Embeddings of Task Mappings to Multicore Systems

Embeddings of Task Mappings to Multicore Systems[⋆]

Andrés Goens and Jeronimo Castrillon

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

Abstract. The problem of finding good mappings is central to designing and executing applications efficiently in embedded systems. In heterogeneous multicores, which are ubiquitous today, this problem yields an intractably large design space of possible mappings. Most methods explore this space using heuristics, many of which implicitly use geometric notions in mappings. In this paper we explore the geometry of the mapping problem explicitly, for finding embeddings of the mapping space that capture its structure. This allows us to formulate new mapping strategies by leveraging the geometry of the mapping space, as well as improving existing heuristics that do so implicitly. We evaluate our approach on a novel mapping heuristic based on gradient descent, as well as multiple existing meta-heuristics. For complex architectures, our methods improved the results of established exploration meta-heuristics by about an order of magnitude in average.

1 Introduction

As the complexity of hardware systems increases, the problem of efficiently programming them not only becomes harder but also more crucial. For Cyber-Physical System (CPS) and embedded systems in general, there is a family of methods called software synthesis [6,3]. Inspired by hardware design flows, it aims to bridge the ensuing software productivity gap by integrating knowledge of the application and target multicore architecture into the compilation process.

A central concept in software synthesis is that of mappings, which divide the tasks in an application between the different processing elements (PEs) of the target architecture. Using mappings allow the compiler to produce code for heterogeneous Instruction-Set Architectures (ISAs), find especially efficient configurations and even increase the predictability in systems with homogeneous ISAs like ARM big.LITTLE [11]. The *mapping problem*, of finding such efficient mappings, is a difficult yet crucial step in this process. Because of the sheer size of the mapping space, which grows prohibitively large with increasing architecture

[⋆] This work has been funded in part by the German Research Council (DFG) through the TraceSymm project (number 366764507) and the Studienstiftung des deutschen Volkes.

and application complexities, exploring it exhaustively is intractable. Moreover, there is a complex relationship between a mapping and its performance, which in general cannot be modeled well analytically, which is why we need simulation to estimate it.

A great deal of research has focused on the mapping problem, spawning many sophisticated heuristics and meta-heuristics to find mappings with different characteristics. A survey of mapping approaches can be found in [29]. Many of these mapping meta-heuristics are based on an intuitive notion of a geometry of the mapping space. For example, the Tabu Search algorithm proposed in [18] relies on exploring neighboring mappings in order to improve their performance. Other similar principles underly methods like Simulated Annealing [22], L_p-adaptation [15] or genetic algorithms [9,24]. This is usually done in an adhoc fashion, without explicitly considering how to best endow the mapping space with such a geometric notion. Mappings are simply considered as integer vectors where the components represent the tasks, and the values represent the PEs these tasks are mapped to.

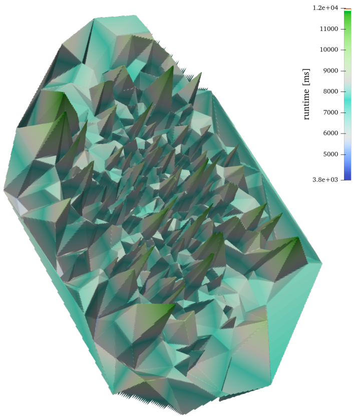


Fig. 1. A visualization of the mapping space. The axes are random proyections in the multi-dimensional space and have no direct interpretation.

Figure 1 shows a rendering of the design space of mappings for an audio filter C for Process Networks (CPN) benchmark onto the MPPA3 Coolidge architecture [16]. We generate this rendering using the methods of [17], generating a smoothening from a triangulation of a random projection of 1000 random mappings as an artistic interpretation that we can visualize with ParaView [1]. The height of the mountains and valleys in this landscape, as well as their coloring, represent the value of the execution time for the mappings being visualized. We see how the mapping space has multiple local minima and maxima, and generally a complex structure. The complexity of this structure is a direct consequence of the geometry we endow it.

In this paper we argue that we can find better geometries for the mapping space, simplifying the mapping problem by construction. We do this by considering a systematic approach to reason about the geometry of the mapping space. We also present some concrete alternative geometric structures for the mapping space, and discuss methods to find embeddings of these geometries to real vector spaces for computation. Since these embeddings can have a very high dimension, we also discuss and evaluate methods to reduce their dimension.

We show how this geometric structure can be leveraged by proposing a mapping algorithm based on the simple and well-known gradient descent method. Other algorithms which implicitly assume an underlying geometric structure also benefit from our approach, and we show how we can improve them as well. Finally, we evaluate these methods on their effect on multiple benchmarks, showing how the geometric structure plays an important role in the mapping problem and can be used to find novel mapping methods as well as improving established ones.

2 Related Work

Many flows exist that enable model-based design in a software synthesis flow [30,23,31,10,5]. In this paper we focus on the mapping problem addressed in these systems. As mentioned in the introduction, many such mapping algorithms implicitly use geometric structures of the mapping space [18,22,15,9,24]. These approaches do not explicitly model and reason about the geometry of the mapping space, this is done in an ad-hoc fashion.

In [32], Thompson and Pimentel exploit the mapping space structure explicitly, making explicit considerations of the geometry for defining operators in a genetic algorithm. These can both be seen as special cases of the methods presented in this paper, albeit for a simpler case with homogeneous architectures. In a related idea, in [33] they also introduce the concept of “shapes”, which is also an explicit consideration of some geometric aspects.

