VGM-Bench: FPU Benchmark suite for Computer Vision, Computer Graphics and Machine Learning applications

Luca Cremona, William Fornaciari, Andrea Galimberti, Andrea Romanoni, and Davide Zoni

Politecnico di Milano, P.zza Leonardo da Vinci, 32, 20133, Milano, ITA
`name.surname@polimi.it`

Abstract. With the Internet-of-things revolution, embedded devices are in charge of an ever increasing number of tasks ranging from sensing, up to Artificial Intelligence (AI) functions. In particular, AI is gaining importance since it can dramatically improve the QoS perceived by the final user and it allows to cope with problems whose algorithmic solution is hard to find. However, the associated computational requirements, mostly made of floating-point processing, impose a careful design and tuning of the computing platforms. In this scenario, there is a need for a set of benchmarks representative of the emerging AI applications and useful to compare the efficiency of different architectural solutions and computing platforms. In this paper we present a suite of benchmarks encompassing Computer Graphics, Computer Vision and Machine Learning applications, which are greatly used in many AI scenarios. Such benchmarks, differently from other suites, are kernels tailored to be effectively executed in bare-metal and specifically stress the floating-point support offered by the computing platform.

Keywords: Benchmarks · Machine Learning · Artificial Intelligence · Floating-Point · FPU.

1 Introduction

In the Internet-of-Things era, the embedded computing platforms are in charge of an ever increasing number of tasks. Novel and complex applications ranging from the Artificial Intelligence (AI) to the computer graphics domain emerged, and they are constantly evolving to improve the QoS perceived by the final user. The computational power required by such tasks pushed to the limit the efficiency requirement for the embedded platforms that are in charge of their execution, since frequently such devices are battery-powered. In particular, most of the AI applications are dominated by floating-point arithmetic computations, which represent a major contribution to the energy consumption. Traditionally, the benchmark suites are employed to deliver a set of representative applications to compare the efficiency of different computing platforms in a way that is reproducible and consistent. Furthermore, they are the key factor enabling the design of energy saving methodologies for any class of computing devices, single-core [33], multi-cores [32] and HPC [23] architectures, or specific arithmetic accelerators [34].

The state-of-the-art reports different benchmark suites tailored to stress specific aspects of the computing platform. SPLASH-2 [29] and PARSEC [8] are employed to stress the multi-threading and multi-tasking capabilities of the computing platform, while SPEC applications [3] target single-threaded performance. Moreover, AI [27] and machine learning [17] benchmark suites are available for High-Performance Computing (HPC) platforms only. However, the critical drawback of any benchmark suite targeting AI applications is the abstraction layer, since each application in the suite assumes the presence of an Operating System as well as power-consuming resources, e.g., GPUs. The typical IoT scenario is different, there are RISC CPUs where few hardware accelerators are employed to offload the most intensive computational tasks from the main CPU. Moreover, the efficiency requirements impose to avoid the use of a fully-fledged OS, which is frequently replaced by a run-time library and a set of low-level peripheral drivers [2, 4]. A valuable benchmark suite for this scenario, should provide bare metal applications to stress the computing capabilities of the target embedded systems, with specific emphasis on the floating-point support, given its increasing relevance in nowadays AI applications.

Contributions - This paper illustrates a set of benchmarks targeting three of the most critical application domains in the IoT era: machine learning, computer graphics and computer vision. Instead of delivering complete applications for each class, every benchmark in the proposed suite targets a different computationally intensive brick of a possible bigger application that falls in one of the three considered classes. In a nutshell, our benchmark suite is *representative* of a wide set of popular application domains, can be employed on *bare metal* without the impact and disturbance of an operating system and, finally, the focus on kernels (micro-benchmarks) allows to dramatically reduce the *execution time* of the test, enabling the possibility of an effective *design space exploration*. Such lightweight configuration is also suitable to analyze low-end computing architectures, such those typically employed for IoT applications. The complete benchsuite, together with a detailed description of each application is available as open source at [1]. To show in practice the value of this benchmark suite, results are collected on a RISC-V System-on-Chip platform meant for IoT. In particular, we demonstrated the impact on the efficiency due to the amount of floating-point resources available on the target device. The rest of the paper is organized as follows. Section 2 examines the state-of-the-art benchmark suites for image processing and machine learning applications. Section 3 presents the proposed benchmark suite and describes the algorithms behind each application. Section 4 is an example of usage of the benchmark suite which analyzes the error rate of the application when varying the precision threshold and the number of truncated bits in the mantissa of their floating-point representation. Section 5 drawn some concluding remarks.

