Proceedings of the 2018 Manufacturing Science and Engineering Conference
MSEC2018
June 18-22 2018, College Station, Texas, USA

MSEC2018-6322

AUTOMATED MULTI-USER ANALYSIS OF VIRTUALIZED VOXEL-BASED CAM ON SHARED GPUS

Roby Lynn
George W. Woodruff School of Mechanical Engineering
Georgia Institute of Technology
Atlanta, GA, USA

Roberto Leo Medrano
George W. Woodruff School of Mechanical Engineering
Georgia Institute of Technology
Atlanta, GA, USA

Didier Contis
College of Engineering
Georgia Institute of Technology
Atlanta, GA, USA

Tommy Tucker
Tucker Innovations, Inc
Waxhaw, NC, USA

Thomas Kurfess
George W. Woodruff School of Mechanical Engineering
Georgia Institute of Technology
Atlanta, GA, USA

ABSTRACT

Computer-aided manufacturing (CAM) software allows for the generation of toolpaths for computer numerical control (CNC) machine tools and enables the creation of sophisticated parts that would not otherwise be possible with conventional manual machining methods. Voxel-based CAM is a recent approach to toolpath planning that enables creation of paths for parts that would be difficult to create with traditional CAM software. However, the use of voxel-based CAM necessitates the presence of powerful hardware (specifically, graphics processing units) in order to perform the necessary computations for creating toolpaths. The concepts of virtualization and desktop-as-a-service offer a promising solution to this challenge, as they allow for many users to access computer hardware that is hosted on a single server. This work investigates the performance impact caused by multiple simultaneous users on voxel-based CAM deployed in a virtualized environment. The implementation of a Python application for multi-user simulation on the virtualized platform is described and timing results gathered from a sequence of simulations are presented and analyzed as the number of users is varied. The results from these simulations demonstrate consistent operational times for a low number of simultaneous users before a period of high performance variation due to resource sharing.

INTRODUCTION

The use of computer-aided manufacturing (CAM) software to plan machining operations is essential for the generation of toolpaths for modern computer numerical control (CNC) machine tools. CAM software reduces the likelihood of manufacturing errors in parts that could be introduced by manual programming and enables more rapid planning and analysis of a machining operation. Traditional CAM software generates toolpaths using analytical methods, where the paths are represented by parametric curves that follow the three-dimensional model of a part. This research employs voxel-based CAM, a promising alternative to traditional parametric software, which represents models as a collection of discrete cubes known as voxels. A voxelized part model is effectively a three dimensional array of voxels, where the presence or absence of material at a given voxel can be represented by a binary value. As a result, voxel models are capable of representing any geometry without needing to fit a curve to the model's surfaces. Voxel-based CAM has a number of advantages over traditional parametric CAM: it is better able to represent complex, freeform surfaces that would be difficult to describe with analytical curves; it enables simpler collision checking between a cutting tool and a workpiece; and the calculation and simulation of material removal along a toolpath consists of a simple summation of removed voxels [1].

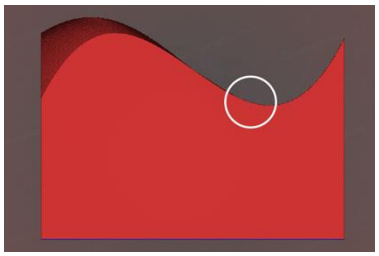
Because the voxel model is a three dimensional array, the speed of CAM operations on the model can be accelerated using a parallel processing platform. Modern graphics processing units (GPUs) are purpose-built for parallel processing, and thus serve as an ideal platform for implementation of the algorithms necessary for voxel-based CAM [2]; NVIDIA GPUs with

compute unified device architecture (CUDA) are an example of such a platform. However, not all computing devices have a GPU installed, and the purchase of a standalone GPU-equipped workstation for each user of the CAM software can be prohibitively expensive. Desktop-as-a-service (DaaS) systems are an alternative to standalone workstations: using DaaS, virtualized desktop environments can be provided to multiple simultaneous users using only a single server machine. Virtualization of a server with a powerful GPU installed allows any user with an Internet connection access to the parallel processing platform. From the user's perspective, the virtual desktop behaves like a physical machine; from the administrator's perspective, however, one server can be used to economically provide multiple cloud-based virtual machines (VMs) to a group of remote users [3]. A hypervisor, such as Citrix XenServer, creates and administers instances of the VMs, dividing the host environment's compute resources among the guest users. As a result, the powerful computer hardware required to efficiently run voxel-based CAM software can be made accessible to manufacturing engineers using only one GPU.

