Modelling and Mapping Framework for Coarse-Grained Programmable Architectures

E. Barbudo, E. Dokladalova and Th. Grandpierre

Laboratoire d'informatique de l'Institut Gaspard Monge
Université Gustave Eiffel
ESIEE Paris

14th Summer School on Modelling and Verification of Parallel Processes

June 26, 2020

Université Gustave Eiffel

ESIEE PARIS

Context

- Machine Vision for Future Construction Sites (DiXite project [1], I-SITE impulsion project).

(a) Autonomous construction machinery [2].

(b) Autonomous unmanned aerial vehicles [3].

Figure 1: Examples of a time-critical application.

- Embedded vision systems
  - Time-critical execution → drastic latency constraints.
  - Multi-processing capabilities → adaptable or programmable computing support.
- Existing solutions: GPPs, GPUs, FPGAs and CGRAs.
- Target hardware family in our context: Coarse-grained Programmable Architectures (CGPA).

GPP= General Purpose Processor, GPU= Graphics Processing Unit, FPGA= Field Programmable Gate Array, CGRA= Coarse-Grained Reconfigurable Architecture

[1] https://dixite.future-isite.fr/ [2] https://www.builtrobotics.com/

[3] https://www.geospatialworld.net/blogs/drones-to-propel-new-technological-innovations-in-the-construction-industry/

Contents

Coarse-Grained Reconfigurable Architectures

- Programmable configuration-driven computing fabric with fixed post-fabrication flexibility.
- There are many variations of CGRAs, depending upon interconnection, type of processing elements or method of reconfiguration.
- The main application of a CGRA is to perform the inner loops of an application.

(a) Generic structure of a CGRA. (b) Generic processing element of a CGRA.

Figure 2: Example of a generic CGRA [1].

[1] M. Hamzeh et al., "EPIMap: Using Epimorphism to map applications on CGRAs". Design Automation Conference 2012, pages pp. 1280–1287.

Coarse-Grained Programmable Architectures

- Heterogeneous linear array of hardware resources, with mixed granularity.
- Multiple configurations, multiple data (MCMD).
- Used as a coprocessor with loose coupling and no shared resources.
- CGRA with fixed hardware resources and a set of parameters to program.

Every hardware resource is fixed

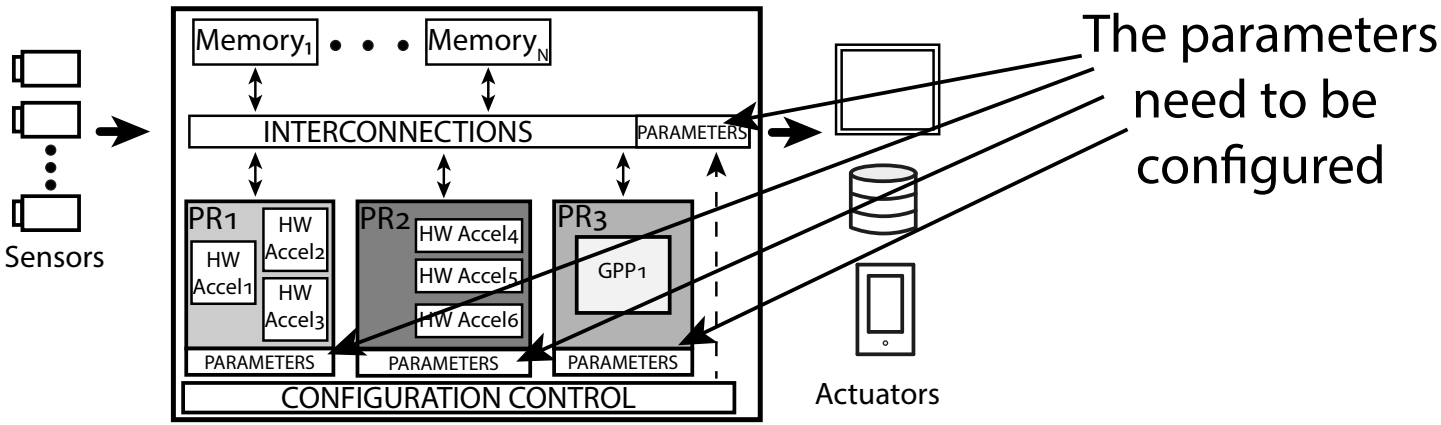
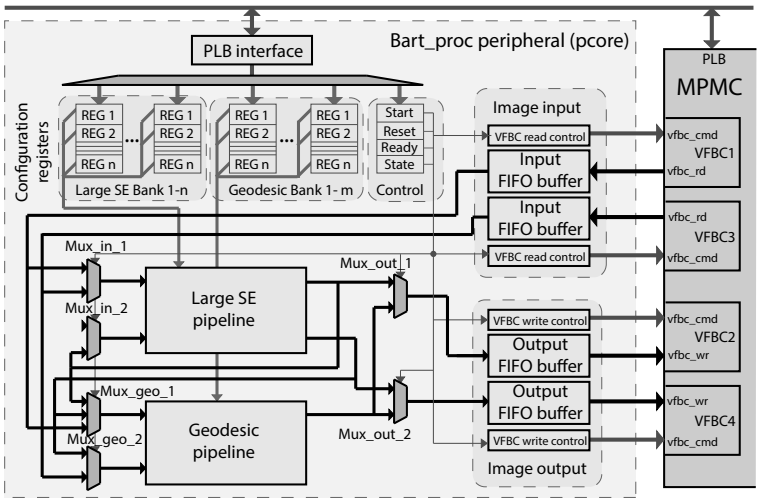


