MEMORY-FRIENDLY SEGMENTATION REFINEMENT FOR VIDEO-BASED POINT CLOUD COMPRESSION

Ismael Seidel[⋆] *Davi R. Freitas*[†] *Camilo Dorea*[†] *Diogo C. Garcia*[†]
Renan U. B. Ferreira[†] *Rogério Higa*[⋆] *Ricardo L. de Queiroz*[†] *Vanessa Testoni*[⋆]

[⋆]Samsung R&D Institute (SRBR) – Campinas, SP, Brazil
[†]University of Brasilia (UnB) – Brasilia, DF, Brazil
{i.seidel, r.higa, vanessa.t}@samsung.br, rabbouni.davi@image.unb.br,
{camilodorea, diogogarcia, renan}@unb.br, queiroz@ieee.org

ABSTRACT

Recently finalized, the MPEG Video-based Point Cloud Compression (V-PCC) standard leverages existing video codecs to compress point clouds. This approach relies on existing video coding hardware accelerators to enable its fast adoption. However, the processing steps to project 3D point clouds into 2D frames still have a considerable complexity. Thus, we propose two modifications to TMC2, V-PCC's test model, considering the memory access pattern: one reducing unnecessary memory allocation and the other increasing data locality through a set of pre-processing steps. Our approach achieved up to 46.31% encoding self-time reduction without changing the resulting bitstream. This paper also provides an analysis of our implementation that may help future V-PCC codecs achieve real-time encoding.

Index Terms— Point Cloud Compression, V-PCC, 3DG, Standard, Complexity reduction, Segmentation Refinement.

1. INTRODUCTION

Point Clouds (PCs) enable us to represent virtual worlds as we get closer to mainstream immersive applications. Given the enormous amount of data involved in representing PCs, a group from Moving Picture Experts Group (MPEG) known as 3 Dimensional Graphics Team (3DG) has been developing standards for efficient PCs representation [1]. 3DG defined three PC categories [2] in its Call for Proposals (CfP) for Point Cloud Compression (PCC) [3]: category 1 includes static objects and scenes; category 2 includes dynamic objects; and category 3 includes dynamic acquisition, such as the ones captured from Lidar devices [4].

The Test Model for Category 2 (TMC2) software leverages existing video coding technologies, such as the High Efficiency Video Coding (HEVC) [5]. For such a characteristic, the standard for compression of category 2 PCs was named Video-based Point Cloud Compression (V-PCC) [4]. V-PCC works by projecting the PCs 3D information into 2D video frames of three types: geometry, texture, and occupancy maps. Although video codec agnostic, TMC2 adopts the HEVC Test Model (HM) [6] as the default codec. The known huge complexity of HM [7] contributes directly to the complexity of TMC2 [8]. Nonetheless, the steps apart from the video codec must be tailored for speed, as V-PCC may take advantage of existing embedded video codecs, i.e., real-time hardware accelerators.

In this aspect, given the large number of points in a PC, one cannot disregard the memory wall problem [9]–[11], i.e., the difference between processor and memory access speeds imposes penalties to programs that disregard such a gap. Thus, this work's main contribution is the proposal and assessment of a memory-friendly version of the refine segmentation method used in TMC2, which, before our proposal, was one of the most time-consuming TMC2 methods. Our contribution may help drive further improvements toward real-time encoding of PCs using V-PCC by avoiding memory bottlenecks.

The remainder of this paper is organized as follows. Section 2 introduces the refine segmentation method along with a summary of MPEG contributions targeting its complexity reduction. Section 3 presents our proposed modifications in the refine segmentation step to speed up the encoder without coding efficiency losses. Section 4 shows the adopted speed-up test method, while the obtained results are discussed in Section 5. Finally, Section 6 brings our conclusions.

2. SEGMENTATION REFINEMENT

V-PCC needs to project the PCs 3D information into 2D video frames to leverage existing video technologies. The 2D frames are composed of patches projected from the PC into planes surrounding it. Optimal patch segmentation is an NP-hard problem, performed in TMC2 using a heuristic approach [1], illustrated in Figure 1.

