Extending Architecture Modeling for Signal Processing towards GPUs

Saman Payvar
Tampere University
Tampere, Finland
saman.payvar@tuni.fi

Jani Boutellier
Tampere University
Tampere, Finland
jani.boutellier@tuni.fi

Antoine Morvan
IETR/INSA Rennes
Rennes, France
antoine.morvan@insa-rennes.fr

Claudio Rubattu
IETR/INSA Rennes and UNISS
Rennes, France and Sassari, Italy
crubattu@uniss.it

Maxime Pelcat
IETR/INSA Rennes - Institut Pascal
Rennes and Clermont Ferrand, France
maxime.pelcat@insa-rennes.fr

***Abstract*—Efficient usage of heterogeneous computing architectures requires distribution of the workload to available processing elements. Traditionally, this mapping is done based on information acquired from application profiling. To reduce the high amount of manual work related to mapping, statistical application and architecture modeling can be applied for automating mapping exploration. Application modeling has been studied extensively, whereas architecture modeling has received less attention. Originally developed for signal processing systems, Linear System Level Architecture (LSLA) is the first architecture modeling approach that clearly distinguishes the underlying computation hardware from software. Up to now, LSLA has covered the modeling of multicore CPUs. This work proposes extending the LSLA model with GPU support, by including the notion of parallelism. The proposed GPU modeling extension is evaluated by performance estimation of three signal processing applications with various workload distributions on a desktop GPU, and a mobile GPU. The measured average fidelity of the proposed model is 93%.**

***Index Terms*—modeling, architecture, design space exploration, signal processing systems**

I. Introduction

Heterogeneous computing platforms that contain GPUs and DSPs alongside general-purpose processors, have become mainstream platforms for many signal processing applications, such as image, video and audio processing. One of the design decisions that should be made in the early stage of programming such systems is the mapping of the application to the platform i.e. workload consideration for processing elements. Unfortunately, the exploration of mapping alternatives is nowadays still mostly performed manually, which is a work-intensive and time consuming task. An approach that considerably reduces the effort is building models of the target platform and the application and exploiting them with automatic techniques or tools. With a suitable modeling approach combined with design space exploration, the efficiency of hundreds or thousands of mapping alternatives can be approximated within seconds.

In statistical system modeling, the application and the architecture are often considered together. Originally introduced for modeling of signal processing systems [1], Linear System Level Architecture (LSLA) [2], however, is the first Model of Architecture (MoA) that clearly separates the underlying architecture from the software running on top of it. LSLA specifically models the architecture and distinguishes the concepts of Model of Computation (MoC) from the MoA. The MoA and MoC separation reduces the modeling effort by formulating the system modeling as mapping of MoC activity to the MoA, so that the MoA and the MoC can be treated independently when needed. In LSLA it is possible to map different types of MoC to the LSLA, such as Synchronous Data Flow (SDF) [3] that is especially popular in signal processing.

In LSLA, an application described by a MoC is mapped to a processing architecture modeled by the LSLA MoA, and by considering the activity of the application, a cost function is computed for each processing element in the platform. For estimating the performance of various mapping alternatives, the cost functions of the processing elements are summed while varying the mapping parameters. In the original LSLA work, Pelcat et al. [2] have modeled the energy consumption of the Odroid XU3 platform with a graph. In this particular case, eight processing elements interconnected by three communication nodes model the asymmetric eight-core CPU of the Odroid platform. LSLA provides a model for parallel programing for CPU cores, while most contemporary platforms include also a GPU, which motivates the proposed work.

The contributions of this work are:

- An extension, called LSLAG, of LSLA is presented for covering GPU units. The proposed GPU extension to the model is linear, similar to the original LSLA that only covers CPU cores.
- For experimental evaluation of the proposed model, three applications are implemented in OpenCL and are executed on two different GPU-equipped platforms. Based on these experiments, the average fidelity of the model is 93%, which is similar to the original LSLA model

978-9-0827-9703-9/19/$31.00 ©2019 IEEE

proposed for the CPU cores of the same platform.

