Towards Exploring Vast MPSoC Mapping Design Spaces using a Bias-Elitist Evolutionary Approach

Wei Quan[†,‡]

Andy D. Pimentel[†]

[†]Informatics Institute
University of Amsterdam
The Netherlands
{w.quan,a.d.pimentel}@uva.nl

[‡]School of Computer Science
National University of Defense Technology
Hunan, China
quanwei02@gmail.com

Abstract—**The problem of optimally mapping a set of tasks onto a set of given heterogeneous processors for maximal throughput has been known, in general, to be NP-complete. Previous research has shown that Genetic Algorithms (GA) typically are a good choice to solve this problem when the solution space is relatively small. However, when the size of the problem space increases, classic genetic algorithms still suffer from the problem of long evolution times. To address this problem, this paper proposes a novel bias-elitist genetic algorithm that is guided by domain-specific heuristics to speed up the evolution process. Experimental results reveal that our proposed algorithm is able to handle large scale task mapping problems and produces high-quality mapping solutions in only a short time period.**

I. Introduction

Heterogeneous MPSoC platforms have in recent years received much attention due to their capability of providing good performance and energy consumption trade-offs [9]. For heterogeneous MPSoC systems, the task mapping problem – consisting of assigning a set of application tasks to processors and binding communications between tasks to communication channels or memories in the system – plays a crucial role in achieving high performance. Many heuristic algorithms exist for this task mapping problem, which can roughly be divided into two categories: the ones that assign one task at a time like Minimum Execution Time (MET) or Minimum Completion Time (MCT) [3] and the algorithms that map all the tasks at once like Simulated Annealing [11] or Genetic Algorithms [1]. Comparing these two classes of algorithms, the former category of algorithms usually has lower algorithmic complexity, which means a shorter computing time, but they also produce poorer results. For the second category of task mapping algorithms, several investigations [3], [4], [13], [6] have shown that Genetic Algorithms (GA) can consistently generate efficient mapping solutions, also in comparison to alternative heuristic search methods like Simulated Annealing (SA), in a relatively short time period. However, for large problem sizes (i.e., search spaces), GAs will typically suffer from large computational costs as a significant number of solution evaluations are needed to find good solutions [16]. Therefore, it is essential to develop effective pruning techniques that can optimize the search process, allowing the design space exploration (DSE) algorithms to explore larger design spaces.

This paper introduces a new GA-based mapping DSE algorithm that allows for effectively pruning the search space in order to reduce the search time. To this end, the algorithm aims at optimizing the genetic operators in the GA that take care of deriving new individuals – representing design points – from the old individuals during search iterations. If the operators can be optimized such that they only generate a small set of chromosomes that has a high probability of containing the optimal or near optimal solutions, then the search time for a good result can be greatly reduced. In this paper, we hypothesize that such an optimization of the genetic operators is possible through the exploitation of domain knowledge as captured by means of heuristics.

Based on this hypothesis, we propose a novel *bias-elitist genetic algorithm* in which the genetic operators have been optimized using application domain knowledge as captured by means of heuristics. We will show that our algorithm is able to find high-quality mapping solutions for applications that contain a large number of tasks, and it will do so in much shorter time frames as compared to a range of other well-known algorithms.

The remainder of this paper is organized as follows. Section II gives some prerequisites for this paper. Section III provides a detailed description of our bias-elitist genetic algorithm. Section IV introduces the experimental environment and presents the results of our experiments. Section V discusses related work, after which Section VI concludes the paper.

II. Prerequisites

A. Target Applications and Architectures

In this paper, we target the multimedia application domain. For this reason, we use the Kahn Process Network (KPN) model of computation [8] to specify application behaviour since this model of computation fits well to the streaming behaviour of multimedia applications. Under this model, an application can be represented as a directed graph $KPN = (P, F)$ where P is the set of processes (tasks) p_i in the application and $f_{ij} \in F$ represents the FIFO channel between two processes p_i and p_j.

The target architectures of this work are heterogeneous MPSoC systems in which each processor may have different computational characteristics, making the task mapping problem more complex. The architecture can be modeled as a graph $MPSoC = (PE, C)$, where PE is the set of processing elements used in the architecture and C is a multiset of pairs $c_{ij} = (pe_i, pe_j) \in PE \times PE$ representing a communication channel (like Bus, NOC, etc.) between processors pe_i and pe_j.

Combining the definition of application and architecture models, the computation cost of task (process) p_i on processing element pe_j is expressed as T_i^j and the communication cost between tasks p_i and p_j on channel c_{xy} is $C_{ij}^{c_{xy}}$. Here, c_{xy} represents the communication channel between processor pe_x and pe_y where tasks p_i and p_j are mapped onto respectively.

