A Combined Mapping and Routing Algorithm for 3D NoCs Based on ASP

Benjamin Andres, Martin Gebser and Torsten Schaub
University of Potsdam, Germany
{bandres,gebser,torsten}@cs.uni-potsdam.de

Christian Haubelt
University of Rostock, Germany
christian.haubelt@uni-rostock.de

Felix Reimann and Michael Glaß
University of Erlangen-Nuremberg, Germany
{felix.reimann,glass}@cs.fau.de

Abstract

Networks on a Chip (NoCs) have been proposed to solve the communication challenges in Multi-Processor Systems on a Chip (MPSoCs) with ever increasing number of processing cores. They provide high bandwidth in combination with high connectivity, which is especially interesting in 3D integrated systems. However, efficiently exploiting both, processors and communication infrastructure, also requires new design approaches when mapping applications onto such complex architectures. While state of the art approaches show good mapping performance for systems with only a few routing options, they tend to fail in the presence of densely connected systems. In this paper, we propose a novel combined mapping and routing algorithm based on Answer Set Programming (ASP). A case-study composed of three dimensional Networks-on-Chip (3D NoC) illustrates the scalability of our proposed mapping and routing approach.

1. Introduction

The increasing complexity of interdependent decisions in embedded computing systems design, namely allocation, mapping & routing, and scheduling & arbitration, demands for compact design space representations and highly efficient automatic decision engines, resulting in automatic *system synthesis* approaches. Especially, formal methods have shown to be useful in past. (Pseudo-)Boolean Satisfiability (PB/SAT; [BHvW09]) solving has been successfully applied in the past to such problems. In particular, explicit modeling of routing decisions in PB formulas has recently enhanced the range of applicability of PB/SAT solvers in synthesizing networked embedded systems [Luk09].

Step-based approaches to combined mapping and routing as employed in [Luk09], work well in the presence of system specifications offering a limited amount of routing options. Such system specifications can be found, e.g., in the bus-based Multi-Processor System-on-Chip (MPSoC) domain.

However, there is a trend towards densely connected networks also for single-chip multi-processor systems. In fact, future MPSoCs are expected to be composed of several hundred processors connected by Networks-on-Chip (NoC) [Bor07]. Hence, mapping and routing approaches will face vast design spaces for densely connected networks, resulting in prohibitively long solving times when using step-based approaches.
In this paper, we investigate a combined mapping and routing algorithm relying on reachability for message routing. We propose a formal approach employing Answer Set Programming (ASP; [Bar03]), a solving paradigm stemming from the area of Knowledge Representation. In contrast to PB/SAT, ASP provides a rich modeling language as well as a more stringent semantics, which allows for succinct design space representations. In particular, ASP supports expressing reachability directly in the modeling language. As a result, much smaller problem descriptions lead to significant reductions in solving time for densely connected networks.
After surveying related work, Section 3 introduces the mapping and routing setting studied in the sequel, while Section 4 gives a brief introduction to problem solving with ASP. Section 5 provides dedicated ASP formulations of the combined mapping and routing problem. The experiments in Section 6 illustrate the effectiveness of our ASP-based approach. Section 7 concludes the paper.

2. Related Work

Symbolic system synthesis approaches based on Integer Linear Programming (ILP) can be found in the area of hardware/software partitioning (cf. [NM97]). Such approaches were often limited to the classical bipartitioning problem, i.e., the target platform is composed of a CPU and an FPGA. An extension towards multiple resources and a simple single-hop communication mapping can be found in [BTT98]. In the same work, SAT is reduced to the problem of computing feasible allocations and bindings in platform-based system synthesis approaches, thereby showing that system synthesis is NP-complete. In turn, [HTFM03] shows how to reduce the system synthesis problem to SAT in polynomial time; this allows for symbolic SAT-based system synthesis. An analogous approach based on binary decision diagrams is presented in [Nee01]. Since the space requirements of binary decision diagrams may grow exponentially, it could only be applied to small systems. In [Luk08], a first approach to integrate linear constraint checking into SAT-based system synthesis is reported, leading to a PB problem encoding. All aforementioned approaches still use simple single-hop communication as underlying model. However, single-hop communication models are no longer appropriate when designing complex multi-core systems.
In [Luk09], the authors show how to perform symbolic system synthesis including multi-hop communication routing with PB solving techniques. However, the PB-based approach published in [Luk09] does not scale well for system specifications permitting many routing options. The reason lies in the step-based routing encoding. In contrast, our proposed approach exploits semantic features of ASP in expressing reachability. As a consequence, symbolic system synthesis can be applied to more complex system specifications based on densely connected communication networks. Topologies providing many routing options are typically based on Networks on Chip (NoCs) [KJS+02]. With the emergence of multi-layered chip technology, novel NoC architectures, utilizing a third routing dimension, become possible (3D NoC). In contrast to traditional NoC architectures these new architecture offer an increased number of interconnections, reducing network latency and power consumption [PF06, FP09]. The additional routing options of 3D NoCs magnify the difficulty of combined mapping and routing. [KPSD11] proposes an XYZ multi-cast