This paper is structured as follows: Section 2 introduces related work and provides a comparison to the proposed work. Section 3 introduces the MoA concept. Section 4 explains the proposed LSLA extension. Section 5 explains the parameters. Section 6 elaborates the performed experiments. Finally, Section 7 explains the conclusions and outlines the future work.

II. Related Works

There are different methodologies in performance modeling studies. One of them, which provides a general model is the statistical analysis method. For example, Moren, et al. [4] present a statistical approach for work load scheduling on heterogeneous platforms consisting of CPU and GPU. They have modified the OpenCL API code for dynamic code feature collection which is used for performance prediction. In modeling methods, it is common to use a graph to present a software or a hardware system, or a system of systems. These methods are divided into two different categories: data flow graphs and non data flow graphs. In a data flow graph, a vertex is used to model a run-to-completion block of computation called an *actor*. *Edges* are used to model data token communications between actors, realized by FIFO queues (First In First Out). In addition, weights on FIFOs, called delays, are used to represent initial data present on edges. The execution of a data flow actor is called *firing* and is triggered when an actor has sufficient data on each input edge. Table I lists modeling approaches and the graph semantics used in related works.

SDF (Synchronous Data Flow) [3] is a well-known static MoC. In SDF, a system is modeled with a data flow graph where the firing rules specify the constant token consumption and production rates for all actors. These constant rates introduce limitations in terms of algorithmic behavior representation.

CFDF (Core Functional Data Flow) [5] is a form of EIDF (Enable Invoke Data Flow) [6] where a limited set of modes influence token consumptions and productions. CFDF limits mode transitions to only one alternative, making the model deterministic.

BSP (Bulk Synchronous Parallel) [7], unlike SDF or CFDF is a system modeling method, and it has its own graph implementation. In BSP, there are processing units with local memories connected over a router. Processing elements access each other's memories by remote access messages.

DAL (Distributed Application Layer) [8] has a dynamic mapping methodology. It employs Kahn process networks to explore application mappings and a finite state machine to represent execution scenarios. Multiple scenarios are precomputed at design-time and the suitable one is selected at run-time.

Bezati et al. [9] present a data flow modeling method according to the CAL language [10]. Their method has six steps. First, two different models for application and architecture are designed. Second, simulation and profiling results are collected. Third, the application is mapped to the architecture. Fourth, C++ and HDL codes are generated from CAL. Fifth, the code is compiled and synthesized. Finally, compiled code is executed.

LSLA [2] is a MoA, separate from the MoC. The LSLA MoA includes Processing Elements (PE) and Communication Nodes (CN). PEs and CNs of the LSLA MoA has cost functions including parameters that may be retrieved from representative platform benchmarking. In that case, the calculated cost functions are obtained from measured application executions and the cost function parameters can be used to predict system efficiency for a set of comparable applications.

The proposed work i.e. LSLAG provides a system modeling approach that as an extension to LSLA has the benefit of reduced modeling effort due to its re-usability. In addition, GPU coverage of LSLAG provides more complete coverage of modern heterogeneous platforms.

TABLE I
Modeling approaches

Method	Target	Graph
SDF	Application	Dataflow
CFDF	Application	Dataflow
BSP	System	Non Dataflow
DAL	System	Non Dataflow
[9]	System	Dataflow
LSLA	Architecture	Non Dataflow
LSLAG	Architecture	Non Dataflow

III. Models of Architecture

The MoA concept [2] is used to distinguish the processing architecture from the MoC, which should only address applications. A MoA is defined as a graph that can be used for reproducible execution cost (time, energy, etc.) calculations. A MoA is designed for each specific processing architecture and it covers the processing elements and their interconnect.

Each element in the MoA graph has a cost function whose parameters can be estimated statistically according to measurement results. The calculated parameter values depend on the application and the application's configuration.

A. Linear System Level Architecture

LSLA is a specific type of MoA that uses *linear* cost functions for each MoA graph element. The total cost of the modeled platform is calculated according to the Equation 1, which depicts the total cost of application activity A on the LSLA graph P. In this equation, the total cost is equal to the sum of the processing cost, and of the communication cost, λ being a scaling coefficient between processing and communication cost units. Tp depicts set of all mapped tokens to the processing elements and Tc shows set of all mapped tokens to the communication nodes. The activity of the mapped MoC is calculated as *tokens*, consisting of *quanta*, resulting in an affine cost model per communication and per processing. The quanta are an application-independent unit of execution cost.

