Dimensioning the Heterogeneous Multicluster Architecture via Parallelism Analysis and Evolutionary Computing

Falko Guderian, Rainer Schaffer, and Gerhard Fettweis
Vodafone Chair Mobile Communications Systems
Technische Universität Dresden, 01062 Dresden, Germany
E-mail:{falko.guderian, rainer.schaffer, fettweis}@ifn.et.tu-dresden.de

Abstract

In the near future, embedded systems containing hundreds of processing elements running multiple concurrent applications will become a reality. The heterogeneous multicluster architecture enables to cope with the challenging hardware/software requirements presented by such systems. This paper shows principles and optimization of multicluster dimensioning aiming at an appropriate distribution of applications onto clusters containing different types of processing elements. The approach represents an initial exploration phase efficiently finding a suitable multicluster configuration in the large design space. Hence, results should be further refined by more accurate but less time-efficient simulation-based techniques. As the starting point, a parallelism value matrix is analytically extracted describing application mappings independently on the architecture and scheduling. A genetic algorithm (GA) and a mixed-integer linear programming (MILP) approach solving the dimensioning problem are introduced and compared. Both solutions use the parallelism value matrix as input. Scalability results show that the GA generates results faster and with a satisfactory quality relative to the found MILP solutions. Finally, the dimensioning approach is demonstrated for a realistic benchmark scenario.

Keywords multicluster architecture · parallelism analysis · evolutionary computing · evolutionary design

1 Introduction

Future embedded systems move from Multi-Processor Systems-on-Chip (MPSoC) to Many-Core systems containing hundreds of heterogeneous processing elements (PEs) allowing to execute multiple, concurrent applications [1, 2]. The multicluster architecture enables to cope with the challenging hardware/software requirements. Farkas et al. [3] show that a multicluster architecture has significant performance enhancements in terms of cycle time, relative to a single-cluster architecture with the same hardware resources. In addition, the size and complexity of components on critical timing paths is reduced. Similar to [3], our multicluster architecture is a decentralized and dynamically scheduled architecture. Due to task-level granularity, heterogeneous PEs as well as interfaces to memory and peripherals are distributed across the clusters. The dimensioning is limited to multiclusters representing sets of heterogeneous PEs. Hence, each cluster corresponds to a MPSoC and includes a control processor (CP) for task scheduling.

Dimensioning the multiclusters requires to efficiently distribute multiple applications among clusters with different PE types. The approach represents an initial design space exploration (DSE) phase efficiently finding a suitable multicluster configuration in the large design space. We introduce a fast dimensioning approach for the appropriate distribution of applications onto clusters with heterogeneous PE types. Each application include deadline and period representing real-time constraints.

In this paper, parallelism analysis is applied to represent the mapping of an application onto heterogeneous PEs via modified parallelism profiles [4, 5]. We use the idea of [5] to extract parallelism profiles which are independently on an architecture and scheduling. The profiles are independently on an architecture because a machine with unlimited resources is considered. To be independent on a scheduling, as soon as possible (ASAP) and as latest as possible (ALAP) schedules are calculated to extract earliest starting and latest fishing times. Resulting slacks are added into the profile. Latency implied by the CP can be easily included by simply shifting the starting and finishing times. From the profiles, average parallelism values are determined for each PE type. Then, a mapping of applications onto a cluster is represented by the accumulated parallelism values. Our approach focuses on deriving a reasonable number of PEs per cluster. The use of average parallelism values enables fast exploration to efficiently search in the large design space. Hence, the dimensioning represents an initial DSE phase. Given the dimensioning solution, system refinement will be performed in more accurate but less time-efficient simulation-based techniques.

From the average parallelism values, a parallelism value matrix is built representing a mapping of multiple applications onto a machine with unlimited resources (heterogeneous PEs). Hence, the parallelism value

matrix is also independently on an architecture and scheduling. The dimensioning uses the parallelism value matrix as input. We propose a heuristic solution technique which ensures efficiency in finding a suitable dimensioning solution. For more than a decade, genetic algorithms (GAs) have been used to solve DSE problems generating high-quality solutions meeting multiple objectives and constraints [6]. Moreover, our dimensioning approach supports problem formulation via Mixed Integer Linear Programming (MILP) to provide an optimal reference. A scalability analysis evaluates the quality of the GA relative to the MILP. Finally, the approach is demonstrated using a realistic application scenario based on the Embedded Systems Synthesis Benchmarks Suite (E3S) [7].

