Hardware-aware 3D Model Workload Selection and Characterization for Graphics and ML Applications

Ruihao Li, Aman Arora, Sikan Li, Qinzhe Wu, Lizy K. John
Department of Electrical and Computer Engineering
The University of Texas at Austin
liruihao@utexas.edu, aman.kbm@utexas.edu, sikanli@utexas.edu, qw2699@utexas.edu, ljohn@ece.utexas.edu

***Abstract*—3D models are widely used in computer graphics, computer vision, and robotics applications. Multiple hardware accelerators are used for running 3D model related applications, since the computations required for models in 3D space are an order of magnitude higher than the computations in 2D space. Due to the high computation intensity of 3D model workloads, using large 3D model datasets for performance characterization is not a feasible choice during accelerator design. Representative subsets are widely used to save the execution or simulation time, e.g, ModelNet10, a subset of ModelNet40, is widely used in the machine learning (ML) domain to save training and inference time. However, this subset is picked by programmers from a software and application perspective.**

In this paper, we deploy statistical analysis based methodologies to guide the identification of hardware-aware representative subsets, which can maintain higher performance accuracy and achieve larger execution time savings with respect to subsets picked by software programmers. We believe that the methodology proposed in this paper can help hardware architects and engineers design efficient graphics or ML accelerators rapidly.

I. Introduction

3D models are widely used in today's multi-media devices and applications. In CAD (Computer Aided Design) domains, 3D models can be the input for devices like 3D printers. 3D models can also be edited in CAD softwares like Autodesk AutoCAD, for industry and architecture designs. Intensive interactions between 3D models exist in video games, which is a sub-discipline of computer graphics. Real scenes are abstracted as 3D models in the field of AR (Augmented Reality) and VR (Virtual Reality). In addition to these traditional fields, 3D models are also used in ML (machine learning) applications, including PointNet [19], PointNet++ [20], O-CNN [25], and PointCNN [12]. These neural networks take the points in 3D models as inputs and extract features of each model for shape classification and segmentation.

In computer graphics and other solid modeling applications, 3D models are usually described as polygon meshes [4]. In a polygon mesh, each point of the model is allocated with specific coordinates, resulting in intensive arithmetic operations when processing large 3D models. Considering the potential computations between pairs of points, the time complexity could be $O(|V|^2)$. $|V|$ represents the number of points, and modern 3D model could contain thousands of points, which is an order of magnitude higher than 2D models [14]. As a result, efficient hardware accelerators with high performance and low energy consumption have been developed in a lot of domains, including VR [28], 3D rendering [27], 3D perception [29], and visualization [11].

Considering the high-intensity computations, when developing hardware accelerators, the time to run simulations on the entire dataset can be prohibitively long, forcing a selection of one representative subset [17]. In Tigris [29], only the first 11 sequences of the entire dataset were used in the evaluation experiments. ModelNet10 [26], a subset of ModelNet40, is also commonly used in the ML domain to save both training and inference time. However, a dataset thus chosen may not be the right dataset for hardware accelerator design because the dataset may not exercise all hardware features like cache behavior, compute unit utilization, etc, and may not expose all the performance bottlenecks. For example, ModelNet10 is chosen by programmers from a software and application perspective. This subset can maintain the recognition/classification/segmentation accuracy compared to ModelNet40 in software applications but cannot guarantee representativeness in hardware performance.

3D models cannot be selected arbitrarily when characterizing workloads or designing hardware accelerators. For example, in 3D video games (computer graphics domains), when processing the collision detection between different objects, the computing intensity for models with complicated surface is higher than models with simple surface. In the field of visualization [11] and 3D model recognition [25], octrees are widely used to group adjacent nodes or extract the surface of the model to shrink the computation. With a balanced octree, it is easier to achieve better performance in hardware accelerators like GPUs. Computations are evenly distributed to each PE (Processing Engines) inside the accelerators, which helps avoid hardware resource under utilization. To guide selecting a hardware-aware representative subset of 3D models, a rigorous methodology is required that would enable rapid and efficient design of accelerators.

To that end, the following are the contributions of this paper:

- We characterize a popular graphics dataset used for robotics, AR/VR and machine learning applications called ModelNet40, and create a hardware-aware representative subset, called ModelNet7H, for GPU explorations.
- We characterize the performance on three existing GPU-CPU hardware platforms that can help guide architects

and designers to efficiently design future architectures.

- We use multiple workloads (two non-ML based and two ML based) to compare the difference between subsets selected from a software/hardware perspective.
- Case studies are designed for the two ML workloads, to show how batch size and hardware platform could affect the performance.
- We compare the difference between ML and non-ML workloads by analyzing the raw performance metric values.

II. Background

When applications expand from 2D space to 3D space, the computing intensity increases by orders of magnitude. The time for running such 3D applications also increases due to the high memory and computation cost [26], [13]. Both prototyping on cycle-accurate hardware simulators and characterization on real hardware platforms suffer due to the single-thread execution limitation of simulators [3] and limited numbers of hardware counters on real platforms. As a result, in performance related studies, a subset is usually used to reduce profiling or simulation time [17].

The methodology for identifying the representative subset we use in this paper is different from the one used in traditional benchmark suites, like SPEC [16], [17]. In SPEC, the performance of each workload is either independent of the input size or linearly correlated to input size. In other words, if the input size is doubled, the execution time should increase to $2\times$ correspondingly, assuming the same cache and control behaviors. As a result, prior works focus on the 'hot' regions ('hot' regions are the representative sections of a program, which could reveal bottlenecks with a shorter execution time, e.g., simpoints [23]) of the program itself instead of the inputs.

