Journal of Signal Processing Systems
https://doi.org/10.1007/s11265-023-01884-6

Design Space Exploration for Partitioning Dataflow Program on CPU-GPU Heterogeneous System

Aurelien Bloch[1] · **Simone Casale-Brunet**[1] · **Marco Mattavelli**[1]

Received: 19 January 2023 / Revised: 11 July 2023 / Accepted: 13 July 2023

Abstract
Dataflow programming is a methodology that enables the development of high-level, parametric programs that are independent of the underlying platform. This approach is particularly useful for heterogeneous platforms, as it eliminates the need to rewrite application software for each configuration. Instead, it only requires new low-level implementation code, which is typically automatically generated through code generation tools. The performance of programs running on heterogeneous parallel platforms is highly dependent on the partitioning and mapping of computation to different processing units. This is determined by parameters that govern the partitioning, mapping, scheduling, and allocation of data exchanges among the processing elements of the platform. Determining the appropriate parameters for a specific application and set of architectures is a complex task and is an active area of research. This paper presents a novel methodology for partitioning and mapping dataflow programs onto heterogeneous systems composed of both CPUs and GPUs. The objective is to identify the program configuration that provides the most efficient way to process a typical dataflow program by exploring its design space. This is an NP-complete problem that we have addressed by utilizing a design space exploration approach that leverages a Tabu search meta-heuristic optimization algorithm driven by analysis of the execution trace graph of the program. The heuristic algorithm effectively identifies a solution that maps actors to processing units while improving overall performance. The parameters of the heuristic algorithm, such as the time limit and the proportion of neighboring solutions explored during each iteration, can be fine-tuned for optimal results. Additionally, the proposed approach allows for the exploration of solutions that do not utilize all hardware resources if it results in better performance. The effectiveness of the proposed approach is demonstrated through experimental results on dataflow programs.

Keywords Heterogeneous systems · GPU programming · Source-to-source compiler · Parallel computing · RVC-CAL · Dynamic dataflow programs · Design space exploration · Tabu-search

1 Introduction

The increasing demand for high computational power in processing platforms is driven by the growing needs of modern application programs. The limitations of Moore's Law in creating smaller circuit components and the challenges posed by rising logic gate frequencies have led to the growing adoption of heterogeneous processing platforms. In order to effectively leverage the available computational power of these platforms, advanced levels of domain-specific hardware specialization need to be explored as opposed to simply scaling up existing processing elements. As examples, Microsoft Arm [1], Nvidia Grace [2] and Jetson [3, 4], Apple silicon [5]) are all clear demonstrations of this trend whether it is for embedded system, personal computing or data center.

Dataflow programming has been demonstrated to be an effective method for managing large and parallel applications, addressing portability concerns across different platforms, and effectively exploring and exploiting parallelism opportunities [6, 7]. This is because dataflow languages are designed to expose the parallelism inherent in the process of executing tasks on data. This enables the rapid evaluation of various settings, such as mapping software kernels

✉ Aurelien Bloch
aurelien.bloch@epfl.ch

Simone Casale-Brunet
simone.casalebrunet@epfl.ch

Marco Mattavelli
marco.mattavelli@epfl.ch

[1] EPFL SCI-STI-MM, École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland

Published online: 31 July 2023

to hardware processing elements, without incurring the costly redesigns required in traditional imperative and/or platform-specific software programming. These redesigns often require manual rewriting and can consume significant amounts of developer time.

Over time, dataflow-based computing models have been effectively utilized in a plethora of industries, including, but not limited to, digital codecs implementation, high-frequency financial applications, and genomics analysis. Due to their expressiveness, mathematical rigor in the models, and independence from a specific architecture, these models have also been employed as a reference and standardization language in digital video coding, specifically for the definition of the recently developed MPEG AVC and HEVC video codecs [8–11].

As the utilization of diverse processing platforms becomes more prevalent, new areas of inquiry have emerged pertaining to the optimal configuration for scheduling, mapping, and partitioning of dataflow-based applications on a given platform. The dataflow model, as it is based on the concept of independent black-boxed units that communicate with one another using specific data channels. These boxes encapsulate the execution kernels that independently and atomically execute the operations designated for them. Communication between these boxes occurs solely through communication buffers, where data packets (termed tokens) are utilized for information exchange. However, even for simple applications, the number of configurations that must be evaluated is excessive for developers to test through trial and error or to exhaustively test all possible combinations. Therefore, systematic and automated methods are essential for identifying and evaluating efficient configurations and designs. Consequently, design space exploration methodologies and tools are crucial in the design process and exploration of effective design points. To effectively identify an efficient set of configuration parameters that meet specific performance requirements, automation tools are imperative. One such tool is TURNUS, a design space exploration tool developed through research by the authors of this paper. The methodology underlying this tool is based on the analysis of an execution graph of the program, called Execution Trace Graph (ETG). In this graph, each execution of a kernel (actor) is represented by a vertex, which are then connected based on functional and data dependencies. This design space exploration technique is made effective because, thanks to the program's dataflow computation model (MoC), it is possible to systematically assign weights to each node and edge of the graph and analyze it through heuristics based on graph theory [12, 13]. This design space exploration methodology has been successfully applied on a wide range of heterogeneous hardware platforms, including Many-Core [14], FPGAs [15], and MPSoCs [16, 17] platforms. These platforms can be standalone or integrated as a component of a larger heterogeneous system. However, the automatic design space exploration for CPU-GPU heterogeneous systems remains an open area of research. The unique architecture of GPUs poses challenges, such as limited control over scheduling and mapping on hardware resources and the dissimilar APIs used to program them.

This work aims to expand upon and provide a comprehensive summary of the design and optimization methodology that is based on dataflow models, as previously presented in [18, 19]. To do so, this work uses the heterogeneous CPU/GPU dataflow methodology from [20] and expand and integrate the profiling and performance estimation methodologies from [21–23]. The ultimate objective of this work is to automatically explore the design space for the partitioning and mapping of dataflow applications onto heterogeneous systems using a Tabu search meta-algorithm.