In addition, the approach allows dimensioning of multiclusters representing sets of homogeneous PEs. The latter assumes that each application can be distributed among clusters. The solution of this multicluster dimensioning is to simply add the average parallelism values for each PE type and the accumulated and rounded up parallelism value shows the number of PEs in a cluster. Hence, dimensioning of multiclusters representing sets of homogeneous PEs is not further mentioned.

In the remainder of the paper, Section 2 reviews related work. Section 3 gives an overview of our approach for dimensioning the multicluster architecture. The GA is explained in detail in Section 4. Then, Section 5 describes the MILP. We demonstrate experiments and corresponding results in Section 6. Finally, extensions for the dimensioning problem are discussed in Section 7. Section 8 concludes our work.

2 Related Work

Evolutionary computing found its way into DSE of embedded systems more than 10 years ago. Blickle et al. [8] applied evolutionary algorithms to map task-level specifications onto heterogeneous hardware/software architectures, known as system-level synthesis. Palesi and Givargis [9] used a GA to explore the design space of a parameterized SoC architecture meeting multiple objectives. Künzli et al. [10] presented a framework for design space exploration of embedded systems. The framework allows to conveniently solve multiobjective DSE problems via GAs. All given DSE examples imply a simulation-based mapping of applications onto architectures. In order to support the exploration of larger design spaces, our DSE approach uses analytical-based mapping in terms of parallelism analysis, as mentioned in Section 1.

Maalej et al. [11] presented a semi-analytical dimensioning approach similar to ours. A pre-exploration step extracts several metrics via task analysis and designer experience. Afterwards, a GA finds solutions in the reduced design space of an underlying clustered heterogeneous multi-processor architecture. Contrasting to the work presented here, the authors represent each task by several metrics and tasks are clustered to build the architecture. In our approach, the focus is on an efficient distribution of applications among clusters. We use a single vector of parallelism values to describe an application mapping which drastically reduces complexity.

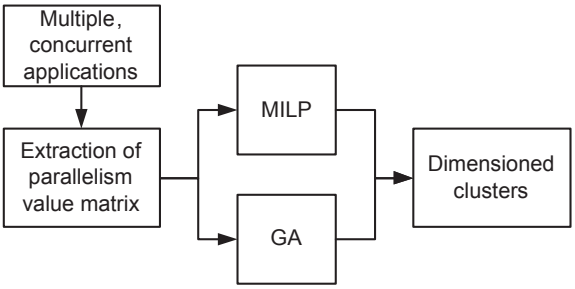

Figure 1: Overview of the heterogeneous multicluster dimensioning approach.

3 Methodology for Dimensioning the Heterogeneous Multicluster Architecture

Figure 1 depicts our methodology to dimension the heterogeneous multicluster architecture by efficiently distributing the given applications amongst the clusters. Multiple concurrent running applications are used as input to extract a parallelism value matrix. Each application is described as high level task graph and the extraction uses performance figures of the available PE types. The parallelism value matrix is forwarded to the solver which assigns each application a cluster. Considering several design objectives and constraints, the approach aims at grouping applications, sharing same PE types, into clusters. Figure 1 shows two solvers: a C++ based GA and a MILP solution implemented on a commercial tool.

3.1 Application and architecture model

Our proposed approach assumes multiple applications, each composed of several tasks. Moreover, applications are considered as threads which are independently on each other. More specifically, both no data and no control-flow dependency exist between threads. It allows us to limit application execution to a cluster and a task is executed on a PE. The target architecture is illustrated in Figure 2 and consists of clusters which are sets of heterogeneous PEs, e.g., General Purpose Processor (GPP), Digital Signal Processor (DSP), and Application-Specific Integrated Circuit (ASIC). Each cluster relates to a heterogeneous MPSoC and includes a control processor (CP) responsible for dynamic task scheduling. The CP is limited by the amount of efficiently handled tasks since it resolves task dependencies at runtime. A resulting scheduling delay motivates to constrain the maximum number of PEs per cluster. In addition, Figure 2 shows an underlying network connecting cluster resources. As the approach aims at deriving the reasonable number of PEs, we consider data transfer effects assuming a fixed communication

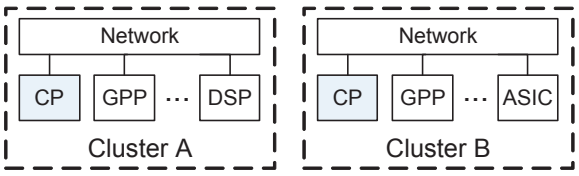

Figure 2: The heterogeneous multicluster architecture.

network topology.

