Towards Adaptive multi-Alternative Process Network

Hasna Bouraoui
Technische Universität Dresden, Germany

Chadlia Jerad
University of Manouba & University of Carthage, Tunisia

Jeronimo Castrillon
Technische Universität Dresden, Germany

Abstract

With the increase of voice-controlled systems, speech based recognition applications are gaining more attention. Such applications need to adapt to hardware platforms to offer the required performance. Given the streaming nature of these applications, dataflow models are a common choice for model-based design and execution on parallel embedded platforms. However, most of today's models are built on top of classical static dataflow with adaptivity extensions to express data parallelism. In this paper, we define and describe an approach for algorithmic adaptivity to express richer sets of variants and trade-offs. For this, we introduce multi-Alternative Process Network (mAPN), a high-level abstract representation where several process networks of the same application coexist. We describe an algorithm for automatic generation of all possible alternatives. The mAPN is enriched with meta-data serving to endow the alternatives with annotations in terms of a specific metric, helping to extract the most suitable alternative depending on the available computational resources and application/user constraints. We motivate the approach by the automatic subtitling application (ASA) as use case and run the experiments on an mAPN sample consisting of 12 randomly selected possible variants.

2012 ACM Subject Classification Theory of computation → Streaming models

Keywords and phrases High level process network, algorithmic adaptivity, automatic subtitling application

Digital Object Identifier 10.4230/OASIcs.PARMA-DITAM.2021.4

Funding *Hasna Bouraoui*: []
Chadlia Jerad: []
Jeronimo Castrillon: []

1 Introduction

With the proliferation of digitalization and the easy access to storage capacity, large volumes of audio data including broadcasts and meetings are increasingly generated and stored. As a result, a growing need for automated processing of human language has emerged, leading to the advent of many audio processing applications. One example is Audio Indexing, enabling the search and retrieval of *who spoke what* from an audio source. Another example is the Automatic Subtitling Application (ASA)[1, 11], where Speaker Recognition (SpkR), Speech Recognition (SpR), and Speaker Diarization (SD) are all combined. For the design of such complex applications, embedded programmers need to understand algorithmic variants implementing the same functionality (i.e. algorithmic adaptivity) and how can they be deployed in parallel into possibly different many-core platforms (i.e. parallelism adaptivity). In any given situation, characterized by available hardware resources and application/user constraints, it is difficult to manually select the adequate algorithm while delivering the required performance.

12th Workshop on Parallel Programming and Run-Time Management Techniques for Many-core Architectures and 10th Workshop on Design Tools and Architectures for Multicore Embedded Computing Platforms (PARMA-DITAM 2021).
Editors: João Bispo, Stefano Cherubin, and José Flich; Article No. 4; pp. 4:1–4:11

OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

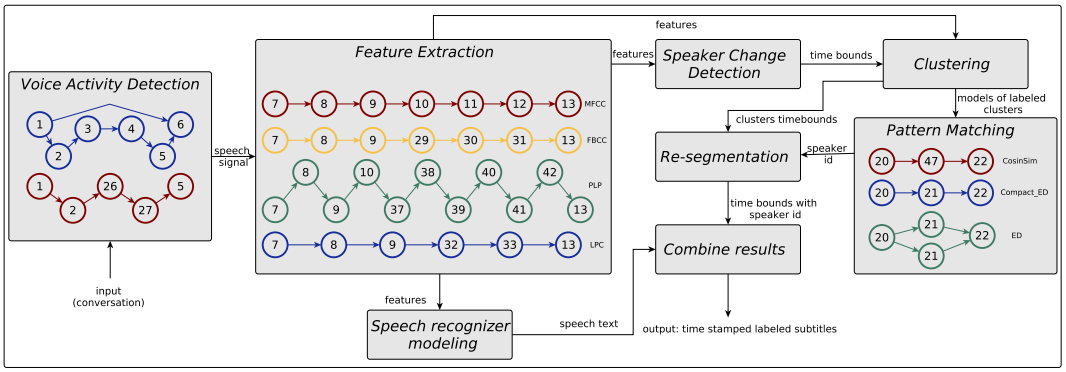

Figure 1 Representation of the Automatic Subtitling Application

These streaming applications can exploit the parallelism of embedded many-core platforms especially when described using appropriate Models of Computation (MoC [18]). For example, Synchronous Dataflow (SDF) work well for describing one particular algorithmic variant. However, it is ill-suited to express the adaptivity required today. For static models (e.g., Cyclo-Static Dataflow (CSDF) [4] and Parameterized Synchronous Dataflow (PSDF) [3]) adaptivity is at the token production and consumption levels. However, Dynamic models allow for topology updates. A prominent example is Scenario-Aware Dataflow (SADF) [21] which expresses behavioral adaptivity by modifying the application graph or varying the amount of parallelism. However, this model suffers from limitations with regard to the size and complexity of the model itself [5, 12]. Otherwise, one option would be to have a different graph for every single variant of the ASA algorithms to be arbitrated at run time. Obviously, such a solution may not be practical and would lead to a combinatorial explosion on top of being prohibitively complex to manage. In this paper, we propose a novel model for compact representation and exploration of multiple algorithmic variants in one single-source specification. Our model, called mAPN, extends the well-known Kahn Process Network (KPN) [15] model. mAPN carries annotations on metrics and enables automatic exploration of the different implementations to choose a well-suited variant. The main contributions of this paper are:

- We introduce mAPN to capture multiple algorithmic variants, beyond what existing models allow, in a compact single-source specification (cf. Sec. 3.1).
- We present an algorithm for the automatic exploration of several variants based on mAPN annotations and the constraints introduced by the designer (cf. Sec. 3.2).
- We demonstrate the analysis fidelity of our approach for the ASA use case. cf. Sec. 4

2 Motivational example

In this paper we use ASA as a case study to demonstrate the need for parallelism and algorithmic adaptivity. ASA is a complex application that combines the functionality of SpkR, SD and SpR, to recognize who is speaking, when s/he is speaking, and what s/he is saying, respectively. The different functionalities share common phases (e.g., Feature Extraction (FE)), as well as common algorithms serving different phases (i.e. Classification, Pattern Matching (PM), and Speaker Change Detection (SCD)). Fig. 1 illustrates a coarse-grained representation of ASA. To better grasp the scale at which the number of possible implementations can grow, we will detail the VAD, FE, and PM phases. For FE, speaker characteristics can be categorized based on different features. Prominent algorithms for feature extraction are MFCC, FBCC, PLP, and LPC [24, 23, 17]. These possible implementations

are schematically shown in Fig. 1. For each particular phase, processes with the same number represent common nodes of these algorithms. The PM phase exhibits three possible implementations: Euclidean Distance (ED) and Cosine Similarity (CS) (i.e. compact, or expanded if we consider parallelism adaptivity). In absolute terms, no variant is better than all the others. Which variant is best depends on the application/user constraints and the desired target hardware.

Depending on these constraints, one has to specialize the phases for each functionality. Phase reuse and algorithmic choices create a large space of possible variants for ASA. Some researchers use classical techniques that are accurate and can run on embedded hardware with resource constraints. Other approaches are based on deep learning, requiring large databases. This gain in complexity makes it hard from a developer point of view to choose the right implementation of the application beforehand. This becomes even harder considering the diversity of possible target hardware, as the achieved performance of an implementation may significantly differ from one hardware to another. In any given situation, it is difficult to manually select an adequate algorithm under application/user/HW constraints. To ease development, applications such as ASA are not written from scratch, but are built from existing implementations. This is known as the Algorithm Selection Problem [19], which shifts the burden from finding the right solution to identifying and composing appropriate existing algorithms. In this paper we propose a novel model to support designers in this process, named mAPN.

3 mAPN

3.1 mAPN Formalization

A KPN is a directed graph composed of a set of concurrent processes (nodes), communicating through unidirectional unbounded First In First Out (FIFO) channels (edges) having blocking reads and non blocking writes semantics. Formally, a KPN is a tuple $G = (P, Ch)$, where P is a set of processes, and $Ch \subseteq P \times P$ a set of channels. A Multi-Alternative Process Network (mAPN) is a graph that concisely represents many different KPNs. We use colors to tag and then generate all possible alternatives. Let $\Xi = \{\xi^1, \xi^2, ..., \xi^n\}$ be the set of colors. $\xi^* \in \Xi$ denotes a particular *neutral* color and $\mathcal{P}(\Xi)$ the power set of Ξ. Similar to KPNs, an mAPN is a directed graph composed of processes and channels. However, and unlike KPNs, channels are annotated with colors indicating the local alternatives that the channel belongs to. Formally,

▶ **Definition 1** (mAPN). *A multi-Alternative Process Network is a tuple $G = (P, Ch, col)$, where P is a set of processes, Ch is a set of channels $Ch \subseteq P \times P$ and col is a function that maps each channel to a subset of colors, that is $col : Ch \rightarrow \mathcal{P}(\Xi)$.*

Let $wr, rd : Ch \rightarrow P$ be two functions that map each channel to the process that writes into it, respectively reads from it. In a similar way, we define $\widehat{wr}, \widehat{rd} : P \rightarrow \mathcal{P}(Ch)$ that map each process into a subset of Ch it writes to, respectively, it reads from (i.e., $\widehat{wr}(p) = \{ch \in Ch, wr(ch) = p\}$). We call *source processes* (Src) the subset of P that do not read from any channel. Analogously, *sink processes* (Snk) do not write to any channel. That is, $Src = \{p \in P, \widehat{rd}(p) = \emptyset\}$, and $Snk = \{p \in P, \widehat{wr}(p) = \emptyset\}$. The function $\widehat{col}_{rd}$ returns the colors a process reads from. Analogously, $\widehat{col}_{wr}$ returns the colors a process writes to (i.e., $\widehat{col}_{wr}(p) = \cup_{ch \in \widehat{wr}(p)} col(ch)$). In the sample mAPN of Fig. 2, the colors of the channel that connects process v to x (ch_v^x), has colors {■, ■}, while $col(ch_b^p) = \{$■$\}$. Note also that $Src = \{a, v\}$, $Snk = \{u\}$ and $\widehat{col}_{wr}(p) = \{$■, ■, ■$\}$.

