FPGA Roofline modeling and its Application to Visual SLAM

Ioanna-Maria Panagou, Maria-Rafaela Gkeka, Alexandros Patras, Spyros Lalis,
Christos D. Antonopoulos, and Nikolaos Bellas
Department of Electrical and Computer Engineering, University of Thessaly, Volos, Greece
{ipanagou, margkeka, patras, lalis, cda, nbellas}@uth.gr

***Abstract*—The Roofline model has been proposed to visually associate application performance against the computational and bandwidth capabilities of the underlying platform. Since FPGAs lack fixed operation units, modifications in the original CPU-based Roofline model should be made. In this paper, we propose a new application-centric approach to construct the FPGA Roofline model extending previous work and encompassing resource and latency constraints to provide a more fitting ceiling. Moreover, we generalize our model to accommodate platforms with multiple accelerators whose execution footprint may be strongly input-dependent due to conditionals and complex loop structures. We evaluate our model and compare it with previous models on KinectFusion, a complex, multi-kernel algorithm for visual Simultaneous Localization and Mapping (vSLAM) used for autonomous agent navigation. Our work makes it feasible to deploy Roofline analysis on a wider range of MPSoC-based FPGAs that consist of more complex HW/SW components and not just single accelerators.**

I. Introduction

Reconfigurable hardware accelerators have become popular in embedded systems, trading higher development time with improved performance and energy efficiency. However, the lack of tools to assist designers in evaluating the effect of FPGA optimizations and guide optimization selection has always been an impediment to FPGA system design. The Roofline model [1], a concept proposed initially for CPU performance analysis, could potentially become established as a solid tool to estimate optimization gains for FPGA designs.

The Roofline model is a 2D graph that estimates performance limits of a given application running on a multi-core or accelerator architecture, by showing its inherent hardware limitations. The vertical Y axis represents the performance (in *ops/sec*), and the horizontal X axis the arithmetic intensity (AI) of the executed application (in *ops/byte*). The AI denotes the number of operations per byte of memory traffic. A horizontal line representing the peak performance and a diagonal line defined by the maximum memory bandwidth, mark the limits of system performance due to hardware. A given kernel implementation with arithmetic intensity AI can be either memory- or compute-bound. The goal is to apply optimizations to move performance closer to the ridge point (intersection of the diagonal and horizontal lines), which corresponds to the highest performance at the lowest amount of resources.

In recent years, there has been an effort to extend the concept of Roofline analysis to FPGA accelerators especially in the HPC domain [2], [3], [4], [5], [6], [7]. This is because the number of compute resources such as programmable DSPs used for floating-point operations has significantly increased [8] making FPGAs appealing for HPC applications, in particular in terms of energy efficiency [9]. Moreover, the proliferation of HLS tools that facilitate FPGA programming using high level programming languages has widen FPGA deployment to developers in domains such as embedded computing with its abundance of emerging workloads such as AI at the Edge [10], Robotics [11], Security [12], Data Analytics, etc. Therefore, the Roofline model should be studied for such FPGA workloads.

In prior work, some Roofline models consider only the resources of a particular FPGA device and estimate a peak performance, either by calculating the number of computing units that fit into the FPGA [13] or through benchmarks [14]. The fully-programmable nature of FPGAs has led others to constructing specific models for each application, since the architecture depends on the algorithm ported to the FPGA [2].

Our work proposes a novel application-centric Roofline model for MPSoC FPGA platforms that considers the resources of the FPGA device, the operational latency and the structure of the kernel code and, hence, is more accurate than previous work. We evaluate our Roofline model on KinectFusion, a well-known visual Simultaneous Localization and Mapping (vSLAM) algorithm. vSLAM is the process of mapping a moving agent's (e.g. robot or drone) observed environment using an image sensor, while concurrently determining the agent's pose within that map. Our contributions are summarized as follows:

- We introduce a novel FPGA Roofline model that considers resource and latency constraints as well as the loop structure of a kernel to achieve a more accurate, but still intuitive and simple to construct Roofline ceiling.
- We extend our model to accommodate multiple kernels implemented in the FPGA fabric and/or in the CPU.
- We illustrate that the arithmetic intensity of an application may vary considerably across various inputs presenting an input-dependent rendition of our Roofline model.
- Finally, we evaluate the Roofline model and compare it with the previously proposed work by visualizing the impact on performance of a series of optimizations on KinectFusion.

