paper we investigate the promise of compact mappings by running extensive simulations on Noxim, a cycle-accurate network-on-chip simulator. Results show the promises of compact mappings do not hold up in practice. When comparing to mappings selected with a heuristic better than simply choosing cores at random, our experiments do not indicate significant advantages from compact mappings. We outline possible reasons for this.

1 Introduction

As multicores become more ubiquitous, Network-on-Chip (NoC) technologies play a role of continuously increasing importance in modern computing systems. Programming these systems is a difficult task, and a central component of it is that of finding a *mapping*, i.e., an allocation of computation and the flow of data on the system's resources. The mapping problem has been extensively studied [15], in particular in static single-application scenarios [5,18,3,13,12]. For multi-application systems, researchers commonly study so-called hybrid approaches, where (partial) single application mappings are combined dynamically at run-time [15,2]. Besides coordinating execution and communication within a single application, the execution of multiple-applications requires that contention between applications is minimized, which is achieved through a (partial) remapping at run-time in multi-application hybrid approaches [19,14,6] or isolation for security concerns [20].

Systems featuring a NoC are typically arranged in a regular fashion, which falls within only a handful of topology types, e.g. meshes, stars, rings or tori.

Out of these, regular $n \times n$ mesh topologies, like the one depicted in Figure 1 for $n = 4$, are by far the most common in the literature. These topologies allow designers to think in geometrical terms of mappings, which is what makes the idea of finding *compact* mappings for multi-application scenarios an evident one.

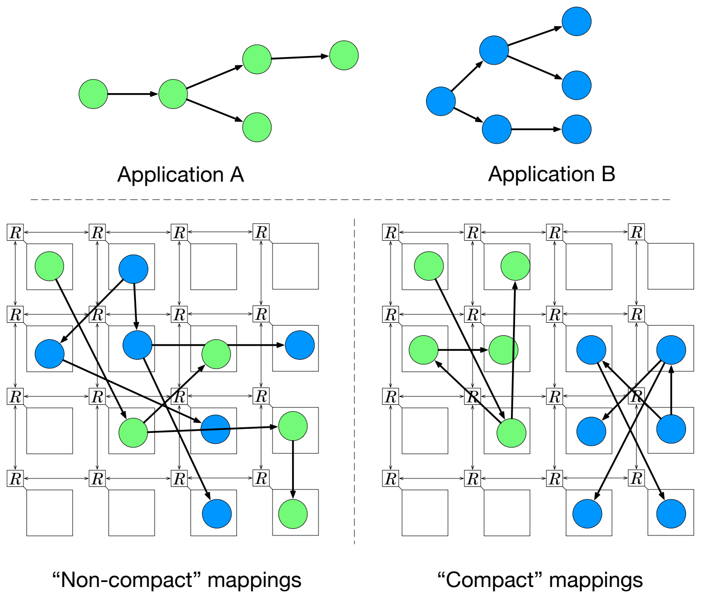


Fig. 1. Comparing compact and non-compact mappings.

Figure 1 illustrates the idea behind compact mappings. It depicts the task graphs of two applications, which are colored in green and blue respectively. The nodes of these task graphs represent computational tasks, and their directed edges the flow of data from one task to another. The figure shows two different multi-application mappings of both applications to a 4×4 mesh network-on-chip topology. Both mappings use exactly the same processing elements (PEs) of the architecture. In fact, if we ignore contention, both mappings should have the exact same behavior: a careful inspection reveals that the distances —in terms of number of hops— between any two communicating tasks in any of the applications are all exactly the same in both mappings. However, it is intuitively obvious that the mapping on the right of the figure is preferable: by being more *compact*, the mapping on the right will isolate communication within both applications. This helps to avoid contention and provides increased security. Additionally, it is intuitively simpler to combine single-application mappings like those on the right, since they are less *fragmented*.

In this paper we explore how the outlined intuition holds up to experimental scrutiny. After a brief formal background (Section 2) and discussion of related work (Section 3), we explore an experiment designed to assess the strategy of mapping applications to compact regions (Section 4). We discuss (Section5) how the results of this experiment do not show significant advantages from compact mappings in practice, and comment on the conclusions (Section 6) we can reach from this work.