3.2 Extracted parallelism value matrix

The dimensioning approach estimates application mappings onto clusters through average parallelism values. Hence, the first step is to extract the parallelism value matrix $\mathbf{\Phi}$ for the given applications as shown in Figure 1. $\mathbf{\Phi}$ defines average parallelism values ϕ_{ij} for each application i and PE type j calculated from the parallelism profiles. An example is the following parallelism value matrix $\mathbf{\Phi}$ for three PE types (rows) and two applications (columns).

$$\mathbf{\Phi} = \begin{pmatrix} 1.70 & 0.50 \\ 0.95 & 1.70 \\ 0.45 & 0 \end{pmatrix}.$$

Referring to the example above, the three parallelism profiles of the first application (first column) are illustrated in Figure 3. The second application (second column) is limited to two PE types. In a parallelism matrix $\mathbf{\Phi}$, average parallelism will be interpreted as average number of PEs necessary to finish an application on time. As mentioned in Section 1, we apply a performance estimation technique based on parallelism analysis developed in [5]. The technique creates modified parallelism profiles [4] to analyze the parallelism over time. A mapping option table is used to consider only feasible application mappings. Moreover, the parallelism analysis addresses a need to meet application performance constraints, such as latency or bandwidth. The ALAP schedule included into the parallelism profile guarantees that a application meets its deadline. Assuming enough available parallelism, shortening a deadline will increase the number of necessary parallelism and the height of the profile, respectively. In a parallelism profile, communication costs (bandwidth constraints) are included via a function of the transferred amount of data resulting in delayed earliest starting times. The function can be adapted to model different communication topologies.

In general, starting time of different applications can be time shifted. This allows profiles of applications in the cluster to be combined in a way that the total profile is almost homogeneous. Different starting times avoid peaks in the resulting profile as far as possible. Because considering all possible time shifts is not suitable, the nearly homogeneous profile will be approximated by an average parallelism value. This value can easily be determined by accumulating the average parallelism values ϕ_{ij} for PE type j and all applications i in the cluster.

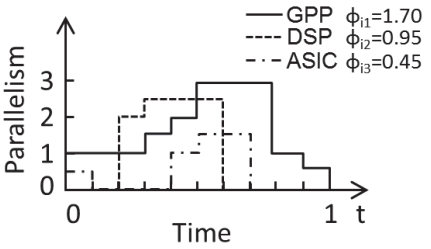

Figure 3: Parallelism profiles and average parallelism/ PE load for an exemplary application i executed on three PE types.

3.3 Focused design objectives

Assuming an upper bound for the number of PEs in each cluster (see Section 3.1), we focus on minimizing (a) the total number of PEs in the system. In addition, (b) the number of clusters must be minimized to reduce control processors (CPs) in the system. Because PEs are usually not fully utilized by an application, several applications should share a PE within the cluster. Hence, the optimization strategy is to efficiently distribute applications among clusters. The PE reuse between several applications minimizes the total number of PEs. Reducing the number of CPs and PEs allows to save system costs (area). Moreover, (c) the variability of cluster size (in terms of number of PEs) is minimized to balance application load among clusters. This allows to reduce application delay resulting from queueing at the CP and task scheduler, respectively. Reducing delay and size (costs) are formulated as single objective function converting the multi-objective problem into a single-objective problem solved through a GA, see Section 4. A MILP approach is used as reference, see Section 5. We restrict our research to these design objectives and a single-objective function because we experienced that MILP solvers could only find the optimal solution for a reduced problem formulation and complexity, respectively. The complexity increases with the number of applications, PE types, and objective variables. In Section 7, we discuss further design objectives and multi-objective problem formulation.

4 Evolutionary Framework

Genetic algorithms (GA) allow fast solutions with sufficiently high quality and easily support multiple objectives. Our steady-state GA applies evolution of overlapping populations of individuals over a number of generations. In the GA, simple one-chromosome individuals (haploid model) are used to describe the dimensioning problem. A chromosome is represented by a vector of genes. Each gene defines an application and a gene value represents an allocated cluster. Given two clusters (1, 2) and three applications, two example chromosomes are defined as follows: $\boldsymbol{g}_1 = (1, 2, 2)$ and $\boldsymbol{g}_2 = (1, 1, 1)$. The first chromosome shows that

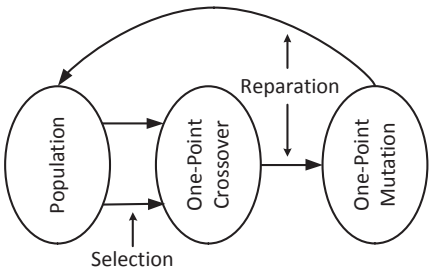

Figure 4: (Steady-state) GA principle.

