A Hypergraph Model and Associated Optimization Strategies for Path Length-Driven Netlist Partitioning

Julien Rodriguez[1,2], François Galea[1], François Pellegrini[2], and Lilia Zaourar[1]

[1] Université Paris-Saclay, CEA List, F-91120, Palaiseau, France
name.surname@cea.fr
[2] Université de Bordeaux & INRIA, F-33405, Talence, France
francois.pellegrini@u-bordeaux.fr

Abstract. Prototyping large circuits on multi-FPGA platforms requires to partition the circuits into sub-circuits, each to be mapped in a given signle FPGA. While most existing partitioning algorithms focus on minimizing cut size, the main issue is not to map long paths across multiple FPGAs, as it may cause an increase in critical path length. To address this problem, we propose a new hypergraph model, for which we design algorithms for initial partitioning and partition refinement. We integrate these algorithm in a multilevel framework, combined with existing min-cut solvers, to tackle simultaneously both path length and cut size objectives. We observe a significant reduction in critical path degradation, by 12%-40%, at the cost of a moderate increase in cut size, compared to path-agnostic min-cut methods.

Keywords: hypergraph partitioning · critical path · VLSI · FPGA · fast prototyping.

1 Introduction

The typical hardware design flow comprises several steps, such as prototyping, verification, floor planning, placement and routing, possibly on very large logic circuits. The methods which perform these steps often take advantage of *netlist partitioning*, which enables divide-and-conquer approaches to separate circuits into parts of smaller size that are easier to handle. Thus, reduce as much as possible the work on the global circuit. This study focuses on netlist partitioning for prototyping on multi-FPGA platforms, for circuits that do not fit into a single FPGA. In this case, the circuit is divided into several parts, one for each FPGA. Valid partitions must respect capacity constraints on each FPGA and their interconnections. In addition, it should mitigate a possible increase in the signal propagation delay of the longest combinatorial path, known as the *critical path*. Indeed, in synchronous circuits, critical path length determines the maximum frequency at which the circuit may operate.mapping long paths across several FPGAs is likely to degrade the critical path. During the last 30 years, several hypergraph partitioning tools have been developed [4, 9, 14]. These tools use a multi-level framework, that consists of three phases: *coarsening*, *initial partitioning*,

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36024-4_50

and *refinement*. The coarsening phase recursively uses a clustering method to transform the hypergraph into smaller ones, that possess the same topological properties. Then, an initial partitioning is computed on the smallest coarsened hypergraph. Finally, for each coarsening level, the solution for the coarser level is prolonged to the finer level, and then refined using a local refinement algorithm. A survey on hypergraph partitioning has been recently issued [5]. Common metrics to measure the quality of hypergraph partitions are f_c, which sums the number of cut hyperedges, and f_λ, which sums the number of connected parts, minus one, for each hyperedge, and which represents the amount of information that needs to be transferred across parts. However, all of these tools, while commonly used in production chains, have not been designed to minimize the critical path length, as shown in [2]. This is why several authors proposed pre- and/or post-processing steps to reduce the degradation of cut paths [2,12]. However, the main objective addressed by these works is to reduce the cut along critical paths as much as possible, not considering the critical path's degradation resulting from subsequent mapping onto a non-fully connected target topology. This paper proposes a cost function to minimize path cost degradation during partitioning, based on a modeling of circuits as *red-black hypergraphs*. We also propose a weighting scheme to drive *min-cut* tools as well as a partitioning scheme comprising several algorithms. It includes an adaptation of a refinement algorithm that aims at preventing cuts across the longest paths while still trying to minimize cut size. The remainder of the paper is organized as follows. Section 2 presents the notations, definitions, and previous work on time-driven partitioning. Section 3 presents our approach, relying on a new initial partitioning algorithm and an extension of the well-known Fiduccia–Mattheyses (FM) algorithm [8] to minimize path cost. Simulation and results are presented in Section 4. We conclude in Section 5.