The main novel contributions of this work can be summarized as follows:

- The adaptation and development of an extension of the Tabu search algorithm to specifically address the needs and requirements of the heterogeneous platform methodology. This includes ensuring that the algorithm can efficiently and effectively handle the unique challenges presented by a heterogeneous system.
- The development of new static and dynamic analysis methodologies suitable for heterogeneous CPU-GPU systems, specifically tailored to the characteristics of these systems, which can help to improve the performance of dataflow applications running on them.
- The further extension and improvement of a simulator engine suitable for performance estimation of dynamic applications. This simulator engine is designed to provide accurate and reliable performance estimates for dataflow applications running on heterogeneous systems.
- The adaptation of the neighboring move generator to the specific heterogeneous CPU-GPU context. Additionally, it allows for the under-utilization of resources when necessary to optimize performance and achieve the best possible outcomes.

The paper is structured as follows: Section 2 outlines the necessary components for exploring the design space of a heterogeneous CPU-GPU platform. In Section 3, the Tabu search algorithm developed for this work is presented, while Section 4 defines and describes the neighborhood move generator. Section 5 introduces the three different methodologies used for evaluating the design points. The experimental results and comparisons of the different methodologies are presented in Section 6. Finally, Section 7 concludes the paper and highlights potential research objectives and future directions.

2 Design Space Exploration

This section presents a summary of the main features of a dataflow programming model that utilizes the concept of actors. Additionally, a review of the key elements necessary for exploring the design space of dataflow applications using the analysis and optimization methodology developed by the authors in [12, 13] is provided.

2.1 Dataflow Programming Model

A Dataflow Process Network (DPN) is a system in which multiple processes operate simultaneously and exchange information through unidirectional, first-in-first-out (FIFO) channels [24]. In this model, writing to the channel is done without interruption, while reading from it is blocking. In a dataflow process network, processes are made up of repeated cycles of computation known as *firings* of a dataflow *actor*. An actor defines a set of often functional, atomic tasks that comprise the overall processing.

In a dataflow network, the flow of data (referred to as *tokens*) between actors is clearly defined and access to shared data is only possible by transmitting packets of data. The research illustrated in this work is based on a dynamic dataflow Model of Computation (MoC) that uses a variation of the DPN mentioned previously. A key aspect of this MoC is that an actor's execution is broken down into a series of atomic computations, also known as *firings*. During each firing, an actor can retrieve a certain amount of input data, transmit a certain amount of output data, and modify its local memory if necessary, based on the input tokens and the values of its state variables. The specific computation performed by a single actor during a firing is referred to as an action. The action executed at any given time is determined by the input tokens and the values of the actor's state variables. This lack of data race and critical sections leads to more robust behavior in dataflow software, regardless of the computation policies being used, whether they are fully parallel or involve interleaving of actor executions [25].

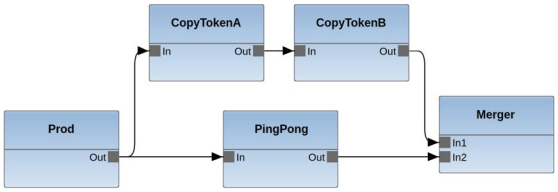
In recent years, a wide range of software languages have been utilized to execute the semantics of dataflow programs [26]. Some imperative languages such as Python, Java, and C/C++ have been improved by incorporating parallel operators while new languages supporting dataflow features like SISAL [27] and Ptolemy [28] have been created and standardized. Among this diverse selection, RVC-CAL [8] is the only formalized dataflow programming language that has ISO compliance and fully encompasses the complete behavioral characteristics of the DPN MoC. RVC-CAL has been employed as a reference and standardization language for novel digital video codec standards such as MPEG AVC and HEVC [8–11]. Each RVC-CAL actor is made up of a group of atomic firing functions, referred to as *actions*, and a set of internal memory that cannot be accessed by other actors, whether they are neighboring or not. Only one function can be executed at a time while the actor is in operation. To put it another way, for each actor, the collection of firing rules determines when an action is permitted to be fired. Each of these rules can be expressed as a function of the actor's internal variables and the availability and values of input tokens. More specifically, a firing rule can be defined as a Boolean function, including a selection of the action input pattern (i.e., specifying the required number of tokens for the action to be fired and to be removed from FIFO buffers) and the action guard condition (i.e., a Boolean expression defined using the actor's internal memory and the values of consumed input data).

The concepts in question are illustrated through the use of Fig. 1, which presents a basic example of an RVC-CAL dataflow application software. The graphic model of the network of actors in the example can be found in Fig. 1a. The dataflow program is made up of five instances of actors, including Prod, PingPong, CopyTokensA, CopyTokensB, and Merger. The RVC-CAL software code for the Prod actor is depicted in Fig. 1b, which features a single action that generates one token per execution and increments an internal counter. Additionally, a guard is in place to prevent the action from being executed more than four times. In the implementation of the PingPong actor, as seen in Fig. 1d, a schedule expression that function as a Finite State Machine (FSM). The execution of actions triggers a change in the actor's state, and the FSM acts as an additional element in determining which action will be executed next. In this example, the FSM alternates between executing the two available actions.

The RVC-CAL programming language is known for its high level of abstraction, making it platform-independent. By utilizing a high level representation of the program's execution, it is possible to create optimized, low-level code for various parallel architectures and platforms. The article utilizes the Open RVC-CAL Compiler (Orcc) [29, 30], for its compilation process. It's worth mentioning that RVC-CAL compilation is also supported by other open-source compilers such as Caltoopia [31, 32], Tÿcho [33], Cal2Many [34, 35], DAL [36] or StreamBlocks [37].

