Time-Scalable Mapping for Circuit-Switched GALS Chip Multiprocessor Platforms

Mohammad H. Foroozannejad, *Student Member, IEEE,* Matin Hashemi, *Member, IEEE,* Alireza Mahini, Bevan M. Baas, *Senior Member, IEEE,* and Soheil Ghiasi, *Senior Member, IEEE*

***Abstract*—We study the problem of mapping concurrent tasks of an application to cores of a chip multiprocessor that utilize circuit-switched interconnect and global asynchronous local synchronous (GALS) clocking domains. We develop a configurable algorithm that naturally handles a number of practical requirements, such as architectural features of the target platform, core failures, and hardware accelerators, and in addition, is scalable to a large number of tasks and cores. Experiments with several real life applications show that our algorithm outperforms manual mapping, integer linear programming-based mapping after ten days of solver run time, and a recent packet-switched network on chip-based task mapper through which, we underscore the unique requirements of task mapping for circuit-switched GALS architectures.**

***Index Terms*—Algorithm, chip multiprocessor (CMP), global asynchronous local synchronous (GALS), task to processor mapping.**

I. Introduction

CHIP multiprocessor (CMP) platforms are commonplace in both research and the commercial marketplace. Technology trends suggest that the trend of integrating more processor cores per chip will continue into the foreseeable future, and domain-specific many-core chips with 1000+ cores seem imminent [2]–[4].

As the number of processors per chip grows, high-speed communication between cores becomes more challenging. Although packet-switched network-on-chip (NoC) architectures [5], [6] offer modular and design reusable solutions, their reliance on a single voltage-clock domain becomes a limiting factor for both performance and power reduction. Circuit-switched globally asynchronous locally synchronous (GALS) interconnects [7], [8] offer a promising alternative to improve both factors [4], [9].

While domain-specific many-cores promise large gains in performance and energy efficiency, development of application software for their utilization remains a major challenge [10]–[16]. Furthermore, there is a pressing need for tools to efficiently map application concurrent tasks to platform resources. In this paper, we study the problem of mapping a given application task graph to the processors of a given CMP platform subject to platform constraints (e.g., limited interconnect resources). In this paper, we aim to devise extensible and scalable mapping algorithms that can: 1) scale to task graphs/platforms with 1000+ tasks/cores and 2) provide a judicious balance between solution quality and tool run time.

Manuscript received June 26, 2013; revised December 13, 2013; accepted December 16, 2013. Date of current version April 17, 2014. This paper was recommended by Associate Editor D. Atienza.

M. Foroozannejad, B. Baas, and S. Ghiasi are with the University of California, Davis, CA 95616 USA (e-mail: mhforoozan@ucdavis.edu; bbaas@ucdavis.edu; ghiasi@ucdavis.edu).

M. Hashemi is with the Department of Electrical Engineering, Sharif University of Technology, Tehran 11365, Iran (e-mail: matin@sharif.edu).

A. Mahini is with the Department of Computer Engineering, Islamic Azad University, Gorgan, Iran (e-mail: president@gorgan-samacolleg.ir).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2014.2299958

II. Related Work

The mapping between virtual and physical resources, which affects application throughput, energy consumption, and quality of service, is one of the challenges of implementing an application on a NoC-based platform.

Marcon *et al.* [17] proposed a simulated annealing algorithm to solve the mapping problem for mesh based NoC architectures, targeting both throughput and power consumption of the system. Ascia *et al.* [18] proposed a multiobjective task mapping approach using genetic algorithms with similar optimization criteria as used in [17]. Murali and De Micheli [19] employ traffic splitting as a technique to reduce the required bandwidth on links of the network. They revisit the problem in [20] with an emphasis on quality of service (QoS) in the final mapping solution.

Hu and Marculescu [21] propose a runtime-aware technique using a branch and bound algorithm, which constructs a mapping solution with a deadlock-free deterministic routing function such that the total communication energy is minimized. Srinivasan and Chatha [22] proposed a technique called MOCA, which utilizes the principles used in [19] with a focus on the energy consumption of the system. Tosun *et al.* [23] formulate the mapping problem using integer linear programming (ILP), and leverage the best solutions found within tolerable solver time to obtain the optimal or high quality mapping solutions. Tosun [1] later proposed another technique called CastNet, which takes advantage of the symmetry in mesh architecture to improve both energy consumption and algorithm runtime compared to NMAP [19] and MOCA [22] algorithms.

The aforementioned approaches target similar packet-based NoC as the underlying interconnect architecture. Thus, they do not readily address our problem at hand, which concerns circuit-switched interconnection of processors. The comparison between the two approaches to interconnect network design is out of the scope of this paper, however, one can find such a comparison in [9].

In our preliminary study [24], we introduced the basic components of a mapping algorithm called balanced mapping space exploration (BAMSE). In this paper, we bring practical considerations, such as usecase scenarios, core failures and fixed functions, into the mapping context. We also explore the trade-off between solution quality and the tool run time via parameter configuration. Furthermore, we statistically analyze the problem of parameter configuration, and outline development of a configuration layer on top of the basic algorithm for arriving at a solution with acceptable quality.

III. Target Platform and Application Model

The focus of this paper is on AsAP-like platforms with statically allocated interconnect architecture. Such platforms offer a promising tradeoff between energy efficiency and programmability for selected applications [25]. Discussing AsAP2 as an example in this section would bring clarity and justification to some of the decisions we make in designing the algorithm. It should also be noted that our approach is generic in nature and potentially applicable to other many-core GALS architectures as well.

AsAP2 [4] is an academic many-core GALS processor and contains 164 programmable cores, three fixed function cores, and three fixed memory modules that are interconnected via mesh topology. Each core in AsAP2 is connected to a router, and each router is connected directly to its four nearest neighbor routers with two unidirectional links in each direction. Longer communications are possible by connecting a series of links between cores. In theory, each core can communicate with any other core on the chip.

Fig. 1(a) illustrates the architectural specifications of AsAP2 [4]. Fig. 1(b) shows the effect of interconnect distance between communicating processors and the clock frequency of the source processor in this particular CMP platform [25]. The drop in source core frequency is due to the fact that AsAP2 uses circuit-switched interconnection for intercore communication, in which the clock signal of the source core is sent along with the data to maintain communication synchrony [25]. Note that even an infrequent low throughput control signal would slow down the clock rate at the source core, if it has to travel far to a destination. The detailed discussion on the hardware implementation of AsAP2 is beyond the scope of this paper and is presented in [4].

Another limiting factor in circuit-switched interconnects is the limited network resources. In AsAP-like circuit-switched architectures, links are statically allocated between two communicating cores at the programming phase after reset when the application is loaded to the processor. Therefore, these links cannot be shared by other intercore connections. This is in contrast to packet-switched networks in which, the physical resources can be shared and the limitation reduces to a constraint on total bandwidth allocated to links. Although in this approach finding deadlock free mappings will not be of any concern (since the resources are not shared), it can also reduce the number of feasible solutions to a great degree.

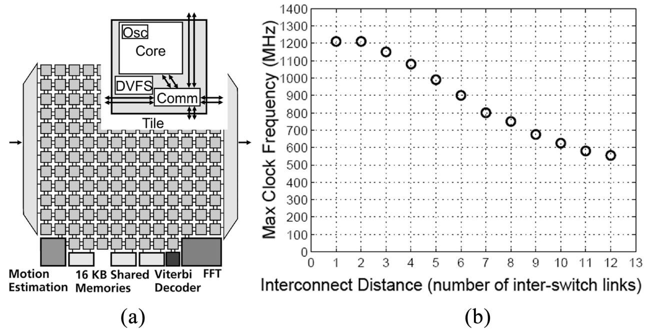


Fig. 1. (a) Block diagram of AsAP2 architecture [4]. Each bidirectional connection depicted in this figure is composed of two separate unidirectional connections in opposite directions. (b) Measured maximum source core clock frequency for interconnect between AsAP2 processors of various distances [25].

