Novel Genome Coding of Genetic Algorithms for the System Partitioning Problem

Bastian Knerr, Martin Holzer, and Markus Rupp
Institute of Communications and RF Engineering
Vienna University of Technology, Austria
Email: {bknerr,mholzer,mrupp}@nt.tuwien.ac.at

Abstract— **The research field of partitioning for electronic systems started to attract significant attention of scientists about fifteen years ago. Gaining ever more importance due to the rampant growth and shortening design cycles in wireless embedded systems, a multitude of formulations for this problem has emerged. Accordingly, a similar multitude can be found in the number of strategies that address system partitioning. A large proportion of the applied strategies utilise the concept of genetic algorithms, or, when based on different strategies, usually compare themselves to standard implementations of genetic algorithms. In this work, the internal mechanisms of this optimisation technique are thoroughly investigated and severe shortcomings of standard implementations are identified. Beneficial observations are made with respect to chromosome coding and mutation dedicated to typical problem graphs revealing a significant impact on the obtained solution quality. New problem oriented codings and operators are introduced and their performance is demonstrated on a task graph set of relevant process graphs. Finally, the modified genetic algorithm competes against Wiangtong's tabu search, Axelsson's simulated annealing, and Kalavade's GCLP algorithm.**

I. Introduction

Hardware/software (*hw*/*sw*) co-design is a design paradigm for the joint specification, design, and synthesis of mixed *hw*/*sw* systems. The interest in automatic co-design techniques is driven by the increasing diversity and complexity of applications employing embedded systems, the need to curb the rising costs of design, verification, and test for such systems, as well as reducing time-to-market. At numerous times during the design process crucial decisions have to be made that dramatically influence the quality and the cost of the final solution. Some design decisions might have an impact of even 90% of the overall cost; among these *hw*/*sw* partitioning is of prominent relevance [1], [2].

Hw/*sw* partitioning is generally defined as the mapping of functional parts of the system description to architectural components of the platform, while satisfying a set of constraints like time, area, power, throughput, delay, etc. *Hardware* usually stands for the implementation of a functional part, e.g. an FIR filter or a Walsh-Hadamard transform, as a custom data path (ASIC, FPGA) that features a high throughput and is often very power efficient. However, such a *hw* unit is expensive to design and inflexible when it comes to future modifications. Contrarily, *software* stands for the compilation of the functionality onto a general-purpose or digital signal processor (DSP). It generally provides flexibility and is cheap to maintain, whereas the required processors are rather power consuming and offer less performance in speed. Therefore, an optimal trade-off between cost, power, performance, and chip area over the complete system is difficult to identify.

Approaches based on genetic algorithms (GA) have been used extensively in this research field. In this work the internal mechanisms of GA are thoroughly analysed and altered towards a better exploitation of their potential. Concretely, to the best of our knowledge, no publication in the field of electronic system design elaborates on the most beneficial way to create a genotype for process graphs with precedence constraints. The main contribution of this work lies in the suggestion of different coding strategies supporting the fundamental schema theorem [3]. Moreover, it is elaborated on recombination and mutation techniques and more appropriate versions are presented that are geared to the chosen partitioning scenario. The new mechanisms are empirically verified against GA implementations with features taken from the literature [4]–[6], as well as against implementations of other heuristic search methods as Wiangtong's tabu search [7], Kalavade's global criticality/local phase (GCLP) algorithm [8], and a simulated annealing approach discussed by Axelsson [9].

The rest of the paper is organised as follows. Section II lists the most reputed work in the field of partitioning techniques. Section III illustrates the basic principles of system partitioning, gives an overview of typical graph representations, and introduces the common platform abstraction. It is followed by a detailed description of the GA mechanisms in Section IV and under which aspects they have to be formulated. In Section V the sets of test graphs are introduced, the different GA versions are compared to each other and the GA with the most promising parameter selection is compared to the aforementioned heuristic approaches. The work is concluded and perspectives to future work are given in Section VI.

II. Related work

Heuristic approaches dominate the field of system partitioning algorithms, since it is known to be an intractable optimisation problem [10], and, in several formulations, $\mathcal{NP}$-complete [8], [11]. Popular candidates are based on simulated annealing [12], [13] and, to a smaller degree, on tabu search [7], [14], Greedy algorithms [15] have also been ap-

This work has been funded by the Christian Doppler Laboratory for Design Methodology of Signal Processing Algorithms.

1-4244-0840-7/07/$20.00 ©2007 IEEE.

plied. Other research groups developed custom heuristics such as the early work from Gupta [16] or the GCLP algorithm, which features a very low algorithmic complexity [8].
With respect to combined partitioning/scheduling approaches, Lopez et al. [13] have to be mentioned. Other approaches also add communication events to links between *hw* units and *sw* functions [4], [17]. The architecture model varies from having one *sw* and one *hw* unit [12], [15], which might be reconfigurable [18], to a limited set of concurrently running *hw* units combined with a general-purpose processor [7].
A wide variety of genetic algorithms for different versions of partitioning exist, more detailed information is given in Section IV: Srinivasan elaborates on a directed task graph scenario to be partitioned on the common architecture model, but chooses to deploy a genome with no specific ordering, uniform crossover and a very basic *bit flip* mutation scheme [5]. Mei decides for identical GA features, but discusses reconfigurability of the hardware processor (FPGA). Wiangtong compares a dedicated tabu search to a GA featuring very much the same mechanisms but replacing the uniform with a single-point crossover recombination [7]. A very mature and thorough work has been published by Blickle [19] performing an architecture exploration on more than two processors. Internally a GA is applied that features an unordered genome for the allocation of tasks to the processing elements and a uniform crossover. Other more recent approaches by Zou [6] or Mudry [20] work on system graphs with a much finer granularity, from basic block level (control flow graphs) down to operational level (expression trees). The latter work comments on the assembly of a specifically ordered chromosome but does not give an evaluation how this choice affects the performance in comparison to other schemes.