2 Related works

The state-of-the-art contains several benchmark suites targeting computing platforms ranging from the embedded to the HPC domains. In particular, each benchmark suite allows to assess different aspects of the target computing platform.

Considering high-end embedded platforms and HPC systems, PARSEC [8] and SPLASH-2 [29] applications are meant to stress the multi-threading and multi-tasking capabilities of the computing platform. Differently, SPEC-CPU2006 benchmarks [3] are used to assess single-threaded performance. We note that applications in those benchmark suites spread over different domains, since they are conceived for general-purpose computing platforms. However, their complexity and the requirement of running on top of an Operating System (OS) make such suites not viable for the assessment of both low-end embedded platforms and in-development devices. The former, i.e., low-end embedded devices, in general do not necessarily exploit an operating system while the latter, i.e., in-development devices, are usually executed by RTL simulators or on prototype boards for which the software ecosystem, e.g., OS and drivers, is not mature enough to support OS-based applications.

To this extent, several general-purpose benchmark suites emerged to assess different embedded system figures of merit. Such applications are usually written in ANSI-C and are meant to to be executed in bare-metal. [14] proposes the Worst-Case Execution Time (WCET) benchmark suite to assess the real-time performance of the considered computing platform. [15] presents a set of open source general-purpose applications, covering security, telecommunication, consumer and networking domains, for which the required computational power and the input size are optimized for the execution on embedded devices.

The AI momentum fueled the proposal of different benchmark suites and frameworks specifically focused on the machine learning, computer graphics and computer vision domains. CortexSuite [27] is a brain-inspired benchmark suite containing a set of applications and algorithms pertaining machine learning, natural language processing and computer vision, also including real-world datasets. The MLBench [17] benchmark suite specifically focuses on machine learning algorithms, such as the Bayes classifiers. However, the applications in both CortexSuite and MLBench are not suitable to be executed in bare-metal mode, thus preventing their use during the microarchitectural design stage in the hardware design flow.

To overcome such limitations, we present the *VGM-Bench* suite. It contains 12 kernel applications targeting three different application domains: computer vision, computer graphics and machine learning. Each application is written in ANSI-C and can be executed in bare-metal. Moreover, the default datasets are sized according to the need of executing the applications where the computing platform is either prototyped, i.e., FPGA-implemented, or still under design, hence simulated using an RTL simulator.

3 Benchmarks details

This section provides an overview of our bench-suite, organized according to the realm of application: Computer Vision, Computer Graphics and Machine Learning. We briefly describe the rationale behind each benchmark, its impact on the state of the art systems and its implementation. Moreover, we discuss whether a benchmark: (i) makes intensive use of floating-point computation and, (ii) traces back to a classification procedure, meaning that we expect it to be more resilient to floating-point errors. Apart from the description of its functionality, each benchmark is described in terms of two properties: floating-point intensiveness and type of application. The floating-point intensiveness measures the percentage of FP-instructions in the benchmark, while the type labels each benchmark as a classification application, or not. We note that the performance of FP-intensive applications strongly depends on the floating-point support, either hardware or software, offered by the computing platform. Moreover, classification applications are less sensitive to low-precision floating-point formats than non-classification ones.

3.1 Computer Vision

Computer Vision addresses the problem of processing one or more images to infer a wide variety of information from the captured scene. For this category, we present four applications which are especially useful while recovering 3D models from images, for instance for augmented reality, environment mapping or camera localization.

Zero-means Normalized Cross Correlation (ZNCC) - It is a widely adopted kernel to compute the correlation between the patches of two images patches. Such kernel is employed in Stereo Matching [11] and 3D reconstruction [22]. The proposed benchmark compares two patches considering of the fountain P-11 dataset [26] as default input set. Images are grayscale converted and, then, encoded as a bi-dimensional integer array where each element corresponds to a pixel whose value is in the interval $[0, 255]$. The expected output of ZNCC is a floating-point between -1 and +1. We note that the kernel procedure performs several FP sums and multiplications, therefore it is floating-point intensive.

Ray-Triangle Intersection - It is a fundamental kernel to estimate the color of each pixel in rendering applications and 3D volumetric reconstruction problems. Given a ray R originating from O and the triangle T, the idea is to apply a transformation such that the transformed ray R' is aligned with the x-axis and the transformed triangle T' lays on the yz-plane. Then, the ray-triangle intersection is simply given by the y z coordinates of the transformed point O'. If its corresponding barycentric coordinates are inside T' then the intersection test has success. The proposed benchmark considers 5 rays and 4 triangles and tests the ray-triangle intersection for each pair, totaling 20 tests. Each test has a binary output, thus making the kernel a classification application.