B. Simulation Framework

In our work, we deploy the open-source Sesame system-level MPSoC simulator [14] to evaluate the fitness of mappings. The Sesame modeling and simulation environment facilitates efficient performance analysis of embedded (media) systems architectures.

Although a Sesame-based simulation of each individual mapping to evaluate its fitness only takes a few seconds, the total evaluation time for solving large task mapping problems may grow to an unacceptable level. This underlines the need of reducing the evolution time of the genetic algorithm that searches the mapping space.

III. Bias-elitist Genetic Algorithm

Our bias-elitist GA combines a form of elitism as found in classic elitist GAs with the concept of a domain knowledge guided GA such as from [1]. It tries to find a task mapping for the target application(s) on a heterogeneous MPSoC system with the objective of maximizing the throughput. The details of our domain knowledge guided GA will be explained in the following subsections.

A. Encoding

Each mapping solution is encoded as a string of integers. The tasks of the target application(s) are arranged in the chromosome according to the topological order in the application KPN. Each gene in the chromosome represents a unique identifier of the processors in the MPSoC system (i.e., denoting the processor the task is mapped on).

B. Fitness Function

In our task mapping problem, as analytical fitness evaluation approaches typically are not capable of accurately evaluating the throughput of applications when both resource contention and task communication are considered, we deploy the open-source Sesame system-level MPSoC simulator [14] to accurately evaluate the fitness of each chromosome, i.e., mapping, in the population.

C. Selection

During each successive generation of the GA, a proportion of the existing population is selected to breed a new generation. Our algorithm uses a roulette wheel selection method in which the best chromosomes are more likely to be selected but the poorer chromosomes also have a small chance to be picked.

To control the population size in each generation, we use a strategy in which the best chromosome from the current population and $n-1$ chromosomes from the newly generated population are selected as the n survived individuals to breed the next new generation. The rationale behind this is that we aim at increasing the diversity of chromosomes in the mapping space that will be searched by keeping as few as possible old individuals in the new population. Therefore, in contrast to a general elitist GA, where the elitists in each generation will survive in the next generation, our GA only preserves the best individual in each generation.

D. Genetic operators

To generate a new generation from the selected chromosomes, two genetic operators – crossover and mutation – are applied. In our algorithm, we have improved the mutation operator so that the algorithm can more quickly find better solutions. For the crossover operator, which produces a new pair of chromosomes from a selected pair of chromosomes, we apply a standard one-point crossover. We have chosen this operator because it is simple and produces similar results compared with other crossover operators.

```
input   : $C$ (old chromosome)
output  : $C^*$ (new chromosome)
1  $PU$ = $pusage(C)$;
2  $x$ = index of processor with $max(PU)$;
3  for task $p_i$ mapped onto processor $pe_x$ do
4      for processor $pe_y$ different than $pe_x$ do
5          $C'$ = migrate $p_i$ from processor $pe_x$ to $pe_y$;
6          $PU'$ = $pusage(C')$;
7          if $max(PU') <= max(PU)$ then
8              $MBF.append(B_i^{xy})$;
9          end
10     end
11 end
12 if array MBF is not empty then
13     $p_k, pe_k$ = task and target processor with maximal migration benefit
       ($max(MBF)$);
14     $C^*$ = migrate $p_k$ from processor $pe_x$ to $pe_k$;
15     goto step 1, start with the new mapping $C^*$;
16 else
17     if no new mapping found in the previous steps then
18         for processor $pe_y$ different than $pe_x$ do
19             $C'$ = switch the tasks mapped onto $pe_x$ and $pe_y$;
20             $PU'$ = $pusage(C')$;
21             if $max(PU') <= max(PU)$ then
22                 $SBF.append(max(PU'))$;
23             end
24         end
25         if array SBF is not empty then
26             $pe_k$ = processor with $min(SBF)$;
27             $C^*$ = switch the tasks mapped onto $pe_x$ and $pe_k$;
28         else
29             shuffle the order of tasks in chromosome;
30             $C^*$ = generate new mapping using the MCT algorithm based on
               the shuffled task order;
31         end
32     end
33 end
34 return $C^*$
```

Algorithm 1: Heuristic guided mutation

The mutation operator is an essential part of our GA. It allows the GA to search new areas in the solution space. In our algorithm, we deploy a heuristic guided mutation operator that optimizes the mappings using domain knowledge. More specifically, the mutation operator considers the affinity of tasks with respect to processors, the communication cost between tasks, and the differences of processor workloads. The details of our mutation operator are outlined in Algorithm 1. By applying the mutation operation, a new chromosome will be derived through one of the following three approaches: task migration (lines 1-15), processor switching (lines 18-27) or a Minimum Completion Time (MCT) algorithm (lines 29-30).

