Slicing Acceleration Unit: Towards Overcoming the Computational Bottleneck in Next-Generation 3D Printing

ABSTRACT

3D Printing is revolutionizing the next-generation manufacturing industry. With the increased design complexity, the computation in prefabrication process is becoming the bottleneck of 3D printing. For example, a multi-scale, multi-material 3D design (e.g. a bionic bone) takes a few hours even days to complete the prefabrication computation. In this paper, we analyze the legacy prefabrication algorithms of 3D printing and identify its bottlenecks in the computation process. To this end, we present an architectural enhancement solution, namely slicing acceleration unit (SAU), to address this challenge. With customized structures and seamless integration into a pipelined CPU, SAU can significantly speed up the prefabrication process of 3D printing. We perform the evaluation with a standard suite of 3D printing benchmarks, and experimental results indicate that SAU can accelerate the slicing process up to 22x compared to the status quo.

Keywords

3D Printing, Bottleneck, Slicing Algorithm, Hardware Acceleration

1. INTRODUCTION

3D printing, also known as additive manufacturing, refers to a process which builds a three-dimensional (3D) object layer by layer from digital data [8]. As an advanced manufacturing technique, 3D printing holds the merit of affordability and customizability. It has been changing the market trends, and resulting in an efficient, responsive, robust and sustainable production paradigm in a wide range applications including aerospace, automobile, defense, biomedical, health and energy industries [10, 3, 12, 4].

There are three steps in 3D printing, as shown in Figure 1. The first step is to create a 3D design model, which is designated by 3D design tools (e.g., Solidworks) or produced by a 3D scanner. The 3D design is represented as a triangle mesh in the STereoLithography (STL) format. The second step is a computing process that transforms a 3D design file into a set of instruction codes, which a 3D printer can execute. This process for a 3D printer is similar to a compiler for a CPU or a synthesizer for an FPGA.This step is also named as $prefabrication$. The last step is the manufacturing process, in which 3D printer fabricates the object with the instruction codes. The design process is offline and done by users, and prefabrication and manufacturing process are online with computers and 3D printers.

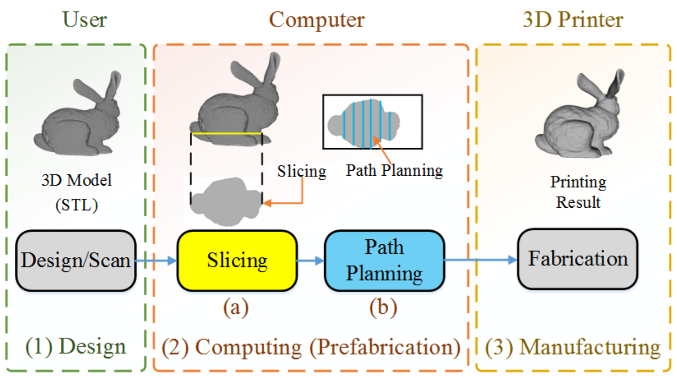

Figure 1: Three stages in 3D printing.

Recently, the prefabrication is becoming the bottleneck in the online process of 3D printing for two reasons. First, with the advance on material, process and machine development, the 3D design product is upgraded from single-scale, single-material to multi-scale, multi-material. The size and complexity of a 3D design is tremendously large. For example, a bionic product design (e.g. bones) is with the size of several Gigabytes and will take a few days in the prefabrication process. Second, the breakthrough of 3D printing technologies significantly reduces the manufacturing time. For example, the recently developed continuous 3D printing technology decreases the manufacturing time by more than two orders of magnitudes, and enables a complex 3D design production within a few minutes only [11]. Therefore, there is an urgent need to address the computational challenges in prefabrication in order to unlock design and manufacturing opportunities enabled by 3D printing and promote its wide adoption to achieve the full industrial revolution.