2 Mapping Tasks to MPSoCs

The problem of mapping can be defined at several levels of granularity. In this paper we focus on the mapping of computational tasks to cores. Let $G = (T, E)$ be the *task graph*, a directed graph representing the application. The nodes $t \in T$ of the task graph represent computational tasks that have to be executed on a core. In this paper we focus on a homogeneous setting, as is the most common setting in multicores featuring regular mesh NoC technologies. For homogeneous cores, we assume all tasks to require a fixed amount of execution time, independent of the core executing them. The focus of a mapping, and what ultimately guides its performance, is the communication between tasks. An edge in the task graph $e = (t_1, t_2) \in E$ represents a (logical) data dependency between tasks. Commonly, tasks graphs are labeled with the computational costs in the nodes and with the communication requirements in the edges. Depending on the status of the communication subsystem and the mapping, an application can incur in different communication costs for the same data. Thus, we do not encode these costs in the problem formulation directly; these have to be estimated, simulated, or better, measured.

A (task) mapping refers to a mathematical mapping $m : T \to \{\mathrm{PE}_{i,j} \mid i, j = 1 \ldots n\}$ from the set of tasks to the set of processing elements (PEs), in this case indexed by two integers representing their position on the mesh. This mapping defines the allocation of tasks to PEs. The mapping problem is that of finding a mapping $m : T \to \{\mathrm{PE}_{i,j} \mid i, j = 1 \ldots n\}$ that fulfills a particular goal, like minimizing execution time, energy consumption or NoC communication. In this paper we will define a mapping as minimizing the execution time, which we will measure by the average latency (network delay) as a proxy for execution time, since the cores are homogeneous. While a mapping m does not need to be injective, i.e., two tasks could be mapped to the same core $m(t_1) = m(t_2)$ for $t_1 \neq t_2$, we will consider mappings where this is not the case. This is common in many NoC-based multicore systems where the cores are smaller and scheduling and computational contention are costly.

3 Related Work

The work focusing on hybrid mapping [19,14,6] usually focuses on transforming single-application mappings or finding mappings at run-time with a particular set of constraints. To the best of our knowledge, none of these approaches directly concern themselves with compact mappings. Other work explicitly investigates the composability of applications in such scenarios, but focuses on explicit hardware support for this, not on leveraging the NoC topology [9,8]. While all of this work proposes sophisticated approaches, far beyond what is presented in this paper, the issue of compact mappings by itself has not been investigated by any of the these approaches. The intention of this paper is not to propose good heuristics for hybrid or composable mapping, but rather, to assess to what extent compactness is useful for this end.

Unlike all the other references focusing on hybrid and composable approaches, authors of [21] do consider compact mappings in multi-application scenarios for NoC architectures explicitly. The rigor of the contribution is unclear, as they compare their strategy to mapping at random. To avoid this, we compare to a non-compact mapping strategy that strives to find good mappings nevertheless. More importantly though, while the work of [21] is focused on proposing a heuristic, we focus punctually on the aspect of compactness in this paper.

4 Evaluation

In this section we evaluate the strategy of defining compact mappings for multi-application scenarios. For this we define an algorithm to find mappings with good communication properties and show how we can use it to generate compact and non-compact mappings.

4.1 Mapping Algorithm

We designed and implemented a mapping heuristic to find mappings with good communication properties. Since we are focusing on homogeneous multicore systems with a NoC in a regular mesh topology, our algorithm is designed for such a regular architecture. However, it would work for any mapping representation which yields a metric space on the architecture and mappings [7].

Our mapping heuristic is described in Algorithm 1. It starts with any node in the application that can be considered a root, i.e., such that there is a path from it to every node in the application. We assume the application graph is (weakly) connected and such a node exists. The algorithm assigns an unused core at random to this node and then iterates through the application graph in a breadth first fashion to assign cores such that the distance from a node to its predecessor is minimized in the mapping. This greedy algorithm does not ensure that the communication is minimized globally for the whole application, but it yields mappings with a total communication close to the minimum. We use it to produce compact mappings as well, by marking every core as occupied, except for an $m \times m'$ rectangle such that $mm' > |V|$, and we choose $\{m, m'\} \subseteq \{\sqrt{|V|}, \sqrt{|V|} + 1\}$ minimal with this property. If we leave out the additional rectangle constraint, we get low-communication mappings that are not necessarily compact.