The work from Richthammer and others [26,27,25] is very similar in nature to the applications discussed in this paper. They also aim to improve Design-Space Exploration (DSE) methods by statically exploiting the architectural structure, although the concrete structure they exploit is different. They leverage the concrete structure of NoC meshes, by considering sub-structures in the architecture.

In previous work we have considered the geometry explicitly [12] but did not apply it to design-space exploration. Similarly, in [13] we discussed some geometric aspects of Network on Chip (NoC)-based architectures. This paper can be seen as an extension on the geometric considerations in these previous works.

3 Mapping Tasks to Multicores

As motivated in the introduction, the mapping problem [19] is the decision problem of assigning physical resources (hardware) to the logical tasks and data (software) of an application. We formulate this problem mathematically as finding graph morphisms. We model the architecture as a graph $A = (V_A, E_A)$. Here, the nodes V_A represent the PEs in the architecture and annotated with core types. The edges E_A represent communication primitives [7], an abstraction that models any method for communicating between PE, like caches, scratchpad memories or Direct Memory Access (DMA). The application we model as a graph $K = (V_K, E_K)$, where the nodes represent computation tasks (actors, processes) and the edges E_K represent data flow or dependencies. We model mappings as functions $m : K \to A$, i.e. assigning physical resources to the logical ones. A mapping also needs to be consistent. If it assigns two tasks $t_1, t_2 \in V_K$ to different PEs, when these tasks exchange data (i.e., $(t_1, t_2) \in E_K$), the data communication channel needs to be mapped to a physical channel that respects the task assignment: we require that $m((t_1, t_2)) = (m(t_1), m(t_2)) \in E_A$. This condition, mathematically, means precisely that a mapping respects the graph structure of K and A. In other words, a mapping is a *morphism of graphs* $m : K \to A$.

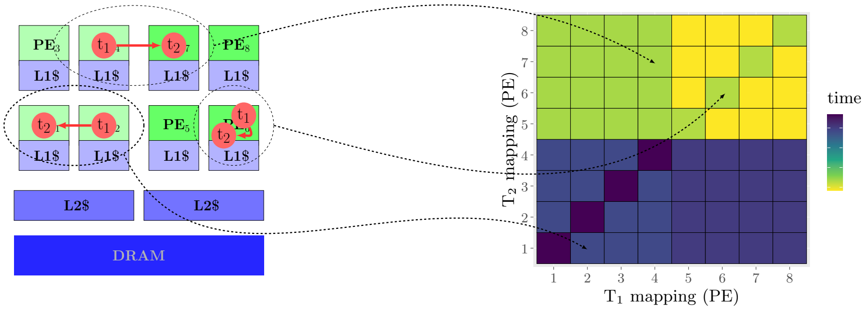


Fig. 2. An example of the mapping space for a simple two-task application.

Figure 2 depicts the mapping problem on a very simple example. The example is based on a telecom application of the E3S benchmark suite [8], chosen specifically because it consists of exactly two tasks, which allows the mapping space to be visualized in two dimensions. The mappings are plotted by encoding the mapping of each of the two tasks as the x and y coordinates of the grid, and the color of the squares in the grid encodes the (simulated) execution time on an Odroid-XU4 architecture. The actual values of the execution time are irrelevant here and have been deliberately omitted. In the figure it is clear that the minimal execution time is obtained by mapping the two tasks to two distinct Cortex-A15 (big) cores.

The example in Figure 2 is chosen deliberately to be so simple that it can be depicted in a figure. There are exactly $8^2 = 64$ mappings in the mapping space. For the audio filter application from the introduction, this space has already $8^8 = 16777216$ mappings and finding the minimal execution time is much less tractable. In general, the mapping space has cardinality $|V_A|^{|V_K|}$, and thus grows exponentially with the number of tasks $|V_K|$. For an 85-core architecture like the MPPA3 Coolidge [16], the mapping space of a moderately-large application with 42 tasks has more than 10^{81} possible mappings, more than there are atoms in the observable universe.

4 Metric Spaces

We endow the mapping space with a geometric structure by using the concept of metric spaces. In mathematics, metric spaces are an abstract structure that describes a space where distances can be measured. As such, it is described as a tuple (M, d), with a set M, the space, and a (non-negative) "distance" function $d : M \times M \to \mathbb{R}_{\geq 0}$, called the *metric*. To be a metric space, this distance function d has to follow the following axioms:

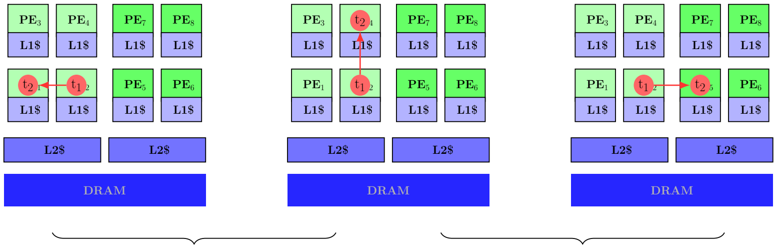
1. The distance of any object to itself is 0:
$$d(x, x) = 0, \text{ for all } x \in M.$$
2. The distance metric is symmetric:
$$d(x, y) = d(y, x), \text{ for all } x, y \in M.$$
3. A version of the triangle inequality:
$$d(x, z) \leq d(x, y) + d(y, z), \text{ for all } x, y, z \in M.$$

