CORE

Metadata, citation and similar papers at core.ac.uk

Provided by Edinburgh Research Explorer

THE UNIVERSITY of EDINBURGH

Edinburgh Research Explorer

Diplomat: Mapping of multi-kernel applications using a static dataflow abstraction

Citation for published version:
Bodin, B, Nardi, L, Kelly, PHJ & O'Boyle, MFP 2016, Diplomat: Mapping of multi-kernel applications using a static dataflow abstraction. in 2016 IEEE 24th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS) . IEEE, London, UK, pp. 241-250, 2016 IEEE 24th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, London, United Kingdom, 19/09/16. DOI: 10.1109/MASCOTS.2016.35

Digital Object Identifier (DOI):
10.1109/MASCOTS.2016.35

Link:
Link to publication record in Edinburgh Research Explorer

Document Version:
Peer reviewed version

Published In:
2016 IEEE 24th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)

General rights
Copyright for the publications made accessible via the Edinburgh Research Explorer is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
The University of Edinburgh has made every reasonable effort to ensure that Edinburgh Research Explorer content complies with UK legislation. If you believe that the public display of this file breaches copyright please contact openaccess@ed.ac.uk providing details, and we will remove access to the work immediately and investigate your claim.

OPEN ACCESS

Download date: 05. Apr. 2019

Diplomat: Mapping of Multi-kernel Applications Using a Static Dataflow Abstraction

Bruno Bodin
The University of Edinburgh,
Edinburgh, United Kingdom
Email: bbodin@inf.ed.ac.uk

Luigi Nardi & Paul H. J. Kelly
Imperial College London,
London, United Kingdom
Email: {l.nardi,p.kelly}@imperial.ac.uk

Michael F. P. O'Boyle
The University of Edinburgh,
Edinburgh, United Kingdom
Email: mob@inf.ed.ac.uk

Abstract—**In this paper we propose a novel approach to heterogeneous embedded systems programmability using a task-graph based framework called Diplomat. Diplomat is a task-graph framework that exploits the potential of static dataflow modeling and analysis to deliver performance estimation and CPU/GPU mapping. An application has to be specified once, and then the framework can automatically propose good mappings. We evaluate Diplomat with a computer vision application on two embedded platforms. Using the Diplomat generation we observed a 16% performance improvement on average and up to a 30% improvement over the best existing hand-coded implementation.**

I. Introduction

Mobile and embedded systems on chip (SoC) integrate multiple computing resources. They contain multiple CPU cores, a GPU and in some cases also a DSP. Such complex architectures have the potential for good performances but are hard to program. They have a *mapping* problem, *i.e.* how to associate computational tasks to hardware resources in order to meet the application's response time, throughput objectives and power budget. OpenCL [1] has emerged as a standard to program heterogeneous systems and allows program portability, *i.e.* the ability of a single program to be executed on different devices. The OpenCL programming interface allows a programmer to express a task under the form of a function (called kernel in OpenCL) and to execute it on compatible resources. However, developing applications using such systems and achieving acceptable performance is a non-trivial task. Even if a programmer has perfect knowledge of the target architecture and a deep understanding of OpenCL, performance portability remains a challenge.

Meanwhile, the emergence of computationally intensive mobile vision applications makes the use of embedded heterogeneous platforms increasingly important. Mobile vision applications have performance constraints and need efficient mappings. They are excellent candidates to evaluate new mapping techniques.

Prior work on mapping exists. While most work deals with single-kernel applications, recent research focuses on multi-kernel applications. These represent more realistic real-world applications. Static dataflow frameworks such as *StreamIt* [2] provide languages to statically express multi-kernel applications and communication channels with specific constraints on the amount of transferred data through the channels. They provide an automatic mapping, yet these models are overly restrictive. As an example, many vision applications include dynamic behavior, such as early exit conditions, which are not supported by those models. Tools such as Qualcomm *Symphony* [3] (previously known as MARE) are more expressive. Symphony lets a programmer represent an application using a task-graph, a more general representation of the dataflow model where communications are no longer static. However, Symphony tasks are dynamically defined and mapping decisions are performed manually. Ideally we would like a framework that is both flexible and able to provide high performance mappings automatically.

In this paper, we address the mapping problem of multi-kernel applications. For this purpose we introduce the *Diplomat* framework, a task-graph model embedded in the Python language and combined with off-line performance characterization to compute relevant mappings. It mainly targets streaming applications, which are common in computer vision. In Diplomat, an application is defined as a task-graph. In contrast with static dataflow modeling, the tasks and communication channels are not necessarily known at compile-time. Diplomat automatically collects performance profiling information to evaluate tasks' workload; then it uses a static dataflow model to evaluate performance and explores different mappings for a given platform. The library generates C++ code, which can then be integrated into an embedded system's firmware. This code is derived automatically from the task-graph, and uses a runtime library to implement the necessary communication and buffering between tasks. We show that a class of computer vision applications can be abstracted to a static dataflow representation via the Diplomat task-graph. We observe up to 30% speed improvement over the best existing hand-written implementation.

The contributions of this paper are:

- A task-graph model, suitable for dynamic streaming applications, and which can be easily abstracted into a Synchronous Dataflow (SDF) model.
- A framework which implements this abstraction and performs workload profiling together with static analysis to find a mapping that achieves best performance on heterogeneous platforms.
- We show that our approach adapts automatically to different architectures for truly performance-portable software.

II. Motivation

To illustrate choices that have to be made while designing an embedded system application on modern devices, we will consider the SLAMBench benchmarking framework [4] and in particular the KFusion benchmark. KFusion performs real-time localization and scene reconstruction for a camera moving through an unknown environment, *i.e.* it estimates the pose of a depth camera while building a highly detailed 3D model of the environment. This application is composed of more than 10 different communicating tasks (refer to Figure 1). To implement this application and to reach acceptable performance, a designer must optimize its algorithm while considering a good mapping at the same time. Thus, in Figure 2 we present several possible mapping solutions.

Running the KFusion benchmark on just one 2GHz ARM A15 core of an embedded SoC (Samsung Exynos 5422, see Table II), we observe low performance on a single CPU core, less than 2.5 Frames Per Second (FPS). By exploiting data parallelism (using the KFusion OpenMP implementation), the speed-up is 2.5x. We can also take advantage of the task parallelism through a task-graph runtime (namely Symphony [3]), and again, we improve performance by a further 20%. Thus, with these optimizations and by only using the multi-core CPU, this application can reach 7.4 FPS. Alternatively, we can use the GPU implementation (with OpenCL), and yields a 2x speed-up comparing with the best CPU version, which is 14.9 FPS.

It is evidently appealing to use both CPU and GPU at the same time and to combine those speed-ups through a more complex mapping, but implementing such behavior is not easy. The communication between CPU and GPU becomes a potential bottleneck, and the resulting performance might be worse.