In this paper, we focus on the 'hot' regions of the 3D model input datasets instead of the complied binaries. Most 3D model based algorithms are heuristics and the program runtime behaviors cannot be predicted without the input information. The behavior of the branch predictor, prefetcher, and cache could differ dramatically when facing different inputs, like sparse models versus dense models, or models with smooth boundaries versus irregular boundaries. We group models from a hardware perspective and demonstrate how each group differs from each other.

III. Datasets and Workloads

To support 3D applications, large 3D datasets are developed. ModelNet40 [26] is a comprehensive clean collection of 3D CAD models developed by Princeton, which contains 12,311 3D CAD models belonging to 40 unique object categories. In this paper, we focus on this dataset to demonstrate the workflow of our methodology. We consider four (two non-ML and two ML) open source workloads, which can take ModelNet40 as inputs, to characterize the performance on hardware.

Non-ML Workloads We use two workloads in the point cloud library (PCL) [22]. PCL is a large scale, collaborative, and open source project for 2D/3D processing, and is widely used in the robotics community. In PCL, 3D models are usually stored in kdtree or octree data structures for better memory efficiency, query speed, and structural simplicity, which are commonly used in graphics [8], machine learning [25], and HPC domains [5]. PCL has GPU support, and we characterize the following two fundamental tree based algorithms.

1) Radius Neighbor Search: Radius Neighbor Search is a variant of the k-nearest neighbor search problem [2]. It would require a target point array, a search radius, and a maximum number of answers. The octree building and neighbor searching are done by separate kernels. The tree building kernel takes a point cloud (one pcd file) as input.

2) Segmentation: 3D point cloud segmentation is the process of classifying one point cloud into multiple homogeneous regions, the points in the same region will have the same properties [15]. In PCL, the *Segmentation* takes a point cloud as input and the computation kernels follow the same pattern as the *Radius Neighbor Search*.

ML Workloads We also include two state-of-the-art 3D-model based ML frameworks in our experiments. Different from non-ML workloads, ML workloads usually have two phases, training and inference. Inference is usually performed on edge devices and is more common in daily life scenarios. Unless otherwise specified, we consider only inference in this paper. In Section VII-B, we enhance our evaluation by considering Training in a case study.

3) PointNet: PointNet is a uniformed architecture that can perform object classification, part segmentation, and scene semantic parsing on 3D models. The basic PointNet architecture consumes original point clouds as input and only uses multi-layer perceptron (MLP) and max pooling layers for inference.

4) O-CNN: O-CNN is an octree based CNN for 3D shape analysis [25]. O-CNN first abstracts the 3D models using the octree data structure. Then back-end CNNs will only perform operations on octrees, which depict boundaries of each 3D model, rather than the entire model. Therefore, the computations in O-CNN will shrink dramatically compared with the voxel-based 3D CNNs [26].

IV. Characterization Methodology

A. Dataset Granularity

The ModelNet40 dataset has 12,311 models that are organized into 40 categories. In each category, a single model can also be accessed arbitrarily. We will analyze the dataset based on two levels of granularity, the coarser granularity (one entire category is considered as one element) and the finer granularity (one single 3D model is considered as one element).

We use the coarser granularity in the two ML workloads, since inputs are usually processed in batches in ML. For example, 256 images are processed in one batch in ResNet [9] and 32 3D models are processed in one batch in O-CNN [25]. As a result, GPU resources can be fully utilized since inputs

TABLE I: Characterization metrics

Categories	Metrics	Normalized Unit
GPU		
CPI	Cycle per instruction	Per instruction
Memory	Read Bytes, Read Transactions, Write Bytes, Write Transactions	Per 1K instructions
Inst Mix	Control Ins, Bit Convert Ins, Load/Store Ins, Floating Point Ins, Integer Ins	Percentage
Utilization	Control Flow FUs, Double Precision FUs, Half Precision FUs, DRAM	$0 \sim 10$
CPU		
CPI	Cycle per instruction	Per instruction
Inst Mix	Kernel Ins, User Ins, Branch Ins, Memory Load Ins, Memory Write Ins	Percentage
Cache&TLB	L1-dcache Misses, L1-icache Misses, LLC Misses, iTLB Misses, dTLB Misses	MPKI

in a batch are processed in parallel. In our experiments, we use 32 3D models as the default batch size.

We use both coarser and finer granularity in the two non-ML workloads. The computation intensity in ML workloads, especially CNNs [21], is usually higher than non-ML workloads. When non-ML based workloads are deployed in real time systems, the latency instead of throughput is a more important factor and inputs are usually processed one by one, which is the case of the finer granularity [10]. Besides the finer granularity, the coarser granularity can also be applied in non-ML based workloads by aggregating the results of all 3D models in one category.

B. *Performance Metrics*

We characterize the workloads on the CPU-GPU heterogeneous system on both ISA and microarchitecture events. There is a trade-off in the performance metrics selection. The more metrics we use, the more accuracy can be obtained for performance modeling with a longer profiling time. In addition, there are also some architecture specific metrics, e.g, AVX instructions, which does not exist in other platforms. Inspired by prior works [16], [18], [7], we measured 25 metrics in total (details metrics listed in Table I).