II. Roofline models

A. Memory Bandwidth Ceilings

The first step to constructing the Roofline model of a MPSoc FPGA accelerator is to calculate the peak memory bandwidth between the accelerator and the CPU host memory. The peak memory bandwidth B_{peak} assumes that the host memory

communicates with the accelerator using the *bursting* mode sending a new beat of data every bus cycle without the need of continuous handshakes:

$$B_{peak} = \sum_{i=1}^{N} d_{width,i} \cdot f_{tran} \quad (1)$$

where *N* is the number of AXI ports, $d_{width,i}$ is width of port *i* of the AXI data bus, and f_{tran} is the maximum bus frequency.

The FPGA device used in our evaluation, the Zynq UltraScale+ ZCU102 Evaluation Board, features 8 I/O ports and has peak theoretical memory bandwidth 2400 MTransfers/sec × 64-bit, which is equal to 19.2 GB/sec. However, using the benchmark presented in [15], the peak achievable performance in ZCU102 was measured at 75% of the theoretical peak for the DDR memory, hence equal to 14.4 GB/sec.

A tighter bandwidth ceiling that corresponds to *random* memory accesses assumes that memory locations are individually and randomly accessed (i.e. no bursting) by the accelerator so that each new transaction has to be preceded by a handshake between the accelerator and the memory controller. For example, the ceiling of the achievable bandwidth for random memory accesses lies much lower than the peak bandwidth ceiling at only 0.17 GB/sec.

B. Computational Ceilings

1) Application-agnostic Roofline model

This model considers only the available FPGA resources (LUTs, DSPs, and FFs) without considering the applications and their implementation. The goal is to determine the peak integer/FP performance by calculating the maximum number of concurrent operations supported by the resources of the FPGA. We use both the integer and the FP add/sub operation to construct two separate peak performance ceilings for the corresponding operation type, since they are basic operations that utilize the minimum number of resources, and, hence maximize peak performance when executing multiple such operations in parallel on the device. We derive the number of FPGA resources required by a single add/sub operation for different implementations of the function that exploit either (i) both DSPs and LUTs or (ii) only LUTs [13]. The maximum number of integer/FP operations that can be executed concurrently on the FPGA is given by:

$$ops_{agnostic} = max\left(\min_t \frac{R_t}{r_{t,all}}, \min_t \frac{R_t}{r_{t,logic}}\right) \quad (2)$$

where R_t is the number of available resources of type t, $t \in$ {LUTs, DSPs, FFs} in the FPGA and $r_{t,all}$ and $r_{t,logic}$ denote the amount of resources of type t needed for a 32-bit addition for each of the implementations (i) and (ii), respectively. The peak performance is given by:

$$PP_{agnostic} = f_{op} \cdot ops_{agnostic} \cdot AF \quad (3)$$

where f_{op} is the operating frequency when the maximum number of add/sub operations is instantiated in the FPGA and *AF* is an adjustment factor used to account for routing resources. We use a default value of 0.8 according to [16].

2) Application-centric Roofline model

One disadvantage of the application-agnostic model is that the *ops/sec* metric might not always be an appropriate performance indicator. If an optimization reduces both the execution time and the number of operations, this ratio could potentially be lower than the baseline implementation despite the improved execution time. An example is approximate computing, where non-critical operations are eliminated to trade-off reduced output accuracy with increased performance. In addition, the application-agnostic approach provides unrealistically high roofline walls that correspond to unattainable performance levels.

To paint a more accurate picture of the performance we choose *Generated Results / sec* as the unit for the Y axis. Examples of generated results could be a cell update for a dynamic programming algorithm [2], and a pixel or a frame for an image processing or video algorithm. The first step of our procedure is to count the operations (*ops*) (FP or integer arithmetic, logic operations, comparisons and indexing) involved in generating the output. The peak performance in *Generated Results / sec* unit is given by:

$$PP_{centric} = f_{op} \cdot \frac{ops_{agnostic}}{ops} \cdot AF = \frac{PP_{agnostic}}{ops} \quad (4)$$