Traditionally, we encode mappings as vectors $m = \left(a_1, \ldots, a_{|V_K|}\right)$ where $a_i \in V_A$ are the PEs where task i is mapped. If we interpret these vectors as being (real) vectors in $\mathbb{R}^{|V_K|}$, we can endow them with a vector distance, like the Euclidean distance $d_{\text{Euclidean}}(v, w) = \sqrt{\sum_i (v_i - w_i)^2}$. This can be generalized to other p-norms, as $d_{L_p}(v, w) = \sum_i ((|v_i - w_i|)^p)^{1/p}$, which is a norm for $p \geq 1$. For $p = 1$, this norm is also known as the Mathattan distance, in allusion to the distance between buildings in a regular mesh like the streets of Manhattan. We can endow the space of mappings with a metric also by using the Hamming distance, which counts only the number of differing entries in the vector. However, none of these metrics are ideal for the mapping space, as we will now explain.

4.1 Metrics

In the example illustrated in Figure 3 we saw intuitively how mappings can be more or less similar. This intuitive notion clearly depends on the underlying architecture. It is the hardware architecture that determines the cost of communicating data between processes. In order to endow the space of mappings with a metric space structure, we should first do so with the architecture.

We can use the intuition behind the example to define a metric that takes latency into account this way [12]. The fundamental observation here is that in



$$\text{dist} = |(2,1) - (2,4)| = \sqrt{(4-1)^2} = 3 \qquad \text{dist} = |(2,4) - (2,5)| = \sqrt{(4-5)^2} = 1$$

Fig. 3. An intuitive example of distance between mappings.

a multicore architecture, communication between different PEs takes different amounts of time. There are multiple problems with using the communication time between PEs directly as a distance between PEs. Firstly, communication times depend on multiple factors: the latency and bandwidth of the communication resources used, the amount of data being sent, the (software) communication protocol, clock synchronization between hardware resources like the PEs and buses, arbitration or other contention issues, etc. Of course, we can model these to various degrees. However, the distance between PEs needs to be a fixed number and not a function of all these factors. As an approximation, however, we can use the expected latency for a package of a standardized size (e.g. 8 bytes). As an expected value, this is a fixed number, but through its statistical nature it can include as much complexity in the model as required[1].

The second issue we run into when using communication times for defining a distance is that, by definition, the distance between a point and itself has to be 0, but usually a PE has to communicate with itself using an $L1$ cache, scratchpad memory or similar, which has a small but non-zero latency. In this sense, the expected communication latency between cores is **not** a metric space distance, but it approximates one well. We propose thus to ignore this latency and set the distance to 0, to obtain the mathematical metric space structure.

Finally, this metric space structure depends strongly on the unit used to measure latency (e.g. cycles, milliseconds, etc), as well as on the absolute speed of the communication sub-architecture. Since the goal of exposing this structure is to leverage it for algorithmic decisions like finding good mappings, it is useful to have comparable distances between different architectures. For this, we propose to norm the metric distance function such that the average distance between PEs is 1.

Put together, these principles yield the following definition:

Definition 1 (Architecture Metric Space). *Let $A = (V_A, E_A)$ be an architecture graph and* $\text{lat} : V_A \to V_A$ *be the expected latency between PEs. Then we*

[1] If communication in the architecture is asymmetric, this will not define a metric. We can average the communication from p to q and from q to p to fix this, but we should probably consider this case separately.

set

$$d_A : V_A \times V_A, (p,q) \mapsto \begin{cases} \operatorname{lat}(p,q), & \textit{if } p \neq q \\ 0, & \textit{otherwise} \end{cases} \tag{1}$$

Remark 1. For an architecture graph $A = (V_A, E_A)$, the tuple (V_A, d_A) is a metric space.

Proof. Obviously $d_A(p,p) = 0$ for all $p \in V_A$, by definition, and $d_A(p,q) > 0$ for $p \neq q$ since the expected latency between PEs is always greater than 0. For $p,q,r \in P$ we have $d_A(p,q) + d_A(q,r) \geq d_A(p,r)$ since the expected latency of moving data from p to q and then to r will always be at least as much as moving it from p to r directly.

In this way we endow M with a discrete metric space structure, with a metric that reflects the memory subsystem of the architecture, or more generally, its communication. This is the metric introduced in [12], which has some issues. In particular, it does not distinguish between core types on heterogeneous systems. To fix this, we propose an alternative metric space structure on M, by adding extra dimensions for the communication and the computation. This is fundamentally very similar to adding channels in the mapping vectors. We thus define a metric on the channels, based on the metric defined by Definition 1. The distance between two channels $c, c' \in E_A$ is defined as $|\operatorname{lat}(c_1) - \operatorname{lat}(c_2)|$ for the communication channel between the cores. We then apply a similar concept for the cores, and take relative values of the expected runtime. Disregarding the ISA or micro-architecture, we can use the frequencies as a first estimation, which is what we do here. Thus, we set the distance between two cores p, p' as $|\operatorname{freq}(p) - \operatorname{freq}(p')|$. Obviously the frequency is not the best estimation of the expected differences in execution times between PEs, but we restrict our consideration to this for the scope of this paper. Future work should focus on finding better metrics for the mapping space.