Kalman Filtering - It is a widely used estimation tool that aims at predicting a series of the state of a process taking as input a series of noisy measurements and the initial state. In Computer vision and Robotics, it plays an important role, especially when dealing with Simultaneous Localization and Mapping (SLAM) [12] and Parallel Tracking and Mapping (PTAM) [16] problems. To this extent, the benchmark mimics the localization case of study in which the state consists of the 3D position and the 3D velocity of a robot. We generated the ground truth trajectory with a constant velocity model, where the velocities are $(10unit/s, 5unit/s, 2unit/s)$ and the measurements are derived from those trajectory perturbed by a Gaussian noise $\nu \sim \mathcal{N}(0, 1)$. The kernel does not falls into the category of classifier kernels. Moreover, the majority of the instructions are due to matrix indexing operations, thus making the kernel an FP-mild one.

K Nearest Neighbor - Finding the K nearest neighbor of a query data $\mathbf{y} \in \mathrm{R}^n$ among a set of elements distributed in the R^n space is a key ingredient of many Computer Vision algorithms, e.g., when computations involve 3D point clouds. Given a set $\mathcal{F}$ of elements, i.e., features, $\mathbf{f} \in \mathcal{F} \subset \mathrm{R}^n$, and a distance function $dist(\cdot, \cdot)$, the k nearest neighbors of a query $\mathbf{y} \in \mathrm{R}^n$ is the set of features $(\hat{\mathcal{F}}_k = \mathbf{f}_1, ..., \mathbf{f}_k) \in \mathcal{F}$ such that $\nexists \mathbf{f} \in \mathcal{F} \setminus \hat{\mathcal{F}}_k and \mathbf{f}_k \in \hat{\mathcal{F}}_k, dist(y, f) < dist(y, \mathbf{f}_k)$. A simple and effective way to implement this algorithm is to store the features in an array and order the array up to the k-th position as in the selection sort method. The proposed kernel is fed with a set of 100 features and tests 20 query points. The output is a binary value for each query, i.e., found or not, thus making the algorithm a classifier. We also note that the kernel involves a balanced mix of floating point instructions together with indexing operations, i.e., integer instructions.

3.2 Computer Graphics

Computer graphics is the branch of computer science that creates, manipulates and stores geometric objects (modeling) and their images (Rendering). In the proposed bench-suite, we identify four key procedures belonging to this category.

Laplacian Smoothing - It is adopted as a fairing step of many computer graphics algorithm, extended by several methods and applied in variational mesh refinement algorithms to implement a smoothing energy term [22]. The proposed benchmark is fed with a deformed sphere made up of 77 vertices and 150 faces. The output is a set of new vertex positions obtained by the smooting process. In addition to this, even if the core of the application is essentially an average of float vertices positions, the application highly rely on indexing operations to retrieve the coordinates of the vertices to be processed at each step.

Facet Normals - Given a 3D mesh, its facets normals are exactly the normal vectors of the plane where each facet lays. They can be computed through the cross product applied for each facet of a mesh. This computation is conceptually easy however, normals constitute a fundamental building block of the computation of lighting in computer graphics rendering pipelines, [9]; they can also be the basis for vertex normal computation [18] which is, in turn, a relevant issue for many rendering operations. Moreover facets normals computation is fundamental during the mesh refinement operation proposed in most of of the works cited in the previous section This benchmark uses as input a sphere with 77 vertices and 150 faces. The expected output is an array of 3D vectors (no classification), 150 in our case, each vector is the normal of the facet with the corresponding index. Even in this case the number of indexing operation to access to vertices positions is significant compared to the floating point ones.

Facet Subdivision - Facet subdivision aims at refining the resolution of the mesh. Given a triangular mesh several approaches have been adopted to improve the resolution of the mesh while smoothing its shape [25]. However, in some situations, there is a need for disentangled resolution increase and smoothing. In such cases, a very common approach turns the facets of the mesh in a 1-to-4 pattern [28]; in this benchmark, we implemented the 1-to-4 pattern as follows. First, we copy the input vertices into the output vector of vertices V_{out}. For each vertex, we store a vector which is populated whenever a new vertex is added to split one of the adjacent facets. For each facet, and each of its three edges, we check from the vector associated to the ending vertex v_{min} whose index is smaller if the mid-point exists; otherwise, we create it and we add it to the last empty element of the vector associated with v_{min}. The four new triangles are then created by adding the indices of the original and the new vertices to I_{out} coherently, where I_{out} is the array of indices building up the facets of the output mesh.

