BAMSE: A Balanced Mapping Space Exploration Algorithm for GALS-based Manycore Platforms

Mohammad H Foroozannejad
Brent Bohnenstiehl
Soheil Ghiasi

Electrical and Computer Engineering
University of California, Davis
e-mail: {mhforoozan, bvbohnen, ghiasi}@ucdavis.edu

Abstract— We study the problem of mapping concurrent tasks of an application modeled as a data flow graph onto processors of a GALS-based manycore platform. We propose a mapping algorithm called BAMSE, which exploits the characteristics of streaming applications and the specifications of the target architecture to optimize the mapping solution. Different configuration parameters embedded into the algorithm enable one to strike a balance between scalability of the approach and the quality of generated solutions. Experiments with several real life applications show that our algorithm outperforms hand-optimized manual mappings up to 65% in terms of longest inter-processor communication link, and as high as 19% with respect to total length of the links, when the two criteria are used as primary and secondary optimization objectives, respectively. Additionally, our algorithm delivers superior mappings compared to ILP generated solutions after 10 days of solver runtime.

I. Introduction

While the conventional uniprocessor architectures have slowed in improving performance and power consumption in the past several years, different variations of chip multiprocessor (CMP) architectures have shown a promising approach to improve both factors [1, 2].

Network-on-Chip (NoC) systems have been developed in recent years to address the communication challenge in distributed memory platforms. While packet-switch NoCs [3, 4] have become the default choice of interconnect in the multicore space, Globally Asynchronous Locally Synchronous (GALS) paradigm [5, 6] offers a promising alternative for improving the communication speed and its energy dissipation by alleviating the complications of clock distribution in large CMP systems [2, 7].

Although different variations of domain-specific many-cores promise large gains in performance and energy efficiency, development of application software for their utilization remains a major challenge. Part of the difficulty lies in providing abstractions and methodologies for productive development of concurrent tasks that faithfully implement the application specification [8, 9]. Furthermore, there is a pressing need for tools that would efficiently map application concurrent tasks to platform resources. This paper presents our work and results on the latter category of challenges considering the requirements of a GALS communication paradigm.

Specifically, we study the problem of mapping concurrent tasks of a streaming application modeled as a data flow graph onto processors of a GALS-based CMP platform. We show that even though communication features of GALS architectures can potentially increase the performance of the system, processor mapping plays a major role in this achievement. We propose a mapping algorithm called BAMSE which exploits the specific characteristics of such systems. Although there exist a number of tools in the literature to address the mapping problem in general NoC architectures [10–12], to the best of our knowledge we are the first to consider limitations and requirements of GALS platforms in solving this problem.

BAMSE is an acronym for "BAlanced Mapping Space Exploration" which stresses the capability of the algorithm to strike a balance between tool runtime and quality of generated solutions. This feature of the algorithm becomes very important especially in online mapping scenarios in which, the tool run time for mapping is constrained. In design of the algorithm, we use a number of configuration parameters to efficiently steer the exploration engine on the time-quality tradeoff spectrum. These parameters also guarantee the scalability of the tool when dealing with a large number of application tasks and platform cores.

II. Target Platform/Applications

In this section we discuss the basic architectural specifications of AsAP2 processor as an example for GALS based CMP architectures and also the target applications discussed in this paper. Although our approach and discussions are generic in nature and applicable to other manycore GALS architectures, discussing this example before entering the details of the main approach can add more clarity and justification to some of the decisions we make in designing the algorithm.

AsAP2 [2] is an academic many-core GALS architecture processor and contains 167 programmable cores with a mesh inter connect 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, however, as it is discussed in [13] (also shown in figure 2), long communications highly affect the nominal frequency of the source core even if the volume of the communication is very low.