There are a few research works to speedup the prefabrication in the manufacturing research community. Chen *et al.* proposed a topology optimization tool to reduce the structure complexity of a 3D design [5]. Zhou *et al.* developed an efficient path planning algorithm to decrease the overhead in the prefabrication. Sabourin *et al.* [15] shorten the time cost by improving the slicing algorithm. Muller *et al.* presented a method to accelerate the slicing in the prefabrication through downsampling a design resolution [14]. However, these methods are all within the scope of software and algorithm, which either compromise the production quality or only improve the prefabrication efficiency with a limited performance benefit.

In this paper, we present a new solution to tackle the prefabrication challenges from a hardware design perspective. Without compromising the production quality, our solution is expected to increase the prefabrication speed by times or even orders of magnitude. Specifically, we first identify that the computational bottleneck in prefabrication is slicing. Then, we describe a novel slicing acceleration unit (SAU) to accelerate the slicing process. According to the indepth analysis of a state-of-the-art algorithm, the SAU design comprises a distance queue (DQ) and a serial of arithmetic unit (AU). For the sake of usability, the SAU design is tightly coupled with a general CPU architecture with customized, pipelined instructions. We evaluate our SAU with a suite of standardized 3D printing benchmark. The experimental results indicate that SAU can reach a performance improvement in prefabrication up to 22x.

The remaining of the paper is organized as follows: Section 2 describes the computational bottleneck of 3D printing is discussed. The analysis of this bottleneck and introduction of our solution are presented in Section 3. Section 4 elaborates The design and architecture of SAU. Section 5 evaluates the SAU performance and discusses the experimental results. The paper is concluded in Section 6.

2. PRELIMINARIES AND RELATED WORK

In this section, we characterize and identify the bottleneck in computing process of 3D printing. Also, we review the existing work on slicing acceleration.

2.1 Characterization of Prefabrication Process

As shown in Figure 1, prefabrication comprises two stages, i.e. *slicing* and *path planning*. *Slicing* is to convert a 3D design model to a set of 2D planar layers. It intersects all the triangles with each slice plane and connects the resulting segments into a group of properly oriented closed contours. *Path planning* transforms a 2D sliced contour into 1D paths which is to direct the printing tools to form the desired shape.

In order to explore the prefabrication time in next-generation 3D printing and identify its bottleneck, we perform the simulation under different manufacturing techniques in our early work. Specifically, the experiment is to simulate the prefabrication process of a 3D design of radial impeller, and the experimental result is shown in Figure 2. Each bar represents the total prefabrication time at each manufacturing process technique. As manufacturing technique improves from 1000um to 5um, the total prefabrication time grows from less than 1 minute to more than 4 hours, which shows prefabrication time grows significantly as manufacturing technique improves. Furthermore, slicing is always dominant in time, which accounts for more than 95% of the total time. Once the manufacturing technique is promoted to nanoscale, it would take days or weeks to finish the slicing of a 3D design. In order to improve the performance and efficiency of next-generation 3D printing, it is urgently

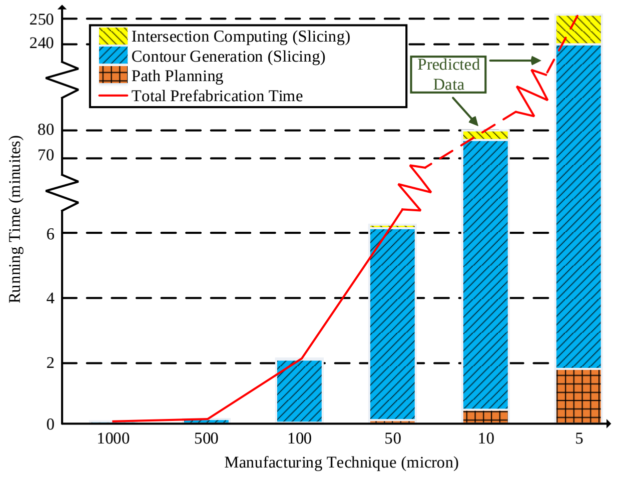

Figure 2: Tendency and time distribution of 3D printing computation.

demanded to increase the computing speed of slicing.