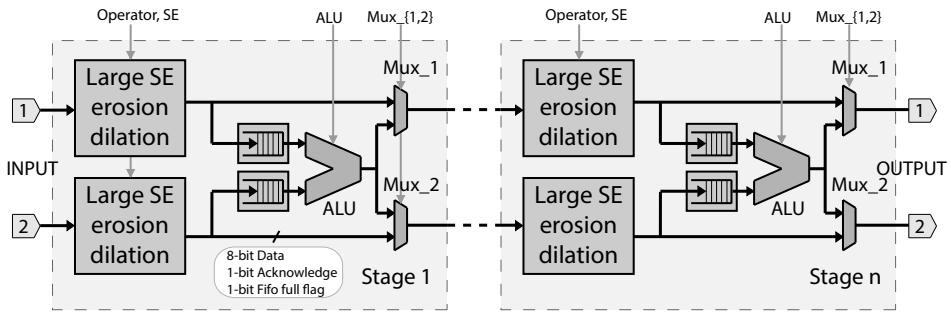
Figure 3: Global architecture of a generic CGPA.

Real life CGPA - motivation hardware

- The morphological co-processing unit (MCPU) is dedicated to image processing.
  - \+ High performance
  - \+ Flexibility
  - – Manual mapping



(a) Architecture of the Morphological co-processing unit.



(b) Large SE pipeline basic stage of the MCPU.

Figure 4: Morphological co-processing unit [2].

[2] J. Bartovský et al., "Morphological co-processing unit for embedded devices". JRTIP, pages pp. 1–12, Jul 2015.

Modelling and mapping framework for CGPA

- Agnostic, to be used in most of the CGPAs on the market.
- Easy reuse of CGPAs.