For single-kernel applications, various solutions have been proposed to handle the mapping problem. But these are not well suited to multi-kernel applications. As an example, following a speed-up based mapping methodology [5] we obtain a 16% slow-down compared to the OpenCL version. A partitioning technique similar to the one presented in [6] also results in a 16% slow-down.

In contrast, using our approach, we are able to generate a mapping that reaches 17.2 FPS, which is a 15% speed-up compared to the OpenCL version. This example shows that mapping using dataflow analysis tools can outperform existing techniques.

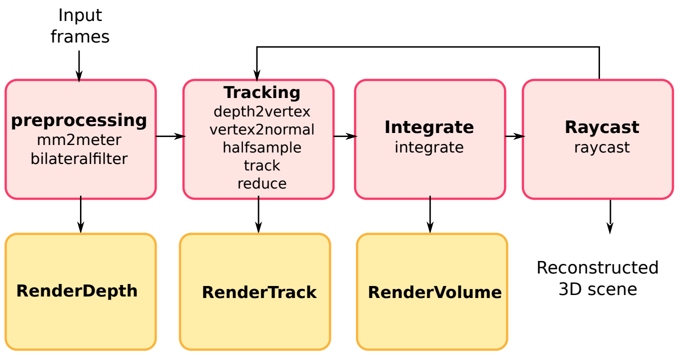


Fig. 1: The KFusion algorithm is composed of 7 main tasks, a total of 12 different OpenCL kernels.

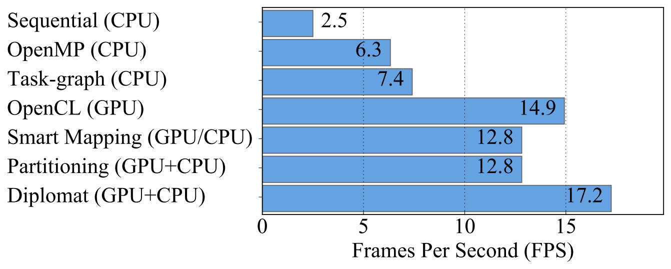


Fig. 2: Performance comparison of different possible mappings for KFusion (higher FPS is better).

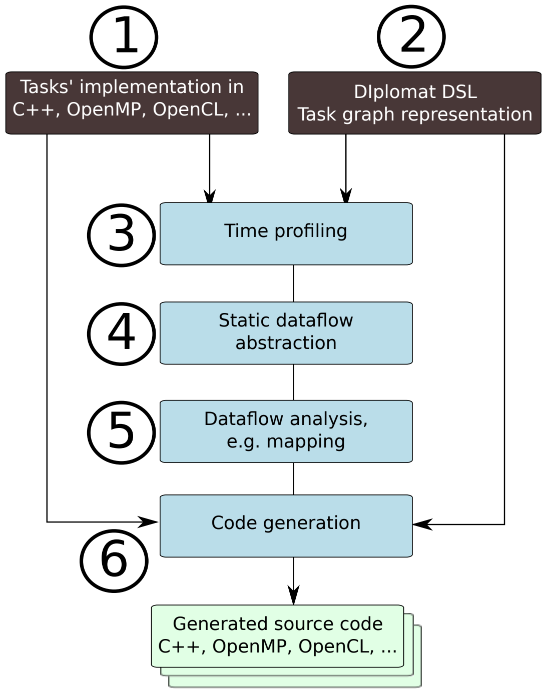

1
2
Tasks' implementation in C++, OpenMP, OpenCL, ...
DIplomat DSL Task graph representation
3
Time profiling
4
Static dataflow abstraction
5
Dataflow analysis, e.g. mapping
6
Code generation
Generated source code C++, OpenMP, OpenCL, ...

Fig. 3: Overview of the Diplomat framework. The user provides (1) the task implementations in various languages and (2) the dependencies between the tasks. Then in (3) Diplomat performs timing analysis on the target platform and in (4) abstracts the task-graph as a static dataflow model. Finally, a dataflow model analysis step is performed in (5) and in (6) the Diplomat compiler performs the code generation.

III. Diplomat Framework

A. *Framework Overview*

The Diplomat framework combines task-graph modeling and static dataflow analysis to perform the mapping of streaming applications. An overview of the framework is given in Figure 3.

1) Front-end: The Diplomat front-end allows the framework to gather fundamental information about the application: the different possible implementations of the tasks, their expected input and output data sizes, and the existing data dependencies between each of them.

2) Static analysis: At compile-time, the framework performs static analysis. In order to benefit from existing dataflow analysis techniques, the initial task-graph needs to be turned into a dataflow model. As the dataflow graph will not be used to generate the code, a representation of the application does not need to be precise. But it needs to be accurate enough to obtain good performance estimations. Diplomat performs the following steps. First, the initial task-graph is abstracted into a static dataflow formalism. This includes a timing profiling step to estimate task durations and communication delays. Then, by using existing static analysis techniques [7], [8], a throughput evaluation and a mapping of the application are performed.

3) Code generation: Once a potential mapping has been selected, a functional C++ code is automatically generated. This produced implementation takes advantage of task-parallelism and data-parallelism. It can use OpenMP, OpenCL and it may apply partitioning between CPU and GPU when it is beneficial.

After this short overview, we now discuss the Diplomat framework in further detail.

B. Diplomat front-end

Diplomat is composed of three classes, namely `Kernel`, `Buffer` and `Loop`. These classes enable the definition of an application as a task-graph, such as in Figure 4. Using the Diplomat framework implies two requirements: *a*) to identify the tasks that compose an application; *b*) to provide different implementations of these tasks for CPU and GPU hardware resources, *i.e.* C++, OpenMP and OpenCL. Once these requirements are fulfilled, and the task-graph expressed, all the remaining work is automated. We now define the three classes of Diplomat.

1) Kernel: The `Kernel` class defines tasks. First, it specifies the name of the C++ function or of the equivalent OpenCL kernel that this task is referring to. In order to respect data dependencies during the code generation, it also specifies input and output buffers. Then, `Argument` is used to specify arguments needed by the corresponding function. Additionally, the `dimensions` and `globalworksize` are both OpenCL specific arguments, required for OpenCL code generation.

```
Kernel(name             = "functionName",
       input            = [inBuffer],
       output           = [outBuffer],
       dimensions       = 2,
       globalworksize   = "computationSize",
       params           = [
           Argument(name   = "outBuffer",
                    buffer = outBuffer,
                    ptr    = True),
           Argument(name   = "inBuffer",
                    buffer = inBuffer,
                    ptr    = True)])
```

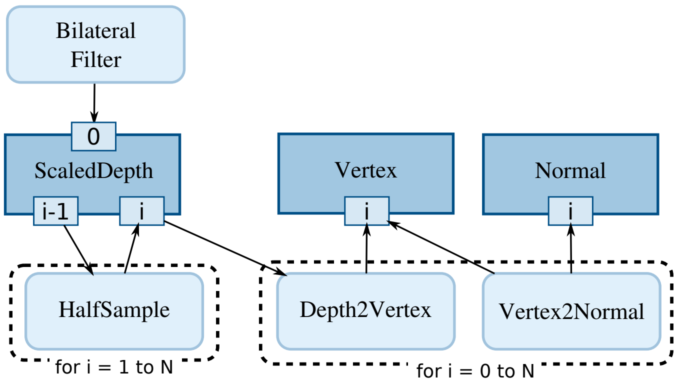


Fig. 4: The Diplomat model is a composition of kernels, buffers, and loops. In this figure kernels are represented with rounded boxes, buffers with straight boxes and loops with dotted boxes. This graph is related to a subgraph issued from the KFusion application visible on Figure 7.