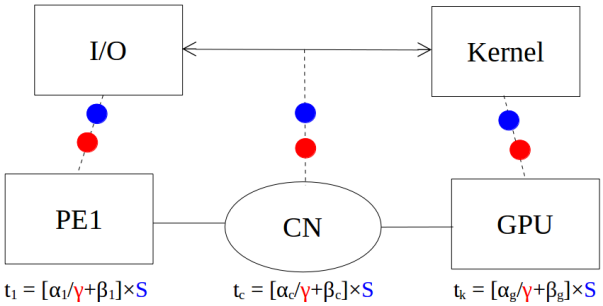

Fig. 1. Mapping of the application to the architecture model.

$$cost(A,P) = \sum_{t \in Tp} cost(t, map(t)) + \lambda \sum_{t \in Tc} cost(t, map(t)) \quad (1)$$

In LSLA, the application and its activity (i.e. the pressure it puts on hardware) are mapped as activity tokens to the LSLA model of the platform. Activity of the application includes *processing tokens* and *communication tokens*. These tokens are mapped to their associated elements in the platform model: processing tokens are mapped to processing elements and communication tokens are mapped to interconnection nodes that are used to transfer data between PEs.

IV. LSLAG: The Proposed Extension to LSLA

The proposed extension to LSLA of this work covers the GPU that can be present in a heterogeneous platform. Figure 1 shows an LSLAG graph that includes a CPU core *PE1*, the GPU, and the interconnection *CN* between the CPU core and the GPU. *PE1* is assumed to act as the host processor that communicates with the GPU. This simple LSLAG model has three elements including two processing elements and one communication node. Each element has its own cost function (presented beneath the nodes) that has two variables named γ and S, as well as two linear parameters α and β whose values are estimated for modeling purposes. The presented LSLAG is used to model the execution time of the platform, thus time samples are used in parameter calculations. As presented in Equation 1, the total cost is a sum of all cost elements, i.e. the execution time of the GPU (t_k), the execution time of the host processor (t_1) and the execution time of the interconnect (t_c). The hypothesis of Equation 2 is justified by consideration that (t_k), (t_1) and (t_c) do not overlap in time, i.e. the kernels of the GPU application are managed by the host device, then executed by the GPU at separate time intervals.

$$t_w = t_k + t_1 + t_c \quad (2)$$

$$t_k(\gamma, S) = (\alpha_g/\gamma + \beta_g) \times S \quad (3)$$

$$t_1(\gamma, S) = (\alpha_1/\gamma + \beta_1) \times S \quad (4)$$

$$t_c(\gamma, S) = (\alpha_c/\gamma + \beta_c) \times S \quad (5)$$

In these equations γ and S are variables, where γ is the parallelism factor, and S is the input data quantity. As it can be seen, increasing the parallelism factor γ decreases the total execution cost asymptotically. Conventional LSLA does not deal with parallelism, which limits its use to CPU cores. LSLAG adds the parallelism factor γ that enables including parallel processing elements. For each GPU-related MoA graph entity (i.e., the GPU itself, the host PE and the interconnect) there are separate parameters α and β, where α can be regarded as the reciprocal of slope, and β as intercept. In designing the model for the factors t_1, t_c, and t_k, simplicity was one of the driving motivations. To this end, t_1, t_c and t_k all have identical equations, and the model fitting will make the parameters settle to values that reflect the real trends of the factors. The next section describes the proposed approach of estimating each α and β.

V. Estimation of Parameters

Acquiring an accurate execution time model for an application running on a GPU requires reliable profiling data. The proposed estimation approach assumes three accurate factors that can be profiled on the platform

- Application total *wall-clock time* t_w,
- Host code execution time t_1, and
- GPU kernel execution time t_k.