For example, in AsAP2 there are only two bidirectional links between any two neighboring cores. The restricted resources are due to the emphasis on simultaneous extreme energy efficiency and programmability philosophy of the platform design, which stresses the significance of link resource management during task mapping.

The platform is particulary efficient in implementing embedded streaming applications, which are primarily characterized by the requirement to process a steady stream of input data as they are presented to the system. Such applications, are well modeled using task graphs in which, graph nodes model tasks and graph edges represent intertask communication channels. In our discussion, we assume that application tasks are already allocated to processors in the system and will be executed on self timed schedule [26]. That is, graph nodes (tasks) are viewed as virtual processors that need to be mapped to physical processors existing on the chip, and all virtual processors continue execution as long as they have input data and space available on their input and output queues, respectively.

IV. Problem Statement

Both application and hardware platform are represented in the form of graphs

$$\text{Task Graph: } G = \ < V, E > \quad (1)$$

$$\text{Hardware Graph: } H = \ < C, L, CAP_L > . \quad (2)$$

In task graph G, V denotes the set of vertices, which model tasks, and E is the set of edges, which represents intertask communication. Unless otherwise noted, we liberally use the notion of task and intertask communication in G to refer to the set of application tasks that are assigned to the same virtual processor (i.e., a coarse-grain task), and the corresponding interprocessor communication, respectively.

The hardware graph H consists of C, which represents a set of available cores on the chip, and a set of links L, which is a subset of $C \times C$. L models possible direct physical links between cores. Each core is connected to its own router, which is responsible for receiving data from and sending data to other cores. These routers are also connected to their neighbors, and can be statically configured to implement longer communication links. CAP_L is a function that assigns a capacity number to links in L.

The mapping solution is characterized by two sets, S and R, which give the mapped processor and the allocated links for intertask communication, respectively

$$map(G, H) \rightarrow < S, R > \quad (3)$$
$$S \subset V \times C \quad (4)$$
$$R = \{\ path_{ij} \in P(L) \mid e_{ij} \in E\} \quad (5)$$

where $P(L)$ is the power set of L. All tasks must be assigned to cores, and a core can execute at most one task. Formally

$$\forall\ v \in V \quad \exists\ c \in C \quad : \quad (v, c) \in S \quad (6)$$
$$(v_1, c) \in S \quad \text{and} \quad (v_2, c) \in S \Longrightarrow v_1 = v_2 \quad (7)$$
$$(v, c_1) \in S \quad \text{and} \quad (v, c_2) \in S \Longrightarrow c_1 = c_2 \quad (8)$$

$path_{ij}$ refers to a lean subset of links that connects v_i to v_j in the mapping solution.

The mapping solution must satisfy capacity constraints

$$\forall\ l \in L : CAP(l) \ \leq \ \Sigma_R \ \text{ paths that contain } l. \quad (9)$$

That is, at most $CAP(l)$ paths can use the link l in a valid mapping solution.

It is hard to accurately estimate performance and energy at the mapping level, however, it is evident that they are both adversely effected by longer connections [Fig. 1(b)]. We use two attributes of the mapping solution as proxies for performance and energy, and use them as the optimization objective. Specifically, we use the longest connection (LC) and total number of connections (TC)

$$LC = \max_R \quad |path_{ij}| \quad (10)$$
$$TC = \Sigma_R \quad |path_{ij}|. \quad (11)$$

The mapping problem as defined has a multiobjective optimization criteria. We use the tuple (LC, TC) to denote the cost of a mapping solution. Given the relative importance of the metrics, different candidate solutions must be compared lexicographically. That is, the longest connection (LC) has the highest priority for optimization regardless of total connection (TC). In comparison of two mappings with equal LC, the one with smaller TC value would be considered to have a smaller cost.

V. BAMSE Algorithm

We proceed to present our algorithm, called BAMSE, for solving the formulated mapping problem. The basic idea is to maintain a list of partial mapping solutions (initialize to empty), and incrementally augment the partial mappings by mapping a new task to an available core, while ensuring that only a small number of promising partial mapping candidates are maintained from iteration to the next. In the remainder of this section, we will discuss these steps in detail. Specifically, we discuss iterative selection of Tasks for mapping (task selection); augmentation of partial mappings by finding suitable cores for mapping the task at hand (core selection); and judicious maintenance of a small subset of partial mappings to avoid exponential growth of retained partial solutions (mapping selection).

A. Task Selection

Task Selection is the process by which tasks are sequentially labeled for incremental mapping. Since the goal is to map a task as close as possible to its connected tasks, breadth first search (BFS) is an intuitive choice for ordering of the tasks. In BFS, ordering the immediate children of a node are favored over farther nodes in the graph, however, standard BFS is silent on the tie breaking strategy when it comes to children of a node.

To order the tasks, we use the principle leveraged by Cuthill-McKee variant of the BFS algorithm [27], which heuristically aims to reduce the bandwidth of the resulting sequence of tasks. Specifically, the tasks are ordered in BFS order, while the children of a task are themselves visited in increasing order of their degree in the graph. In this context, the term bandwidth is historically used to denote the maximum distance (the number of tasks) between any parent and its children in the sequence, and should not be confused with bandwidth of communication channels. To avoid confusion, we use maximum distance to children (MDC) to refer to bandwidth in graph theory, and restrict the use of bandwidth to communication performance discussions.

Fig. 2(a) shows the generated task sequence using Cuthill-McKee BFS, while another variant of the BFS is used to generate the task sequence shown in Fig. 2(b). The difference is that in the former, node C is selected as the first child of node A over node B due to its smaller degree, whereas in the latter, node B has been selected as the immediate child after node A. These child ordering policies lead to MDC = 2 for the Cuthill-McKee BFS (only nodes H and D stand between node B and its farthest child node F), and MDC = 3 for the basic BFS algorithm. Intuitively, the task sequence with smaller MDC gives the advantage of visiting the other tasks connected to the current task earlier in the process, which heuristically results in mapping them closer to the current task.

B. Core Selection

A partial mapping (PM) is a mapping for the first k tasks in the sequence, where $k \neq |V|$. Let, v_{next} refer to the next task, namely task $(k + 1)$, in the sequence. Since the task sequence is created by BFS of a connected graph, the parents of a task are visited prior to the task itself. Thus, in any partial mapping there is a non-zero number of cores (mapped tasks) that are connected to the next task. Let connected mapped cores refer to the set of such cores.

In order to assign v_{next} to a core, the core selection process identifies a number of unoccupied cores as potentially good matches based on their distance to the connected mapped cores. The lower bound on expected number of potential matches is a configurable parameter, called minimum number of potential candidate cores (MPC). Smaller values of MPC would force the algorithm to behave more greedily, while the