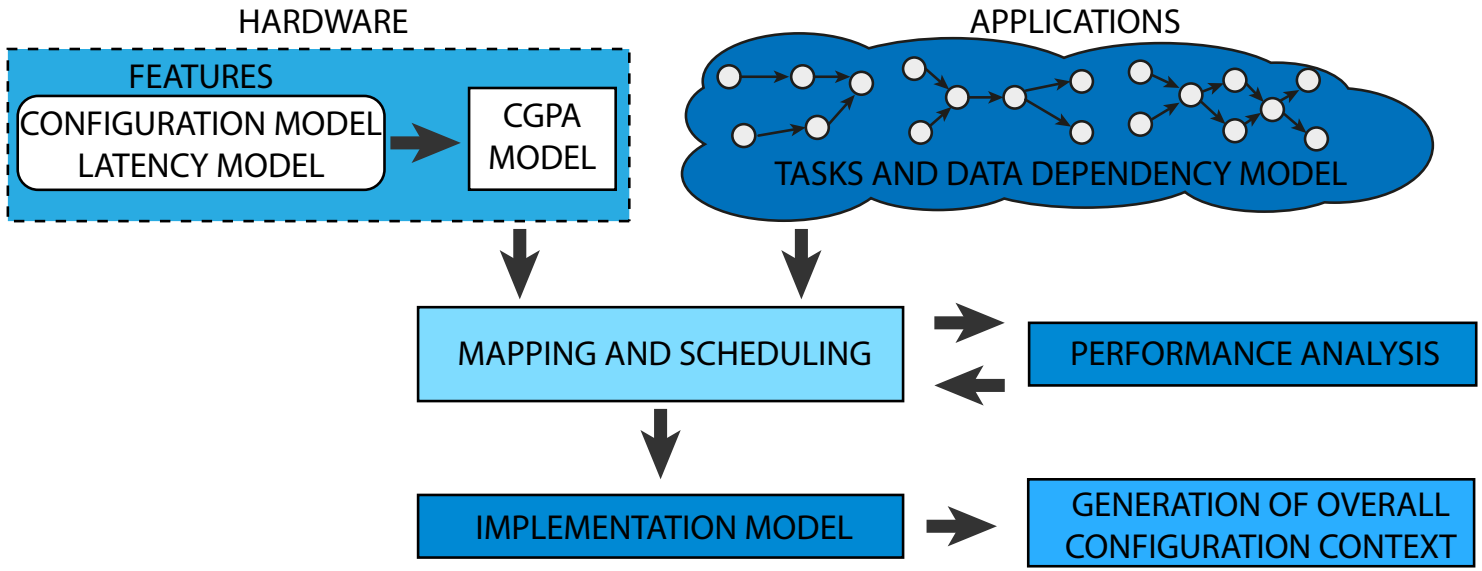


Figure 5: Overall scheme of our modelling and mapping framework for CGPA.

Application model

- $G_{APP}(T, D)$ is a directed hypergraph, with $t_i \in T$ further described as $(type_i, p_i)$.
- $type_i$ represents the transformation of the data and p_i a vector of parameters.

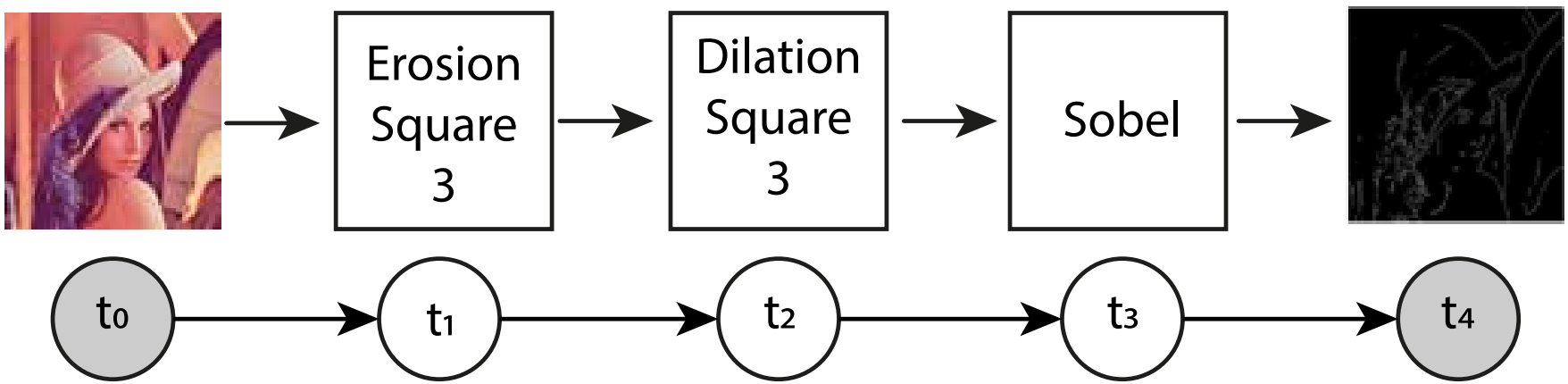


Figure 6: Generic example of an application graph.

Hardware model

- $G_{HW}(S, K)$ is a directed hypergraph, where S represents the hardware resources and K the set of oriented hyperedges.

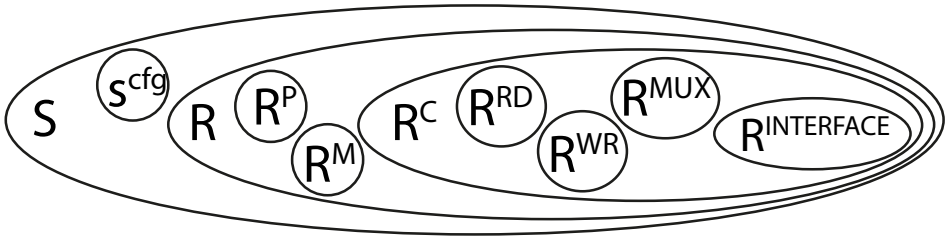


Figure 7: General hierarchy of the set S.