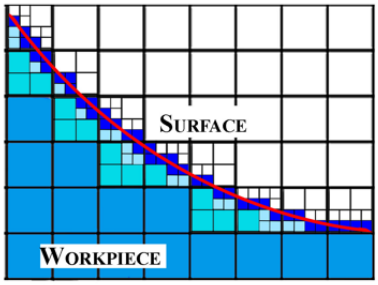
Previous work using NVIDIA's Kepler GPU architecture has demonstrated the feasibility of performing voxel-based toolpath planning on GPU-equipped VMs; however, sharing of the compute resources on the server machine can be problematic and lead to decreased performance of the CAM software [4], [5]. The aim of this research is to develop an experimental protocol for the automatic performance analysis of voxel-based CAM on a cloud-hosted DaaS system equipped with a current-generation NVIDIA Maxwell GPU; specifically, the computation times of various toolpath planning operations on the DaaS system are measured as the number of simultaneous users on the machine is varied. Results of these measurements characterize the performance degradation caused by large numbers of simultaneous users. The resulting data can be used to properly design DaaS systems for voxel-based CAM deployment. The remainder of this paper is organized as follows: first, the toolpath planning algorithms used to benchmark the DaaS system are described; second, the methodology for automating performance analysis is presented; next, results from running a series of multi-user trials on a laboratory DaaS system are shown and analyzed; and finally, directions for future work are discussed.

VOXEL-BASED CAM

This research leverages a software known as SculptPrint [6], which is a GPU-accelerated voxel-based CAM system for 5-axis toolpath planning. In SculptPrint, models are represented models using individual voxels; as pixels are small squares that comprise a two-dimensional image, voxels are small cubes that compose a complex three-dimensional model [7]–[9]. With models represented this way, the accuracy of highly complex shapes and surfaces is only dependent on individual voxel size. A cross-section view of a three-dimensional voxel model is shown in Figure 1a, and an enlarged view of the voxelized surface representation is shown in Figure 1b. The dark squares are the voxels that comprise the surface, and the smooth line through the



a. 2D Cross-Section View of a 3D Voxel Model



b. Enlarged View of Surface Voxels

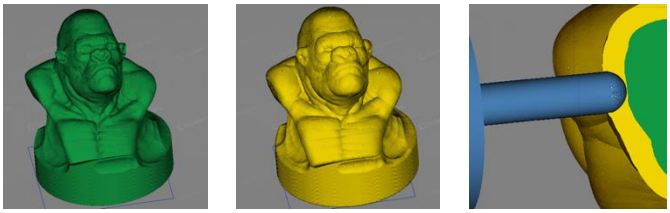
Figure 1. Voxelized Surface Representation

surface voxels represents the equivalent analytical representation of the surface. Upon completion of the toolpath planning operation, the voxel-based CAM system creates G-Code suitable for execution on a CNC machine tool.

Volumetric Offsetting

Surface offsetting is a critical operation in 5-axis toolpath planning that determines allowable cutting tool positions in Cartesian space for a given part geometry. An offset surface is defined as a surface created from a starting surface where all points of the offset surface lie at a constant distance from the starting surface along the normal direction of the starting surface. An offset volume is simply the volume that is enclosed by a surface offset generated from a part model. Offsetting a surface with a constant equal to the radius of a given tool, perhaps with an added cutting allowance, enables the generation of an offset volume that is usable for toolpath planning; in the case of voxel-based CAM, the offset volume is generated following the algorithms presented in [10] and [8].

The surface of the offset volume represents the collection of points where the spherical center of a ball-end cutting tool can reside without overcutting the target part geometry. In effect, a contact volume determines the maximum amount of material a given pass can remove without contacting the final end volume.



a. Target Voxel Model

b. Generated Offset Volume For Voxel Model

c. Cross-Section View of Offset Volume

Figure 2. Volumetric Offsetting for Toolpath Planning

Figure 2 demonstrates the use of an offset volume: Figure 2a shows an example voxel model, which represents the target part; Figure 2b shows the result of a volumetric offset by the radius of a certain cutting tool; and Figure 2c shows a cross-sectional view of the same voxel model where the spherical center of a ball-end tool is following the surface of the offset volume. In this case, the offset amount is equal to the radius of the cutting tool. Thus, the tool tip will tangentially touch the final part surface during machining. A toolpath can be created along the final offset surface by connecting adjacent voxel centers with straight lines; the resulting toolpath will therefore consist of many small linear movements known as steps.