Initially, a normal is estimated for every point. The initial segmentation uses the computed normals to associate each

PART OF RESULTS PRESENTED IN THIS WORK WAS OBTAINED THROUGH THE "POINT CLOUDS COMPRESSION" PROJECT, FUNDED BY SAMSUNG ELETRÔNICA DA AMAZÔNIA LTDA., UNDER THE BRAZILIAN INFORMATICS LAW 8.248/91.

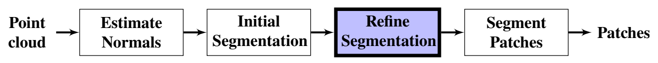

Fig. 1. Point cloud to patches flow in TMC2. Adapted from [12].

point to an oriented plane. However, such a step has limited clusterization capability. Therefore, a refinement step (highlighted in Figure 1) is performed to improve this segmentation, considering the points' nearest neighbors along with their estimated normals in a combined score. After the refinement, points associated with the same planes are grouped through a connected components algorithm [13], forming the patches.

The refine segmentation step seems to be one of the most time-consuming ones of TMC2 from its inception as `refineSegmentation` method. Given its complexity, `refineSegmentation` was addressed by a few complexity reduction contributions. Samsung proposed an alternative implementation that is a bit-exact match with the anchor implementation [14]. Such a proposal brought the total encoding time down by about 60%.

However, the `refineSegmentation` method still had a high complexity. Thus, Samsung proposed a grid-based version, called `refineSegmentationGridBased` [15], [16]. The key idea is to group neighboring points into a grid, reducing the total number of score computations.

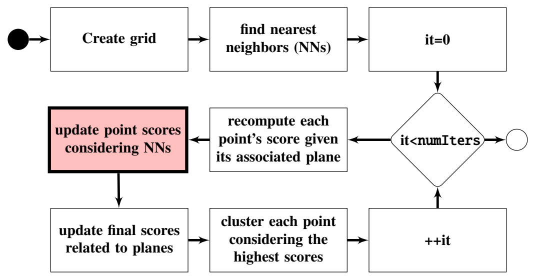
Create grid
find nearest neighbors (NNs)
it=0
update point scores considering NNs
recompute each point's score given its associated plane
it<numIters
update final scores related to planes
cluster each point considering the highest scores
++it

Fig. 2. `refineSegmentationGridBased` simplified flow chart. Adapted from [12].

A hash map is adopted to represent the grid, facilitating the search for points in this grid. Four parameters control the `refineSegmentationGridBased` method [15]:

1. `numIters`: number of refinement iterations (default=10).
2. `voxDim`: voxel (grid) dimension (default=4).
3. `nnSearchRadius`: nearest-neighbor searching radius of kd tree (default=192).
4. `maxNNCount`: maximum number of nearest-neighbors for each point (default=1024).

Further complexity analysis of TMC2 showed that, despite being faster than the original refine segmentation method, the grid-based version continued to dominate time complexity [17], ranging from 77% to 91% of the computing cycles. A tweak in that method's implementation, avoiding redundant hash map accesses, was able to bring TMC2 self-time down by a further 41% [18]. Such a tweak was integrated into TMC2 v9 [19].

2.1. Complexity assessment

To assess `refineSegmentationGridBased` complexity in the current TMC2 encoder (v11), we compared its execution time (measured with `std::chronos`) with the total TMC2 self-time. This experiment was executed on AMD Ryzen 9 3900X @ 4GHz processor, with TMC2 v11 compiled with GCC v9.3.0 in release mode. Also, we configured CMake scripts to use HM as an external software instead of linking its libraries in a stand-alone TMC2 binary. Such a configuration allows for TMC2 to report HM time as child process time. Thus, the self-time does not include HM's execution, which is more realistic considering the future adoption of embedded codecs.

We encoded 32 frames of Loot, RedAndBlack, Soldier, and Longdress PCs [20], with one and six threads, random access configuration (C2RA), and rate point R5 [21]. Figure 3 shows the obtained results.