Figure 1 illustrates the architectural specifications of AsAP2 [2]. Figure 2 shows the effect of interconnect distance between communicating processors and the clock frequency of the source processor in this particular CMP platform [13]. The same clock used to supply the source processor is used for the

978-1-4673-3030-5/13/$31.00 ©2013 IEEE

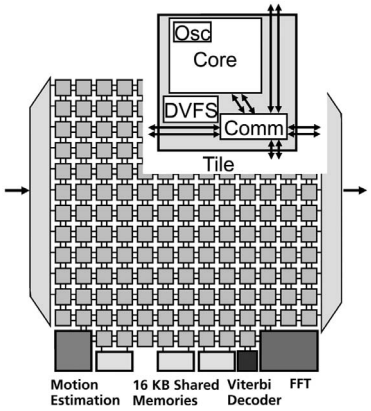

Fig. 1. Architectural specifications of AsAP2 [2].

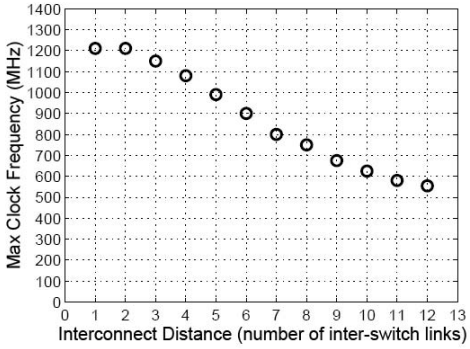

Fig. 2. Measured maximum clock frequencies for interconnect between AsAP2 processors over various interconnect distances [13].

communication regardless of the data bandwidth between the connecting cores. Therefore even a simple control signal slows down the source if it has to go too far to get to the receiver.

Another limiting factor in GALS NoC architectures is the restricted network resources. In circuit-switch architectures when a connection is made between two cores it cannot be used for any other connection, whereas in packet-switch architectures the limitation is on the total bandwidth of each link and not the number of physical connections. Even a simple control signal occupies the resources used for the connection for the entire run of the application. In AsAP2 there are only two bidirectional links between any two neighboring cores indicating the importance of the link assignment step in the resource allocation process. Sometimes finding any solution in mapping complex applications becomes the ultimate goal for the designer.

Minimizing the total communication distances, and the corresponding communication energy, is also desired as it is in regular packet-switch NoC architectures [10].

A significant group of embedded applications are characterized by the requirement to process a steady stream of input data as they are presented to the system. Such applications, generally referred to as streaming applications, are well modeled using various forms of data flow graph programming models and often targeted for parallel hardware platforms [14]. In this paper we use a generic data flow graph model to present the benchmark streaming applications used to evaluate the proposed technique. In section A.1 we discuss these applications in more details.

III. Problem Statement

Both application and hardware platform can be represented in the form of graphs:

$$\text{Task Graph: } G = \ < V, E > \tag{1}$$
$$\text{Hardware Graph: } H = \ < C, L, CAP_L > \tag{2}$$

In task graph G, V is the set of vertices which are known as tasks at the application level, and E is the set of edges which represents inter-task communication.

The hardware graph H consists of C, which represents a set of available cores on the chip, and L, which is a subset of $C \times C$. L models possible direct physical links between 'neighboring' cores. Here the assumption is that 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 pass the data from one to another for longer communications. Since each core has its own router, we assume that the connections are between cores themselves for simplicity. CAP_L is a function that assigns a capacity number to every member of L.

The solution consists of two sets; S and R:

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

All tasks must be assigned to cores, and a core can execute at most one task. Formally:

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

The $path_{ij}$ in R is a lean subset of links that connects v_i to v_j in the mapping solution. In this paper, we consider a path to be valid if only the length of the path ($|path_{ij}|$) is minimal (for example the Manhattan Distance in mesh architectures).

The capacity constraints are formulated as the following:

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

The objective is to find the mapping, which will give the best performance or energy. Since performance and energy are hard to estimate at the mapping level, we use a cost function of three mapping attributes as a proxy. Specifically, we use the attributes Longest Connection (LC), Total number of Connections (TC), and Area(A):

$$LC = \max_R \quad |path_{ij}| \tag{10}$$
$$TC = \Sigma_R \quad |path_{ij}| \tag{11}$$
$$A = Area(S, L) \tag{12}$$

LC denotes the length of the longest path in the mapping. TC is the total length of all paths in the mapping. A is a number that represents some geometrical notion of area occupied by the cores that are assigned a task. For example, it could be the bounding box area of such cores.