Accessibility Analysis

In the case of 5-axis machining, the CAM software is responsible for determining axis commands that control both the position and orientation of a cutting tool. The five effective degrees of freedom of the cutting tool are shown in Figure 3. The cutting

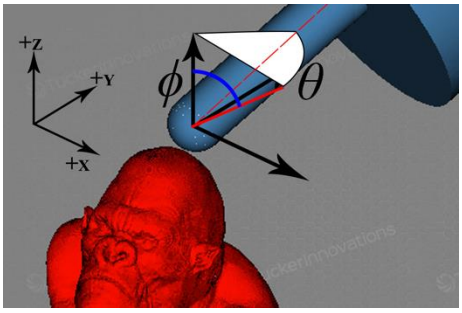
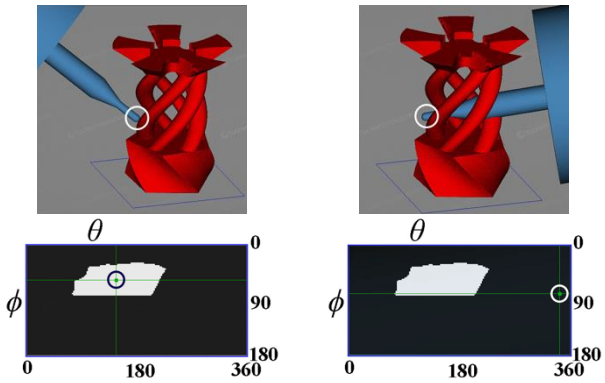


Figure 3. Degrees of Freedom for a 5-axis Machine Tool (Tool not to Scale)

tool has three translational degrees of freedom (X, Y and Z) and two rotational degrees of freedom (θ and φ). A tool orientation must be assigned at the beginning and end of every step of toolpath created using the offset surface. Efficient tool orientation assignment will result in a toolpath where both rotary and translational axis accelerations change smoothly.

The rotary axis positions that define the tool orientation at every step of a voxel-based toolpath can be determined using the accessibility map algorithm developed and presented in [10], [11] and [12]. In SculptPrint, the accessibility map algorithm is used at each step to check every realizable tool orientation for collisions between the tool geometry, the workpiece, and the workholding assembly. The set of realizable tool orientations consists of the combinations of θ and φ that do not violate the physical axis limits of the machine. The creation of an accessibility map at every step of a toolpath results in a series of binary images, collectively referred to as access maps, that define accessible and inaccessible space for each step. Accessible space is the collection of tool orientations for a given step that do not result in a collision between the cutting tool assembly and the workpiece or workholding; in contrast, inaccessible space is the collection of orientations that do result in a collision.

A graphical explanation of the accessibility map algorithm is presented in Figure 4, where two distinct tool orientations are displayed for a complex part. The green dot representing a given



a. An Accessible Point *b. An Inaccessible Point*

Figure 4. Accessibility Analysis

tool's θ and φ angle positions is located at different coordinates in the two images. Figures 4a and b show the resulting accessibility map for this step, where realizable θ and φ angles are plotted on the horizontal and vertical axes of the map, respectively. The current tool orientation is denoted by the green dot on the map. If the dot is located inside the white accessible space, as it is in Figure 4a, the tool orientation does not result in a collision; conversely, if the dot is located in the surrounding black inaccessible space, as it is in Figure 4b, the orientation results in a collision between the part and the tool. Once the maps for each step have been determined, an access path can be formed through accessible space. Figure 5 shows an example curve through accessible space on sequential accessible maps. The

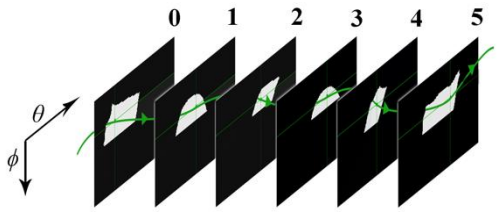


Figure 5. Access Path Through Accessible Space

resulting access path defines the progression of tool orientation along the toolpath.