2.2 Execution Modeling

Dataflow programs can be effectively partitioned and mapped, resulting in correct executions without the need for software rewriting. However, this approach presents its own challenges in terms of identifying the most efficient



(a) An example with five actors (i.e. *Prod*, *CopyTokenA*, *CopyTokenB*, *PingPong* and *Merger*).

```
actor Producer () ==> int O:
  uint counter := 0;

  p: action  ==> O:[counter]
  guard counter < 4
  do  counter := counter + 1; end
end
```

(b) Producer.cal

```
actor CopyTokens (String name) int I ==> int O:
  c: action  I:[val] ==> O:[val] end
end
```

(c) CopyTokens.cal

```
actor PingPong () int I ==> int O:

  pp1: action  I:[val] ==> O:[val]
  do println("PingPong[pp1]:" + val); end

  pp2: action  I:[val] ==> O:[-val]
  do println("PingPong[pp2]:" + val); end

  schedule fsm a_pp1:
    a_pp1(pp1) --> a_pp2;
    a_pp2(pp2) --> a_pp1;
  end
end
```

(d) PingPong.cal

```
actor Merger () int I1, int I2 ==> :
  uint counter := 0;

  m: action I1:[ v1 ], I2:[ v2 ] ==>
  do
    println("Merger("+counter+"):"+v1+";"+v2);
    counter := counter + 1;
  end
end
```

(e) Merger.cal

Figure 1 RVC-CAL program example: dataflow network topology and actors source code.

configurations for partitioning, mapping, and scheduling. Even for simple designs, the number of possible design points is so large that it would be impractical or infeasible to be done manually through trial and error. This necessitates the use of automatic and systematic methods for identifying and evaluating optimal design point configurations.

TURNUS [12, 13] is a design space exploration framework that has been developed by the authors of this work to address this purpose. The tool is based on a high-level abstract model of computation, generated by the dataflow network structure and the actor execution model, which

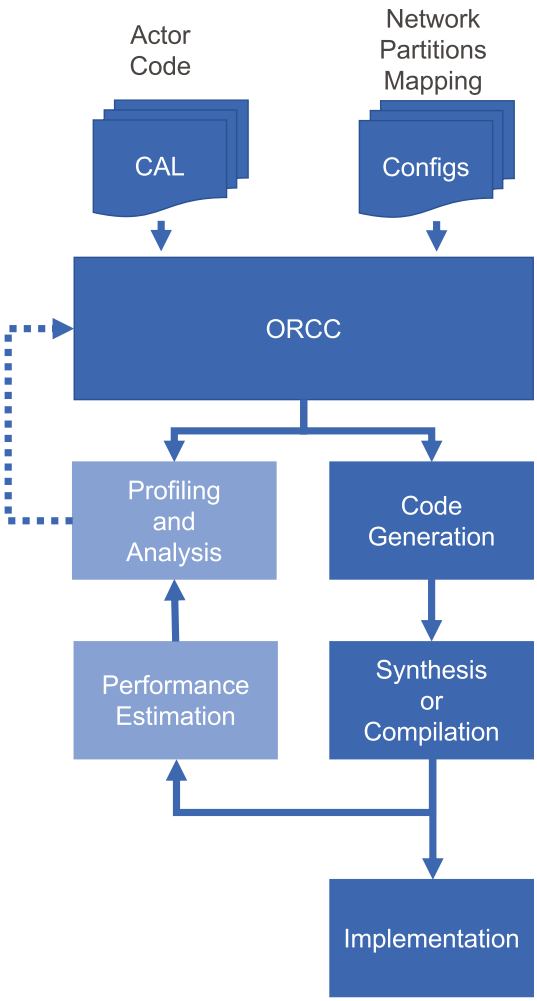


Figure 2 Design space Exploration - Tool flow for parameters optimization.

is further enriched by profiling measures of each atomic execution obtained on the specific heterogeneous processing platform. This allows for the exploration of the configuration design space and identification of efficient configurations. Figure 2 shows the design space exploration tool flow used when working with TURNUS to optimize dataflow program in the ORCC framework. First of all the CAL representation of the application program together with the different configurations files (network, partition, buffer sizes) are fed to the compiler. An optimization loop is initiated, which continues until the user is satisfied with the performance achieved. The optimization objectives can vary, such as critical path reduction, minimizing overall execution time, maximizing resource utilization, and so on. In the particular context of this paper, the loop runs for a fixed amount of time predetermined by the user at the start of the process. This loop start by the compiler generating a platform specific source code implementation for the targeted architecture containing performance evaluation code. Then,

this code is compiled or synthesized to an executable using compilers specific to the platform (here the Exelixi CUDA backend). Once executed and the performance metric from the platform extracted and the Execution trace graph (ETG) is labeled, the performance estimation of the application can then be evaluated with different configuration, without the necessity to compile and execute each time. The system then perform analysis of the ETG to evaluate different evaluation such as the critical path evaluation or the buffer dimensions to propose new configurations to be tested on the platform.

2.3 Heterogeneous CPU/GPU Platform Modeling

The field of automatic design space exploration for CPU/GPU heterogeneous systems remains an open research topic. Although GPU platforms are also programmable through software, the methods used for multi-core architectures cannot be directly applied to CPU/GPU heterogeneous systems due to numerous differences between the two. One such difference is the limited granularity of the APIs available. Even the advanced CUDA API, which provides fine-grained control over the hardware and open access to software context and data movement, does not offer access to scheduling and mapping of software kernels to available computational resources, as this task is delegated to the CUDA runtime. Another difference is clock-accurate profiling, which becomes more challenging with actors running on platforms with different frequencies. This issue is addressed in the methodology presented in Sections 5.1 and 5.2 through data normalization. Another crucial aspect is data movement, as on-chip cache coherency cannot handle this issue automatically. To transfer data between the CPU main memory and the GPU main memory, as well as the faster GPU shared memory and register-file, explicit data movement is required. Therefore, proper profiling of FIFO communication is essential, particularly for SIMD reads and writes. For a more detailed discussion of this issue, interested readers can refer to [23]. Additionally, compared to FPGA platform synthesis, software generation, compilation, and dataflow application loading on CPU/GPU heterogeneous platforms are relatively fast. This advantage is leveraged in the design point evaluation methodology presented in Sections 5.2 and 5.3 by enabling direct execution on the platform at each iteration.