2) Buffer: The `Buffer` class represents memory locations as logical regions. To make the code generation possible, data type and data size of these memory locations are required.

```
Buffer(name = "floatDepth",
       ptr  = True,
       type = "float",
       size = "n")
```

In this example, `floatDepth` is a `float` pointer. In the previous `Kernel` example, the variables `inBuffer` and `outBuffer` were referring to `Buffer` instances.

3) Loop: In the Diplomat task-graph representation, a task can only be executed once. Because this requirement is restrictive, we introduce the `Loop` operator, and two more concepts, the *buffer instances* and the *buffer initialization values*.

A `Loop` expresses the replication of `Kernel`. Let us consider a classical loop of $N - 1$ iterations of a function (e.g. `halfSample`):

```
for(int i = 1 ; i < N ; i++){
  halfSample(scaledD[i],scaledD[i-1])
}
```

We can use the *Loop* operator to express it :

```
HFLoop = Loop(variable ="i",
              from     ="1",
              to       ="N-1")
halfSample = Kernel(...,
             input  = [ScaledD.sub("i-1")],
             output = [ScaledD.sub("i")],
             loop   = HFLoop)
```

After instantiating a `Loop` (*i.e.* `HFLoop`), we can associate a `Kernel` to this loop using the `loop` argument. The arguments `variable`, `from` and `to` defined in the `Loop` class are used to enumerate the replication of this `Kernel`.

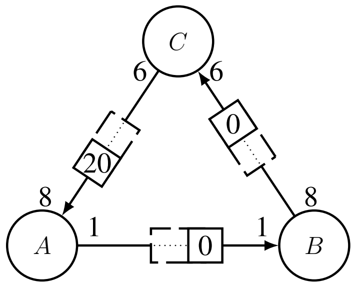


Fig. 5: An example of SDF.

a) Buffer instances: To easily maintain determinism, a buffer can only have one writer, *i.e.* it is only defined once as an output in `Kernel` arguments. Thus, to reuse them through a `Loop`, we introduce the *buffer instances*. A `Buffer` can be composed of logical instances accessible with the `sub(id)` function. The `id` argument of the `sub` function specify an identifier for each instance. An example of buffer instances is given in the previous `Loop` example.

b) Buffer initialization values: A `Buffer` can be initialized. This means that its first instance (identified by `sub(0)`) contains an initial value.

```
Buffer(name      = "pose",
       pointer   = False,
       type      = "Matrix4",
       initvalue = "*oldPose",
       init      = True
       size      = "sizeof(Matrix4)")
```

In this example, `pose` is a `Buffer` with an initialized value set to `*oldPose`. It is important to understand that a `Kernel` cannot specify this `Buffer` as an output unless using an instance different than `sub(0)`. The advantage of this initial value is that it can be used with `Loop` to express data reuse; this can later result in a deadlock-free dependency cycle in the obtained dataflow abstraction.

C. Static analysis

In order to determine potentially good mappings, Diplomat relies on dataflow static analysis. It consists of two main stages. First, Diplomat abstracts the task-graph into a static dataflow; this includes automatic timing profiling. Second, it combines throughput evaluation and mapping exploration to select high-performance candidates.

1) Dataflow modeling and static analysis: Applications are described as directed graphs where nodes represent operations, or functions, and arcs represent data dependencies. One of the most common models of this kind is the Kahn Process Network (KPN) [9]. In a KPN each arc corresponds to a FIFO buffer with only one writer and one reader. Two basic rules have to be respected: *a*) read operations are blocking, which means that when a task is reading an empty buffer it has to wait until new data arrives; *b*) tasks are *non-reentrant*, which means, a task cannot be executed if a previous execution of the same task is still in progress. Following these rules, a KPN is deterministic, *i.e.* whatever the execution order of the KPN tasks is, the overall execution result remains the same. Yet, the KPN does not provide enough information to predict a valid execution schedule of an application at compile-time.

To overcome this restriction, static dataflow models can be used. One popular static model is the Synchronous Dataflow (SDF) model [10]. The SDF is a dataflow model where each execution produces and consumes a known constant amount of data. Even though the SDF has reduced expressivity compared to other models, it is an appropriate model for the Diplomat analysis module that targets streaming applications. This kind of application, even with data-dependences and dynamic behaviors, keeps following the same execution pattern. When this pattern is successfully captured by the static dataflow model, efficient mapping candidates can be found.

SDFs describe an application as a directed graph in which:

- Each node t is a task.
- Each arc $b = (t, t')$ is an unbounded FIFO buffer which connects a task t to t'.
- Buffers contain data (called tokens) and the initial amount of tokens in a buffer is called its initial marking.

Tasks can be executed; then they consume (*resp.* produce) tokens in their inputs (*resp.* outputs). A task execution is only possible if there are enough tokens in its inputs. The SDF model also assumes that each task has a known execution duration. Figure 5 shows an SDF with three tasks A, B and C communicating through three buffers. The initial number of tokens for the buffer between C and A is 20. At each execution of the task A, 8 tokens will be consumed in b, and at each execution of the task C, 6 tokens will be produced.

2) Static abstraction: Static dataflow models are not expressive enough to represent dynamic applications. However, despite dynamic behavior, streaming applications often keep following constant execution patterns. This is an important observation that is at the core of the Diplomat language targeting streaming applications. This execution pattern regularity is caught by the front-end and abstracted to a SDF model.

To perform this abstraction, we limit our analysis to specific configurations of the application by fixing dynamic values (through the application arguments). This selection is done only once, by the designer of the application. In Section VIII we explore potential candidates to replace the SDF model in order to relax this constraint.

a) Model generation: The abstract SDF model is composed of tasks and buffers which respectively correspond to Diplomat `Kernels` and `Buffers`. The initial values in Diplomat will imply initial values in the SDF. The production and consumption of each task will be set to 1. As an example, with the task-graph in Figure 4, and by following this abstraction method, we obtain the SDF on Figure 6.