This definition would not produce a metric, since distinct cores with the same frequency will have a distance of 0, and similarly channels with the same latency. To deal with this, we add a minimal distance between distinct cores and channels (e.g. 0.1 times the distance between the next two core types).

Application distances To go from A to M, we can use the same principle as the L_p norms and define $d(m,m') = (\sum_i d(m_i, m'_i)^p)^{1/p}$, which can immediately be checked to be a metric on M. This way we can consider, as a metric space (embedding), the structure of A to be

$$M \underbrace{\perp \ldots \perp}_{\times |V_K|} M, \text{ i.e. } M \underbrace{\times \ldots \times}_{\times |V_K|} \times M \text{ with } d(M_i, M_j) = \{0\} \text{ for all } i \neq j. \tag{2}$$

There are multiple issues with this as well. A very crucial problem with it is that this does not consider the dependencies between tasks in the application graph A, nor does it consider how multiple tasks might be more or less relevant. Many methods can be considered to account for this fact, like having factors for the dimensions of the copies of M in the orthogonal sum.

5 Low-Distortion Embeddings

We have seen so far how we can endow the mapping space with multiple metrics $d_M : M \times M \to \mathbb{R}_{\geq 0}$ to define distances between mappings. A problem with this is that the mapping space is a discrete space, with a very large cardinality. To algorithmically do any computation in this space, e.g. in DSE, we need to iterate through the whole space. For example, we might have a mapping m_0, for which we want to find all mappings that are within a radius r of it, i.e. compute the ball $B_r(m_0)$ with radius r around m_0. For this we need to iterate over all $m \in M$ and calculate if $d_M(m_0, m) \leq r$, which is intractable for all but the simplest examples.

To deal with this, we use established methods from discrete geometry to calculate *low-distortion embeddings*. A mapping $\iota : M \hookrightarrow \mathbb{R}^n$ such that there exists a $D > 0$ with

$$D^{-1} d(x, y) \leq \|\iota(x) - \iota(y)\| \leq d(x, y) \tag{3}$$

is called an embedding with distortion D. In other words, the *relative error* of the distances is at most D. Using convex optimization [20], we can calculate a low-distortion embedding for a finite metric space. This allows us to work with vectors of real numbers which make many algorithmic tasks scalable, e.g. computing random points in a ball.

Since the size of the mapping space grows exponentially with the number of tasks and changes for every application, computing such an embedding for a large mapping space every time we want to do DSE would also be intractable. We can avoid this by using the orthogonal sum construction from Equation 2. Given an embedding $\iota : A \hookrightarrow \mathbb{R}^k$ with distortion D for the architecture with a given metric d_A, we can construct an embedding ι^k of the mapping space defined as in Equation 2 with distortion D [12].

The mapping space can still have a very high dimension, a problem usually called the *curse of dimensionality.* With this construction, for the metric without the extra dimensions, the dimension of the embedding ι^k is $k|V_A| = |V_K||V_A|$. The Johnson-Lindenstrauss lemma can be used to reduce the dimension with a projection [20]. We do this with an iterative method, described in Algorithm 1

Algorithm 1 exponentially increases the dimension, running `numIterationPerDim` iterations of a Johnson-Lindenstrauss transform and testing the distortion to see if a target distortion has been reached. Using this algorithm, or variants thereof, we can control the trade-off between the distance and the dimension of the embedding.

To compare the different metrics and embeddings, for each of them we calculated 1000 mappings of an audio filter benchmark from the MAPS framework [5] on the Odroid XU4 platform. For a random subset of the $1000^2 = 10^6$ pairs of mappings we calculated the (relative) distance between two mappings and the relative runtime of the simulation on these two mappings.

There is basically no correlation between mappings distance and the (relative) runtimes. Two mappings can be very far apart and have (almost) the same

Algorithm 1 Iterative dimensionality reduction via the Johnson-Lindenstrauss lemma.

```
input: A discrete metric space $M$, a low-distortion embedding $\iota : M \hookrightarrow \mathbb{R}^n$ and a
    target distortion $D$.
output: An embedding with dimension $\leq n$ and distortion at most $D$.
dim $\leftarrow$ 1
while dodim $\leq n$
    for $\_ \in$ numIterationsPerDim do
        $\tilde{\iota} \leftarrow$ JLReduction($\iota$, dim)
        $\tilde{D} \leftarrow$ CalculateDistortion($\tilde{\iota}$)
        if $\tilde{D} \leq D$ then return $\tilde{\iota}$
    dim $\leftarrow$ 2dim
return $\iota$
```

execution time. This seems very plausible if we consider the symmetries of the problem [14], where multiple mappings are equivalent yet distinct. There are also other similarities in mappings. For example, audio filter benchmark computes an Fast Fourier Transform (FFT) and inverse FFT (IFFT) which are virtually identical, yet not precisely so.

A perhaps better assessment of the metrics is to ask what is the *maximal* relative execution time possible for a given distance. While we understand why two similar mappings that are far apart will have similar results, we would expect two mappings that are close to each other to have similar execution times with a good metric. To test this, we just consider the maximal relative execution time for two mappings which are (at most) the given distance apart. In the figure, the metrics described in this section are labeled as follows: We call `SimpleVector` the Euclidean norm on the mappings described as simple vectors. The metric based on the latencies as motivated from Figure 3 we denote as `Emedding`, whereas we add the annotation `ED` for the metric with extra dimensions which accounts for heterogeneous PEs.