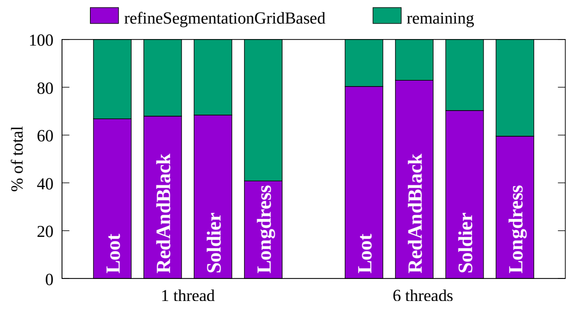

Fig. 3. `refineSegmentationGridBased` time shares in TMC2 v11 with one and six threads.

Even considering the single-thread executions, the share of grid-based segmentation refinement is over 50% for three out of the four tested PCs. The lower share for Longdress PC is due to the method's configuration in that case. All tested PCs use the default configuration for `nnSearchRadius` and `maxNNCount`. However, Loot, RedAndBlack, and Soldier use `voxDim`=2 (smaller than the default), and Longdress uses a larger number of iterations (50). A larger voxel (`voxDim`) means fewer score computations due to a smaller number of grid positions. Therefore, despite executing more iterations in the case of Longdress, the smaller number of voxels results in the smaller time-share for refine segmentation method in such a PC observed in Figure 3.

We used `perf` tools to help identify the reasons for the still high complexity of the refine segmentation method. Such tools can record stack frames by sampling, which can be further used to generate the so-called "flame graphs" [22]. For such a case, TMC2 was compiled in debug mode. Thus, the obtained time-shares may be slightly different than in release mode.

A large time-share within refine segmentation step is due to the `computeAdjacencyInfoInRadius` method. In such a method, about 2/3 of the time is spent by the nanoflann library [23]. The remaining 1/3 is due to the results vector's resizing. A similar share is due to the access of elements from the hash map, i.e., the grid. Despite the search for elements in a hash map having O(1) expected time complexity [24], the access time starts to increase when such a map is too large. This happens because memory access times may impose huge penalties when the memory hierarchy (cache) cannot help due to poor data locality. Moreover, the hash map access operator's time-share is much larger when TMC2 is compiled in release mode, as it became evident after obtaining our results.

3. MODIFICATIONS IN TMC2

To tackle the observed sources of complexity, we propose two modifications in TMC2. The first, minor one, deals with unnecessary memory allocations. The second one adapts the grid-based refine segmentation method to increase locality.

3.1. Memory allocation

`PCCKdTree::searchRadius` needs to copy the results from the nanoflann library [23]. However, the resulting vector is always resized to the maximum allowed number of results `num_results`, even when the number of results from nanoflann (let us call it `ret_size`) is smaller than the maximum. Therefore, we propose to use minimum(`num_results`, `ret_size`) to resize the resulting vector to the optimal size. This small modification keeps functionality while dramatically reduces the total amount of allocated bytes.

3.2. Cache-friendly segmentation refinement

As pointed out in Section 2.1, the access to the grid represented as a hash map (`std::unordered_map`) is responsible for a large share of `refineSegmentationGridBased` complexity. The loop depicted in Figure 2 contains the bulk of those accesses that concentrate mostly on the highlighted step.

We propose to add a set of pre-processing steps before such a loop to increase the data locality in memory intensive parts of the code, aiming to reduce execution time. The key that allows our proposal is that while each point's orientation may change, their positions in the grid remain constant during execution. The pre-processing consists mainly of:

1. Using two hash maps instead of one, reducing the size of a single instance of the hash map (increasing locality):
   (a) One to map grid center to point indices;
   (b) Another to map the grid's center to the vector of scores;
2. Pre-computing the weight for each grid voxel, considering the number of adjacent points and a pre-defined lambda (as used in the original method [12]);
3. Sorting the adjacent points to reduce the address distance between consecutive accesses;
4. Zipping the contents from the two hash maps using a vector of pairs to enable the scores' sequential update.