both clusters are allocated by the applications. In contrast, only cluster 1 has been used in the second chromosome. Moreover, each individual is assigned a fitness value determined by the fitness rate (objective) function. Figure 4 shows the principle of the GA. An initial population will be randomly generated. In each generation, individuals with the best fitness values are selected and crossed to generate new individuals and new individuals are mutated according to a mutation rate. Variation through one-point crossover and one-point mutation enables an iterative improvement of the offspring. Crossover and mutation can change cluster configurations of an individual in a way that the number of allowed PEs is exceeded changing it to an illegal individual. Reparation is applied to transform from illegal to legal individuals allowing for significantly improved statistics of fitness values compared to no reparation is applied. The scalability analysis in Section 6 is used as example assuming a crossover rate of 60 % and a mutation rate of 20 %. Compared to no reparation, variation of the fitness values improves by around 10 % and the average fitness values increase by around 2 %. Despite reparation results in around 100 % longer solution time for the example, it significantly improves the quality of solution.

Table 1 gives the constant terms used in our work for both the GA and MILP implementation. Assume that we are given N number of applications, where $1 \leq i \leq N$. Applications are executed on M PE types, where $1 \leq j \leq M$. Parallelism values for each application i and each PE type j are expressed by $\phi_{ij} \in \mathbb{R}$ and they represent elements of parallelism value matrix $\mathbf{\Phi} \in \mathbb{R}^{M \times N}$. The GA places applications on C cluster candidates, where $1 \leq k \leq C$. In case the number of cluster candidates C is larger than the number of allocated clusters, some cluster candidates will be unused. The maximum allowed number of PEs in a cluster is given by $P_{\max}$.

P_k defines the number of PEs in each cluster candidate, where $0 \leq P_k \leq P_{\max}$. The maximum and minimum number of PEs amongst the allocated clusters are expressed by $U_{\max}$ and $U_{\min}$. That is:

$$U_{\max} = \max_k(P_k)$$
$$U_{\min} = \min_k(P_k) \quad \forall P_k > 0.$$

In addition, we use $b_k' = 1$ to indicate that cluster candidate k contains zero PEs ($P_k = 0$).

Table 1: The constant terms used in our GA/ MILP implementation. These are either architecture or application specific.

Constant	Definition
N	number of applications
M	number of PE types
C	number of cluster candidates
$P_{\max}$	maximum number of PEs in cluster
$U_{\max\,/\,\min}$	maximum/ minimum number of PEs extracted from the allocated clusters
ϕ_{ij}	parallelism value of PE type j in application i (obtained as explained in Section 3.2)
$\mathbf{\Phi}$	parallelism value matrix

The fitness value determines the best individuals from a population which are selected to generate a new population. As mentioned in Section 3.3, our fitness rate function is determined by the design goals which try to minimize size and delay:

$$\underbrace{\sum_k P_k}_{\text{\#PEs}} + \underbrace{C - \sum_k (b_k')}_{\text{\#clusters}} + \underbrace{U_{\max} - U_{\min}}_{\text{variability}} \rightarrow \min \quad (1)$$

The first term in (1) represents a primary optimization goal minimizing the total number of PEs. This is because the remaining terms generally have smaller values. Hence, the minimal number of clusters (term 2) and equally distributed PE load amongst the clusters (term 3) are considered as secondary goals without explicit weighting. For example, increasing the number of applications and keeping $P_{\max}$ constant can implicitly render term 2 higher than term 3. In contrast, less clusters could cause more delay at the CP (larger variability). If necessary, (1) can be modified to explicitly weight the terms.

In the GA, Algorithm 1 (below) performs reparation of an illegal individual after crossover and mutation. Reparation is applied by randomly moving applications (line 7) between clusters until each cluster does not exceed $P_{\max}$ (line 6). Before each reparation, the current numbers of PEs in the clusters is determined (line 3). The assigned cluster of an application has been taken from an according gene value. In addition, the moving of applications (line 7) is randomly initialized to improve GA results. Lines 4 and 8 ensure that reparation proceeds until the individual gets legal.

Referring to Figure 4, reparation can occur after mutation and crossover if $P_{\max}$ has been exceeded in a cluster. As mentioned before, the number of reparations in each generation depends on the $P_{\max}$ constraint. Given an application scenario, the number of reparations increases with smaller $P_{\max}$. This is because cluster size becomes closer to $P_{\max}$ making illegal individuals more likely after variation through crossover and mutation.