3 Tabu Search

The Tabu Search (TS), introduced by Glover [38], is a metaheuristic optimization algorithm that can be applied to solve a wide range of optimization problems. TS involves iteratively exploring the solution space, making moves to solutions that are deemed "better" than the current one based on the given evaluation function. In the context of this study, the evaluation function refers to the minimal overall runtime of the application that needs to be optimized.

The core algorithm incorporates an explicit memory structure, known as the Tabu list, to prevent revisiting previously explored solutions and promote the exploration of the solution space, thereby avoiding getting stuck in local optima. This list, known as the Tabu list, can be adjusted in size and content to balance between exploration and exploitation in the search, and to ensure good overall performance in terms of quality of results, speed, robustness, simplicity and flexibility. It can be applied to problems with continuous or discrete variables, multiple objectives and is often used when the search space is large or the optimization problem is difficult to solve with other methods. In order to adapt TS to a specific problem, the representation of solutions, the neighborhood structure, the Tabu list structure, and the stopping criterion must be designed.

The Tabu meta-algorithm is applied in the context of partitioning actors across CPU and GPU partitions in order to find a solution that is defined as a map of actors and processing units (sequential CPU partitions or the parallel GPU partition). The number of actors is fixed by the dataflow application model, and the initial partition provides the upper bounds for the number of CPU and GPU partitions that can be used in a valid solution. It is not required that all partitions be used (meaning, no actor is be mapped to them), and allowing empty partitions as valid solutions allows for the discovery of optimal solutions that may not use all hardware resources if it results in better performance. It is important to note that not all actors are necessarily compatible with being run on the GPU, so the initial partition provided to the TS meta-algorithm should assign actors that can be mapped to either the CPU or GPU to the GPU partition, and actors that can only run on the CPU to a CPU partition.

To move from a valid solution to a neighboring one, the following types of moves are possible:

- REINSERT: move an actor from one partition to another.
- SWAP: exchanging the partitions of two actors that are currently assigned to different partitions. This can be done by moving each actor to the other actor's partition.

These basic moves can be combined in various ways to create different exploration strategies. Some examples of these strategies are discussed in Section 4.

TS can tuned using a couple of parameters a, b, t_{ab}, ϵ and T. When an actor, j, is transferred from one partition, ρ, to another, it is prohibited from returning to ρ for a certain number of iterations, t_{ab}. This number, t_{ab}, is a randomly chosen integer from the range $[a, b]$, and in previous experiments outlined in [18], the values of a and b were set to 5 and 15, respectively. Smaller values

of t_{ab} do not allow for escape from local optima, while larger values do not allow for intensification of the search around promising solutions. There are two other adjustable parameters in the TS algorithm: ϵ (the proportion of neighboring solutions explored during each iteration) and T (the time limit). If the time limit T is reached, the search will be immediately terminated and the best solution found thus far will be returned. With a fixed T, a small value of ϵ results in more iterations being performed but fewer neighbors being examined in each iteration, resulting in more diversification. A large value of ϵ, on the other hand, plays an intensification role (more solutions around the current one are explored). Finally, a small (large) value of t_{ab} strengthens the intensification (diversification) ability of the search, respectively.

Figure 3 shows the generic design flow for using TS for design space exploration in the context of this work. The optimization process begins with a user-provided initial mapping and execution trace, and generates a tentative new mapping solution. The initial mapping and execution trace provided by the user can have a big impact on the efficacy of the solution. Regarding the execution trace the input stimulus provided to the application during the trace generation need to be a good representation of the general input space so that the conclusion drawn would be applicable to the performance of the application in production. Regarding the initial mapping, since the size of the design space is too big to feasibly be fully explore, starting from an initially good partition is very important to lead to best possible results using this methodology. This new suggested solution is then evaluated to determine its performance and analyzed to provide the meta-algorithm with sufficient information to continue exploring the design space.

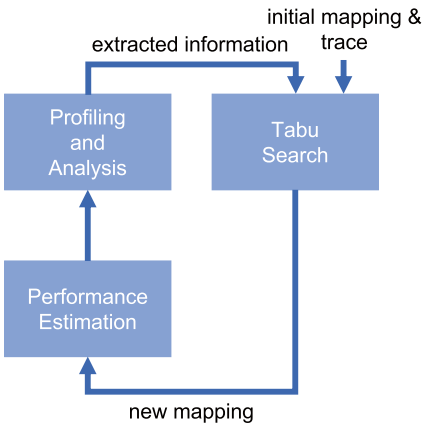


Figure 3 Generic design flow when doing design space exploration using tabu search.

4 Neighborhood Move Generator

Below is a list of neighborhood move generators used in this work during the Tabu search. They are mainly implemented using REINSERT moves. They can be used on their own during a design space exploration loop, or multiple of them can be combined.

- Balancing ($N^{(B)}$): move randomly an actor from the partition with the least idle time and move it to the partition with the most idle time. The partition idle time is defined as the time frame during which no actor are executing due to constraint such as dependency or others.
- Idle ($N^{(I)}$): move consecutively each actor whose idle time is greater than its processing time to the most idle partition that is different from the current partition of that actor. The idle time of an actor is defined as the time frame when it could execute according to the satisfaction of its firing rules, but it has to wait to be scheduled because another actor in the same partition is currently executing.
- Communication frequency ($N^{(CF)}$): if an actor has a higher communication frequency (i.e., more token transfers) with actors in a given partition than with the ones in its current partition, move the actor to that partition.
- Random ($N^{(R)}$): choose randomly an actor and move it to a different partition also randomly chosen.

5 Design Point Evaluations

Three different DSE optimization loops have been developed and used for comparison in this paper. Each of them uses a different design point evaluation methodology. These are the Static, Dynamic, and Measured methodologies and are presented in the following subsections.