- Each hardware resource is described with its programmable parameters, configuration cost function and latency function.

Hardware model

- We propose to differentiate the **input latency** and **computing latency** of each hardware resource.
- The **input latency** is the number of clock cycles needed to read all the samples required to start to compute the first result.
- The **computing latency** is the number of clock cycles necessary to produce the result once all input samples are available.

Implementation model

- G_{MAP} is obtained by graph transformation between G_{APP} and G_{HW}.
- G_{MAP} is a weighted directed hypergraph.
- Similar structure as G_{HW} with fixed parameters.
- Composed by time slots, which are a subset of resources configured to perform a subset of tasks. Hence, adding a time slot allows re-using the hardware which can be reconfigured between time slots when resources are missing.

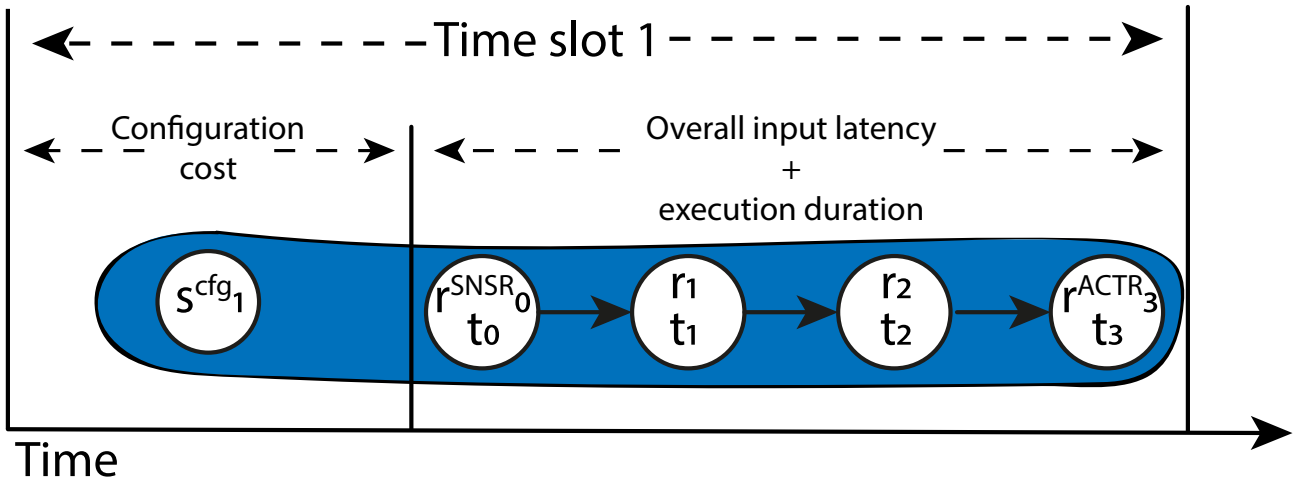


Figure 8: Generic example of a implementation graph.

Mapping algorithms

- Mapping an application onto hardware leads to a very large number of possible implementations.
- The mapping process needs to be automatized.
- There are many mapping algorithms, we propose to use the following two:
  - An exhaustive algorithm, that provides the best result and it can be used as a ground reference.
  - An list-scheduling heuristic based on look-ahead techniques to reduce the exploration time.

Exhaustive algorithm

- The principle is to obtain all the possible mappings.
- We consider the possibility to map one task in one time slot.
- The basic process is:
  - We start with the topological sorting of the application graph.
  - We test the possibility to map a task to any available hardware resource that complies with the requirements.

\+ Optimal mapping.

– Very long exploration time (From hours to days).