4. EVALUATION METHOD

We implemented the proposed modifications in TMC2 v11 for obtained the results presented in this paper and as a contribution to MPEG committee [25]. Our contribution has been accepted and integrated into TMC2 v12 [26] and thus, for more details on our proposal, one can check out the complete code. The compilation process is the same presented in Section 2.1, considering debug and release mode. We compiled TMC2 in debug mode and used Valgrind's xtree-memory tool to verify the memory allocation modification impact. On the other hand, to evaluate speed-up, we encoded 32 frames of Loot, RedAndBlack, Soldier, and Longdress PCs using C2AI (all intra) and rate point R5 [21] in TMC2 release mode. We compared the MD5 checksums of the encoded PCs reported by both original and modified TMC2 to ensure they match. Equal MD5 checksums guarantee that our proposal does not change coding efficiency. As our implementation aims to leverage data locality, we experimented on three platforms with different topologies:

- AMD Ryzen™ 9 3900X@4GHz, with 64MB, 6MB, and 786KB total cache sizes for level 3, level 2, and level 1, respectively;
- Intel© Core™ i7 8565U@3.1GHz with 8MB, 1MB, and 256KB total cache sizes;
- Intel© Core™ i5 6200U@3.1GHz with 3MB, 512KB, and 128KB total cache sizes.

The first one is a desktop processor, whereas the other two are mobile ones. Despite the different total amount of cache in each processor, there are some similarities in their topology, shown in Figure 4. Level 1 has equal capacity on all platforms. The AMD processor has twice the level 2 capacity than the other two, which are equal among them. All processors have unique level 3 sizes shared by groups of 2, 3, or 4 cores.

We obtained the execution time of each run from TMC2 reports and computed self-time reduction (%) as:

$$\text{self-time reduction} = \left(1 - \frac{\text{modified self-time}}{\text{original self-time}}\right) \times 100\% \quad (1)$$

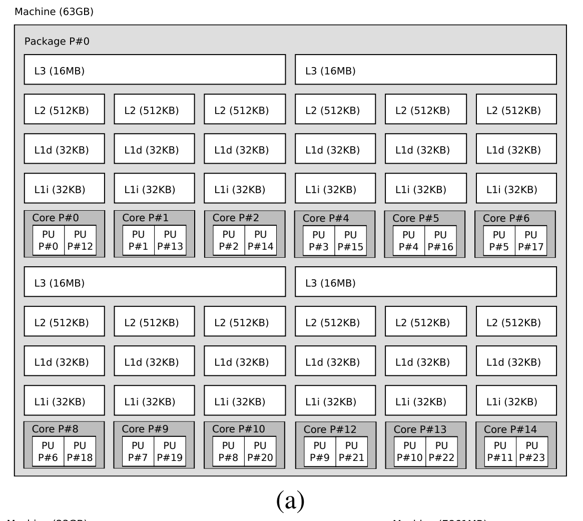
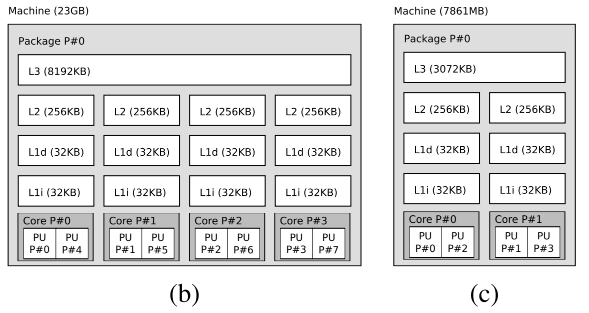
Fig. 4. Topology of (a) AMD Ryzen™ 9 3900X, (b) Intel© Core™ i7 8565U, and (c) Intel© Core™ i5 6200U, as reporded by `lstopo` tool from `hwloc` [27].

5. RESULTS

The main benefit of the memory allocation modification is a much smaller number of total bytes allocated during the execution of TMC2. Before the modification, a total of 3.19 TiB of memory was allocated. After the modification, only 248.42 GiB was still being allocated (reduction of 92.39%). Notice that these values are not peak memory, but total bytes allocated throughout the execution.

Figure 5 shows the obtained speed-up results, considering both modifications. The memory allocation modification played only a small role in these results, being a fraction of the cache-related modification benefits. Considering the PCs, Longdress has smaller benefits. This is expected because the total share of refine segmentation method was already small for this PC compared to the other ones (Figure 3). The speed-up results considering the remaining PCs are similar among them.

Also as expected, the smaller benefits are observed for the processor with the larger caches. Moreover, level 2 caches seems to have played a major role since the difference between Ryzen and i7 is quite large (there is a difference starting in L2 cache size) and the difference from i7 to i5 is smaller (there is a difference only in L3 cache size).