4.2 Experimental Setup

For our evaluation, we use a slightly modified version of the NoC simulator Noxim [4]. Noxim is a SystemC based simulator that is capable of modelling a

Algorithm 1 A greedy heuristic for low-communication mapping

```
input: A connected application graph $\Gamma = (V, E)$, the size of the mesh $n$, a set of
    occupied cores $X \subseteq \{1, \ldots, n\} \times \{1 \ldots, n\} =: M$
output: A mapping $m : V \to M$
 1: CurNode $\leftarrow$ RandomFrom($M \setminus X$)
 2: $v_0 \leftarrow$ Root($\Gamma$)
 3: mapping $\leftarrow$ ($v_0 \mapsto$ CurNode)
 4: $X \leftarrow X \cup \{$CurNode$\}$
 5: for $e = (n_1, n_2) \in$ BreadthFirstEdgeSearch($\Gamma$) do
 6:     CurNode $\leftarrow$ mapping($n_1$)
 7:     $d \leftarrow \min_{d=1\ldots n}\{a \in M \setminus X \mid |a -$ CurNode$| \leq d\} \neq \emptyset$
 8:     $q \leftarrow$ RandomFrom($\{a \in M \setminus X \mid |a -$ CurNode$| \leq d\}$)
 9:     mapping($n_2$) $\leftarrow q$
10:     $X \leftarrow X \cup \{q\}$
    return mapping
```

wide range of NoC configurations and traffic patterns. We modified Noxim to obtain more detailed statistics that also include the variance of packet delays in order to evaluate the predictability of given mappings. In our configuration, Noxim simulates a mesh topology with xy-routing and worm-hole switching. This basic setup is comparable to current research platforms [1] as well as commercial products like Intel Xeon Phi [16], Intel Xeon Scalable Platform [17] and Mellanox Technologies TILE-Gx series [10,11]. We choose a network size of 10×10 nodes for our experiments.

The execution of dataflow application is simulated by providing a traffic table to Noxim. For each edge in the application task graph we add one entry to the table which specifies source node, target node and the packet injection rate. During the simulation, each node randomly injects packets at the given rate. By varying the packet injection rate of the channels, we can simulate low and high network loads. In the following, we report experiments using a fixed packet size of 32 flits. We did test several packet sizes, up to $2^{12} = 4096$ flits, but found the results to be comparable for all packet sizes tested.

We use 10 applications with 4 to 6 nodes, and generated 100 compact mappings, 100 non-compact mappings and 100 completely random mappings for these 10 applications. Algorithm 1 has some random choices, which results in different mappings for each of the applications. These 100 iterations together with the random choices are intended to account for the variance between mappings and scenarios.

4.3 Results

Figure 2 shows how the average network delay (latency) in the system varied across the different mapping strategies and corresponding mappings for three distinct values of the injection rate. We see how compact mappings are indeed significantly better than a random mapping, in all cases. However, a comparison of the compact and non-compact mappings shows no significant difference between

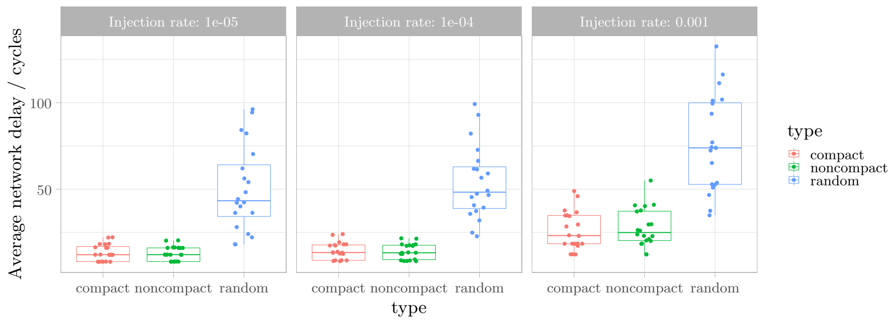


Fig. 2. Comparison of the average network delay of 100 mappings for different strategies and injection rates. Additional to the box and whiskers plot, the actual points are overlayed with random horizontal jitters for visibility.

both strategies, in terms of the average network delay. While non-compact mappings obtained with Algorithm 1 are also designed to reduce the average delay, one would expect more contention between applications to worsen the average delay. This does not seem to be the case in Figure 2.