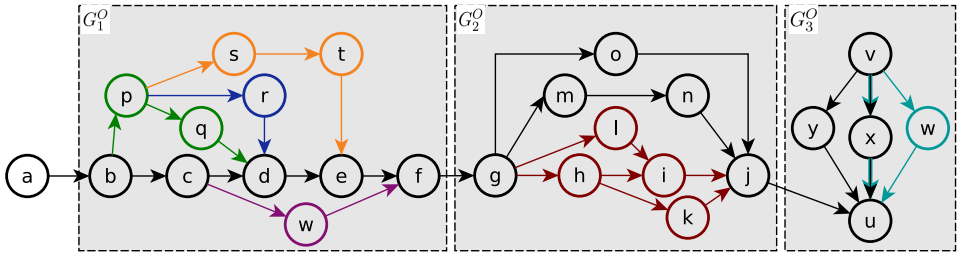

Figure 2 mAPN of a synthetic example

A colored subgraph $G^{\xi} = (P^{\xi}, Ch^{\xi})$ of an mAPN $G = (P, Ch)$ gathers all the channels having the same color ξ, and the processes connected to them. The neutral color ξ^* (black color in Fig. 2) marks the *nominal end-to-end* implementation of the application (cf. Definition 2). Colored subgraphs represent local alternatives. For example, the purple subgraph connecting processes c and f is a local alternative to the black subgraph, replacing the functionality of d and e. For clarity reasons, nodes in the figure receive the colors of the local alternative they belong to. Local alternatives can be nested, as is the case of the blue and green ones. Consequently, the flow from b to d can go through c (black), through p and q (green), or through p and r (composition of green and blue).

▶ **Definition 2** (Nominal alternative). *A nominal alternative* $G^{\xi^*} = (P^{\xi^*}, Ch^{\xi^*})$ *of an mAPN* $G = (P, Ch, col)$ *is an end-to-end subgraph of* G *where* $\forall ch \in Ch_n, \xi^* \in col(Ch_n)$, *and* $Src, Snk \subset P^{\xi^*}$.

We distinguish processes from/at which local alternatives fork/join. These processes are important since they identify anchor nodes for generating alternatives. We classify them into the following subsets:

- $\mathcal{F}$ is the subset of processes of P that write different channels with different colors. Formally, $\mathcal{F} = \{p \in P, \exists ch_i, ch_j \in \widehat{wr}(p), i \neq j, col(ch_i) \neq col(ch_j)\}$.
- $\mathcal{J}$ is the subset of processes of P that read different channels with different colors. Formally, $\mathcal{J} = \{p \in P, \exists ch_i, ch_j \in \widehat{rd}(p), i \neq j, col(ch_i) \neq col(ch_j)\}$.

In the mAPN of Fig. 2, $\mathcal{F} = \{b, c, p, v, g\}$ and $\mathcal{J} = \{d, e, f, j, u\}$. Based on the nominal alternative and the collection of local alternatives forming an mAPN, one can generate all possible alternatives. Generation is based on a set of assumptions that the mAPN is a well-formed. Concretely:

▶ **Definition 3** (Well-formed mAPN). *A well-formed mAPN* $G = (P, Ch, col)$ *has the following properties: (i) existence of a nominal alternative, (ii) every sink node of a colored subgraph is a join node, (iii) every source node of a colored subgraph is a fork node, (iv) colors cannot be re-used in disjoint subgraphs, and (v) preserving KPN semantics (that is, for a fork node, the number of write channels per color must be the same).*

KPN semantics preservation does not include a condition over the number of read channels per color for a join node. Process u in Fig. 2 is a counter example, as we have two branches of alternatives coming from two distinct source nodes.

