Analysis of Process Traces for Mapping Dynamic KPN Applications to MPSoCs

Andrés Goens, Jeronimo Castrillon

TU Dresden, Center for Advancing Electronics Dresden (cfaed),
Chair for Compiler Construction
Dresden, Germany
{andres.goens,jeronimo.castrillon}@tu-dresden.de

Abstract. Current approaches for mapping Kahn Process Networks (KPN) and Dynamic Data Flow (DDF) applications rely on assumptions on the program behavior specific to an execution. Thus, a near-optimal mapping, computed for a given input data set, may become sub-optimal at run-time. This happens when a different data set induces a significantly different behavior. We address this problem by leveraging inherent mathematical structures of the dataflow models and the hardware architectures. On the side of the dataflow models, we rely on the monoid structure of histories and traces. This structure help us formalize the behavior of multiple executions of a given dynamic application. By defining metrics we have a formal framework for comparing the executions. On the side of the hardware, we take advantage of symmetries in the architecture to reduce the search space for the mapping problem. We evaluate our implementation on execution variations of a randomly-generated KPN application and on a low-variation JPEG encoder benchmark. Using the described methods we show that trace differences are not sufficient for characterizing performance losses. Additionally, using platform symmetries we manage to reduce the design space in the experiments by two orders of magnitude.

1 Introduction

Architecture trends show a growing number of processors and heterogeneity in embedded systems. The problem of leveraging the growing complexity of modern multi-processor systems-on-chip (MPSoCs) is as relevant as ever. In many application domains it is well-established to use programming abstractions such as Kahn Process Networks (KPN) [10] or actor-based data flow models like Synchronous Data Flow (SDF) [12] and dynamic data flow (DDF) [3] for describing applications. These abstractions allow synthesis tools to reason on a high-level about physical resource allocation within the chip. They model the application by using a directed graph, where so-called actors or processes, represented by the nodes in the graph, communicate with each other via channels, which are in turn represented by edges. Much work has been done regarding the problem of mapping KPN and data flow applications to complex hardware architectures

for optimal throughput, resource usage or energy-efficiency [16]. The heuristics used for this, however, rely on a well-defined program behavior. In the case of SDF applications, for example, the very nature of the model allows synthesis tools to reason about mapping by using a topology matrix and finding repetition vectors in its kernel, which fully describe the communication behavior between actors [12]. Finding near-optimal solutions in more general models, which do not have constraints on the program behavior as strong as those of SDF, is a much more complex task. There are several current approaches to static mapping [7,14,18,7]. All these approaches are sensitive to the selection of the input stimuli that induce the observed trace. To deal with multiple different executions, authors suggest to compute a mapping for every situation and then pick the best configuration. For example, for buffer sizing, one approach is to select the largest size across all configurations [2]. An alternative to deal with variations is to use the so-called real-time calculus [19], in which events are modeled by arrival curves that describe upper and lower bounds on the event rates.

In this paper we seek to improve the current state of trace-based mapping flows to better support multiple traces for one application. We do this in two ways: by using trace theory, defining metrics in order to compare application traces and by using group theory to describe and utilize symmetries in the architecture. Trace theory has been a well-established model for concurrency for decades since its first formal formulation in 1977 by Mazurkiewicz [13]. Metrics for traces have been defined in very different contexts [8], or for similar applications very specific metrics have been considered [11]. To the best of our knowledge, however, trace metrics have never been used in the general context of analyzing KPN processes.

2 Process Traces and Histories

In this section we present our proposed trace analysis methods for the application side. To this end we introduce the formal concepts of traces and histories, explain their relationship and define a metric on the space of traces and of histories. We then describe experimental results obtained by applying these methods to randomly-generated KPN traces and on a JPEG encoder.

2.1 Traces and Histories

Traces and *histories* are both generalizations of strings. They are well-known as models for concurrently executing processes. Informally, we model concurrently executing processes as a string over an alphabet Σ, where the words of the alphabet represent events of the system. In a regular string all occurring characters (or events) have a well-defined sequential ordering. When two contiguous characters, however, represent independent events in the system, then we do not distinguish their order in the trace: we consider two traces as equal when we can convert one to the other by just rearranging independent characters.