Once an application is modeled through this SDF model, a range of analysis tools can be used. However, to fully apply them, timings are required.

b) Timing profiling: The timing information is specific to each platform. It consists of the computation time of every

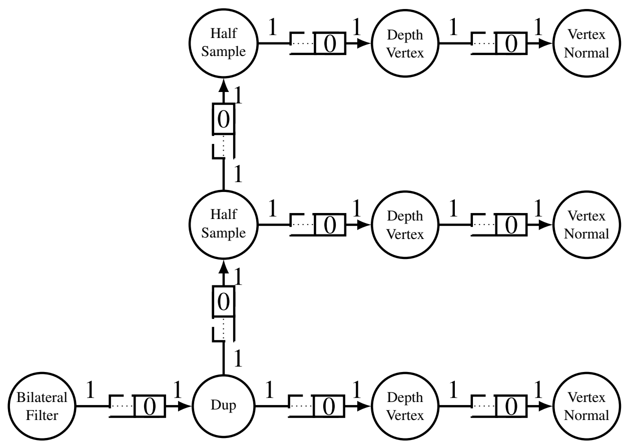


Fig. 6: A dataflow model generated by fixing dynamic values of this application ($level = 3$).

task on every resource, *i.e.* CPU-only and GPU-only measurements in C++, OpenMP and OpenCL, and the communication overhead between CPU and GPU memory transfers. To get timing estimations, the Diplomat framework automatically generates and run samples which summarize enough mapping configuration to cover all the possible timing configurations. As a result a timed SDF can be created, which provides insights on the application general execution behavior.

The partitioning is expressed using the partitioning ratio that is a decimal from 0 to 1 included. The partitioning execution time is inferred simply by assuming that the computation time of a kernel on a CPU or GPU resource vary linearly with respect to the amount of the input data or iteration space. As an example, if the partitioning ratio of a given kernel is 0.5, Diplomat will assume that the timing to execute on CPU will be half of the CPU-only timing and that the timing to execute on GPU will be half of the GPU-only timing.

3) Throughput evaluation: Once a timed SDF is available, the Diplomat framework can evaluate the maximum throughput of this application on specific platforms. This evaluation takes into account all possible partitioning ratios. A first analysis provides an upper bound of the maximum reachable throughput of the application. This is valuable information as it will inform the designer that, for any possible mapping of the application, there is no better possible performance on the chosen platform. This analysis also provides a list of critical tasks which must be optimized at first in order to improve performance.

Considering a particular mapping on a platform, the throughput evaluation is also used in Diplomat to estimate its performance. This throughput evaluation method will be used as a performance metric to solve the mapping decision problem in the next step.

4) Mapping: The mapping of a Diplomat application associates computation resources to each task. These computation resources can be either the CPU or the GPU, or both at the same time in case of kernel partitioning. The solution space of this problem is exponential. Application mapping on heterogeneous systems has been widely studied in the last decades [8]. In Diplomat, the mapping is selected using a genetic-based space exploration technique. In this genetic algorithm, candidate solutions are specific mapping of the application. Their properties are the partitioning ratio for each task on each computing resource. The fitness metric of this exploration is the throughput evaluation method previously defined.

After exploring the space, in order to eliminate potentially bad predictions, we run the most interesting candidates on the targeted platform. This step results in a number of good mappings ready to be generated.

D. Code generation

In the code generation phase, Diplomat generates an actual implementation of the application for a specific mapping. Because each kernel implementation has already been provided, Diplomat need to produce a version of the task-graph which will call each kernel in the specified implementation. In the case of partitioning, Diplomat required a particular implementation of the CPU kernels (where the data size is an extra argument).

It is important to note that this code generation is not done from the dataflow model, but from the task-graph representation. The internal SDF model is exploited by Diplomat to explore mapping solutions but it cannot be used for code generation. The code generation produces multiple threads depending on the parallelism of the application. Each thread is then responsible for a set of task executions on different devices. The task scheduling over those threads is decided by data dependencies as expressed by the Diplomat task-graph model.

IV. Experimental setup

A vision application, namely KFusion, is used as a case study to evaluate Diplomat on two heterogeneous embedded platforms. We first present the use case and the targeted platforms.

A. The KFusion application

KFusion is a computationally intensive vision algorithm and a good candidate for augmented reality applications. In this section we present the core algorithm of KFusion and the configuration we considered in the experiments.

1) KFusion algorithm: In the context of computer vision, simultaneous localization and mapping[1] (SLAM) systems aim to perform real-time localization and mapping "simultaneously" for a camera moving through an unknown environment. The KFusion algorithm [11] utilizes a depth camera to perform localization and mapping. KFusion records and fuses the noisy stream of measured depth frames obtained as the scene is viewed from different viewpoints into a highly detailed 3D geometric map.

[1]Unlike previous sections, the term *mapping* in the computer vision domain refers to the 3D reconstruction of an environment.

```
for depthFrame in depthStream do
    mm2meters(floatDepth,depthFrame)
    bilateralFilter(scaledD[0],floatDepth)
    for i in range (1,len(levels)) do
        halfSample(scaledD[i],scaledD[i-1])
    end for
    for i in range (0,len(levels)) do
        depth2vertex(scaledV[i],scaledD[i])
        vertex2normal(scaledN[i],scaledV[i])
    end for
    oldP = P
    for lev in range (len(levels),0) do
        for i in range (0,levels[lev]) do
            track(tracking,scaledV[lev],scaledN[lev],V,N,P)
            reduce(reduction,tracking)
            if updatePose(P,reduction) then
                break
            end if
        end for
    end for
    if checkPose(P,oldP,reduction) then
        integrate(volume,floatDepth,P)
    end if
    raycast(V,N,volume,P)
    renderVolume(out,volume)
    renderTrack(out,tracking)
    renderDepth(out,floatDepth)
end for
```

Algorithm 1: KFusion algorithm

In our work, we focus on the SLAMBench [4] implementation of KFusion. SLAMBench is a benchmark that provides portable KFusion implementations in C++ (sequential), OpenMP, CUDA and OpenCL, that have been evaluated on desktop and embedded sytems [4]. This fulfills the Diplomat prerequisites of implementations. A simplified pseudo-code of KFusion is given in Algorithm 1. For the sake of brevity, we removed arguments and behaviors that do not induce any data dependencies.

2) *Configuration selection:* To fulfill static analysis requirements, four KFusion configurations have been selected (see Table I). We denote them by the configurations 0, 1, 2 and 3. These configurations have been selected by a (external to our work) computer vision expert. Each parameter is extremely sensitive, and a wrong combination of them could result in incorrect behavior (*i.e.* a loss of the camera position or a poor scene 3D reconstruction). These parameters also have a strong impact on the application speed and workload.