We structure an mAPN so that alternatives are created from products over the sets of generated sub-KPNs. These subsets are generated from mAPN subgraphs, and we call them *closed*. Nesting alternatives occur only within such subgraphs. Three subgraphs are marked by dashed gray boxes in Fig. 2, each one itself, a well-formed mAPN. By adding anchor nodes to every element of the product of sub-KPNs, we get the set of all possible alternatives. We define a closed mAPN as follows:

Algorithm 1 Generation of Kahn Process Networks

param: $G : (P, Ch, col), M, V, C$
return: $KPNs : \{kpn_i\}_i$

```
1:  procedure AltGen
2:      KPNs, KPN'_s ← ∅
3:      {G_i^O}_i ← getClosedGraphs(G)
4:      (P_com, Ch_com) ← rmCommon(G, {G_i^O}_i)
5:      for every G_i^O do
6:          KPN'_s ← AltGenRecur(G_i^O)
7:          KPNs ← mixNmatch(KPNs, KPN'_s)
8:      for kpn_i = (P_i, Ch_i) ∈ KPNs do
9:          kpn_i ← (P_i ∪ P_com, Ch_i ∪ Ch_com)
10:         v = ν(kpn_i, M, V)                    ▷ Metrics evaluation of kpn_i
11:         if v ⊭ C then                         ▷ Checking constraints
12:             KPNs ← KPNs \ {kpn_i}
13:     return KPNs
```

▶ **Definition 4** (Closed mAPN). *A closed mAPN $G^O = (P^O, Ch^O, col^O)$ of a well formed mAPN $G = (P, Ch, col)$ has the following properties: (i) G^O is a well formed mAPN, $Src^O \subset \mathcal{F}$ and $Snk^O \subset \mathcal{J}$, (ii) no loop backs are allowed across closed mAPNs, (iii) a closed mAPN is minimal, that is it cannot include a composition of two or more closed sub-mAPNs, and (iv) the set of colors that fork within a closed subgraph is the same that joins.*

3.2 Exploration algorithm

Algorithm 1 describes the generation process by: (i) extracting closed graphs (line 3), (ii) generating the set of KPNs for each one and mixing and matching them (lines 5–7), (iii) adding anchor nodes for each partially constructed KPN (line 9). After adding adding anchor nodes, the KPN is complete and can then be evaluated (line 10) using the rules of aggregation, and checked against the given constraints (line 11). If the constraints are not satisfied, that particular KPN is removed from the set of alternatives to return (line 12).

The recursive procedure `AltGenRecur()` (Algorithm 2) visits source nodes (line 4) and generates a set of KPNs for each one. Given a source process p, the algorithm iterates over the local alternatives, i.e., colors of the channels p writes to (`for` in line 6). For every colored subgraph (kpn_ξ) having p as source (line 7), if kpn_ξ reaches sink processes without including any fork, then it is a complete generated local KPN (lines 8, 9), and is added to the set of generated KPNs starting from p (line 19). This is the example of the purple colored subgraph starting from the fork source c of G_1^O. Otherwise, a recursive call over the computed subgraph G' (line 17) will return alternatives, to be mixed and matched (line 18). Computing G' depends on whether there are nested alternatives, or a sink of G is not reached. In the first case, G' is the subgraph of G that starts from the immediately encountered fork nodes of the colored subgraph kpn_ξ (lines 11-14). For example, if we consider the green colored subgraph, G' will be the entire subgraph of G_1^O that has process p as source. In the second case, i.e. no sink is reached, G' is the subgraph that starts from sink processes of kpn_ξ (lines 16). This is the case of the three colored subgraphs of G_1^O starting from process p. `mixNmatch()` takes two sets of KPNs and computes all possible combinations of their elements (i.e., set product).

Algorithm 2 Recursive generation of Kahn process networks

param: $G \ : \ (P, Ch, col)$
return: $KPNs : \{kpn_i\}_i$

procedure AltGenRecur
  $KPNs, KPN_s^p \leftarrow \emptyset$
  $Src \leftarrow \{p \in P, \widehat{rd}(p) = \emptyset\}$
  for $p \in Src$ **do**
    $KPN'_s \leftarrow \emptyset$
    for $\xi \in \widehat{col}_{wr}(p)$ **do** ▷ Iterate over colors
      $kpn_\xi = (P_\xi, Ch_\xi) \leftarrow subgraph(G, \{p\}, \xi)$
      if $(P_\xi \cap F = \{p\}) \wedge (Snk_{/kpn_\xi} \subseteq Snk)$ **then**
        $KPN'_s \leftarrow \{kpn_\xi\}$
      else
        if $(P_\xi \cap F \neq \{p\})$ **then** ▷ Nested alt.
          $\{p_j\}_j \leftarrow getClosest(p, P_\xi \setminus \{p\} \cap F)$
          $G' \leftarrow subgraph(G, \{p_j\}_j)$
          $kpn_\xi \leftarrow kpn_\xi \setminus \{kpn_\xi \cap G'\}$
        else ▷ No sink is reached
          $G' \leftarrow subgraph(G, Snk_{/kpn_\xi})$
        $KPN'_s \leftarrow AltGenRecur(G')$
        $KPN'_s \leftarrow mixNmatch(\{kpn_\xi\}, KPN'_s)$
      $KPN_s^p \leftarrow KPN_s^p \cup KPN'_s$
    $KPNs \leftarrow mixNmatch(KPNs, KPN_s^p)$
  return $KPNs$