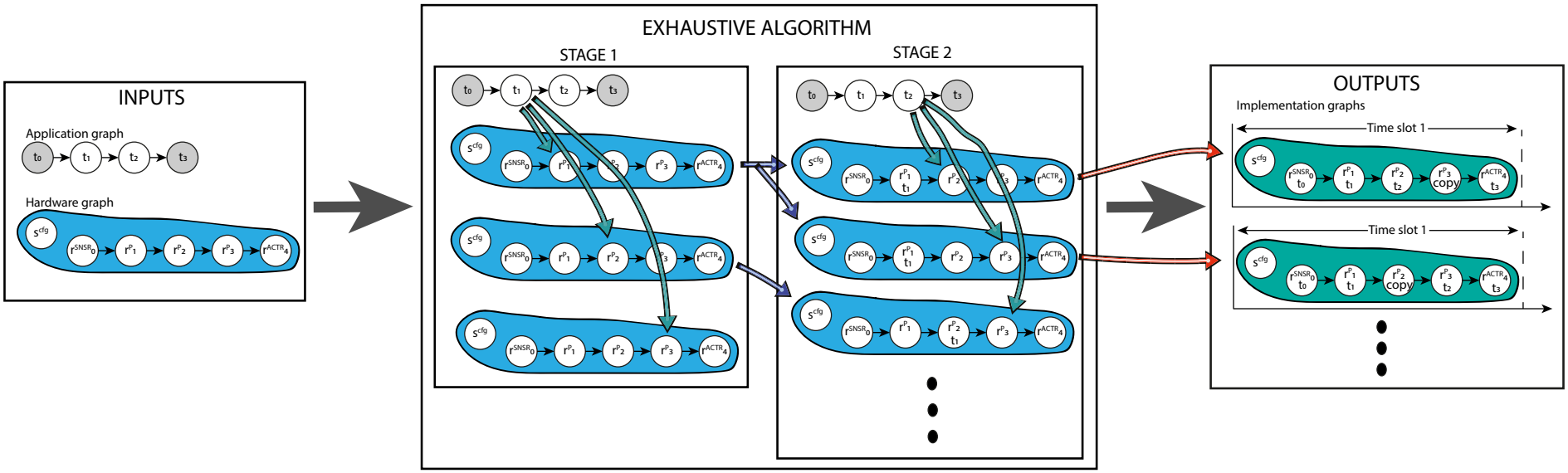


Figure 9: Exhaustive algorithm principle.

List-scheduling heuristic

- Based on look-ahead techniques.
- We evaluate the mapping of the successors of the task onto the descendants of the resource.
- The basic process is:
  - We start with a topological sorting of the application graph and a list with the source nodes of the hardware graph, which are the candidates.
  - We select the first task $t_i \in G_{APP}$ and try to map it to any source node of the hardware graph.
  - We get the successors of t_i and the descendants of the source node $r_j \in G_{HW}$. We compute the chance of allocating successfully the successors of task t_i onto the descendants, taking into account the topological distance.

Probability of mapping success

$$MS_{F_j} = \sum_{b=1}^{x} \frac{d \mid Q_b \mid}{C \sum_{k=1}^{n} \mid F_k \mid} \tag{1}$$

where:

- x is the number of successors of t_i.
- n is the number of possible resource candidates.
- d is the shortest distance to a resource node that complies with the task's successor of interest.
- C is the critical path of the subgraph made by the descendants of all the possible resource candidates.
- Q_b is the set of descendants of r_j that complies with the task's successor of interest.
- F_k is the set of the descendants of r_k.

Performance analysis

- Computing cost (CC) as a metric.
- Computes the input and computing latency of all the elements of the critical path.

$$CC = \sum_{i=1}^{N} (T_{C_i} + \overbrace{(CL_i)(TS)}^{\mathbf{Tex}_i} + \overbrace{\sum_{j=1}^{jCP_ij \ \ 1} ((\mathcal{L}_j^{IN} - 1)(\alpha_j) + \mathcal{L}_j^{CL} + 1))}^{\mathbf{Tin}_i} \quad (2)$$

where

- N is the number of time slots.
- T_{C_i} configuration cost of time slot i.
- T_{IN_i} refers to the overall input latency of the resources of time slot i.
- T_{EX_i} is the execution duration of time slot i.
- CL_i is the worst computing latency of the critical path of time slot i.
- TS is the total of input samples.