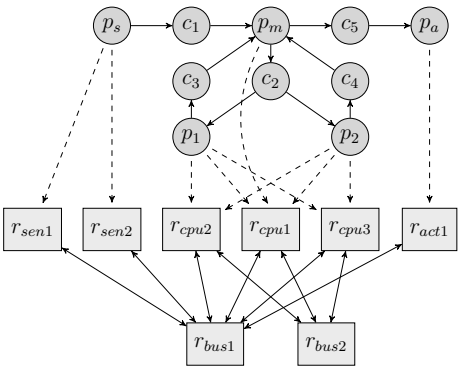


Figure 1: A system model consisting of a task graph, a platform graph and mapping options.

routing approach. Multible-layer-per-hop [RL09] routing reduces the hardware consumption of the network at the cost of reduced routing options. In [YL08] an application specific synthesis algorithm for 3D NoC with separated mapping and routing is proposed. However, no combined mapping and routing approach for 3D NoCs based on ASP exists today.
The potential of ASP for system synthesis was already discovered in [IMB$^+$09], where it was shown to outperform an ILP-based approach by several orders of magnitude. In contrast to our work, the system synthesis problem considered in [IMB$^+$09] does not involve multi-hop communication routing. Moreover, contrary to the genuine ASP encoding(s) developed in Section 5, the one in [IMB$^+$09] was derived from an ILP specification without making use of any elaborate ASP features.

3. Combined Mapping and Placement Problem

In order to automate the mapping and routing of an application onto a a system, the application is modeled by a *task graph* (T, E_T). Its vertices T represent *tasks* and are bipartitioned into *process* tasks P and *communication* tasks C, that is, $T = P \cup C$ and $P \cap C = \emptyset$. The directed edges $E_T \subseteq (P \times C) \cup (C \times P)$ model data and control dependencies between tasks, where every communication task has exactly one predecessor and an arbitrary (positive) number of successors, thus assuming single-source multicast communication.
An exemplary task graph is shown in the upper part of Figure 1. The leftmost task p_s reads data from a sensor and sends it to a master task p_m via communication task c_1. The master task then schedules the workload and passes data via communication task c_2 on to the worker tasks p_1 and p_2. Both workers send their results back to the master via communication tasks c_3 and c_4. Finally, the master uses the combined result to control an actuator task p_a via communication task c_5.
The system architecture is modeled by a *platform graph* (R, E_R). Its vertices R represent resources like processors, buses, memories, etc., and the directed edges $E_R \subseteq R \times R$ model communication connections between them. Given a task graph $(P \cup C, E_T)$ and a platform graph (R, E_R), mapping options of processing tasks $p \in P$ are determined by $R_p \subseteq R$, providing resources on which p can be implemented. The lower part of Figure 1 shows a platform graph containing six computational and two communication resources along with 18 connections. It is assumed that communication tasks can be routed via every resource. The mapping and routing model also includes a function $f : P \times R \to \mathbb{N}$, returning the work load generated by mapping a process to a resource in percent.

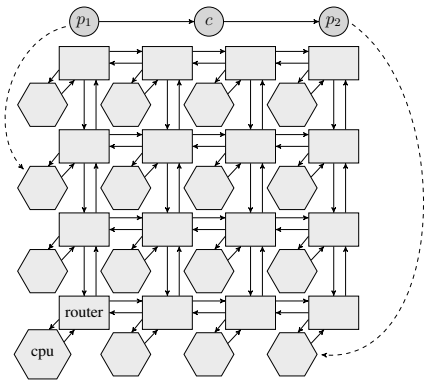


Figure 2: A possible mapping of two communicating processes to resources connected via a 4x4 mesh network.

Following the system synthesis model from [Luk09], the combined mapping and routing problem can be defined as follows. For (T, E_T) and (R, E_R) as above, select a binding $b : T \to 2^R$ such that the following conditions are fulfilled:

- $b(t) \subseteq R_t$ for each task $p \in P$,
- $|b(p)| = 1$ for each process task $p \in P$,
- $\sum_{b(p)=r} f(p, r) \leq 100$ for all resources R, and
- for each $(p, c) \in (P \times C) \cap E_T$, there is an arborescence $(b(c), E)$ with root $r \in b(p)$ such that $E \subseteq E_R$ and $\{\hat{r} \mid (c, \hat{p}) \in E_T, \hat{r} \in b(\hat{p})\} \subseteq b(c)$.