4 Evaluation

4.1 The mAPN model of ASA

We demonstrate our approach on an ASA application. Recall the coarse-grained graph presented in Fig. 1. The figure shows an example of existing implementations and commonalities across them, characterizing the large design space of algorithmic variants for this application. Several approaches and algorithms for each phase can be found in the literature. Readers may refer to [6] for a detailed survey. In this section we explain the most common algorithms for SpkR, SR and SpR, and how they can be combined in one graph to create a different ASA implementation. Each phase in Fig. 1 is replaced by one or more possible implementations using different colors. Fig. 3 represents one possible compact graph for the ASA application. The voice activity detection (VAD) has two different variants while FE has 7. By mixing and matching these two phases, we generate 14 possible variants implementing the part ending at node 13. So far, we have excluded adaptive parallelism in the variants. To exploit Task-level parallelism (TLP), expanded/compacted versions of a Process Node (PN) can be added as an additional algorithmic variants. This is similar to the work presented in [20]. The algorithm that implements the MFCC FE can be executed by one PN (i.e. Node 28), or expanded over several PNs (5 PNs:9-13). The same process can be applied to all the local variants of the FE phase leading to 7 additional algorithmic variants just for this phase. Similarly, Data-level parallel (DLP) versions of some phases can be deployed to balance the load across application phases, as seen in [16]. For example, in the pattern matching phase (node 15), a new alternative can be created where multiple *Euclidean Distance (ED)* nodes run in parallel and then send the results to the *merge* node. Every added possibility implementing a local phase in the compact graph greatly enriches the space of implementation variants. After only adding the discussed TLP/DLP alternatives, the number of possible variants reaches 672.

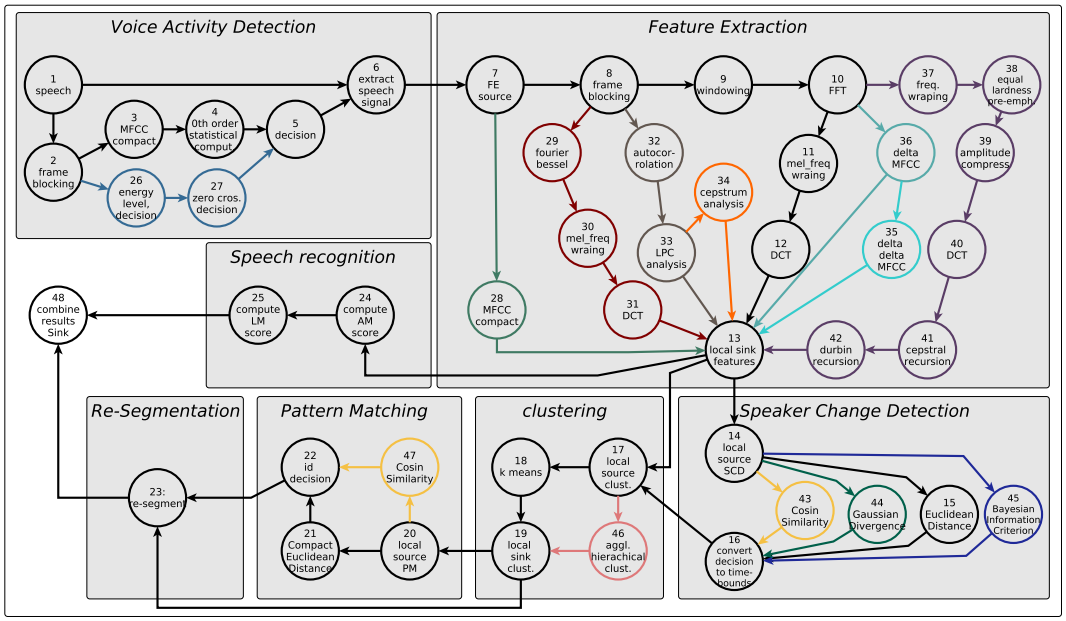

Figure 3 Multi-Alternative Process Network for ASA

4.2 Experimental Results

For the experimentation, we implemented for the VAD the black alternatives corresponding to an MFCC-based VAD as presented in Fig. 3. In phase FE, we explore parallelism adaptivity by applying TLP to the implementation of the MFCC algorithm (compact version, 28, vs. expanded version, 8 to 13), and algorithmic adaptivity by adding an FBCC implementation (9-29-30-31). SCD phase presents algorithmic adaptivity: ED (node 15) vs. CS (43); while the k-means algorithm (18) is used for clustering. Finally, DLP is applied to the PM phase by varying the number of ED-Compact nodes (21) to be either 1 or 4. We implemented these algorithms and end up with 12 possible alternatives in total (combinations of 32 different process ids).