2.2 Related Work in Slicing Acceleration

There are several research work on improving the efficiency of slicing process for 3D printing. Kirschman *et al.* invented the first slicing algorithm [9]. Teta *et al.* improved this algorithm by sorting the triangles in STL file before slicing [16]. McMains *et al.* adopted the topology information to increase the speed of slicing [13]. Vatani *et al.* proposed the state-of-the-art slicing algorithm [17], which is currently used in manufacturing industry.

All these efforts intend to improve slicing algorithm efficiency, and there is few work to speed up the slicing process from the perspective of hardware and architecture. Customized hardware has been successfully applied to accelerate domain-specific computing [6, 18, 7]. In this work, we will explore a specific hardware acceleration for slicing in 3D printing.

3. OUR APPROACH

In this section, we introduce the state-of-the-art slicing algorithm, followed by exploring its character and the hints for hardware acceleration. Then we propose SAU, a hardware acceleration module that can promote the performance of slicing process.

3.1 State-of-the-art Slicing Algorithm

Figure 3 shows the block diagram of state-of-the-art slicing algorithm is shown in yellow boxes. The input file for slicing is a STL file, which describes the triangulated surface of 3D object by verticals of triangle and unit normal [17]. The output is a common layer interface (CLI) file [1], which stores the contour information of all layers of 3D models. CLI file will be used in a further step of prefabrication.

Slicing process contains a slicing iterator, which iterates through all layers of a 3D model. Each iteration includes two stages. The first stage is *intersection computing*, which computes the intersection segments between the 3D model and current layer, then stores these segments in the intersection queue in processing order. The second stage is *traversal contour connection*. It selects the first segment

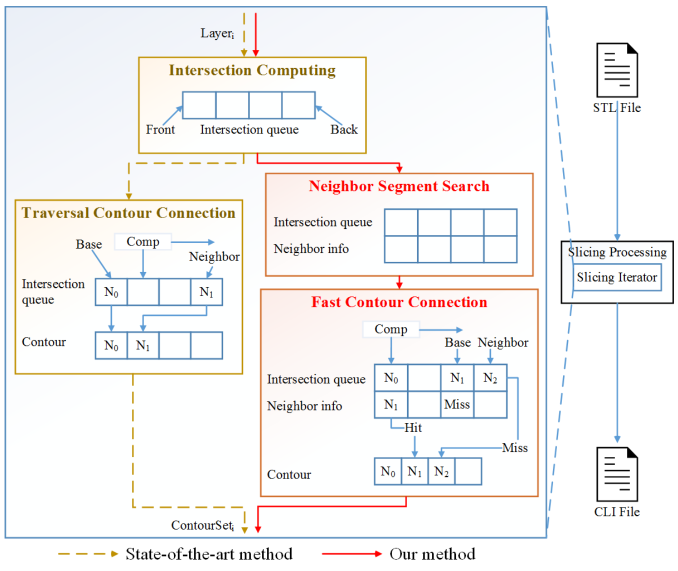

Figure 3: Block diagram of state-of-the-art slicing algorithm and our approach.

in the intersection queue as the *base segment*. By traversing through all the other segments in the intersection queue, the segment with nearest distance to *base segment* is found, which is called the *neighbor segment*. When the *base segment* is connected to the *neighbor segment*, *neighbor segment* is set to be a *base segment*. The above process will iterate until all the segments in the intersection queue are connected. One or more contours will be generated at the end of this stage.

3.2 Analysis of 3D Slicing

In this section, we identify the computational bottleneck in slicing process, and then analyze its characteristics for acceleration.

3.2.1 Bottleneck in slicing process

The time distribution of each stages in state-of-the-art slicing algorithm is indicated in Figure 2, it is observed that contour generation is the time dominant part. This is because searching all other segments in the intersection queue is required to find the neighbor for each intersection segment.

3.2.2 Analysis of slicing process

Here, we take an in-depth analysis on the contour generation stage in stat-of-the-art slicing algorithm and characterize its procedure with two metrics, i.e, search distance and data locality.