3) Application-centric Roofline model with latency constraints

Evidently, the peak performance of a kernel is achieved when all its loop iterations are executed concurrently, constrained by the resource availability (as in the first and second methods), and also by the latency of a loop iteration. A kernel may consist of multiple nested multi-path loops L with data dependencies between them or within operations in the same loop (Fig. 1). Since our objective is to compute the theoretical peak performance of the kernel, we assume all inner loops are fully unrolled by the compiler and that there are no data dependencies within and across loops, so that they can be scheduled to execute concurrently. The peak performance PP of the kernel is constrained by the operator with the highest latency across all possible execution paths since the kernel cannot be faster than its slowest operation. Note that we exclude paths that execute under error conditions since they may provide unrealistic performance ceilings.

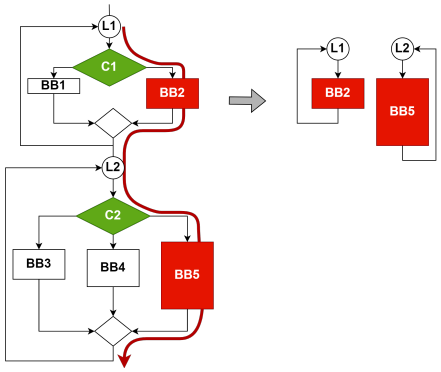


Fig. 1: Kernel that consists of two loops with multiple execution paths. The height of each basic block denotes its latency, which equals to the maximum latency among the operators in the respective block. Kernel latency is constrained by the blocks with the highest latency (in our case *BB2* and *BB5*).

We compute the ceiling as follows: first, we count the number of operations ops_{kernel} required by each kernel invocation. Let each operator $op \in \{+, -, *, /, \%, etc\}$ have a dynamic count

of c_{op}. Then:

$$ops_{kernel} = \sum_{\forall op} c_{op} \quad (5)$$

The percentage of the total kernel operations for each operator op is equal to

$$P_{op} = \frac{c_{op}}{ops_{kernel}}, \quad \forall op \quad (6)$$

We define ops_{max} as the maximum number of operators that fit in the FPGA, while preserving the relative (%) contribution of each operator op in the kernel. Therefore:

$$\sum_{\forall op} ops_{max} \cdot P_{op} \cdot r_{t,op} \leq R_t, \quad \forall t \quad (7)$$

where $r_{t,op}$ is the amount of resources of type *t* needed by operator *op* and is different from the one defined in Section II-B1. R_t is the available amount of resources of type *t* in the FPGA. Solving for ops_{max} and replacing the inequality we obtain:

$$ops_{max} = \frac{R_t}{\sum_{\forall op} P_{op} \cdot r_{t,op}}, \quad \forall t \quad (8)$$

Since Eq. 8 produces a different ops_{max} for each resource *t*, we choose the ops_{max} corresponding to the most constraining resource as:

$$ops_{max} = \min_t \frac{R_t}{\sum_{\forall op} P_{op} \cdot r_{t,op}} \quad (9)$$

Then, the number of *Generated Results* that fit in the FPGA is

$$GR = \frac{ops_{max}}{ops_{result}} \quad (10)$$

where ops_{result} is the number of operations in *Ops/Generated Result* involved in the generation of a result. Finally, the peak performance is computed as

$$PP = \frac{GR \cdot f_{op}}{l_{max}} \quad (11)$$

where l_{max} is the highest latency among all kernel operations.

4) Combined Roofline Model

Most complex applications typically consist of multiple kernels that should be optimized in tandem in order to achieve peak performance. Therefore, to gain a more realistic performance of our application, we can no longer assume that each kernel has the entire FPGA fabric for itself. To construct the combined roofline model, we calculate the operations ops_{total} in all kernels, summing the partial sums yielded by Eq.5. In order to maintain the same contribution of a kernel to total execution time, we regard each operation in each kernel separately. For example, we view a FP add in the bilateral filter as a different operation from a FP add in the integration kernel. Then, we apply Eqs. 6-11, with the key differences that (1) ops_{kernel} is substituted with ops_{total}, (2) $ops_{results}$ refers to the total operations in all kernels and (3) l_{max} to the operation with the highest latency across all kernels.