2 Preliminaries

2.1 Notations and Definitions

Oriented hypergraphs are a generalization of oriented graphs in which the notion of arc is extended to that of hyperarc, which can connect one or more source vertices to one or more sink vertices. In our context, we consider only hyperarcs that comprise a single source vertex. Let $\mathcal{H} \stackrel{\text{def}}{=} (\mathcal{V},\mathcal{A},\mathcal{W}_v,\mathcal{W}_a)$ be a directed hypergraph, defined by a set of vertices $\mathcal{V}$ and a set of hyperarcs $\mathcal{A}$, with a vertex weight function $\mathcal{W}_v : \mathcal{V} \rightarrow \mathbb{R}^+$ and a hyperarc weight function $\mathcal{W}_a : \mathcal{A} \rightarrow \mathbb{R}^+$. Every hyperarc $a \in \mathcal{A}$ is a subset of vertex set $\mathcal{V}$: $a \subseteq \mathcal{V}$. Let $s^+(a)$ be the source vertex set of hyperarc a, and $s^-(a)$ its sink (destination) vertex set. We consider here that each hyperarc has a single source, so $\forall a, |s^+(a)| = 1$. As hyperarcs connect vertices, let $\Gamma(v)$ be the set of neighbor vertices of vertex v, and $\Gamma^-(v) \subseteq \Gamma(v)$ and $\Gamma^+(v) \subseteq \Gamma(v)$ the sets of its inbound and outbound neighbors, respectively. In the model we propose, hypergraphs that model circuits are represented as sets of interconnected Directed Acyclic Hypergraphs (DAHs), associated with a red-black vertex coloring scheme. Red vertices correspond to I/O (Input/Output) ports and registers, and black vertices to combinatorial components. Let $\mathcal{V}^R \subset \mathcal{V}$ and $\mathcal{V}^B \subset \mathcal{V}$ be the red and black vertex subsets of $\mathcal{V}$, such that $\mathcal{V}^R \cap \mathcal{V}^B = \emptyset$

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36024-4_50

and $\mathcal{V}^R \cup \mathcal{V}^B = \mathcal{V}$. A hypergraph or sub-hypergraph $\mathcal{H}$ is a DAH iff its red vertices $v_R \in \mathcal{V}^R$ are either only sources or sinks (*i.e.*, $\Gamma^-(v_R) = \emptyset$ or $\Gamma^+(v_R) = \emptyset$), and no cycle path connects a vertex to itself. Using this definition, we can represent circuit hypergraphs as *red-black hypergraphs*, which are sets of DAHs that share their red vertices. Let $\mathbf{H}(\mathbf{V},\mathbf{A}) \stackrel{\text{def}}{=} \{\mathcal{H}_i, i \in \{1...n\}\}$ be a *red-black hypergraph*, such that every $\mathcal{H}_i$ is a DAH and an edge-induced sub-hypergraph of $\mathbf{H}$. In our model, the paths in $\mathbf{H}$ to consider when addressing the objective of minimizing *path-cost* degradation during partitioning are only the paths interconnecting red vertices, as these red-red paths represent register-to-register paths. Since only red vertices are shared between DAHs in $\mathbf{H}$, any red-red path only exists within a single DAH and can never span across several DAHs.

Let us define $\mathbf{P}$ as the set of red-red paths in $\mathbf{H}$. From these paths and a function $d_{max}(u, v)$ which computes the maximum distance between vertices u and v of some DAH $\mathcal{H}$, we can now define the longest path distance for $\mathcal{H}$ as: $d_{max}(\mathcal{H}) \stackrel{\text{def}}{=} \max(d_{max}(u, v) | u, v \in \mathcal{H})$ and, by extension, for $\mathbf{H}$, as: $d_{max}(\mathbf{H}) \stackrel{\text{def}}{=} \max(d_{max}(\mathcal{H}) | \mathcal{H} \in \mathbf{H})$.