Defination 1: For each segment in the intersection queue, the minimum number of segments to find a neighbor segment is called **searching distance**. The comparing sequence is according to the intersection queue order. If the neighbor segment is ahead of this segment, the searching distance is positive. Otherwise its negative.

Defination 2: the overall distribution of searching distance of all the segments in one layer of a STL file is called **data locality**.

In order to understand the correlation of data locality and STL files, we perform a set of experiments with four benchmarks, and the result is shown in Figure 4. It is observed

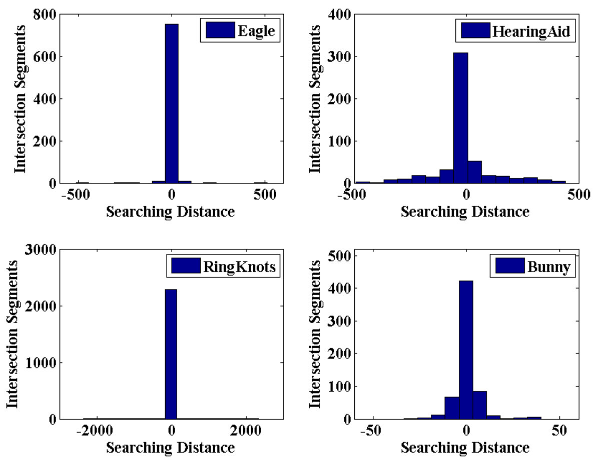

Figure 4: Data locality in four STL files.

that the searching distance of most intersection segments are very small, which means that for most segments, their neighbor can be found by only comparing the segments within a certain range. Furthermore, although data locality is related to STL files, yet most of them has a similar distribution profile with a small variation. This is because the triangles in a STL file is placed neither with a totally random order nor a totally sorted order. In fact, they are placed according to one coordinate of one vertex of each triangle, which is a semi-sorted order. Therefore, after computing the intersection of current layer, these intersection segments should also be in a semi-sorted order. The neighbor of one segment is in a close range with high probability.

3.3 Slicing Acceleration Unit

Based on this observation, we propose our slicing acceleration unit, SAU. Figure 3 shows the block diagram of SAU in red boxes. It follows a new path integrating two new stages. One is *neighbor segment search*, which receives the intersection queue from the *intersection computing* and search the neighbor of each segment in SAU. In fact, SAU can only find the neighbor within the range of SAU size (reasons will be explained in Section 4). If the neighbor is within this range, it will be a SAU hit. Otherwise, it will be a SAU miss. The other stage is *Fast contour connection*. It has two states. If it is an SAU hit, this segment will be connected based on the neighbor information provided by SAU. If it is an SAU miss, however, all other segments in intersection queue needs to be compared with for the neighbor search. We can see that the processing speed on an SAU hit will be much faster than an SAU miss, because it does not need to traverse all the intersection queue to find the neighbor. Therefore, the performance of SAU is related to SAU miss rate.

The relationship of miss rate and speedup is formulated in Equation (1). In this equation, T_{base} stands for the slicing time of state-of-the-art algorithm, T_{SAU} means slicing time of our approach. $T_{overhead}$ means the time for connection on the case of SAU hit. Considering that the $T_{overhead}$ is relatively small comparing to $T_{baseline}$, it can be negligible. Therefore, we can obtain a reversely proportional relationship between miss rate and performance with a scaling factor α in Equation (1).

$$\begin{aligned} Speedup &= \frac{T_{base}}{T_{SAU}} \\ &= \frac{T_{SAU}}{MissRate * T_{base} + T_{overhead}} \\ &\approx \alpha * \frac{1}{MissRate} \end{aligned} \tag{1}$$

There are two factors that influence miss rate. One factor is data locality. If the data locality is more concentrated to the center, the probability that the neighbor of this segment can be found within the range of SAU size is higher, thus less likely a SAU miss occurs. The other factor is the SAU size. The larger SAU size, the more tolerant to data with searching distance far away, hence the lower miss rate. Related experimental results will be shown in Section 5.