5) Input-dependent Roofline models

All Roofline models proposed in the literature assume that performance and AI are input-independent, or, in the best case, they consider the average values of these two metrics across the input set. To accommodate the most general case, we propose the input-dependent Roofline model which produces a different data point for each accelerator invocation. In the experimental evaluation section, we show that this model can be used to provide valuable insights for the behavior of the application that would not be available using the conventional input-agnostic models.

III. KinectFusion Algorithm

KinectFusion is a vSLAM algorithm that receives a constant stream of depth maps and uses it to track the camera pose and to update a global 3D model of an indoor scene [17]. Fig. 2 outlines the block diagram of KinectFusion. We briefly describe the functionality of each kernel below.

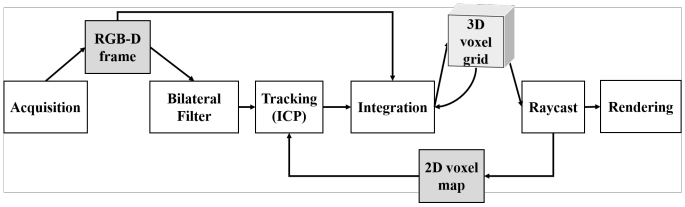


Fig. 2: KinectFusion data processing pipeline [18]

The **Bilateral Filter** is an edge-preserving, stencil-based blurring filter that reduces the impact of noise and invalid depth data by applying a convolution between the depth image and a 5x5 coefficients array. **Tracking** is based on the Iterative Closest Point (ICP) algorithm which estimates the 3D pose of the agent by registering the input depth frame with the 2D voxel map (the output of the raycast). **Integration** uses the output of the tracking to fuse the input depth frame into the 3D voxel grid, which represents the global map of the agent's environment. **Raycasting** is a well known computer graphics algorithm that converts 3D scenes into 2D images.

The integration and raycast kernels contribute more than 70% of total execution time when running the C++/OpenMP KinectFusion implementations on an 4-core ARM Cortex-A53 CPU [19].

A. Precise and Approximate Optimizations

Precise optimizations retain the accuracy of the baseline code. They include loop unrolling, loop pipelining, loop interchange to improve access locality, prefetching data from DRAM to the internal block RAMs to reduce latency, and instantiating multiple compute units to exploit task-level parallelism.

Approximate optimizations may negatively affect the accuracy of the baseline code. Approximate optimizations are critical in achieving real-time KinectFusion performance in an FPGA [19], [20]. Table I shows the approximate optimizations for each KinectFusion kernel.

B. Ranking of the Impact of KinectFusion Optimizations

The relative impact of KinectFusion optimizations on performance is quantified using Lasso [22], a regularized linear regression method that builds a model for coefficient selection.

Table II shows the features with the largest coefficients which indicate a stronger correlation between the corresponding feature and performance. Since using features with higher degree provides better model accuracy (lower mean square error, MSE), we choose to report features up to the second degree for each kernel.

TABLE I: Approximate optimizations of the KinectFusion kernels.

Optim. Name	Description
Bilateral Filter	
BF_Coeff	Use 3x3 coefficients instead of 5x5
BF_HP	Use 16-bit floating-point (fp16) arithmetic
BF_Range	No range filter (Eliminates an exponent function)
Tracking	
Tr_LP	Loop perforation (Skips loop iterations [21])
Tr_HP	Use fp16 arithmetic
Tr_LvlIter	Skip pyramid levels and reduce max. # of iterations
Integration	
Int_LP	Loop perforation
Int_HP	Use fp16 arithmetic
Int_Br	Eliminate checking of special conditions
Int_FPOp	Eliminate expensive FP ops (square, root-square)
Raycast	
R_Step	Larger ray steps
R_LP	Skip computing of rays (loop perforation)
R_TrInt	Use 2 points instead of 8 for trilinear interpolation
R_Fast	Use –ffast-math
R_Rate	Perform raycast every other frame

As expected, the most significant optimizations are related to loop transformations such as loop interchange, pipelining, and unrolling. For example, unrolling the inner loop of the Bilateral Filter (*BF_Unroll*), pipelining the inner loop of the Integration kernel (*Int_Pipe*) and applying loop perforation at the Raycast kernel (*R_LP*) are the most impactful optimization of these three kernels.