III. System Partitioning

This section covers the fundamentals of system partitioning and the platform abstraction. Due to limited space only a general discussion of the basic terms is given in order to ensure a sufficient understanding of our contribution. For a detailed introduction please refer to the literature [1], [2].
In embedded system design the term *partitioning* can essentially be described as the binding of parts of the system's functionality to a set of architectural components. The term *mapping* is more precise, as the selection of the architectural components is often fixed beforehand. Usually a number of requirements, or *constraints*, are to be met in the final solution, for instance execution time, area, throughput, power consumption, etc. The system functionality is commonly abstracted into a task graph $G = (\mathcal{V}, \mathcal{E})$ representation. In Fig. 1 on the left, five vertices $\mathcal{V} = \{a, .., e\}$ are depicted which are connected by five edges $\mathcal{E} = \{e_1, .., e_5\}$. The vertices cover the functional objects of the system (*processes*), whereas the edges mirror data transfers between different processes. Depending on the granularity of the graph representation, the vertices may stand for a single operational unit (MAC, Add, Shift) or have the rich complexity of an MPEG decoder. The majority of the partitioning approaches [7], [8], [17], [18] decide for medium sized vertices that cover the functionality of FIRs, IDCTs, quicksort or similar procedures. On the

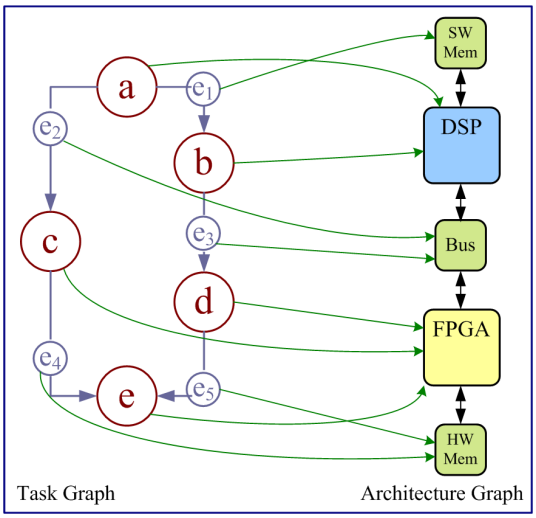

Fig. 1. *Mapping of a task graph to an architecture graph .*

right in Fig. 1 a platform model is depicted in a graph-like fashion. It adheres to the common notion of a combination of a flexible digital signal processor (DSP) for the more control-oriented functionality (*sw*) and an FPGA (or a set of ASICs) for the more data-driven functionality (*hw*). Unlike in the classical binary partitioning problem, in which just two implementation alternatives for every process exist, we allow for multiple possibilities on any resource, comparable to the work of Kalavade [8]. Additionally, the set of available resources $\mathcal{R}$ can be set freely beforehand by the designer. Hence, if $\mathcal{R}_v \subseteq \mathcal{R}$ is the set of available resources for vertex v and $A_{i,j}(v)$ the of jth implementation alternative on resource i, then

$$\forall\, v \in \mathcal{V}\ \exists\, \mathcal{I}_V(v) = \{\ A_{i,j}(v) : i \in \mathcal{R}_v, j = 0..|\mathcal{R}_v|-1\}. \quad (1)$$

Here, $A_{i,j}(v)$ is a triple of characteristic values. In electronic system design measures for consumed area (gate count gc) and required execution time (et) are most common to characterise a process' implementation type. In our work code size (cs) is added as third value to obtain a more symmetric situation for the constraints, since code size is typically a limited resource on a DSP in opposition to gate count being a limited resource on a chip (ASIC) or an FPGA. However, this modification does not change the character of the multi-objective optimisation problem. These values are usually obtained by high level synthesis [8] or estimation techniques like static code analysis [21] or profiling [22]. The plurality of implementation alternatives results from a variation of synthesis parameters, e.g. the unfolding factor, pipelining, register usage on the FPGA or the compiler options for the DSP (trading off stall cycles against code size).
In a similar fashion the transfer times tt for the data transfer edges e, are considered, since different communication resources may exist in the design: bus access to shared

memory for interresource communication, or local memory for intraresource communication,

$$\forall\, e \in \mathcal{E}\ \exists\, \mathcal{I}_E(e) = \{\ (tt_{i,rd}(e), tt_{i,wr}(e)) : i \in \mathcal{C}\}, \qquad (2)$$

where $\mathcal{C}$ is the set of communication resources. The subscripts rd and wr distinguish between read and write access. This abstraction yields a high degree of flexibility to assemble multi-core platforms. On any DSP, bus, or memory, a simple collision arbitration is applied: earliest job first. More sophisticated schemes have already been evaluated [23], [24], but are not in the scope of this paper.

A. Fitness Function and Constraints

In the following the multi-objective character of this optimisation problem is described and basic terms are introduced to conceive the quality of the obtained solutions.
A mapping is called *feasible*, if every $v \in \mathcal{V}$ is mapped to a single implementation alternative $A_{i,j}(v)$ and if every edge is mapped like follows: to the local memory of resource i if both tail vertex and head vertex of this edge are mapped to this resource, or, when these are mapped to different resources, to the shared memory (via the bus).
A mapping is called *valid*, if it is feasible and the following objectives are met:

- The makespan of the task graph T is equal or smaller than the deadline constraint T_{lim}.
- The totally consumed area G_i, measured in gates, for any resource i is equal or smaller than the constraint $G_{lim,i}$ for this resource.
- The total code size S_i, measured in bytes, for any resource i is equal or smaller than the constraint $S_{lim,i}$ for this resource.