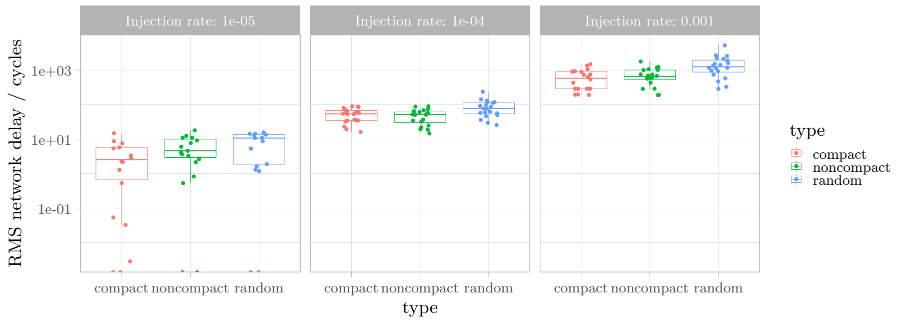
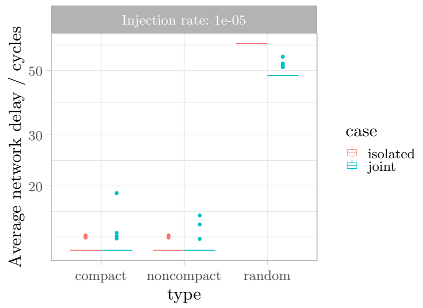
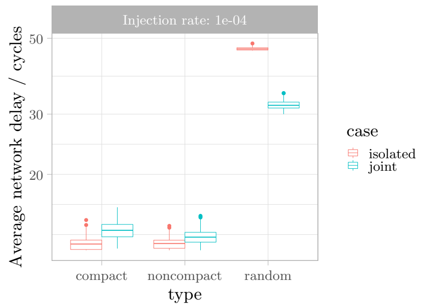


Fig. 3. Comparison of the root mean square network delay (log) of 100 mappings for different strategies and injection rates. Additional to the box and whiskers plot, the actual points are overlayed with random horizontal jitters for visibility.

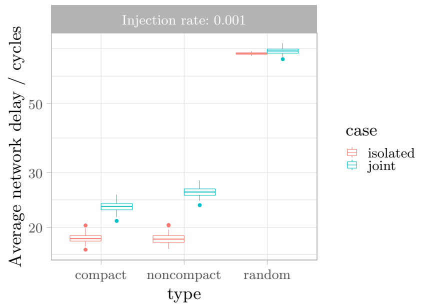
Predictability is of comparable importance to the average latency in many use cases. To measure how predictable a mapping behaves we used the slightly modified version of Noxim to obtain the root mean square of the network delay. The results for the 100 mappings can be seen in Figure 3 (note the logarithmic axis). We see how, in a few cases, compact mappings had a significantly higher predictability. However, in general, there does not seem to be statistically significant differences between compact and non-compact mappings.



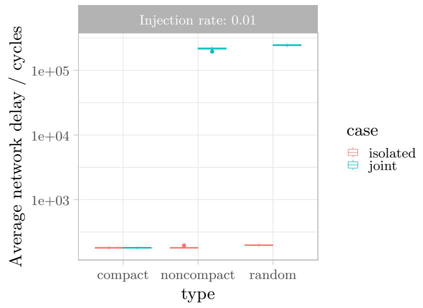
(a) Inejection rate of 0.00001



(b) Inejection rate of 0.0001



(c) Inejection rate of 0.001



(d) Inejection rate of 0.01

Fig. 4. Comparison of average network delay (log) for the first application, running in isolation vs a joint environment with 9 additional applications.