On the basis of the above analysis, we discuss the data flow of our approach. For each layer, firstly *Intersection Computing* is done to compute the intersection segment queue. Then this intersection queue is passed to SAU, which search the neighbor of each segment. Then the neighbor information as well as the intersection queue is passed to CPU, which performs *Fast Contour Connection* based on an SAU hit/miss. If there is an SAU hit, connection is done by simply checking the information from SAU. If there is an SAU miss, CPU needs to traverse all the intersection queue to find the neighbor segment and do connection.

4. DESIGN AND ARCHITECTURE

The architecture of SAU is indicated in Figure 5. It shows three levels from top to bottom, i.e., design level, architecture level, and microarchitecture level. Design level describes the two functional module of SAU *distance queue* (DQ) and *arithmetic unit* (AU). Architecture level introduces the Instruction Set Architecture (ISA) modification and communication between SAU and CPU. Microarchitecture illustrates the detailed structure of DQ and AU. In this section, we will firstly introduce the ISA modification for supporting communication of SAU with CPU, and then describe the architecture and microarchitecture of SAU.

4.1 ISA Support

Figure 5 shows that SAU is tightly coupled to CPU, and their communication is implemented in the execution stage. The ISA of CPU is extended, and two more instructions are added to support the communication of CPU with SAU as follows:

- dqify %r, %t: CPU send one value from register r to entry t of SAU.
- dqofy %r, %t: CPU fetch one value from entry t of SAU to register r.

For ease of the implementation, we use inline assembly to specify these new instructions in the code implementation of the slicing algorithm.

4.2 SAU Architecture

SAU contains two functional modules: DQ and AU, as shown in Figure 5. DQ is used to store intersection segments, and the number of entries is defined as the SAU size.

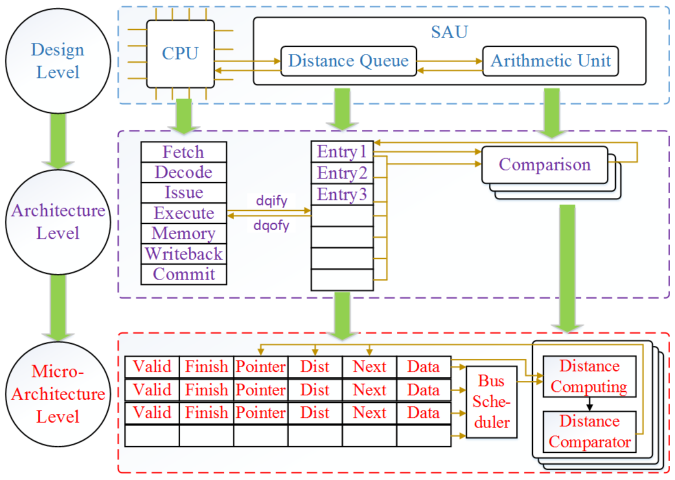

Figure 5: SAU Instruction Set Architecture.

Each entry in DQ stores one intersection segment, and it will compute the distance with another entry in every cycle. The computing ends up when this entry has compared with all other entries in DQ. If the intersection queue is smaller than the SAU size, then SAU will be able to find the neighbor among all the segments. If intersection queue is larger than SAU size, however, new coming intersection segments will replace old segments. Thus SAU can only find the neighbor of each segments within the range of SAU size.

Microarchitecture level shows the details of each entry in DQ. Each entry contains six sub-entries, and the intersection segment is saved in *Data*. *Pointer* is to store which entry it is comparing with at this cycle. At the end of this cycle, *Pointer* will be updated. *Dist* denotes *distance*, which stores the closest distance up to date. *Next* is a pointer that directs the entry of its closest neighbor. When this entry finishes computing, the *Complete* flag will be set to one, which means it is ready for output.

AU is associated with each DQ entry. It contains two stages: (1) distance computing which computes the distance of two segments, and (2) distance comparison which compares the computed distance with previous closest distance, and choose the smaller one as the current closest distance.