More formally, let $\Sigma_1, \ldots, \Sigma_n$ be n alphabets, and consider the alphabet $\Sigma = \Sigma_1 \cup \ldots \cup \Sigma_n$, the union of those alphabets. This union is not necessarily disjoint. We define a *dependence subset* D of Σ^2 by $D = \Sigma_1^2 \cup \ldots \cup \Sigma_n^2$. From this we define the set $I = \Sigma^2 \backslash D$. It can be used to define an equivalence relation $\sim$ on the set of strings Σ^*. We say that $ab \sim ba$, if and only if $(a, b) \in I$. This induces an equivalence relation $\sim$ on Σ^* by extending it to all strings (the reflexive, transitive symmetric closure). We define the *set of traces* as the factor set of equivalence classes $\Sigma^* / \sim$. Since strings with concatenation have the algebraic structure of a monoid, and concatenation and the epimorphism to the equivalence class $\sim$ commute, $\Sigma^* / \sim$ is also a monoid with concatenation. It is therefore usually called the *trace monoid* [9].

Histories are similar. Instead of an arbitrary concatenation of the various independent strings, we consider a history to be a tuple of strings, one in each of the alphabets Σ_i. The individual alphabets represent the possible events for individual processes. These alphabets can have some common characters between them, in which case $\Sigma_i \cap \Sigma_j \neq \varnothing$ holds. These common characters represent synchronization events: they happen in two or more processes at the same time.

We can think of a history as a log of an application which represents all events in a parallel execution with different tasks or processes. The projection onto the alphabet of a single process represents the individual history of that process, independent of the others. With respect to component-wise concatenation, the set of histories over the alphabet $\Sigma = \Sigma_1 \cup \ldots \cup \Sigma_n$ is also a monoid, which is why it is often called the *history monoid* [9].

These two structures, the trace and the history monoid, are isomorphic. We either list events sequentially in a trace, where we don't distinguish the order of independent events, or we define the sequential history of each process independently. A formal proof of this fact can be found in [9].

2.2 Metrics

A metric acts as a way of measuring distance between objects. If we consider traces and histories as descriptions of the behavior of individual executions of a software built of concurrent processes, a metric acts as a way of comparing said execution behaviors.

There exists a plethora of metrics on strings, which are used from coding theory to DNA analysis and approximate string matching. Notable examples include the Hamming distance which only counts the number of equal letters, or the edit distance, which counts the minimal number of deletions, insertions and substitutions needed to go from one string to another. We can generalize these metrics to histories (and thus, traces) with the following theorem:

Theorem 1. *Let $\Sigma = \Sigma_1 \cup \ldots \cup \Sigma_n$ be an alphabet and d be a metric on the strings Σ^* over Σ. Then d induces a metric $\bar{d}$ on the set of histories H over $(\Sigma_1, \ldots, \Sigma_n)$ with projections $\pi_1, \ldots, \pi_n$ by*

$$\bar{d}(x = (x_1, \ldots, x_n), y = (y_1, \ldots, y_n)) = \sum_{i=1}^{n} d(x_i, y_i) = \sum_{i=1}^{n} d(\pi_i(x), \pi_i(y)) \quad (1)$$

Proof. Let $x, y, z \in H$ be histories.

1. *Let $\bar{d}(x, y) = 0$. Then $d(\pi_i(x), \pi_i(y)) = 0$ for all $i = 1, \ldots, n$. Since d is a metric, it means that $\pi_i(x) = \pi_i(y)$ for all i. This implies that $x = y$ since it holds for all projections.*
2. *By definition (Eqn. 1) it is immediately obvious that, since d is a metric*

$$\bar{d}(x, y) = \sum_{i=1}^{n} d(\pi_i(x), \pi_i(y)) = \sum_{i=1}^{n} d(\pi_i(y), \pi_i(x)) = \bar{d}(y, x)$$