As can be seen in (3), the fitness function is a weighted linear combination of the characteristic values for makespan, area, and code size, due to its simple and easily extensible structure. The quality of the obtained solution, the fitness value Ω for a given mapping, is then:

$$\Omega = \alpha\ p_T(T)\ \frac{T - T_{min}}{T_{lim} - T_{min}} + \sum_{i \in \mathcal{R}} \left(\beta_i\ p_G(G_i)\ \frac{G_i}{G_{lim,i}}\ +\ \xi_i\ p_S(S_i)\ \frac{S_i}{S_{lim,i}} \right). \qquad (3)$$

Here, T is the makespan of the graph; G_i is the sum of the gate counts of all processes mapped to resource i; S_i the sum of the code sizes of all processes mapped to i. With the weight factors $\alpha,\ \beta_i, \xi_i, i \in \mathcal{R}$ the designer can set individual priorities. If not stated otherwise, these factors are set to 1. If for instance $G_i > G_{lim,i}$ is true, a penalty function is applied to enforce solutions within the limits:

$$p_G(G_i) = \begin{cases} 1.0 & ,\ G_i \le G_{lim,i} \\ \left(\frac{G_i}{G_{lim,i}}\right)^{\eta} & ,\ G_i > G_{lim,i} \end{cases} \qquad (4)$$

p_T and p_S are defined analogously. If not stated otherwise, η is set to 4.

The validity $\Upsilon = N_{valid}/N \times 100\%$ is the percentage of the number of valid solutions N_{valid} to the number of all solutions N, for N runs of the algorithm.
The constraints are specified by three ratios C_T, C_G, C_S to give a better understanding of their strictness. The ratios are obtained by the following equations:

$$C_T = \frac{T_{lim} - T_{min}}{T_{max} - T_{min}}, \quad C_G = \frac{\sum_{i=1}^{r} G_{lim,i}}{G_{max}}, C_S = \frac{\sum_{i=1}^{r} S_{lim,i}}{S_{max}}. \qquad (5)$$

The values A_{max}, S_{max}, and T_{max} are simply built by the sum of the maximum gate counts gc, code sizes cs, and execution times et of every process (plus the maximum transfer time tt of every edge), respectively. The computation of T_{min} is obtained by scheduling the graph under the assumption of minimum execution and transfer times and full parallelism. Thus, T_{min} and T_{max} are the lower and upper bounds on the makespan.
The computational runtime has been measured in clock cycles by the high resolution timer *QueryPerformanceCounter* of the MS Windows API on a PC (AMD Athlon 64 3000+, 1.8GHz Processor).

IV. Genetic Algorithm

This section briefly introduces fundamental terms, sketches how the GA concept is typically applied to system partitioning, reveals where of the flaws of such a typical deployment lie and finally demonstrates how to significantly improve the GA's performance.
The inspiration of GA originates from the modifications to the chromosomes of a species caused by natural reproduction that iteratively improve the fitness of the species. According to the natural mechanisms, abstracted concepts of *recombination*, *mutation*, and *selection* exist in the algorithm. A population of individuals (or solutions) exists in a generation, of which a subset of individuals is chosen to serve as the parents of the population of the next generation. This *selection* process is guided by the fitness (quality of solution) of the individuals. The creation of a new individual for the next generation by mating of the parent individuals is called *recombination*. The concept of *mutation* is a random mechanism that affects parts of a chromosome of an individual with a certain probability. *Mutation* ensures a persistent diversity in the number of individuals in any population. Depending on the problem formulation there exists a large variety of concrete implementations for all these mechanisms, that cannot be covered within the scope of this paper. In the following we adhere to the classical terms and definitions used by Goldberg [3].

A. Chromosome Coding

The fundament of any GA is the genome, which captures all necessary information to derive a solution for a problem instance. Many different approaches exist, but most common

in general and for system partitioning in particular is a string representation. An intuitive and comprehensive way to represent a solution for the partitioning problem in form of a genome is depicted in Fig. 2. Assume a system graph with $|\mathcal{V}|$ processes shall be partitioned. A vector of length $|\mathcal{V}|$ is provided, in which every entry, a *gene*, corresponds to a specific process: The value of a gene (*allele*) identifies

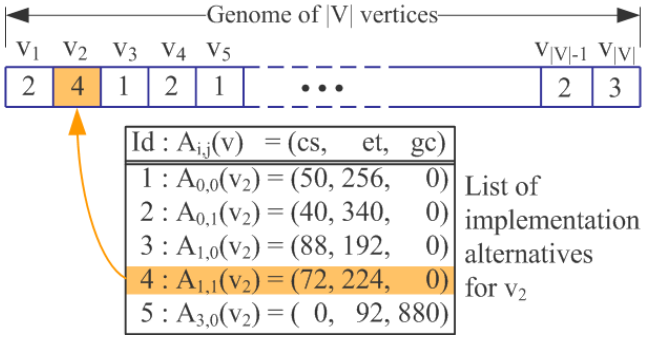

Fig. 2. Chromosome coding for the system partitioning problem.