At the beginning of the mutation, the task migration method will be used to find a new chromosome based on the input chromosome. In this process, the usage of each processor U_k under a given task mapping is calculated by equation 1 in the function at line 1. Lines 3-11 of Algorithm 1 try to find a task (among the tasks mapped onto the most heavily loaded processor) that has a maximal "migration benefit" under the condition of line 7. This task migration benefit, with regard to task p_i migrated from processor pe_x to pe_y, is labeled as B_i^{xy}. It is calculated by equation 2 where M_i^x and M_i^y represent the cost of p_i on pe_x and pe_y respectively. Here, the cost not only considers the task computation time but also the accumulated communication costs of the task in question.

$$U_k = \sum_{p_i \mapsto pe_k, p_j \mapsto pe_y} (T_i^k + C_{ij}^{c_{ky}}) \tag{1}$$

where $a \mapsto b$ implies that a is mapped onto b.

$$B_i^{xy} = M_i^x - M_i^y \tag{2a}$$

$$M_i^t = T_i^t + \sum_{\substack{f_{ij} \in F, c_{tk} \in C \\ p_i \mapsto pe_t, p_j \mapsto pe_k, f_{ij} \mapsto c_{tk}}} C_{ij}^{c_{tk}} \tag{2b}$$

If a task can be found for migration after the steps in lines 3-11, lines 13-14 in Algorithm 1 will generate a new mapping by migrating this task to the corresponding target processor. Subsequently, the above process is repeated – using the new mapping as input – until no new mapping can be found anymore.

However, if the above task migration approach cannot find a new chromosome, then the processor switching method will be applied to the input chromosome. As shown in lines 18-27 in Algorithm 1, the new chromosome will be generated by exchanging the tasks mapped onto the heaviest loaded processor with the tasks mapped onto the processor which satisfies the conditions on line 21 and line 26 (the processor that will maximally reduce the value of $max(PU)$ by processor switching).

In the case that no new mapping can be found by using any of the two previous approaches, a heuristic-based random mutation operator will be applied. A totally new chromosome, which means all the genes in the chromosome are different from the ones in the input chromosome, might be generated in this approach. The heuristic used for generating a new chromosome is the Minimum Completion Time (MCT) algorithm. The MCT algorithm assigns each task, in arbitrary order, to the processor with the minimum expected completion time for that task [2]. Different task assignment orders will produce different mapping results. Therefore, each time before generating a new chromosome using MCT, the task order in the chromosome is shuffled. Consequently, different well-balanced chromosomes will be added to the new population of our GA. This helps our GA to explore the mapping space with more gene diversity and prevents our GA from getting stuck in a local minimum.

E. Termination

With respect to the stopping conditions for our GA, two conditions are used: (1) if the best solution has not changed after a pre-defined number of generations, then our GA will terminate automatically and (2) a maximum number of generations is adopted to guarantee that the evolution process will stop.

IV. Experiments

For our experiments, we have selected a real multi-media application to investigate various aspects of our GA: a MP3 decoder consisting of 27 application processes (tasks). The target architecture considered in our experiments consists of 5 heterogeneous processors and 1 IO processor (for IO tasks). These processors are connected via a bus to a shared memory. In the MP3 task mapping problem, the total number of possible mapping solutions is $2.98 * 10^{17}$. Our Bias-Elitist Genetic algorithm (BEG) and several other algorithms will be used to explore this vast solution space to find the (near) optimal mapping with the objective to maximize the throughput.

For the purpose of comparison, two other GA-based mapping algorithms are studied as well: a general Elitist Genetic (EG) algorithm [5] and a Genetic Algorithm with a 3-Step Mutation (GA3SM) [1]. For those genetic algorithms (BEG, EG and GA3SM), the parameters in the experiments of single-application task mapping are listed in Table I. The parameters of each GA are optimized for each experiment. Notice that the parameter of mutation probability used in our experiments is a chromosome-level concept[1]. It differs from the mutation probability used in typical GAs which is considered at the gene level[2] and is usually small (< 0.1). In our experiments, the gene-level mutation probability only exists in the EG algorithm and its value is 0.05. For the purpose of a fair comparison, the same randomly generated initial population is provided to the EG and BEG algorithms. For the GA3SM algorithm, the initial population is derived by replacing the worst individual in the randomly generated initial population with the result of the Min-Min heuristic. This is according to the original GA3SM algorithm. The results of all experiments have been averaged over 10 execution runs to deal with the stochastic behaviour of the GAs. For all experiments, we have used a PC with a 2.93GHz Intel Core i7 CPU.