Algorithm 1 Reparation of an illegal individual

```
Select genome g
while g is illegal do
    determineNumberPEsInClusters(g)
    g ← legal
    for each gene g in g do
        if P_g > P_max then
            randomlyMoveApplication(g)
            g ← illegal
            exit for
        end if
    end for
end while
```

Our GA implementation is based on GaLib [12] and overwrites the genetic operators and the fitness rate function. With the experimental evaluation in Section 6, we show that the GA produces solutions faster and with adequate quality compared to the found MILP solutions. As an example from the complexity scaling analysis, we generated results in around 200 milliseconds for a larger scenario including 20 applications and 10 PE types.

5 MILP Formulation

We present a MILP formulation of the problem of minimizing size and delay. MILP allows to solve problems with both discrete decisions and continuous variables. The MILP framework is used to compare results with our GA implementation introduced in the previous section. The problem has been formulated in linear programming (LP) format to be further solved by CPLEX [13], a commercial tool.

Our MILP approach uses binary variables to place applications in the clusters. In order to express the location of application i in cluster k, we use the b_{ik} variable. More specifically,

$$b_{ik} := \begin{cases} 1 & \text{if application } i \text{ placed in cluster } k \\ 0 & \text{otherwise.} \end{cases}$$

Additional integer variables are used to calculate the allocated PEs and to ensure the $P_{\max}$ constraint. The number of instances of PE type j allocated in cluster k is indicated by the P_{jk} variable. That is,

- P_{jk} . . . instances of PE type j in cluster k.

Minimum and maximum values for the number of PEs amongst the clusters are needed to calculate the variability in cluster size. Their calculation requires comparison constraints in the MILP using an upper limit L. It corresponds to the worst case of the total number of PEs. Hence, ϕ_{ij} for application i and PE type j are rounded up and added up:

- $L = \sum_{ij} \lceil \phi_{ij} \rceil$ . . . upper limit for number of PEs.

Please note that the *ceil* ($\lceil . \rceil$) operator requires additional real variables and constraints which are not shown for clarity reasons. After describing the binary and integer variables, we continue presenting our constraints. The first constraint is regarding a unique assignment of an application, that is, an application i can be executed only on one cluster k:

$$\sum_{k=1}^{C} b_{ik} = 1 \quad \forall i \in N \tag{2}$$

A PE reuse amongst applications is enabled by merging parallelism values for shared PE types. Hence, all instances of a PE type j are aggregated for each cluster candidate k and its allocated applications i:

$$P_{jk} = \lceil \sum_{i=1}^{N} b_{ik}\phi_{ij} \rceil \quad \forall j \in M;\ \forall k \in C \tag{3}$$

In (3), the parallelism value ϕ_{ij} is addressed for application i and PE type j. For each cluster candidate k and PE type j, the aggregated instances are rounded up. Note that the necessary rounding constraints are not presented for clarity reasons. Now, the resulting PEs are aggregated for each cluster candidate considering the limited cluster size $P_{\max}$:

$$P_k = \sum_{j=1}^{M} P_{jk} \qquad \forall k \in C \tag{4}$$

$$P_k \leq P_{\max} \qquad \forall k \in C \tag{5}$$

All PE types j included in the cluster candidates k are summed up in (4). Hence, P_k is the number of PEs in the cluster candidate k. Referring to Section 3.1, each cluster is constrained by $P_{\max}$, the maximum number of allowed PEs. This is ensured by (5). In order to calculate the variability in cluster size, (6), (7), (8), and (9) determine the minimum and maximum value of P_k:

$$P_k \leq U_{\max} \qquad \forall k \in C \tag{6}$$

$$b_k' L + P_k \geq U_{\min} \qquad \forall k \in C \tag{7}$$

$$P_k - 2L(1 - b_k') \leq 0 \qquad \forall k \in C \tag{8}$$

$$P_k + 2Lb_k' \geq 1 \qquad \forall k \in C \tag{9}$$

In (6) and (7), the maximum and minimum number of PEs in the clusters are calculated. From (7), you can see that for $b_k' = 1$, cluster candidate k is excluded from the minimum calculation because only allocated cluster will be considered. In addition, (8) and (9) are necessary to set $b_k' = 1$ if cluster candidate k includes zero PEs. Given the necessary constraints in the MILP formulation, our objective function corresponds to the fitness rate function in Section 4 which minimizes size and delay of our optimization problem.

6 Experimental Evaluation

First, the GA performance is evaluated using the MILP solution as reference. Processing times are normalized

to a system with an AMD Opteron running at 2.2 GHz using one core. Second, realistic benchmark results demonstrate the multicluster dimensioning approach. GA parameters have been optimized based on the presented experimental setup. More specifically, 60 % crossover rate, 20 % mutation rate, and population size of 50 have been choosen from the optimization results.