Figure 4 shows this maximal relative execution times for the data of the Odroid XU4. It also includes a linear regression of the points for each metric and embedding. We can see that indeed, most of the metrics are pretty good as an *upper bound* on the relative runtime, as seen by the linear behavior on the figure.

The Odroid XU4 architecture is comparatively small, which obviously has consequences for the mapping space. The smaller (discrete) space results in an embedding space that is not as high-dimensional. Figure 5 shows how this situation changes for the MPPA3 Coolidge.

Similar to the case for the Odroid XU4, Figure 6 shows the same comparison with the maximal run-time difference for the MPPA3 Coolidge. Again we see that many metrics seem to be a decent bound for the difference in execution time, although less so than for the simple Odroid XU4 platform. The Euclidean norm on the simple vector mappings, for example, is considerably worse than in this case than in the Odroid XU4. We can quantify more precisely how good

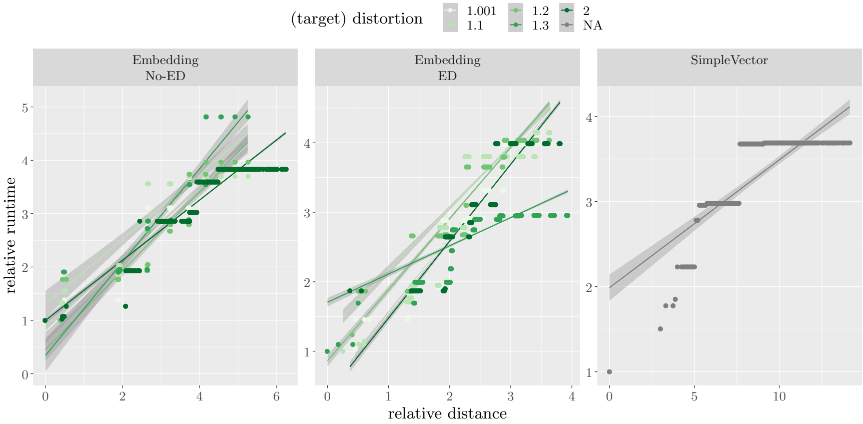


Fig. 4. Comparison of multiple distance metrics as predictors of the *maximal* run-time difference on the Odroid XU4 platform.

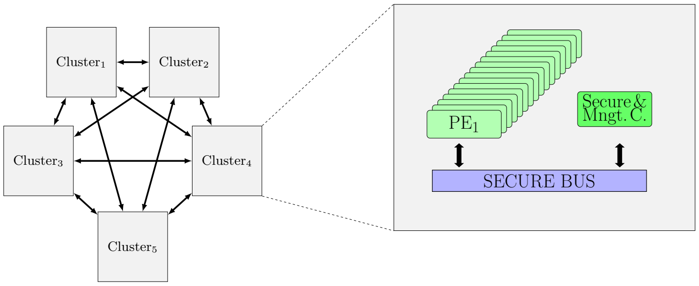


Fig. 5. The topology of the MPPA3 Coolidge platform, which consists of five clusters fully connected with a NoC, each cluster consisting of 16 identical general-purpose cores, as well as a secure and managment core..

metrics are as a bound for the execution time by comparing the R^2 value as goodness of fit assessment of the depicted linear regressions.

Figure 7 shows the R^2 value, comparing the predictive power of the different distance metrics and their embeddings. Here it is also very clear that the Euclidean norm on simple vectors is not so good for the MPPA3 Cooldige, while it is comparable to other metrics in the Odroid XU4. We also see how the curse of dimensionality yields a trade-off not only in the computation time (for larger-dimensional spaces), but also in the predictive quality of the different norms. This is more visible on the MPPA3 Coolidge. We see that the trade-off between the predictive power and the distortion is not very clear from this preliminary results. Future work should investigate this trade-off more in-depth.

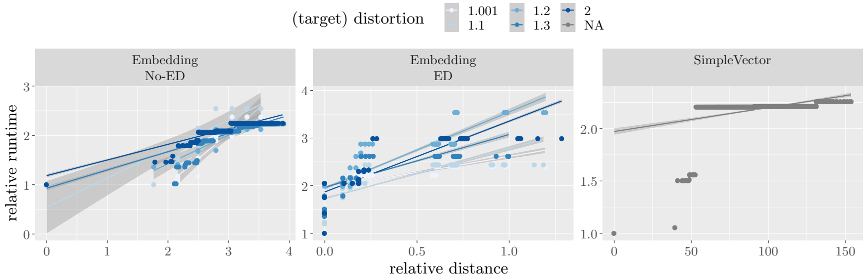


Fig. 6. Comparison of multiple distance metrics as predictors of the *maximal* run-time difference on the MPPA3 Coolidge platform.

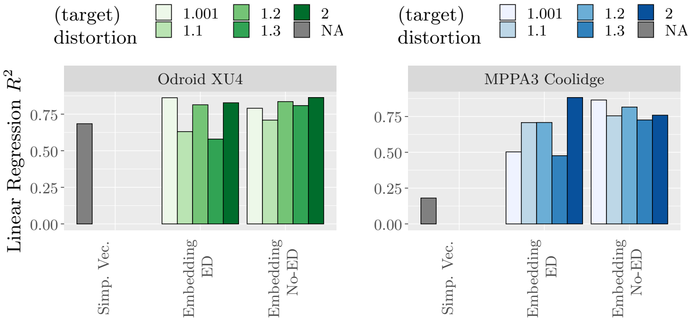


Fig. 7. Comparison of the predictive power of multiple distance metrics.