The first condition requires that each processing task may only mapped to a valid resource. Furthermore, each processing task must be mapped exactly once, while the total workload of any resource in the network may not exceed 100 percent, as enforced by the second and third condition respectively. The last condition requires that each communication task is routed (acyclicly) form the sender resource to all resources of its targets.
For the example in Figure 1, a feasible implementation is given by with the following mapping b:

$$\begin{array}{ll} b(p_s) = \{r_{sen1}\} & b(c_1) = \{r_{sen1}, r_{bus1}, r_{cpu1}\} \\ b(p_m) = \{r_{cpu1}\} & b(c_2) = \{r_{cpu1}, r_{bus1}, r_{cpu2}, r_{bus2}, r_{cpu3}\} \\ b(p_1) = \{r_{cpu2}\} & b(c_3) = \{r_{cpu2}, r_{bus2}, r_{cpu1}\} \\ b(p_2) = \{r_{cpu3}\} & b(c_4) = \{r_{cpu3}, r_{bus2}, r_{cpu1}\} \\ b(p_a) = \{r_{act1}\} & b(c_5) = \{r_{cpu1}, r_{bus1}, r_{act1}\} \end{array}$$

A possible routing of c_2 leads from the resource r_{cpu1} of the master task p_m over r_{bus1}, r_{cpu2}, and r_{bus2} to r_{cpu3}, thus visiting the resources r_{cpu2} and r_{cpu3} of the workers p_1 and p_2.
The approach in [Luk09] utilizes a step-based routing algorithm, working well for sparcely connected networks, inducing a limited amount of of routing options. However, the representation of routing options scales proportionally to $|E_R| * |R|$, given that resources may be pairwisely connected and each resource may be visited in the worst case.
As a consequence, for densely connected networks, such as (3D) NoCs, the size required for a step-based representation of routing options can be prohibitively large. For example, let us consider

possible routes from (the resource of) a sender p_1 to p_2 available in the 4x4 mesh network shown in Figure 2. The longest of these routes pass all 16 routers and potentially visit any of them at each of the 15 intermediate steps. This yields $16*15 = 240$ routing options per communication task to represent the message exchange between routers, since the task may potentially be routed over all of the 16 routers at each of the 15 intermediate steps. On the other hand, for inductively verifying whether a message reaches its target(s), it is sufficient to consider individual routing hops without relying on an explicit order given by steps. The latter strategy scales linearly in $|E_R|$, thus avoiding a significant blow-up in space. As the semantics of ASP inherently supports efficient representations of inductive concepts like reachability, the potential space savings motivate our desire to switch from the PB-based approach in [Luk09] to using ASP instead.

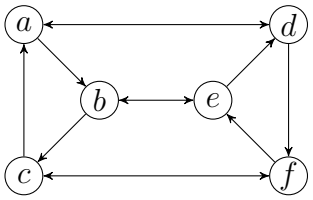
4. Answer Set Programming

The basic idea of ASP is to represent a given problem by a logic program[1] such that particular models, called answer sets, correspond to solutions, and then to use an ASP solver for finding answer sets. While ASP's input language is inspired by Logic Programming and thus allows for specifying first-order logical rules, the computation of answer sets relies on instantiation (or grounding) followed by Boolean Constraint Solving. The model-oriented approach of ASP shares similarities with PB/SAT-based problem solving; for instance, Kautz and Selman [KS92] pioneered SAT planning by devising propositional theories such that models (not proofs) describe solutions, and logic programs whose answer sets represent plans were provided by Lifschitz [Lif02]. Although we here refrain from going into semantic details (cf. online appendix), an important advantage of ASP in comparison to PB/SAT lies in its more stringent notion of modelhood, requiring any true atom to be "constructible" by applying the rules of a logic program. This constructive flavor allows for more succinct representations of inductive concepts like closures, fixpoints, and reachability than available in PB/SAT.[2]

For a pragmatic introduction to ASP, we outline its application to the well-known Hamiltonian cycle problem. The first step in solving this problem for the graph shown in Figure 3(a) consists of describing the graph in terms of facts, yielding the ASP *instance* (i.e., a logic program consisting solely of facts) in Figure 3(b). Observe that the representation of a graph by an associated ASP instance is straightforward; for example, the atoms `node(a)`, `edge(a,b)`, and `edge(a,d)` stand for node a along with its outgoing edges (a, b) and (a, d). The second and more sophisticated step is to specify logical rules such that answer sets satisfying them match problem solutions. Our rules describing Hamiltonian cycles (for arbitrary instances) are shown in Figure 3(c); such an instance-independent logic program part is called an ASP *encoding*. While Line 1, 4, and 7 give comments only, the rules in Line 2 and 3, "generating" Hamiltonian cycle candidates, can be read as follows: for each instantiation of first-order variable `X` (or `Y`) such that `node(X)` (or `node(Y)`) holds (i.e., belongs to an answer set), `cycle(X,Y)` must hold for exactly one instantiation of `Y` (or `X`) such that `edge(X,Y)` holds. Given the facts in Figure 3(b), the variable-free (or ground) rules obtained by instantiating `X` (or `Y`) with `a` are as follows:

[1] In view of ASP's quest for declarativeness, the term *program* is of course a misnomer but historically too well established to be dropped.