For ISA characterization, we examine the instruction mix, considering the potential large variance between different 3D input models. More control instructions are contained in spare 3D models with irregular boundaries and more computation instructions are included in dense 3D models with smooth boundaries. We also include the CPI (cycles per instruction) and utilization in our metrics. The batch size of ML workloads affects the resource utilization drastically and so, we characterize the performance on different batch sizes. The memory behaviors also need to be monitored, since the on-chip memory capacity is not enough to store the huge amount of intermediate results generated in 3D model applications [11]. The data movement between DRAM and on-chip memory in GPUs, and Cache & TLB behaviors in CPUs can reflect the memory behaviors.

C. *Hardware & Software Configuration*

To characterize the performance of the workloads for the next generation accelerator chips, simulators and existing hardware platforms are two available options. It is very difficult to directly deploy these workloads on simulators because of missing support of ML frameworks and other graphics libraries on simulation platforms. Instead, we use three state-of-the-art GPU platforms (Nvidia GTX 1080Ti, Tesla P100, and Tesla V100) for characterising these workloads. The CPU hosts are Intel Xeon E5-2620 v4 and Xeon Platinum 8160. The hardware configuration details are listed in Table II.

TABLE II: Hardware configurations

Platform	Name	Host	Memory
GPU	GTX 1080Ti	Xeon E5-2620	11GB GDDR5X
GPU	Tesla P100	Xeon Platinum 8160	16GM HBM2
GPU	Tesla V100	Xeon Platinum 8160	16GM HBM2
CPU	Xeon E5-2620	N/A	128GB DDR4
CPU	Xeon Platinum 8160	N/A	192GB DDR4

TABLE III: Loading factors of the top 3 metrics on the four principal components (O-CNN, P100)

PC1 (0.471)		**PC2 (0.225)**	
Metric	Load	Metric	Load
dTLB_mpki	0.308	inst_fp	0.419
L1_dcache_mpki	0.306	inst_bit_convert	-0.396
inst_kernel	0.305	inst_control	-0.379
PC3 (0.116)		**PC4 (0.071)**	
Metric	Load	Metric	Load
dram_write_transactions	-0.455	gpu_cpi	-0.640
dram_write_bytes	-0.455	dram_read_bytes	0.376
inst_compute_ld_st	-0.453	dram_read_transactions	0.376

The performance on the next generation hardware can be predicted using current existing platforms [30]. In this case, we consider the E5-GTX and Platinum-P100 platforms as existing platforms and consider the Platinum-V100 platform as the next generation hardware. We used the characterization results on the Platinum-P100 (Platinum-P100 is the default platform if we did not specify) system for selecting the subset of ModelNet40. We cross validate the results and also perform several case studies on the Platinum-V100 system. A relative old platform, E5-GTX, is used as the baseline machine, which serves the same function as the Sun reference machine in SPEC to help normalize the execution time [24].

We use the Linux *perf* [1] tool for collecting CPU microarchitecture hardware performance counters, and Nvidia *nvprof* [6] for collecting GPU related events. We use Python 3.7, and Pytorch 1.6.0 for ML based workloads, GCC 7.3.0, and PCL 1.11 for non-ML based workloads, and CUDA 10.1 for both ML and non-ML workloads.

V. Redundancy in Hardware Metrics

In this section, we will discuss methodologies for finding the potential redundancy of our defined characterization metrics. We will use the PCA (Principal Component Analysis) technique similar to prior works [16], [18], [7] to find potential redundancy within the 25 metrics in Table I.

Characterization metrics are supposed to be independent of each other and able to cover the variances contributing to the performance differences. However, in real cases, some metrics are correlated to some extent, which means there are redundancies. In other words, if the value of one metric changes, the value of other metrics will change correspondingly. For example, an increase in the percentage of control instructions can lead to a reduction in other instructions.

High-dimension data, in which correlations exist between different dimensions, can be down sampled to lower dimensions with the PCA technique, as well as cover most of the variance [18]. The same technique can be used to reduce the redundancy in our characterization metrics. We pick the top four PCs since four PCs can cover the majority of the variance in benchmark suites [18]. The loading factors of the first four PCs are listed in Table III in descending order with the variance of each PC also listed. Some loading factors are negative because we perform data standardization before PCA. Due to space limitation, we use the O-CNN workload deployed on the Platinum-P100 system as an example and only list the top 3 factors. The top 4 PCs we selected cover 88.2% of the variance.

VI. Representative Subsets

This section explains the methodology we used to identify the representative model subsets with both coarser and finer granularity.

To identify the hardware aware representative subsets, the first step is to quantify the relationship between each category/model in the ModelNet40 dataset. In this case, we use the linkage distance between the first four PCs of each category/model in ModelNet40. Euclidean distance is used to generate the linkage distance matrix. Once the similarity between each category/model is quantified, hierarchical clustering can be used to group the entire dataset into several clusters. The representative subset can be obtained by selecting one category/model from each cluster.

A. Subsets with the Coarser Granularity

When using a coarser granularity, we select an 8-category subset out of the 40 categories in the entire ModelNet dataset using hierarchical clustering [16], [17]. Considering ModelNet10 uses 10 categories, we intend to use a smaller number of categories for this dataset, while maintaining a higher hardware representative accuracy. The size of the subset is flexible and it could be changed based on different demands.