IV. Roofline Analysis

For our experiments, we use the KinectFusion implementation of the SLAMBench suite [18]. We target the Xilinx UltraScale+ MPSoC ZCU102 board which includes a quad-core 1.2 GHz ARM Cortex-A53 processor. The FPGA fabric is clocked at 300 MHz. As input, we use the 882-frame *lr.kt2* trajectory from ICL-NUIM [23]. In this section, we present the Roofline model for each kernel and for the whole application. We confirm the results of the analysis visually in the Roofline model by progressively and cumulatively incorporating optimizations in descending order of impact for each kernel according to Table II, and observe that the leaps in performance gradually become smaller.

A. Kernel Evaluation

Bilateral Filter. The rooflines in Fig. 3 visualize the performance gains yielded from precise and approximate optimizations on the Bilateral Filter accelerator. After unrolling the main loop (*BF_Unroll*), which also included row-wise streaming of the input frame to the FPGA BRAMs, memory accesses were considerably reduced and the design moved towards the compute-bound area in all three models (Fig. 3). The application-agnostic model failed, however, to capture the actual contribution of approximate optimizations to the throughput, validating our claim that *ops/sec* is not always indicative of performance. As already mentioned, approximate optimizations may reduce both the AI and the *ops/sec* which makes the latter a misleading indicator of their effect on throughput (remember that optimizations down the list are cumulative). For example, the *BF_Range* point is located below the loop pipeline (*BF_Pipeline*) point in the application-agnostic model, even though the former configuration has higher throughput. *BF/sec* is a more meaningful metric as confirmed by the two application-centric models. As shown in the two application-centric models, the most impactful optimizations according to Lasso analysis (*BF_Unroll* and *BF_Pipe*) provide the highest performance improvement from 0.54 Hz (baseline HW) to 328 Hz ($607x$ speedup). The remaining optimizations increase throughput to 624 Hz (just $1.9x$ incremental speedup), which almost touches the 722.84 Hz computational ceiling.

Similarly, Fig. 4 shows the Roofline model of the **Integration** kernel (only the approach of Section II-B3 is shown). The throughput of the Integration accelerators ranges from 0.72 Hz to 60.11 Hz. Unlike the BF filter which consists of a single data-parallel loop and, hence, can be thoroughly optimized, the Integration kernel has a multi-path loop and its theoretical peak performance PP is defined by a fast path that is executed infrequently.

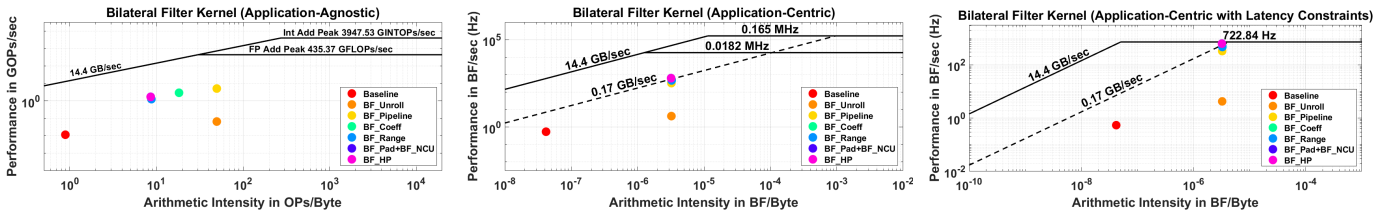
The input-dependent Roofline model of Fig. 5 indicates that there are large variations in performance and AI across frames (AI ranges from 5.72×10^{-7} to $26.6 \times 10^{-7} Integrations/byte$ in the fastest approximate implementation), hence, justifying the holistic approach to Roofline analysis that accommodates variations of the input data. An interesting remark inferred from the upward and rightward orientation of the data point trajectories of Fig. 5 is that higher AI generally corresponds to higher performance, validating that memory is the most constraining factor for the performance of the Integration accelerator and should be the first optimization target.

The **Raycast kernel** is executed on the ARM 4-core CPU because its irregular memory access pattern makes HW implementation very challenging. The difference between the (software) Roofline model of Fig. 6 and a hardware Roofline model is that for the former, performance is inversely proportional to the AI, indicating that for the host CPU the priority should be to minimize the computational demands. Moreover, the input-dependent model of Raycast in Fig. 7 portrays this exact association of the input with AI; data points that correspond to higher AI, typically exhibit lower performance.