3. *Finally, the triangle equation also follows in a similar fashion:*

$$\bar{d}(x, y) = \sum_{i=1}^{n} \underbrace{d(\pi_i(x), \pi_i(y))}_{\leqslant d(\pi_i(x), \pi_i(z)) + d(\pi_i(z), \pi_i(y))} \leqslant \sum_{i=1}^{n} d(\pi_i(x), \pi_i(z)) + d(\pi_i(z), \pi_i(y))$$

$$= \sum_{i=1}^{n} d(\pi_i(x), \pi_i(z)) + \sum_{i=1}^{n} d(\pi_i(z), \pi_i(y)) = \bar{d}(x, z) + \bar{d}(z, y)$$

Similar to this construction, and inspired by the l_p norms, we can define other metrics on histories (and traces).

Let $p \in \mathbb{R}_{\geq 1}$ be a real number, greater than or equal to one. Further let $\Sigma = \Sigma_1 \cup \ldots \cup \Sigma_n$ be a history alphabet and let $d_i' : \Sigma_i^* \to \mathbb{R}_{\geq 0}$ be a metric on $\Sigma_i, i = 1, \ldots, n$. Let $H \subseteq \Sigma_1^* \times \ldots \times \Sigma_n^*$ be the set of histories on Σ, with the corresponding projections $\pi_i : H \to \Sigma_i, i = 1, \ldots, n$. We call the mapping

$$d_p : H \times H \to \mathbb{R}_{\geq 0}, (x, y) \mapsto \sqrt[p]{\sum_{i=1}^{n} d_i'(\pi_i(x), \pi_i(y))^p}$$

the p-metric on the histories. Similarly, we can define a ∞ metric d_∞ as $d_\infty(x, y) = max_{i=1,\ldots,} d(\pi_i(x), \pi_i(y))$. The proof that these induce metrics is very similar to that of Theorem 1.

2.3   Trace analysis

To have controlled differences in our traces, we use random KPN traces. We generate them with a modification of the open-source software tool sdf^3 [17]. Concretely, we generate a random SDF application, and subsequently modify it to have a less static behavior. We do this by generating a set of possible, different input/output behaviors and randomly varying between them at run-time. For realistic behavior, we do this only on some KPN processes, while others keep their static (SDF) behavior. This method is inspired by the random KPN generation described in [5]. Once the application has been generated, different traces are created. This is achieved by fixing the possible behaviors and only randomizing the frequencies of occurrence.

For evaluating mappings we use a discrete-event-simulator similar to the one described in [6]. As the target architecture we use a virtual platform, also similar
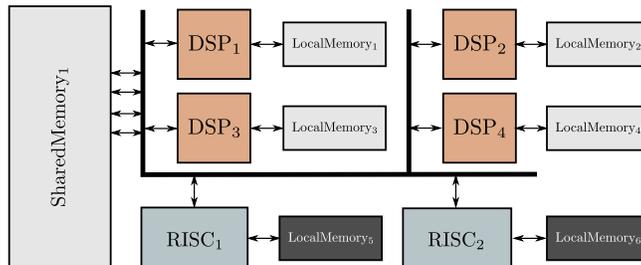
Fig. 1: A diagram of the test architecture used

to the one described in [6]. It has two identical RISC (ARM) processors and four identical vector DSPs. A diagram of this test architecture can be seen in Figure 1.

For evaluating the methods proposed in this section we used a fixed, randomly generated process network which had four different processes and four FIFO channels. We generated 1000 different random process traces of diverse lengths and behaviors. For each of these 1000 traces we calculated the optimal mapping by using the discrete event simulator and exhaustively evaluating all $6^4 = 1296$ different possible (process-to-processor) mappings. For buffer sizing we used a simple strategy assigning the same size to the buffers on all traces for an accurate comparison. This approach is inefficient and time-consuming, but only by using truly optimal mappings can we achieve a valid analysis. Without the optimality of the mapping there is no guarantee that it is good for a trace, even if it was specifically calculated for it.

