Application of ASP for Automatic Synthesis of Flexible Multiprocessor Systems from Parallel Programs[⋆]

Harold Ishebabi, Philipp Mahr, Christophe Bobda, Martin Gebser, and Torsten Schaub[⋆⋆]

University of Potsdam, Institute for Informatics, August-Bebel-Str. 89, D-14482 Potsdam
{ishebabi,pmahr,bobda,gebser,torsten}@cs.uni-potsdam.de

Abstract. Configurable on chip multiprocessor systems combine advantages of task-level parallelism and the flexibility of field-programmable devices to customize architectures for parallel programs, thereby alleviating technological limitations due to memory bandwidth and power consumption. Given the huge size of the design space of such systems, it is important to automatically optimize design parameters in order to facilitate wide and disciplined explorations. Being a combinatorial problem, system design can be modeled and solved as such, but the amount of parameters renders the problem difficult to solve for large instances. However, as the synthesis problem usually exhibits structure, Answer Set Programming (ASP), for which solvers utilizing techniques from the propositional satisfiability domain are available, can be effectively employed. This paper presents a design flow based on ASP that uses the solver clasp as back-end engine. Synthesis experiments demonstrate the effectiveness of the approach.

1 Design Flow

The input to the flow in Figure 1 is a parallel program, and optionally information on task periods. The application is simulated and analyzed to obtain inter-task data traffic and task precedence information. This information is used to specify an instance of an Integer Linear Programming (ILP) problem or an ASP program. Similar to related work in this area, the other input to the design flow is information on available processing elements and communication networks, as well as their costs and constraints. In our approach, the design space is not pre-constrained, and the problem dimensions are not ranked, which ensures the optimality of solutions. For realtime systems, it is often sufficient to meet timing constraints so that the interest is not to find the fastest solution. In such situations, the flow can be used to find the smallest system instead.

The solution obtained from an ILP/ASP solver is used to generate an abstract description of the system, which is passed to further tool chains described in [1] to generate the configuration bit-stream. Because post-synthesis results could deviate from initial cost models used, new cost models can optionally be extracted after placement and routing to start a new iteration.

[⋆] Long version of this paper will appear in International Journal of Reconfigurable Computing.
[⋆⋆] Affiliated with Simon Fraser University, Canada, and Griffith University, Australia.

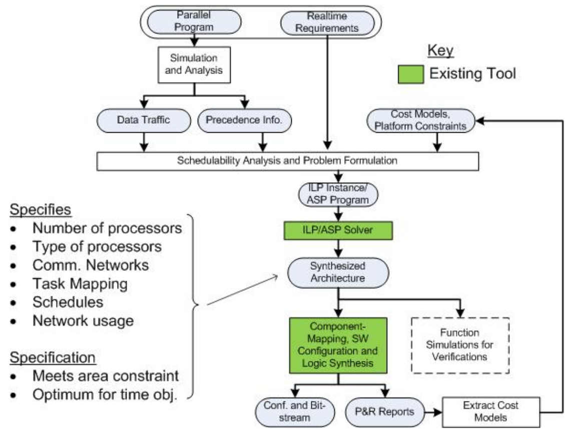


Fig. 1. Architecture Synthesis Flow

2 ASP Approach

For specifying synthesis in ASP, we build upon the ILP model proposed in [2, 3] and convert existing ILP instances into ASP programs. The general problem is to map a number of concurrent tasks on processors and communication resources such that hard space constraints are satisfied, while the throughput or the overall execution time of a parallel program is subject to optimization. We use the following notations: $I_i \in \{I_0, \dots, I_n\}$ is a task in a parallel program, $J_j \in \{J_0, \dots, J_m\}$ is a processor in an Intellectual Property (IP) library, and $C_k \in \{C_0, \dots, C_p\}$ is a communication resource in an IP library. Boolean variables x_{ij} are used to indicate whether a task I_i is mapped on processor J_j in a synthesized multiprocessor system.