The **combined** Roofline model of Fig. 8 merges both software and hardware ceilings. The peak ARM performance (red horizontal ceiling) was calculated by dividing 9.6 GFLOPs/sec with the number of operations executed in the Raycast kernel. The (red) memory bandwidth ceilings are derived from the specifications of the ARM processor. The dotted line is the bandwidth of random memory accesses for the kernels that are executed on hardware. Note that the peak performance at 58.03 Hz is much lower than the ceilings of the respective kernels, since we need to limit the resources assigned to each kernel to fit all of them in the FPGA (Section II-B4). Optimizations such as *Int_LoopInter* and *Int_Pipeline* rank very high in Lasso analysis, but exert a small impact on performance when applied alone. They act as enablers for the remaining optimizations, and no high performance implementation is possible without them.

TABLE II: Lasso analysis of KinectFusion throughput

Bilateral Filter (MSE=0.039)		**Integration** (MSE=0.0008)		**Raycast** (MSE=0.0016)		**Combined** (MSE=0.0048)			
Feature	Coeff.	Feature	Coeff.	Feature	Coeff.	Feature	Coeff.	Feature	Coeff.
BF_Unroll^2	0.126	Int_Pipe^2	0.184	R_LP	-0.145	Int_Pipe^2	0.085	Int_LP	-0.034
BF_Pipe*BF_Unroll	0.099	Int_Pipe*Int_Inter	-0.167	R_Rate	-0.110	Int_Inter*Int_Pipe	-0.056	Tr_Pipe*R_LP	0.028
BF_Pipe	-0.095	Int_Inter^2	0.156	R_Step	-0.057	Tr_Pipe^2	0.044	BF_Pipe*BF_Unroll	0.028
BF_Unroll*BF_Coeff	0.064	Int_LP	-0.120	R_Fast	-0.056	Int_Pipe*Int_Unroll	-0.043	Int_Unroll	0.026
BF_Coeff	-0.060	Int_Inter*Int_LP	0.113	R_LP*R_Rate	0.049	Int_Inter*Int_Unroll	0.039	BF_Pipe	-0.023
		Int_Unroll	-0.039	R_TrInt	-0.047	BF_Unroll^2	0.037	Int_Pipe*Int_LP	0.020



(a) (b) (c)

Fig. 3: (a) Application-agnostic, (b) application-centric, (c) application-centric with latency constraints Roofline models for the Bilateral Filter (BF) accelerator. Some data points might overlap and appear missing. Peak performance in (b) and (c) is in # BF invocations/sec (Hz). *BF_NCU* optimization uses N=2 BF accelerators. GOPs could denote GINTOPs or GFLOPs in Figures (a) and (b). KFusion uses GFLOPs.

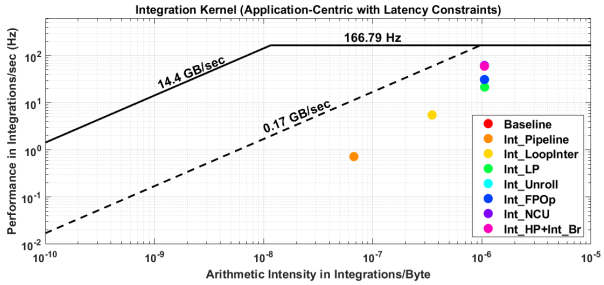


Fig. 4: Application-centric Roofline model with latency constraints for the Integration kernel. *Int_NCU* optimization uses N=4 accelerators.

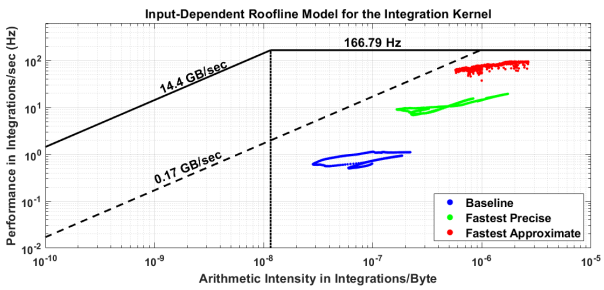


Fig. 5: Input-dependent Roofline model for the Integration kernel.