The input of this benchmark is a portion of a sphere with 39 vertices and 58 facets, and the expect output a mesh with 135 vertices and 232 facets, therefore a mix of index (integer output) and vertices positions (float output). Facet subdivision involves many indexing operation both to access to vertices positions and to define the new vertices and their connectivity inside the new mesh; floating point operations is limited to the vertex averaging.

Self Intersections - Mesh self-intersections are those facets belonging to the same mesh and intersecting with each other. So, given mesh Γ containing a set of facets $\mathcal{F}$, we define two facets $f_1, f_2 \in \mathcal{F}$ as self-intersecting if they intersect each other; the set mesh self-intersections $\hat{\mathcal{F}}$ contains all facets f_i such that $\exists f_j \neq f_i \in \hat{\mathcal{F}}$ with which it has an intersection. Self-intersections are usually considered an issue especially since they are an unrealistic representation of a real-world surface. For this reason, several Computer Graphics algorithms, especially those modifying the mesh shape, requires the detection of self-intersection to get rid of them or preemptively avoid a certain mesh deformation that would produce them [30]. Similarly in Zaharescu et al.,[31] evolves a mesh to refine its appearance, and at the same time checks which are the self-intersections and they fix them. To implement self-intersection detection we check if, each facet intersects one of the others using the triangle-triangle intersection algorithm proposed by Möller [19]. The benchmark takes as input a mesh made up of 47 vertices and 72 facets. The mesh has been intentionally designed with 16 self-intersecting facets.

Therefore, the procedure performs binary classification for each facet. However the number of floating-point operation is significant, due to the triangle-triangle intersection test

3.3 Machine Learning

Machine learning is an application of artificial intelligence (AI) that provides systems the ability to automatically learn and improve from experience without being explicitly programmed. Machine learning focuses on the development of computer programs that can access data and use it to learn for themselves. VGM-Bench suite proposes four benchmarks to address this field.

Linear Regression - It is one of the most basic and relevant Machine Learning tools whose aim is to fit a linear model into a set of labeled data. Given a set $\mathbf{x}$ of input data and the corresponding $\mathbf{y}$ labels, the goal is to find a linear model $\mathbf{y} = \theta_1\mathbf{x} + \theta_2$ that is the better represents the input-label relationship, i.e., it minimizes the mean-squared error. Linear regression is one of the simplest machine learning methods which is the basis of some more complex approaches[20]. The benchmark is made up of 67 sets of 100 points where a linear 2D model has to be fit. The output of linear regression, as suggested by the name is float (no classification) and is composed by a good balance of indexing and floating points operations.

k-Means - It is a classical clustering algorithm widely adopted for data analysis or compression. Given a number K of clusters, and a set of input data $\mathbf{x}$, K-means aims at finding the position of the centroids of the clusters and to which each input data has to be associated. In the following, we describe the algorithm considering the input data as points in the space. the same procedure can be applied to whatever input data belonging to R^n. K-means is an Expectation-Maximization method that minimizes the function:

$$J = \sum_{i=1}^{m}\sum_{k=1}^{K} w_{ik} \left\| x^i - \mu_k \right\|^2 \tag{1}$$

where $w_{ik} = 1$ if point i is associated to cluster k, otherwise $w_{ik} = 0$, x_i the i-th input point and μ_k the k-th the cluster centroid. Therefore it alternates between an Expectation and a Maximization step. After initializing the position of each centroid μ_k, usually spread uniformly in the domain of the input data, in the Expectation step it minimizes J w.r.t. w_{ik} keeping μ_k fixed. In the maximization step, K-means minimizes J w.r.t. μ_k keeping w_{ik} fixed, i.e., In other words, it recomputes the centroids according to the new assignments in w_{ik}. In the benchmark, two clustering problems are reported with 5 centroids and 9 input points. Since the aim is to choose which point belongs to which centroid, k-means traces back to classification. Even in this case the procedure is made up of a balanced mix of floating point and indexing operations.