The mapping has to optimize a multi-objective cost function of attributes. In our work, we use the tuple (LC, TC, A) to denote the cost of a mapping. Comparison between mapping costs is lexicographical. Therefore, LC has the highest priority for optimization. In comparison of two mappings with equal LC, the one with smaller TC value would be considered to

have a smaller cost.

IV. BAMSE Algorithm

BAMSE is a constructive approach which incrementally maps the concurrent tasks of a task graph into the cores of the given hardware platform. The key idea is to arrange the concurrent tasks in a sequence called Task Sequence and read through this sequence to gradually construct the final mapping solution.

Given a list of promising partial mappings for a prefix of length k in the Task Sequence, the algorithm augments the partial mappings by mapping the $k+1$'th node in the sequence, effectively generating partial mappings for prefix of length $k+1$. In order to avoid exponential blowup of maintained partial mappings, the list of maintained partial mappings is trimmed at every iteration.

In the remainder of this section, we discuss these three key policies that make up BAMSE. Specifically, ordering of nodes in the task sequence (Node Selection); augmentation of partial mappings of prefix k to those of prefix $k+1$ (Core Selection); and judicious maintenance of a selected subset of partial mapping to avoid exponential growth of retained partial mappings (Mapping Selection).

A. Node Selection

The process in which the Task Sequence is generated is called "Node Selection". Since the ultimate goal is to place each node of the task graph as close as possible to its neighboring nodes, the Breadth First Search (BFS) seems to be a natural choice for generating the Task Sequence. In BFS, the immediate children of each node have the priority to be selected over farther nodes in the graph.

From different variations of BFS we use the same principle used in Cuthill-McKee algorithm [15] which heuristically tries to reduce the maximum distance (how many nodes are on the way) between the place of any parent and the place of its children in the sequence resulting from BFS. We use MDC for this distance which is abbreviated from "Maximum Distance to Children" in the remaining of this paper.

B. Core Selection

If a "Partial Mapping" is a mappings for a prefix of length k in the Task Sequence in which $k \neq |V|$, the task number $(k+1)$ in the sequence is called "Next Node" (v_{next}). Since in BFS the parents of each node are selected prior to the selection of the node itself, in any Partial Mapping there is at least one core (more than one in general) which is connected to the next node. We call the set which contains all such cores "Connected Cores" or "CC".

In order to assign a core to v_{next}, the core selection process nominates a few cores as being good candidates based on the length of the paths between each of them and the cores in $CC_{v_{next}}$. The set which contains these nominees is called "Potential Candidate Cores" of v_{next} ($PCC_{v_{next}}$).

To determine $PCC_{v_{next}}$ in each Partial Mapping, the "Neighboring Set" of each core in $CC_{v_{next}}$ is created level by level and is called $NS_{c_i}^{level_i}$. From each level to the next, the accepted distance is increased by one unit (one directed link) and thus the resulting set becomes larger. The intersection between all the neighboring sets for all cores in $CC_{v_{next}}$ determines the $PCC_{v_{next}}^{level_i}$ in each level. Figure 3 illustrates an example.

We can continue expanding the neighboring sets until the desired number of candidate cores are selected or the maximum number of levels is reached. This desired number which is in fact an input to the algorithm is called MPC or "Minimum number of Potential candidate Cores". In Figure 3 $MPC = 1, 2, 3, 4$ all result in the same number of candidates which is four, but $MPC = 5$ will result in ten potential candidates.

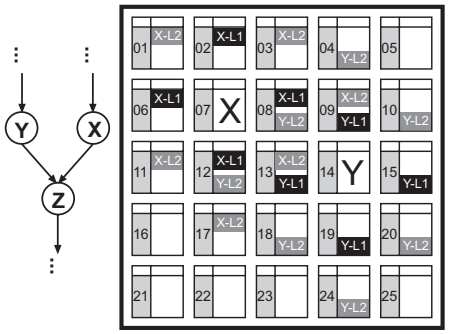

Fig. 3. In the given 5×5 mesh architecture, $v_{next} = Z$ in Read Sequence (RS) and $CC_Z = \{c7, c14\}$. The process of finding potential candidates is as follows: $NS_{c7}^{level_1} = \{c6, c2, c8, c12\}$, $NS_{c14}^{level_1} = \{c9, c15, c19, c13\}$, $PCC_Z^{level_1} = \{\}$. $NS_{c7}^{level_2} = \{c6, c2, c8, c12, c1, c3, c9, c11, c13, c17\}$, $NS_{c14}^{level_2} = \{c9, c15, c19, c13, c4, c8, c10, c12, c18, c20, c24\}$, $PCC_Z^{level_2} = \{c8, c9, c12, c13\}$.