Table I: Parameters of genetic algorithms

Parameter	Experiment 1 all GAs	Experiment 2 all GAs
Initial pop. size	8	128
Generation pop. size	8	128
Crossover prob.	0.7	0.7
Mutation prob.	0.8	0.8
Max. # of generations	128	128

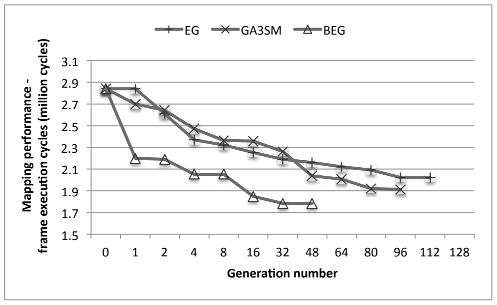

Figure 1: The convergence behavior of each GA with a small population size.

In the first experiment, we compare our BEG algorithm to the (Sesame-based) EG and GA3SM algorithms on three aspects: (1) the quality (frame execution time) of the final mapping solution, (2) the algorithm execution time and (3) the convergence behavior of an algorithm.

Table II shows the quality of the final mappings derived from the different algorithms as well as the algorithm execution cost of the search algorithms. From the results of Experiment 1 in Table II, we can see that our BEG algorithm can produce much better solutions than the other GAs. Our BEG algorithm also takes less time to find the final mapping solution as compared to the other two algorithms. The reason for this is that our BEG algorithm can converge much faster than the other two GAs, as shown in Figure 1. This graph shows the convergence behavior of the execution run for each algorithm that produced the best final solution out of 10 runs.

In the second experiment, we studied the behaviour of each

[1]Chromosome-level mutation probability: the likelihood of mutating a particular chromosome.

[2]Gene-level mutation probability: the likelihood of mutating each gene (bit) of a chromosome in mutation.

GA using a larger search space by increasing the population size. The results are shown in Table II of Experiment 2. From the results, we can see that our BEG algorithm again outperforms the other algorithms with respect to the quality of the final mapping solution. Compared with the first experiment, each algorithm produces better mapping results. However, these mapping solution improvements come at the expense of a higher exploration time.

Considering this and the previous experiment together, we can see that our BEG algorithm always yields the best solutions, and on top of this, it can already find a good mapping result in a relatively short time by reducing the population size. The EG and GA3SM algorithms, on the other hand, require algorithm execution times that are about an order of magnitude higher to find similar good mapping results as our algorithm. The interested reader is referred to [15] for a more extensive explanation and a more rigorous experimental evaluation of our BEG.

V. Related Research

In recent years, much research has been performed in the area of task mapping for embedded systems.

In the context of static mapping performance optimization, some classic algorithms such as Simulated Annealing (SA) [12], Genetic Algorithm (GA) [4], [1], [13], Tabu Search [10] and Integer Linear Programming (ILP) [7] have been proposed. Among these algorithms, the GA is considered to be a good mapping algorithm because it can obtain a good result in a short time period [3]. There are different forms of GAs that can be used to obtain a better solution. For instance, [13] uses eight heuristics to initialize the GA population for getting better solutions. Alexandrescu et al. [1] propose a GA with a 3-Step Mutation which aims at increasing the solution's convergence rate by using a combination of methods to mutate a chromosome. In contrast to these GAs, our domain-knowledge guided GA is proposed to solve the large scale task mapping problems on the heterogeneous MPSoC systems where the computation and communication cost of tasks and resource contention in the system are carefully considered in the evolution process.

VI. Conclusion

Even though genetic algorithms have a proven track record in solving such problems, these algorithms still need to be carefully designed in order to obtain high-quality solutions in an acceptable time. In this paper, we have proposed a bias-elitism genetic (BEG) algorithm where the mutation operator has been optimized for our task mapping problem. More specifically, we have added domain-specific heuristics as well as a Minimum Completion Time heuristic to the mutation operator. In addition, the selection method in our genetic algorithm has also been tailored for the purpose of finding a good mapping in a short time period. In various experiments, different state-of-the-art algorithms have been compared to our BEG algorithm. These experiment results clearly confirm the effectiveness of our algorithm.