[2] Under common assumptions in complexity theory, any vocabulary-preserving translation from ASP to SAT must be worst-case exponential [LR06], while there are linear-size as well as modular translations from SAT to ASP.



(a) A directed graph.

```
node(a). edge(a,b). edge(a,d).
node(b). edge(b,c). edge(b,e).
node(c). edge(c,a). edge(c,f).
node(d). edge(d,a). edge(d,f).
node(e). edge(e,b). edge(e,d).
node(f). edge(f,c). edge(f,e).
```

(b) ASP instance describing the graph in (a) by facts.

```
1 % GENERATE
2 1 { cycle(X,Y) : edge(X,Y) } 1 :- node(X).
3 1 { cycle(X,Y) : edge(X,Y) } 1 :- node(Y).
4 % DEFINE
5 reached(X) :- X := #min[ node(Y) = Y ].
6 reached(Y) :- reached(X), cycle(X,Y).
7 % TEST
8  :- node(Y), not reached(Y).
```

(c) ASP encoding of the Hamiltonian cycle problem.

Figure 3: Solving the Hamiltonian cycle problem with ASP.

```
1 { cycle(a,b), cycle(a,d) } 1 :- node(a).
1 { cycle(c,a), cycle(d,a) } 1 :- node(a).
```

Observe that the atoms `cycle(a,b)` and `cycle(a,d)` (or `cycle(c,a)` and `cycle(d,a)`) correlate with outgoing (or incoming) edges of node a in Figure 3(a). The lower as well as upper bound `1` of the so-called cardinality constraints over outgoing or incoming edges, respectively, express that exactly one edge of each kind must contribute to a Hamiltonian cycle, while atoms representing the edges as such are merely used to instantiate, but not subject to, the cardinality constraints. The rules in Line 5 and 6 contribute the key ingredient to our encoding by "defining" reachability wrt a generated Hamiltonian cycle candidate. While the rule in Line 5, making use of built-in features offered by the ASP grounder *gringo* [GKK$^+$11], is instantiated to a fact providing some (arbitrary) starting node of a cycle, instantiations of the rule in Line 6 trace the edges of a Hamiltonian cycle candidate to thus obtain the closure of reached nodes. For example, the ground rules obtained from the rule in Line 5 and by instantiating `Y` with `b` in Line 6 are:

```
reached(a).
reached(b) :- reached(a), cycle(a,b).
reached(b) :- reached(e), cycle(e,b).
```

Ground rules similar to the last two are also obtained by instantiating `Y` with `c`, `d`, `e`, and `f` instead of `b` in Line 6. As a consequence, an atom `reached(`n`)` is constructible by applying (ground) rules precisely if node n is reached from a singular starting node via the edges of a Hamiltonian cycle candidate. Given this, the so-called integrity constraint in Line 8 (the omitted left-hand side of the implication expressed by "`:-`" refers to an implicitly false consequence) denies candidates such that `reached(`n`)` cannot be constructed for some node n. For example, by instantiating `Y` with `b`, we get:

```
:- node(b), not reached(b).
```

While such (ground) integrity constraints require `reached(`n`)` to hold for each node n, they

do not provide a construction of `reached(`n`)`; reachability of n (from some starting node) is thus faithfully captured by `reached(`n`)`. Indeed, the Hamiltonian cycle (a, b, c, f, e, d, a) of the graph in Figure 3(a) coincides with an answer set containing `cycle(a,b)`, `cycle(b,c)`, `cycle(c,f)`, `cycle(f,e)`, `cycle(e,d)`, and `cycle(d,a)`. Unlike this, the candidate including `cycle(a,b)`, `cycle(b,c)`, `cycle(c,a)`, `cycle(d,f)`, `cycle(f,e)`, and `cycle(e,d)` does not yield an answer set since `reached(d)`, `reached(e)`, and `reached(f)` are not constructible (in view of unconnected subcycles). Finally, note that the size of an instantiation of the encoding in Figure 3(c) is linear in the size of an instance, such as the one in Figure 3(b), given that each first-order rule refers to an individual node or edge specified within the instance.