6.1 Experimental setup

For the complexity scaling analysis, we generated parallelism value matrices $\mathbf{\Phi}$ with pseudo-random values. The probability that an application executes on a PE type has been set to 50% ($\cong$ 1 − sparsity of $\mathbf{\Phi}$). The number of applications i and the number of PE types j are increased to scale towards more complex problems ($\mathbf{\Phi}$). We apply the MILP solution from Section 5 to evaluate the performance of the GA implementation from Section 4. Each GA experiment runs for 1.000 generations and is repeated 100 times to account for the non-deterministic nature of a GA. Then, minimum, maximum, and average fitness values are extracted from the results. With $C = N$, the number of applications determines the number of cluster candidates. Moreover, $P_{\max}$ has been set to a value that allows for more than one cluster and several applications can share a single cluster.

The second setup focuses on dimensioning results using a multi-application scenario based on the E3S Benchmark Suite [7]. E3S is largely based on data from the Embedded Microprocessor Benchmark Consortium [14]. Task graphs describe the periodic applications which are concurrently executed in the scenario. We extract $\mathbf{\Phi}$ from the 20 provided applications which range from automotive, industrial, telecommunication, networking, to general-purpose applications. In addition, we limited ourselves to five PE types from the 17 available processors for clarity reasons without loss of generality. The sparsity of $\mathbf{\Phi}$ is 63 %, indicating that an application will be executed on 1.85 PE types on average. The number of cluster candidates is also determined by $C = N$. With $P_{\max} = 4$, we constrain each cluster to a maximum of four PEs.

6.2 Complexity scaling results for the GA and MILP solutions

Figure 5 depicts the performance behavior of the GA and MILP in case the number of applications and PE types increase to more complex problems ($\mathbf{\Phi}$). A MILP solution could only be found for maximum 10 applications. (For larger problem sizes, the solver aborts due to memory limitations, in our case 2 GB.) The found MILP solution is mostly a sufficient but not necessarily an optimal solution. For example, the MILP problem with 20 applications and 10 PE types includes 1125 linear constraints, 200 reals, 427 integers, and 620 binaries. Solving this problem, the MILP solver aborted after 4448 seconds resulting its best found solution with objective value of 204. In contrast, GA executes each run in around 200 milliseconds with an average deviation of 1.9 % from the MILP solution. One reason for the low difference to MILP is that $P_{\max}$ drastically reduces the solution space otherwise assigning N applications to C clusters would result in C^N combinations. Hence, GA generates sufficiently high quality solutions in a fraction of time whereas MILP could not even find an optimal solution. Referring to Figure 5, GA generates better solutions than the commercial MILP solver, e.g., for 20 applications and 6 / 8 PE types. Moreover, the difference between max. and min. objective values increases with larger $\mathbf{\Phi}$. In general, the solution time, after GA and MILP converge, increases with the problem complexity. Whereas a larger sparsity of $\mathbf{\Phi}$ decreases MILP problem complexity after variable optimization in the solver. In the two examples before, the GA converges after around 5.000 / 10.000 generations whereas less than 1.000 generations are required for smaller problems. In contrast, small MILP problems already have hundreds of variables and constraints, meaning a large number of subproblems, solved by the branch & cut algorithm. Despite solution time also depends on the way how linear problems are described, it gives a reason why the MILP converges after several hours for smaller problems.

As mentioned in Section 4, the quality of GA solutions has been improved applying reparation after crossover and mutation at the expense of larger simulation time. As mentioned before, MILP does not find an optimal solution for a larger number of applications and PE types. In contrast, the nature of GAs does not guarantee an optimal solution. For larger problem sizes, the GA shows increasingly varying fitness values affecting the quality of solution. A major advantage is that the GA easily scales to larger problem sizes than considered in the experiments. Regarding the quality of solution the GA enables, there is no further conclusion possible due to missing optimal MILP solutions for larger problems.

6.3 Dimensioning result for the E3S multi-application scenario

Since the sparsity of $\mathbf{\Phi}$ is relatively large, MILP was able to find the optimal solution shown in Figure 6. Moreover, the GA has found this solution due to the 100 reruns. The aggregated parallelism values and the resulting number of PEs are illustrated. The result represents a heterogeneous multi-cluster solution since all clusters are heterogeneous in terms of PE types (described by different shades of grey). In the solution, eleven PEs are distributed over three clusters. In contrast, a single-cluster solution would require nine PEs. From the figure, we see that the PEs of TI TMS320C6203 are utilized relatively homogeneously in the clusters. Despite the IBM PowerPCs are not much used, they occur in each cluster because the PE type is often allocated by the 20 applications. In addition, the AMD K6-IIIE is limited to cluster 2 since it is called by only one application. Referring to Figure 6, cluster 3 includes one IDT79RC64575

hardly used because the PE is already fully utilized in cluster 1. For example, this information can be feed back to the parallelism analysis to reuse one of the remaining PE types. If possible this would lead to further improvements in the multi-cluster dimensioning.