the implementation alternative for the respective process. A specific partitioning solution is then coded as such a vector filled with the implementation types for all its processes. A coding in this form is very beneficial, since recombination and mutation schemes can be easily defined, as can be seen later on. Nearly all publications in this field adhere to this concept. However, the question, in which order $(v_1, v_2, .., v_{|\mathcal{V}|})$ the processes are aligned in the genome vector, is hardly ever raised and almost always said to be arbitrary. In fact, this is problematic, because of the consequences of the fundamental schema of genetic algorithms [3], when precedence graphs are considered and optimisation is subject to time. The theorem states explicitly that short, low-order, and highly fit schemata are sampled, recombined, and resampled to form strings of potentially higher fitness. A trivial observation is that a mapping is very beneficial with respect to timing for *two* reasons: first, if - in general - process implementations are chosen, which feature rather low execution times; second, if the parallelism in the system graph is mapped to the parallelism in architecture graph in a (near-)optimal (isomorphic) manner. It is the latter aspect, which causes serious performance differences depending on the chosen vertex order in the genome. Assume the graph in Fig. 3 features a highly fit mapping with respect to its timing for the substring containing the vertices e, f, g, h due to a clever exploitation of the parallelism by mapping these vertices to *different* resources in the architecture graph. In the lower genome coding it is almost certain that the assumed beneficial mapping for e, f, g, h will be destroyed during the next recombination cycle, since the *defining length* δ of this schema is very high for all standard recombination operators. Due to the strong relation between timing and the parallelism exploitation it is of major importance to align neighbouring vertices in the system graph preferentially next to each other in the chromosome. This trait, although being a fundamental

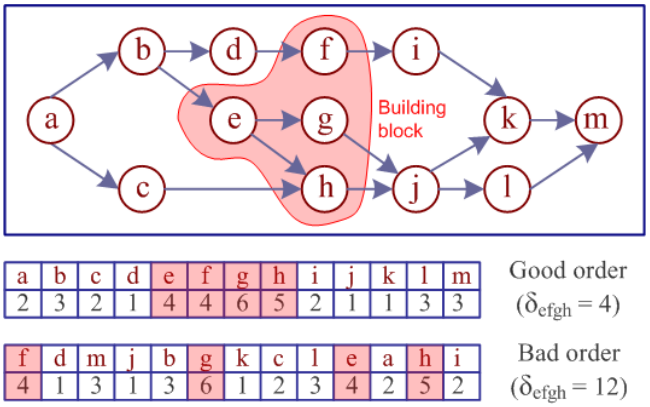

Fig. 3. Examples for good and bad genome orderings.

neighbourhood property of this problem formulation, seems to be completely neglected in the field of system partitioning. In Section V-A different chromosome codings are proposed and the strong dependency between the chromosome coding and the timing payoff is demonstrated. Additionally it is shown, that these considerations do not in any way affect other objective values like code size or area, since these are independent from the graph structure. The theoretical underpinning of the survival of schemata and neighbourhood examinations can be found in the literature [3].

B. 1st Operator - Selection

At any stage of the genetic algorithm, i.e. *generation*, among the individuals present in the population, some have to be selected to serve as *parents* for the individuals of the next generation. Again a multitude of selection criteria exist with varying effects on convergence, robustness and solution quality. Although the focus of this work is not primarily set on an evaluation of this feature, three classical schemes have been examined to complete the picture: survival of the fittest-, binary tournament-, and roulette wheel-selection.

Selection based on survival of the fittest (SOTF) means that from a population consisting of $|\mathcal{P}|$ individuals the best $|\mathcal{P}|/2$ individuals are chosen to serve as parents for the next generation. Binary tournament (BT) means to select consecutively random pairs out of the population, whose fitness values are compared. The fitter one gets the parent status, whereas the other is discarded. Both are removed from the population to avoid multiple selections of the same individual. Roulette wheel (RW) selection distributes probabilities proportional to the fitness values among the individuals, not uniformly as the name suggests. Since our fitness function Ω returns the lower values the better the individual is, the cost-to-fitness transformation $\hat{\Omega} = \Omega_{max,k} - \Omega$ is applied, with $\Omega_{max,k}$ being the worst fitness of the last $k = 10$ iterations. Hence, the selection probability $p_{sel}(i)$ of an individual i in the current population $\mathcal{P}$ calculates to:

$$\forall i \in \mathcal{P} : p_{sel}(i) = \frac{\hat{\Omega}_i}{\sum_{i \in \mathcal{P}} \hat{\Omega}_i} \tag{6}$$

Note, that for the RW selection the highly non-linear character of the fitness function when evaluating invalid solutions, due to the penalty exponent η in (4), leads to an undesirable effect: the probabilities for invalid solutions become small very quickly. This is circumvented by scaling η with the number of invalid individuals in the population $|\mathcal{P}_{inv}|$ to $\eta = \eta_0 - (\eta_0 - 1)\frac{|\mathcal{P}_{inv}|}{|\mathcal{P}|}, \eta_0 = 4$. Refer to Section V-B for the results of the comparison.

C. 2nd Operator - Recombination

Once a subset of individuals has been selected, the so-called *mating* takes place, that is the creation of new offspring individuals from the parents. Combined with the aforementioned chromosome coding, single- and multi-point as well as uniform crossover are very simple to implement and ensure the creation of offspring solutions that are always feasible. In

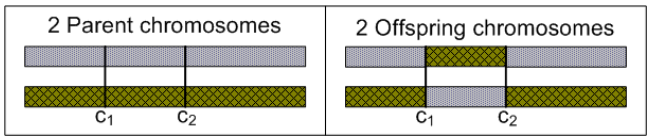

Fig. 4. Recombination via 2-point crossover with cut points c_1, c_2.

Fig. 4 a 2-point crossover is illustrated. The two randomly chosen individuals from the parent subset on the left are cut at two points c_1 and c_2 and recombined by permuting the two substrings between the cut points. One- and multi-point crossover recombination are performed analogously. Uniform crossover means a simple iteration over the genes of the parent chromosomes and selecting the allele from one of the two parents with a certain probability. Normally this probability is set to 0.5. In this work, uniform as well as multi-point crossover, reaching from one cut point to $|\mathcal{V}|/10$ cut points, are evaluated and a significant difference can be observed especially for larger graphs. To keep the population size constant, any parent individual is used twice in the crossover scheme with alternating partners. See Section V-C for results.