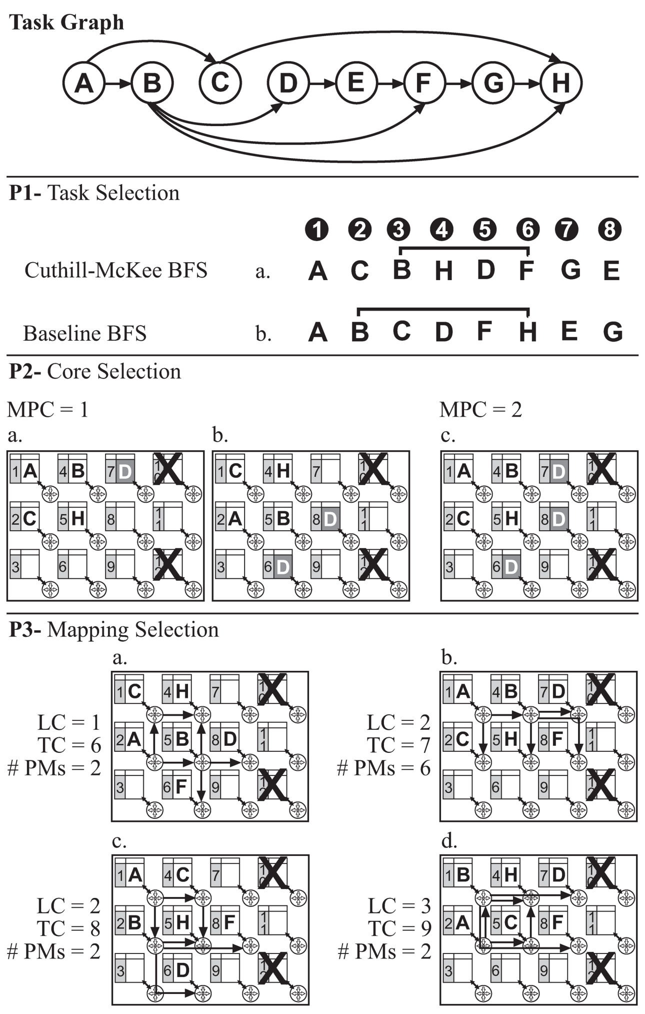


Fig. 2. Snapshots of BAMSE steps on an example task graph.

larger values tilt the balance toward more thorough search of the solution space.

To create at least MPC potential candidate cores, the neighboring set of each connected mapped core is created in levels. From one level to the next, the acceptable radius that defines neighborhood distance in Manhattan is incremented. The intersection between the neighboring sets of all connected mapped cores determines the potential candidate cores at a particular level, since mapping v_{next} to any of them would not create a link longer than the acceptable radius. The neighboring radius is incremented until enough number (at least MPC) of potential candidate cores are generated. Once the list of potential candidate cores are generated, existing partial mappings are augmented by mapping v_{next} to all of them. That is, for every existing partial mapping at least MPC augmented mappings are created.

In Fig. 2.P2, tasks A, B, C, and H are assumed to have been mapped in previous iterations. The task that is being mapped in the current iteration (v_{next}) is D. Task B is the only task connected to D among the mapped tasks, thus the connected mapped cores set in the partial mapping 2.P2.a is $\{core4\}$, and in the partial mapping 2.P2.b is $\{core5\}$. These two partial mappings are only shown as examples, and potentially, there exist many other partial mappings at this iteration. The potential candidate cores for node D for each partial mapping are shown in gray color.

If $MPC = 1$, the closest available cores to the connected mapped cores set are explored until at least one possible candidate is found. At radius one, partial mapping 2.P2.a has one potential candidate core, however, there are two potential candidate cores for the partial mapping 2.P2.b. If MPC is 2, we would still have enough candidate cores in the case of Fig. 2.P2.b. However, the number of potential candidate cores in 2.P2.a would not be enough (there is only one in the set). Therefore, the neighborhood radius is incremented, and farther neighbors are explored to find at least MPC total candidates. Fig. 2.P2.c illustrates the result of expanded neighborhood.

The process does not favor any of the candidates over the others, and it accepts all potential candidates. The term minimum number in MPC underscores that the number of potential candidate cores for augmentation of a partial mapping can be greater than or equal to MPC.

A subset of processor cores might be unavailable due to various factors that are discussed in Section V-E. Unavailable cores are provided to the algorithm as part of platform resource description. At each step of finding potential candidate cores, the candidates are compared to the list of unavailable cores and eliminated from the set if they match the list. After this elimination the number of remaining potential candidate cores is compared to MPC to decide if incrementing neighborhood radius and exploration of farther neighbors is necessary.

C. Mapping Selection

In the first iteration of BAMSE, the first (start) task of the sequence is mapped to a core. Given the BFS nature of our task selection process, we ensure that the start task interfaces to the application input. The location of the start task might be restricted to a subset of all cores, as it has to interface with the input data stream. For example in AsAP2, the cores on the leftmost column of the chip have access to the input pins of the chip. It follows that the list of partial mappings is initialized with such possible mapping of the start task. Mapping of the start node creates a set of partial mappings to start the process. In subsequent iterations existing partial mappings are augmented.

For a given partial mapping in a subsequent iteration, the task under consideration can potentially be mapped on any of the cores in its corresponding potential candidate cores set. As a result, multiple augmented partial mappings are created from an existing partial mappings, and are added to a list, called mapping list. The mapping list contains all points in the solution space that might have a chance to evolve into the final solution. To avoid state explosion, the mapping list is sorted in ascending order based on the cost of each partial mapping. The size of the list is also limited by the configurable parameter window size (WS). If the mapping list has WS partial mappings, each newly generated augmented partial mapping with a cost greater than the last partial mapping in the list (i.e., highest cost) is dropped; otherwise the new partial mapping is placed in the list based on its mapping cost, and the last mapping is removed from the list to maintain its

WS size. Smaller values of window size force the algorithm to be more greedy, while larger values tilt the balance toward more thorough exploration of the space at the expense of longer algorithm runtime.

Fig. 2.P3 shows the number of possible partial mappings, when mapping task F. There are 12 partial mappings in this stage, and they all fall into one of the four cost profiles shown in Fig. 2.P3.a, b, c, and d. For each cost profile, only one of the partial mappings is depicted as a representative of the group. The number of partial mappings (PM) in each category is also reported.

In this example if window size (WS) is 2, only the partial mappings in Fig. 2.P3.a are passed to the next iteration. The comparison criteria is the multiobjective cost function presented in Section IV, which uses longest connection (LC) as the primary, and total number of connections (TC) as the secondary cost component. If window size is set to 8, all partial mappings in Figs. 2.P3.a and b are kept in the mapping list. In case window size is 12, all of the partial mappings will be passed to the next iteration, where their augmentation with the next node (G) is considered.

The algorithm performs under the general assumption that the input task graph does not violate any immediate feasibility constraint. For example, we assume that the maximum connectivity degree in the task graph does not exceed the architecture connectivity limitation. That is, to use AsAP2 as an example, there exists no task connected to nine other tasks. Similarly, we assume that the number of input or output tasks are not more than the number of input or output cores on the chip. Note that checking for these conditions is rather trivial, so assumption is practically the same as ensuring that graph does not violate immediate feasibility constraints.

1) *Look Ahead Technique (LAT):* In practice many partial mappings at the end of the list are likely to have identical cost tuples. In Fig. 2.P3, for example, if window size is 4 then some of partial mappings in Fig. 2.P3.b must be dropped. In this section, we introduce a tie-breaking technique that enables us to differentiate between such partial mappings with identical costs, based on the likelihood that they will lead to superior solutions down the road.

A naive way to solve this problem would be to select an excessively large window size to ensure that most of promising partial mappings are maintained in the list throughout the run time of the algorithm. Very large window sizes, however, have adverse implications on algorithm runtime and memory requirements. Rather than increasing the window size, our approach to overcoming this problem is to look ahead in the task sequence, and quickly estimate the cost of augmented partial mapping that would result from a given partial mapping at the current iteration of the algorithm. For a given partial mapping, our look ahead technique (LAT) quickly maps a few more of upcoming tasks. The anticipated cost of such augmented partial mappings, called secondary costs, is used as the tie-breaker to sort the existing partial mappings.