`-c` is the input image ratio: the bigger this number is, the smaller is the input data size.
`-v` is the data size of the 3D model representation.
`-r` is the update frequency of the 3D model: an update is computationally intensive, the smaller this number is, the slower the application.
`-m` is a distance that defines the update region in the 3D model: the bigger this number is, the slower an update.
`-l` is the tracking precision: the smaller this value is, the longer the tracking step.

These parameters imply that the configurations 2 and 3 are the fastest, but also less accurate. In contrast, the configuration 0 and 1 are slower, but more accurate.

Platform name	ODROID-XU3	Arndale
SoC Name	Exynos 5422 Octa	Exynos 5250
CPU	ARM	ARM
CPU Name	Cortex-A7+A15	Cortex-A15
Number of cores	4 + 4	2
Core frequency	2GHz / 1.4GHz	1.7GHz
GPU	Mali-T628	Mali-T604
GPU frequency	600MHz	533MHz
RAM Size	2GB	2GB
Operating system	Ubuntu 11.4	Ubuntu 12.04.5 LTS
C++ Compiler	GCC 4.8.2	GCC 4.8.1
OpenCL Version	OpenCL 1.1 ARM	OpenCL 1.1 ARM

TABLE II: Summary of considered platforms during our experiments.

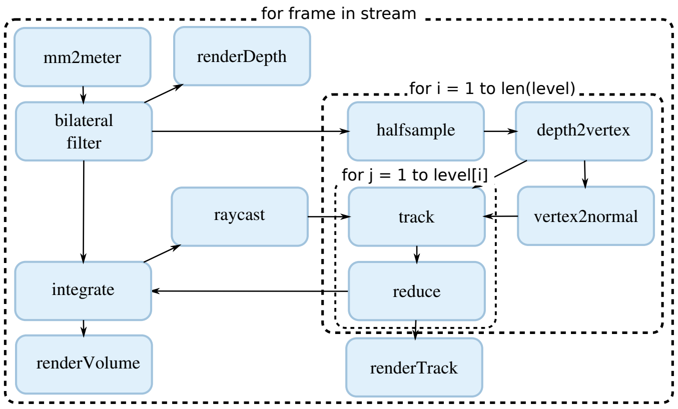


Fig. 7: Graphical representation of KFusion Diplomat internal representation. Here buffers are represented by arrows.

B. *Targeted heterogeneous platforms*

As the motivation of this work is to ease mapping of application on heterogeneous platforms, we applied our methodology on two embedded platforms, the Arndale and the HardKernel ODROID-XU3 boards. A brief summary of these platforms is available on Table II.

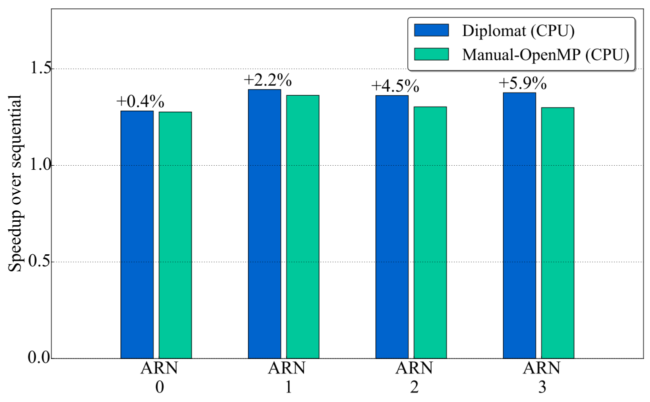
C. *Implementing KFusion with Diplomat*

With a basic knowledge of the KFusion algorithm, describing KFusion with the Diplomat framework took less than a day. This includes the debugging effort. Furthermore, the whole description is less than 200 lines of code. However, we do not claim this positive result can be generalized to any applications. The purpose of Diplomat is not to be directly used by a programmer, but to provide efficient mapping solution for higher-level languages, such as DSLs for vision.

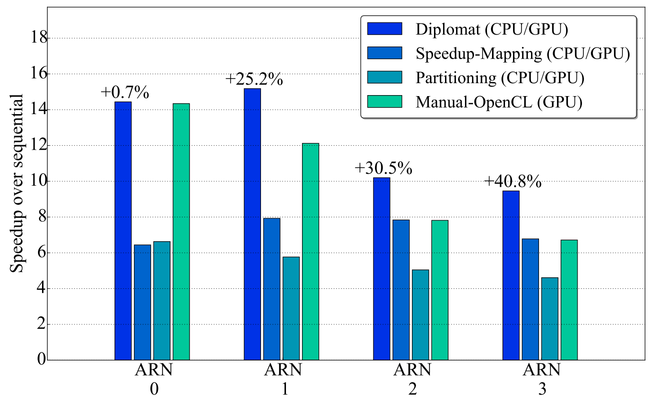
Once an application is described using Diplomat, an internal representation of the task-graph is available; Figure 7 shows this graph for the KFusion application.

Config.	Arguments	Accuracy (cm)	1-CPU Perf. (FPS)		1-GPU Perf. (FPS)	
			Arndale	ODROID	Arndale	ODROID
0	-c 1 -m 0.2 -l 1e-04 -v 128 -r 20	< 4.1cm	0.24	0.33	3.48	3.76
1	-c 2 -m 0.2 -l 1e-05 -v 128 -r 10	< 4.2cm	0.76	1.28	9.26	10.43
2	-c 4 -m 0.075 -l 1e-06 -v 256 -r 5	< 4.4cm	1.49	2.50	11.69	14.79
3	-c 4 -m 0.1 -l 1e-05 -v 128 -r 10	< 4.6cm	2.34	3.91	15.69	20.04

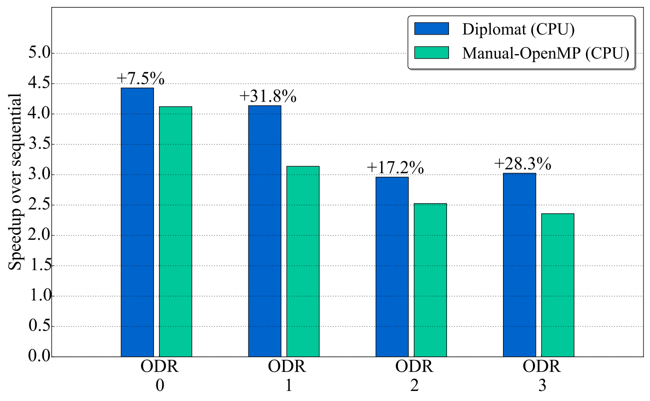
TABLE I: Selected configurations of KFusion for the evaluation of Diplomat. Each of these parameters (`-c`,`-m`...) affects the application behavior and its workload.