The remaining factor t_c, in contrast, is derived using t_w, t_1 and t_k. The proposed procedure for acquiring accurate measurements for the factors are as follows: t_w is measured using the operating system clock, and t_k is read from the profiling data available from the GPU application programming interface. The measurement of t_1 is performed by modifying the application so that all GPU-related calls are disabled and the application only performs data I/O. Finally, t_c is derived from the other factors by subtracting t_1 and t_k from t_w.

VI. Experiments

The experiments presented below serve to illustrate the suitability of the proposed model and Equations 2-5 for real-life GPU-equipped platforms. Typical signal processing applications were used as case studies: matrix multiplication, digital predistortion and Gaussian image filtering. The applications were written in OpenCL and were executed on two GPU-equipped platforms: the Odroid XU3 containing a Mali T628 GPU and a desktop workstation with the AMD RX 460 (Baffin) GPU.

The α and β parameters of the cost functions were estimated with a Matlab script that invoked a least squares fitting algorithm (see Section 2 of [11]). In the Matlab script, the `lsqlin` function was used with a positive solution constraint.

Each application was profiled with two application variables, i.e., S and γ. Each variable had six values where S = { 512, 1024, 2048, 4096, 8192, 16384 } and γ = { 8, 16, 32, 64, 128, 256}. The global work size of OpenCL applications

was set application dependently. For matrix multiplication and predistortion, the work size set was calculated by 256*gamma, while for gaussian filtering, it is 1024*γ. The reason for this variation is in the input data types i.e. gaussian filtering reads 1-byte data, while matrix multiplication and predistortion read 4-byte data items. For each (γ, S) combination the execution time was measured 10 times, giving a total of 360 samples per application/architecture combination.

A. Application-architecture mapping

In OpenCL, when computations are performed on a GPU, the CPU works as the host device that reads data from I/O, sends it to the GPU for processing, receives the computed result and stores it back to I/O. Based on the dataflow [3] MoC, a generic model for OpenCL applications was created. Data reading and writing of the CPU is mapped to an *I/O* node (see Figure 1). The *Kernel* node represents the computations performed on the GPU, whereas the communication between the *I/O* and *Kernel* nodes is presented with a bidirectional arrow in Figure 1.

In each actor firing of the application graph, actors and the communication FIFO provide a token, which is mapped to their associated PE or CN node of the model. In other words, the tokens of the node *I/O* are mapped to the *PE1* architecture node, the tokens of *Kernel* are mapped to the *GPU* architecture node, and the communication FIFO tokens to the *CN* node. The cost functions shown below the architecture nodes have two variables, thus two tokens on the mapping lines in Figure 1 are used to present the number of quanta for each variable.

B. Results and Discussion

This section shows how the proposed GPU execution time model fits with the measured execution time samples. In Figure 2, Figure 3, and Figure 4 the bottom axes depict the variables S and γ, whereas the vertical axis depicts execution time. The dots represent the average of individual measured execution time samples. *The measured execution time samples are t_w (wall-clock time) values, and the mesh depicts the model-based sum of $t_k + t_1 + t_c$.* For clarity, the measured time samples depict the average of the 10 measurements for each (γ, S) coordinate.

Table II depicts the calculated α and β parameter values for each application on Baffin and Mali GPUs. These parameters are used in the Equation 2 for calculating t_w. The α value represents the cost of a token and equals to the slope of the mesh. The β value represents the constant time offset of the relevant LSALG element and is the *tw* intercept of the mesh graph. Due to technical difficulties, values for the digital predistortion application were not acquired on the Mali platform. App 1 is matrix multiplication, App 2 is digital predistortion and App 3 is Gaussian filtering. M stands for Mali and B for Baffin platforms.

Table III demonstrates the fitting error between the model and measured samples for each application/platform combination as *fidelity* values. To highlight the improvement of the proposed LSLAG model over conventional LSLA for GPU targets, Table III also shows the fidelity value for LSLA. Fidelity is Kendall's Tau Coefficient value calculated by the `corr` function of Matlab. For computing the fidelity, the 360 execution time samples were randomly divided into a training set of 288 samples, and a test set of 72 samples. Fidelity calculations are performed similarly to [12] and present a value between 0 and 1 where 1 would represent a perfect match between model and samples.