5.1 Static Evaluation

In this version of the Tabu search design space exploration, the static heterogeneous estimation is used to evaluate the performance of the proposed new mapping configuration, as described in [22]. To evaluate performance, profiling weights are necessary for all possible combinations of FIFO buffers and actor platform assignments (i.e., CPU or GPU). Four distinct configurations were analyzed and are depicted in Fig. 4. The first and second configurations consist of all actors assigned to either CPUs or GPUs respectively, allowing for the evaluation of computation time of the action body and scheduling time on each platform. The other two configurations involve the use of a special *HostFifo* represented

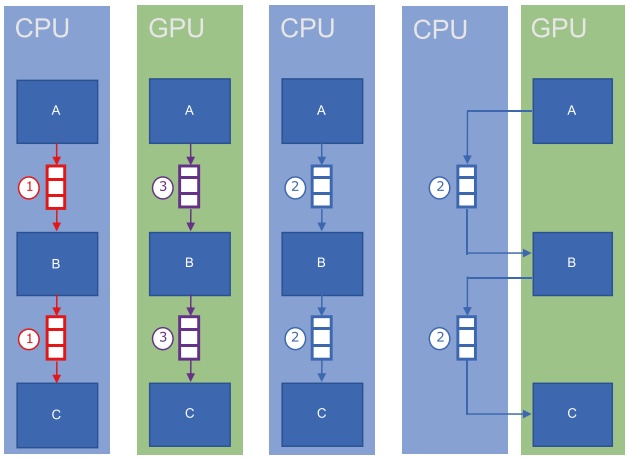


Figure 4 Illustration of the four static configurations required during profiling.

in blue with the number 2, allowing for the measurement of data transmission time for cross-platform communication (i.e., CPU-to-GPU or GPU-to-CPU). This approach considers all possible design points.

Figure 5 illustrates this methodology variant. Initially, the initial mapping is provided, along with the execution trace and a heterogeneous set of weights (generated as described previously) that have been generated using that initial mapping on the actual hardware platform using the profiled application generated by the Exelixi CUDA backend. This information is then used to suggest a new mapping, which is

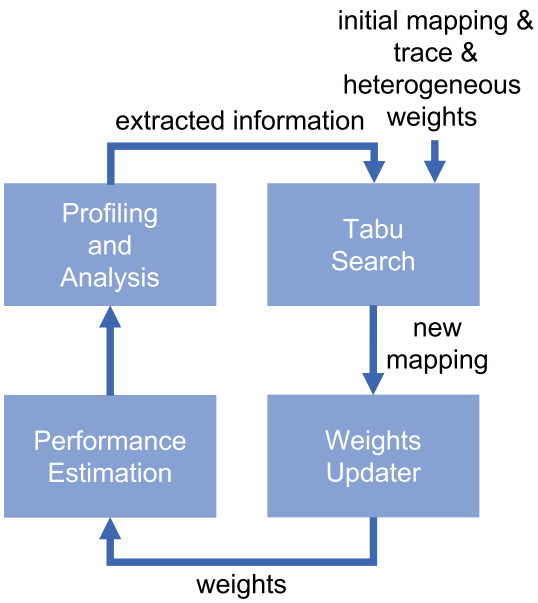


Figure 5 Design flow when doing design space exploration using tabu search and the static evaluation strategy.

fed to the 'Weights Updater'. The appropriate weights from the heterogeneous set of weights are selected based on the new mapping, meaning that the appropriate set of weights will be used depending on whether an actor is being mapped to a CPU or GPU. The same applies for the four sets of communication weights (CPU-CPU, CPU-GPU, GPU-CPU, GPU-GPU) based on the platform the actor it is communicating to is assigned. Finally, the updated weights are fed to the performance estimation engine, and the model analysis is conducted to extract information that helps with the neighboring move generation, after which the optimization loop can start again.

This methodology does not necessitate continuous access to the hardware platform as only the four static configurations need to be evaluated beforehand. During the design space exploration loop, only post-mortem access to the weights and execution trace is needed. This methodology places no restrictions on the move generators that can be used and any of the Neighborhood move generators defined in Section 4 or others can be used. However, it should be noted that this solution is not the most precise in terms of performance evaluation and may be relatively slow due to the use of the TURNUS performance estimation engine.

5.2 Dynamic Evaluation

In this version of the Tabu search design space exploration, the dynamic heterogeneous estimation described in [23] is used to evaluate the performance of the proposed new mapping configuration. In summary, the dynamic methodology leverages a feature from the Exelixi CUDA backend that generates a tailored version of the application for optimization. This software implementation packages both the instrumented CPU and GPU versions for each actor within the same binary. At runtime, the actual mapping configuration is provided and only the correct version is instantiated with the appropriate communication channel using the correct FIFO implementation. This allows for the generation of performance weights in the exact configuration, with the same resource contention and utilization as during the actual execution of the application program.

Figure 6 depicts this methodology variant. First, the initial mapping and the execution trace is provided by the user to the Tabu search meta-algorithm. In addition, a compiled version of the application using the profiled dynamic network methodology and generated by the Exelixi CUDA backend is made available to the optimization loop and is depicted in dark blue on the schema. During each iteration of the loop, the new mapping configuration is fed to the dynamic app, which is executed with this new mapping and generates new performance weights directly on the hardware platform. These new weights are then integrated into the execution trace and used by the TURNUS performance

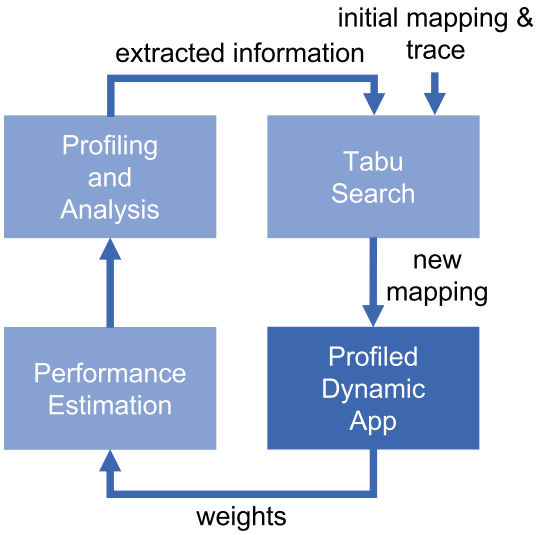


Figure 6 Design flow when doing design space exploration using tabu search and the dynamic evaluation strategy.