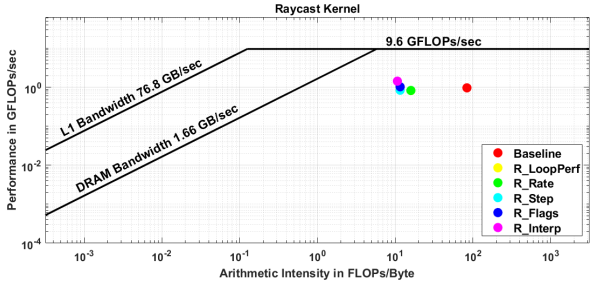


Fig. 6: Roofline model for the Raycast kernel.

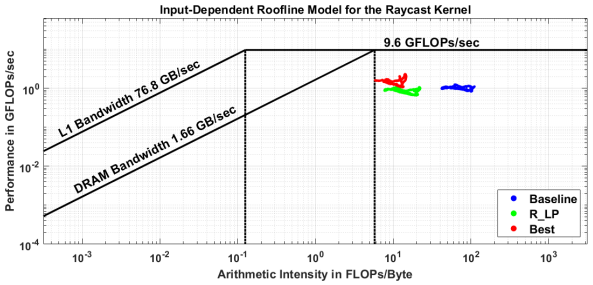


Fig. 7: Input-dependent Roofline model for the Raycast kernel.

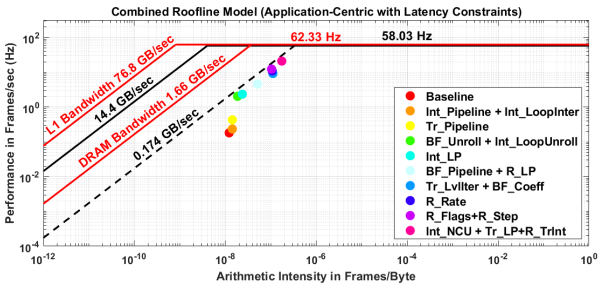


Fig. 8: Combined Roofline model.

V. Conclusions

We generalize Roofline analysis to the more general setting of multi-kernel MPSoC systems that combine CPU execution with HW accelerators. We evaluated two existing Roofline models and proposed a new application-centric model that considers the operation latency and imposes a more realistic bound on FPGA acceleration. We also proposed the input-dependent model outlining the importance of considering input-centric parameters when designing an optimization policy. We validated these concepts on a multi-kernel vSLAM application, and we indicated how our contribution can be used to better understand application characteristics.

REFERENCES

[1] Samuel Williams et al. Roofline: An insightful visual performance model for multicore architectures. Commun. ACM, 52(4):65–76, April 2009.

[2] Marco Siracusa, Marco Rabozzi, Emanuele Del Sozzo, Lorenzo Di Tucci, Samuel Williams, and Marco D. Santambrogio. A CAD-based methodology to optimize HLS code via the Roofline model. In IEEE/ACM International Conference On Computer Aided Design, (ICCAD) San Diego, CA, USA, November 2-5, 2020, pages 116:1–116:9.

[3] M. Yali. FPGA-Roofline: An Insightful Model for FGPA-based Hardware Acceleration in Modern Embedded Systems. 2015.

[4] Enrico Calore and Sebastiano Fabio Schifano. Performance assessment of FPGAs as HPC accelerators using the FPGA Empirical Roofline. In 31st International Conference on Field-Programmable Logic and Applications, (FPL) 2021, Dresden, Germany, August 30 - Sept. 3, 2021, pages 83–90.

[5] Enrico Calore and Sebastiano Fabio Schifano. Energy-Efficiency Evaluation of FPGAs for Floating-Point Intensive Workloads. In Proceedings of the International Conference on Parallel Computing, PARCO 2019, Prague, Czech Republic, September 10-13, 2019, volume 36, pages 555–564, 2019.

[6] S. Nabi and W. Vanderbauwhede. FPGA design space exploration for scientific HPC applications using a fast and accurate cost model based on roofline analysis. J. Parallel Distributed Comput., 133:407–419, 2019.

[7] Marco Siracusa, Emanuele Delsozzo, Marco Rabozzi, Lorenzo Di Tucci, Samuel Williams, Donatella Sciuto, and Marco Domenico Santambrogio. A Comprehensive Methodology to Optimize FPGA Designs via the Roofline Model. IEEE Transactions on Computers, 2021.