D. 3rd Operator - Mutation

The last major operator is a randomised mechanism, which processes the offspring generation and alters small portions of the chromosome with a certain probability. Its main purpose is to provide a chance to (re)introduce new or lost regions of the solution space, and thus to ensure a persistent diversity in the solution subspace covered by the population. Almost omnipresent for the the string coding of the chromosome is a simple one-gene mutation (M_{1g}), that is the alteration of an allele typically with a low probability $p_{1g} = 0.01 \ldots 0.05$. A related scheme especially in symmetric multi-processor scenarios is a swap mutation (M_{sw}), that is the exchange of two process assignments to different processors. In this work one-gene and swap mutation are evaluated for varying probabilities. Due to similar deliberations as in Section IV-A, it can be reasoned that a one-gene mutation does not tap the full potential, especially with respect to the late stages of the genetic algorithm. Assume, a GA has proceeded through several generations, so that the short low-order building blocks in Fig. 2 represent on average partial solutions with a rather good quality. As described before, the solution quality then depends to a large degree on the strong exploitation of the parallelism in system and architecture graph, as illustrated in Fig. 5. One-gene mutation (M_{1g}) is likely to destroy

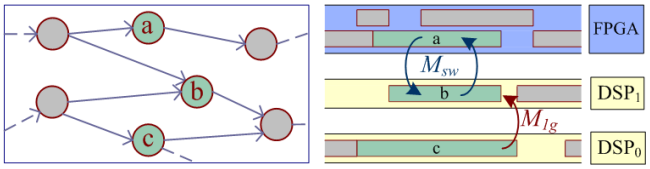

Fig. 5. Partial system graph and schedule: one-gene versus swap mutation.

the beneficial combination of process assignment to *different* processors, when mutating c onto DSP_1 behind b, even if the new implementation alternative of c is better suited to DSP_1. Consequently, M_{sw} seems to be much more appropriate, e.g. when exchanging a and b. Since we allow for platform abstractions with more than two processor units, we extend the swap mutation towards a building block mutation (M_{bb}): on a number of consecutive genes $|\mathcal{R}|$ swaps are applied with a certain probability, $|\mathcal{R}|$ is the set of available resources. The result is a permutation in a limited range of the chromosome that corresponds to a local region (subgraph) of the system graph. The latter is implicitly true, if a chromosome order is chosen that reflects the locality of the processes in the graph. See Section V-D for results.

E. Miscellaneous

This section completes the description of the genetic algorithm with the discussion of the parameters population size $|\mathcal{P}|$ and termination criterion. The first of which is of major importance, since it has a dramatic impact on the solution quality in a direct trade-off with the GA's run time: the higher, the better the solution quality. Up to a certain degree the algorithm designer can choose freely depending on his project's time frame. However, it is obligatory to consider certain policies: $|\mathcal{P}|$ has to be large enough to yield a sufficient diversity in the initial population in order to guarantee a good search space coverage. In general, it is reasonable to bind $|\mathcal{P}|$ to the same parameters that determine the problem size and to ensure that any possible allele per gene is present in the initial population. In this scenario, the population size is then a function, $|\mathcal{P}| = f(|\mathcal{V}|, \overline{|\mathcal{I}_V|})$, with the latter parameter being the average number of implementation alternatives per process. We found the simple product $|\mathcal{V}|\ \overline{|\mathcal{I}_V|}$ to be appropriate.
The termination criterion, i.e. when the GA ceases to breed further generations, scales typically in a similar fashion. However, we found, that terminating after $|\mathcal{V}|$ generations without improvement gave enough room to evaluate the operators and showed sufficient convergence.
As stated before, there exist many more parameters and mechanisms for genetic algorithms: elitism, crowding model,

overlapping generations, to name a few. A complete discussion of those would be far beyond the scope of this paper. We concentrate on the main operators and try to give interpretations of their performance in the next section.

V. Results

To obtain a sustainable fundament for the test runs of the different algorithms, a large set of typical graph structures is mandatory. The deployed sets are based on two sources: the graph generation scheme proposed by Kalavade [8] and the standard task graph (STG) set of the Kasahara Lab [25]. The STG set does not contain area and code size information, which was augmented according to the instructions given in the first source. In the following, every algorithm has been applied to 60 different graphs (30 based on STG, 30 based on Kalavade), for any of which 30 different runs have been performed. Three graph sizes are provided: $|\mathcal{V}| = 20, 50, 100$. If not stated otherwise, the constraints are set medium values: $(C_T, C_G, C_S) = (0.5, 0.5, 0.5)$. The platform model is composed according to the most common case of one *hw* processor (FPGA) and one *sw* processor (DSP). Both feature local memories and are connected by a shared memory via a system bus.
The remaining section surveys the GA operators and tries to demonstrate their impact on the overall performance. Whenever the selected information indicates a relevant interdependence of the operators, an interpretation is given.

A. Results - Chromosome Coding

The most important outcome of this work is the dramatic relevance of the composition of the genome. Three different codings are proposed: a random order (*rand*) of genes, an order based on the vertices' rank in the graph (*rank*), and a more elaborate order based on the medium start times of an *as soon as possible (ASAP)* and *as late as possible (ALAP)* schedule of the graph (*aslap*). In Fig. 6 a small example graph is depicted, in which the ranks of the vertices are annotated. Additionally the two schedules, *ASAP* and *ALAP*, are depicted. In the interest of clarity, communication is omitted and the execution times are averaged over all implementation alternatives per process. It becomes obvious that a genome, in which the genes are ordered according to the rank of the corresponding vertex, mirrors the vicinity relations of the graph already in a reasonable way. A further intensification of this relation lies in the integration of the time dimension and the vertices' dynamic range in the graph. This information is then brought in by the application of the two indicated schedules, of which the start times serve as base for the genome ordering. For instance, process a features in both schedules the same start times $st_{ASAP}(a) = st_{ALAP}(a)$, whereas process b exhibits different start times $st_{ASAP}(b) \neq st_{ALAP}(b)$. In the latter case the mean is taken as base for the genome ordering. It has to be stated, that for all processes more than one execution time exists, so the generation of the two schedules has to rely on the average execution time. Nevertheless, an *aslap* based ordering of the genome preserves the locality of the processes in the system graph more effectively. The empirical demonstration of this trait can be seen in the bar chart in Fig. 7 with averaged cost $\overline{\Omega}$ on the y-axis. The bar groups a),$b2$), and c) result from