A better measure of predictability, however, is the comparison between the application running in isolation, i.e. alone in the system with no contention, versus the same application running in a multi-application scenario with possible contention. To asses this, we reproduced the previous setup, but instead of all ten applications, we executed only the first one in isolation. We compared it to the values for that same application in the multi-application scenario. In an ideal case, with no contention, these scenarios would result in the same latencies. However, as we can see in Figure 4, this is not the case. All three mapping types, compact, non-compact and random, suffered significant performance penalties in the multi-application scenario. Moreover, for the lower injection rates (figures 4a and 4b), the results seem to be again very similar between compact and non-compact mappings. However, with an injection rate of 0.001, which means that roughly every 1000 cycles a package is injected, a small albeit significant difference emerges, as seen in Figure 4c. This injection rate is already extremely fast: it means in a system clocked at 1 GHz, a package is sent every μs. It is unrealistic for many applications to actually fire at these rates. Nevertheless, to investigate it further we repeated the setup with an injection rate of 0.01, shown in Figure 4d. Here, indeed, compact mappings very significantly outperform non-compact ones when contention is present in a multi-application scenario. Such a high rate is however of no practical relevance.

5 Discussion

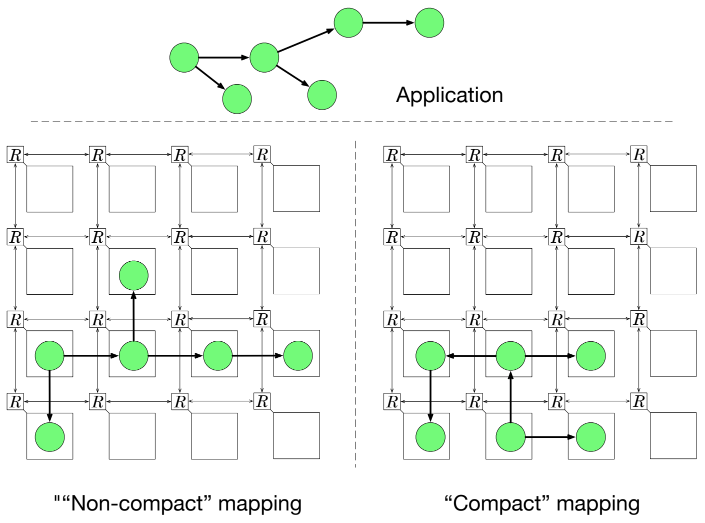


Fig. 5. Comparing a typical compact and non-compact mapping.

The results from Section 4 show basically no difference between compact and non-compact mappings for realistic scenarios. The advantages of compact mappings only seem apparent when the network traffic is so high that the average latency becomes hundreds of times higher than the computation time, with computation times between communication instances that would be smaller than a μs in most modern systems. We believe these scenarios are not of practical relevance. Thus, from the results we cannot conclude that compact mappings are better. However, we can neither conclude they are worse from our results. It is evident that ensuring compactness is an additional restriction, which makes algorithms more complex, but also reduces the space of possible mappings. Thus, the engineering investment to find and exploit compact mappings does not seem to be worth the effort.

These results defy intuition. To better understand why this is the case, we can return to the motivating example of Figure 1. If we simulate both scenarios in a setup like the one used for our evaluation, the non-compact variant turns out to be unequivocally better. If we look at the mappings more carefully, especially the compact ones, they generate a large amount of inter-application contention. In short, they simply are bad mappings. Compare this to the illustration from Figure 5. The two mappings shown here are typical results of mapping using

Algorithm 1. They are significantly better mappings than the mappings shown in Figure 1 (note that the application graph is slightly different). The two mappings from Figure 5 are also identical in terms of their topology: the distances, in terms of number of hops, between any two nodes, are identical in both mappings. However, these are also good mappings, since said distances are short (they are all 1 hop). The fact that their geometry is very different becomes irrelevant in practice, as shown from our evaluation. We believe the key takeaway from our evaluations to be that focusing on finding good mappings for an application seems to be more important than finding mappings that can be easily combined, if the latter come at the cost of the performance of the application in isolation. Inter-communication contention is significant for concurrent multi-processor applications.

5.1 Limitations

The evaluation of this paper has some clear limitations, which should be pointed out as part of this discussion.