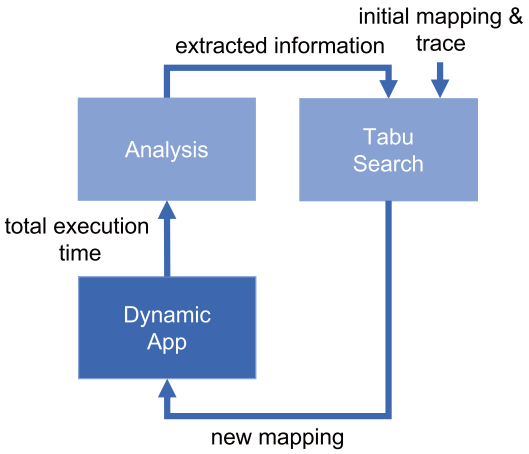


Figure 7 Design flow when doing design space exploration using tabu search and the measured evaluation strategy.

estimation engine to estimate the performance and provide insight for the analysis, allowing for the generation of neighboring moves.

This methodology requires continuous access to the hardware platform, as each new candidate mapping configuration needs to be evaluated by executing the profiled application. This methodology has no restrictions on the move generators that can be used; all the Neighborhood move generators defined in Section 4 and more can be used. This solution is relatively precise compared to the static methodology. However, it is the slowest due to the combination of full execution on the hardware platform with profiling weight generation and the TURNUS performance estimation engine.

5.3 Measured Evaluation

In this version of the Tabu search design space exploration, no estimation methodology is used, instead the performance are directly measure using the actual application on the hardware platform. Figure 7 depicts this methodology variant. First, the initial mapping and the execution trace is provided by the user to the Tabu search meta-algorithm. In addition, a compiled version of the application using the dynamic network methodology and generated by the Exelixi CUDA backend is made available to the optimization loop and is depicted in dark blue on the schema. During each iteration of the loop, the new mapping configuration is fed to the dynamic app, which is executed with this new mapping directly on the hardware platform and the total execution time is measured. This information is then used directly to generate neighboring moves.

This methodology requires continuous access to the hardware platform, as each new candidate mapping configuration needs to be evaluated by executing the dynamic application. This solution is the most precise, as the evaluation takes place in a setting that is quasi-identical to the 'production execution' and is also the fastest, as no post-mortem analysis is required and the performance evaluation time is as fast as the final execution time. However, not all the Neighborhood move generators defined in Section 4 can be used. In particular, generators that use insight from the performance estimation engine to suggest a new mapping cannot be used. This is the case for the Balancing and Idle generators, which use the Idle metrics provided by the analysis.

6 Results

A simple program example has been selected to evaluate the methodologies introduced in this work. The application is based on well-known code and can be freely downloaded from the open-source *Orc-apps* Github project [39]. The goal is to determine how the different neighborhood move generators and design point evaluations affect the results obtained when exploring the design space using Tabu search. This example also illustrates how the method can be applied to optimize any particular application.

6.1 Experimental Setup

This methodology can be applied to any heterogeneous platform that is compatible with the CUDA computing platform and programming interface, whether it is an embedded platform like NVIDIA Jetson or a data center platform such as NVIDIA Grace. The experimental results were performed using the Nvidia GeForce GTX 1660 SUPER, which has 6 Gigabytes of VRAM, as the GPU and the Intel Skylake

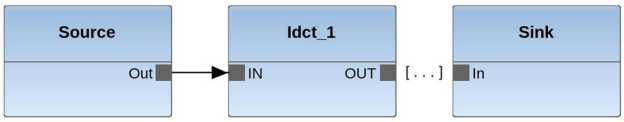


Figure 8 Illustration of the test RVC-CAL IDCT dataflow network.

i5-6600, which has 16 Gigabytes of DDR4 memory, as the CPU. To ensure reproducibility, the clock speed of the GPU was kept at 1.8GHz and the clock speed of the CPU at 2.9GHz.

6.2 IDCT Example

The synthetic application used to demonstrate this technique is derived from the JPEG decoder available on the Orc-app Github project [39]. Figure 8 illustrates the top-level representation of the dataflow network. It consists of a Source and a Sink actor, which are responsible for feeding and pulling data to and from the network, respectively. Additionally, the network comprises a chain of five IDCT actors that compute the two-dimensional inverse discrete cosine transform on their input, this algorithm is used in video and image decoding [40, 41]. The IDCT operates on an 8x8 matrix containing frequency domain information to convert it into a block of image data. To execute the two-dimensional IDCT, it is broken down into a combination of two separate one-dimensional IDCTs through a process known as row-column decomposition. This chain of actors is a representation of an intensive computation that could be found in a real-world application.

All results have been generated allowing the solution to use a single CPU core and the parallel GPU partition. The initial partition for all DSE runs is set to the sequential CPU partition, with all actors mapped to it. The value T has been set to 20 minutes as in the related work [18], meaning the algorithm halt after 20 minutes without discovering a better-performing configuration.

Table 1 presents the summary of experimental results obtained using three evaluation methods (Static, Dynamic, Measured) and four neighboring move generators (Balancing, Communication Frequency, Idle, Random). Additionally, two composite generators were used: Joint, which combines moves from the four move generators, and Prob, which selects moves from one of the four generators with a 25% probability initially, increasing over time if the generator yields better configurations. The third column in the table displays the number of configurations evaluated, while the fourth column shows the number of empty iterations, or the number of iterations where the move generator could not suggest a neighboring configuration from the current one. The "Time to best" column indicates the amount of time, in minutes, required to find the best-performing configuration among those tested. The "Best Perf" column shows the overall time needed, in seconds, to run the program using the best configuration found by the Tabu search algorithm. Table 2, shows the mapping of the bests/worst partitions between CPU and GPU and there corresponding performances.