TABLE II
CALCULATED COST FUNCTION PARAMETERS

App.	α_g	β_g	α_1	β_1	α_c	β_c
B1	0.001	0.008	0.000	0.004	0.005	0.068
M1	0.003	0.009	0.000	0.009	0.016	1.554
B2	0.005	0.049	0.002	0.003	0.060	0.000
B3	0.000	0.051	0.002	0.000	0.004	1.074
M3	0.003	0.023	0.022	0.000	0.082	0.460

TABLE III
FIDELITY OF THE TEST SETS ON EXECUTION PLATFORMS.

Application	Platform	Fidelity LSLAG (proposed)	Fidelity LSLA
1	Baffin	0.88	0.75
1	Mali	1.00	0.70
2	Baffin	0.90	0.92
3	Baffin	0.91	0.62
3	Mali	1.00	0.92

In the Table III results, it can be seen that conventional LSLA yields considerably worse fidelity than the proposed LSLAG for GPU architectures. The reason for this is evident: LSLA does not capture parallelism (γ), which is an integral part of GPU processing. An exception to this is the predistortion application on the Baffin GPU, where LSLA and LSLAG yield almost identical fidelity. The reason for this is that on this platform, communication time dominates over parallelized kernel execution, making the whole application behave almost similar to a sequential application. Dominance of communication can be seen in Table II as the high value of coefficient α_c for application B2.

The measured fidelity values also show that the proposed linear LSLAG model fits better the Mali platform than the Baffin platform. The difference is likely related to the different memory architectures; Mali uses a shared memory between the CPU and the GPU, whereas the Baffin GPU is connected over PCI Express.

VII. CONCLUSION AND FUTURE WORK

In this work, LSLAG which is a GPU extension to the LSLA model, was proposed. The proposed model is linear, like the original LSLA model that is intended for multicore CPU platforms. The validity of the proposed model was evaluated by profiling three OpenCL applications on two GPU-equipped platforms, and the achieved model fidelity was 93% for the considered set of signal processing use cases.

Similar to the LSLA model, LSLAG can be used for design space exploration and performance prediction of signal

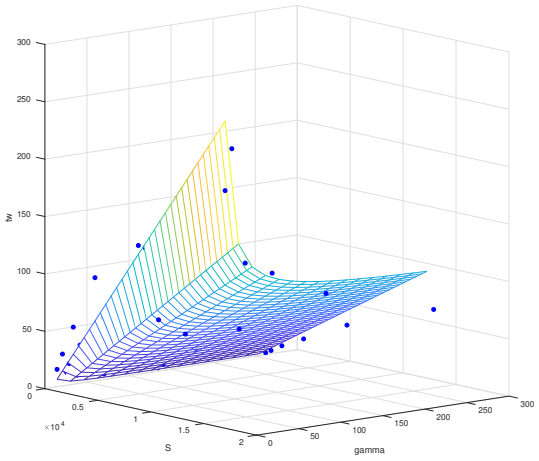

Fig. 2. Matrix multiplication on the Baffin GPU.

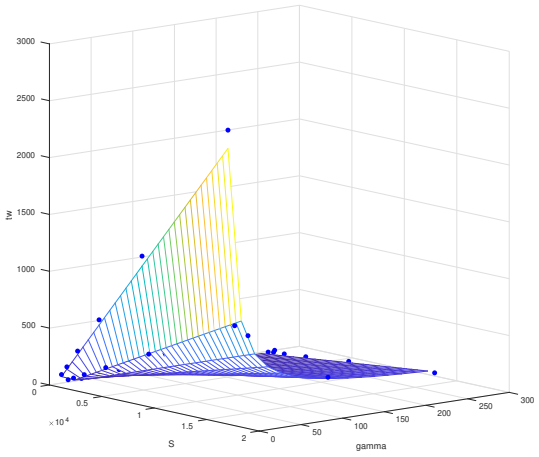

Fig. 3. Gaussian filtering on the Baffin GPU.