Since the "Core Selection" process can not favor any of the candidates over the others in each level on its own, in this stage of the algorithm all potential candidates are accepted in each level. Assume i is the level in which $PCC_{v_{next}}^{level_i}$ meets MPC requirement, thus we have:

$$|PCC|_{v_{next}}^{level_{i-1}} < MPC \leq |PCC|_{v_{next}}^{level_i} \quad (13)$$

It is left to the "Mapping Selection" part of the algorithm to differentiate between these candidates based on the cost of assigning them to v_{next}.

C. Mapping Selection

Since v_{next} can be potentially mapped on each of the cores in $PCC_{v_{next}}$, therefore, the number of Partial Mappings in the iteration $k+1$ which are created from only of the Partial Mappings in iteration k is $|PCC_{v_{next}}| \geq MPC$ (according to equation 13). These Partial Mappings (created from all pervious Partial Mappings) are kept in a list called "Mapping List". To avoid state explosion, the Mapping List is kept as a sorted list based on the mapping cost of each Partial Mapping, in ascending order. The size of the list is also limited and called "Window Size" (WS). In a case that the current size of the Mapping List is equal to WS, each new Partial Mapping with a cost greater than the last Partial Mapping in the list is removed from the considered mappings; otherwise this new Partial Mapping is placed in the list based on its mapping cost, and the last mapping is removed from the list.

To further reduce the search space in this step of the algorithm and consequently elevate the quality of the solution, two other heuristics are proposed to better organize the mapping list:

C.1 Look Ahead Technique (LAT)

A moderately large WS is required to make sure that most of the good Partial Mappings which eventually lead to the desired solution at the end are kept in the list throughout the run time

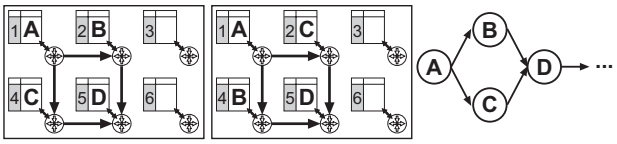

Fig. 4. Both Partial Mappings in the figure suggest the same outcome in continuum of the algorithm.

of the algorithm. Having a big WS also requires having more memory in the system. The memory constraint becomes problematic particularly when dealing with complex graphs.

One way to overcome this problem is to look ahead and predict the future quality of the present Partial Mappings at each step of the algorithm. To this end, in the "Look Ahead Technique" (LAT) a few more nodes are mapped for each Partial Mapping at each step, and the new mapping costs (called "secondary costs") are used to sort the current Partial Mappings.

Since the secondary costs are only used as an estimation for sorting and trimming the Mapping List and do not contribute to the actual costs, a fast and aggressive technique is suggested to map the extra nodes. In another words, the BAMSE algorithm is run for a few more steps with $WS = 1$ and $MPC = 1$ for each Partial Mapping in the list. The resulting mapping costs are then used to sort the Mapping List.

The number of extra nodes mapped at each step is called "Forward Number" (FN). The intuition we use to find a good FN is that all children of v_{next} need to be mapped in the LAT phase in order to have a sufficient estimation of the future mapping costs. The maximum distance between each node of the task graph and its children in the Task Sequence can be easily obtained by simply traversing the Task Sequence once and is called MDC (as explained in section A).

This technique effectively reduces WS. Although the LAT increases the run time of the algorithm in the asymptotic sense, dealing with a smaller WS eventually pays off and balances the additional time spent on the extra mappings.