The results show that the Measured evaluation methodology was able to find the best time for mapping all actors to the GPU. The Joint Generator was found to be the most efficient, as it found the best-performing configuration in under 3

Table 1 Summary table of results with three evaluation methods (Static, Dynamic, Measured), and four neighboring move generators

Evaluation	Generator	# Iterations	# Empty Iterations	Time to Best [min]	Best Perf [sec]
Static	Balancing	2	0	x	1.35
	Comm Freq	1	x	x	1.35
	Idle	2	0	x	1.35
	Random	2	0	x	1.35
	Joint	2	0	x	1.35
	Prob	2	0	x	1.35
Dynamic	Balancing	2	0	20	5.6
	Comm Freq	1	x	x	1.35
	Idle	2	0	21	4.86
	Random	2	0	39	2.64
	Joint	2	0	22	4.66
	Prob	2	0	23	4.94
Measured	Comm Freq	1	x	x	1.35
	Random	212	77	5	1.17
	Joint	205	71	3	1.17
	Prob	3030426	3030376	5	1.17

Table 2 Mapping of the bests/worst partitions between CPU and GPU and there corresponding performances

	Partition		Perf [sec]
	CPU	GPU	
Best Measured	x	All actors	1.17
Best Dynamic/Static	All actors	x	1.35
Worst	All other actors	Idct_5	5.6

minutes with the lowest number of total tries (205). In contrast, the Static and Dynamic methodologies were found to be too slow to generate meaningful results within the given time limit, with one simulation taking around 18 minutes. This approach would likely require increasing significantly the T value. However, the insights offered by these two other methods are still valuable as shown in the related work cited in the introduction of this paper. This showed that to improve the results, it would be beneficial to investigate, in further research, combining the speed of the Measured evaluation methodology with the insights gained from the Static and Dynamic methods.

It should be noted that the limited information available for the Communication Frequency move generator is due to the fact that, starting from the initial configuration, it is unable to generate any neighboring configurations as the mapping is already balanced. However, this does not mean that this move generator is not useful.

It is also interesting to note that the Measured Joint method outperforms the Measured Random and Comm Freq methods. The Measured Joint method combines the benefits of different generators, which leads to a faster and more efficient search for good partitions. The Random generator is good for exploring the search space and avoiding getting stuck in a local minimum. On the other hand, other generators may converge faster to a good local partition. By combining these generators in the Measured Joint method, the algorithm can explore the search space efficiently and converge quickly to a good partition. Overall, the Measured Joint method seems to strike a good balance between exploration and exploitation of the search space, leading to improved performance compared to other methods.

7 Conclusions

The presented work describes an effective design space exploration approach for partitioning and mapping dataflow programs on CPU-GPU heterogeneous systems. To do so it used the Tabu search meta-heuristic optimization algorithm with several implemented exploration strategies to find efficient design point. Through experimentation on a dataflow program, the effectiveness of this approach in addressing the NP-complete problem of partitioning and mapping dataflow components onto heterogeneous processing elements was demonstrated. The efficiency and quality of the different move generators and design point evaluation methodology have been compared.

As a potential future research direction, the design space methodology could be expanded by incorporating SIMD parameters as part of the configuration input for each actor. The specific number of SIMD cores assigned to a specific action of a specific actor can significantly impact performance. For the dynamic and measured method, this would require defining a proper input method or API to specify the number of SIMD cores used at instantiation time. For the static methodology, it would involve profiling actors with a range of different SIMD numbers and providing all of these weights as input at the beginning of the DSE.

Funding Open access funding provided by EPFL Lausanne.

Data Availability The data and results obtained during the course of this work are presented within this publication. The software utilized to acquire said data is accessible through the following references: [12, 30, 39, 42].

References

1. Microsoft ARM. https://www.microsoft.com/en-us/surface/business/surface-pro-x/processor. Online, Accessed January 2023.
2. Nvidia grace. https://nvidianews.nvidia.com/news/nvidia-introduces-grace-cpu-superchip. Online, Accessed January 2023.
3. Jetson AGX Xavier. https://developer.nvidia.com/embedded/jetson-agx-xavier. Online, Accessed May 2020.
4. NVIDIA Jetson Nano is a tiny AI computer for $99 and up. https://liliputing.com/nvidia-jetson-nano-is-a-tiny-ai-computer-for-99-and-up. online, Accessed January 2023.
5. Apple M1. https://www.apple.com/newsroom/2020/11/apple-unleashes-m1. Online, Accessed January 2023.
6. Michalska, M., Casale-Brunet, S., Bezati, E., & Mattavelli, M. (2017). High-precision performance estimation for the design space exploration of dynamic dataflow programs. *IEEE Transactions on Multi-Scale Computing Systems, 4*(2), 127–140.
7. Goens, A., Khasanov, R., Castrillon, J., Hähnel, M., Smejkal T., & Härtig, H. (2017). 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*, pp. 11–20.