We explore the space of alternatives to select feasible ones and compare it to a brute force approach, where all implementations are generated and executed. As an evaluation metric, we use the execution time as a metric under a constraint of *55s*. To factor the errors introduced by estimation, we use actual measurements using the estimator of the SLX tool suite[1] on the target hardware. Based on these estimations, evaluation of alternatives are performed using the max-plus algebra traditionally used for timing analysis of dataflow graphs [13]. Experiments are made on a speech of 5 minutes of length, and considering two platforms: Odroid XU4[2] and a GPP[3] (GPP).

As shown in Table 1, column *Estimations* contains the execution time estimates on both platforms. Each assessed alternative is then considered feasible (✓), or rejected (✗) depending on whether it respects the defined constraint or not (not exceeding 55s). Under column *Real Exp.Results*, we present the experimentally measured wall-clock time for these alternatives. The column *R/F* (right/false decision) lists the correctness of the decision made about the feasibility of each alternative. By using a time constraint of 55s, Table 1

[1] https://www.silexica.com/
[2] Exynos 5422 big.LITTLE chip: 4 Cortex-A15 and 4 Cortex-A7 cores
[3] 3.40GHz quad-core Intel(R) Core(TM) i7-6700 CPU

	Estimations				Real Exp. Results			
	Odroid		GPP		Odroid	R/F	GPP	R/F
Alt 1: {VAD-FE(Bessel)-SCD(ED)-Cl-PM(DLP-4)}	102.36	✗	26.24	✓	56.43	R	20.66	R
Alt 2: {VAD-FE(Exp.MFCC)-SCD(ED)-Cl-PM(DLP-4)}	52.11	✓	11.53	✓	24.43	R	9.94	R
Alt 3: {VAD-FE(Comp.MFCC)-SCD(ED)-Cl-PM(DLP-4)}	51.80	✓	10.56	✓	23.66	R	9.66	R
Alt 4: {VAD-FE(Bessel)-SCD(ED)-Cl-PM(DLP-1)}	115.20	✗	30.74	✓	67.69	R	25.73	R
Alt 5: {VAD-FE(Exp.MFCC)-SCD(ED)-Cl-PM(DLP-1)}	64.95	✗	16.03	✓	53.78	F	20.54	R
Alt 6: {VAD-FE(Comp.MFCC)-SCD(ED)-Cl-PM(DLP-1)}	64.64	✗	15.06	✓	52.78	F	19.51	R
Alt 7: {VAD-FE(Bessel)-SCD(CS)-Cl-PM(DLP-4)}	102.35	✗	26.23	✓	45.72	F	20.93	R
Alt 8: {VAD-FE(Exp.MFCC)-SCD(CS)-Cl-PM(DLP-4)}	52.10	✓	11.52	✓	23.10	R	10.05	R
Alt 9: {VAD-FE(Comp.MFCC)-SCD(CS)-Cl-PM(DLP-4)}	51.79	✓	10.55	✓	23.68	R	9.32	R
Alt 10: {VAD-FE(Bessel)-SCD(CS)-Cl-PM(DLP-1)}	115.19	✗	30.73	✓	56.88	R	20.42	R
Alt 11: {VAD-FE(Exp.MFCC)-SCD(CS)-Cl-PM(DLP-1)}	64.94	✗	16.02	✓	53.74	F	19.94	R
Alt 12: {VAD-FE(Comp.MFCC)-SCD(CS)-Cl-PM(DLP-1)}	64.63	✗	15.05	✓	55.47	R	20.01	R

Table 1 Experimental results

shows that estimations reaches a success rate of 66.66% and 100% on Odroid and GPP respectively. Besides being able to decide on which alternative to use, we are also interested on how fast such a decision can be made. For the *brute force* approach, and considering the Odroid platform, choosing among the 12 alternatives requires an execution time of 537.37s. By extrapolating to 672 alternatives, this would correspond to around 8h. Conversely, the mAPN approach only requires to assess 32 different nodes to cover the 12 alternatives which takes 206.2s. This extrapolates to 16m considering the 672 alternatives (covered by 49 nodes as in Fig. 3). We notice that the estimations are generally slower than the real results, which can be explained by the fact that we are not taking into consideration the pipelining parallelism hidden in the KPN dataflow graph. In addition, using better assessment leads to more accurate estimation. However, this is kept out of the scope of this paper.

Further analysis is done to show the fidelity of the estimation. Fig. 4 illustrates the correlation plot between the estimated cost of an alternative and the actual execution time of the real implementation. The grey area in the plot represents the 95% confidence intervals of the results following the smoothing method. We also sort the results of the 12 alternatives and compute the rank correlation to measure the ordinal association between them. This results in 0.936 and 0.802, using the *Spearman's* ρ and *Kendall's* τ methods respectively. Recall that 1 represents a perfect alignment between the two rankings. For both methods, very high agreement levels are achieved, proving that our mAPN, ensure an acceptable ordering of the alternatives in terms of the considered metric. Even with the noticed deviation between the estimations and the measures, we are still able to say which alternative is better than the other without a time consuming evaluation of all alternatives in the target hardware. This helps the user to decide on the feasible and adequate ones in a large space of variants.