Secondary costs are merely needed as a tie-breaking mechanism to better sort the partial mappings at the end of the list. Thus, it is reasonable to use a quick greedy technique

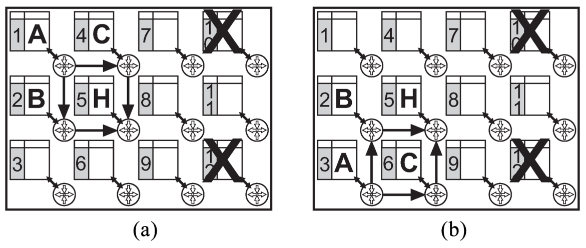


Fig. 3. Both partial mappings lead to equivalent augmented mappings.

to map the upcoming nodes, and to establish the secondary costs. In particular, we choose to run BAMSE algorithm with $WS = 1$ and $MPC = 1$ settings for a few more steps for partial mappings that need tie-breaking. Note that the look ahead technique is only performed to estimate secondary costs, and estimated mappings are not actually committed.

We use a configurable parameter called forward number (FN) to indicate the number of upcoming tasks that are mapped by the look ahead technique. Intuitively, a good choice of forward number should allow us to consider all children of the current task in the look ahead phase. Recall from our discussion in Section V-A that the maximum distance between a task and its children in the task sequence is readily obtained by traversing the sequence, and is called maximum distance to children (MDC). Thus, we set the forward number to the sequence MDC.

The use of look ahead technique enables us to better sort the partial mappings in the list, and effectively reduces the required window size to obtain quality solutions, when compared to the baseline approach. Although it increases the complexity of the algorithm in the asymptotic sense, dealing with a smaller window size has positive effect on the runtime of every iteration. Thus, its true impact on the overall algorithm runtime is a balancing act.

2) *Redundant Mappings Elimination Technique (RMET):* Although no two partial mappings in the mapping list are identical, not all of them lead to different mappings in terms of solution quality. For example, the two partial mappings shown in Fig. 3 are different, however, they yield equivalent augmented mappings of as the algorithm progresses. As such, one of them is redundant. Accurate identification of redundant partial mappings is an instance of graph isomorphism, which is a well known NP-hard problem. In order to avoid the computational cost, we settle for an approximate test, which in practice identifies redundant partial mappings with sufficient accuracy. Specifically, we use the cost of a partial mapping, and the locations of its terminal cores as a measure of redundancy among partial mappings. The terminal cores refer to the set of cores in a partial mapping that are connected to unmapped upcoming tasks in the application (set $\{B, H\}$ in Fig. 3). Intuitively, if two partial mappings have the same cost and their cores with open connection are at the same locations, it is very likely that one of them is redundant. Elimination of a redundant partial mapping allows us to utilize the limited space on the mapping list more efficiently.

D. Integrated Link Assignment

Due to limited network resources on many of the circuit-switched platforms, not all task mappings would result in feasible implementations with valid link assignment between connected cores. In our experiments with AsAP, link assignment was a very constraining factor for some benchmark applications. We conclude that it is more efficient to integrate link assignment in the task mapping process, to avoid generation of infeasible mapping solutions. Since link assignment needs to be applied to the many partial mappings that are considered by BAMSE, an important design preference is fast runtime over thorough exploration of the search space.

To this end, we developed a XY link assignment algorithm that incrementally assigns links to surviving augmented partial mappings. Our XY link assignment technique only considers paths that entirely lie in the bounding box of two connected cores. In mesh interconnects, the length of all such paths is exactly the Manhattan distance between the two cores. A book-keeping table records the occupied link resources for each partial mapping. In each iteration of BAMSE, an augmented partial mapping inherits the table from its parent partial mapping, and incrementally adds new information on allocated links to the table. Subsequently, the capacity of the allocated links are updated. Infeasible partial mappings, i.e., those that cannot successfully establish all required intercore communications with remaining link resources, are eliminated from the mapping list.

E. Unavailable Cores and Fixed Functions

Due to a number of reasons such as prior mapped applications in multiapplication mapping or core failure caused by imperfect manufacturing yield and exhaustion of on-chip electronic components, a subset of cores might become unavailable for task mapping. BAMSE can be readily extended to handle unavailable cores. Specifically, unavailable cores are considered during construction of potential candidate cores in the core selection stage of the algorithm. In the example discussed in Section V-B, the set $\{core10, core12\}$ is unavailable for mapping.

Another practical requirement is to map specific tasks of the application to certain cores that contain special resources, for example, custom accelerators or memory resources, or other unique capabilities. For example in AsAP2, only the cores on the first (last) column of the chip can be connected to input (output) pins of the chip. In addition, AsAP2 has six tiles that implement the following functions [Fig. 1(a)]: motion estimation, Viterbi decoder, fast Fourier transform, and three shared memory modules. These resources have fixed locations on the chip, which must be taken into consideration when corresponding tasks are mapped to cores.

We propose to extend BAMSE to handle such constrained choices. Let fixed function refer to tasks that have constraints on matching cores. Intuitively, it is efficient to map fixed functions early in the process, so the subsequent connected tasks would be mapped to adjacent cores. As such, we extend the node selection process to initialize the BFS queue with the fixed functions. Subsequently, the aforementioned BFS-based task sequencing scheme orders the remaining tasks by visiting the children of the existing tasks in the queue. The core selection process naturally honors the dictated constraints, and otherwise the algorithm operates as described before.

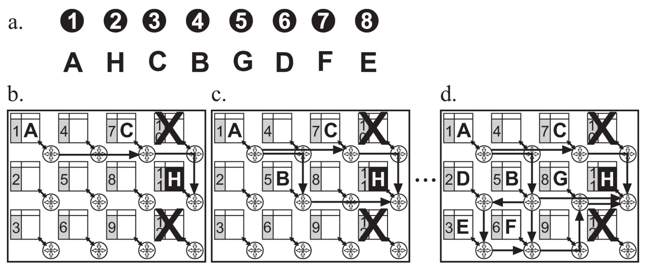


Fig. 4. (a) Task sequence of the task graph depicted in Fig. 2 after fixing task H on $core11$. The input task is always fixed to the first column on AsAP2. (b) and (c) Example partial mappings created from mapping nodes C and B, respectively. (d) Final generated mapping.

For example, let us assume that the output task H of the task graph depicted in Fig. 2 is constrained to be mapped to $core11$. Fig. 4(a) illustrates the resulting task sequence. Fig. 4(b) and (c) show sample partial mappings after mapping tasks C and B. Fig. 4(d) shows the final mapping generated by BAMSE. Note that both tasks B and C have connections to tasks A and H, and the surviving cores for both tasks have the minimum distance of two to the fixed functions.

F. Complexity Versus Quality

Larger values of WS and MPC increase the algorithm runtime, albeit in different ways. In addition, more partial mappings are explored, which should generally lead to better final mappings. In the merely-hypothetical impractical case that $MPC = |C|$ and $WS = |C|^{|V|}$, the entire search space would be exhaustively explored to find the optimal solution. In finding the proper value of parameters, a simple, though helpful, observation is that if a partial mapping whose augmentation could lead to an optimal final mapping survives at each iteration, the optimal solution could be successfully generated. The issue is further discussed in Section VI-D.

VI. Experiment Evaluation

In this section, we present our empirical study, whose results showcase the effectiveness of BAMSE in rapid generation of quality mapping solutions.

A. Setup

We use AsAP2 manycore chip as the target platform for mapping a number of applications. The relevant architectural features of AsAP2 are highlighted in Section III.

1) *Benchmark Applications:* To evaluate the proposed technique we selected five different streaming applications as our benchmarks. The benchmark applications, which were previously reported in a number of technical publications, are developed, manually optimized (including task mapping) and validated for correct functionality. They include Viterbi decoder [28], wireless LAN 802.11a baseband receiver [29], two different implementations of advanced encryption standard (AES) encryption algorithm [30], and H.264/AVC video encoder [31] kernels. These kernels frequently appear in many

TABLE I

BENCHMARK APPLICATION SET SPECIFICATIONS: D IS THE TASK GRAPH DEGREE, AND MDC IS THE MAXIMUM DISTANCE BETWEEN A PARENT AND ITS CHILDREN AFTER SEQUENTIAL ORDERING OF TASKS USING BFS

Application Name	# Tasks	# Channels	D	MDC
Viterbi decoder	30	35	3	4
802.11a baseband receiver	25	40	6	9
small AES	59	79	3	4
large AES	137	176	6	8
H.264/AVC encoder	115	165	7	24

higher-level streaming applications that are widely used in many embedded systems.