6 A Heuristic for Design-Space Exploration

Having defined a geometric interpretation for the mapping space, we show how we can leverage this in DSE. For this, we proposed a simple mapping algorithm based on the geometric structure of the mapping space. We discuss and evaluate our methods on the example objective of execution time, but do not use its structure directly. As such, we expect them to generalize to other objectives, like energy consumption.

Our algorithm is based on an observation of the geometry mapping space. The design spaces of mappings seem to consist of multiple islands of performance with similar properties, separated by poorly-performing mappings. Our “performance islands” hypothesis implies the mapping space is full of local minima. Guiding a local search towards an optimum should thus not be as conducive to good results. Instead, we can use a simple and fast meta-heuristic to find a local minimum quickly and apply it to multiple points spread around the design space’s geometry. As meta-heuristic for finding local minima we use the well-known gradient descent optimization algorithm with the momentum method [28]. For the step-size we use the Barzilai-Borwein [2] method.

In its regular form, this heuristic will quickly get stuck in a local minimum and produce poor mapping results, as confirmed by experiments (which we omit here). However, we can add a simple additional meta-heuristic to leverage the "performance islands" hypothesis. We start the heuristic at multiple random points, uniformly distributed in the design space, as defined the distance metric. In these spread-out locations we execute (parallel) gradient descent optimizations which we cancel as soon as they reach a local minimum, which empirically happens after a handful of iterations. The meta-heuristic returns the fastest mapping found in any of the different starting locations.

We can also improve other meta-heuristics by changing the vectors on which they operate, instead of the simple vectors of an ad-hoc geometry, we use our embeddings [12].

7 Evaluation

To evaluate our methods, we implemented the Tabu Search [18] and Simulated Annealing [22] mapping heuristics in mocasin [21], a framework for evaluating mapping algorithms. We also implemented our gradient-descent-based mapper. We configure the meta-heuristic to run on 5 different locations with a maximum of 20 iterations each, even though this maximum is almost never reached in practice in the experiments. We compare the results of these two heuristics on two benchmark suites, one being the Embedded System Synthesis Benchmarks Suite (E3S) [8] and another one based on MAPS (based on a language called CPN) [5]. The E3S suite consists of task graphs for 20 benchmarks from 5 different domains: auto-indust., networking, telecom, consumer and office-automation. The CPN benchmarks, on the other hand, are three benchmarks: a two-channel audio filter, a Histogram of Oriented Gradients (HOG)-based pedestrian recognition application and speaker recognition application [4]. For each meta-heuristic, each representation and each benchmark application, we measure the results of 10 runs with different random seeds.

Figure 8 shows the results of these experiments for the Odroid XU4 platform. The columns labeled as `SimpleVector` correspond to the Euclidean norm on the simple mapping vectors used commonly in most mapping scenarios. On the other hand, the label `MetricSpaceEmbedding` corresponds to the algorithms using the embedding as discussed here. Concretely, the embedding of the metric with the extra dimensions, without dimensionality reduction.

The logarithmic scale of the figure shows two different comparison criteria, the relative results of the mapping and the relative exploration time of the DSE. Both are normed to the results of the simulated annealing heuristic with the `SimpleVector` representation. For the DSE results, we summarize execution time as the geometric mean of the relative times of the benchmark, as simulated. The other metric is the relative exploration time. This is the time that the DSE needed to explore the design space. The error bars show the variance between the different benchmarks and the 10 different runs with different random seeds.

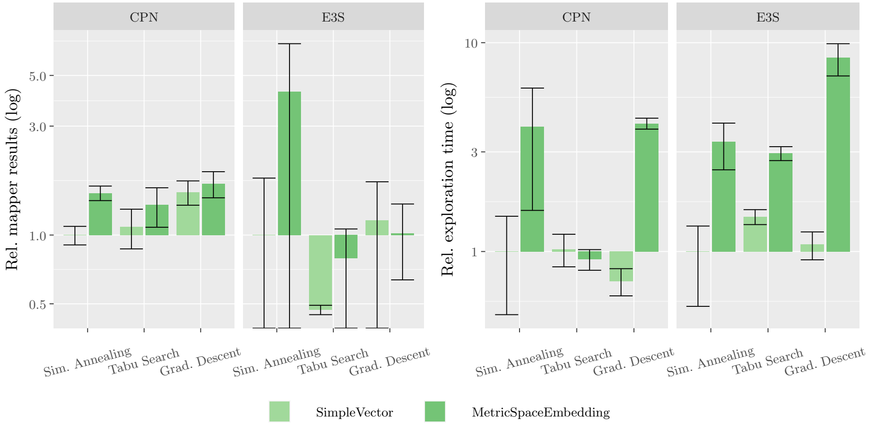


Fig. 8. The effect of embedding-based representations on the Odroid XU4 platform.

We see that changing geometry of the design space is not very effective for this simple architecture, although it does show more improvement for the E3S benchmarks. The gradient descent meta-heuristic with our performance island hypothesis is on par with the other meta-heuristics, which is already a strong result given the simplicity of the algorithm. As was seen before on the comparison of the metrics, the Euclidean norm on the simple vector representation is a decent metric for this space, which explains the results. On the other hand, using embeddings increases the execution time. This is because of the large dimension, and the necessity to do a nearest-neighbor approximation. In future work, applying methods for improving nearest-neighbor algorithms like in the Annoy library [2] could improve this time. We also did not reduce the dimension for this evaluation, to see the effects on the algorithm. In future work this trade-off could be exploited to improve the execution time.