8. I. 23001-4:2011. (2011). Information technology - MPEG systems technologies - Part 4: Codec configuration representation.
9. Ab Rahman, A. A. -H., Casale Brunet, S., Alberti, C., & Mattavelli, M. (2013). Dataflow program analysis and refactoring techniques for design space exploration: Mpeg-4 avc/h.264 decoder implementation case study. *In 2013 Conference on Design and Architectures for Signal and Image Processing*, pp. 63–70.
10. de Saint Jorre, D., Alberti, C., Mattavelli, M., & Casale-Brunet, S. (2014). Exploring mpeg hevc decoder parallelism for the efficient porting onto many-core platforms. *In 2014 IEEE International Conference on Image Processing (ICIP)*, pp. 2115–2119.
11. Jerbi, K., Renzi, D., de Saint-Jorre, D., Yviquel, H., Raulet, M., Alberti, C., & Mattavelli, M. (2014). Development and optimization of high level dataflow programs: the HEVC decoder design case. *In 48th Asilomar Conference on Signals, Systems and Computers, Pacific Grove, USA*.
12. TURNUS source code repositoy. http://github.com/turnus. Online, Accessed January 2023.
13. Casale-Brunet, S. (2015). Analysis and optimization of dynamic dataflow programs, Ph.D. dissertation, EPFL STI, Lausanne.
14. Michalska, M., Casale-Brunet, S., Bezati, E., Mattavelli, M., & Janneck, J. (2016). Trace-based manycore partitioning of stream-processing applications. *In 2016 50th Asilomar Conference on Signals, Systems and Computers*, pp. 422–426.
15. Casale-Brunet, S., Bezati, E., & Mattavelli, M. (2017). Design space exploration of dataflow-based smith-waterman fpga implementations, in. *IEEE International Workshop on Signal Processing Systems (SiPS), 2017*, 1–6.
16. Brunet, S. C., Bezati, E., Bloch, A., & Mattavelli, M. (2017). Profiling of dynamic dataflow programs on mpsoc multi-core architectures. *In 2017 51st Asilomar Conference on Signals, Systems, and Computers*, pp. 504–508.
17. Bezati, E., Brunet, S. C., & Mattavelli, M. (2017). Execution trace graph based interface synthesis of signal processing dataflow programs for heterogeneous mpsocs. *In 2017 51st Asilomar Conference on Signals, Systems, and Computers*, pp. 494–498.
18. Michalska, M., Zufferey, N., & Mattavelli, M. (2016). Tabu search for partitioning dynamic dataflow programs. *Procedia Computer Science, 80*, 1577–1588.
19. Michalska, M. M. (2017). Systematic design space exploration of dynamic dataflow programs for multi-core platforms, Ph.D. dissertation, EPFL STI, Lausanne.
20. Bloch, A., Bezati, E., & Mattavelli, M. (2020). Programming Heterogeneous CPU-GPU Systems by High-Level Dataflow Synthesis. *In 2020 IEEE Workshop on Signal Processing Systems (SiPS). IEEE*, 1–6.
21. Bloch, A., Casale-Brunet, S., & Mattavelli, M. (2021). Methodologies for synthesizing and analyzing dynamic dataflow programs in heterogeneous systems for edge computing. *IEEE Open Journal of Circuits and Systems, 2*, 769–781.
22. Bloch, A., Brunet, S. C., & Mattavelli, M. (2021). Performance estimation of high-level dataflow program on heterogeneous platforms. *In 2021 IEEE 14th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC). IEEE*, 69–76.
23. Bloch, A., Casale-Brunet, S., & Mattavelli, M. (2022). Performance estimation of high-level dataflow program on heterogeneous platforms by dynamic network execution. *Journal of Low Power Electronics and Applications, 12*(3), 36.
24. Lee, E., & Parks, T. (1995) Dataflow Process Networks. *In Proceedings of the IEEE*, pp. 773–799.
25. Casale-Brunet, S. (2015). Analysis and optimization of dynamic dataflow programs, Ph.D. dissertation.
26. Johnston, W., Hanna, J., & Millar, R. (2004). Advances in dataflow programming languages. *ACM Computing Surveys (CSUR), 36*(1), 1–34.
27. Feo, J. T., Cann, D. C., & Oldehoeft, R. R. (1990). A report on the sisal language project. *Journal of Parallel and Distributed Computing, 10*(4), 349–366.
28. Eker, J., Janneck, J., Lee, E., Liu, J., Liu, X., Ludvig, J., Neuendorffer, S., Sachs, S., & Xiong, Y. (2003). Taming heterogeneity - the ptolemy approach. *Proceedings of the IEEE, 91*(1), 127–144.
29. Yviquel, H., Lorence, A., Jerbi, K., Cocherel, G., Sanchez, A., & Raulet, M. (2013). Orcc: Multimedia Development Made Easy. *In Proceedings of the 21st ACM International Conference on Multimedia, ser. MM '13. ACM*, pp. 863–866.
30. Orcc source code repositoy. http://github.com/orcc/orcc. Online, Accessed January 2023.
31. Siyoum, F., Geilen, M., Eker, J., von Platen, C., & Corporaal, H. (2013). Automated extraction of scenario sequences from disciplined dataflow networks. *In 2013 Eleventh ACM/IEEE International Conference on Formal Methods and Models for Codesign (MEMOCODE 2013). IEEE*, pp. 47–56.
32. Caltoopia. https://github.com/Caltoopia. Online, Accessed January 2023.
33. Cedersjö, G., & Janneck, J. W. (2019). Tÿcho: a framework for compiling stream programs. *ACM Transactions on Embedded Computing Systems (TECS), 18*(6), 1–25.
34. Gebrewahid, E. (2017). Tools to compile dataflow programs for manycores, Ph.D. dissertation, Halmstad University Press.
35. Savas, S., Ul-Abdin, Z., & Nordström, T. (2020). A framework to generate domain-specific manycore architectures from dataflow programs. *Microprocessors and microsystems, 72*, 102908.
36. Boutellier, J., & Ghazi, A. (2015). Multicore execution of dynamic dataflow programs on the distributed application layer. *In IEEE 2015 global conference on signal and information processing (GlobalSIP). IEEE*, 893–897.
37. Bezati, E., Emami, M., Janneck, J., & Larus, J. (2021). Streamblocks: A compiler for heterogeneous dataflow computing (technical report), arXiv preprint arXiv:2107.09333
38. Glover, F. (1989). Tabu search-part i. *ORSA Journal on computing, 1*(3), 190–206.
39. Orcc-Apps source code repositoy. https://github.com/orcc/orc-apps. Online, Accessed January 2023.
40. Wallace, G. (1992). The jpeg still picture compression standard. *IEEE Transactions on Consumer Electronics, 38*(1), xviii–xxxiv.
41. Eric, H. (1992). Jpeg file interchange format version 1.02. http://www.w3.org/Graphics/JPEG/jfif3.pdf
42. CAL Exelixi Backends source code repositoy. https://bitbucket.org/exelixi/exelixi-backends. Online, Accessed January 2023.