To represent linear constraints of 0-1 ILP instances in ASP, we use weight constraints [4] having the general form:

$$l\ [\ \ell_0 = w_0,\ \ell_1 = w_1,\ \dots,\ \ell_n = w_n\ ]\ u. \tag{1}$$

In (1), literals $\ell_0, \ell_1, \dots, \ell_n$ are associated with weights $w_0, w_1, \dots, w_n$. The lower bound l and upper bound u can be omitted, in which case they are identified with $-\infty$ and ∞, respectively. Given this, a weight constraint (1) represents linear (in)equality $l \leq \ell_0 * w_0 + \ell_1 * w_1 + \cdots + \ell_n * w_n \leq u$. Notably, weights and bounds of weight constraints are integers, while ILP usually admits rational numbers. If not mentioned otherwise, we deal with this by rounding l as well as $w_0, w_1, \dots, w_n$ up and u down when translating ILP to weight constraints. In principal, rounding can change the semantics of a constraint, but it was unproblematic in our application.

In what follows, we describe how ASP programs are derived from ILP instances. To begin with, for each task I_i, the associated task mapping constraint [2] is given by

$$1\ [\ x_{i0} = 1,\ x_{i1} = 1,\ \dots,\ x_{im} = 1\ ]\ 1. \tag{2}$$

Such constraints stipulate that each task is mapped on exactly one processor J_j.

Processor sharing constraints [2] on program sizes s_{ij} of tasks and program memory s_j of a processor J_j are specified as

$$[\, x_{0j} = s_{0j},\ x_{1j} = s_{1j},\ \ldots,\ x_{nj} = s_{nj} \,]\ s_j. \tag{3}$$

The omission of a lower bound in (3) reflects that we admit J_j to remain unallocated, while the upper bound makes sure that the memory capacity of J_j is not exceeded.

The area constraint for processors on an FPGA can be specified in terms of three linear constraints [2]. The first one forces a Boolean variable v_j to be true if at least one task is mapped on processor J_j:

$$[\, x_{0j} = 1,\ x_{1j} = 1,\ \ldots,\ x_{nj} = 1,\ v_j = -(n+1) \,]\ 0. \tag{4}$$

The second constraint stipulates v_j to be false when J_j remains unallocated:

$$[\, x_{0j} = -1,\ x_{1j} = -1,\ \ldots,\ x_{nj} = -1,\ v_j = 1 \,]\ 0. \tag{5}$$

Finally, the third constraint connects areas (coefficients a_j) required by processors J_j on an FPGA with the available area A_J on the FPGA:

$$[\, v_0 = a_0,\ v_1 = a_1,\ \ldots,\ v_m = a_m \,]\ A_J. \tag{6}$$

Note that, if $a_0+a_1+\cdots+a_m > A_J$, it is not permissible to allocate all of the available processors.

We next consider network resources, needed whenever two communicating tasks I_{i_1}, I_{i_2} are mapped on different processors J_{j_1}, J_{j_2} for $i_1 < i_2$ and $j_1 \neq j_2$. Such a situation is indicated by an auxiliary atom $\alpha_{i_1 i_2 j_1 j_2}$, defined by the following rule:

$$\alpha_{i_1 i_2 j_1 j_2} \leftarrow x_{i_1 j_1},\ x_{i_2 j_2}. \tag{7}$$

Given this, the next constraint stipulates another auxiliary atom $\lambda_{i_1 i_2}$ to be true precisely when communicating tasks I_{i_1}, I_{i_2} require a communication resource:

$$0\,[\, \alpha_{i_1 i_2 j_0 j_1}\!=\!1, \alpha_{i_1 i_2 j_0 j_2}\!=\!1, \ldots, \alpha_{i_1 i_2 j_m j_{m-2}}\!=\!1, \alpha_{i_1 i_2 j_m j_{m-1}}\!=\!1, \lambda_{i_1 i_2}\!=\!-1 \,]\ 0. \tag{8}$$

The following constraints then stipulate exactly one communication resource C_k to be allocated for distributed communicating tasks I_{i_1}, I_{i_2}, and y_k indicates allocation of C_k:

$$0\,[\, z_{0 i_1 i_2} = 1,\ z_{1 i_1 i_2} = 1,\ \ldots,\ z_{p i_1 i_2} = 1,\ \lambda_{i_1 i_2} = -1 \,]\ 0. \tag{9}$$

$$[\, z_{k i_1 i_2} = 1,\ y_k = -1 \,]\ 0. \tag{10}$$

For a communication resource allocation, the next constraints finally check that the capacity M_k of C_k as well as the available FPGA area A_C for communication infrastructure, where a coefficient a_k gives the area required by C_k, are not exceeded:

$$[\, z_{k i_1 i_2} = 1,\ z_{k i_1 i_3} = 1,\ \ldots,\ z_{k i_{n-2} i_n} = 1,\ z_{k i_{n-1} i_n} = 1 \,]\ M_k. \tag{11}$$

$$[\, y_0 = a_0,\ y_1 = a_1,\ \ldots,\ y_p = a_p \,]\ A_C. \tag{12}$$

Furthermore, we consider scheduling feasibility constraints [3], where a parameter $F_{gj} \in \{0, 1\}$ indicates whether any group $G = \{I_{g_0}, I_{g_1}, \ldots, I_{g_l}\}$ of tasks can be mapped on a processor J_j without violating realtime constraints. The fact that all tasks in G are mapped on the same processor and the feasibility requirement are combined in the following rule:

$$1\,[\,\mathcal{M}_{gj} = 1\,]\,F_{gj} \;\leftarrow\; x_{g_0 j},\; x_{g_1 j},\; \ldots,\; x_{g_l j}. \tag{13}$$

Note that an atom $\mathcal{M}_{gj}$ is derived when all tasks in the group are mapped on J_j, and when each task in the group can meet its deadline. In the worst case, all nonempty groups of tasks in the power set of I are considered in (13), so that the cardinality of I is critical for the problem size. Avoiding such space blow-up is a subject to the future (cf. Section 4).

Finally, we turn our attention to the objective function to be minimized, dealing with scheduling costs of largest groups mapped on processors [3]. For a group G mapped on processor J_j and associated super-groups indicated by $\mathcal{M}_{s_1 j}, \ldots, \mathcal{M}_{s_h j}$, we use the next rule to derive γ_{gj} precisely when G is the largest group mapped on J_j:

$$\gamma_{gj} \;\leftarrow\; \mathcal{M}_{gj},\; \text{not}\, \mathcal{M}_{s_0 j},\; \text{not}\, \mathcal{M}_{s_1 j},\; \ldots,\; \text{not}\, \mathcal{M}_{s_h j}. \tag{14}$$

The objective function can now be stated in terms of a minimize statement [4]:

$$\begin{aligned}\text{minimize}\,[\; \ldots,\; x_{ij} = T_{ij},\; \ldots,\; \gamma_{gj} = (T'_{gj} * t_j + O_j),\\ \ldots,\; z_{ki_1i_2} = (L_k * D_{i_1i_2} + \tau_k * p_k * B_{i_1i_2}),\; \ldots\;].\end{aligned} \tag{15}$$

Note that, when optimizing in makespan mode, the weights T_{ij} expressing task execution times are set to zero for tasks I_i not on any critical path. Similarly, weights $T'_{gj} * t_j$ and O_j representing scheduling and operating system overhead are set to zero when the corresponding group G contains no critical task. Weights $L_k * D_{i_1i_2}$ and $\tau_k * p_k * B_{i_1i_2}$ represent data transfer latency and network arbitration overhead [2], respectively.

Notably, it is important to recognize that the weights in (15) cannot simply be rounded up (as with constraints) because doing so disrupts the cost structure of a problem unless small time units are used. However, using such small units can result in huge numbers, which can easily overflow the computation of the value of the objective function. Instead of time units, we thus use processor cycles, normalized by the slowest processor and the smallest weight occurring in the objective function.

3 Comparison of Synthesis Results

We applied conflict-driven learning ASP solver clasp [5], embedded into ASP system clingo [6] (version 2.0.2), to synthesis problems translated from ILP. Previous empirical investigations [5] have shown that learning solvers perform well, at least for structured problems like the ones on synthesis. For comparison, we used ILP solver lp_solve [7] (version 5.5.0.14). We conducted experiments with five applications described in [2]: filtering (FIR), Derivation, Simpson's method, N-Body problem, and matrix Inversion. Figure 2 summarizes runtimes for synthesis scenarios using 16 processors and 5 communication resources for increasing number of tasks from 4 to 22. All benchmarks were run on a machine equipped with a 1.66GHz T5500 processor and 2048Mb of main memory, using a timeout of 28,800 seconds.