C.2 Redundant Mapping Elimination Technique (RMET)

Although all Partial Mappings in the mapping list are different, not all of them offer completely different mappings in terms of the quality of the solution. For example, the two partial mappings shown in figure 4 are different but offer the same solution quality for any mapping constructed from any of them. We call these two partial mappings to be "Redundant".

Finding redundant solutions can be a difficult task on its own, however, there are easy ways to find a close enough supersets of them. One way is to compare the locations of the nodes in "terminal set" and the "current cost" of each Partial Mapping and eliminate the redundances. The terminal set is the set of all nodes in a Partial Mapping which still have connections to the nodes outside of this Partial Mapping (set $\{D\}$ in figure 4). If two Partial Mappings offer the same locations for all nodes in terminal set and also have the same current cost, they are considered redundant. Eliminating these Partial Mappings from the Mapping List allows us to keep track of a larger variety of Partial Mappings, avoiding wasting space in the WS on redundant ones.

D. Link Assignment

Due to limited network resources on many of the many-core platforms, not all mappings can offer valid link assignment between connected cores. In fact, there are cases where the link assignment becomes the most constrained criteria in the mapping. Therefore it is wise to keep track of used network resources and identify invalid mappings from the beginning. To this end, a simple XY link assignment algorithm is run on each Partial Mapping when a new node is mapped. In XY link assignment only forward paths for each pair of connected cores are considered. The distance between connected cores then exactly matches the Manhattan Distance between the two cores in mesh architectures.

A "Bookkeeping Table" keeps track of used link resources for each Partial Mapping. In each step of the algorithm, each new Partial Mapping inherits this table from its parent mapping and add the new assigned resources to the table. After assigning a path to an edge from the graph, the weight of the edge is subtracted from the capacity of all the links on the assigned path. The remaining capacity of each assigned link is also kept in the Bookkeeping Table. A valid Partial Mapping is a partial mapping in which all the remaining capacities in its Bookkeeping Table are bigger than or equal zero. Invalid Partial Mappings are eliminated from the Mapping List in each step of the algorithm.

V. Complexity vs. Quality

We start with some level of abstraction in analyzing the run time of the algorithm. Let us assume the run time of allocating one core to a task of the graph is T_{core} (including link assignment). For each partial mapping from the mapping list, $|PCC|$ (Potential Candidate Cores) number of cores are selected for mapping a new node from the graph. It is obvious that $|PCC|$ can not grow bigger than $|C|$ (total number of cores in the architecture). On the other hand we know from section IV that MPC is the minimum of $|PCC|$. At each step of the algorithm, there are maximum WS and minimum one Partial Mapping(s) in the mapping list (ML). Therefore we have:

$$MPC \leq PCC \leq |C| \tag{14}$$

$$1 \leq |ML| \leq WS \tag{15}$$

The number of nodes in the graph is $|V|$ and the above process is repeated for the mapping of each node. Therefore, using asymptotic operations O and Ω, the total run time of the main algorithm (T_{main}) is as follows:

$$T_{main} < O(WS \times |V| \times |C| \times T_{core}) \tag{16}$$

$$T_{main} > \Omega(MPC \times |V| \times T_{core}) \tag{17}$$

By increasing WS and MPC the run time of the algorithm increases, however not in the same way. Also more candidate mappings are explored; thus, intuitively better solutions are expected. In fact, if $MPC = |C|$ and $WS = |C|^{|V|}$ the entire search space is explored and the optimal solution is found.

As discussed in section C.1, LAT is an improvement technique over the main approach and is basically another run of BAMSE only for FN number of nodes with $WS = 1$ and $MPC = 1$. Therefore, following the same principles in calculating T_{main}, the run time of LAT (T_{LAT})is as follows:

$$T_{LAT} < O(FN \times |C| \times T_{core}) \tag{18}$$

$$T_{LAT} > \Omega(FN \times T_{core}) \tag{19}$$

Putting it all together, the total run time of the algorithm (T_{total}) would be the following. Note that RMET (sectionC.2) does not asymptotically affect the time if proper hashing tech-

TABLE I
BENCHMARK APPLICATION SET SPECIFICATIONS: D IS THE DEGREE OF THE GRAPH AND MDC IS THE MAXIMUM DISTANCE BETWEEN A PARENT AND ITS CHILDREN WHEN THE TASK GRAPH IS TRAVERSED USING BFS.