Figure 1 shows the hierarchical clustering results. We use the O-CNN deployed on the Platinum-P100 platform as the example due to the space limitation. In Figure 1, each leaf node represents one ModelNet40 category. Two nodes with the shortest linkage distance merge into a parent node recursively until the root node is generated. For example, a 2-element representative subset can be generated by selecting one category out of the first 24 (from *plant* to *bathtub*) and one out of the last 16 (from *bench* to *person*).

To identify a more representative subset among all 8-category subsets, we introduce the variable **score** to quantify the performance accuracy and representative of the selected

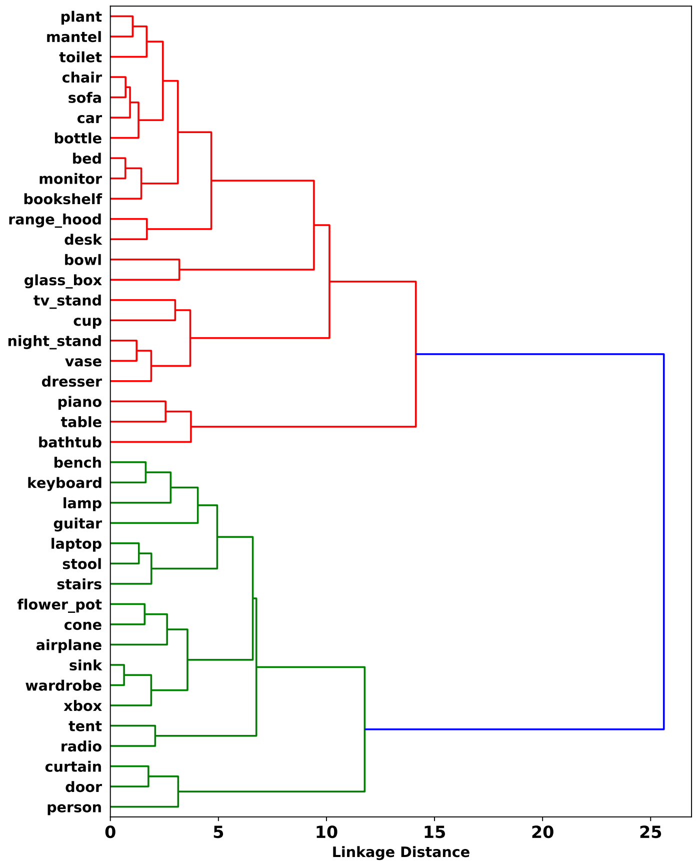


Fig. 1: A dendrogram showing the similarity between different categories in O-CNN (Platinum-P100)

subsets. The **score** is an execution time based metric [7], [16], since the purpose of our subset is performance prediction during the chip/accelerator design. In this case, the execution time (could be throughput or other metrics on other workloads) is the best metric to indicate the effectiveness of the subset selection methodology. When using a subset, speedups can be achieved on characterization machines and time savings can be achieved in accelerator design simulations.

We use the execution time on the E5-GTX node as the baseline machine to define the **score**. The **score** of machine A is defined as

$$Score_A = \frac{execution\ time\ on\ the\ baseline\ machine}{execution\ time\ on\ machine\ A}$$

The same methodology is applied in other workloads, and we list the 8-category 'optimal' subset for each workload in Table IV, along with the commonly used ModelNet10 subset. We define the 'optimal' subset as the one achieves the highest performance accuracy, which is defined as

$$Accuracy = 1 - \frac{|subset\ score - full\ set\ score|}{full\ set\ score}$$

B. Subset Validation

We observe that the 'optimal' subset for each workload is not the same. In real cases, it is difficult to alter the optimal

TABLE IV: 'Optimal' subsets of the four workloads, along with ModelNet10

O-CNN	Pointnet	PCL-Search	PCL-Segmentation	ModelNet10
lamp	bottle	tv_stand	sofa	bathtub
bottle	sink	desk	table	bed
piano	piano	night_stand	dresser	chair
flower_pot	radio	keyboard	sink	desk
night_stand	bowl	bowl	person	dresser
bowl	wardrobe	guitar	guitar	monitor
tent	car	toilet	night_stand	night_stand
door	chair	lamp	lamp	sofa
				table
				toilet

subset based on different workloads. So, we pick a 7-category subset *['lamp', 'bottle', 'piano', 'night_stand', 'bowl', 'sink', 'guitar']* as our global representative subset with each one appearing at least twice on the optimal subsets for the four workloads. We name the 7-category subset **ModelNet7H**.

In this part, we verify the performance accuracy of the 7-category subset ModelNet7H. At first, we will compare ModelNet7H with ModelNet10 to see if ModelNet7H is more representative from a hardware perspective. In addition, we will cross-validate ModelNet7H on the Platinum-V100 system to prove the effectiveness of our methodology in identifying subsets.

The upper part of Figure 2 lists the performance accuracy results of the four workloads on two platforms. On the two non-ML workloads, both ModelNet7H and ModelNet10 achieve more than 99% performance accuracy. For the two ML workloads, ModelNet10 achieves a 93.7% and 99.4% performance accuracy on the Platinum-P100 system. ModelNet7H achieves a 95.9% and 99.7% performance accuracy on the Platinum-P100 system, which is better than ModelNet10.

On the Platinum-V100 system, ModelNet10 only achieves a 89.8% and 83.8% performance accuracy. The number for ModelNet7H is 93.3% and 91.4% respectively, which is much better than ModelNet10. These results confirm the effectiveness of our methodology for selecting subsets for the next generation hardware platforms.