5. EVALUATION

In order to evaluate the performance of SAU, we propose a standard benchmark suite of 3D printing, and apply it to SAU. We test the performance improvement of SAU over unmodified CPU, followed by measuring the parameter sensitivity and trade-offs of SAU.

5.1 Benchmark Characterization

We choose seven STL files for the standard benchmark suite, as shown in Figure 6. The details are listed in Table 1. It has diversity in classification, which covers seven different categories from daily products to industry parts. For each category, one frequent-used benchmark is chosen as representative. These benchmarks also differ in complexity, and the number of triangles varies from thousands to hundreds of thousands. In order to emulate the real-world manufacturing, we apply 200 μm manufacturing technique, which is wildly used in 3D printing industry. Specifically,

Figure 6: The selected seven benchmarks.

Table 1: Benchmark Standardization

Name	Triangle	Height	Layer	Description
Cylinder	3160	10.0	50	geometry
Club	3290	20.1	100	Tool
Eagle	17291	12.0	60	Animal
HearingAid	32762	15.9	79	Health
RingKnots	33730	17.3	86	Living
Bunny	69664	46.3	230	Classical
TurbineBlade	124366	24.0	120	Industry

these benchmarks are sliced every 200 μm. The number of layers are also listed in Table 1.

5.2 Simulation Setup

In the experiment, we use gem5 [2] as the simulation platform. The gem5 simulator is a open-source simulation platform for computer-system architecture research, encompassing system-level architecture as well as processor microarchitecture. It supports customized architecture design and evaluation. The parameters we use to configure gem5 model are shown in Table 2.

5.3 Performance Evaluation

In this experiment, we would like to investigate the performance improvement of SAU comparing to conventional unmodified CPU. We apply the standard benchmark suite for SAU and CPU, respectively, and the total number of cycles is used to quantitatively evaluate time performance. The experimental result is shown in Figure 7. Compared to the unmodified CPU, SAU demonstrates a remarkable performance benefit'. The average speedup improvement is 12.1x, with a standard derivation of 5.5, and the highest speedup is 22x.

In order to tackle deeper into the relationship between performance and testbench, we take a look at Figure 8, which shows a box plot of miss rate over these standard benchmarks. Each column indicates miss rate of each layer for

Table 2: Gem5 Simulation Setup

Parameter	Configuration
ISA	ARM
CPU	O3
Fetch & Issue & Commit Width	3 & 8 & 8
Load & Store Queue Entries	16 & 16
Issue & ROB Queue Entries	32 & 40
L1I & L1D Cache Size	32kB & 32kB
L1 Associativity	2
L2 Cache Size	1MB
L2 Associativity	16
DQ Entries	200
SAU Communication Latency	1 Cycle

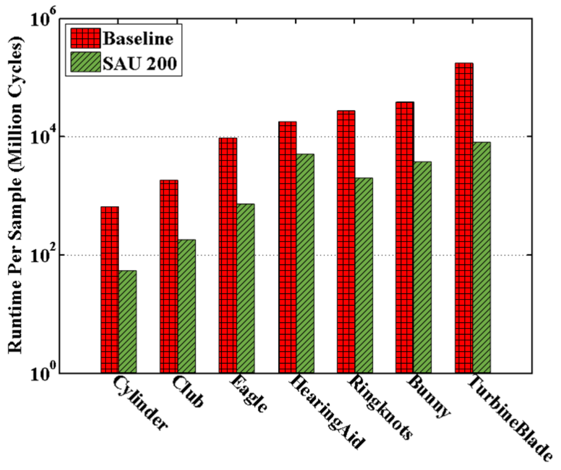

Figure 7: Performance comparison of SAU with general CPU on a standard benchmark suite.