Performance analysis

$$T_{\mathrm{IN}_i} = \sum_{j=1}^{jCP_ij \;\; 1} ((\mathcal{L}_j^{IN} - 1)(\alpha_j) + \mathcal{L}_j^{CL} + 1) \tag{3}$$

where

- CP_i is a set of resources that belong to the critical path of time slot i.
- $\mathcal{L}_j^{IN}$ is the input latency of the resource.
- $\mathcal{L}_j^{CL}$ is the computing latency of the resource.
- α_j is an expression of the propagation of computing latency. Let $\alpha_j = max(\alpha_{j \;\; 1}, \mathcal{L}_j^{CL}{}_1)$, where $\alpha_{j \;\; 1}$ is the α of the predecessor and $\mathcal{L}_j^{CL}{}_1$ is the computing latency of the predecessor.

Evaluation of the methodology

- We consider the morphological co-processing unit (MCPU) [2] as a candidate for the use of our framework and two applications.

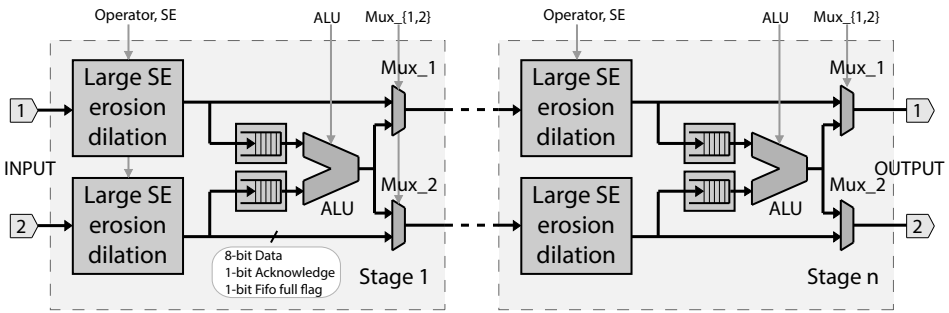


Figure 10: Large SE pipeline basic stage of the MCPU.

[2] J. Bartovský et al., "Morphological co-processing unit for embedded devices". JRTIP, pages pp. 1–12, Jul 2015.

Hardware graph

- For the purpose of the evaluation, we only use the Large SE pipeline double stage (Fig. 10).

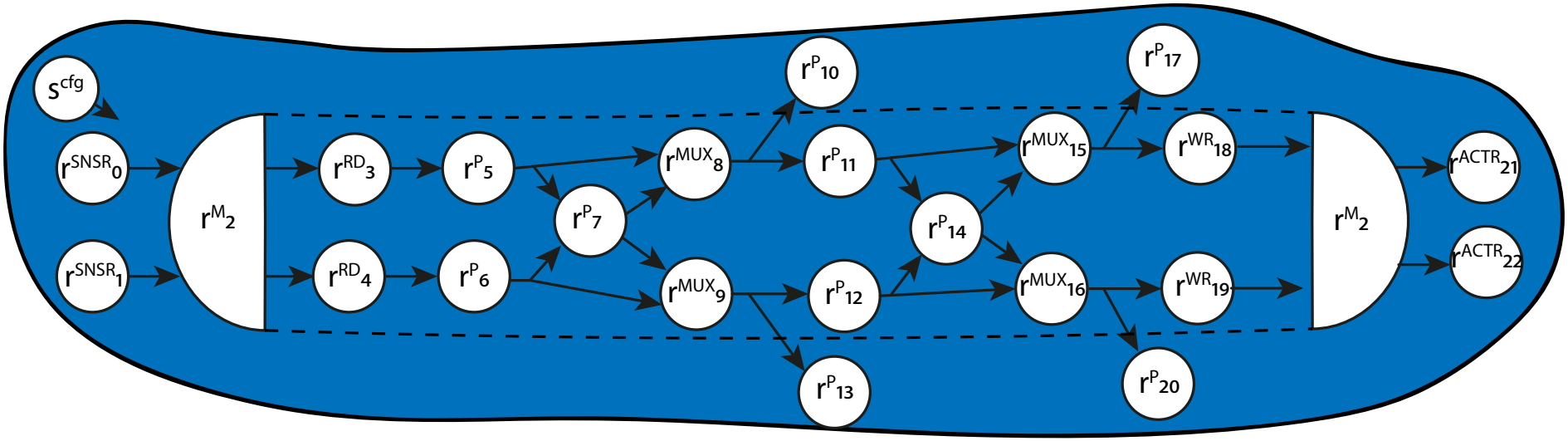
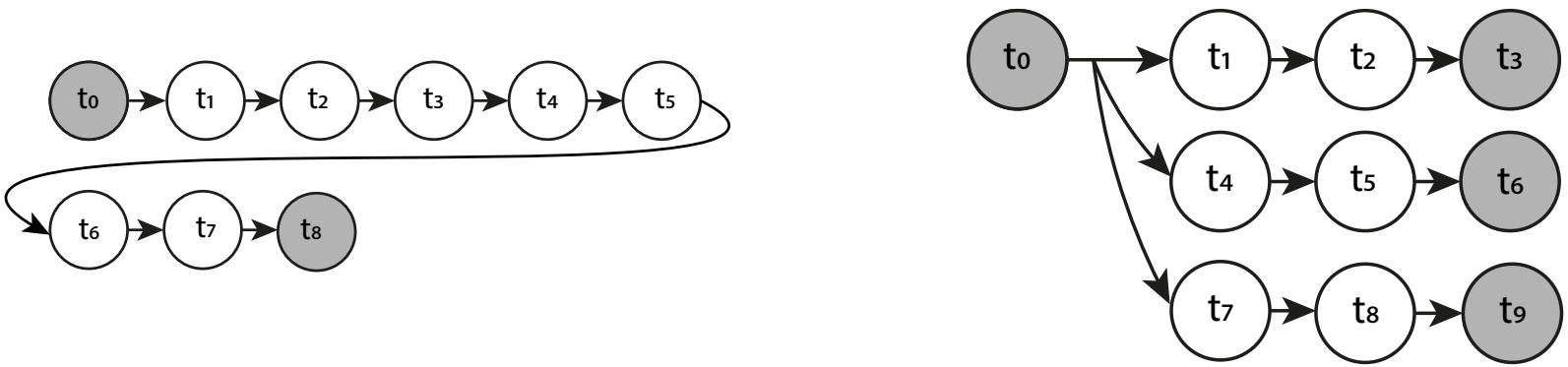