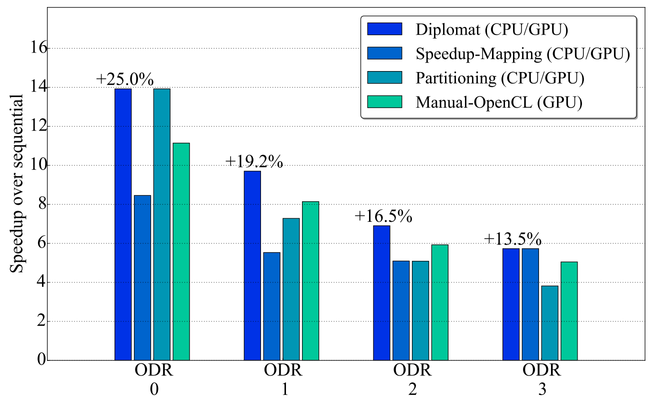
(a) CPU-Only on Arndale



(b) CPU and GPU on Arndale



(c) CPU-Only on ODROID



(d) CPU and GPU on ODROID

Fig. 8: Evaluation of the best result obtained with Diplomat for CPU and GPU configurations, and compared with hand-written solutions (OpenMP, OpenCL) and automatic heuristics (Partitioning, Speed-up mapping). The associated number is the configuration number and the percent on top of Diplomat bars are the speed-up over the `MANUAL` implementation.

V. Performance results

We evaluated four configurations of KFusion on two different platforms. We compared the performance of Diplomat with two families of solutions. We considered the original OpenMP and OpenCL implementations of SLAMBench [4] which are hand-written implementations, and we also considered two automatic mapping methods.

- `MANUAL-OpenMP` and `MANUAL-OpenCL` are respectively the OpenMP and OpenCL hand-written version from SLAMBench.
- `PARTITIONING` is a partitioning [6] of every kernel between CPU and GPU in order to maximize the speed and based on timing profiling.
- `SPEED-UP-MAPPING` is a mapping strategy [5] which selects the fastest resource to compute each kernel.

The performance of the mapping obtained with Diplomat are shown in Figure 8. For each case, the mapping selected by Diplomat is the best known, after having evaluated the 20 best estimated mappings.

A. CPU only

To compare the performance of Diplomat on CPU-only platforms, we used `MANUAL-OpenMP`, the OpenMP implementation provided in SLAMBench. This implementation is not using vectorization. Diplomat does not apply code transformation on the kernel code, thus it does not use vectorization either.

1) Arndale: The Arndale has a two-core CPU and, as visible in Figure 8a, the benefit of MANUAL-OpenMP on this device is limited (+30% on average). With this reduced number of cores the mapping possibilities for Diplomat are small. But Diplomat takes advantage of task-parallelism and provides a small performance improvement on OpenMP (no more than 5%).

2) ODROID: As shown in Figure 8c, MANUAL-OpenMP is more efficient on the ODROID. The ODROID has a heterogeneous CPU composed of 4 big and 4 LITTLE cores. MANUAL-OpenMP yields a 4x speed-up for the configuration 0. This configuration's workload is large, this is the best condition for OpenMP data-parallelism. The performance of OpenMP diminishes when the workload is reduced, but for the smallest workload from configuration 3 this is still a 3x speed-up. The benefit of Diplomat in this context remains important and exploits task-parallelism. With the increase in number of cores, Diplomat is able to exploit more parallelism across configurations. On average, it provides a 20% speed-up. While the MANUAL-OpenMP will sequentially execute preprocessing tracking and rendering, Diplomat executes them simultaneously.

B. CPU and GPU

We compared Diplomat with

- MANUAL-OpenCL, the OpenCL implementation of SLAMBench,
- PARTITIONING, a partitioning solution [6] based on timing profiling,
- and SPEED-UP-MAPPING, a speed-up based mapping technique [5].

1) Arndale: In Figure 8b the performance of MANUAL-OpenCL is good: we see a speed-up between 7x and 14x depending on the configuration. This implementation has been made in a way that it minimizes the data movement, most of the computation happens on the GPU. On the Arndale platform, both PARTITIONING and SPEED-UP-MAPPING perform worse than MANUAL-OpenCL. The main reason for this performance drop is the communication between kernels. If automatic mapping decisions are locally good for a kernel, these imply communications in the whole application. This is also amplified because the Arndale CPU is not powerful enough to provide substantial benefits.

For this platform Diplomat achieved performance improvement mostly due to the code generation method we used. Indeed the best mapping selected by Diplomat on the Arndale board still uses the GPU only. The main difference with the hand-written version concerns the way OpenCL jobs are distributed. They are concurrently launched by several POSIX threads automatically generated by Diplomat. Such a way of running OpenCL jobs gives a more efficient use of the Arndale GPU. This modification could have been done manually, but this clearly is something a programmer would not consider.

2) ODROID: With the ODROID, Figure 8d, the MANUAL-OpenCL performs well too. However this device has a better CPU, thus classical mapping techniques such as PARTITIONING and SPEED-UP-MAPPING can provide performance improvement. For the configurations 0, thanks to an important workload, the PARTITIONING provides 25% speed-up over the manual. For configurations 1,2 and 3, as their workloads diminish, the PARTITIONING performance decreases. The SPEED-UP-MAPPING performs badly with the large workloads of configuration 0: this is due to the higher impact of bad mapping decisions. But when the workload is smaller, this mapping strategy can schedule tasks more efficiently; we see performance improvement of 13% over the manual with configuration 3.

Because Diplomat has sufficient knowledge of the application, it can avoid bad mapping decisions. For the large workload case, it will privilege a strategy similar to PARTITIONING, while for the other configurations, it will adapt the mapping decision in consequence. We also noticed that for two cases (configurations 0 and 3), Diplomat just selected strategies similar to PARTITIONING and SPEED-UP-MAPPING because it did not find better solutions.

VI. Analysis

In this section, we analyse the evaluation process of Diplomat and its prediction accuracy.

A. Timing profiling

For each platform, Diplomat profiled the task duration and the communications overhead of each configuration within less than one hour.

B. Throughput evaluation

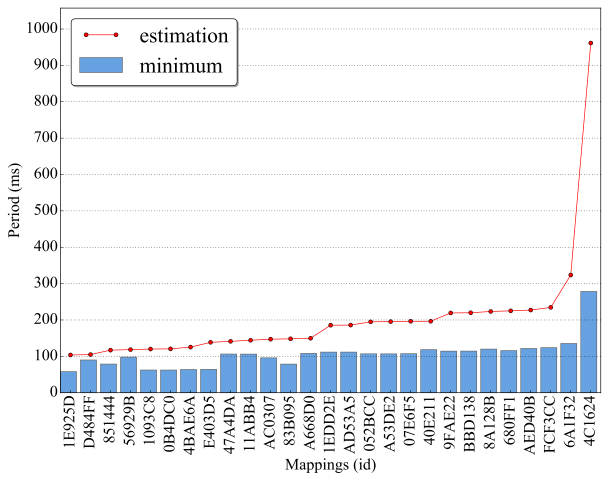
For each configuration and each platform, an upper bound of the maximum possible throughput is computed. Those are available in Table III. The upper bound is not realistic, as it considers instant memory copy and an unlimited number of resources. But this analysis provides potentially useful information about the application behavior. Typically, the static dataflow scheduler identifies the critical circuit of the application for these platforms which is passing through the integrate, raycast, track and reduce kernels. In a scenario where a programmer is hand-tuning the application, in order to improve performance, one of the kernels in the critical circuit must be improved. This circuit is visible in Figure 7.