We also want to clarify that the representative subset ModelNet7H is hardware platform and workload dependent, which means it might change when using other hardware platforms, e.g, FPGAs, or other workloads (other graphics/ML applications using ModelNet40). However, the methodology remains the same. Representative subsets can be generated with the profiling results on current existing chips. When predicting the performance of the next generation chip, the simulation time can be saved by using the representative subset instead of the entire ModelNet40 dataset, with an acceptable performance accuracy loss.

C. Runtime Savings Using Subsets

The motivation for a representative subset is to save the execution/simulation time as well as maintaining the performance accuracy. In this part, we will discuss the execution time savings when using ModelNet7H and ModelNet10, with

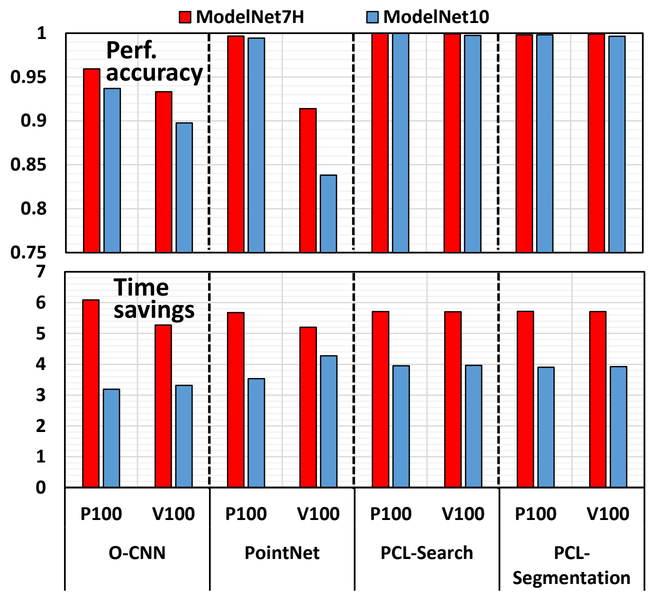


Fig. 2: Perf. accuracy (closer to 1 is better) and execution time savings (larger is better) with respect to ModelNet40

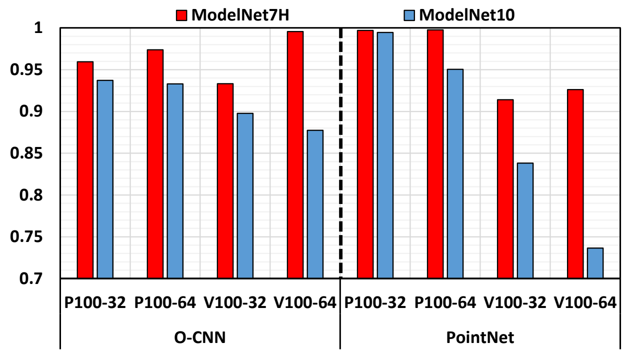
respect to ModelNet40. Since the number of models in different categories is not the same, we still need to verify that ModelNet7H achieves more time savings than ModelNet10 to prove the efficacy of our methodology.

The lower part of Figure 2 shows the execution time saving comparisons between ModelNet7H and ModelNet10 with respect to ModelNet40. When O-CNN is deployed on the Platinum-P100 system, ModelNet7H achieves the best time saving (1.91×) compared with ModelNet10. The worst case is deploying PointNet on the Platinum-V100 system, while ModelNet7H still achieves 1.22× time savings. Considering the average results of the 8 cases, ModelNet7H achieves 1.52× time savings compared with ModelNet10. As a result, our selected 7-category subset, ModelNet7H, achieves better performance accuracy as well as better time savings than ModelNet10.

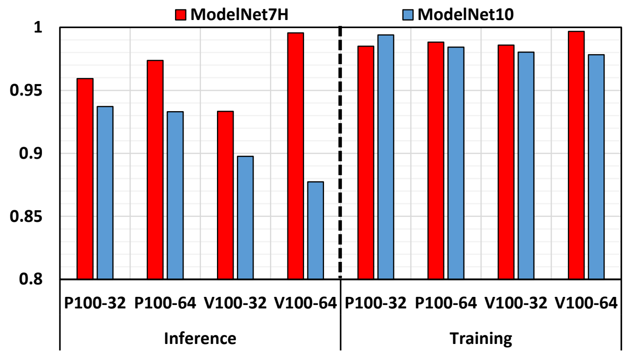
D. Subsets with the Finer Granularity

In the two non-ML workloads, a finer granularity is also supported since the input of two algorithms in PCL is one single 3D model, which means the batch size is one. We apply the same methodology which we use in identifying the subset in a coarser granularity (Section VI-A). For example, we picked a 7/8-category subset out of the 40 categories. Inside each category, e.g., bowl, we can also pick a 8-model (*['bowl_0048', 'bowl_0021', 'bowl_0045', 'bowl_0050', 'bowl_0069', 'bowl_0037', 'bowl_0022', 'bowl_0031']*) subset out of the 84 models, which achieves a 99.8% performance accuracy on Platinum-P100 system and 99.9% on Platinum-V100 system. The time savings on Platinum-P100 system and Platinum-V100 system are 10.37×, and 10.33×, respectively.