Figure 11: G_{HW} of the morphological co-processor unit.

Application graph

- The first application example is a long linear pipeline of tasks. This application exceeds the available resources (Fig. 12a).
- The second application example represents a highly parallel task organization (Fig. 12b).



(a) Application example 1.

(b) Application example 2.

Figure 12: Application examples.

Evaluation

- We evaluate our algorithms on two metrics computing cost and exploration time. Table 1 summarize the results. The values of computing cost are in clock cycles.

Table 1: Algorithms evaluation

	Number of nodes	Algorithm	Computing cost	Exploration time	Improvement percent
Application example 1 (Linear set of tasks)	9	Exhaustive	62470	70 minutes	N/A
		Heuristic	62470	0.53 seconds	99 %
Application example 2 (Parallel set of tasks)	10	Exhaustive	61628	19 minutes	N/A
		Heuristic	62628	0.9 seconds	99 %

Resulting mappings

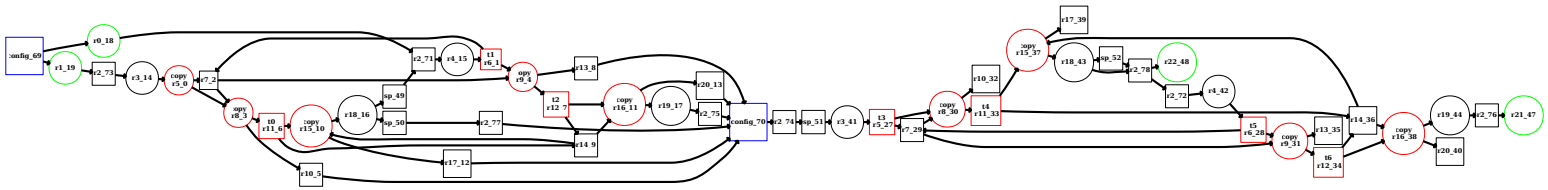


Figure 13: Exhaustive final mapping for application example 1.