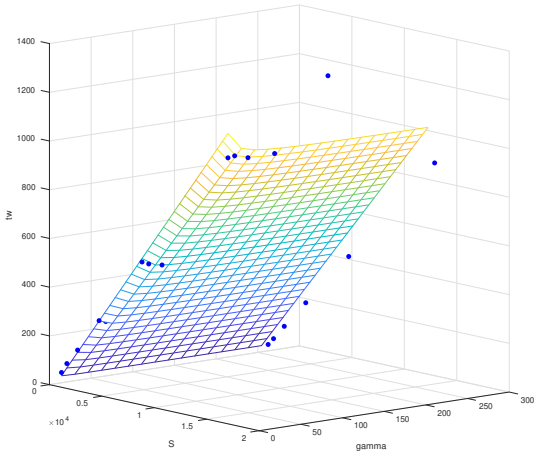

Fig. 4. Predistortion on the Baffin GPU.

processing systems, with the difference that the proposed model also covers GPU-equipped architectures.

Future work involves extending the modeling approach to cover energy measurements, and connecting the GPU extension to larger application graphs.

VIII. Acknowledgment

This work is partially supported by the ITEA3 project 16018 COMPACT and by the European Union Horizon 2020 research grant 732105 CERBERO.

References

[1] Maxime Pelcat, Karol Desnos, Luca Maggiani, Yanzhou Liu, Julien Heulot, Jean-François Nezan, and Shuvra S. Bhattacharyya, "Models of architecture: Reproducible efficiency evaluation for signal processing systems," in *IEEE International Workshop on Signal Processing Systems (SiPS)*. IEEE, 2016, pp. 121–126.

[2] Maxime Pelcat, Alexandre Mercat, Karol Desnos, Luca Maggiani, Yanzhou Liu, Julien Heulot, Jean-François Nezan, Wassim Hamidouche, Daniel Ménard, and Shuvra S. Bhattacharyya, "Reproducible evaluation of system efficiency with a model of architecture: From theory to practice," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 10, pp. 2050–2063, 2018.

[3] Edward A. Lee and David G. Messerschmitt, "Synchronous data flow," *Proceedings of the IEEE*, vol. 75, no. 9, pp. 1235–1245, 1987.

[4] Konrad Moren and Diana Göhringer, "Automatic mapping for OpenCL-programs on CPU/GPU heterogeneous platforms," in *International Conference on Computational Science*. Springer, 2018, pp. 301–314.

[5] William Plishker, Nimish Sane, Mary Kiemb, Kapil Anand, and Shuvra S. Bhattacharyya, "Functional DIF for rapid prototyping," in *IEEE/IFIP International Symposium on Rapid System Prototyping (RSP)*. IEEE, 2008, pp. 17–23.

[6] William Plishker, Nimish Sane, Mary Kiemb, and Shuvra S. Bhattacharyya, "Heterogeneous design in functional DIF," in *International Workshop on Embedded Computer Systems*. Springer, 2008, pp. 157–166.

[7] Leslie G. Valiant, "A bridging model for parallel computation," *Communications of the ACM*, vol. 33, no. 8, pp. 103–111, 1990.

[8] Lars Schor, Iuliana Bacivarov, Devendra Rai, Hoeseok Yang, Shin-Haeng Kang, and Lothar Thiele, "Scenario-based design flow for mapping streaming applications onto on-chip many-core systems," in *International conference on compilers, architectures and synthesis for embedded systems*. ACM, 2012, pp. 71–80.

[9] Endri Bezati, Richard Thavot, Ghislain Roquier, and Marco Mattavelli, "High-level dataflow design of signal processing systems for reconfigurable and multicore heterogeneous platforms," *Journal of real-time image processing*, vol. 9, no. 1, pp. 251–262, 2014.

[10] Johan Eker and Jorn Janneck, "Cal language report," Tech. Rep., Tech. Rep. ERL Technical Memo UCB/ERL, 2003.

[11] Richard C. Aster, Brian Borchers, and Clifford H. Thurber, *Parameter estimation and inverse problems*, Elsevier, 2018.

[12] Neal K. Bambha and Shuvra S. Bhattacharyya, "A joint power/performance optimization algorithm for multiprocessor systems using a period graph construct," in *International symposium on system synthesis*. IEEE Computer Society, 2000, pp. 91–97.