5 Related work

Many research works propose methods for parallelism adaptivity. Flextream [14] studied adaptable compilation of a running application to the changing hardware characteristics. Adaptivity is achieved by replicating stateless processes to form a more fine-grained graph. Authors in [22] went further and proposed actor merging for sequentially executing processes. Likewise, Lee et al [7] proposed an optimal method to adapt the running application to available resources. Optimality refers to memory footprint under core-count constraints to reduce energy consumption. [10] extended the semantics of SDFs to increase expressiveness and enable the specification of dynamic reconfigurable signal processing applications. They

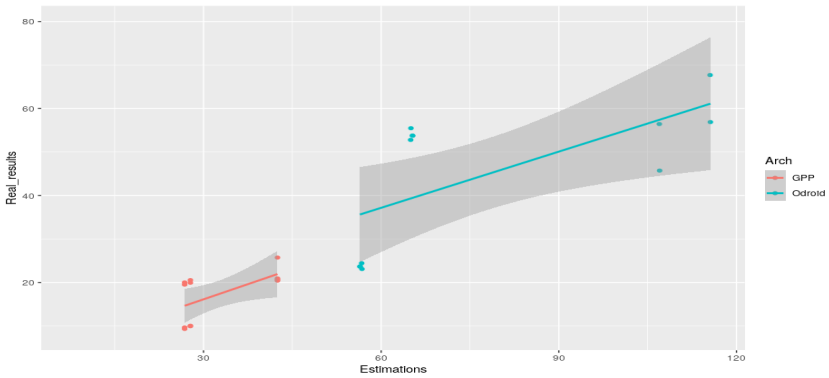

Figure 4 Fidelity analysis of the mAPN

exploit static and adaptive task, data and pipeline parallelism. The work in [9] determines the minimal required performance of the hardware to be used for Macro Dataflow. Similarly, authors in [16], vary parallelism at run-time for KPN applications by duplicating stateless processes. Previous works considered one possible implementation of a particular application. They expressed adaptivity by breaking its tasks further down, or consolidate them into less tasks if few hardware resources are available. They do not support deeper implementation changes where different algorithms can be used to perform the same task. Authors in [8] present a methodology that allows for multiple algorithmic variants for a given actor in an application. These algorithmic variants are passed as metadata to the compiler, which selects the best implementation for a given platform. Authors in [2] introduce a new programming language to ensure algorithmic choice at the language level. They provide algorithmic choices to the compiler. However, these two approaches do not support adapting the graph topology. Our approach is more general, combining parallelism adaptivity with algorithmic adaptivity.

6 Conclusion

In this paper we introduced mAPN, a novel model where multiple algorithmic variants are concisely represented in one compact graph. The abstract high-level model we proposed is built on top of KPN, taking advantage of its formal properties. Provided with additional information (annotation in terms of chosen metric), this model enlarges the variant space and eases the process of retaining feasible variants while meeting application/user/HW constraints. Furthermore, our approach is generic, combining parallelism adaptivity with algorithmic adaptivity. We motivated the mAPN approach with variants of ASA. We evaluated the end-to-end paths of the co-existing algorithmic variants in the graph using annotation on processes in terms of execution time. Being able to reason about these metrics and algorithmic adaptivity with a concise model will be key to achieve the most suitable implementation in terms of considered application/user/HW constraints. In future work, we will investigate efficient run-time algorithmic switching mechanisms, and considering annotation over more abstract domain-specific metrics like accuracy or robustness.

References

1 C. Aliprandi, C. Scudellari, I. Gallucci, N. Piccinini, M. Raffaelli, A. del Pozo, A. Alvarez, Ha. Arzelus, R. Cassaca, T. Luis, et al. Automatic live subtitling: state of the art, expectations and current trends. In *Proceedings of NAB Broadcast Engineering Conference: Papers on Advanced Media Technologies, Las Vegas*, page 23, 2014.

2 Jason Ansel, Cy Chan, Yee Lok Wong, Marek Olszewski, Qin Zhao, Alan Edelman, and Saman Amarasinghe. Petabricks: a language and compiler for algorithmic choice. *ACM Sigplan Notices*, 44(6):38–49, 2009.

3 Bishnupriya Bhattacharya and Shuvra S Bhattacharyya. Parameterized dataflow modeling for dsp systems. *IEEE Transactions on Signal Processing*, 49(10):2408–2421, 2001.

4 G. Bilsen, M. Engels, R. Lauwereins, and J. Peperstraete. Cycle-static dataflow. *IEEE Transactions on Signal Processing*, 44(2):397–408, 1996.