The accessibility map implementation in SculptPrint grants the user control over the resolution of the maps by parameterizing the quantization interval of rotary axis position. For some toolpaths (such as those on a purely convex surface), low resolution maps may be sufficient; for more complex parts, such as the one shown in Figure 4, higher resolution maps are required to ensure the toolpath is collision free. Varying the resolution of the maps changed the orientation difference per pixel of the map; a smaller orientation change per pixel provides more precision within a map, indicating a higher resolution. Both volumetric offsetting and accessibility analysis are required operations for the generation of a toolpath from SculptPrint. Each operation is computationally expensive, and thus execution

of these operations constitutes a large portion of overall toolpath generation time.

GPU VIRTUALIZATION AND RESOURCE SHARING

Virtualization and sharing of GPUs has shown promise as an effective way to provide access to a parallel processing platform to remote users through a DaaS strategy. Numerous researchers have already demonstrated that GPUs distributed through this strategy accelerate operation performance in a virtual environment. Vinaya, *et al* demonstrated that a Xen PCI-passthrough virtualization configuration performed well, but required an entire dedicated GPU per VM [13]. Gupta, *et al* addressed this concern by developing a front-end virtualization solution, GViM, that allowed multiple VMs to draw on the power of a single GPU [14]. Xiao, *et al* corroborated this approach by demonstrating VOCL, a framework that supports shared GPUs between VMs [15]. The development of this approach suggests the possibility of offshoring scientific or engineering operations to a cloud-based server equipped with powerful hardware, bypassing the need for the user to have access to such hardware locally.

Virtualization of Voxel-Based CAM

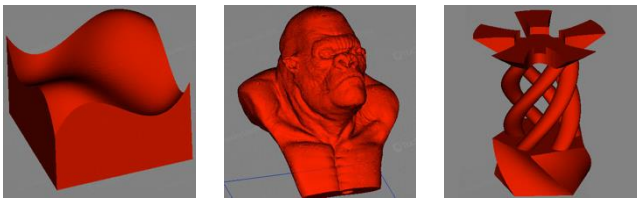
Despite the stringent hardware power requirements necessary to perform toolpath planning with voxel models, successful deployment of the voxel-based CAM system on a DaaS platform for an educational environment has been demonstrated by Lynn, *et al* [16]–[18]. The educational deployment relied on the implementation of a GPU-accelerated DaaS system equipped with an NVIDIA GRID (Kepler) GPU developed in [4] and [5]. The system consisted of two Windows Server 2012 R2 machines constructed in Georgia Tech's virtual laboratory (Vlab) and was successful in providing CAM access to ten simultaneous users. Windows Server 2012 R2 allows multiple users to share the same processor, memory, and GPUs on separate desktop sessions, through the use of Microsoft's Remote Desktop Protocol (RDP) [19]. RDP adds a layer of abstraction between the user and the computer by distributing the computational power of the GPUs and other hardware on a physical machine across all users [20]. However, this distribution of resources among users results in a performance impact to any user operations performed on the VM [21].

For the best user experience, the total operation times for voxel-based CAM should remain sufficiently low so as to not substantially impact performance. Even though the presence of more powerful server hardware accelerates CAM operation performance and offsets the impact that resource sharing causes, there still exists a quantifiable performance impact that varies based on the number simultaneous users. This work aims to characterize this impact using a DaaS system that is more powerful than the one implemented in previous works; the resulting performance analysis can be used to provide recommendations on the user load for a DaaS system for voxel-based CAM.

EXPERIMENTAL PROTOCOL

The DaaS system developed for this work is a Dell R720 containing two Intel Xeon E5-2680-v3 CPUs, each of which has eight physical cores and runs on a 2.2 GHz clock. The system has 192GB of memory, four Intel X520 10GB network adapters, and one NVIDIA Tesla M60 (Maxwell) GPU. The hypervisor used in this system is Citrix XenServer 6.5 SP1. The VM used for performance evaluation was configured with Windows Server 2012 R2, two virtual CPUs (with four cores each) and direct passthrough of one of the GPUs on the M60 [22].

Three different voxel models of varying complexity were used to text both offsetting and accessibility analysis performance under varying user load. Figures 6a-6c show the



a. Wiggle *b. Kong* *c. Candle Holder*

Figure 6. Voxel Models Used for Analysis

three voxel models used in this work: from left to right, the models are referred to as the Wiggle, the Kong, and the Candle Holder. Numeric data on these models is shown in Table 1. The

Table 1. Surface Complexity Metrics for the Voxel Models Used for Analysis

Model	Number of Surface Voxels	Number of Toolpath Steps
Wiggle		
Kong		
Candle Holder		

computation time for each operation on each model was measured for each number of simultaneous users to complete one trial; once the data for all users had been collected, the trial was repeated 5 times to give a range of measurements.