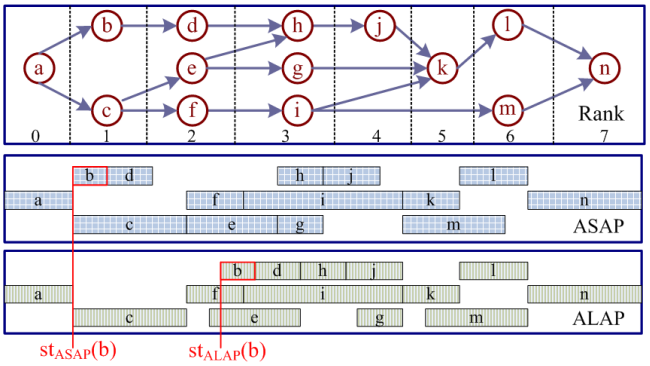

Fig. 6. Example graph with annotated ranks, *ASAP* and *ALAP* schedule.

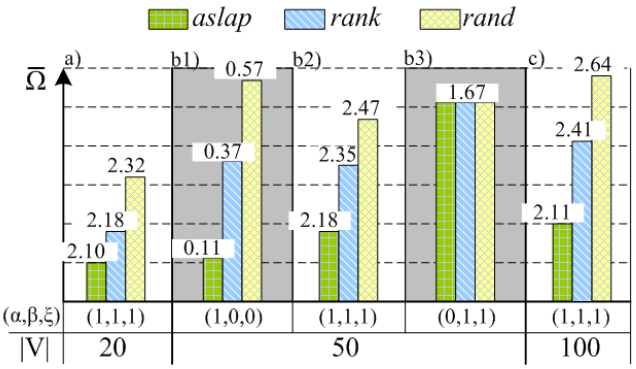

Fig. 7. Averaged cost for different genome codings on all graph sizes.

GA runs with identical weight factors $(\alpha, \beta, \xi) = (1, 1, 1)$, see Equation 3. The effect of the chromosome coding is dramatic with the biggest difference for large graphs: up to 20% better than random coding. This is a reasonable result, since small graphs mean short chromosomes, in which a disorder has limited impact.
For $|\mathcal{V}| = 50$ two more tests are depicted in bar groups $b1$) and $b3$) with a variation in the fitness function. When optimisation is only subject to time $(\alpha, \beta, \xi) = (1, 0, 0)$, a GA with randomly ordered genome performs tremendously worse. In opposition, when neglecting time completely $(\alpha, \beta, \xi) = (0, 1, 1)$, the ordering does not matter at all, leading to identical results. Note, that the demonstrated effect is persistent, even when varying the following three operators *selection*, *recombination*, and *mutation*. Further parameters for this test are: binary tournament selection, uniform crossover, and no mutation.

B. Results - Selection

The evaluation of the first main operator exposes an interesting interdependence with the mutation operator. In the bar chart in Fig. 8 the three selection schemes binary tournament (BT), roulette wheel (RW), and survival of the fittest

(SOTF) are depicted for different values of common one-gene mutation on the x-axis. Graph size is $|\mathcal{V}| = 50$ with uniform crossover and *aslap*-ordered genome. BT and RW selection

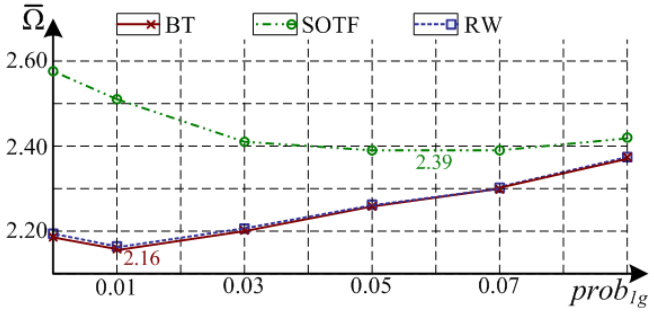

Fig. 8. Result for different selection schemes over varying mutation probabilities.

exhibit an almost identical performance with minimum cost for low mutation probabilities. In fact, mutation improves the results by only about 1% if $p_{1g} = 0.01$ but degrades the outcome for larger values. The SOTF selection shows a very different behaviour improving $\overline{\Omega}$ substantially with best values for $0.05 \geq p_{1g} \geq 0.07$. A cause of this result may lie in the relatively high implicit diversity of the BT and RW selection opposing the SOTF selection. Hence, mutation, which is another guarantor of diversity, degenerates in the first case rather quickly, but leads in the second case to substantial improvements. For mutation probabilities higher than 0.1 the degradation in cost becomes quickly outrageous for all three selection schemes.