5 Adnan Bouakaz, Pascal Fradet, and Alain Girault. A survey of parametric dataflow models of computation. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 22(2):1–25, 2017.

6 Hasna Bouraoui, Chadlia Jerad, Anupam Chattopadhyay, and Nejib Ben Hadj-Alouane. Hardware architectures for embedded speaker recognition applications: a survey. *ACM Transactions on Embedded Computing Systems (TECS)*, 16(3):78, 2017.

7 Dai Bui and Edward A Lee. Streamorph: a case for synthesizing energy-efficient adaptive programs using high-level abstractions. In *Proceedings of EMSOFT*, page 20. IEEE Press, 2013.

8 Jeronimo Castrillon, Stefan Schürmans, Anastasia Stulova, Weihua Sheng, Torsten Kempf, Rainer Leupers, Gerd Ascheid, and Heinrich Meyr. Component-based waveform development: The nucleus tool flow for efficient and portable software defined radio. *Analog Integrated Circuits and Signal Processing*, 69(2-3):173–190, December 2011.

9 Marco Danelutto, Daniele De Sensi, and Massimo Torquati. A power-aware, self-adaptive macro data flow framework. *Parallel Processing Letters*, 27(01):1740004, 2017.

10 Karol Desnos, Maxime Pelcat, Jean-François Nezan, Shuvra S Bhattacharyya, and Slaheddine Aridhi. Pimm: Parameterized and interfaced dataflow meta-model for mpsocs runtime reconfiguration. In *2013 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*, pages 41–48. IEEE, 2013.

11 Brecht Desplanques, Kris Demuynck, and Jean-Pierre Martens. Adaptive speaker diarization of broadcast news based on factor analysis. *Computer Speech & Language*, 46:72–93, 2017.

12 Pascal Fradet, Alain Girault, Ruby Krishnaswamy, Xavier Nicollin, and Arash Shafiei. RDF: Reconfigurable Dataflow (extended version). Research Report RR-9227, INRIA Grenoble - Rhône-Alpes, December 2018. URL: `https://hal.inria.fr/hal-02079683`.

13 Amir Hossein Ghamarian, Marc CW Geilen, Sander Stuijk, Twan Basten, Bart D Theelen, Mohammad Reza Mousavi, Arno JM Moonen, and Marco JG Bekooij. Throughput analysis of synchronous data flow graphs. In *Sixth International Conf. on Application of Concurrency to System Design (ACSD'06)*, pages 25–36. IEEE, 2006.

14 Amir H Hormati, Yoonseo Choi, Manjunath Kudlur, Rodric Rabbah, Trevor Mudge, and Scott Mahlke. Flextream: Adaptive compilation of streaming applications for heterogeneous architectures. In *Proceedings of PACT*, pages 214–223. IEEE, 2009.

15 Gilles KAHN. The semantics of a simple language for parallel programming. *In Information Processing*, 74:471–475, 1974.

16 Robert Khasanov, Andrés Goens, and Jeronimo Castrillon. Implicit data-parallelism in Kahn process networks: Bridging the MacQueen Gap. In *Proceedings of PARMA-DITAM*, pages 20–25. ACM, 2018.

17 Mohaddeseh Nosratighods, Eliathamby Ambikairajah, and Julien Epps. Speaker verification using a novel set of dynamic features. In *18th International Conference on Pattern Recognition (ICPR'06)*, volume 4, pages 266–269. IEEE, 2006.

18 Claudius Ptolemaeus. *System Design, Modeling, and Simulation using Ptolemy II, 2014*. Ptolemy.org, 2014.

19 John R. Rice et al. The algorithm selection problem. *Advances in computers*, 15(65-118):5, 1976.

20 Lars Schor, Iuliana Bacivarov, Hoeseok Yang, and Lothar Thiele. Adapnet: Adapting process networks in response to resource variations. In *Proceedings of CASES*, page 22. ACM, 2014.

353 **21** Sander Stuijk, Marc Geilen, Bart Theelen, and Twan Basten. Scenario-aware dataflow:
354 Modeling, analysis and implementation of dynamic applications. In *Proceedings of SAMOS*,
355 pages 404–411. IEEE, 2011.
356 **22** Anastasia Stulova, Rainer Leupers, and Gerd Ascheid. Throughput driven transformations
357 of synchronous data flows for mapping to heterogeneous mpsocs. In *Proceedings of SAMOS*,
358 pages 144–151. IEEE, 2012.
359 **23** Roberto Togneri and Daniel Pullella. An overview of speaker identification: Accuracy and
360 robustness issues. *IEEE Circuits and Systems Magazine*, 11(2):23–61, 2011.
361 **24** OB Tuzun, M Demirekler, and KB Nakiboglu. Comparison of parametric and non-parametric
362 representations of speech for recognition. In *Electrotechnical Conference, 1994. Proceedings.,*
363 *7th Mediterranean*, pages 65–68. IEEE, 1994.