Table I reports the number of tasks, number of intertask communication channels (i.e., task graph edges), task graph degree D (maximum number of connected tasks to a task), and MDC (maximum distance between a parent and its children after tasks are sequentially ordered) for all of the applications. Wireless LAN 802.11a baseband receiver uses FFT hardware accelerator, and H.264/AVC encoder utilizes both motion estimation and FFT hardware accelerators of AsAP2 as fixed function nodes (Section V-E).

2) *System and Algorithm Configuration:* BAMSE algorithm uses two configurable parameters: WS and MPC. In our experiments, we run the algorithm with 2400 different combinations of these parameters. Specifically, WS = 1, 2, ..., 300 and MPC = 1, 2, ..., 8. Each of these configuration points (WS, MPC) represents a level of greediness/thoroughness characteristics of the algorithm.

The objective of the mapping is to minimize the multiobjective cost function presented in Section IV, with the priority order of longest connection (LC) and total connection (TC). The experiments are performed on a Unix PC with Intel Xeon CPU running at 3.07 GHZ, 8192 KB of cache, and 6 GB of main memory.

B. Results

Fig. 5 illustrates the improvement of BAMSE over manual mapping in longest connection and total connections of the mapped task graph. All applications were developed and mapped before the start of our mapping project, and the developers had the incentive to improve the mapping result as it simplified their work and impacted their reported performance and throughput figures in their published work [28]–[31]. Therefore they are representative of what manual mapping can achieve. This is contrast to the predominant notion of comparison with manual optimization in CAD community in which, manual results are derived in parallel to automated results.

In principle, hand optimized mapping should give the optimal result, however, the sheer size and complexity of solution space (large applications with 100+ tasks and 150+ links, and architecture interconnect constraints) prevent humans from efficient exploration of the search space. Out of 2400 configuration settings, both best case and average case results are depicted in the figure. The improvements in longest connection (LC) are as high as 65%. In most cases, the

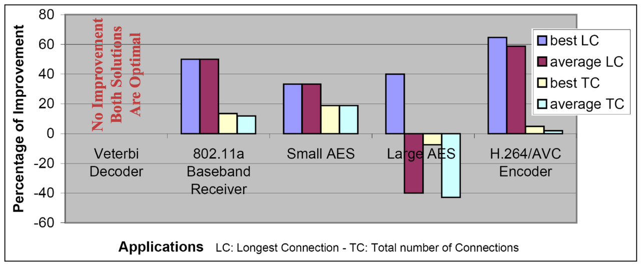


Fig. 5. Improvement of BAMSE over manual mapping in longest connection (LC) and total connections (TC).

second objective of total connection (TC) is also improved. Given the priority-based definition of cost function, it is not necessarily possible to minimize both LC and TC in the absolute sense. In fact, sometimes reducing one increases the other as a compromise. However, Fig. 5 shows that for most applications manual mappings are improved in terms of both objectives. The average result of both objectives is also shown in this figure.

For comparison, we also generated integer linear program (ILP) instances of the simplified (excluding link assignment) mapping problem. Details of ILP formulation of the mapping problem is not presented, for brevity. Although ILP is a well known NP-hard problem and its runtime scales very poorly with input problem size, it is potentially helpful in establishing lower bounds on possible solution quality, and in quantifying the gap between the generated solutions and the optimal mappings.

In order to accelerate the ILP solver runtime, we occasionally leveraged knowledge of the problem to expose a smaller search space to the solver. Specifically, for smaller applications it is sometimes evident that the optimal solution would only use cores in a small region of AsAP2. In such cases, a smaller instance of the hardware mesh was used in generation of the ILP instance. In case of this paper, in particular, a 6x6 mesh of cores was used as the target platform for Viterbi decoder and 802.11 baseband receiver. We tried to solve the ILP instances using CPLEX, a commercial grade ILP solver. The solver was allowed to run for a maximum of ten days.

In addition to manual mappings and ILP solutions, we also implement CastNet algorithm [1] introduced in Section II. CastNet is a fast constructive algorithm specific to packet-switched NoC platforms with the ultimate goal of reducing the energy consumption of the system. CastNet uses the bandwidth information from the task graph and the distance between the connecting cores in the mapping solution to calculate the energy consumed for communication in the mapped application. As it was discussed in Section III, bandwidth is not an important measure in the context of circuit-switched GALS architectures. In order to adapt the benchmark applications for the CastNet algorithm and at the same time maintain the constraints and requirements of the AsAP2 architecture, we equally assign a fixed bandwidth on each edge of the task graph.

Table II shows the comparative study data. In addition to the longest connection and total connection of mapping,

TABLE II

BAMSE VERSUS ALTERNATIVES: RESULTS FOR LONGEST CONNECTION AND TOTAL NUMBER OF CONNECTIONS ARE REPORTED. A SMALLER HARDWARE PLATFORM (A 6X6 MESH OF CORES) IS USED FOR GENERATION OF ILP* INSTANCES TO ACCELERATE THE SOLVER RUNTIME. THE ILP** NUMBERS ARE OBTAINED BY TERMINATING THE SOLVER AFTER TEN DAYS, AND ARE NOT OPTIMAL

Application		Long Conn	Total Conn	Time
Viterbi Decoder	Manual	1	35	-
	BAMSE	1	35	1 (sec)
	CastNet	8	100	< 1 (sec)
	ILP*	1	35	46 (hours)
802.11a Base-band Receiver	Manual	6	58	-
	BAMSE	3	51	13 (sec)
	CastNet	8	79	< 1 (sec)
	ILP*	3	51	58 (hours)
Small AES	Manual	3	106	-
	BAMSE	2	86	2 (sec)
	CastNet	11	228	< 1 (sec)
	ILP**	3	105	10 (days)
Large AES	Manual	5	254	-
	BAMSE	3	273	170 (sec)
	CastNet	16	995	< 1 (sec)
	ILP**	5	328	10 (days)
H.264/AVC Encoder	Manual	17	353	-
	BAMSE	6	336	273 (sec)
	CastNet	16	702	< 1 (sec)
	ILP**	7	288	10 (days)

algorithm runtime is reported. As expected, manual mapping can generate good results for small graphs, however, the approach does not scale to somewhat more complex task graphs. In cases that we could solve the ILP formulation, BAMSE indeed had found the optimal solution dramatically faster. For the three more complex applications the ILP solver did not finish after ten days. Interestingly, in these cases the best solution found by ILP after ten days is far inferior to BAMSE results generated in fraction of the time.

The CastNet results in Table II suggest that although the mapping problems in circuit-switched and network-switched architectures are somewhat similar, the impact on the outcome can be dramatic if the subtle differences between these two architectures are not taken into account. For example, one of the main reasons that CastNet and similar algorithms like it perform so poorly when it comes to mapping task graphs for AsAP-like architectures is that their methodology relies greatly on the bandwidth information of the graph, whereas, in AsAP-like architectures bandwidth looses its significance in the problem. When the applications are given to CastNet with equal bandwidth on all communication channels (each edge of the task graph), the algorithm performs almost as it is making random choices at each step.

Another contributing factor which amplifies the effect of lacking bandwidth information is the size of the benchmark graphs. When dealing with large graphs, any early mistakes or bad choices at the beginning of the mapping process can lead to a great departure at the end. The reported results in Table II highlight the need for a new approach specific to solving the mapping problem for circuit-switch GALS architectures.