5. ASP Encoding

The ASP instance for a task graph $(P \cup C, E_T)$ and a platform graph (R, E_R) along with the underlying mapping options, $(R_p)_{p \in P}$ and the work load function f is defined as follows:

$$\begin{aligned}
\{\texttt{pt(}p\texttt{)} \texttt{.} \mid p \in P\} \cup \\
\{\texttt{send(}p,c\texttt{)} \texttt{.} \mid (p,c) \in E_T, p \in P, c \in C\} \cup \\
\{\texttt{read(}p,c\texttt{)} \texttt{.} \mid (c,p) \in E_T, p \in P, c \in C\} \cup \\
\{\texttt{pr(}r\texttt{)} \texttt{.} \mid p \in P, r \in R_p\} \cup \\
\{\texttt{load(}p,r,l\texttt{)} \texttt{.} \mid p \in P, r \in R_p, l = f(p,r)\} \cup \\
\{\texttt{edge(}r,s\texttt{)} \texttt{.} \mid (r,s) \in E_R\} \cup \\
\{\texttt{s(}i\texttt{)} \texttt{.} \mid i \in \{1,\ldots,n\}\}
\end{aligned} \tag{1}$$

While the first six sets capture primary constituents of a problem instance, the introduction of atoms `s(`i`)` for $1 \leq i \leq n$ is needed to account for the PB formulation in [Luk09] in a faithful way. Two alternative ASP encodings for combined mapping and routing in 3D NoCs systems are shown in Figure 4(a) and 4(b). The rule in Line 2 of each encoding specifies that every processing task provided in an instance must be mapped to exactly one of its associated options. Observe that the mapping of processing tasks p to resources r is represented by atoms `map(`p`,`r`)` in an answer set. This provides the basis for further specifying communication routings. The integrity constraint in Line 4 of both encodings ensures that the workload of every resource is not above 100 percent.

Despite of syntactic differences, the step-oriented encoding ASP(S) in Figure 4(a) stays close to the original PB formulation of constraints, from [Luk09]. In particular, it uses atoms `reached(`c`,`r`,`i`)` to express that some message of communication task c is routed over resource r at step i. Note that the omission of lower and upper bounds for the cardinality constraint in the rule form Line 9 means that there is no restriction on the number of atoms constructed by applying the rule. The (trivially satisfied) cardinality constraint is still important because, it allow us to successively construct `reached(`c`,`r`,`i`)`. Given such atoms, instantiations of the rule in Line 12 (where "_" stands for an unreused anonymous variable) further provide us with projections `reached(`c`,`r`)`, These are used in the integrity constraints in Line 14 and 16, excluding cases where a communication task is routed over the same resource at more than one step or does not reach some of its targets, respectively.

The encoding in Figure 4(b), denoted by ASP(R), utilizes ASP's "built-in" support of recursion to implement routing without step counting. To still guarantee an acyclic routing of communication tasks, the idea of ASP(R) is to (recursively) construct non-branching routes from resources of communication targets back to the resource of a sending task, where the construction stops. This

```
% map each process task to a resource
1 { map(P,R) : pr(R) } 1 :- pt(P).
% ensure the limits of processing units
:- pr(R), 101[ cost(R,P,C) : map(P,R) = C ].

% step zero of communication task
reached(C,R,0) :- send(P,C), map(P,R).
% forward steps of communication task
{ reached(C,S,I+1) : edge(R,S) } :- reached(C,R,I), s(I+1).

% resources of communication task
reached(C,R) :- reached(C,R,_).
% reach each resource at most once
 :- reached(C,R), 2 { reached(C,R,_) }.
% reach communication target resources
 :- read(P,C), map(P,R), not reached(C,R).
```

(a) Step-oriented encoding ASP(S).

```
% map each process task to a resource
1 { map(P,R) : pr(R) } 1 :- pt(P).
% ensure the limits of processing units
:- pr(R), 101[ cost(R,P,C) : map(P,R) = C ].

% root resource of communication task
root(C,R) :- send(P,C), map(P,R).
% resources of communication task per target
sink(C,R,P) :- read(P,C), map(P,R).
sink(C,R,P) :- sink(C,S,P), reached(C,R,S).
% reach communication root resource
 :- read(P,C), root(C,R), not sink(C,R,P).

% resources of communication task
reached(C,R) :- sink(C,R,_).
% backward hops of communication task
1 { reached(C,R,S) : edge(R,S) } 1 :- reached(C,S), not root(C,S).
```

(b) Recursive encoding ASP(R).