The reason for such considerable gains in coding speed is

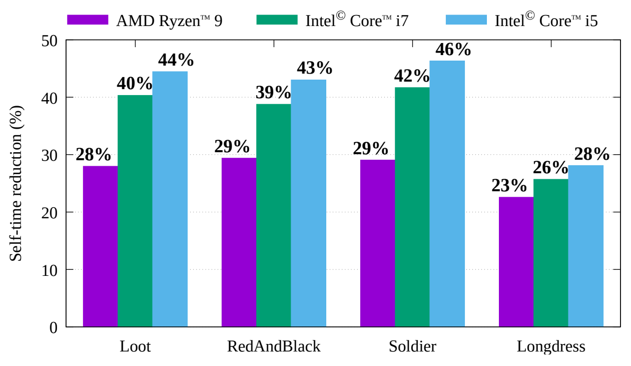

Fig. 5. Self-time reduction results.

that our implementation successfully ensured better cache locality, thus reducing miss ratio. The better locality is achieved by shortening the distances between consecutive memory accesses. The distribution of memory distances between consecutive assesses during the most memory intensive part of the first iteration from refine segmentation method is shown in Figure 6. Thus, it is clear that after our modification, there is a larger probability that the next required data is already at the nearer cache level.

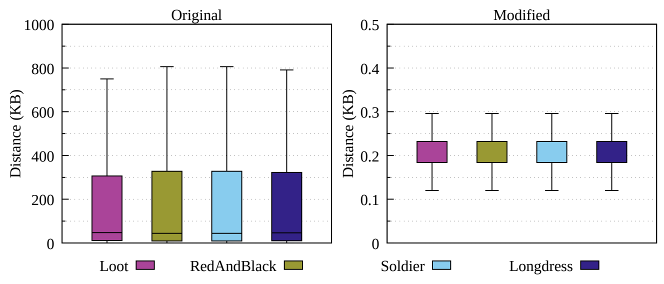

Fig. 6. Distance of consecutive memory accesses before and after our modification. There is a huge difference in the y-axis scale.

6. CONCLUSION

This paper presented and evaluated a new memory-friendly version of the grid-based refine segmentation method for V-PCC. Our proposal speeds up TMC2 while keeping its coding efficiency intact, i.e., the encoded PCs using our modified implementation are bit-exact matches compared to the anchor ones. The proposed implementation has still room for improvement but already showed promising speed-up results, with up to 46.31% encoding self-time reduction (34% on average). Such a speed-up depends on the platform (cache size) and the parameters of the `refineSegmentationGridBased` method. Moreover, we brought up for discussion the well known, but sometimes forgotten, memory wall problem. This paper's lesson is that avoiding the memory bottlenecks is of the utmost importance to achieve fast and even real-time software encoding of category 2 PCs.

References

[1] S. Schwarz *et al.*, "Emerging MPEG standards for point cloud compression," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 1, pp. 133–148, 2019.

[2] 3DG, "ISO/IEC JTC 1/SC29/WG11 N16716: Draft test conditions and complementary test material," Geneva, CH, Jan. 2017.

[3] ——, "ISO/IEC JTC 1/SC29/WG11 N16732: Call for proposals for point cloud compression," Geneva, CH, Jan. 2017.

[4] E. S. Jang *et al.*, "Video-based Point-Cloud-Compression standard in MPEG: From evidence collection to committee draft [standards in a nutshell]," *IEEE Signal Process. Mag.*, vol. 36, no. 3, pp. 118–123, 2019.

[5] ISO Central Secretary, "Information technology – High efficiency coding and media delivery in heterogeneous environments – Part 2: High efficiency video coding," en, International Organization for Standardization, Geneva, CH, Standard ISO/IEC 23008-2, 2013.

[6] K. Suehring and K. Sharman, *HM HEVC reference software v16.20*, `https://vcgit.hhi.fraunhofer.de/jct-vc/HM`, 2019.

[7] G. Correa, P. Assuncao, L. Agostini, and L. A. da Silva Cruz, "Performance and computational complexity assessment of high-efficiency video encoders," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 12, pp. 1899–1909, 2012.

