High-Level Synthesis Scheduling and Allocation using Genetic Algorithms

M.J.M. Heijligers L.J.M. Cluitmans J.A.G. Jess

Eindhoven University of Technology
Design Automation Section, Room EH 7.23
P.O. Box 513
5600 MB Eindhoven
the Netherlands
tel: -31 40 473238, fax: -31 40 464527
e-mail: M.J.M.Heijligers@ele.tue.nl

Abstract

In this article a scheduling method is presented which is capable of allocating supplementary resources during scheduling. This makes it very suitable in synthesis strategies based on lower bound estimations techniques. The method is based on genetic algorithms. Special coding techniques and analysis methods are used to improve the runtime and quality of the results. The scheduler can easily be extended to cover other architectural issues and (for example) provides ways to make trade-offs between functional unit allocation and register allocation. Experiments and comparisons show high quality results and fast run times that outperform results produced by other heuristic scheduling methods

1 Introduction

High-level synthesis translates behavioral descriptions into digital network structures. During this translation the cycle steps in which operations start their execution must be determined (scheduling problem). A schedule induces a resource allocation (because some operations are executing simultaneously) and a completion time (the cycle in which the last operation finishes its execution). Solving this problem efficiently is a non-trivial matter because of the NP-complete nature of most high-level synthesis scheduling problems.

To restrict the search space of a scheduler, lower bound estimation techniques as reported in [Timm93b], [Jain92] and [Rim92] can be used. Given a time constraint and a data flow graph these techniques try to estimate an accurate lower bound on the number of resources which are needed to schedule the graph within the time constraint. In some cases these techniques find a lower bound for which no feasible schedule exists, i.e. the completion time of the schedule exceeds the specification. Schedulers which are used in such an environment should be able to cope with such a behavior. When exact solution methods like IP scheduling ([Hwan91], [Gebo90]) are used, the danger exists that they will perform an exhaustive search because they cannot detect that a combination of constraints is infeasible, and large run times may be the result. Heuristic methods like list scheduling [Thom90], critical path scheduling [Park86] and force directed (list) scheduling [Paul89], [Verh92] are faster, but may produce unsatisfactory results because of their greedy characteristics. Methods based on approximation schemes [Deva89], [Nest90], [Wehn91] may produce better results, but in general these algorithms suffer from large run-times.

In this article a new scheduling method is presented which is based on genetic algorithms. Some previous work about genetic algorithms used for high-level synthesis scheduling will be presented, and the shortcomings of these methods will be discussed. After that, a genetic solution to the resource constrained scheduling problem is given, in such a way that it combines the speed of a heuristic algorithm with the quality of an approximation scheme. It is shown that the incorporation of problem specific knowledge improves the quality of such a scheduling method even more.

An efficient time constrained scheduler is obtained by extending the genetic encoding of the resource constrained scheduler with resource allocation information, resulting in a scheduler which has the possibility to allocate extra hardware during scheduling. Some well-known benchmarks will be presented to show the quality of results and fast execution times of this method. By incorporating other issues like register costs into the schedule method, a trade-off between functional unit allocation and register allocation can be made.

2 Genetic Algorithms and Scheduling

Genetic algorithms are probabilistic search algorithms. Given an optimization problem they try to find an optimal solution. Genetic algorithms start by initializing a set (population) containing a random selection of encoded points of the search space (individuals). By decoding the individual and determining its cost the fitness of an individual can be determined, which is used to distinguish between better and worse individuals. A genetic algorithm iteratively tries to improve the average fitness of a population by construction of new populations. A new population consists of individuals (children) constructed from individuals of the old population (parents) by use of re-combination operators. Better (above average) individuals have a higher probability to be selected for re-combination than other individuals (survival of the fittest). After some criterion is met, the algorithm returns the best individual of the population.

A theoretical foundation of genetic algorithms and their conver-

gence to an optimal solution can be found in [Gold89] (schemata theory and building block hypothesis). In contrast to the theoretical foundations, genetic algorithms have to deal with limited population sizes and a limited number of generations. This limitation can lead to premature convergence, which means that the algorithm gets stuck at local optima. A lot of research has been undertaken to overcome premature convergence (for an overview, see [Mich92]). The strategies proposed suggest different selection schemes, scaling of fitness functions, use of alternative re-combinators and/or en(de)codings. Experiments have shown that incorporation of problem specific knowledge generally improve genetic algorithms. In this paper attention will be paid how to incorporate schedule specific information into a genetic algorithm.