Figure 4: Two alternative ASP encodings of system synthesis.

recursive approach connects each encountered target resource to exactly one predecessor, where the only exception is due to the sender of a communication task, whose resource, specified by an atom `root(`c`,`r`)`, is not connected back. Finally, the integrity constraint in Line 10 requires that each target of a communication task is located on a route starting at the sender's resource. Note that the target-driven routing approach implemented in ASP(R) intrinsically omits redundant message hops (not leading to communication targets). While this is an improvement over ASP(S), it is not the real achivement of ASP(R), but abolishing one problem dimension by disusing explicit step counters is.

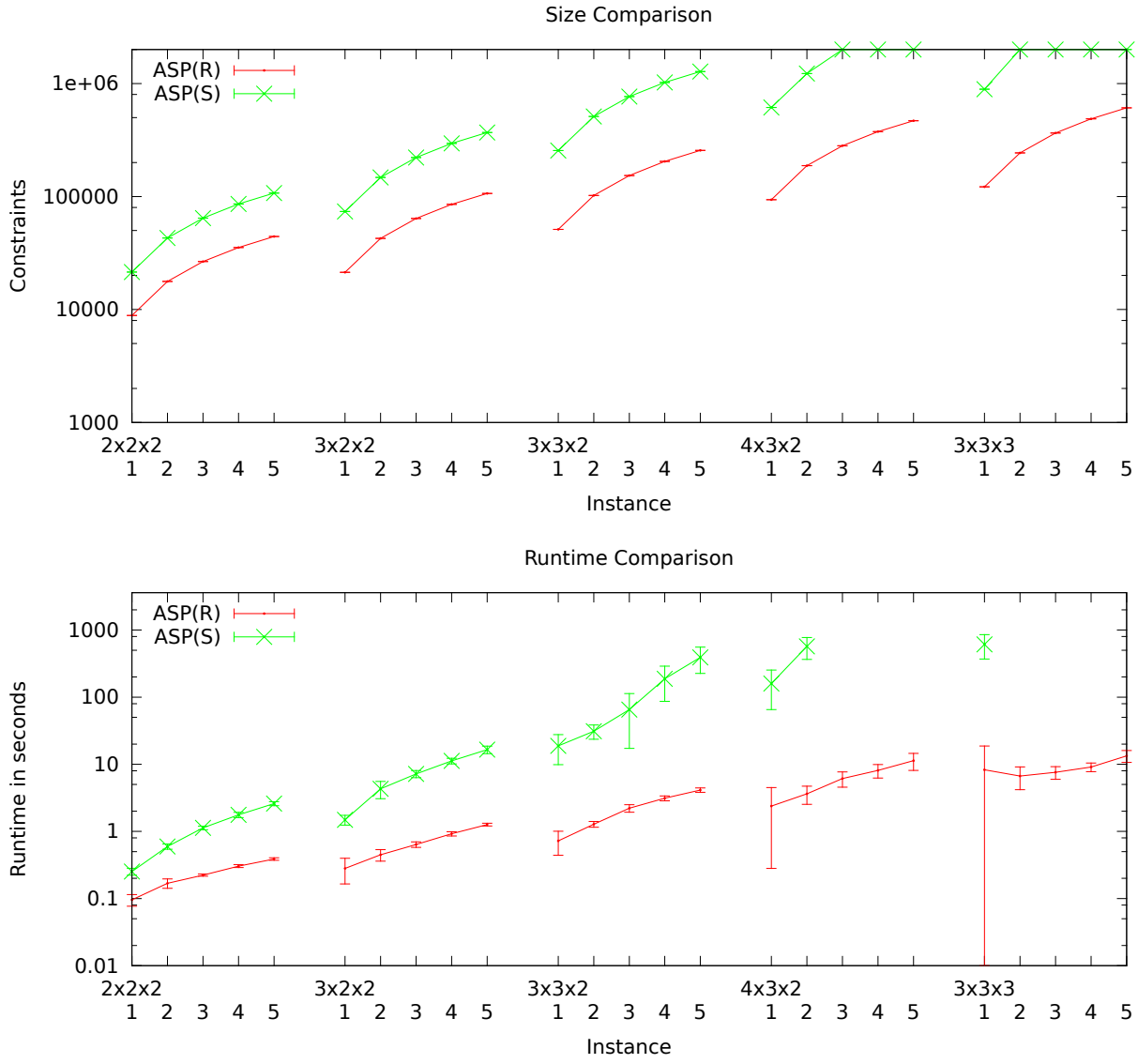


Figure 5: Average numbers of constraints and runtimes in seconds for 3D NoC of varying sizes and task numbers.