Figure 9 summarizes the results of this experiments for the MPPA3 Coolidge platform, for which we showed that the metric space structure of our embedding-based representations is better than the canonical metric in the `SimpleVector` representation. We see that the results of the exploration are significantly better for both meta-heuristics with the representations based on this better distance metric. More importantly, the gradient-descent-based heuristic performs considerably better even. In some cases, the results of this simple heuristic are *on average* over an order of magnitude better than the other unmodified meta-heuristics. This is perhaps a statement about how poorly established meta-heuristics perform on a very complex design space, more so than a testament in favor of our gradient-descent-based heuristic. It shows thus, that our geometric representations are particularly useful in more complex architectures . Additionally, the effect on the exploration time is much less pronounced in this case, since the

[2] https://github.com/spotify/annoy

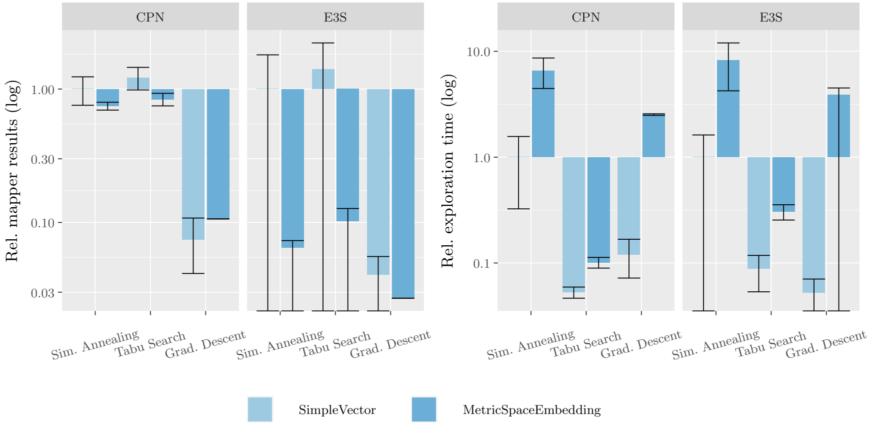


Fig. 9. The effect of embedding-based representations on the MPPA3 Coolidge platform.

overhead of the linear algebra involved becomes a smaller portion of the total exploration time.

8 Conclusions

In this paper we have seen how to endow the space of mappings to multicores with a geometric interpretation, and defined some metrics that might be better suited to describe the space than the ad-hoc simple vector structure used commonly. We have seen from experiments that this structure helps especially well in the DSE of more complex architectures. Importantly, it allows us to use simple algorithms like gradient descent for mapping, which otherwise was infeasible. For two different sets of benchmark suites, our heuristic armed with this geometric interpretation managed to find good mappings much more reliably than established heuristics on the complex architecture topology of the MPPA3 Coolidge.. Mapping heuristics based on tabu search and simulated annealing produced mappings about an order of magnitude worse *on average* for this architecture.

We believe the main contribution of this paper is the geometric view of the mapping space, not the metrics themselves. Future work should focus on finding better metrics. This might be especially conducive to machine learning algorithms for mapping, which usually work with embeddings as the ones described in this paper.

References

1. Ahrens, J., Geveci, B., Law, C.: Paraview: An end-user tool for large data visualization. The visualization handbook **717**(8) (2005)