Q-learning - Q-Learning is one of the most popular algorithms of Reinforcement learning, i.e., the branch of Machine learning in which an agent interacts with the environment and it has to learn how to reach a specific goal, knowing that a reward is associated with certain actions. A reinforcement learning problem is often described as a Markov Decision Process. We define a set of states (S) a set of actions (A) that the agent can take, a transition probability $P(s, s')$ of going from s to s' and a reward function $R_a(s', s)$ of going from s' to s with an action a. Q-learning is a reinforcement learning algorithm that seeks to find the best action to take given a state; more in general, Q-learning looks for the policy $\pi(a, s)$, i.e., the probability that an action a has to be applied given a state s. To do so, a table state-action, named q-table, is kept updated as the agent explores the environment. In general, giving an agent starting in a state s_1, it takes an action a_1 and receives a reward r_1 and ends in a state s_2. To choose a_1 either the agent chooses randomly, or it considers the action that according to the q-table will give the highest expected reward. The choice between these two alternatives is the well-known trade-off between exploration and exploitation. This process is usually iterated multiple times for all the states until convergence. Let notice that the random number adopted to choose the random action, have been extracted in advanced and saved in an array, to obtain deterministic results. In the proposed benchmark we consider a Markov decision Process with 6 states, where the sixth is the goal. The input is just the Markov Decision Process (MDP) description and the expected output is the updated q-table. The computation involves many matrix multiplication therefore a significant amount of indexing operations, i.e., integer instructions, compared to the FP ones.

Long-Short Term Memory - Long-Short Term Memory (LSTM) Neural Network is a special kind of Recurrent Neural Networks (RNN) that in addition to the short term dependencies learned by RNN, it also aims at learning long-term ones [13]. Such networks usually process input sequences as strings of text, and output some kind of prediction, depending on the task and depending on the training data. The idea in LSTM is to have the so-called cell, where each element of the input is processed; then, it uses input, output and forget gates to learn dependencies among data. At the end of this process, to map the hidden state of dimension $h \times 1$ to the desired dimension d_o, e.g. a single predicted value, a linear layer is added after the output of the hidden layer. In the benchmark we implemented all matrix and vector operations, the *sigmoid*, the *tanh* and so the exponential function. In the proposed application we can find a neural network aimed for time-series predictions, where the input is a series of 20×1 vector ($d = 20$) elements, and we use a single hidden layer of dimension $h = 32$. To generate the data we followed the procedure described in [5] and we trained our model with PyTorch [21].

The output of LSTM is a prediction of the temporal series (no classification). LSTM involves numerous floating point operations but also significant amount of indexing operations, especially due to array and matrix multiplications.

4 Experimental Evaluation

As already mentioned, most of the nowadays IoT devices are battery-powered and cost-sensitive. Thus, during the design of the computing platforms, it is important to consider both the size and the computing power of all the components of the system.

In particular, one of the most critical components of the CPU is the FPU (Floating Point Unit). Especially in small embedded processors, its power con-

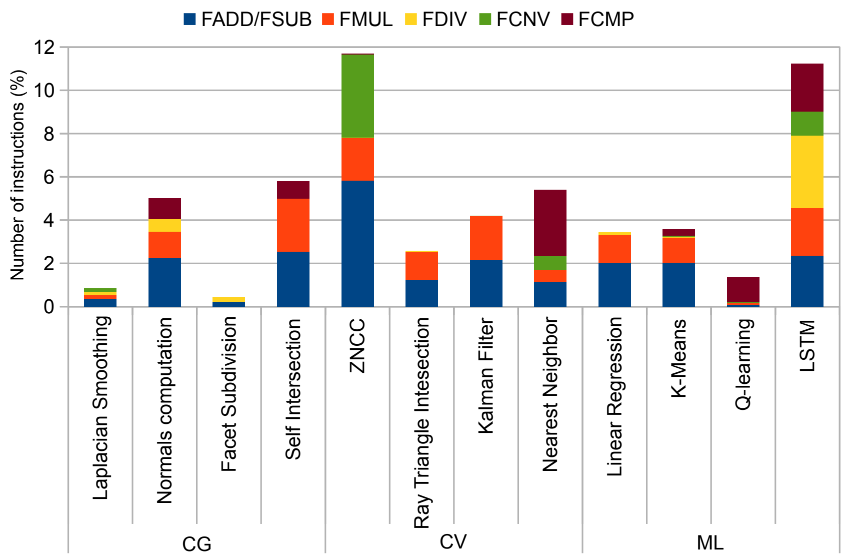
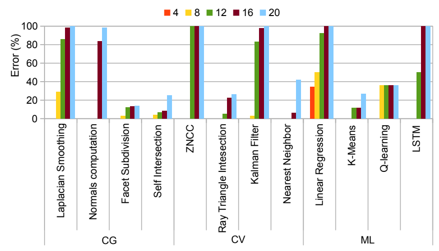


Fig. 1: Floating point instructions mix for the proposed benchmarks. Results are collected from a RISC-V compliant platform employing a custom CPU embedding the hardware FPU from the ORPSoC-v3 project.