Application Name	# Nodes	# Edges	D	MDC
Viterbi Decoder	30	35	3	4
802.11a B.B. Rx.	25	40	6	9
Small AES	59	79	3	4
Large AES	137	176	6	8
H.264/AVC Encoder	115	165	7	24

niques are used.

$$T_{total} < O(FN \times WS \times |V| \times |C|^2 \times T_{core}^2) \quad (20)$$

$$T_{total} > \Omega(FN \times MPC \times |V| \times T_{core}^2) \quad (21)$$

Although increasing these parameters intuitively elevates the quality of the final solution, the quality of the solution is not entirely dependant on the size of these parameters. Refer to section VI for more discussion.

VI. EXPERIMENTAL EVALUATION

A. Setup

As it is discussed in section II AsAP2 is used as the target platform in this paper. More information about the benchmark applications and system setup is given in the following subsections:

A.1 Benchmark Applications

To evaluate the proposed technique we selected five different streaming kernels as our benchmarks. These benchmark applications have been proposed and manually mapped for AsAP2 in different academic papers. They include Viterbi Decoder [16], Wireless LAN 802.11a Baseband Receiver [17], two implementations of Advanced Encryption Standard (AES) encrypter [18], and H.264/AVC Encoder [19] kernels. These kernels frequently appear in many higher-level applications that are used in portable embedded systems.

Table I specifies the complexity of each application by showing the number of nodes, number of edges, D which is the degree of the graph (maximum fan-in/fan-out of nodes), and MDC (maximum distance between a parent and its children when the graph is traversed) in the task graph of each application.

A.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 for WS = 1, 2, 3, ..., 300 and MPC = 1, 2, ..., 8. Each of these configuration points (WS, MPC) represents a level of greediness/awareness characteristics of the algorithm. The goal is to find a balance between these greediness and awareness characteristics such that a fast high quality mapping solution is obtained.

The objective of the mapping is set to minimizing the same multi-objective cost function which was presented in section III with the priority order of Longest Connection (LC), Total Connection (TC), and Area. This decision is made in accordance with the characteristics and requirements of GALS-based CMP architectures. Area as the third objective did not make any significant difference in any of the benchmarks and is not presented in the results. The best case and average case for each individual application are calculated among these 2400 runs.

TABLE II
MANUAL MAPPING VS. BAMSE MAPPING: BOTH LONGEST CONNECTION AND TOTAL NUMBER OF CONNECTIONS ARE SHOWN IN THE TABLE. THE ILP* NUMBERS HERE ARE NOT THE OPTIMAL SOLUTIONS FOUND USING ILP. THESE RESULTS ARE FROM TERMINATING THE SOLVER AFTER 10 DAYS. THE ILP** ARE OPTIMAL, BUT A SMALLER INSTANCE OF THE HARDWARE IS GIVEN TO THE SOLVER TO HELP IT FINISH THE JOB (A MESH OF 6X6 CORES).

Application		LC	TC	Time
Viterbi Decoder	Manual	1	35	-
	BAMSE	1	35	1 (sec)
	ILP**	1	35	46 (hours)
802.11a B.B. Rx.	Manual	6	58	-
	BAMSE	3	51	13 (sec)
	ILP**	3	51	58 (hours)
Small AES	Manual	3	106	-
	BAMSE	2	86	2 (sec)
	ILP*	3	105	10 (days)
Large AES	Manual	5	254	-
	BAMSE	3	273	170 (sec)
	ILP*	5	328	10 (days)
H.264/AVC Encoder	Manual	17	353	-
	BAMSE	6	336	273 (sec)
	ILP*	7	288	10 (days)

The experiments are performed on a Unix PC with Intel Xeon CPU running at 3.07GHZ, 8192KB of cache, and 6GB of main memory.