[8] M. Gonçalves *et al.*, "Encoding efficiency and computational cost assessment of state-of-the-art point cloud codecs," in *2019 IEEE International Conference on Image Processing (ICIP)*, 2019, pp. 3726–3730.

[9] W. A. Wulf and S. A. McKee, "Hitting the memory wall: Implications of the obvious," *SIGARCH Comput. Archit. News*, vol. 23, no. 1, pp. 20–24, Mar. 1995.

[10] N. R. Mahapatra and B. Venkatrao, "The processor-memory bottleneck: Problems and solutions," vol. 5, no. 3es, 2–es, Apr. 1999.

[11] D. Efnusheva, A. Cholakoska, and A. Tentov, "A survey of different approaches for overcoming the processor - memory bottleneck," *Computer Science and Information Technology*, vol. 9, pp. 151–163, 2017.

[12] 3DG, "ISO/IEC JTC 1/SC29/WG11 N19526: V-PCC codec description," 2020.

[13] D. Graziosi *et al.*, "An overview of ongoing point cloud compression standardization activities: Video-based (V-PCC) and geometry-based (G-PCC)," *APSIPA Transactions on Signal and Information Processing*, vol. 9, e13, 2020.

[14] E. Faramarzi and M. Budagavi, "ISO/IEC JTC 1/SC29/WG11 M46201: On complexity reduction of TMC2 encoder," Samsung Electronics, Marrakech, MA, Jan. 2019.

[15] E. Faramarzi, M. Budagavi, and R. Joshi, "ISO/IEC JTC 1/SC29/WG11 M47600: Grid-based partitioning," Samsung Electronics, Geneva, CH, Mar. 2019.

[16] E. Faramarzi *et al.*, "ISO/IEC JTC 1/SC29/WG11 M49587: CE2.27 report on encoder's speedup," Samsung Electronics, Gothenburg, SE, Jul. 2019.

[17] H. Becerra, R. Higa, P. Garcia, and V. Testoni, "ISO/IEC JTC 1/SC29/WG11 M50659: V-PCC encoder performance analysis," Geneva, CH, Oct. 2019.

[18] R. Higa, P. Garcia, and V. Testoni, "ISO/IEC JTC 1/SC29/WG11 M52200: V-PCC encoder performance optimization and speed up," Brussels, BE, Jan. 2020.

[19] 3DG, "ISO/IEC JTC 1/SC29/WG11 N19092: V-PCC codec description," 2020.

[20] E. d'Eon, B. Harrison, T. Myers, and P. A. Chou, "ISO/IEC JTC1/SC29 Joint WG11/WG1 (MPEG/JPEG) m40059/M74006: 8i voxelized full bodies, version 2 – a voxelized point cloud dataset," 8i, Geneva, CH, Jan. 2017.

[21] 3DG, "ISO/IEC JTC 1/SC29/WG11 N19324: Common test conditions for point cloud compression," Alpbach, (online), Apr. 2020.

[22] B. Gregg, "The flame graph," *Communications of the ACM*, vol. 59, no. 6, pp. 48–57, 2016.

[23] J. L. Blanco and P. K. Rai, *Nanoflann: A C++ header-only fork of FLANN, a library for nearest neighbor (NN) with kd-trees*, `https://github.com/jlblancoc/nanoflann`, 2014.

[24] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT press, 2009.

[25] I. Seidel *et al.*, "ISO/IEC JTC 1/SC29/WG7 M55143: Cache-friendly refine segmentation for TMC2 speed-up," Samsung and Universidade de Brasilia, Online, Oct. 2020.

[26] J. Ricard, "ISO/IEC JTC 1/SC29/WG7 M56073: TMC2 software v12.1 improvements and evaluations," Online, Jan. 2021.

[27] F. Broquedis *et al.*, "hwloc: a Generic Framework for Managing Hardware Affinities in HPC Applications," in *PDP 2010 - The 18th Euromicro International Conference on Parallel, Distributed and Network-Based Computing*, IEEE, Ed., Pisa, Italy, Feb. 2010. [Online]. Available: `https://hal.inria.fr/inria-00429889`.