Random traces provide only limited insight into this problem. To validate our approach we also considered a JPEG encoder with an existing implementation as a KPN. The JPEG encoder needs to perform run-length encoding, which exhibits dynamic behavior for the KPN channels. We executed the JPEG encoder on a benchmark consisting of 200 images adapted from the BSDS500 Benchmark [1].

The exponential scaling of the exhaustive mapping evaluation is also the reason why a network with only four processes was chosen for the random traces. For larger applications where the problem size makes exhaustive evaluation prohibitively long, as is the case for the JPEG encoder, good meta-heuristics like evolutionary algorithms can be considered as a replacement. While this does not guarantee the same accuracy for the comparison, using the results of a good meta-heuristic should produce a solid basis for comparison nevertheless. For the JPEG encoder we use simple heuristics from the literature (e.g., load balancing).

2.4 Results

We chose a reference trace for comparing. Then, for each of the 1000 random traces we compared the optimal run-time obtained using the optimal mapping with the run-time obtained using the reference mapping (in general only optimal for the reference trace). From the quotient of both we obtained a slowdown factor $\geqslant 1$. Similarly, we calculated the distance between each trace and the

reference one. Using this we analyze the correlation between trace distance and the slowdown from using the sub-optimal mapping. The results can be seen in Figure 2. This figure uses three different induced metrics from two string metrics for a total of six metrics. They have been normalized to one within the data-set for comparison. The axes on Figure 2b were adjusted not to show the trivial points at $(0, 1)$ (for the reference trace). This is for the sole purpose of a better visual scaling of the plot.

(a) Induced from the edit distance   (b) Induced from the Hamming distance
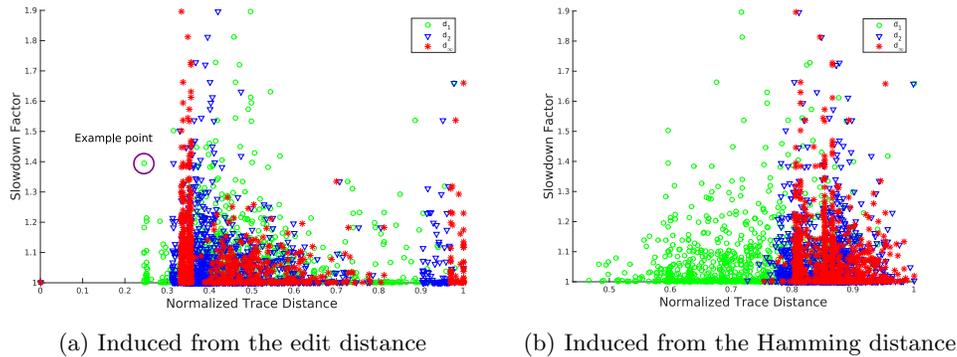
Fig. 2: Application slowdown as a function of different trace distances

As an example, consider the point marked in Figure 2a. This point has the coordinates $(0.24, 1.42)$. It means that the distance between the trace corresponding to the point, and the reference trace was 24% of the maximal distance in the plot (concretely, $d_1 = 101$ with a maximal distance of 424). The 1.42 slowdown factor means that the execution time of the trace with the reference mapping was 42% slower than with its own optimal mapping.

Altogether, Figure 2 shows a low correlation between the trace distance and how good the mapping of one trace is for the other one. Concretely, the correlation coefficients are -0.014, -0.077, -0.095, 0.119, 0.010, and -0.059, for the d_1, d_2 and d_∞ norms induced by the edit and Hamming distances, in that order.

JPEG Encoder

Figure 4 shows a histogram of different trace metrics for the 200 JPEG encoder executions. The traces were normalized with the distance from the reference trace to the empty trace, to give an idea of how much variation was between the traces. The JPEG encoder has variation in traces due to the run-length encoder, which is a small function that sends a different amount of tokens depending on the compressed data. However, the majority of the computation time is due to the discrete cosine transform, which has a static behavior. Even though the

run-length encoder represents just a small fraction of the computation, we found performance and trace behavior deviations. By using the mapping tailored for a different trace, a slowdown of up to 1.77% was observed. More importantly though, we see that different inputs yield different behaviors, represented by different traces. We also see that these differences have a negative impact on performance, albeit a small one in this case. In the future we plan to investigate further applications where the dynamic data flow part of the application amounts to a more significant percentage of the execution.