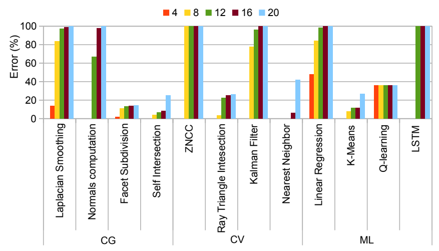
sumption can represent up to 40% of the overall count. As a consequence, it is crucial the sizing of the FPU operands, to prevent the cost of a full-size FPU implementation. To this purpose, we will show the benefit of using our suite to tackle this type of problem, by considering a representative use case. The analysis is split over two sections. The experimental setup is described in Section 4.1, while the experimental results are discussed in Section 4.2.

4.1 Experimental setup

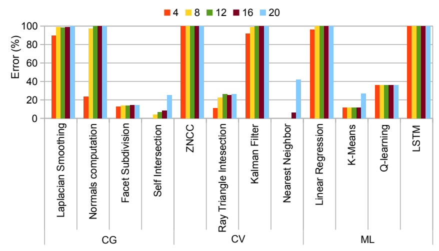
We employed the System-on-Chip (SoC) presented in [24] as our reference computing platform. The SoC embeds a fully-compliant RISC-V CPU that features a single-issue, in-order, 5-stage pipeline, and implements the Integer (I), Integer Multiplication and Division (M) and Single-Precision Floating-Point (F) RISC-V 32-bit ISA extensions. The floating-point unit adopted in this processor has been taken from the ORPSoC-v3 project and has been adapted to comply with the RISC-V extension specification. The applications have been simulated on the considered processor using the *xsim* RTL simulator, within the *Xilinx Vivado 2018.2* design suite, and by setting a 50MHz clock frequency. The floating-point instruction mix has been extracted through continuous monitoring of the performance counters implemented in the processor (more details can be found in [1])
.



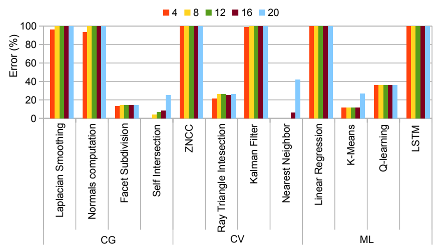
(a) Threshold $= 10^{-3}$



(b) Threshold $= 10^{-4}$



(c) Threshold $= 10^{-6}$



(d) Threshold $= 10^{-7}$

Fig. 2: Error rate of the benchmark applications when varying the precision threshold and the number of truncated bits in the mantissa of their floating-point representation.

4.2 Experimental results

Figure 1 shows the floating-point instruction mix for the bench-suite: for each application on the X axis, we plot the stacked percentages of the different types of floating-point instructions on the Y axis. Figure 2, instead, reports the number of errors generated by the applications when truncating the mantissa of the floating-point operands. This truncation is mimicking a possible reduction of the operand size to investigate if a smaller FPU can be adopted to reduce the size and power consumption of the device. Given a certain accuracy threshold varying from 10^{-3} to 10^{-7} ((a), (b), (c), (d)), each plot displays the normalized percentage of results not satisfy the desired accuracy. We consider 5 cases, truncating from 4 to 20 bits of the 23-bit mantissa.

From these two figures, we can notice that there is no correlation between the percentage of floating-point instructions in the application with the number of errors generated. For example, ZNCC has 11.8% of floating-point instruction against the 0.8% of Laplacian Subdivision, but the number of error generated is around 100% when truncating more than 8 bits of mantissa. The number of errors depends more on the application algorithm rather than on the number of truncated bits. For example let us consider a classifier, this applications returns a boolean or an integer (number of the class) value, e.g., in the case of Ray Triangle