All columns that terminate at the boundary of 28,800s in Figure 2 indicate that a suboptimum solution was found. Columns that exceed the boundary, i.e., those which touch the 100,000s line, indicate that no solution was found by timeout. Otherwise, optimum solutions were obtained. The results show that ASP-based synthesis outperforms ILP for 4 to 12 tasks. Beyond that region, ILP-based synthesis is either faster or at least finds a suboptimum solution by timeout. A closer look however showed that, when ASP-based synthesis performs badly, then most of the runtime is spent on reading ASP programs. In fact, in almost all cases where clingo timed out, clasp did not get any chance to start searching. The reason was that, when reading large ASP programs generated for greater number of tasks, clingo was running out of memory and swapped. As scheduling feasibility (13) is mainly responsible for the observed file size explosion, a more compact representation of it would likely enable scaling up ASP-based synthesis.

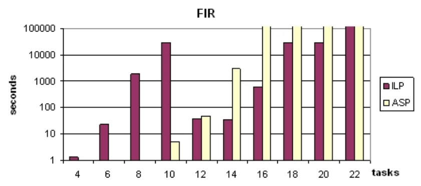
4 Summary and Conclusion

We have presented a method for automated architecture synthesis of FPGA multiprocessor systems using ASP. We showed that ASP-based synthesis has a great potential for solving difficult system design problems. Our continuing work addresses the compact representation of scheduling feasibility constraints to avoid file size explosion. Moreover, as our ASP programs were obtained from existing ILP instances, we made use of available ASP solving technology, but not (yet) of knowledge representation capacities of ASP. Future work includes the development of direct ASP solutions for automated synthesis, building on a uniform encoding and grounding.

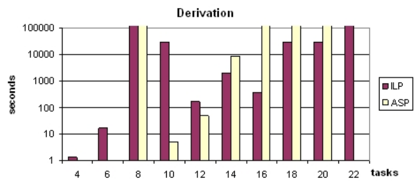
Acknowledgments. This work was partially funded by DFG under Grant BO 2480/5-1 and under Grant SCHA 550/8-1.

References

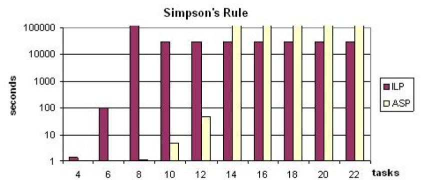
1. Bobda, C., Haller, T., Mühlbauer, F., Rech, D., Jung, S.: Design of adaptive multiprocessor on chip systems. In Petraglia, A., Pedroni, V., Cauwenberghs, G., eds.: Proceedings of the Twentieth Annual Symposium on Integrated Circuits and Systems Design (SBCCI'07), ACM Press (2007) 177–183
2. Ishebabi, H., Bobda, C.: Automated architecture synthesis for parallel programs on FPGA multiprocessor systems. Microprocessors and Microsystems **33**(1) (2009) 63–71
3. Ishebabi, H., Mahr, P., Bobda, C.: Automatic synthesis of multiprocessor systems from parallel programs under preemptive scheduling. In Torres, C., ed.: Proceedings of the International Conference on Reconfigurable Computing and FPGAs (ReConFig'08), IEEE Press (2008) 19–24
4. Simons, P., Niemelä, I., Soininen, T.: Extending and implementing the stable model semantics. Artificial Intelligence **138**(1-2) (2002) 181–234
5. Gebser, M., Kaufmann, B., Neumann, A., Schaub, T.: Conflict-driven answer set solving. In Veloso, M., ed.: Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI'07), AAAI Press/MIT Press (2007) 386–392
6. Gebser, M., Kaminski, R., Kaufmann, B., Ostrowski, M., Schaub, T., Thiele, S.: A user's guide to `gringo`, `clasp`, `clingo`, and `iclingo`. Available at `http://potassco.sourceforge.net`
7. `http://lpsolve.sourceforge.net/5.5/`



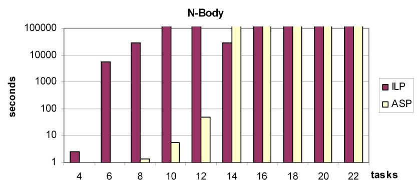
(a) FIR



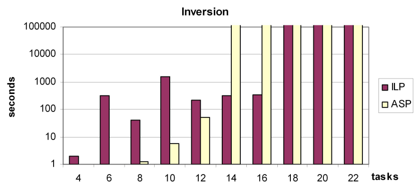
(b) Derivation



(c) Simpson



(d) N-Body



(e) Inversion

Fig. 2. ASP-ILP Comparison: Increasing the number of tasks