3 Previous Work

In [Wehn91] a scheduling method is presented based on genetic paradigms. This method does not use constraints, but searches for a trade-off between resource allocation and completion time by re-weighting a cost function. The method is based upon assigning a displacement $d_a(v)$ to each operation $v \in V$. Disadvantage of the method is that displacement of critical path operations has a large impact on the completion time of the schedule. In [Wehn91] special initialization routines are presented to construct a population containing schedules within their 'time constraint' by distributing displacements over critical paths. An improvement of the quality of the results is reported, however no attention has been paid to adapt re-combinators such that this property will be preserved during the run of the genetic algorithm.

Our own experience using a time constrained scheduler derived from this method show that only a few individuals are constructed that represent feasible schedules (i.e. within their time constraint). A possible explanation for the failure of the algorithms is the lack of problem specific knowledge inside the encoding. For example, the method does not prevent 2 additions being scheduled simultaneously, even if both operations have large schedule ranges. Simple algorithms to prevent such a behavior, without missing out on the optimal solution, are not known to us. Therefore new encodings have been developed, which make use of specific schedule knowledge.

4 Resource Constrained Scheduling

Given are a data flow graph (precedence constraints) and a resource constraint. The goal of a resource constrained scheduler is to minimize the completion time of the schedule. A well-known resource constrained scheduling technique used in high-level synthesis is list scheduling [Thom90].

The advantage of a list scheduler is that the schedules constructed always satisfy the precedence constraints and the resource constraints. A disadvantage of a list scheduler is the influence of the priority function on the quality of its results. To overcome this disadvantage a genetic algorithm can be used to search for a good priority function to direct a list scheduler. Inside the genetic algorithm the encoding of a schedule consists of a permutation of operations which can be used as a priority list for the list scheduler. The completion time of the resulting schedule is used to calculate the fitness of the individual.

Initially a population with individuals is constructed, each containing a random permutation. Schedules are constructed by decoding permutations into priority lists and applying a list scheduler. The genetic algorithm selects individuals for re-combination using stochastic sampling with replacement (also known as roulette wheel selection). Using this strategy, fit individuals have higher probability to be selected than non-fit individuals. To avoid premature convergence of the algorithm, duplication of individuals is avoided by selecting individuals at most once. Uniform crossover, mutation and inversion as discussed in [Sysw91] and [Mich92] are used as re-combinators to generate new individuals.

In general the relative difference between the completion times is quite small (less than 30%). In that case stochastic sampling with replacement acts like random search because fit individuals have no substantial higher probability to be selected than non-fit individuals. Linear scaling is used to avoid random search by subtracting a lower bound of the completion time from the completion time of the resulting schedule. An accurate lower bound of the completion time using precedence relations and resource constraint information can be calculated using the method reported in [Timm93a].

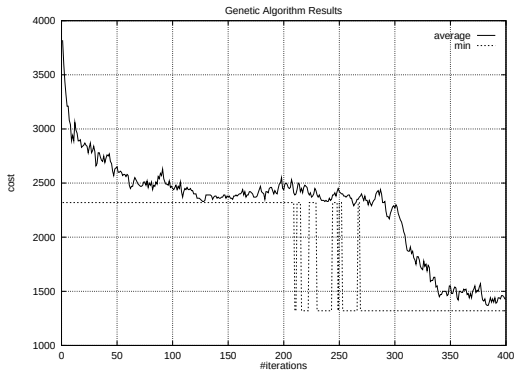

Figure 1: Convergence of genetic scheduling, fdct, 2 multipliers, 2 adders

The genetic parameters have been determined empirically, however changing these parameters within a reasonable range (< 50%) doesn't alter the performance of the algorithm substantially. Care has been taken that the average fitness of the population converges gradually towards the minimum value found so far (see figure 1). The population consists of 100 individuals, distribution of re-combinators is crossing (40%), inversion (9%), mutation (1%) and copying (50%). The genetic algorithm stops if it meets the lower time bound or if the number of iterations is 100 (in figure 1 more iterations are drawn to be able to observe the convergence behavior of the algorithm). In figure 2 the results of a random search can be found, which shows no convergence behavior and doesn't find the optimal solution.