3 Permutations of Mappings

From the trace analysis above we see that distance analysis itself does not suffice to infer the performance of different mappings. Instead, in this section we consider the problem from the perspective of the mappings and the architecture, as opposed to that of the traces and the application. We take advantage of the fact that heterogeneous platforms have some degree of symmetry. We formally define and explore this symmetry, and present a strategy to reduce the design space that leverages it.

3.1 Problem Formulation

Mathematically, we can formulate our problem as follows: Let P be a set of physical resources (e.g. processing elements, on-chip memories) and let L be a set representing logical elements (e.g., processes, FIFO channels). We define a valid mapping $m : L \to P$ as a mapping in the mathematical sense (a function), such that it respects the KPN structure. Formally, let G be a subgroup of the symmetric group of the physical resources S_P. The canonical action of the group G on P induces an action on the set of mappings $m : L \to P$: for $g \in G$ and $m : L \to P$ a mapping, i.e. $(g \cdot m)(l) := g \cdot m(l)$ for all $l \in L$. We require of a symmetry group that the run-time for all traces is an invariant of the group action. In particular, this means that the action of G on the set of mappings restricts to an action on the set of valid mappings. This implies, for example, that we only consider symmetries of the architecture that map processors to processors and communication resources to equivalent communication resources. We define equivalence classes for mappings: we say two mappings m, m' are equivalent if there exists a symmetry of the architecture $g \in G$ such that $g \cdot m = m'$, i.e., if m and m' are in the same orbit under the action induced by G on the set of valid mappings.

For example, let $P = \{\mathrm{RISC}_1, \mathrm{RISC}_2, \mathrm{DSP}_1, \ldots, \mathrm{DSP}_4\}$ be the processor set of the architecture from the experimental setup in last section (see Figure 1), and let $L = \{p_1, \ldots, p_4\}$ be the process set of the four-process KPN used in the example from last section. For simplicity, we consider an elementary, symmetric communication model in this example where communication resources and processors are coupled. Then the group G that can swap both RISC processors and allows any permutation of the four DSP processors is the symmetry group of

this architecture. It is isomorphic to $S_2 \times S_4$, i.e., the direct product of the symmetric groups on two and four elements respectively. As an example, consider the mappings

$$m_1 : p_1 \mapsto \mathrm{RISC}_1, p_2, p_3 \mapsto \mathrm{DSP}_2, p_4 \mapsto DSP_3$$
$$m_2 : p_1 \mapsto \mathrm{RISC}_1, p_2, p_3 \mapsto \mathrm{DSP}_1, p_4 \mapsto DSP_4$$
$$m_3 : p_1 \mapsto \mathrm{RISC}_1, p_2, p_4 \mapsto \mathrm{DSP}_2, p_3 \mapsto DSP_3.$$

Then, m_1 and m_2 are equivalent, however neither of them is equivalent to m_3.

The motivation for this definition of equivalence is that if two processors are equal, then it usually should make no difference if one or the other is chosen for the mapping. This can also be used for taking communication into account, for example when there is additional symmetry from multiple memories or differences in local memories break the processor symmetry.

Groups with this structure are by far the most common symmetry group for heterogeneous architectures. A heterogeneous architecture which has n_1 equivalent processing elements of type 1, n_2 equivalent processing elements of type 2, and so forth, will have a symmetry group isomorphic to $S_{n_1} \times S_{n_2} \times \cdots$. However, the symmetry group of a subset of equivalent processing elements need not be a full symmetric group. For example, consider a simple homogeneous four-core architecture with a Network-on-Chip (NoC), such that the communication latency between adjacent processors is considerably lower than to non-adjacent ones. Then the adjacency of the processors should be kept with any symmetry transformation, which means the symmetry group is a dihedral group of a regular polygon with 4 sides, instead of the full symmetric group on 4 points. This group is called D_4, though some references call it D_8 because it has 8 elements. Figure 3 shows a schematic of this symmetry and an example of an allowed symmetry, one of the two generators, and a permutation that is not a symmetry of the architecture. It depicts the symmetry transformations with the green or red arrows, and an example of the action on a mapping of four processes, represented by the green or red circles.