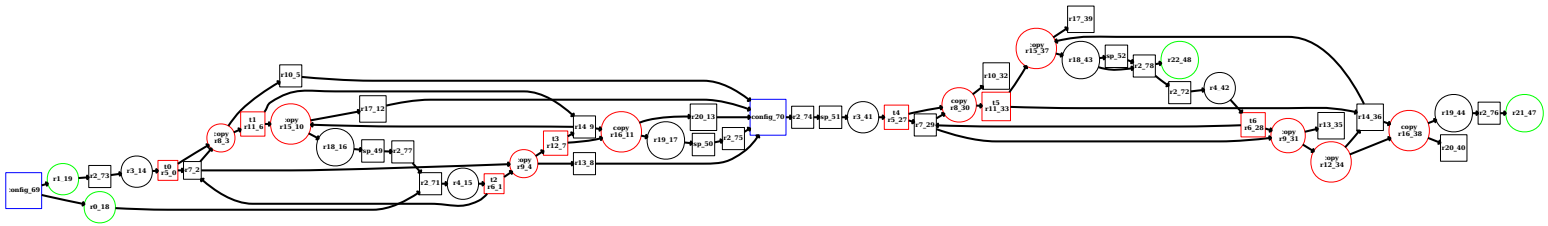


Figure 14: Heuristic final mapping for application example 1.

Resulting mappings

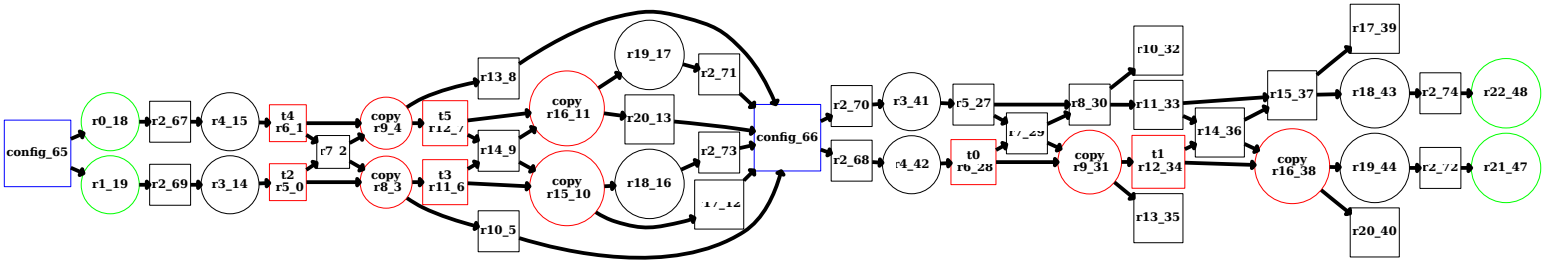


Figure 15: Exhaustive final mapping for application example 2.

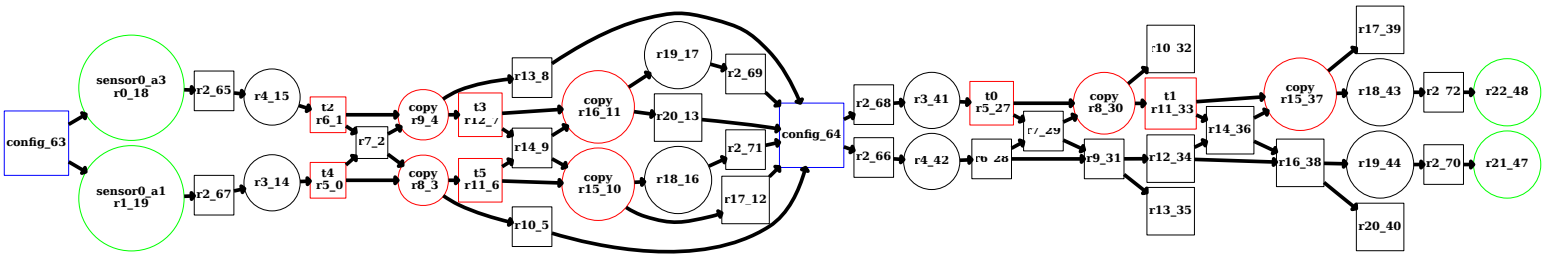


Figure 16: Heuristic final mapping for application example 2.

Conclusions

- We have introduced our proposed modelling and mapping framework.
- Our models provide the means to abstract the configuration cost and the heterogeneous latency of a CGPA.
- We presented two mapping approaches. The exhaustive approach is able to produce an optimal mapping in terms of latency but at the cost of long exploration time. The look-ahead-based heuristic approach provides a mapping in less exploration time.
- Additionally, we define a method for performance analysis based on the computing cost over the critical path.

Future work

- Analyze the complexity of the mapping algorithms.
- Inclusion of the notion of latency in the heuristic equation.
- Evaluate extensively the framework with real hardware examples.

Thank you for your attention

Questions

elias.barbudo@esiee.fr