C. Results - Recombination

In this section the largest graphs, $|\mathcal{V}| = 100$, are tested with four different recombination schemes (uniform and 10-,5-,1-point crossover) on two genome orderings (*aslap* and *random*. Parameters are: BT selection, and no mutation. The analysed

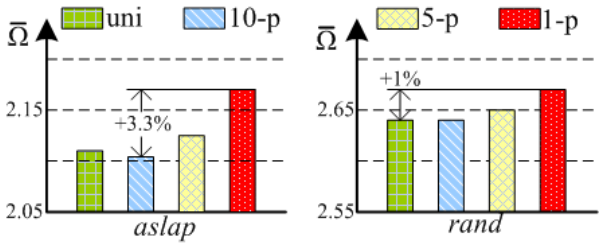

Fig. 9. Result for different recombination schemes for two genome orderings.

schemes in Fig. 9 reveal negligible differences for the GA with randomly ordered genome. But in the *aslap*-case on the left a quite remarkable gap of more than 3% can be observed, which is accompanied by an 15% shorter run time for the GA with 10-point (or uniform) crossover. This effect is still observable to a minor degree for medium graphs ($|\mathcal{V}| = 50$), and is negligible for small graphs ($|\mathcal{V}| = 20$).

D. Results - Mutation

Unlike before the results in this section are related to different platform models to demonstrate the dependency of the building block mutation with the number of available processors $|\mathcal{R}| = 2, 3, 4$ with local memories and connected by a system bus to a shared memory. Again the outcome for the largest graphs is considered, as the differences turned out to be most perceptible. Further operators are *aslap*-ordered genome, BT selection, and 10-point crossover. From Fig. 10

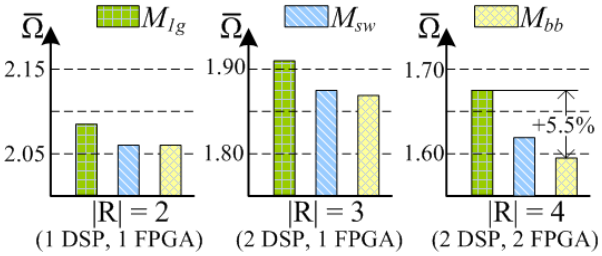

Fig. 10. Result for different mutation schemes on three different platforms.

it can be seen, that a mutation, that permutes the assignment of processes among the available processors, is very beneficial in comparison with the most common one-gene mutation. Remember, that the permuted processes should lie in the same region on the time scale, which is (very likely) implicitly true for adjacent genes in an *aslap*-ordered genome. Hence, the building block mutation is very simple to implement and does not cause any run time overhead.

E. Results - Heuristics

Besides the discussion of GA mechanisms, it is of interest how a dedicated GA performs in comparison to other dedicated heuristic optimisation methods. For this purpose, well reputed algorithms have been implemented to serve as benchmarks: the *penalty reward* tabu search (*pw*TS) [7], a simulated annealing (SA) approach [9], and the GCLP heuristic [8]. Except from minor modifications to the first two approaches (e.g. adding code size, aligning the fitness functions), the problem formulations are nearly identical allowing for a direct comparison of the algorithms on a platform with *hw* and *sw* processor. In Table I the results are listed for a strict constraint

	*pw*TS		SA		GCLP		GA	
$\|\mathcal{V}\|$	$\overline{\Omega}$	$\overline{\Upsilon}$	$\overline{\Omega}$	$\overline{\Upsilon}$	$\overline{\Omega}$	$\overline{\Upsilon}$	$\overline{\Omega}$	$\overline{\Upsilon}$
20	2.58	82.3	2.63	80.8	3.09	35.9	**2.53**	**89.1**
50	2.75	81.8	3.07	11.6	3.13	21.2	**2.71**	**98.3**
100	2.69	**100**	2.99	14.0	3.81	8.1	**2.64**	**100**

TABLE I

RESULTS OBTAINED FOR THE FOUR HEURISTICS.

setting $(C_T, C_G, C_S) = (0.4, 0.4, 0.5)$ to force a number of invalid results, that are indicated by the averaged validity $\overline{\Upsilon}$ percentage.

The GA has been implemented with *aslap*-ordered chromosome, BT selection, uniform crossover for small graphs and $|\mathcal{V}|/10$-point crossover for medium and large graphs, and swap mutation with $p_{sw} = 0.03$ due to $|\mathcal{R}| = 2$ processors. The

best parameter set found for SA included geometric cooling with factor $\alpha = 0.95$, temperature update criterion $t_{up} = 400, 1000, 2500$ for $|\mathcal{V}| = 20, 50, 100$, and termination when the temperature reaches the lower bound of the cost Ω_{LB}. The *pw*TS features a neighbourhood size $S_N = \lfloor\sqrt{|\mathcal{V}|/2}\rfloor$, number of tabu degrees $N_{td} = \lfloor\sqrt{|\mathcal{V}|/2}\rfloor$, calculating to the tabu list length $L_T = S_N N_{td} = |\mathcal{V}|/2$, a long term memory region covering five chromosome elements corresponding to $\overline{|\mathcal{I}_V|}^5 \leq 3125$ regions with $\overline{|\mathcal{I}_V|} \leq 5$, and the *pw*TS terminates after $4|\mathcal{V}|$ iterations without improvement. The GCLP was applied in an unaltered fashion.

The modified GA performs better than all other algorithms opposing the result of the referred publications, in which the GA performance drags substantially behind the tabu search implementations. With respect to our results the main reason can easily be identified. The genome coding has erratically not been considered as relevant element of the GA. Moreover, a one-point crossover as for instance applied by Wiangtong is not appropriate for large genomes and the one-gene mutation is less beneficial than a swap- or building block mutation scheme. The very low performance of the GCLP approach lies in its basically greedy approach with a low algorithmic complexity. The run time of GA, *pw*TS, and SA lies in the range of 10^9 to 10^{10} clock cycles, whereas the GCLP's run time is two orders of magnitude lower.

VI. Conclusion

In this work the classical three-operator genetic algorithm has been thoroughly analysed in application to the system partitioning problem. The significant relevance of the underlying genome coding for typical problem formulations was demonstrated, which has been completely neglected in a large number of publications in this field. Hence, the standard GA implementations performed misleadingly worse than for example dedicated tabu search implementations. Proposals for a better exploitation of the GA's potential with respect to genome coding and mutation were made without imposing additional complexity. In extensive test runs the superior performance of the proposed problem-oriented GA in comparison with the most common GA version and other well-reputed heuristic methods was revealed.