3.2 Algorithmic Considerations

To identify equivalent mappings we need to find out if two elements are in the same orbit. Specifically, if m, m' are mappings, we need to test if $m' \in Gm$. This can in general be done with $\mathcal{O}(|Gm||S|)$ group element applications, where $|S|$ is a generating set of the group G, see Theorem 2.1.1 of [15]. Since we do not plan to deal with very complex symmetry groups, however, we used a different approach. Our approach is tailored for groups that have the form $\times_{i=1}^{k} S_{n_i}$, for $n_1, \ldots, n_k \in \mathbb{N}$. It takes advantage of the fact that group membership testing is a simple task in groups of this family. We devised a strategy that given mappings m, m' generates a tentative mapping $\sigma : \{1, \ldots, |P|\} \to \{1, \ldots, |P|\}$ such that if there exists a $\tau \in S_{|P|}$ such that $\tau \cdot m = m'$, then σ is a permutation and it holds that $\sigma \cdot m = m'$. We achieve this by iterating over all elements e in the definition domain of mapping m and updating σ to be correct for that element

(a) The generator element for the D_4 symmetry
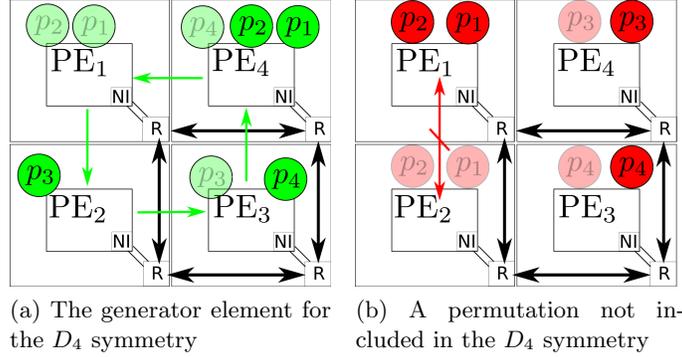
(b) A permutation not included in the D_4 symmetry

Fig. 3: Schematic representation on of the symmetry of a 4-core NoC Architecture

(i.e. $(\sigma m)(e) = m'(e)$), without guaranteeing that it remains a permutation. Using this tentative mapping strategy, we can find out if two mappings are in the same orbit, and if so, obtain a permutation that maps one to the other.

Algorithm 1 Orbit membership testing for direct products of symmetric groups

INPUT: $m, m', n = |P|$
OUTPUT:
if $Gm = m'$ **then**
    an element $g \in G$: $gm = m'$,
else false
end if
ALGORITHM:
$\sigma = \text{tentativeMapping}(m, m', n)$;
$\text{permutation} = \text{isPermutation}(\sigma)$;
$\text{ingroup} = \text{isInGroup}(\sigma, G)$;
$\text{maps} = \text{mappingsEqual}(\sigma \cdot m, m)$;
if permutation **and** ingroup **and** maps **then**
    return σ
else
    return false
end if

Algorithm 1 is more efficient than the standard algorithm. It uses a constant, single group application instead of $\mathcal{O}(|Gm||S|)$. However, it relies on the fact that if the proposed element σ is not in G, then there exists no element $g \in G$ mapping m to m', which is by no means obvious if G is not of the form $\times_{i=1}^{k} S_{n_i}$. For the general case, the standard black-box group algorithms should be used (see [15]).

The permutation approach has limited scalability. Using Burnside's Lemma [4], it is straightforward to prove that the factor by which the size of the search space is reduced is bounded by the cardinality of the symmetry group. In partic-

ular, the asymptotic scaling behavior of the size of the search space is the same, it still is in $\mathcal{O}(|P|^{|L|})$. However, we see in the experiments in the next section that not all equivalence classes of mappings are equally common. Further investigation could concentrate on identifying the most important equivalence classes and their corresponding traces.