6. Experiments

For evaluating our approach, we conducted systematic experiments contrasting our two ASP encodings, ASP(S) and ASP(R), in terms of problem representation size and solving time. To this end, we consider a series of crafted three dimensional mesh network system models of varying sizes. Each instance is generated with a number of CPUs $(x * y * z)$ connected in a mesh network structure and a fixed ratios (r) between the number of processing tasks and available CPUs. Each task consumes between $\frac{50}{x*y*z*r}$ and $\frac{100}{x*y*z*r}$ percent processing power of one CPU and sends one message to one other, randomly chosen, task. The ASP grounder *gringo* (version 3.0.3) combines these instances with either of our encodings, ASP(S) and ASP(R), generating standardized text formats, processable by the ASP solver *clasp* (version 2.0.3; [GKK$^+$11]).

After instance generation, accomplished offline, we measured (sequential) runtimes of *clasp* on a Linux machine equipped with 3.4GHz Intel Xeon CPUs and 32GB RAM total. The search strategies of *clasp* were configured via command-line switches `--heuristic=vsids` and `--save-progress`, which in preliminary experiments turned out to be helpful for solving ASP(S) as well as ASP(R) instances. In order to compensate for randomness in problem generation, we report averages over 16 distinct instances per mesh size and task number. Also note that all generated instances are satisfiable. We restricted single runs of *clasp* on a ASP(S), or ASP(R) instance to 3600 seconds time and 1GB RAM. Noise effects are excluded by taking the mean runtime over three (reproducible) runs of *clasp* per instance.

Figure 5 displays average numbers of of constraints, as reported by *clasp*, and average runtimes of *clasp*, with timeouts taken as 3600 seconds, over mesh networks of varying sizes ($2 \times 2 \times 2$, $3 \times 2 \times 2$, ...) and increasing task numbers (1, 2, ... per CPU), both given along the x-axes; standard deviations are shown as vertial bars through measurements. The average numbers of constraints reported in the left chart provide an indication of problem representation size incurred by ASP(S) and ASP(R). We observe regular scalings here, and ASP(S) is not only the more space-consuming approach, but also scales less as the network size becomes larger. In fact, the ASP(S) approach is not able solve networks with $4 \times 3 \times 2$ ($3 \times 3 \times 3$) CPUs and more than 2 (1) tasks

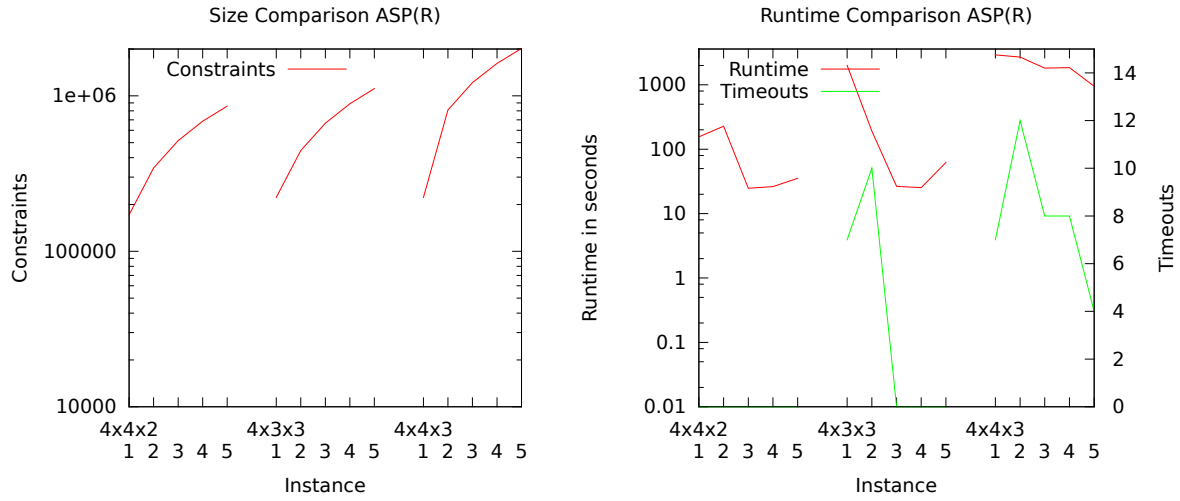


Figure 6: Average numbers of constraints and runtimes in seconds for 3D NoC of varying sizes and task numbers for scaling the ASP(R) approach.

per CPU with 1GB RAM. The corresponding average runtimes in the lower chart tightly correlate to representation sizes. While ASP(S) can still cope with small instances, it is more than one order of magnitude slower than ASP(R) for larger instances.
For investigating the further scaling behavior, we applied ASP(R) to larger 3D NoCs as shown in Figure 6. While the problem representation size scales as expected form the previous runs, the average runtime rises drasticly, even encountering timeouts. Since the problem representation size (cf. numbers of constraints) is linear in the input for ASP(R), the timeouts on large instances are explained by increasing variance of solving performance in view of randomness in problem generation. Interestingly, instances with fewer tasks per CPU are more prone to timeouts than those with larger numbers. This correlates with the workload assignment, limiting the placement options for instances with fewer tasks per CPU.