A k-partition Π of $\mathbf{H}$ is a splitting of $\mathbf{V}$ into k vertex subsets $\pi_1..\pi_k$, called parts, such that: (i) all parts π_i, given a capacity bound C_i, respect the capacity constraint: $\sum_{v \in \pi_i} \mathcal{W}_v(v) \leq C_i$; (ii) all parts are pairwise disjoint: $\forall i \neq j, \pi_i \cap \pi_j = \emptyset$; (iii) the union of all parts is equal to $\mathbf{V}$: $\bigcup_i \pi_i = \mathbf{V}$. For a given partition Π of $\mathbf{H}$, the connectivity $\lambda_\Pi(a)$ of some hyperarc $a \in \mathcal{A}$ is the number of parts connected by a. If $\lambda_\Pi(a) > 1$, then a is said to be cut; otherwise it is entirely contained within a single part and is not cut. The cut of partition Π is the set $\omega(\Pi)$ of cut hyperarcs, such as $\omega(\Pi) \stackrel{\text{def}}{=} \{a \in \mathcal{A}, \lambda_\Pi(a) > 1\}$. The cut size is defined as $f_c \stackrel{\text{def}}{=} \sum_{a \in \omega(\Pi)} \mathcal{W}_a(a)$. The *connectivity-minus-one* cost function f_λ of some partitioned hypergraph $\mathbf{H}^\Pi$ can then be defined as: $f_\lambda = \sum_{a \in \mathbf{A}} (\lambda_\Pi(a) - 1)$. Consequently, in our model, the distance between two vertices u and v may increase during partitioning due to the additional cost of routing paths between two (or more) parts. Let D_{ij} be the penalty associated with parts i and j such that if u is in part i and v is in part j, then: $d^\Pi_{max}(u,v) \geq d_{max}(u,v) + D_{ij}$. The objective function f_p can therefore be defined as the minimization of the longest path of $\mathbf{H}$ subject to partition Π: $f_p = \min d_{max}(\mathbf{H}^\Pi)$.

2.2 Related Work

Many previous works exploit existing *min-cut* partitioning tools, used as black boxes, and try to account for additional constraints. For instance, [2] presents a multi-objective approach based on hMetis. In [1] is proposed a pre-processing coupled with a recursive bi-partitioning algorithm using hMetis. They use path length values a hyperarc weights, to channel the partitioner towards minimizing path cost. Since path length values can change as the result of a bi-partitioning step, hyperarc weights are reevaluated after each of them to identify critical hyperarcs. Reference [6] compares a classic method using hMetis for partitioning, followed by a placement algorithm, with a derived approach consisting in placing and routing during the partitioning step. More recently, [10] performs some pre- and post-processing on the hypergraph to capture the critical path minimization objective within the cut-size metric, using hMetis as a partitioning tool. However, minimizing cut size is often not the most

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36024-4_50

relevant objective, and biasing *min-cut* cost functions to take *path-cost* minimization into account does not allow to handle properly platform topologies.

3 Contributions

Our first contribution consists of two initial partitioning algorithms, called DBFS and DDFS. They are based on breadth-first-search and depth-first-search methods, respectively. Both are driven by *vertex criticality*, *i.e.*, the length of the longest path traversing a vertex. These algorithms are combined with some cut minimization tools to achieve both objectives of preserving critical paths and minimizing the cut. Our second contribution consists of an extension of the FM heuristic [8], called DKFM. It aims to perform moves that minimize the cost function f_p, taking into account the target topology. The objective is to use DKFM as a post-processing cut minimization tool, for instance as a refinement method within a multilevel framework.

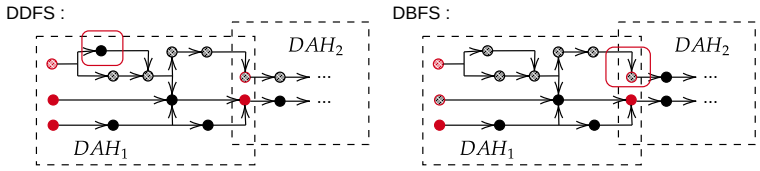


Fig. 1: A hint on DDFS vs. DBFS. All hatched nodes will be placed in the same part. DBFS avoids multiple cuts along paths within a DAH, while DDFS does not (see the red circle in the DDFS example). However, if a critical path starts in DAH_2 from a sink of DAH_1, DBFS can produce a cut along this critical path, while DDFS cannot (see the red circle in the DBFS example).