C. Mapping exploration

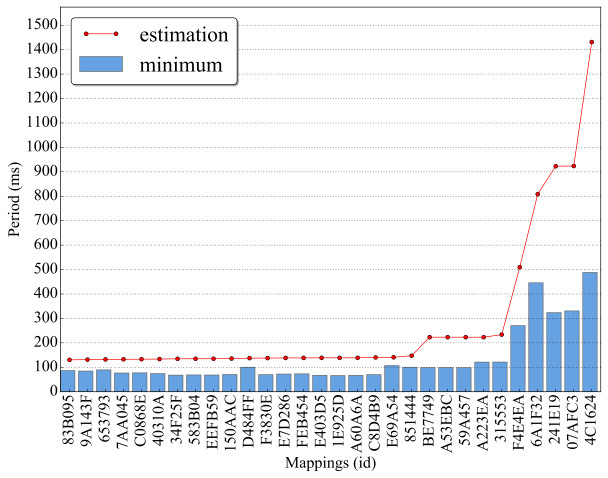
The mapping heuristic used in Diplomat provided more than 20 potentially good mappings in less than an hour.

Configuration	Throughput estimation (FPS)	
	Arndale	ODROID
0	2.55	2.85
1	9.16	9.53
2	10.11	10.03
3	24.24	19.50

TABLE III: This table shows the maximum throughput estimated by Diplomat.



(a) ODROID-XU3



(b) Arndale board

Fig. 9: Estimation of mapping performance on several possible mappings compared with their actual performance on the platforms of Table II using configuration 2. The last case (`4C1624`) is the sequential case.

We manually ran the best of them and compared the actual performance to the predicted performance in Figure 9. Figure 9(a) and Figure 9(b) represent the mapping predictions respectively on the Arndale and the ODROID boards. Even if the order of magnitude is the same, we can see that there is a significant gap between prediction and actual performance. This is because we use a static model and worst case execution times to predict a dynamic application. The important observation regarding those predictions, is that, by considering them ordered by performance (as done in Figure 9), the ordering of the actual performance is similar. This means that mapping can be performed using the prediction model instead of actual execution.

VII. Related work

The *Deadalus RT* [12] framework targets hard-real-time embedded streaming systems and is probably the most similar approach. It provides a specific input language, the Polyhedric Process Network (PPN), and relies on Cyclo-Static Dataflow (CSDF) to analyse the application. But the PPN expressivity is limited to static affine nested loop programs and the CSDF generated can be used to produce the real-time application schedule. On the contrary, there is no such constraint in Diplomat. The SDF model considered will remain an abstraction of the task-graph as this model is less expressive. Similarly, toolchains like *StreamIt* [2] or *Sigma-C* [13] express applications through a static dataflow model. They provide all the static analysis tools required to make efficient mapping decisions. But these models do not support OpenCL generation, and are not expressive enough to support the modern vision applications we are considering.

In [14], a framework is proposed to generate heterogeneous code (using OpenCL and C++) from an SDF representation of an application. This framework suffers from the same limitation regarding the expressivity, and does not take partitioning into account.

The *MAPS* [15] framework undertakes this expressivity limitation using the KPN model. Applications are described using the C for process networks (CPN) language. Throughput analysis and mapping are only possible using trace-based analysis tools, which remains less accurate than the information collected by Diplomat. Furthermore, *MAPS* does not provide an OpenCL back-end nor a task partitioning policy. Meanwhile, it is important to note that MAPS supports multi-application analysis. Concurrent applications analysis is important on modern embedded systems where devices are no longer designed to serve a single application. Diplomat is currently not dealing with multi-application.

In [16] a similar two-step framework is proposed. The tuning of the kernels is performed first and then a mapping is provided by solving a Constraint Program (CP). For the mapping part, they use a Directed Acyclic Graph (DAG) representation of the application, which is a less sophisticated representation than the one used in Diplomat.

In contrast, *StarPU* [17], *OmpSs* [18] and *OpenStream* [19] are runtime systems that make it possible to dynamically express an application as a set of tasks connected by data dependencies. These frameworks dynamically schedule tasks and were initially focused on homogeneous platforms. The main limitation of these frameworks is that they only consider runtime scheduling with limited static knowledge of the current application. Then, they do not deliver enough information to estimate achievable performance or predict good mappings beforehand. In a recent StarPU extension [20], the runtime scheduling policy is improved to target embedded platforms. Yet, the same limitation aforementioned applies also to their extension. With heterogeneous platforms, dynamic policies are becoming less efficient. More recently, the *Symphony* [3] task-graph runtime has been proposed to target heterogeneous embedded SoC using both CPU, GPU and DSP at the same time. However, this tool requires to manually specify the mapping.

The *PetaBricks* [21] framework shares common features with Diplomat but it mainly deals with single-kernel applications. In PetaBricks, a DSL is used to represent an algorithm as the combination of several methods and transformation rules. In a recent extension [22], it also considers OpenCL code generation and partitioning. Expressing a multi-kernel application in this formalism is difficult: it requires to express too many rules and the search space becomes too big.

To improve the information provided by OpenCL, *Helium* [23] provides a high-level representation of the application. It avoids unnecessary data movements and integrates multi-kernel transformations such like kernel merging, which are not considered in Diplomat. This framework is dynamic and it has the advantage to be transparent for the user. However, Helium does not gather required information, *i.e.* timings, to make mapping decisions.

VIII. Conclusion

In this work we apply dataflow static analysis to perform static mapping of multi-kernel applications in heterogeneous embedded platforms. Using our methodology, we observed performance improvement over state-of-the-art for a streaming computer vision application.

In future work we will consider more expressive dataflow models. To cope with the poor expressivity of the SDF model used in this work, several dynamic variables have been fixed to perform static analysis of the application. To fix these values automatically, Diplomat has to consider more expressive models such as the PSDF [24], the SADF [25] or PiMM [26]. From the HPC community, DAGuE [27] is a runtime system modeled using an intermediate language, namely JDF. This language is based on the compact DAG representation [28], which is similar to the model proposed by Diplomat. An interesting research direction is to consider JDF as a Diplomat front-end and to directly tackle the static analysis from this language.