To improve the method, special attention has been paid to prevent constructing priority lists which lead to identical schedules. In the

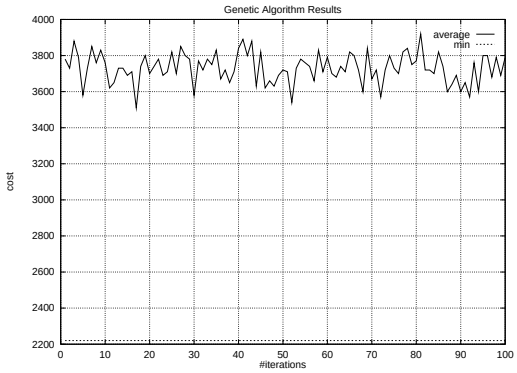

Figure 2: Random search, fdct, 2 multipliers, 2 adders

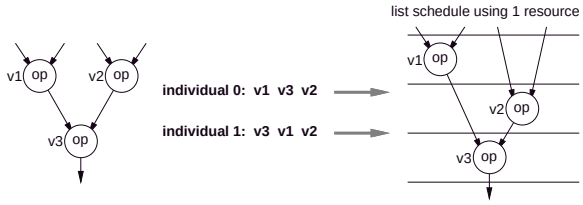

Figure 3: Permutations and lists schedules

example of figure 3 individual 0 and 1 lead to the same list schedule, hence exchanging operation v_1 and v_3 inside the permutation makes no sense.

Several observations assist in avoiding the exchange of operations leading to identical schedules:

- Only the exchange of operations whose schedule ranges (or execution intervals) have cycles in common will lead to priority lists with possibly different schedules.
- Only the exchange of nodes which don't have flow of data in common (i.e. can be executed in parallel) can lead to priority lists with different schedules. This relation among nodes can be determined by taking the complement of the transitive closure of the data flow graph.
- Only the exchange of operations which have common resource types makes sense.

These observations have been incorporated inside the genetic recombinators as follows. Before scheduling starts for each node u a list of nodes $L(u)$ is constructed that satisfies the observations mentioned before. During mutation first a node u from a permutation is selected randomly, then a node $v \in L(u)$ is selected randomly, and after that u and v are exchanged inside the permutation. During inversion first a node u from a permutation is selected randomly, then the first node $v \in L(u)$ in the permutation which comes after u inside the permutation is selected, and

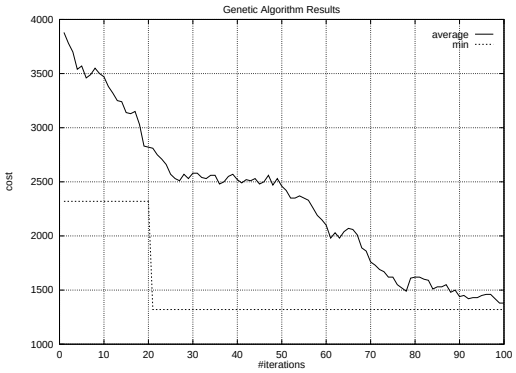

Figure 4: Convergence after incorporation of schedule specific information

a position bigger or equal than the position of node v is selected. These new extensions greatly add to a more efficient investigation of the design space (see figure 4).

In table 1 and 2 some results of the new genetic scheduler and an ordinary list scheduler using the critical path as priority function can be found (examples from [DeWi85] and [Mall90]). In all cases the new scheduler finds the optimal solution. The results show a substantial improvement of the genetic scheduler versus the ordinary list scheduler. The run times of the proposed scheduler are between 0.1 and 20 seconds for each entry of the table using an HP 9000/735 computer.

Table 1: Results for fifth-order elliptical filter

resource constraint		completion time		
mult	add	optimal	genetic	list
3	3	17	17	17
2	2	18	18	19
1	2	21	21	22
1	1	28	28	28

5 Time Constrained Scheduling

During time constrained scheduling the completion time of a schedule is restricted. The goal is to find a schedule that induces a minimal resource allocation.