B. Results

Table II compares the outcome of the BAMSE algorithm with the manual mapping for each application. The manual mappings are obtained after days of manual trial and error and are presented in the cited reference paper for each application. The improvement over the manual mapping concerning the first objective (LC) can be as high as 65% in the best case mappings. In most cases, there are improvements for the second objective (TC) as well. Note that reducing LC and TC are not necessarily the same criteria in terms of mapping cost. In fact, sometimes reducing one increases the other as a compromise. The reported time is the run time of the first occurrence of a minimum cost solution.

For comparison, we also generate ILP formulations for each application for only the mapping part of the problem (not for the link assignment). These simpler formulations allow use of an ILP solver in an attempt to find the lower bound on the results in a reasonable time period. The ILP formulations are not presented in this paper for brevity. The objectives assigned to the solver (CPLEX) are to minimize the Longest Connection (LC) and Total number of Connections (TC). The numbers in the table are the results of 10 days running (or less if it is able to finish the job) of such formulations on the same systems used for BAMSE.

Let the relative cost (Rel_Cost) in each case be defined as follows:

$$Rel\_Cost_{(ws,mpc)}^{app} = \frac{cost_{(ws,mpc)}^{app}}{mincost^{app}} \quad (22)$$

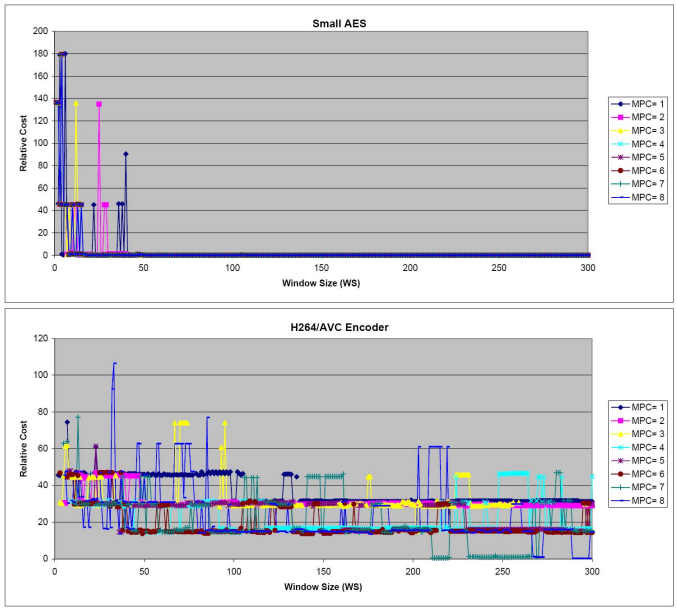

Fig. 5. The effect of increasing WS on the quality of the solution for different MPC values. Relative Cost numbers are calculated from equation 22.

The charts in figures 5 illustrate the relationship between the quality of mapping solutions (Rel_Cost) and the Window Size (WS) for different MPC parameters. The two applications in this figure (the small AES and H264) are chosen from the benchmark set for brevity to represent small and large application graphs. Although these charts show the non-monotonic behavior of the algorithm in both WS and MPC dimensions in general, they also suggest a non-monotonic decrease of the cost toward larger window sizes. In fact in the case of small AES, all configuration points of (WS, MPC) after a point in the chart offer a minimum cost mapping. It appears that if the WS is large enough (the break point of $WS > 40$ in small AES), finding the best mapping solution is guaranteed. The same pattern occurs in the 802.11a Baseband Rx application and even more quickly in the Viterbi Decoder application.

In complex applications on the other hand, this break point can be very large and unrealistic to reach in terms of computation time and memory requirements. However, by looking at the charts in figure 5 one can find the possibility of obtaining a minimum cost solution even before reaching this break point due to the non-monotonic nature of the problem. In other words, a substantial subset of the parameters space configurations leads to favorable results, therefore, a random trial appears to be practical. We leave this discussion to future work.

VII. Conclusion

In this paper we study the unique characteristics of processor mapping problem in GALS based CMP platforms. We show that the limitations and requirements of GALS platforms can affect the way mapping is performed. We also propose an algorithm called BAMSE based on these characteristics to find high quality mappings of application task graphs on 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 inter-processor 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.