Automation of Multi-User Analysis

Assembling a large number of human users to simultaneously test VM performance is logistically difficult, so a Python application was developed to simulate a typical CAM session and automate the analysis procedure. The application implemented Python's Multiprocessing module to instantiate a variable number of simultaneous SculptPrint processes, where each one represented an individual CAM user [23]. Once all the user processes were created, the application instructed them to run both offset volume creation and accessibility analysis on the three different voxel models. This simulates a "worst-case" scenario in a DaaS configuration, in which all users request the same operation at the same time. Algorithm 1 describes the procedure used to simulate the simultaneous users. After the user processes completed an operation on a part, the Python application calculated the total time spent to perform the CAM operation and stored it in an output array.

The Python application simulated a grouping containing a one to ten simultaneous users. Each of these groupings was designated a trial; the application ran numerous trials to ensure that detected trends were valid and all time data were averaged

Algorithm 1: Simulate Multiple Users

Inputs: Number of consecutive users to simulate, ***numUsers***, number of trials to run, ***numTrials***, *and CAM operation to perform,* ***Operation***
Output: Array of data ***data*** holding operation times for each users across each trial

```
1: foreach trial in numTrials do
2:     Instantiate processes ← list of simultaneous processes of length numUsers
3:     foreach process in processes do
4:         Perform Operation in process simultaneously with each other process in processes
5:         Measure and append total operation time for process to data
6:     end
7: end
8: return data
```

across the trials. Two correction factors, $k_{\text{Offsetting}}$ and $k_{\text{Accessibility Maps}}$, were used to account for the three voxel models' different levels of geometric complexity. The first correction factor, $k_{\text{Offsetting}}$, was used to scale the offset time by the number of surface voxels in the model and is given by Equation 1,

$$k_{\text{Offsetting}} = \left(\frac{V_{\text{Boundary}}}{s^3} \right)^{-1} \quad (1)$$

where V_{Boundary} is the summation of volume that is occupied by boundary voxels and s is the side length of a voxel; multiplying a given offset time by $k_{\text{Offsetting}}$ is therefore equivalent to dividing the time by the number of voxels on the surface of the model. As a result, the reported offset time is actually the offset time *per surface voxel*. The correction factor for accessibility maps, $k_{\text{Accessibility Maps}}$, is given by Equation 2,

$$k_{AccessibilityMaps} = \left(\sum \text{Step}_i \right)^{-1} \quad (2)$$

where Step_i is a step along the toolpath used for accessibility analysis. Thus, multiplying access map computation time by $k_{\text{Accessibility Maps}}$ is equivalent to dividing the time by the total number of steps (where each step requires one map to be created) to yield computation time *per accessibility map*. The use of these correction factors enables direct comparison of time results across different models by normalizing results to a unit complexity.

Multi-User Volumetric Offsetting Performance

The first CAM operation simulated using the Python application was the creation of surface offsets. In this phase of the procedure, up to ten simultaneous users were simulated on the three sample parts, and each user generated a surface offset with a distance of 3.175 mm. Figure 7 displays the aggregated results of the offset procedure for the three parts. Operation times were averaged across each user performing the operation and across the set of

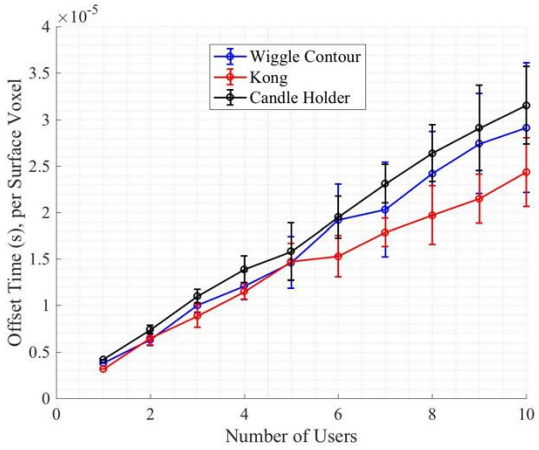


Figure 7. Normalized Offset Operation Times for Variable User Load

trials. Prior to normalization, the operation times varied significantly due to the different complexity of the parts; operational times for the Candle Holder were especially high, due to its irregular geometry creating an increased surface from which to offset. The resulting data were then normalized using the k_{Offset} correction factor. The general relation between the number of users and the operation times is linear, regardless of part. The error bars denoting one standard deviation widen with the increasing number of users, demonstrating increasing variation in operational times as the number of users increases.