In consequence of the empirical results, future work will concentrate on the theoretical underpinning of the genome coding for combined scheduling and partitioning of precedence graphs onto general multi-processor environments. Based on the analysis of subgraph isomorphism between system and architecture graphs more powerful coding schemes and main operators shall be developed in an analytical manner.

References

[1] G. de Micheli, R. Ernst, and W. Wolf, *Readings in Hardware/Software Co-Design*. San Francisco, CA, USA: Morgan Kaufman Publishers, Academic Press, 2002.

[2] P. Marwedel, *Embedded System Design*. Dordrecht, The Netherlands: Kluwer Academic Publishers, 2003.

[3] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1989.

[4] B. Mei, P. Schaumont, and S. Vernalde, "A hardware-software partitioning and scheduling algorithm for dynamically reconfigurable embedded systems," in *Proceedings of ProRISC*, 2000.

[5] V. Srinivasan, S. Radhakrishnan, and R. Vemuri, "HW/SW Partitioning with Integrated Hardware Design Space Exploration," in *Proc. of the Conf. on Design, Automation and Test in Europe (DATE)*. Washington, DC, USA: IEEE Computer Society, 1998, pp. 28–35.

[6] Y. Zou, Z. Zhuang, and H. Chen, "Hw-sw partitioning based on genetic algorithm," in *Congress on Evolutionary Computation (CEC)*, Portland, Oregon, June 2004, pp. 628–633.

[7] T. Wiangtong, P. Cheung, and W. Luk, "Comparing three heuristic search methods for functional partitioning in hardware-software codesign," *Design Automation for Embedded Systems*, vol. 6, no. 4, pp. 425–449, Sept 2002.

[8] A. Kalavade, "System-level codesign of mixed hardware-software systems," Ph.D. dissertation, University of California, Berkeley, CA, USA, 1995.

[9] J. Axelsson, "Architecture synthesis and partitioning of real-time systems: A comparison of three heuristic search strategies," in *Proc. of the 5th Int. Workshop on HW/SW Codesign, (CODES/CASHE)*, 1997, pp. 161–165.

[10] J. Hromkovič, *Algorithmics for hard problems*, 2nd ed. New York, NY, USA: Springer-Verlag, Inc., 2004.

[11] P. Arató, Z. Á. Mann, and A. Orbán, "Algorithmic aspects of hardware/software partitioning," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 10, no. 1, pp. 136–156, 2005.

[12] J. Henkel and R. Ernst, "An Approach to Automated HW/SW Partitioning Using a Flexible Granularity Driven by High-Level Estimation Techniques," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 9, no. 2, pp. 273–290, 2001.

[13] M. Lopez-Vallejo and J. Lopez, "On the HW/SW Partitioning Problem: System Modeling and Partitioning Techniques," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 8, no. 3, pp. 269–297, 2003.

[14] P. Eles and Z. Peng and K. Kuchcinski and A. Doboli, "System Level HW/SW Partitioning Based on Simulated Annealing and Tabu Search," *Design Automation for Embedded Systems*, vol. 2, pp. 5–32, 1997.

[15] J. Grode, P. V. Knudsen, and J. Madsen, "Hardware Resource Allocation for HW/SW Partitioning in the LYCOS System," in *Proc. of the Conf. on Design, Automation & Test in Europe (DATE)*. Washington, DC, USA: IEEE Computer Society, 1998, pp. 22–27.

[16] R. Gupta and G. De Micheli, "HW/SW Cosynthesis for Digital Systems," *Readings in HW/SW Co-Design*, pp. 5–17, 2002.

[17] R. Dick and N. Jha, "MOGAC: A Multiobjective Genetic Algorithm for the Co-synthesis of HW/SW Embedded Systems," in *Proc. of the IEEE/ACM Int. Conf. on Computer-Aided Design (ICCAD)*. Washington, DC, USA: IEEE Computer Society, 1997, pp. 522–529.

[18] K. Chatha and R. Vemuri, "MAGELLAN: Multiway HW/SW Partitioning and Scheduling for Latency Minimization of Hierarchical Control-Dataflow Task Graphs," in *Proc. of the 9th Int. Symposium on HW/SW Codesign (CODES)*. New York, NY, USA: ACM Press, 2001, pp. 42–47.

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

[20] P.-A. Mudry, G. Zufferey, and G. Tempesti, "A dynamically constrained genetic algorithm for hardware-software partitioning," in *GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation*. New York, NY, USA: ACM Press, 2006, pp. 769–776.

[21] M. Holzer and M. Rupp, "Static Estimation of the Execution Time for Hardware Accelerators in System-on-Chips," in *International Symposium on System-on-Chip*, November 2005, pp. 62–65.

[22] C. Brandolese, W. Fornaciari, and F. Slice, "An Area Estimation Methodology for FPGA Based Designs at SystemC-Level," in *Design Automation Conference*, June 2004, pp. 129–132.

[23] B. Knerr, M. Holzer, and M. Rupp, "A Fast Rescheduling Heuristic of SDF Graphs for HW/SW Partitioning Algorithms," in *Proc. of IEEE Conf. on Communication System, Software and Middleware*, New Delhi, India, January 2006.

[24] Y.-K. Kwok and I. Ahmad, "Dynamic critical-path scheduling: An effective technique for allocating task graphs to multiprocessors," *IEEE Trans. Parallel Distrib. Syst.*, vol. 7, no. 5, pp. 506–521, 1996.

[25] "Standard task graph set," 2006, Kasahara Laboratory, Dept. of Electrical Engineering, Waseda University.