[8] Bajaj Ronak and Suhaib A. Fahmy. Mapping for Maximum Performance on FPGA DSP Blocks. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 35:573–585, 2016.

[9] Tan Nguyen, Colin MacLean, Marco Siracusa, Douglas Doerfler, Nicholas J. Wright, and Samuel Williams. FPGA-based HPC accelerators: An evaluation on performance and energy efficiency. Concurrency and Computation: Practice and Experience, 2021.

[10] Hsin-Yu Ting, Tootiya Giyahchi, Ardalan Amiri Sani, and Elaheh Bozorgzadeh. Dynamic Sharing in Multi-accelerators of Neural Networks on an FPGA Edge Device. 2020 IEEE 31st International Conference on Application-specific Systems, Architectures and Processors (ASAP), pages 197–204, 2020.

[11] Weikang Fang, Yanjun Zhang, Bo Yu, and Shaoshan Liu. FPGA-based ORB feature extraction for real-time visual SLAM. 2017 International Conference on Field Programmable Technology (ICFPT), pages 275–278, 2017.

[12] Kaspar Matas, Tuan Minh La, Nikola Grunchevski, Khoa Dang Pham, and Dirk Koch. Invited Tutorial: FPGA Hardware Security for Datacenters and Beyond. Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, 2020.

[13] Michael Parker. Understanding Peak Floating-Point Performance Claims. Intel White paper, 2017.

[14] Marius Meyer, Tobias Kenter, and Christian Plessl. Evaluating FPGA Accelerator Performance with a Parameterized OpenCL Adaptation of Selected Benchmarks of the HPCChallenge Benchmark Suite. In 2020 IEEE/ACM International Workshop on Heterogeneous High-performance Reconfigurable Computing, (H2RC@SC), Atlanta, GA, USA, November 13, 2020, pages 10–18.

[15] Kristiyan Manev, Anuj Vaishnav, and Dirk Koch. Unexpected Diversity: Quantitative Memory Analysis for Zynq UltraScale+ Systems. FPT, 2019.

[16] Russell Tessier and Heather Giza. Balancing logic utilization and area efficiency in FPGAs. In International Workshop on Field Programmable Logic and Applications, pages 535–544. Springer, 2000.

[17] Richard A. Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J. Davison, Pushmeet Kohli, Jamie Shotton, Steve Hodges, and Andrew W. Fitzgibbon. KinectFusion: Real-time dense surface mapping and tracking. In 10th IEEE International Symposium on Mixed and Augmented Reality, (ISMAR), October 26-29, pages 127–136, 2011.

[18] Mihai Bujanca et al. SLAMBench 3.0: Systematic Automated Reproducible Evaluation of SLAM Systems for Robot Vision Challenges and Scene Understanding. In International Conference on Robotics and Automation, ICRA, Montreal, QC, Canada, May 20-24, 2019, pages 6351–6358.

[19] Maria Rafaela Gkeka, Alexandros Patras, Christos D. Antonopoulos, Spyros Lalis, and Nikolaos Bellas. FPGA architectures for approximate dense SLAM computing. In Design, Automation & Test in Europe Conference & Exhibition, (DATE), Grenoble, France, February 1-5, 2021, pages 828–833.

[20] Quentin Gautier, Alric Althoff, and Ryan Kastner. FPGA Architectures for Real-time Dense SLAM. In 30th International Conference on Application-specific Systems, Architectures and Processors (ASAP), pages 83–90, 2019.

[21] Stelios Sidiroglou-Douskos et al. Managing Performance vs. Accuracy Trade-Offs with Loop Perforation. In ESEC/FSE, Szeged, Hungary, Sept., 2011.

[22] R. Tibshirani. Regression Shrinkage and Selection via the Lasso. Journal of the Royal Statistical Society (Series B).

[23] Ankur Handa, Thomas Whelan, John McDonald, and Andrew J. Davison. A benchmark for RGB-D visual odometry, 3D reconstruction and SLAM. In 2014 IEEE International Conference on Robotics and Automation, (ICRA), Hong Kong, China, May 31 - June 7, 2014, pages 1524–1531.