A time constrained scheduler can be obtained from a resource constrained scheduler by applying a resource allocation, and after scheduling check whether the time constraint is met (see figure 5). To be sure that not too many resources are allocated (which lead to non-optimal solutions), lower bound estimations can be used [Timm93b]. In some cases these techniques find a lower bound for which no feasible schedule exists, i.e. the resource allocation induces a completion time for which the resource constrained

Table 2: Results for fast discrete cosine transform filter

resource constraint		completion time		
mult	add	optimal	genetic	list
8	4	8	8	8
5	4	10	10	10
4	3	11	11	13
4	2	13	13	15
3	2	14	14	17
2	2	18	18	21
2	1	26	26	27
1	1	34	34	40

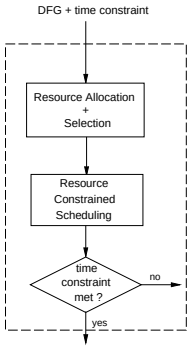

Figure 5: Time Constrained Scheduling using a Resource Constrained Scheduler

scheduler exceeds the specification. Consequently the method must have the possibility to allocate extra hardware.

The resource constrained scheduler described in the previous section can be extended to enable encoding of extra resource allocation [Clui92]. The difference between the maximum and minimum number of resources needed is encoded in a string. A lower bound on the number of resources is obtained by using the technique of [Timm93b], and an upper bound on the number of extra resources needed can be obtained by looking at the maximal parallelism for each resource type. This problem can be modelled as a min-flow max-cut problem on a comparability graph [Golu80], which can be solved in polynomial time. Each resource not part of the initial lower bound is accompanied by a binary variable which denotes whether a resource is available (1) or is not available (0) for scheduling [Clui92] (see figure 6). Standard cross and mutate re-combinators can be used to modify these classic bit-vector representations during the run of the genetic algorithm. The advantage of using a genetic strategy for the allocation of extra resources is that no complex feedback paths are necessary for re-allocation, the results of which might heavily depend on the resource constrained scheduler used and the initial resource allocation taken, which is the case in for instance [Kuma91]. The use of lower and upper bounds in combination with the list scheduler also results in a reduction of the search space of the genetic algorithm.

Figure 6: Encoding of supplementary resource allocation

The fitness of an individual is determined by a combination of the completion time and the resource allocation. A small penalty on the fitness is used to favor individuals representing schedules that are within the time constraint. If both the time constraint and resource constraint are satisfied or the number of populations exceeds 100, the scheduler stops and returns the resulting schedule.

Table 3: Scheduling run times in seconds for fdct

time constraint	genetic	ifds
10	1.30	17.7
20	1.30	101.5
30	1.34	203.0
40	1.40	327.6

The transformed resource constrained scheduler is very fast (0.1 - 20 seconds) and produces optimal solutions for all examples that have been tested.

Table 4: Results for fifth-order elliptical filter

	optimal		genetic		ifds	
cycles	mult	add	mult	add	mult	add
17	3	3	3	3	3	3
18	2	2	2	2	2	3
19	2	2	2	2	2	2
20	2	2	2	2	2	2
21	1	2	1	2	1	2
22	1	2	1	2	1	2
23	1	2	1	2	1	2
24	1	2	1	2	1	2
25	1	2	1	2	1	2
26	1	2	1	2	1	2
27	1	2	1	2	1	2
28	1	1	1	1	1	1

The schedulers proposed in this article have been compared with heuristic time constrained scheduling methods. One state-of-the-art high-level synthesis time constrained scheduling method is (improved) force-directed scheduling [Verh92]. Comparison