Tiny Subset To achieve a even better time savings, a tiny subset can be created by picking 8 objects from each category



(a) Inference of O-CNN and PointNet (batch size 32 and 64)



(b) O-CNN inference and training (batch size 32 and 64)

Fig. 3: Subset performance accuracy on ML workloads.

of ModelNet7H, which is a combination of coarser and finer granularity. The tiny subset achieves 219.27×, 220.09× time savings compared with ModelNet40, and 88.03×, 88.98× time savings compared to ModelNet10 on the two non-ML workloads. This tiny subset can be used in cycle-accurate micro-architecture behavioral performance simulations or even circuit-level simulation during chip design.

VII. Case studies on the two ML Workloads

As the results in Figure 2 suggest, the subset performance accuracy for the two non-ML workloads is better than the two ML workloads. In addition, there is less variance between different hardware platforms in the two non-ML workloads. So, we undertake a case study on the two ML workloads to present more insights on the characterization results.

A. Batch Size

We use the batch size 32 as the default setting for both O-CNN and PointNet. When using larger batch sizes, a higher degree of parallelism can be achieved with the cost of a higher CPU and GPU utilization. The increment of utilization rate will also make the values of performance metrics change accordingly. In this part, we discuss how batch size affects the subset performance accuracy, since the subset generated in batch size 32 may not be a representative one when the batch size changes to 64.

Figure 3a lists the performance accuracy of our selected 7-category subset, ModelNet7H, and ModelNet10 on both Platinum-P100 and Platinum-V100 systems. When the batch size changes from 32 to 64, ModelNet7H is more robust. The average performance accuracy of ModelNet7H increases from 95.09% to 97.33%, while the average performance accuracy of ModelNet10 decreases from 91.69% to 87.43%.

Takeaway: Our proposed subset is more representative than ModelNet10 in both batch size 32 and 64, since our subset is selected from a hardware perspective.

TABLE V: PC value difference between batch size 32/64

Workload	Euclidean Distance	Manhattan Distance
Inference-P100 (fig. 4a)	0.692	0.801
Training-P100 (fig. 4b)	3.606	4.084
Training-V100 (fig. 4c)	1.771	2.080

B. Training & Inference

Besides the batch size in the two ML workloads, we also make comparisons between network training and inference. We investigate the training and inference procedures of O-CNN on both batch size 32 and 64 in this section. The similar subset performance accuracy verification is also discussed in the training and inference comparisons.

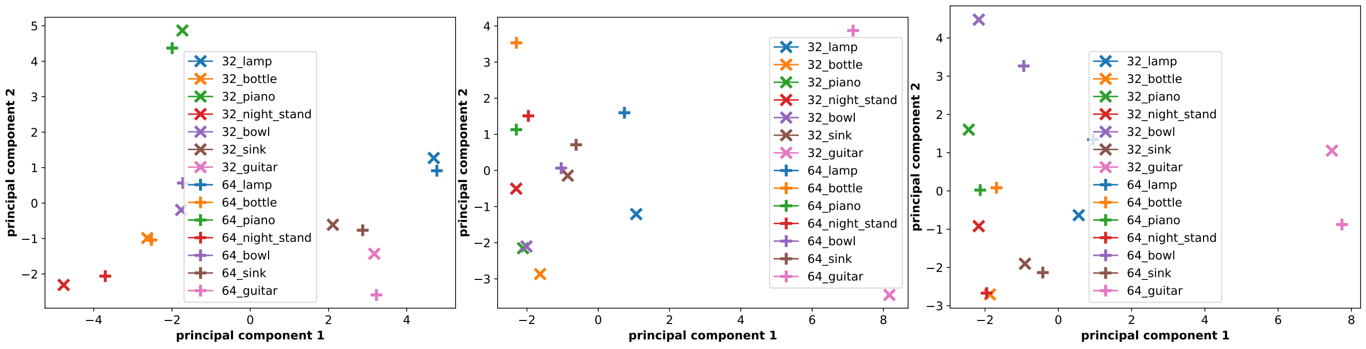
Figure 3b shows the performance accuracy of ModelNet7H and ModelNet10 on both Platinum-P100 and Platinum-V100 systems. In the inference procedure, ModelNet7H achieves an average performance accuracy of 94.6% when the batch size is 32, and the performance accuracy is 98.5% when the batch size increases to 64. On ModelNet10, the two values are 91.7% and 90.5%, respectively. In the training procedure, both ModelNet7H and ModelNet10 achieves a higher performance accuracy (all cases achieve a performance accuracy higher than 98%). As a result, the training procedure is less sensitive to the batch size change than the inference procedure.

Takeaway: Our proposed subset is more representative in training than inference for both batch size 32 and 64. Considering the long execution time of ML training, ML training chip designer could also benefit from our proposed subset.

C. Combined Effect

In this part, we discuss the combined effect of batch size and hardware platforms on the performance of ML workloads. Figure 4 shows the scatter-plot based on the first two PCs of our defined metrics in Table I. Instead of the 40-category entire dataset, ModelNet7H is used.