3.1 Initial Partitioning Based on Breadth-First Search Driven by Vertex Criticality (DBFS)

This first initial partitioning algorithm is based on an extended search along the vertices of **H**, driven by criticality. It groups neighboring vertices of each vertex v according to their criticality, to avoid multiple cuts along a long path. In our context, the multiple-cut constraint is also important because we need to avoid cutting the same path multiple times. Indeed, the cutting cost D_{ij} is often larger than the path length. However, many paths can be shorter than the longest path before partitioning. Therefore, our algorithm considers the criticality of each vertex v in addition to the criticality of its neighbors. To select vertices to consider, we use their in-degree value, decremented for all outgoing neighbors of v when visiting v. This gives us a topology-driven hypergraph traversal, considering vertex criticality. This algorithm allows us to perform a walk along the topological order in the current DAH, by

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36024-4_50

selecting, at each step, a neighbor of maximum criticality. This choice allows us to favor the grouping, within the same part, of neighboring vertices with high criticality. As long as the size constraint is respected, each selected vertex will be placed in the same part. An example can be found in Figure 1.

3.2 Initial Partitioning Based on Depth-First Search and Vertex Criticality (DDFS)

This second initial partitioning method performs a depth-first traversal driven by the criticality of the vertices. The main difference with the previous method is that all vertices will be inserted in the same priority queue, regardless of whether they are red or black. It enables a more compact visit order concerning the criticality value, but does not consider the topological structure of DAHs in $\mathbf{H}$. The main idea behind this method is to be able to pack interconnected critical paths of several DAHs in the traversal order. If the topology allows for it, the most critical neighboring paths will be placed in the same part, as long as the size constraint is respected. An example can be found in Figure 1. However, DDFS may induce cuts within the DAHs and a possible path cost degradation for paths of smaller criticality. The relative efficiency of DBFS and DDFS is likely to vary, depending on circuit topologies and the distributions of path lengths.

3.3 FM-Based Local Optimization Heuristic (DKFM)

A commonly used heuristic in partitioning tools is the algorithm proposed by Fiduccia and Mattheyses (FM) [8]. It is based on *local search* to move vertices across parts so as to reduce the cut of balanced hypergraph bipartitions. FM computes a move gain for each vertex and performs a single move at each step.

We propose an extension of this algorithm to minimize critical path degradation during partitioning. Let Π be a partition of $\mathbf{H}$; the gain function for some vertex v and some candidate partition Π' in which v may be moved to part π_k is defined as: $gain(\Pi,\Pi',\mathbf{H}) \stackrel{\text{def}}{=} d^{\Pi}_{max}(\mathbf{H}) - d^{\Pi'}_{max}(\mathbf{H})$.

4 Experimental Results

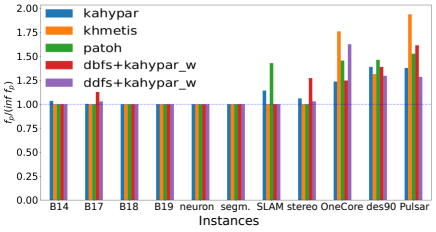
4.1 Methdology

The hypergraphs of our benchmark are taken from the Titan23 [11] and ITC99 [7] benchmarks, and *Chipyard* [3]. We also defined two target topologies composed of four elements. Test Architecture 1 connects its four FPGAs as a cycle π_0,π_1,π_2,π_3, while Architecture 2 is a chain π_0,π_1,π_2,π_3. In order to evaluate the DBFS and DDFS algorithms, we first generated an initial partition and then ran kKaHyPar to refine it, using the objective function f_λ. To test the DKFM refinement algorithm, we first ran PaToH-S, khMetis, and kKaHyPar on the weighted hypergraph instances *xxx_ w*, with our weighting scheme driven by vertex-hyperarc criticality. Then, we applied our DKFM algorithm as post-processing to these outputs, to study the *path-cost* improvement and possible degradation of the *connectivity-minus-one*

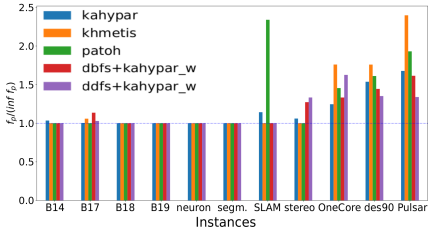
ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36024-4_50

metric induced by our refinement pass. We chose KHMETIS over HMETIS, used in previous works, because we get better results in the majority of cases for both target topologies. We select PATOH in its "speedy" (S) version to see whether it could produce acceptable solutions in combination with DKFM, because computation time is a critical aspect of industrial-size circuit prototyping.