3.3 Experimental results

For evaluating this approach, we used the same basic setup as in Section 2. Using Algorithm 1 we identified equivalence classes in the optimal-run-time mappings of the same set of 1000 random process from Section 2. We selected one trace and identified all traces which yielded mappings equivalent to it. In general, for a system with 6 processors total where there are two groups of 4 and 2 equivalent processors respectively, there exist exactly 83 possible mappings of four processes. This fact can be verified using Burnside's Lemma. Out of the 1000 traces a total 23 were equivalent to the first one. They all had a slowdown factor of exactly 1, as would be expected of equivalent mappings. This is, however, only a fraction of the 161 mappings with a slowdown factor of 1 compared to the first trace.

Furthermore, of all 83 possible mappings, considering symmetry, only 30 were present in the traces. Figure 5 shows a bar plot of the percentage of traces belonging to each group, for the 30 groups up to symmetry which had a trace with an optimal mapping in this group. They are ordered from most common to least common, and the remaining 53 unrepresented groups are not depicted.

This results show that while there are quite a few possible equivalence classes of mappings, 83 in this case, only very few are actually good mappings. The two most common equivalence classes are optimal for almost 30% of the traces, while the five most common ones actually account for more than half the traces.

The JPEG encoder was not considered for this since it would be too computationally intensive to calculate optimal mappings, and it would have yielded limited insight for the lack of optimality variations between traces.
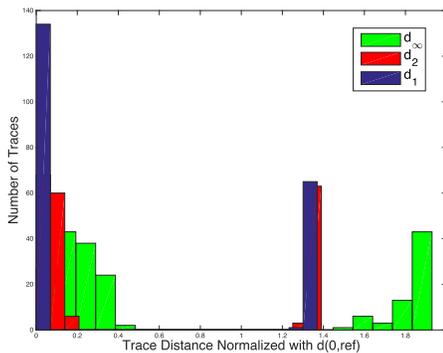

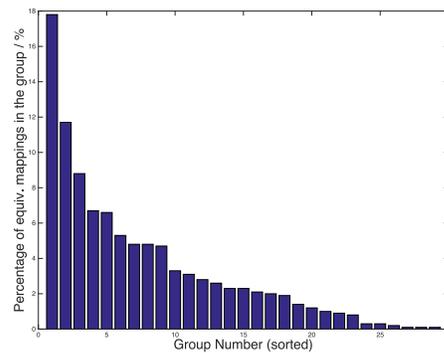Fig. 4: Histogram of normalized traces differences (JPEG encoder)

Fig. 5: Frequency of the equivalence classes of optimal mappings

4   Conclusion

In this paper we have considered the differences in execution behaviors of KPN and dynamic data flow applications as process traces or histories. We defined a metric space structure on traces and used it to measure the relationship between the trace distance, and how good the optimal mapping of one trace works for the other. For this, we also developed a framework for comparing them, which included exhaustive search on small examples to find true optimal mappings, for a solid comparison base.

The results from the JPEG encoder showed behavioral variations for different inputs in a real application. Additionally, the results from our analysis on random traces suggest no correlation between the trace distance and the goodness of the mappings of one to the other. This is a very revealing result. Its implications are twofold. First, it means that the difference between two traces does not suffice to t if we can use the same mapping for both. In particular this means we should devise more elaborate strategies for trace grouping, probably application-specific ones. The second, less obvious implication, is that very small differences in traces can have a very big impact on performance. Further work will focus on real applications with more dynamic behavior than the JPEG encoder that was used.

Apart form the behavior in the form of the traces, we also considered the problem from the perspective of the mappings. We defined a strategy to leverage symmetries in the architecture and evaluated it with the experiments used for the traces. We managed to reduce the search space from 1296 possible mappings to 83 possible equivalence classes of mappings, and found that very few equivalence classes of mappings account for the optimal throughput in the majority of traces.