Intersection, it returns true if the ray intersects the triangle, false otherwise. It doesn't consider the position of the intersection or the incidence angle, so even if there are errors due to the truncations, the final result, in most cases, can be correct anyway. This consideration becomes fundamental when there is a need to use an FPU dedicated for a specific type of algorithm: in this case the number of bits for the mantissa can be tailored to meet its specific requirements. Lets considering again, as an example, Facet Subdivision, Self Intersection, Ray Triangle Intersection, Nearest Neighbour and K-Means. We can notice that the number of errors does not increase significantly when decreasing the required precision. In fact, the average error percentage for these applications goes from close to 18% up to around 22%. The other applications, since their final outputs are floating-point values, are more sensitive to the reduction of the number of bits used to represent the mantissa of the operands, pulling the number of errors rapidly up to the maximum value. Thanks to the use of our benchmark suite exploiting selected micro-kernels, this design space exploration has been carried out in less than two hours on a Xeon E5-2650 V3 running @2.3GHz.

5 Conclusions

In this paper we propose *VGM-bench*, a benchmark suite containing bare-metal applications coming from computer vision, computer graphics and machine learning areas. These applications aim to stress the whole hardware architecture of the target computing platform, with a special focus on the floating point unit, and can be particularly useful during the prototyping phases of a design. To show the effectiveness of VGM-bench, the full suite has been simulated on a 32-bit RISC-V CPU [24] leveraging Xilinx Vivado 2018.2 design suite, to perform an extensive design space exploration. In addition to the analysis of the mix of floating-point instructions generated by the proposed benchmarks, we explored the impact of different mantissa truncations on the results accuracy, since this aspect is related to the cost and power consumption of embedded processors. Such design optimization has taken less than 2 hours of simulation on a mid-range desktop, thus making such optimization step fully affordable. The source code of the bench-suite is available for free download at [1], together with a complete description of each application.

6 Acknowledgments

Work supported by the H2020 FET-HPC project “RECIPE”, G. A. no. 801137. More information can be found in [7], [6] and [10].

References

1. Lamp: Lightweight application-specific modular processor, http://www.lamp-platform.org

2. Freertos (2003), https://www.freertos.org
3. Spec cpu2006 (2006), https://www.spec.org/cpu2006/
4. Mbed (2009), https://www.mbed.com/
5. Lstms for time series in pytorch. https://www.jessicayung.com/lstms-for-time-series-in-pytorch/ (2019), accessed: 2019-08-05
6. Agosta, G., Fornaciari, W., Atienza, D., Canal, R., Cilardo, A., Flich Cardo, J., Hernandez Luz, C., Kulczewski, M., Massari, G., Tornero Gavilá, R., Zapater, M.: The RECIPE Approach to Challenges in Deeply Heterogeneous High Performance Systems. Microprocessors & Microsystems (2020)
7. Agosta, G., Fornaciari, W., Atienza, D., Canal, R., Cilardo, A., Hernndez, C., Kulczewski, M., Massari, G., Gavil, R., Zapater, M.: Challenges in deeply heterogeneous high performance systems. In: 2019 22nd Euromicro Conference on Digital System Design (DSD) (08 2019). https://doi.org/10.1109/DSD.2019.00068
8. Bienia, C.: Benchmarking Modern Multiprocessors. Ph.D. thesis, Princeton University (January 2011)
9. Fan, W., Wang, K., Cayre, F., Xiong, Z.: 3d lighting-based image forgery detection using shape-from-shading. In: 2012 Proceedings of the 20th European Signal Processing Conference (EUSIPCO). pp. 1777–1781. IEEE (2012)
10. Fornaciari, al.: Reliable power and time-constraints-aware predictive management of heterogeneous exascale systems. In: Mudge, T.N., Pnevmatikatos, D.N. (eds.) Proceedings of the 18th International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation, Pythagorion, Greece, July 15-19, 2018. pp. 187–194. ACM (2018). https://doi.org/10.1145/3229631.3239368, https://doi.org/10.1145/3229631.3239368
11. Geiger, A., Roser, M., Urtasun, R.: Efficient large-scale stereo matching. In: Asian conference on computer vision. pp. 25–38. Springer (2010)
12. Geneva, P., Maley, J., Huang, G.: An efficient schmidt-ekf for 3d visual-inertial slam. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 12105–12115 (2019)
13. Greff, K., Srivastava, R.K., Koutník, J., Steunebrink, B.R., Schmidhuber, J.: Lstm: A search space odyssey. IEEE transactions on neural networks and learning systems **28**(10), 2222–2232 (2016)
14. Gustafsson, J., Betts, A., Ermedahl, A., Lisper, B.: The mälardalen wcet benchmarks - past, present and future. In: Proceedings of the 10th International Workshop on Worst-Case Execution Time Analysis (July 2010)
15. Guthaus, M.R., Ringenberg, J.S., Ernst, D., Austin, T.M., Mudge, T., Brown, R.B.: Mibench: A free, commercially representative embedded benchmark suite. In: Proceedings of the Workload Characterization, 2001. WWC-4. 2001 IEEE International Workshop. pp. 3–14. WWC '01 (2001)
16. Klein, G., Murray, D.: Parallel tracking and mapping for small ar workspaces. In: Proceedings of the 2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality. pp. 1–10. IEEE Computer Society (2007)
17. Leisch, F., Dimitriadou, E.: mlbench: Machine Learning Benchmark Problems (2010), r package version 2.1-1
18. Max, N.: Weights for computing vertex normals from facet normals. Journal of graphics tools **4**(2), 1–6 (1999)
19. Möller, T.: A fast triangle-triangle intersection test. Journal of graphics tools **2**(2), 25–30 (1997)
20. Montgomery, D.C., Peck, E.A., Vining, G.G.: Introduction to linear regression analysis, vol. 821. John Wiley & Sons (2012)

21. Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., Lerer, A.: Automatic differentiation in pytorch (2017)
22. Romanoni, A., Matteucci, M.: Mesh-based camera pairs selection and occlusion-aware masking for mesh refinement. Pattern Recognition Letters **125**, 364–372 (2019)
23. Sansottera, A., Zoni, D., Cremonesi, P., Fornaciari, W.: Consolidation of multi-tier workloads with performance and reliability constraints. In: 2012 International Conference on High Performance Computing Simulation (HPCS). pp. 74–83 (2012). https://doi.org/10.1109/HPCSim.2012.6266893
24. Scotti, G., Zoni, D.: A fresh view on the microarchitectural design of fpga-based risc cpus in the iot era. Journal of Low Power Electronics and Applications **9**, 19 (02 2019). https://doi.org/10.3390/jlpea9010009
25. Shiue, L.J., Jones, I., Peters, J.: A realtime gpu subdivision kernel. In: ACM Transactions on Graphics (TOG). vol. 24, pp. 1010–1015. ACM (2005)
26. Strecha, C., von Hansen, W., Van Gool, L., Fua, P., Thoennessen, U.: On benchmarking camera calibration and multi-view stereo for high resolution imagery. In: Computer Vision and Pattern Recognition. pp. 1–8. IEEE (2008)
27. Thomas, S., Gohkale, C., Tanuwidjaja, E., Chong, T., Lau, D., Garcia, S., Taylor, M.B.: CortexSuite: A Synthetic Brain Benchmark Suite. In: International Symposium on Workload Characterization (IISWC) (Oct 2014)
28. Vu, H.H., Keriven, R., Labatut, P., Pons, J.P.: Towards high-resolution large-scale multi-view stereo. In: Computer Vision and Pattern Recognition. pp. 1430–1437. IEEE (2009)
29. Woo, S.C., Ohara, M., Torrie, E., Singh, J.P., Gupta, A.: The splash-2 programs: Characterization and methodological considerations. In: Proceedings of the 22nd Annual International Symposium on Computer Architecture. pp. 24–36. ISCA '95
30. Xu, J., Sun, Y., Zhang, L.: A mapping-based approach to eliminating self-intersection of offset paths on mesh surfaces for cnc machining. Computer-Aided Design **62**, 131–142 (2015)
31. Zaharescu, A., Boyer, E., Horaud, R.: Transformesh: a topology-adaptive mesh-based approach to surface evolution. In: Asian Conference on Computer Vision. pp. 166–175. Springer (2007)
32. Zoni, D., Cremona, L., Fornaciari, W.: All-digital control-theoretic scheme to optimize energy budget and allocation in multi-cores. IEEE Transactions on Computers **69**(5), 706–721 (2020). https://doi.org/10.1109/TC.2019.2963859
33. Zoni, D., Cremona, L., Fornaciari, W.: All-digital energy-constrained controller for general-purpose accelerators and cpus. IEEE Embedded Systems Letters **12**(1), 17–20 (2020). https://doi.org/10.1109/LES.2019.2914136
34. Zoni, D., Galimberti, A., Fornaciari, W.: Flexible and scalable fpga-oriented design of multipliers for large binary polynomials. IEEE Access **8**, 75809–75821 (2020). https://doi.org/10.1109/ACCESS.2020.2989423