2. Barzilai, J., Borwein, J.M.: Two-point step size gradient methods. IMA journal of numerical analysis **8**(1), 141–148 (1988)
3. Bhattacharyya, S.S., Murthy, P.K., Lee, E.A.: Software synthesis from dataflow graphs, vol. 360. Springer Science & Business Media (2012)
4. Bouraoui, H., Castrillon, J., Jerad, C.: Comparing dataflow and openmp programming for speaker recognition applications. In: Proceedings of PARMA-DITAM'19. pp. 1–6 (2019)
5. Castrillon, J., Leupers, R., Ascheid, G.: Maps: Mapping concurrent dataflow applications to heterogeneous mpsocs. IEEE Transactions on Industrial Informatics **9**(1), 527–545 (2011)
6. Castrillon, J., Sheng, W., Leupers, R.: Trends in embedded software synthesis. In: 2011 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation. pp. 347–354. IEEE (2011)
7. Castrillon, J., Tretter, A., Leupers, R., Ascheid, G.: Communication-aware mapping of kpn applications onto heterogeneous mpsocs. In: DAC Design Automation Conference 2012. pp. 1262–1267. IEEE (2012)
8. Dick, R.: Embedded systems synthesis benchmark suite (e3s) (2008), http://ziyang.eecs.umich.edu/~{}dickrp/e3s/
9. Erbas, C., Cerav-Erbas, S., Pimentel, A.D.: Multiobjective optimization and evolutionary algorithms for the application mapping problem in multiprocessor system-on-chip design. IEEE Transactions on Evolutionary Computation **10**(3), 358–374 (2006)
10. Erbas, C., Pimentel, A.D., Thompson, M., Polstra, S.: A framework for system-level modeling and simulation of embedded systems architectures. EURASIP Journal on Embedded Systems **2007**, 1–11 (2007)
11. Goens, A., Khasanov, R., Hähnel, M., Smejkal, T., Härtig, H., Castrillon, J.: Tetris: a multi-application run-time system for predictable execution of static mappings. In: Proceedings of the 20th International Workshop on Software and Compilers for Embedded Systems (SCOPES'17). SCOPES '17 (2017)
12. Goens, A., Menard, C., Castrillon, J.: On the representation of mappings to multicores. In: Proceedings of the IEEE 12th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC-18) (2018)
13. Goens, A., Menard, C., Castrillon, J.: On compact mappings for multicore systems. In: Pnevmatikatos, D., Pelcat, M., Jung, M. (eds.) Proceedings of the IEEE International Conference on Embedded Computer Systems Architectures Modeling and Simulation (SAMOS). vol. 11733, pp. 325–335. IEEE, Springer, Cham (Jul 2019)
14. Goens, A., Siccha, S., Castrillon, J.: Symmetry in software synthesis. ACM Transactions on Architecture and Code Optimization (TACO)
15. Hempel, G., Goens, A., Asmus, J., Castrillon, J., Sbalzarini, I.F.: Robust mapping of process networks to many-core systems using bio-inspired design centering. In: Proceedings of the 20th International Workshop on Software and Compilers for Embedded Systems (SCOPES '17). SCOPES '17 (2017)
16. inc, K.: Kalray mppa3 coolidge anouncement (2020), https://www.kalrayinc.com/release-of-third-generation-mppa-processor-coolidge/
17. Li, H., Xu, Z., Taylor, G., Studer, C., Goldstein, T.: Visualizing the loss landscape of neural nets. In: Neural Information Processing Systems (2018)
18. Manolache, S., Eles, P., Peng, Z.: Task mapping and priority assignment for soft real-time applications under deadline miss ratio constraints. ACM Transactions on Embedded Computing Systems (TECS) **7**(2), 1–35 (2008)

19. Marwedel, P., Bacivarov, I., Lee, C., Teich, J., Thiele, L., Xu, Q., Kouveli, G., Ha, S., Huang, L.: Mapping of applications to mpsocs. In: 2011 Proceedings of the Ninth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS). pp. 109–118. IEEE (2011)
20. Matoušek, J.: Lectures on discrete geometry, vol. 212. Springer Science & Business Media (2002)
21. Menard, C., Goens, A., Hempel, G., Khasanov, R., Robledo, J., Teweleitt, F., Castrillon, J.: Mocasin – rapid prototyping of rapid prototyping tools: A framework for exploring new approaches in mapping software to heterogeneous multi-cores. In: Proceedings of the 13th RAPIDO Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools, co-located with 16th International Conference on High-Performance and Embedded Architectures and Compilers (HiPEAC). RAPIDO '21, ACM, New York, NY, USA (Jan 2021)
22. Orsila, H., Kangas, T., Salminen, E., Hämäläinen, T.D., Hännikäinen, M.: Automated memory-aware application distribution for multi-processor system-on-chips. J. of Sys. Arch. **53**(11), 795–815 (2007)
23. Pelcat, M., Desnos, K., Heulot, J., Guy, C., Nezan, J.F., Aridhi, S.: Preesm: A dataflow-based rapid prototyping framework for simplifying multicore dsp programming. In: 2014 6th european embedded design in education and research conference (EDERC). pp. 36–40. IEEE (2014)
24. Quan, W., Pimentel, A.D.: Towards exploring vast mpsoc mapping design spaces using a bias-elitist evolutionary approach. In: 2014 17th Euromicro Conference on Digital System Design. IEEE (2014)
25. Richthammer, V., Fassnacht, F., Glaß, M.: Search-space decomposition for system-level design space exploration of embedded systems. ACM Transactions on Design Automation of Electronic Systems (TODAES) **25**(2), 1–32 (2020)
26. Richthammer, V., Glaß, M.: On search-space restriction for design space exploration of multi-/many-core systems. In: MBMV (2018)
27. Richthammer, V., Glaß, M.: Efficient search-space encoding for system-level design space exploration of embedded systems. In: 2019 IEEE 13th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC). pp. 273–280. IEEE (2019)
28. Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning representations by back-propagating errors. nature **323**(6088), 533–536 (1986)
29. Singh, A.K., Shafique, M., Kumar, A., Henkel, J.: Mapping on multi/many-core systems: survey of current and emerging trends. In: 2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC). pp. 1–10. IEEE (2013)
30. Stuijk, S., Geilen, M., Basten, T.: A predictable multiprocessor design flow for streaming applications with dynamic behaviour. In: 2010 13th Euromicro Conference on Digital System Design: Architectures, Methods and Tools. IEEE (2010)
31. Thiele, L., Bacivarov, I., Haid, W., Huang, K.: Mapping applications to tiled multiprocessor embedded systems. In: Seventh International Conference on Application of Concurrency to System Design (ACSD 2007). pp. 29–40. IEEE (2007)
32. Thompson, M., Pimentel, A.D.: Exploiting domain knowledge in system-level mpsoc design space exploration. Journal of Systems Architecture
33. Weichslgartner, A., Wildermann, S., Götzfried, J., Freiling, F., Glaß, M., Teich, J.: Design-time/run-time mapping of security-critical applications in heterogeneous mpsocs. In: Proceedings of the 19th International Workshop on Software and Compilers for Embedded Systems. pp. 153–162 (2016)