7 Discussion and Future Work

Since MILP does not find a solution for larger problem sizes, we only want to discuss the extension of the GA in terms of more objectives and constraints. For example, the study limits all clusters to one P_{max} value estimating the maximum allowed CP load by the number of PEs the CP is able to handle. In case of heterogeneous PEs, this is a rough estimation because some PEs can execute tasks faster than others. A more precise approach would be to estimate the maximum allowed CP load by the number of tasks a CP is able to schedule per time unit. Parallelism analysis allows to extract the number of scheduled tasks per time unit for each application. During the dimensioning, these numbers are aggregated and then compared with the maximum allowed CP load in each cluster. In addition, P_{max} can also vary among clusters, e.g., to consider different cluster sizes and CP types.

Further extensions of the work will be to limit or minimize each cluster in terms of area, power consumption, and heat dissipation. In addition, balancing of power and temperature, respectively, would allow to equally distribute heat amongst the clusters for the purpose of improved reliability. The dimensioning approach can also be used to efficiently distribute applications amongst clusters with a fixed number of PEs. Moreover, the static dimensioning approach can be extended to analyze dynamically clustered systems. The communication topology can also be optimized using our approach. For that reason, transfer task parallelism is extracted and the communication interfaces, e.g., DMA and peripherals, are considered as an additional PE type in Φ. Despite the focus on task-level parallelism, the work could be easily used to dimension clustered heterogeneous multi-processor architectures, such as clustered VLIW processors. Hence, parallelism analysis is applied to extract instruction-level parallelism values related to all functional unit types.

Since the paper focuses on principles and optimization of multicluster dimensioning realizing an initial exploration phase, future work aims at comparing and combining the approach with more accurate but less time-efficient simulation-based exploration, e.g., via SystemC simulation.

8 Conclusion

We introduced a GA-based approach which uses only one parallelism value matrix to dimension the heterogeneous multicluster architecture. The GA generates results faster and with a satisfactory quality relative to the found MILP solutions. The parallelism values represent application mappings which are independently on an architecture and scheduling. Hence, the approach enables an initial DSE phase to search for first designs which will be further refined in more accurate but less efficient DSE phases. We propose several enhancements to extend the dimensioning problem to additional objectives, constraints, and more diverse communication and computation topologies. Finally, our GA-based dimensioning returns statistics of the utilization of PE types in the clusters. In future work, this information is fed back to enable the continuous improvement of parallelism analysis and dimensioning, respectively. This iterative enhancement of multicluster dimensioning results in higher cluster utilization by executing on less PEs.

References

[1] S. Borkar, "Thousand core chips: a technology perspective," in *Proc. of DAC*, 2007, pp. 746–749.

[2] M. Horowitz and W. Dally, "How scaling will change processor architecture," in *Proc. of ISSCC*, 2004, pp. 132–133 Vol.1.

[3] K. I. Farkas, P. Chow, N. P. Jouppi, and Z. Vranesic, "The multicluster architecture: Reducing processor cycle time through partitioning," *International Journal of Parallel Programming*, vol. 27, pp. 327–356, 1999.

[4] K. C. Sevcik, "Characterizations of parallelism in applications and their use in scheduling," in *Proc. of SIGMETRICS*, 1989, pp. 171–180.

[5] B. Ristau, T. Limberg, O. Arnold, and G. Fettweis, "Dimensioning heterogeneous MPSoCs via parallelism analysis," in *Proc. of DATE*, 2009, pp. 554–557.

[6] H. Esbensen and E. Kuh, "Design space exploration using the genetic algorithm," in *Proc. of ISCAS*, vol. 4, 1996, pp. 500–503.

[7] R. Dick. (2011, Sep.) Embedded system synthesis benchmarks suite. [Online]. Available: http://ziyang.eecs.umich.edu/∼dickrp/e3s/

[8] T. Blickle, J. Teich, and L. Thiele, "System-level synthesis using evolutionary algorithms," *Design Automation for Embedded Systems*, vol. 3, pp. 23–58, 1998.