Another direction for future work is to define strategies for identifying traces at run-time and using trace-specific information about the optimal mapping for dynamically improving adaptive execution. The analysis framework can be used to consider the problem of buffer sizing for multiple traces, which was not addressed in this work.

Acknowledgments

References

1. Arbelaez, P., Maire, M., Fowlkes, C., and Malik, J. Contour detection and hierarchical image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell. 33*, 5 (May 2011), 898–916.

2. Brunet, S. C. *Analysis and optimization of dynamic dataflow programs*. PhD thesis, Ecole Polytechnique Federale de Lausanne (EPLFL), 2015.

3. Buck, J. T., and Lee, E. A. Scheduling dynamic dataflow graphs with bounded memory using the token flow model. In *Acoustics, Speech, and Signal Processing, 1993. ICASSP-93., 1993 IEEE International Conference on* (1993), vol. 1, IEEE, pp. 429–432.

4. Burnside, W. *Theory of groups of finite order*. 1911.

5. Castrillon, J., and Leupers, R. *Programming Heterogeneous MPSoCs: Tool Flows to Close the Software Productivity Gap*. Springer, 2014.

6. Castrillon, J., Leupers, R., and Ascheid, G. Maps: Mapping concurrent dataflow applications to heterogeneous mpsocs. *IEEE Transactions on Industrial Informatics*, 99 (2011), p–19.

7. Castrillon, J., Tretter, A., Leupers, R., and Ascheid, G. Communication-Aware Mapping of KPN Applications onto Heterogeneous MPSoCs. In *DAC '12: Proceedings of the 49th annual conference on Design automation* (2012).

8. de Bakker, J., and Zucker, J. I. Denotational semantics of concurrency. In *Proceedings of the fourteenth annual ACM symposium on Theory of computing* (1982), ACM, pp. 153–158.

9. Diekert, V., Rozenberg, G., and Rozenburg, G. *The book of traces*, vol. 15. World Scientific, 1995.

10. Gilles, K. The semantics of a simple language for parallel programming. In *In Information Processing74: Proceedings of the IFIP Congress* (1974), vol. 74, pp. 471–475.

11. Kengne, C. K., Ibrahim, N., Rousset, M.-C., and Tchuente, M. Distance-based trace diagnosis for multimedia applications: Help me ted! In *Semantic Computing (ICSC), 2013 IEEE Seventh International Conference on* (2013), IEEE, pp. 306–309.

12. Lee, E. A., and Messerschmitt, D. G. Synchronous data flow. *Proceedings of the IEEE 75*, 9 (1987), 1235–1245.

13. Mazurkiewicz, A. Concurrent program schemes and their interpretations. *DAIMI Report Series 6*, 78 (1977).

14. Pimentel, A. D., Erbas, C., and Polstra, S. A systematic approach to exploring embedded system architectures at multiple abstraction levels. *Computers, IEEE Transactions on 55*, 2 (2006), 99–112.

15. Seress, Á. *Permutation group algorithms*, vol. 152. Cambridge University Press, 2003.

16. Singh, A. K., Shafique, M., Kumar, A., and Henkel, J. Mapping on multi/many-core systems: survey of current and emerging trends. In *Proceedings of the 50th Annual Design Automation Conference* (2013), ACM, p. 1.

17. Stuijk, S., Geilen, M., and Basten, T. SDF^3: SDF For Free. In *Application of Concurrency to System Design, 6th International Conference, ACSD 2006, Proceedings* (June 2006), IEEE Computer Society Press, Los Alamitos, CA, USA, pp. 276–278.

18. Thiele, L., Bacivarov, I., Haid, W., and Huang, K. Mapping applications to tiled multiprocessor embedded systems. In *Application of Concurrency to System Design, 2007. ACSD 2007. Seventh International Conference on* (2007), IEEE, pp. 29–40.

19. Thiele, L., Chakraborty, S., and Naedele, M. Real-time calculus for scheduling hard real-time systems. In *Circuits and Systems, 2000. Proceedings. ISCAS 2000 Geneva. The 2000 IEEE International Symposium on* (2000), vol. 4, pp. 101–104 vol.4.