The setup, including the applications evaluated are, in a sense, very homogeneous. All tasks have identical fire rates and the system is a regular mesh with homogeneous cores. While this type of architecture is indeed very common, it neglects heterogeneous multicores and more exotic topologies. It is possible to define and investigate compactness in these scenarios [7]. However, we do not see a compelling reason to believe these systems would behave differently.

Similarly, most applications have heterogeneity in the amounts of computation and data transfer that occur between tasks. This limitation is exacerbated by the fact that our simulation only considers traffic. This certainly limits the conclusive power of our experiments. However, we believe there is validity in the results even when accounting for these limitations. An application with heterogeneous amounts of data transfer will always have a *critical path* of nodes that effectively bounds the maximal achievable performance. Applications with a homogeneous behavior, like used in this paper, can be seen as an approximation of the behavior of this critical subgraph that neglects the minor effects from the non-critical parts of the application. Similarly, the mappings considered all had no computational scheduling, and thus at most one task per core. This is increasingly common in multi- and manycore systems, but it certainly restricts the generality of the investigation. Additionally, while there was a non-zero probability of cores overlapping, this happened at an average of 0.03 cores per scenario. It means that for a core executing a task in our simulations, chosen uniformly at random, there was a probability of less than 0.0006 that it was a core shared between two different tasks. Even in this case the simulation is not necessarily wrong, if the computational contention is not high enough. This makes the error introduced by this fact negligible.

Finally, we only considered performance as seen by latency in this work. Thus, any conclusions we might draw about compact mappings will only apply to performance-related issues. There might me other reasons for which compact mappings could prove to be a valuable idea, like privacy (see [20]), or when considering thermal effects.

6 Conclusion

In this paper we investigated the concept of compact mappings for multicore systems based on a mesh NoC. Counter-intuitively, our experiments did not show any advantages of compact mappings w.r.t. latency, both in terms of average and deviation, i.e. predictability. For extremely high and unrealistic traffic rates, this ceased to hold and compact mappings performed better than non-compact ones. However, the results seem to suggest that for any practical applications, focusing on finding a good mapping is more important than finding a mapping that can be easily composed in multi-application scenarios.

These conclusions are based on simulations of traffic and with a particular setup of applications. We see reasons to believe this would probably not change even if these limitations were removed. While we cannot rule out the usefulness of compact mappings, especially for goals other than system performance, we believe that there might be more worthwhile avenues of research in the field of multi-application mapping.

Acknowledgments

This work was supported in part by the Center for Advancing Electronics Dresden (cfaed) and the Studienstiftung des deutschen Volkes.

References

1. Balkind, J., McKeown, M., Fu, Y., Nguyen, T., Zhou, Y., Lavrov, A., Shahrad, M., Fuchs, A., Payne, S., Liang, X., Matl, M., and Wentzlaff, D. OpenPiton: An open source manycore research framework. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems* (New York, NY, USA, 2016), ASPLOS '16, ACM, pp. 217–232.
2. Castrillon, J., Leupers, R., and Ascheid, G. MAPS: Mapping concurrent dataflow applications to heterogeneous MPSoCs. *IEEE Transactions on Industrial Informatics 9*, 1 (Feb. 2013), 527–545.
3. Castrillon, J., Tretter, A., Leupers, R., and Ascheid, G. Communication-Aware Mapping of KPN Applications onto Heterogeneous MPSoCs. In *DAC '12: Proceedings of the 49th annual conference on Design automation* (2012).
4. Catania, V., Mineo, A., Monteleone, S., Palesi, M., and Patti, D. Cycle-accurate network on chip simulation with noxim. *ACM Trans. Model. Comput. Simul. 27*, 1 (Aug. 2016), 4:1–4:25.
5. Eker, J., Janneck, J. W., Lee, E. A., Liu, J., Liu, X., Ludvig, J., Neuendorffer, S., Sachs, S., and Xiong, Y. Taming heterogeneity-the ptolemy approach. *Proceedings of the IEEE 91*, 1 (2003), 127–144.