We investigate how PC values change if the batch size increases from 32 to 64. The average difference of each one in ModelNet7H is listed in Table V, considering both Euclidean distance and Manhattan distance. The results show that in the training procedure, the performance of each category changes more than that in the inference procedure. In addition, the Platinum-P100 changes more than the Platinum-V100 system. The reason is that training consumes more hardware resources than inference. The hardware resources are under utilized for both batch size 32 and 64 in the inference procedure. Since the



(a) Platinum-P100 O-CNN Inference (b) Platinum-P100 O-CNN Training (c) Platinum-V100 O-CNN Training

Fig. 4: Comparisons between different batch sizes on O-CNN. PC1/PC2 is dominated by CPU/GPU metrics

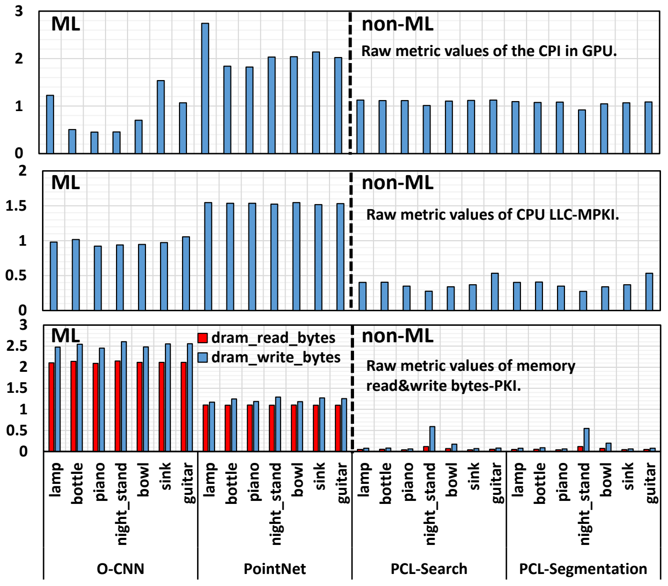


Fig. 5: Raw metric values

Platinum-P100 is a less powerful system than Platinum-V100, the performance wall is hit with a smaller batch size than Platinum-V100 in the training procedure. That explains the larger difference in the Platinum-P100 system than Platinum-V100.

Takeaway: When using our methodology, the difference in compute capability between the characterization machine and next generation platform being designed should also be considered (usually the next generation machine is more powerful). The closer the resource utilization ratio, the higher the performance prediction accuracy.

VIII. Case studies using raw metrics numbers

In this part, we study several raw metric values and show the insights we observed from these data. We show the CPI (cycle per instruction) in the GPU, CPU LLC-MPKI (last level cache miss per 1k instruction), and memory read&write bytes in the GPU of the four workloads in Figure 5. We choose the three metrics since CPI indicates the GPU performance directly, and LLC-MPKI, memory read&write bytes indicates the memory behaviors on CPUs and GPUs.

Overall Analysis: The average CPI in the GPU for the four workloads is 0.976, 1.534, 0.383, and 0.383, respectively. The CPI of the two non ML workloads is lower than that of the two ML workloads. The reason is that the two non-ML workloads are less computing intensive. For the GPU *Inst Mix*, there is no floating point instructions on the two non-ML workloads, while the number for the two ML workloads are 20.926 and 13.220 (PKI). In addition, the PointNet workload contains more integer instructions than the other 3 workloads (the number is 4.048, 13.672, 7.972, and 7.972 for the four workloads, respectively), which is the reason for the highest CPI.

The two ML workloads also show higher memory intensity than the two non-ML workloads. The higher memory intensity is also a factor leading to a higher CPI in the GPU. In addition, one interesting insight on the two ML workloads is that PointNet has a smaller number of memory read&write bytes PKI (2.328) than O-CNN (4.642). However, the CPU LLC-MPKI of PointNet (2.092) is larger than O-CNN (0.850). The reason is that the octree data structure is used in O-CNN for partitioning the model into small pieces and processing the model piece by piece, which is more cache-friendly and results in a lower LLC-MPKI in O-CNN. The cost of the octree data structure is more memory transactions between on-chip and off-chip memory [11] for data synchronization, which explains the larger number of memory read&write bytes in O-CNN.

Unique data points and Takeaway: We observe that the CPI behavior of *lamp* in ML, and memory behavior of *night_stand* in non-ML workloads is different from other workloads. Unique data points always exist in workload characterization results, especially when using a representative subset (datasets with similar behaviors are grouped together and only one is selected and added to the representative subset).

The *night_stand* is in ModelNet10 while *lamp* is not. If ModelNet10 is used to drive accelerator design, some unique data points, which are important for domain specific ASIC accelerators, may be ignored. This is another reason for using a hardware-aware subset.

IX. Conclusion

In this paper, we have demonstrated a methodology for selecting hardware-aware subsets of large 3D model datasets. We used ModelNet40 to illustrate the workflow, and a 7-category dataset ModelNet7H is generated. ModelNet7H is $1.52\times$ faster than the existing ModelNet10 subset and achieves a performance accuracy of 97.5% with respect to ModelNet40 (compared with 95.7% achieved by ModelNet10). We characterize both ML and non-ML workloads on three hardware platforms to cross-validate our result. Case studies are undertaken to illustrate the efficacy of our methodology. Subsets thus generated can be used during trade-off evaluations for designing future accelerators.

X. Acknowledgement

This research was supported in part by NSF grant numbers 1763848. We thank anonymous reviewers for the detailed comments on the paper. Authors would also like to acknowledge computational resources from Texas Advanced Computing Center (TACC). Any opinions, findings, conclusions or recommendations are those of the authors and not of the National Science Foundation or other sponsors. We thank Anthony Do for the proof reading of the paper.