4.2 Initial Partitioning Algorithms



(a) Comparison of f_p with respect to inf f_p on Architecture 1



(b) Comparison of f_p with respect to inf f_p on Architecture 2

Fig. 2: Comparison of f_p on Architecture 1 (a) and Architecture 2 (b).

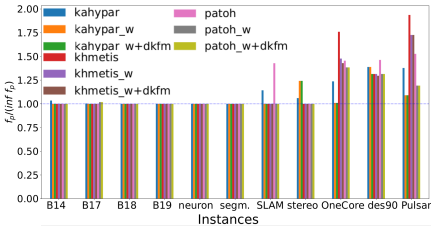
Figures 2a and 2b evidence that in many cases, initial partitioning algorithms DBFS and DDFS, combined with our weighting scheme, can influence a cut minimization partitioning tool to improve path cost. However, for some instances (*b17*, *OneCore*), the algorithms perturb the cut minimization tools and produce worse results. Combined with a cut minimization tool, these two algorithms yield an average improvement in 18% (DBFS) and 45% (DDFS) of the instances when mapping onto Architecture 1, and an average improvement in 36% of the instances on Architecture 2, for both algorithms. Compared to the cut minimization approach, DBFS improves the path cost metric by 8% and 7% on the two architectures, respectively, while DBFS improves path cost by 7% and 17%, respectively. Moreover, in many instances, the produced path cost is equal to the lowest possible cost, and it is impossible to improve it further. It shows that both algorithms indeed prevented critical paths from being cut. We already see a gain at this scale, which is encouraging for testing these methods on more significant cases. The quality of the partitions produced by the two algorithms varies, depending on the instances. We suppose this somewhat reflects the topological properties of the instances (e.g., whether the black vertices are heavily interconnected at each level). Further analysis are required to determine relevant criteria. Our two algorithms differ in the way vertices are visited: DDFS (depth-first) is more likely to cluster long paths in the same part, while DBFS (breadth-first) is more likely to cluster together vertices on the same level with respect to source red vertices. We assume that these algorithms succeed in preventing the same path from being cut multiple times, by packing together the neighborhood of black vertices within DAHs. It allows them to produce results that are topologically compatible with the different hardware

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36024-4_50

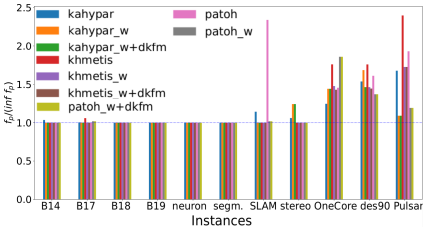
topologies. Notably, DBFS is adapted to linear (streamlined) hardware topologies, where vertices belonging to the same levels of computation must be packed together.

4.3 Refinement Algorithm

Figures 3a and 3b display path cost values for the smallest possible value of f_p (that is, the one obtained when no edge is cut).



(a) Comparison of f_p with respect to inf f_p on Architecture 1, with 20% DKFM moves and ρ=0



(b) Comparison of f_p with respect to inf f_p on Architecture 2, with 20% DKFM moves and ρ=0

Fig. 3: Comparison of f_p on Architecture 1 (a), and Architecture 2 (b).

Figure 3a and 3b evidence an improvement in maximum path cost on all instances, ranging between 27% and 54%, depending on the architectures. Interestingly, despite using the fast mode, PaToH-S +DKFM provides similar results. In some cases, DKFM does not improve the solutions produced by the cut minimization tools or can even produce worse solutions (e.g., for *stereo_vision*). We attribute this behavior to difficulties experienced by the DKFM algorithm to get out of the local minima computed by the min-cut algorithms, all the more in the case of weighted cut minimization. This, already partially addresses the path cost minimization objective. On the other hand, DKFM can improve a weighted cut minimization solution (e.g., for *des90*) on both architectures. Finally, in instances not already of minimal path cost, DKFM improves the path cost by 12% to 17% for Architecture 1, and by 12% to 40% for Architecture 2. At this scale, we already see a gain from a solution close to a local minimum, which is encouraging to test DKFM on larger target topologies and with a dedicated cut minimization algorithm that allows more movement. Indeed, Architecture 2 evidences the drawbacks of plain partitioners when communication costs vary significantly between different parts. DKFM is more likely to benefit to partitions in which vertices have been mapped onto remote parts instead of closer ones. In this case, moving a few vertices to the right parts can make a huge difference. It is especially true for case *SLAM_spheric*, for which the improvement is of more than 50% (highly influencing the computation of the average gain over all instances for Architecture 2).