7. Conclussion

We proposed a novel approach to combined mapping and routing using ASP. The succinct ASP formulation of reachability, as required in multi-hop routing, outperforms the step-based approach when applied to densely connected 3D NoC, providing vast routing options. Such performance gains are made possible by considerably smaller design space representations and accordingly reduced search efforts. Given that ASP solvers like *clasp* also support optimization, the presented ASP approach could be extended to linear and, with some adaptions, even be utilized for non-linear optimization, as previously performed in design space exploration via evolutionary algorithms [Luk08]. At user level, the declarative first-order modeling language of ASP facilitates prototyping as well as adjustment of ASP solutions for new or varied application scenarios, making it a worthwhile alternative to purely propositional formalisms.

References

[Bar03] Baral, C.: *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press, 2003.

[BHvW09] Biere, A., M. Heule, H. van Maaren, and T. Walsh (editors): *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2009.

[Bor07] Borkar, S.: *Thousand core chips: a technology perspective*. In *Proc. of DAC '07*, pages 746–749, 2007.

[BTT98] Blickle, T., J. Teich, and L. Thiele: *System-level synthesis using Evolutionary Algorithms*. J. Design Automation for Embedded Systems, 3(1):23–58, 1998.

[FP09] Feero, B. and P. Pande: *Networks-on-chip in a three-dimensional environment: A performance evaluation*. IEEE Trans. Computers, 58(1):32–45, 2009.

[GKK+11] Gebser, M., R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, and M. Schneider: *Potassco: The Potsdam answer set solving collection*. AI Communications, 24(2):105–124, 2011.

[HTFM03] Haubelt, Christian, Jürgen Teich, Rainer Feldmann, and Burkard Monien: *SAT-Based Techniques in System Design*. In *Proc. of DATE '03*, pages 1168–1169, 2003.

[IMB+09] Ishebabi, H., P. Mahr, C. Bobda, M. Gebser, and T. Schaub: *Answer set vs integer linear programming for automatic synthesis of multiprocessor systems from real-time parallel programs*. Journal of Reconfigurable Computing, 2009. Article ID 863630.

[KJS+02] Kumar, S., A. Jantsch, J. P. Soininen, M. Forsell, M. Millberg, J. Oberg, K. Tiensyrja, and A. Hemani: *A Network on Chip Architecture and Design Methodology*. pages 105–112, 2002.

[KPSD11] Kamali, M., L. Petre, K. Sere, and M. Daneshtalab: *Formal modeling of multicast communication in 3d nocs*. In *Proceedings of the 2011 14th Euromicro Conference on Digital System Design*, DSD '11, pages 634–642. IEEE Computer Society, 2011, ISBN 978-0-7695-4494-6.

[KS92] Kautz, H. and B. Selman: *Planning as satisfiability*. In Neumann, B. (editor): *Proceedings of the Tenth European Conference on Artificial Intelligence (ECAI'92)*, pages 359–363. John Wiley & sons, 1992.

[Lif02] Lifschitz, V.: *Answer set programming and plan generation*. Artificial Intelligence, 138(1-2):39–54, 2002.

[LR06] Lifschitz, V. and A. Razborov: *Why are there so many loop formulas?* ACM Transactions on Computational Logic, 7(2):261–268, 2006.

[Luk08] Lukasiewycz et al., Martin: *Efficient symbolic multi-objective design space exploration*. In *Proc. of ASP-DAC '08*, pages 691–696, 2008.

[Luk09] Lukasiewycz et al., Martin: *Combined System Synthesis and Communication Architecture Exploration for MPSoCs*. In *Proc. of DATE '09*, pages 472–477. IEEE Computer Society, 2009.

[Nee01] Neema, Sandeep: *System Level Synthesis of Adaptive Computing Systems*. PhD thesis, Vanderbilt University, Nashville, Tennessee, May 2001.

[NM97] Niemann, Ralf and Peter Marwedel: *An Algorithm for Hardware/Software Partitioning Using Mixed Integer Linear Programming*. Design Automation for Embedded Systems, 2(2):165–193, 1997.

[PF06] Pavlidis, V. and E. Friedman: *3-d topologies for networks-on-chip*. In *SoCC*, pages 285–288. IEEE, 2006.

[RL09] Ramanujam, R. and B. Lin: *A layer-multiplexed 3d on-chip network architecture.* Embedded Systems Letters, 1(2):50–55, 2009.

[YL08] Yan, S. and B. Lin: *Design of application-specific 3d networks-on-chip architectures*. In *ICCD*, pages 142–149. IEEE, 2008.