Table 5: Results for fast discrete cosine transform

	optimal		genetic		ifds	
cycles	mult	add	mult	add	mult	add
8	8	4	8	4	8	4
9	8	4	8	4	8	4
10	5	4	5	4	5	4
11	4	3	4	3	4	4
12	4	3	4	3	4	3
13	4	2	4	2	4	3
14	3	2	3	2	3	3
15	3	2	3	2	3	3
16	3	2	3	2	3	3
17	3	2	3	2	3	3
18	2	2	2	2	3	2
19	2	2	2	2	2	2
20	2	2	2	2	2	2
21	2	2	2	2	2	2
22	2	2	2	2	2	2
23	2	2	2	2	2	2
24	2	2	2	2	2	2
25	2	2	2	2	2	2
26	2	1	2	1	2	2
27	2	1	2	1	2	2
28	2	1	2	1	2	2
29	2	1	2	1	2	2
30	2	1	2	1	2	2
31	2	1	2	1	2	2
32	2	1	2	1	2	1
33	2	1	2	1	2	2
34	1	1	1	1	2	1

shows that the genetic time constrained scheduler offers better results, in particular for more complex schedule problems like in 5. Comparison of run times (see table 3) using our own implementation of an improved force directed scheduler shows that especially for large time constraints the new method is much faster.

6 Extensions

Because of the flexible nature of the genetic algorithm it can be easily extended with all kind of design issues, like register costs, interconnect costs and support for complex libraries in which a single operation type can get different values for its delay. In some cases only the cost function needs to be extended, in other cases the resource constrained scheduler needs to be expanded to take care of other architectural constraints. If, for instance, a trade-off between register costs and functional unit costs must be made, the cost of a schedule can be extended by the incorporation of the register allocation of the resulting schedule. The time constrained algorithm presented in the preceding section searches for the schedule with the lowest cost possible, and may increase functional unit cost to obtain a lower overall cost. Assuming that each incoming value of the data flow graph has to be stored at cycle step 0, the total area including register costs can be found in table 6. The results show that the genetic algorithm including register optimization finds schedules in which less registers are needed.

Table 6: Area fdct with register costs: area mult = 100, add = 10, reg = 10

cycles	genetic with reg. opt.	genetic without reg. opt.	ifds with reg. opt.
8	960	960	940
13	520	540	530
18	340	350	420
23	320	340	440
28	320	340	330
33	320	340	320

7 Conclusions

An efficient scheduler based on genetic algorithms has been presented which offers high quality results and fast execution times. The new resource constrained scheduler improves the results of list scheduling considerably. New re-combinators have been presented, and show that incorporation of schedule specific knowledge improves the results of the genetic scheduling algorithm. New re-combination and encoding schemes have been developed and presented to transform the resource constrained scheduler into a time constrained scheduler, which provides the possibility to allocate extra resources during scheduling. Comparison with other high-level synthesis scheduling methods show that these schedulers offer better results than other existing heuristics. Extensions, like the incorporation of register costs can be easily made, and give good results. Although the scheduling technique does not guarantee optimality, the algorithm produced optimal results for all examples tested.

References

[Clui92] L.J.M. CLUITMANS. Using Genetic Algorithms for Scheduling Data Flow Graphs. Technical Report 92-E-266, Eindhoven University of Technology, 1992.

[Deva89] S. DEVADAS AND A.R. NEWTON. Algorithms for Hardware Allocation in Data Path Synthesis. *IEEE Transactions on Computer-Aided Design*, 8(7):768–781, July 1989.

[DeWi85] P. DEWILDE, E. DEPRETTERE, AND R. NOUTA. *Parallel and Pipelined VLSI Implementation of Signal Processing Algorithms*, pages 258–264. Prentice Hall, Englewood Cliffs, 1985.

[Gebo90] C.H. GEBOTYS AND M.I. ELMASRY. A Global Optimization Approach for Architectural Synthesis. In *Digest of Technical Papers of the IEEE International Conference on Computer-Aided Design*, pages 258–261, Santa Clara, November 1990.

[Gold89] D.E. GOLDBERG, editor. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, 1989.

[Golu80] M.C. GOLUMBIC, editor. *Algorithmic graph theory and perfect graphs*. Academic Press, 1980.

[Hwan91] C.T. HWANG, Y.C. HSU, AND Y.L. LIN. A formal Approach to the Scheduling Problem in High Level Synthesis. *IEEE Transactions on Computer-Aided Design*, 10(4):464–475, April 1991.

[icc90] *Digest of Technical Papers of the IEEE International Conference on Computer-Aided Design*, Santa Clara, November 1990.