C. Impact of Core Failures

To demonstrate BAMSE ability to handle mappings in the existence of unavailable cores, the following three scenarios are provided. In one scenario all cores are available, and in the other two some number of cores are failing thus are avoided in the mapping process. The failing cores are selected randomly and kept the same in different applications. In the first failing case four cores are failing, which is almost 2% of AsAP2 cores, and In the second case nine are failing that is almost 5% of the cores. In each case we run BAMSE with 50 different configuration points as follows and choose the best results: WS = 30, 60, ..., 300 and MPC = 1, 3, ..., 9. The results are reported in Table III.

TABLE III

CORE FAILURES: RESULTS FOR LONGEST CONNECTION AND TOTAL NUMBER OF CONNECTIONS IN THREE DIFFERENT CORE FAILURE SCENARIOS ARE REPORTED

Application		0 core No Failure	4 cores 2% Failure	9 cores 5% Failure
Viterbi Decoder	Long Conn	1	1	1
	Total Conn	35	35	35
802.11a Base-band Receiver	Long Conn	3	3	3
	Total Conn	51	53	53
Small AES	Long Conn	2	2	2
	Total Conn	86	87	88
Large AES	Long Conn	3	3	4
	Total Conn	269	285	298
H.264/AVC Encoder	Long Conn	7	8	8
	Total Conn	310	343	340

The discrepancy between the results in Tables II and III in the case when none of the cores are failing is because of the difference between the number of times the algorithm is run with different configuration points (2400 in Table II versus 50 in Table III).

The discrete nature of mapping optimization problem combined with randomly selected failing cores makes it impossible to predict the outcome of these scenarios. Moreover, no other mapping technique handles such cases to be able to compare our results against their mappings. However, the closeness of the results between the constrained failing cases and non-failing cases proves the effectiveness of our approach.

D. Impact and Selection of Configuration Parameters

In this section, we discuss our experimental results on the impact of configuration parameters WS and MPC on the algorithm runtime and mapping quality. Since cost of the optimal mapping solution is not generally known, we measure the quality of a particular mapping with respect to the best mapping that we could find in the target parameter space. Namely, let the *Relative Cost* of a particular mapping be the normalized increase in its mapping cost, relative to the best mapping found in the explored configuration parameter space

$$Relative\ Cost^{app}_{(ws,mpc)} = \frac{mapping\ cost^{app}_{(ws,mpc)}}{min\ mapping\ cost^{app}} - 1. \quad (12)$$

Mapping cost is the multiobjective cost function presented in Section IV, with the priority order of longest connection (LC) and total connection (TC), respectively.

The small charts in Fig. 6 show the algorithm runtime at each configuration point (WS, MPC) in the parameter space.

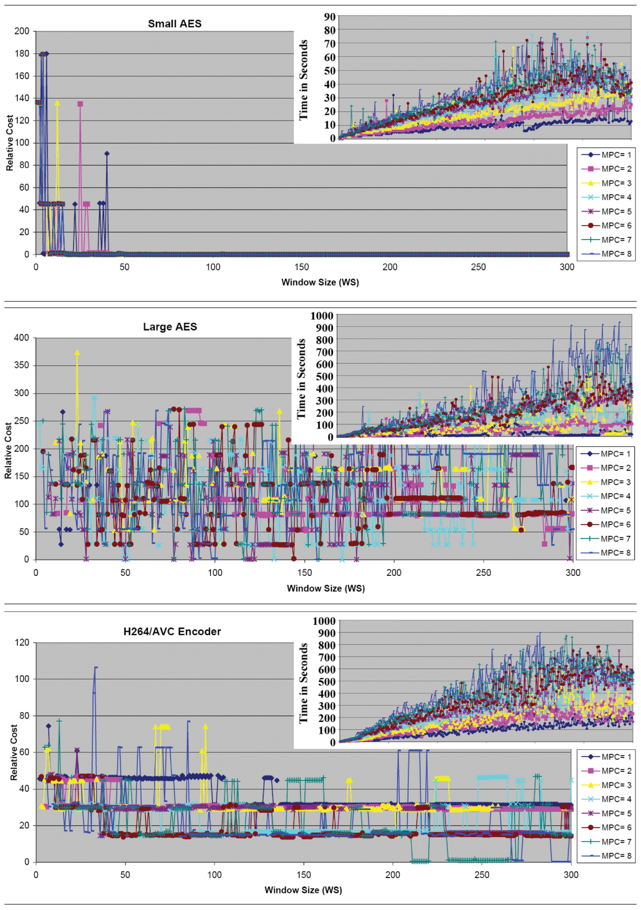


Fig. 6. Effect of increasing WS on the quality of mapping solutions for different *MPC* and benchmark applications. Relative cost numbers are calculated using (12), and quantify the normalized distance from the best known solution in the (WS, MPC) configuration parameter space. The effect of increasing WS on BAMSE runtime for different MPC values and benchmark applications is also given in the smaller charts. The recursive and application-dependent nature of link assignment is the primary reason for runtime fluctuations.

The charts for Viterbi and 802.11a baseband receiver applications are not presented in this figure for brevity. However, small AES application well represents these two applications as well since they all consist of relatively small task graphs compare to the other two bigger applications.

Although the general direction relationship in runtime with increase in both parameters is evident, the relationship is not strictly monotonic. This is primarily due to the difference in the time spent on link assignment, which tends to vary differently from one partial mapping to another. Within the link assignment algorithm, BAMSE recursively resolves conflicts as it allocates links to paths between connected cores. The conflict frequency and degree of congestion, which determine the frequency and depth of recursive calls, are highly benchmark dependent. Nevertheless from a high level viewpoint, the general trend is a direct linear relationship with WS and MPC parameters.

The aforementioned relationship trend between runtime and parameters holds fairly consistently for solution quality as well. The charts in Fig. 6 illustrate the relationship between the *Relative Cost* of generated mapping solutions, and the

TABLE IV

Mapping Quality and the Number of Acceptable Solutions ($N(\theta)$) Under Different Thresholds (θ): The Longest and Total Connection Values Correspond to the Lowest Quality Mapping that Met the Given Threshold

Application	Threshold (θ)	Long. Conn.	Total Conn.	#of sol. ($N(\theta)$).	95% conf. Trials
Viterbi decoder	0%	1	35	2154	2
	10%	1	35	2154	2
	50%	1	35	2154	2
802.11a baseband receiver	0%	3	51	162	43
	10%	3	53	1983	2
	50%	3	58	2385	1
Small AES	0%	2	86	2248	2
	10%	2	97	2334	1
	50%	3	100	2368	1
Large AES	0%	3	273	1	2280
	10%	3	328	10	621
	50%	4	343	155	45
H.264/AVC encoder	0%	6	336	4	1265
	10%	6	356	10	621
	50%	9	456	2150	2

window size (WS) for different MPC parameters. Similar to runtime, the data show a general trend of cost decrease with growth of configuration parameters, although the trend is not strictly monotonic, due to the discrete nature of the underlying search space.

For example, in case of small AES, all of (WS, MPC) configuration points larger than a small threshold result in the best (known) mapping. In such cases, if the WS are large enough (e.g., the break point of $WS > 40$ in small AES), the best mapping solution is universally found. The same pattern exists for the other small applications as well. In more complex applications on the other hand, the anticipated threshold values might be prohibitively large, in terms of computation time and memory requirements.

Consequently, an important objective is to select the configuration parameters such that a reasonably optimized solution, as compared to the best mapping solution that exists in the configuration parameter space, is generated. Clearly, one would like to accomplish this without explicit knowledge of the task graph structure, its relevant properties and without exhausting all points in the configuration parameter space. Given the discrete nature of the problem, it is not possible to guarantee optimality in the target parameter space without exhausting all of their possibilities. Thus, we expect the users to define their acceptance threshold for degradation in the quality of the final mapping solution. This decision can be made by users based on various criteria such as required throughput, energy budget, and the reasonable tool runtime.