References

[1] Khronos, "The OpenCL 1.2 specification," 2012.

[2] W. Thies, M. Karczmarek, and S. Amarasinghe, "StreamIt: A language for streaming applications," *Compiler Construction*, pp. 179–196, 2002.

[3] Qualcomm, "Qualcomm MARE: Enabling Applications for Heterogeneous Mobile Devices," Qualcomm Inc., Tech. Rep., 2014.

[4] L. Nardi, B. Bodin, M. Z. Zia, J. Mawer, A. Nisbet, P. H. J. Kelly, A. J. Davison, M. Luján, M. F. P. O'Boyle, G. Riley, N. Topham, and S. Furber, "Introducing SLAMBench, a performance and accuracy benchmarking methodology for SLAM," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, May 2015, arXiv:1410.2167.

[5] V. T. Ravi, M. Becchi, W. Jiang, G. Agrawal, and S. Chakradhar, "Scheduling concurrent applications on a cluster of cpu-gpu nodes," in *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (Ccgrid 2012)*, ser. CCGRID '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 140–147.

[6] K. Kofler, I. Grasso, B. Cosenza, and T. Fahringer, "An automatic input-sensitive approach for heterogeneous task partitioning," in *Proceedings of the 27th International ACM Conference on International Conference on Supercomputing*, ser. ICS '13. ACM, 2013, pp. 149–160.

[7] A. H. Ghamarian, M. Geilen, S. Stuijk, T. Basten, A. Moonen, M. J. Bekooij, B. D. Theelen, and M. Mousavi, "Throughput Analysis of Synchronous Data Flow Graphs," in *International Conference on Application of Concurrency to System Design (ACSD'06)*, 2006, pp. 25–36.

[8] A. Singh, M. Shafique, A. Kumar, and J. Henkel, "Mapping on multi/many-core systems: Survey of current and emerging trends," in *Design Automation Conference (DAC)*, May 2013, pp. 1–10.

[9] G. Kahn, "The semantics of a simple language for parallel programming," *Information processing*, 1974.

[10] E. A. Lee and D. G. Messerschmitt, "Synchronous dataflow," *Proceedings of the IEEE*, vol. 75, no. 9, pp. 1235–1245, 1987.

[11] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohli, J. Shotton, S. Hodges, and A. Fitzgibbon, "KinectFusion: Real-time dense surface mapping and tracking," in *ISMAR*, 2011.

[12] M. Bamakhrama, J. Zhai, H. Nikolov, and T. Stefanov, "A methodology for automated design of hard-real-time embedded streaming systems," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2012*, March 2012, pp. 941–946.

[13] T. Goubier, R. Sirdey, S. Louise, and V. David, "ΣC: A programming model and language for embedded manycores," in *Algorithms and Architectures for Parallel Processing*. Springer, 2011, pp. 385–394.

[14] L. Schor, A. Tretter, T. Scherer, and L. Thiele, "Exploiting the parallelism of heterogeneous systems using dataflow graphs on top of opencl," in *Embedded Systems for Real-time Multimedia (ESTIMedia), 2013 IEEE 11th Symposium on*, Oct 2013, pp. 41–50.

[15] J. Castrillon, R. Leupers, and G. Ascheid, "Maps: Mapping concurrent dataflow applications to heterogeneous mpsocs," *Industrial Informatics, IEEE Transactions on*, vol. 9, no. 1, pp. 527–545, Feb 2013.

[16] E. Paone, F. Robino, G. Palermo, V. Zaccaria, I. Sander, and C. Silvano, "Customization of OpenCL Applications for Efficient Task Mapping under Heterogeneous Platform Constraints," in *Design, Automation & Test in Europe (DATE)*, 2015.

[17] C. Augonnet, S. Thibault, R. Namyst, and P.-A. Wacrenier, "StarPU: a unified platform for task scheduling on heterogeneous multicore architectures," *Concurrency and Computation: Practice and Experience*, vol. 23, pp. 187–198, 2011.

[18] A. Duran, E. Ayguadé, R. M. Badia, J. Labarta, L. Martinell, X. Martorell, and J. Planas, "Ompss: a proposal for programming heterogeneous multi-core architectures," *Parallel Processing Letters*, vol. 21, no. 02, pp. 173–193, 2011.

[19] A. Pop and A. Cohen, "A OpenStream: Expressiveness and Data-Flow Compilation of OpenMP Streaming Programs," *TACOS*, vol. V, no. January, 2013.

[20] H. Zhou and C. Liu, "Task mapping in heterogeneous embedded systems for fast completion time," *Proceedings of the 14th International Conference on Embedded Software - EMSOFT '14*, pp. 1–10, 2014.

[21] J. Ansel, C. Chan, Y. L. Wong, M. Olszewski, Q. Zhao, A. Edelman, and S. Amarasinghe, "Petabricks: A language and compiler for algorithmic choice," in *Proceedings of the 30th ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI '09. New York, NY, USA: ACM, 2009, pp. 38–49.

[22] P. M. Phothilimthana, J. Ansel, J. Ragan-Kelley, and S. Amarasinghe, "Portable performance on heterogeneous architectures," in *Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '13. New York, NY, USA: ACM, 2013, pp. 431–444.

[23] T. Lutz, C. Fensch, and M. Cole, "Helium: a transparent inter-kernel optimizer for opencl," in *Proceedings of the 8th Workshop on General Purpose Processing using GPUs, GPGPU@PPoPP 2015, San Francisco, CA, USA, February 7, 2015*, 2015, pp. 70–80.

[24] B. Bhattacharya and S. Bhattacharyya, "Parameterized dataflow modeling for dsp systems," *Signal Processing, IEEE Transactions on*, vol. 49, no. 10, pp. 2408–2421, Oct 2001.

[25] S. Stuijk, M. Geilen, B. D. Theelen, and T. Basten, "Scenario-aware dataflow: Modeling, analysis and implementation of dynamic applications," in *2011 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation, SAMOS XI, Samos, Greece, July 18-21, 2011*, 2011, pp. 404–411.

[26] K. Desnos, M. Pelcat, J.-F. Nezan, S. Bhattacharyya, and S. Aridhi, "Pimm: Parameterized and interfaced dataflow meta-model for mpsocs runtime reconfiguration," in *Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XIII), 2013 International Conference on*, July 2013, pp. 41–48.

[27] G. Bosilca, A. Bouteiller, A. Danalis, T. Herault, P. Lemarinier, and J. Dongarra, "Dague: A generic distributed dag engine for high performance computing," ICL, University of Tennessee, Tech. Rep., 2010.

[28] M. Cosnard, E. Jeannot, and T. Yang, "Compact dag representation and its symbolic scheduling," *J. Parallel Distrib. Comput.*, vol. 64, no. 8, pp. 921–935, Aug. 2004.