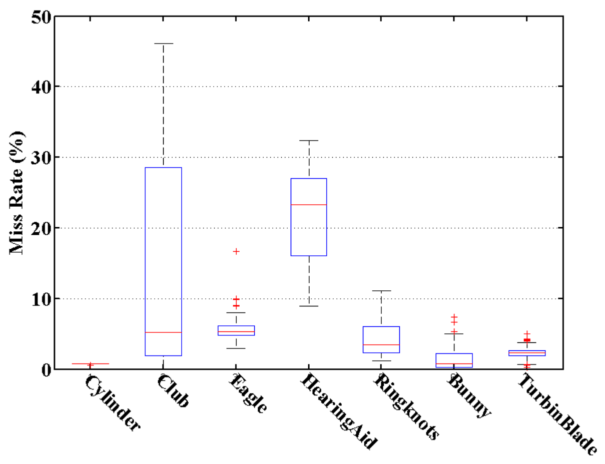

Figure 8: Miss rate among different benchmarks.

each benchmark. The average miss rate of all these benchmarks is 7.8%, with a standard derivation of 8.2. We can see that there is a variation in miss rate for these benchmarks because miss rate depends on data locality. Different benchmarks have different data locality, as indicated in Figure 4. These difference, although small, affects their miss rate. For example, the data locality for *HearingAid* is not as concentrate to the center as *RingKnots* or *Bunny*. Therefore, the miss rate of *HearingAid*, 23.0%, is larger than that of *RingKnots*, 4.3%, or *Bunny*, 3.1%, as indicated in Figure 8. As shown in Figure 7 and 8, we can see that, generally, there is a reverse proportion between miss rate and performance. This observation justifies that data locality will affect miss rate, which influence the performance of SAU, as mentioned in Section 3.

5.4 Parameter Sensitivity

In this experiment, we test the parameter sensitivity and trade-offs. Our goal is to explore the trade-offs between the SAU size and performance. We change the number of entries in DQ from 100 to 800 and test the corresponding

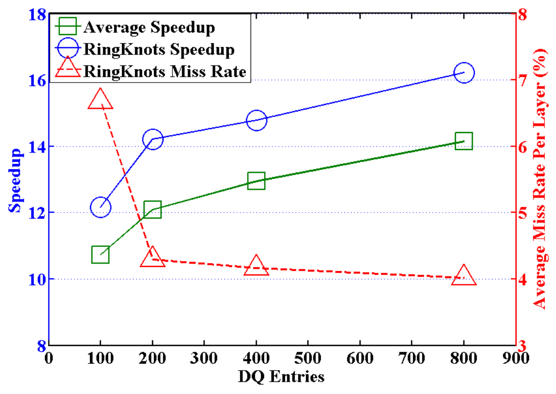

Figure 9: Speedup and miss rate for different DQ entries.

speedup. We apply the same standard benchmark suite, and the experimental result is shown in Figure 9.

There are three curves in this figure. Square indicates the average speedup of these seven benchmarks. Circle indicates the speedup of a specific benchmark, *RingKnots*, and triangle is used to show the corresponding miss rate. The square shows that there is a trade-off between speedup and SAU size. The larger SAU size, the higher speedup. As the number of DQ entries grows from 100 to 800, the average speedup increases from 10.7x to 14.2x. In order to investigate the relationship of performance and SAU size, we focus on one benchmark, *RingKnots*. As SAU size increases from DQ entries 100 to 800, miss rate decreases from 6.7% to 4%, and speedup increases from 12.1x to 16.2x. The result echoes the analysis in Section 3 that SAU size will affect miss rate, which influences the performance of SAU. A more interesting observation is that both miss rate and speedup have a turning point at DQ entries 200, after which they become saturated. The reasons are illustrated as follows: all STL files share a similar data distribution that concentrate on searching distance, as shown in Figure 4, which has very high bins in the center. On the other hand, there will be an SAU hit for all the segments within the searching distance of SAU size, and miss for the rest. When SAU size increases to be larger than the volume of the high bin regions in Figure 4, the DQ hit rate becomes saturated. There is a turning point of miss rate in Figure 9 because of the DQ $overflow$. In terms of the turning point of speedup, because SAU speedup and miss rate has a reciprocal relationship as described in Equation (1) in Section 3, and miss rate has a turning point at DQ entries 200, the speedup incremental rate will also have sudden drop at this point.