We propose to randomly select different configuration parameters from the target space, run BAMSE with the selected parameters, and output the best solution found in the trials. As the number of trials increases, the probability of reaching an acceptable solution quickly improves. Specifically, the failure to find an acceptable solution requires failure in all of the trials, whose probability quickly decreases for realistic data sets, e.g., those illustrated in Fig. 7.

Specifically, let θ be the threshold for the acceptable degradation in mapping quality, and $N(\theta)$ represent the number of

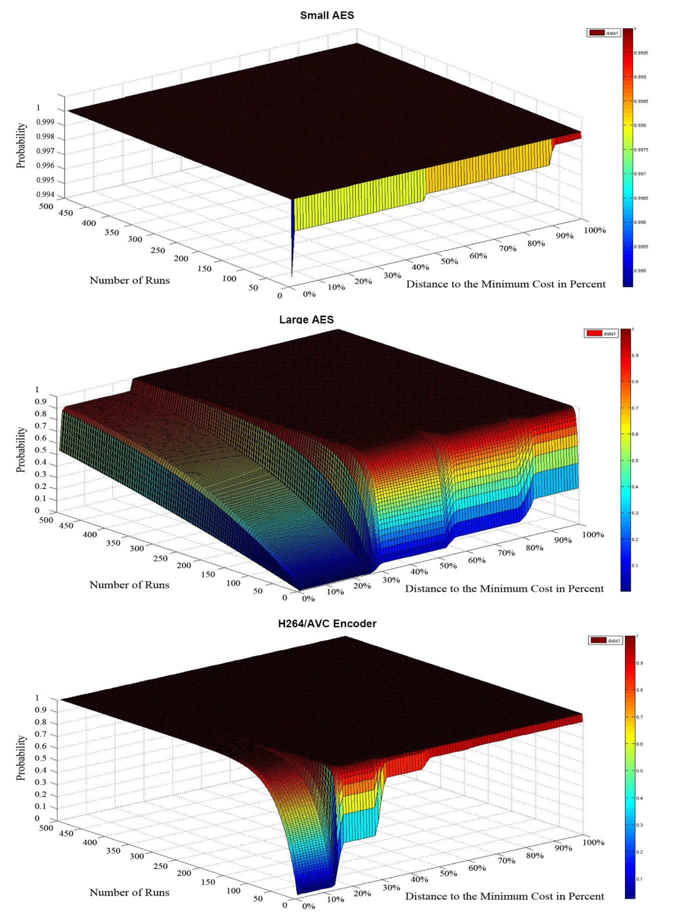


Fig. 7. Probability of generating an acceptable solution under a given acceptance threshold and different number of random parameter trials. 0% distance represents the solutions with the same cost as the best solution, and 100% distance represents the solutions twice as costly as the best solution.

solutions in the target parameter space, whose cost is not more than $1+\theta$ times larger than the best solution in the parameter space. The probability of not finding an acceptable solution within this acceptance level can be calculated using

$$P_\theta^k = 1 - \prod_{i=1}^{k} \frac{N-i+1-N(\theta)}{N-i+1} \tag{13}$$

where k is the number of trial runs, and N is the total number of points in the parameter space. Assuming a lower bound on the number of acceptable solutions in the space, $N(\theta)$, one can calculate the number of trials that lead to generation of an acceptable solution with the desired confidence level.

For three different acceptance thresholds θ, Table IV illustrates the number of acceptable solutions in the parameter space $N(\theta)$. For each solution, the mapping cost in terms of the longest and total connection are reported for comparison. The value $\theta = 0$ corresponds to the optimal solution in the target parameter space, which has 2400 points. Note that as the acceptance threshold is relaxed the number of acceptable solutions grows, and hence the number of trials required to find an acceptable solution with 95% confidence is decreased. Fig. 7 visualizes the relationship between number of runs (k), acceptance threshold (θ), and the probability of obtaining an acceptable solution for the benchmark applications.

As a design decision, the user can decide the number of trial runs and the range in which the tool should explore the parameters. Given the relatively short runtime of BAMSE, multiple runs are likely to be justified to improve the mapping quality in offline mapping scenarios. In the time constrained online mapping use cases, tool runtime tends to be a hard constraint, and hence, a small number of runs are appropriate.

VII. Conclusion

In this paper, we studied the unique characteristics of processor mapping problem in GALS based CMP architectures. We presented an algorithm called BAMSE, which generates high quality mappings of application task graphs for such platforms. Experiments show that the BAMSE mapping algorithm outperforms the time consuming manual mappings of real life existing applications up to 65% for the longest interprocessor communication link, and up to 19% for total length of the links, when the two criteria are used as primary and secondary optimization objectives, respectively. Furthermore, the reported results from employing one of the previously published mapping techniques specific to packet-switched NoC architectures (CastNet) on the presented benchmark set given the constraints of a packet-switched GALS architecture demonstrate the effectiveness of our approach in solving the mapping problem for GALS platforms. Additionally, BAMSE generates the mappings very fast, and it matches or beats solutions generated by solving ILP instances after 10 days of solver runtime.

References

[1] S. Tosun, "New heuristic algorithms for energy aware application mapping and routing on mesh-based nocs," *J. Syst. Architec.*, vol. 57, no. 1, pp. 69–78, 2011.
[2] K. Asanovic, R. Bodik, J. Demmel, T. Keaveny, K. Keutzer, J. Kubiatowicz, *et al.*, "A view of the parallel computing landscape," *Commun. ACM*, vol. 52, pp. 56–67, Oct. 2009.
[3] S. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, *et al.*, "An 80-tile 1.28tflops network-on-chip in 65 nm CMOS," in *Proc. ISSCC*, 2007, pp. 98–99.
[4] D. Truong, W. Cheng, T. Mohsenin, Z. Yu, A. Jacobson, G. Landge, *et al.*, "A 167-processor computational platform in 65 nm CMOS," *IEEE J. Solid-State Circuits*, vol. 44, no. 4, pp. 1130–1144, Apr. 2009.
[5] D. Bertozzi, A. Jalabert, S. Murali, R. Tamhankar, S. Stergiou, L. Benini, *et al.*, "Noc synthesis flow for customized domain specific multiprocessor systems-on-chip," *IEEE Trans. Parallel Distrib. Syst.*, vol. 16, no. 2, pp. 113–129, Feb. 2005.
[6] J. Dielissen, A. Rădulescu, K. Goossens, and E. Rijpkema, "Concepts and implementation of the Philips network-on-chip," *IP-based SoC Design*, 2003, pp. 1–6.
[7] J. Muttersbach, T. Villiger, H. Kaeslin, N. Felber, and W. Fichtner, "Globally-asynchronous locally-synchronous architectures to simplify the design of on-chip systems," in *Proc. 12th Annu. IEEE Int. ASIC/SOC Conf.*, 1999, pp. 317–321.
[8] U. Ogras, R. Marculescu, P. Choudhary, and D. Marculescu, "Voltage-frequency island partitioning for gals-based networks-on-chip," in *Proc. 44th ACM/IEEE DAC*, 2007, pp. 110–115.
[9] K.-C. Chang, J.-S. Shen, and T.-F. Chen, "Evaluation and design trade-offs between circuit-switched and packet-switched nocs for application-specific socs," in *Proc. 43rd Annu. DAC*, 2006, pp. 143–148.
[10] M. Hashemi, M. H. Foroozannejad, S. Ghiasi, and C. Etzel, "Formless: Scalable utilization of embedded manycores in streaming applications," *SIGPLAN LCTES*, vol. 47, no. 5, pp. 71–78, Jun. 2012.
[11] M. H. Foroozannejad, T. Hodges, M. Hashemi, and S. Ghiasi, "Postscheduling buffer management trade-offs in streaming software synthesis," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 17, no. 3, pp. 1–31, Jul. 2012.
[12] M. Hashemi and S. Ghiasi, "Versatile task assignment for heterogeneous soft dual-processor platforms," *IEEE Trans. Computer-Aided Design Integr. Circuits Syst.*, vol. 29, no. 3, pp. 414–425, Mar. 2010.