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36024-4_50

5 Conclusion and Future Work

This paper presents the red-black oriented hypergraph model and its associated red-red path cost metric. Our results show that the DBFS and DDFS algorithms are relevant and complementary initial partitioning methods, depending on circuit instances and their underlying topologies. DKFM results show an average improvement in critical path length ranging between 12% and 40%, compared to the initial min-cut solutions. These algorithms seem to be a good approach to improve performance of multi-FPGA prototyping. However, these methods can degrade cut size by a factor from 1.2 up to 5.0. At this stage, only the DKFM refinement algorithm takes into account the topology of the target hardware and correctly manages outliers. Hence, it may not be easy to improve incrementally and locally on results provided by the architecture-unaware algorithms DBFS and DDFS. It would be interesting to work on the placement of vertices according to the target topology from the initial partitioning phase, as SCOTCH [13] does for unoriented graphs.

References

1. Ababei, C., Bazargan, K.: Timing minimization by statistical timing hMetis-based partitioning. In: 16th International Conference on VLSI Design, 2003. Proceedings.
2. Ababei, C., Navaratnasothie, S., Bazargan, K., Karypis, G.: Multi-objective circuit partitioning for cutsize and path-based delay minimization. In: ICCAD 2002.
3. Amid, A., Biancolin, D., Gonzalez, A., Grubb, D., Karandikar, S., Liew, H., Magyar, A., Mao, H., Ou, A., Pemberton, N., Rigge, P., Schmidt, C., Wright, J., Zhao, J., Shao, Y.S., Asanović, K., Nikolić, B.: Chipyard: Integrated design, simulation, and implementation framework for custom socs. IEEE Micro **40**(4), 10–21 (2020)
4. Çatalyürek, Ü., Aykanat, C.: PaToH. Springer US, Boston, MA (2011)
5. Çatalyürek, U., Devine, K., Faraj, M., Gottesbüren, L., Heuer, T., Meyerhenke, H., Sanders, P., Schlag, S., Schulz, C., Seemaier, D., Wagner, D.: More recent advances in (hyper)graph partitioning. ACM Comput. Surv. **55**(12) (mar 2023)
6. Chen, M.H., Chang, Y.W., Wang, J.J.: Performance-Driven Simultaneous Partitioning and Routing for Multi-FPGA Systems. In: 2021 58th ACM/IEEE DAC (2021)
7. Corno, F., Reorda, M., Squillero, G.: RT-level ITC'99 benchmarks and first ATPG results. IEEE Design & Test of Computers **17**(3), 44–53 (2000)
8. Fiduccia, C., Mattheyses, R.: A Linear-Time Heuristic for Improving Network Partitions. In: 19th DAC (1982). https://doi.org/10.1109/DAC.1982.1585498
9. Karypis, G., Kumar, V.: Hmetis: a hypergraph partitioning package. ACM Transactions on Architecture and Code Optimization (1998)
10. Liou, S.H., Liu, S., Sun, R., Chen, H.M.: Timing Driven Partition for Multi-FPGA Systems with TDM Awareness. Ass. Comp. Mach., New York, NY, USA (2020)
11. Murray, K.E., Whitty, S., Liu, S., Luu, J., Betz, V.: Timing-Driven Titan: Enabling Large Benchmarks and Exploring the Gap between Academic and Commercial CAD. ACM Trans. Reconf. Technol. Syst. **8**(2) (mar 2015)
12. Ou, S.L., Pedram, M.: Timing-driven partitioning using iterative quadratic programming
13. Pellegrini, F., Roman, J.: SCOTCH: A Software Package for Static Mapping by Dual Recursive Bipartitioning of Process and Architecture Graphs. In: HPCN Europe (1996)
14. Schlag, S.: High-Quality Hypergraph Partitioning. Ph.D. thesis, Karlsruhe Institute of Technology, Germany (2020)

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36024-4_50