[Jain92] R. JAIN, A.C. PARKER, AND N. PARK. Predicting System-Level Area and Delay for Pipelined and Non-pipelined Designs. *IEEE Transactions on Computer-Aided Design*, 11(8):955–965, August 1992.

[Kuma91] A KUMAR, A. KUMAR, AND M. BALAKRISHNAN. A Novel Integrated Scheduling and Allocation Algorithm for Data Path Synthesis. *International Symposium on VLSI Design*, pages 212–218, January 1991.

[Mall90] D.J. MALLON AND P.B. DENYER. A New Approach To Pipeline Optimisation. In *Proceedings of the European Conference on Design Automation*, pages 83–88, Glasgow, March 1990.

[Mich92] Z. MICHALEWICZ. *Genetic Algorithms + Data Structures = Evolution Programs*. Artificial Intelligence. Springer Verlag, 1992.

[Nest90] J.A. NESTOR AND G. KRISHNAMOORTHY. SALSA: A New Approach to Scheduling with Timing Constraints. In *Digest of Technical Papers of the IEEE International Conference on Computer-Aided Design*, pages 262–265, Santa Clara, November 1990.

[Park86] A.C. PARKER, J.T. PIZARRO, AND M. MLINAR. MAHA: A program for data path synthesis. In *Proceedings of the 23th ACM/IEEE Design Automation Conference*, pages 461–466, Las Vegas, June 1986.

[Paul89] P.G. PAULIN AND J.P. KNIGHT. Force-Directed Scheduling for the Behavioral Synthesis of ASIC's. *IEEE Transactions on Computer-Aided Design*, 8(6):661–679, June 1989.

[Rim92] M. RIM AND R. JAIN. Estimating Lower-Bound Performance of Schedules Using a Relaxation Technique. In *Proceedings of the IEEE International Conference on Computer Design: VLSI in Computers and Processors*, pages 290–294, Cambridge, October 1992.

[Sysw91] G. SYSWERDA AND J. PALMUCCI. The Application of Genetic Algorithms to Resource Scheduling. In R.K. BELEW, L. BOOKER, AND J.D. SCHAFFER, editors, *Proceedings of the 4th International Conference on Genetic Algorithms*, pages 502–508, San Diego, July 1991. Morgan Kaufmann.

[Thom90] D.E. THOMAS, E.D. LAGNESE, R.A. WALKER, J.A. NESTOR, J.V. RAJAN, AND R.L. BLACKBURN. *Algorithmic and Register-Transfer Level Synthesis: The System Architect's Workbench*. Kluwer Academic Publisher, 1990.

[Timm93a] A.H. TIMMER, M.J.M. HEIJLIGERS, AND J.A.G. JESS. Fast System-Level Area-Delay Curve Prediction. In *first Asia Pacific Conference on Hardware Description Languages, Standards and Applications*, pages 198–207, Brisbane (Australia), December 1993.

[Timm93b] A.H. TIMMER, M.J.M. HEIJLIGERS, L. STOK, AND J.A.G. JESS. Module Selection and Scheduling using Unrestricted Libraries. In *Proceedings of the European Conference on Design Automation with the European Event in ASIC Design*, pages 547–551, Paris, February 1993.

[Timm93c] A.H. TIMMER AND J.A.G. JESS. Execution Interval Analysis under Resource Constraints. In *Digest of Technical Papers of the IEEE International Conference on Computer-Aided Design*, pages 454–459, Santa Clara, November 1993.

[Verh92] W.F.J. VERHAEGH, P.E.R. LIPPENS, E.H.L. AARTS, J.H.M. KORST, A. VAN DER WERF, AND J.L. VAN MEERBERGEN. Efficiency Improvements for Force-Directed Scheduling. In *Digest of Technical Papers of the IEEE International Conference on Computer-Aided Design*, pages 286–291, Santa Clara, November 1992.

[Wehn91] N. WEHN, M. GLESNER, AND M. HELD. A Novel Scheduling and Allocation Approach for Datapath Synthesis based on Genetic Paradigms. In *IFIP Working Conference on Logic and Architecture Synthesis*, pages 47–56, Paris, 1991.