References

[1] E. Lau, J. E. Miller, I. Choi, D. Yeung, S. Amarasinghe, and A. Agarwal, "Multicore performance optimization using partnervcores," *HotPar*, 2011.

[2] D. Truong, W. Cheng, T. Mohsenin, Z. Yu, A. Jacobson, G. Landge, M. Meeuwsen, C. Watnik, A. Tran, Z. Xiao, E. Work, J. Webb, P. Mejia, and B. Baas, "A 167-processor computational platform in 65 nm cmos," *Solid-State Circuits, IEEE Journal of*, vol. 44, no. 4, pp. 1130 –1144, april 2009.

[3] D. Bertozzi, A. Jalabert, S. Murali, R. Tamhankar, S. Stergiou, L. Benini, and G. De Micheli, "Noc synthesis flow for customized domain specific multiprocessor systems-on-chip," vol. 16, no. 2, pp. 113 – 129, feb. 2005.

[4] M. Millberg, E. Nilsson, R. Thid, and A. Jantsch, "Guaranteed bandwidth using looped containers in temporally disjoint networks within the nostrum network on chip," in *Design, Automation and Test in Europe Conference and Exhibition, 2004. Proceedings*, vol. 2, feb. 2004, pp. 890 – 895 Vol.2.

[5] 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 *ASIC/SOC Conference, 1999. Proceedings. Twelfth Annual IEEE International*, 1999, pp. 317 –321.

[6] U. Ogras, R. Marculescu, P. Choudhary, and D. Marculescu, "Voltage-frequency island partitioning for gals-based networks-on-chip," in *Design Automation Conference, 2007. DAC '07. 44th ACM/IEEE*, june 2007, pp. 110 –115.

[7] 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 *Proceedings of the 43rd annual Design Automation Conference*, ser. DAC '06, 2006, pp. 143–148.

[8] 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.

[9] 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.

[10] J. Hu and R. Marculescu, "Energy- and performance-aware mapping for regular noc architectures," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 24, no. 4, pp. 551 – 562, april 2005.

[11] G. Ascia, V. Catania, and M. Palesi, "Multi-objective mapping for mesh-based noc architectures," in *Hardware/Software Codesign and System Synthesis, 2004. CODES + ISSS 2004. International Conference on*, sept. 2004, pp. 182 – 187.

[12] K. Srinivasan and K. S. Chatha, "A technique for low energy mapping and routing in network-on-chip architectures," in *Proceedings of the 2005 international symposium on Low power electronics and design*, ser. ISLPED '05, 2005, pp. 387–392.

[13] A. T. Tran, D. N. Truong, and B. Baas, "A reconfigurable source-synchronous on-chip network for gals many-core platforms," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 29, no. 6, pp. 897 –910, june 2010.

[14] K. Asanovic, R. Bodik, J. Demmel, T. Keaveny, K. Keutzer, J. Kubiatowicz, N. Morgan, D. Patterson, K. Sen, J. Wawrzynek, D. Wessel, and K. Yelick, "A view of the parallel computing landscape," *Commun. ACM*, vol. 52, pp. 56–67, October 2009.

[15] E. Cuthill and J. McKee, "Reducing the bandwidth of sparse symmetric matrices," in *Proceedings of the 1969 24th national conference*, ser. ACM '69, 1969, pp. 157–172.

[16] 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, University of California, Davis*, September 2007.

[17] A. Tran, D. Truong, and B. Baas, "A complete real-time 802.11a baseband receiver implemented on an array of programmable processors," in *Signals, Systems and Computers, 2008 42nd Asilomar Conference on*, oct. 2008, pp. 165 –170.

[18] B. Liu and B. Baas, "A high-performance area-efficient aes encipter on a many-core platform," *to appear in the IEEE Asilomar Conference on Signals, Systems and Computers (ACSSC), Pacific Grove, CA*, November 2011.

[19] Z. Xiao, S. Le, and B. Baas, "A fine-grained parallel implementation of a h.264/avc encoder on a 167-processor computational platform," *to appear in the IEEE Asilomar Conference on Signals, Systems and Computers (ACSSC), Pacific Grove, CA*, November 2011.