6. Goens, A., Khasanov, R., Hähnel, M., Smejkal, T., Härtig, H., and Castrillon, J. Tetris: a multi-application run-time system for predictable execution of static mappings. In *Proceedings of the 20th International Workshop on Software and Compilers for Embedded Systems (SCOPES'17)* (New York, NY, USA, June 2017), SCOPES '17, ACM, pp. 11–20.
7. Goens, A., Menard, C., and Castrillon, J. On the representation of mappings to multicores. In *Proceedings of the IEEE 12th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC-18)* (Vietnam National University, Hanoi, Vietnam, Sept. 2018), pp. 184–191.
8. Hansson, A., Goossens, K., Bekooij, M., and Huisken, J. Compsoc: A template for composable and predictable multi-processor system on chips. *ACM Transactions on Design Automation of Electronic Systems (TODAES) 14*, 1 (2009), 2.
9. Kumar, A., Mesman, B., Theelen, B., Corporaal, H., and Ha, Y. Analyzing composability of applications on mpsoc platforms. *Journal of Systems Architecture 54*, 3 (2008), 369–383.
10. Mellanox Technologies. TILE-Gx36 processor. `http://www.mellanox.com/related-docs/prod_multi_core/PB_TILE-Gx36.pdf`, 2015. [Online; accessed 2019-05-22].
11. Mellanox Technologies. TILE-Gx72 processor. `http://www.mellanox.com/related-docs/prod_multi_core/PB_TILE-Gx72.pdf`, 2015. [Online; accessed 2019-05-22].
12. Nikolov, H., Thompson, M., Stefanov, T., Pimentel, A., Polstra, S., Bose, R., Zissulescu, C., and Deprettere, E. Daedalus: toward composable multimedia mp-soc design. In *Proceedings of the 45th annual Design Automation Conference* (2008), ACM, pp. 574–579.
13. Pimentel, A. D., Erbas, C., and Polstra, S. A systematic approach to exploring embedded system architectures at multiple abstraction levels. *Computers, IEEE Transactions on 55*, 2 (2006), 99–112.
14. Quan, W., and Pimentel, A. D. A hybrid task mapping algorithm for heterogeneous mpsocs. *ACM Transactions on Embedded Computing Systems (TECS) 14*, 1 (2015), 14.
15. Singh, A. K., Shafique, M., Kumar, A., and Henkel, J. Mapping on multi/many-core systems: survey of current and emerging trends. In *Proceedings of the 50th Annual Design Automation Conference* (2013), ACM, p. 1.
16. Sodani, A., Gramunt, R., Corbal, J., Kim, H., Vinod, K., Chinthamani, S., Hutsell, S., Agarwal, R., and Liu, Y. Knights landing: Second-generation intel xeon phi product. *IEEE Micro 36*, 2 (Mar 2016), 34–46.
17. Tam, S. M., Muljono, H., Huang, M., Iyer, S., Royneogi, K., Satti, N., Qureshi, R., Chen, W., Wang, T., Hsieh, H., Vora, S., and Wang, E. Skylake-sp: A 14nm 28-core xeon processor. In *2018 IEEE International Solid - State Circuits Conference - (ISSCC)* (Feb 2018), pp. 34–36.
18. Thiele, L., Bacivarov, I., Haid, W., and Huang, K. Mapping applications to tiled multiprocessor embedded systems. In *Application of Concurrency to System Design, 2007. ACSD 2007. Seventh International Conference on* (2007), IEEE, pp. 29–40.
19. Weichslgartner, A., Gangadharan, D., Wildermann, S., Glass, M., and Teich, J. Daarm: Design-time application analysis and run-time mapping for predictable execution in many-core systems. In *Hardware/Software Codesign and System Synthesis (CODES+ ISSS), 2014 Int. Conf. on* (2014), IEEE, pp. 1–10.

20. Weichslgartner, A., Wildermann, S., Götzfried, J., Freiling, F., Glass, M., and Teich, J. Design-time/run-time mapping of security-critical applications in heterogeneous mpsocs. In *Proc. of the 19th Int. Workshop on Software and Compilers for Embedded Systems* (2016), ACM, pp. 153–162.
21. Yang, B., Guang, L., Xu, T. C., Säntti, T., and Plosila, J. Multi-application mapping algorithm for network-on-chip platforms. In *2010 IEEE 26-th Convention of Electrical and Electronics Engineers in Israel* (2010), IEEE, pp. 000540–000544.