6. CONCLUSION

3D printing is changing the world of manufacturing industry rapidly, among which computing is becoming a bottleneck. Slicing, from the perspective of significance and time consuming, is the most worth-researching and challenging stage. In order to solve this challenge, we review the state-of-art slicing algorithm and find it's implied character, data locality. Based on this observation we present and evaluate a hardware acceleration unit, called SAU. The experimental result under a standard benchmark suite shows that, comparing to general CPU, SAU has a speedup up to 22x. This performance improvement is unaffected by the scale of STL file. but only related to miss rate. There are still works to be done regarding this research, such as further improving the performance of SAU, finding an optimal solution between space and performance, and finding a math function to describe data locality.

7. REFERENCES

[1] Common layer interface (cli). `https://www.forwiss.uni-passau.de/~welisch/papers/cli_format`. Accessed: 2015-11-22.

[2] The gem5 simulator. `http://www.gem5.org/Main_Page`. Accessed: 2015-11-18.

[3] D. Bak. Rapid prototyping or rapid production? 3d printing processes move industry towards the latter. *Assembly Automation*, 23(4):340–345, 2003.

[4] B. Berman. 3-d printing: The new industrial revolution. *Business horizons*, 55(2):155–162, 2012.

[5] S. Chen and W. Chen. A new level-set based approach to shape and topology optimization under geometric uncertainty. *Structural and Multidisciplinary Optimization*, 44(1):1–18, 2011.

[6] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam. Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. In *ACM SIGPLAN Notices*, volume 49, pages 269–284. ACM, 2014.

[7] J. Cong and Y. Zou. Fpga-based hardware acceleration of lithographic aerial image simulation. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 2(3):17, 2009.

[8] J. Excell and S. Nathan. The rise of additive manufacturing. *The engineer*, 2010.

[9] C. Kirschman and C. Jara-Almonte. A parallel slicing algorithm for solid freeform fabrication processes. *Solid Freeform Fabrication Proceedings, Austin, TX*, pages 26–33, 1992.

[10] E. Kroll and D. Artzi. Enhancing aerospace engineering students' learning with 3d printing wind-tunnel models. *Rapid Prototyping Journal*, 17(5):393–402, 2011.

[11] M. S. Lavine. Fast, continuous, 3d printing. *Science*, 347(6228):1325–1325, 2015.

[12] B. Leukers, H. Gülkan, S. H. Irsen, S. Milz, C. Tille, M. Schieker, and H. Seitz. Hydroxyapatite scaffolds for bone tissue engineering made by 3d printing. *Journal of Materials Science: Materials in Medicine*, 16(12):1121–1124, 2005.

[13] S. McMains and C. Séquin. A coherent sweep plane slicer for layered manufacturing. In *Proceedings of the fifth ACM symposium on Solid modeling and applications*, pages 285–295. ACM, 1999.

[14] S. Mueller, S. Im, S. Gurevich, A. Teibrich, L. Pfisterer, F. Guimbretière, and P. Baudisch. Wireprint: 3d printed previews for fast prototyping. In *Proceedings of the 27th annual ACM symposium on User interface software and technology*, pages 273–280. ACM, 2014.

[15] E. Sabourin, S. A. Houser, and J. Helge Bøhn. Adaptive slicing using stepwise uniform refinement. *Rapid Prototyping Journal*, 2(4):20–26, 1996.

[16] K. Tata, G. Fadel, A. Bagchi, and N. Aziz. Efficient slicing for layered manufacturing. *Rapid Prototyping Journal*, 4(4):151–167, 1998.

[17] M. Vatani, A. Rahimi, F. Brazandeh, and A. S. Nezhad. An enhanced slicing algorithm using nearest distance analysis for layer manufacturing. In *Proceedings of World Academy of Science, Engineering and Technology*, volume 37, pages 721–726, 2009.

[18] L. Wu, A. Lottarini, T. K. Paine, M. A. Kim, and K. A. Ross. Q100: the architecture and design of a database processing unit. In *ACM SIGPLAN Notices*, volume 49, pages 255–268. ACM, 2014.