[9] M. Palesi and T. Givargis, "Multi-objective design space exploration using genetic algorithms," in *Proc. of CODES*, 2002, pp. 67–72.

[10] S. Künzli, L. Thiele, and E. Zitzler, "Multi-criteria decision making in embedded system design," in *In B. Al-Hashimi (Ed.), System On Chip: Next Generation Electronics, IEE Press*, 2006, pp. 3–28.

[11] I. Maalej, G. Gogniat, J. L. Philippe, and M. Abid, "System level design space exploration for multiprocessor system on chip," in *Proc. of Symposium on VLSI*, 2008, pp. 93–98.

[12] M. Wall. (2011, Sep.) Galib: A c++ library of genetic algorithm components. [Online]. Available: http://lancet.mit.edu/ga/

[13] IBM. (2011, Sep.) Ibm ilog cplex optimizer. [Online]. Available: http://www.ibm.com/software/integration/optimization/cplex-optimizer/

[14] EEMBC. (2011, Sep.) The embedded microprocessor benchmark consortium. [Online]. Available: http://www.eembc.org/

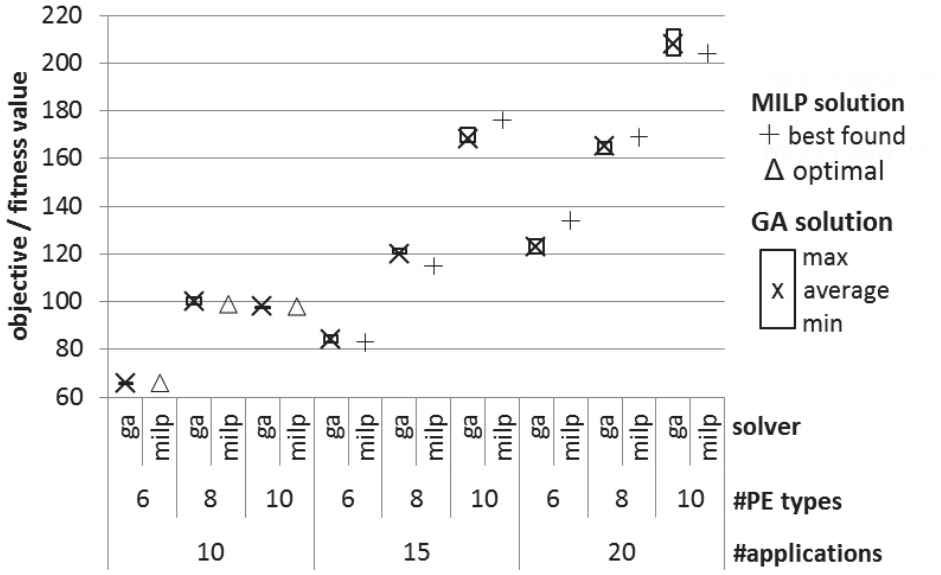

Figure 5: Complexity scaling results for MILP and GA.

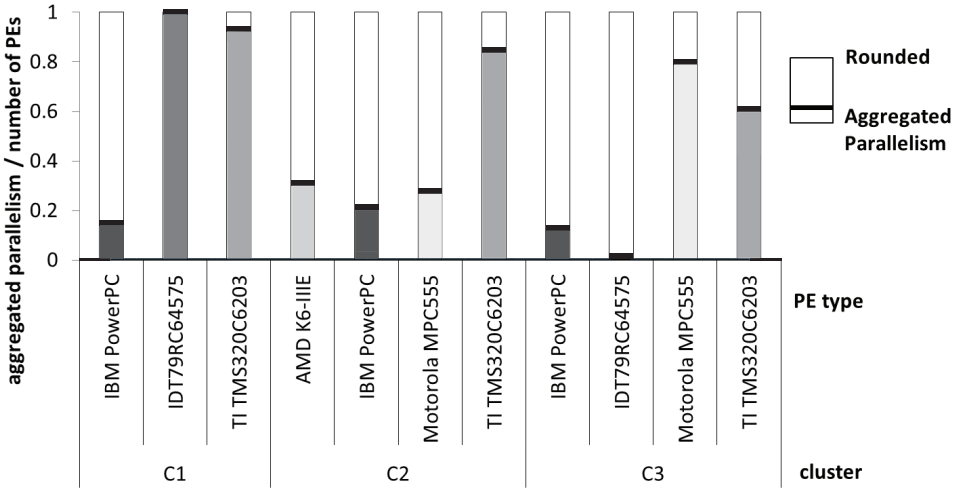

Figure 6: Heterogeneous multi-clusters optimized for the E3S [7] benchmark scenario.