References

[1] "Linux perf tool," https://perf.wiki.kernel.org/index.php/Main_Page.

[2] J. Behley, V. Steinhage, and A. B. Cremers, "Efficient radius neighbor search in three-dimensional point clouds," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 3625–3630.

[3] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti *et al.*, "The gem5 simulator," *ACM SIGARCH computer architecture news*, vol. 39, no. 2, pp. 1–7, 2011.

[4] M. Botsch, L. Kobbelt, M. Pauly, P. Alliez, and B. Lévy, *Polygon mesh processing*. CRC press, 2010.

[5] C. Burstedde, L. C. Wilcox, and O. Ghattas, "p4est: Scalable algorithms for parallel adaptive mesh refinement on forests of octrees," *SIAM Journal on Scientific Computing*, vol. 33, no. 3, pp. 1103–1133, 2011.

[6] N. Corporation, "Nvidia cuda toolkit documentation," https://docs.nvidia.com/cuda/profiler-users-guide/index.html/, 2022.

[7] A. Deshmukh, R. Li, R. Sen, R. R. Henry, M. Beckwith, and G. Gupta, "Performance characterization of .net benchmarks," in *2021 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2021, pp. 107–117.

[8] T. M. Fruchterman and E. M. Reingold, "Graph drawing by force-directed placement," *Software: Practice and experience*, vol. 21, no. 11, pp. 1129–1164, 1991.

[9] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[10] W. Jiang, E. H.-M. Sha, Q. Zhuge, L. Yang, X. Chen, and J. Hu, "Heterogeneous fpga-based cost-optimal design for timing-constrained cnns," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2542–2554, 2018.

[11] R. Li, S. Song, Q. Wu, and L. K. John, "Accelerating force-directed graph layout with processing-in-memory architecture," in *2020 IEEE 27th International Conference on High Performance Computing, Data, and Analytics (HiPC)*. IEEE, 2020, pp. 271–282.

[12] Y. Li, R. Bu, M. Sun, W. Wu, X. Di, and B. Chen, "Pointcnn: Convolution on χ-transformed points," in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, 2018, pp. 828–838.

[13] D. Maturana and S. Scherer, "Voxnet: A 3d convolutional neural network for real-time object recognition," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2015, pp. 922–928.

[14] K. Mo, S. Zhu, A. X. Chang, L. Yi, S. Tripathi, L. J. Guibas, and H. Su, "Partnet: A large-scale benchmark for fine-grained and hierarchical part-level 3d object understanding," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 909–918.

[15] A. Nguyen and B. Le, "3d point cloud segmentation: A survey," in *2013 6th IEEE conference on robotics, automation and mechatronics (RAM)*. IEEE, 2013, pp. 225–230.

[16] R. Panda, S. Song, J. Dean, and L. K. John, "Wait of a decade: Did spec cpu 2017 broaden the performance horizon?" in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2018, pp. 271–282.

[17] A. Phansalkar, A. Joshi, and L. K. John, "Analysis of redundancy and application balance in the spec cpu2006 benchmark suite," in *Proceedings of the 34th annual International Symposium on Computer architecture*, 2007, pp. 412–423.

[18] A. Prokopec, A. Rosà, D. Leopoldseder, G. Duboscq, P. Tma, M. Studener, L. Bulej, Y. Zheng, A. Villazón, D. Simon *et al.*, "Renaissance: Benchmarking suite for parallel applications on the jvm," in *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2019, pp. 31–47.

[19] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 652–660.

[20] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," *arXiv preprint arXiv:1706.02413*, 2017.

[21] V. J. Reddi, C. Cheng, D. Kanter, P. Mattson, G. Schmuelling, C.-J. Wu, B. Anderson, M. Breughe, M. Charlebois, W. Chou *et al.*, "Mlperf inference benchmark," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020, pp. 446–459.

[22] R. B. Rusu and S. Cousins, "3d is here: Point cloud library (pcl)," in *2011 IEEE International Conference on Robotics and Automation*, 2011, pp. 1–4.

[23] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder, "Automatically characterizing large scale program behavior," *ACM SIGPLAN Notices*, vol. 37, no. 10, pp. 45–57, 2002.

[24] SPEC org, "What is a "reference machine"? why use one?" https://www.spec.org/cpu2017/Docs/overview.html#Q18, 2021.

[25] P.-S. Wang, Y. Liu, Y.-X. Guo, C.-Y. Sun, and X. Tong, "O-cnn: Octree-based convolutional neural networks for 3d shape analysis," *ACM Transactions on Graphics (TOG)*, vol. 36, no. 4, pp. 1–11, 2017.

[26] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao, "3d shapenets: A deep representation for volumetric shapes," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1912–1920.

[27] C. Xie, S. L. Song, J. Wang, W. Zhang, and X. Fu, "Processing-in-memory enabled graphics processors for 3d rendering," in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2017, pp. 637–648.

[28] C. Xie, X. Zhang, A. Li, X. Fu, and S. Song, "Pim-vr: Erasing motion anomalies in highly-interactive virtual reality world with customized memory cube," in *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2019, pp. 609–622.

[29] T. Xu, B. Tian, and Y. Zhu, "Tigris: Architecture and algorithms for 3d perception in point clouds," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 629–642.

[30] X. Zheng, L. K. John, and A. Gerstlauer, "Accurate phase-level cross-platform power and performance estimation," in *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 2016, pp. 1–6.