[13] M. Hashemi, M. H. Foroozannejad, and S. Ghiasi, "Throughput-memory footprint trade-off in synthesis of streaming software on embedded multiprocessors," *ACM Trans. Embed. Comput. Syst.*, vol. 13, no. 3, pp. 46:1–46:26, Dec. 2013.

[14] M. I. Gordon, W. Thies, M. Karczmarek, J. Lin, A. S. Meli, A. A. Lamb, *et al.*, "A stream compiler for communication-exposed architectures," in *Proc. Int. Conf. Architec. Supp. Program. Lang. Oper. Syst.*, 2002, pp. 291–303.

[15] W. Thies, J. Lin, and S. Amarasinghe, "Partitioning a structured stream graph using dynamic programming," in *Proc. 5th Workshop Media Stream. Processors*, 2003.

[16] S. Stuijk, T. Basten, M. C. W. Geilen, and H. Corporaal, "Multiprocessor resource allocation for throughput-constrained synchronous dataflow graphs," in *Proc. DAC*, 2007, pp. 777–782.

[17] C. Marcon, N. Calazans, F. Moraes, A. Susin, I. Reis, and F. Hessel, "Exploring noc mapping strategies: An energy and timing aware technique," in *Proc. Conf. DATE*, 2005, pp. 502–507.

[18] G. Ascia, V. Catania, and M. Palesi, "Multi-objective mapping for mesh-based noc architectures," in *Proc. Int. Conf. Hardw./Softw. Codesign Syst. Synth.*, 2004, pp. 182–187.

[19] S. Murali and G. De Micheli, "Bandwidth-constrained mapping of cores onto NoC architectures," in *Proc. Conf. DATE*, 2004, pp. 896–901.

[20] S. Murali, L. Benini, and G. de Micheli, "Mapping and physical planning of networks-on-chip architectures with quality-of-service guarantees," in *Proc. ASP-DAC*, 2005, pp. 27–32.

[21] J. Hu and R. Marculescu, "Energy- and performance-aware mapping for regular noc architectures," *IEEE Trans. Computer-Aided Design Integr. Circuits Syst.*, vol. 24, no. 4, pp. 551–562, Apr. 2005.

[22] K. Srinivasan and K. S. Chatha, "A technique for low energy mapping and routing in network-on-chip architectures," in *Proc. ISLPED*, 2005, pp. 387–392.

[23] S. Tosun, O. Ozturk, and M. Ozen, "An ilp formulation for application mapping onto network-on-chips," in *Proc. Int. Conf. AICT*, 2009, pp. 1–5.

[24] M. Foroozannejad, B. Bohnenstiehl, and S. Ghiasi, "Bamse: A balanced mapping space exploration algorithm for gals-based manycore platforms," in *Proc. 18th ASP-DAC*, 2013.

[25] A. T. Tran, D. N. Truong, and B. Baas, "A reconfigurable source-synchronous on-chip network for gals many-core platforms," *IEEE Trans. Computer-Aided Design Integr. Circuits Syst.*, vol. 29, no. 6, pp. 897–910, Jun. 2010.

[26] A.-H. Ghamarian, M. C. W. Geilen, S. Stuijk, T. Basten, A. J. M. Moonen, M. Bekooij, *et al.*, "Throughput analysis of synchronous data flow graphs," in *Proc. 6th Int. Conf. ACSD*, 2006, pp. 25–36.

[27] E. Cuthill and J. McKee, "Reducing the bandwidth of sparse symmetric matrices," in *Proc. 24th ACM Nat. Conf.*, 1969, pp. 157–172.

[28] E. W. Work, "Algorithms and software tools for mapping arbitrarily connected tasks onto an asychronous array of simple processors," M.S. thesis, Office Graduate Studies, Univ. California, Davis, CA, USA, Sep. 2007.

[29] A. Tran, D. Truong, and B. Baas, "A complete real-time 802.11a baseband receiver implemented on an array of programmable processors," in *Proc. 42nd Asilomar Conf. Signals, Syst., Comput.*, 2008, pp. 165–170.

[30] B. Liu and B. Baas, "A high-performance area-efficient AES cipher on a many-core platform," in *Proc. 45th ASILOMAR*, 2011, pp. 2058–2062.

[31] Z. Xiao, S. Le, and B. Baas, "A fine-grained parallel implementation of a h.264/avc encoder on a 167-processor computational platform," in *Proc. 45th ASILOMAR*, 2011, pp. 2067–2071.

Mohammad H. Foroozannejad (S'13) received the B.S. degree in computer engineering from Shahid Beheshti University, Tehran, Iran, in 2001, and the M.S. degree in computer engineering from the Department of Electrical and Computer Engineering, University of California, Davis, CA, USA, in 2011, where he is currently pursuing the Ph.D. degree in electrical and computer engineering.

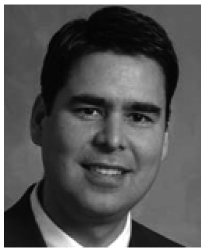
His current research interests include architecture and software development for low-power and high-performance computation systems.

Matin Hashemi (S'10–M'13) received the B.S. degree in electrical engineering from Sharif University of Technology, Tehran, Iran, in 2005, and the M.S. and Ph.D. degrees in electrical and computer engineering from the University of California, Davis, CA, USA, in 2008 and 2011, respectively.

Currently, he is an Assistant Professor of Electrical Engineering at Sharif University of Technology. His current research interests include design and optimization of embedded computing systems.

Alireza Mahini received the master's degree in computer systems architecture from Iran University of Science and Technology, Tehran, Iran, in 2006. Currently, he is pursuing the Ph.D. degree at the Islamic Azad University, Tehran, Iran.

He is currently a member of Academic Staff at Islamic Azad University, Gorgan branch, and the President of Gorgan SAMA College. His current research interests focus on distributed systems, high performance IP route lookup, network processor architecture, and network on chip.



Bevan M. Baas (M'99–SM'11) received the M.S. and Ph.D. degrees in electrical engineering from Stanford University, Stanford, CA, USA, in 1990 and 1999, respectively.

He was with Hewlett Packard's Computer Systems Division, Sunnyvale, CA, USA. He was also the second full-time employee after the founders at Atheros Communications, San Jose, CA. In 2003, he joined the Department of Electrical and Computer Engineering at the University of California, Davis, CA, where he is now an Associate Professor. His current research interests are include algorithms, architectures, arithmetic, circuits, and VLSI design for high-performance, energy-efficient, and area-efficient programmable and special-purpose computation.

Dr. Baas is a recipient of the National Science Foundation CAREER Award, the Best Paper Award at ICCD 2011, and several Best Paper Nominations. He also supervised the research that earned the Best Doctoral Dissertation Honorable Mention in 2013. From 2007 to 2012, he was an Associate Editor for the IEEE JOURNAL OF SOLID-STATE CIRCUITS, and has served on many conference committees, Co-chairships, and Guest Editorships. He is a fellow of NSF and NASA GSR.

Soheil Ghiasi (M'05–SM'10) received the B.S. degree from Sharif University of Technology, Tehran, Iran. in 1998, and the M.S. and Ph.D. degrees in computer science from the University of California, Los Angeles, CA, USA, in 2002 and 2004, respectively.

He is currently an Associate Professor of Electrical and Computer Engineering at the University of California, Davis, CA. His current research interests include architecture, design methodologies, and design automation techniques for embedded systems.

Dr. Ghiasi has served on the organizing and technical program committees of numerous conferences, and currently serves as an Associate Editor of the *Journal of Reconfigurable Computing*.