Table II: Comparison of final mapping quality in Frame Execution Time (cycles, the smaller the better) and algorithm execution cost (seconds) of GAs.

	Experiment 1			**Experiment 2**		
	EG	GA3SM	BEG	EG	GA3SM	BEG
Max. FET	2342502	2218522	1979684	2030686	1953746	1821842
Min. FET	2022538	1911142	1784318	1862988	1768808	1762048
Average FET	2197809	2064753	1885810	1943892	1847738	1779620
Max. cost	4897	3074	2160	42729	55002	47824
Min. cost	2145	1637	875	21342	27814	29778
Average cost	3217	2245	1567	33381	43274	39496

References

[1] A. Alexandrescu, I. Agavriloaei, and M. Craus. A genetic algorithm for mapping tasks in heterogeneous computing systems. In *System Theory, Control, and Computing (ICSTCC), 2011 15th International Conference on*, pages 1–6, 2011.

[2] R. Armstrong, D. Hensgen, and T. Kidd. The relative performance of various mapping algorithms is independent of sizable variances in run-time predictions. In *In 7th IEEE Heterogeneous Computing Workshop*, pages 79–87, 1998.

[3] T. D. Braun, H. J. Siegel, N. Beck, L. L. Bölöni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen, and R. F. Freund. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *J. Parallel Distrib. Comput.*, 61(6):810–837, June 2001.

[4] J. Choi, H. Oh, S. Kim, and S. Ha. Executing synchronous dataflow graphs on a spm-based multicore architecture. In *Proceedings of the 49th Annual Design Automation Conference*, DAC '12, pages 664–671, New York, NY, USA, 2012. ACM.

[5] S. Dey and S. Majumder. Task allocation in heterogeneous computing environment by genetic algorithm. In *Distributed Computing*, volume 2571 of *Lecture Notes in Computer Science*, pages 348–352. Springer Berlin Heidelberg, 2002.

[6] C. Erbas, S. Cerav-Erbas, and A. Pimentel. Multiobjective optimization and evolutionary algorithms for the application mapping problem in multiprocessor system-on-chip design. *Evolutionary Computation, IEEE Transactions on*, 10(3):358–374, 2006.

[7] H. Javaid and S. Parameswaran. A design flow for application specific heterogeneous pipelined multiprocessor systems. In *Proceedings of the 46th Annual Design Automation Conference*, DAC '09, pages 250–253, New York, NY, USA, 2009. ACM.

[8] G. Kahn. The semantics of a simple language for parallel programming. In *Information processing*, pages 471–475. North Holland, Amsterdam, Aug 1974.

[9] R. Kumar, D. M. Tullsen, P. Ranganathan, N. P. Jouppi, and K. I. Farkas. Single-isa heterogeneous multi-core architectures for multi-threaded workload performance. In *Proceedings of the 31st annual international symposium on Computer architecture*, ISCA '04, pages 64–75, Washington, DC, USA, 2004. IEEE Computer Society.

[10] S. Manolache, P. Eles, and Z. Peng. Task mapping and priority assignment for soft real-time applications under deadline miss ratio constraints. *ACM Trans. Embed. Comput. Syst.*, 7(2):19:1–19:35, Jan. 2008.

[11] H. Orsila. *Optimizing Algorithms for Task Graph Mapping on Multiprocessor System on Chip*. PhD thesis, Tampere University of Technology, Finland, 2011.

[12] H. Orsila, T. Kangas, E. Salminen, T. D. Hämäläinen, and M. Hännikäinen. Automated memory-aware application distribution for multi-processor system-on-chips. *J. Syst. Archit.*, 53(11):795–815, Nov. 2007.

[13] A. J. Page, T. M. Keane, and T. J. Naughton. Multi-heuristic dynamic task allocation using genetic algorithms in a heterogeneous distributed system. *Journal of parallel and distributed computing*, 70(7):758–766, July 2010.

[14] A. D. Pimentel, C. Erbas, and S. Polstra. A systematic approach to exploring embedded system architectures at multiple abstraction levels. *IEEE Trans. Computers*, 55(2):99–112, 2006.

[15] W. Quan and A. D. Pimentel. Exploring Task Mappings on Heterogeneous MPSoCs using a Bias-Elitist Genetic Algorithm. http://arxiv.org/abs/1406.7539.

[16] A. K. Singh, M. Shafique, A. Kumar, and J. Henkel. Mapping on multi/many-core systems: survey of current and emerging trends. In *Proceedings of the 50th Annual Design Automation Conference*, DAC '13, pages 1:1–1:10, New York, NY, USA, 2013. ACM.