Variability in computation time can be have a large effect on user experience, as the responsiveness of the system will change depending on load. Figure 8 shows the computation time

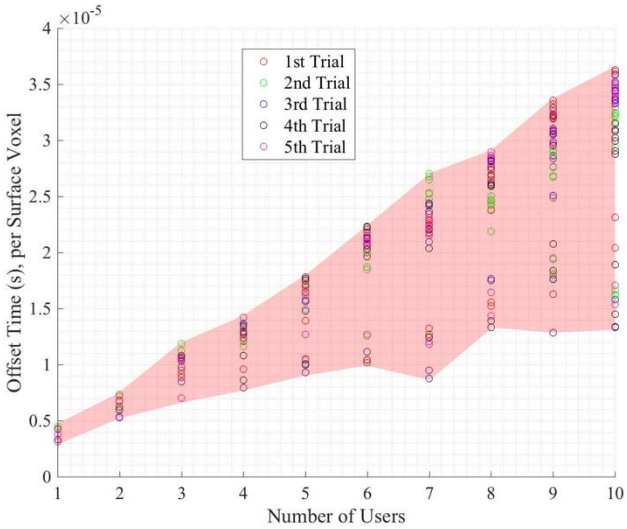


Figure 8. Raw Offset Operation Times for Wiggle Part

response for the offset simulation for the Wiggle part. In this figure, the times are not averaged across users or trials. The red highlighted area denotes the range of computation time, which increases in magnitude as the number of users increases. This suggests that, although all users are performing identical operations on the machine simultaneously, the performance

experienced by some users will be different from that experienced by others.

Multi-User Accessibility Map Performance

The next operation tested was the creation of accessibility maps for a generic toolpath. Similar to the results of the offset operation, the data from the Candle Holder were significantly higher than for the other parts, due to its irregular geometry. Normalization of computation time was performed by multiplying the measured results by $k_{\text{Accessibility Maps}}$ from Equation 2. Both low resolution (2 degree orientation change per map point) and high resolution (1 degree orientation change per map point) access maps were generated in separate instances of the experiment to determine the precise impact that map resolution had on operation times. Because the accessibility map resolution is a user-defined parameter, it is logical to quantify the performance of the VM at different map resolutions.

Figure 9 shows the results for the accessibility map generation experiment, where computation times were averaged

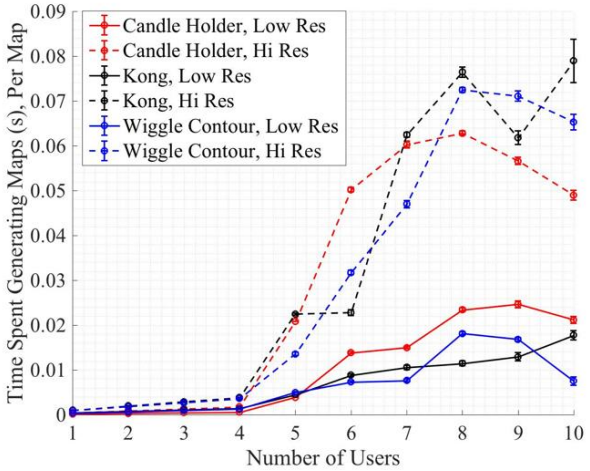


Figure 9. Normalized Accessibility Map Generation Times for Variable User Load

across users and across trials. Several trends can be observed from these results: the normalized operation times for the higher resolution maps are larger than the times for the lower resolution maps by roughly a factor of eight, especially for a higher number of users, suggesting that access map computation time is not linearly proportional to the number of map points; the operation times are linearly increasing for user count less than or equal to the number of CPUs on the VM (user range 1-4); the operation times follow a generally increasing trend above four users, although they do so nonlinearly; and finally, for the majority of the simulated user range, the operation times increase. However, when the user count is high enough (above 8 simultaneous users), the average operation times can actually begin decreasing; this phenomenon is especially evident in the higher resolution maps. This behavior is caused by operating system (OS)-level scheduling of compute resources [24]; because the experiment only measures the elapsed time of the operation (and does not start the clock when the operation is *requested*), the results suggest that some users are forced to wait until others complete. This trend is especially apparent in Figure 10, which shows the results of sixteen simultaneous users performing the

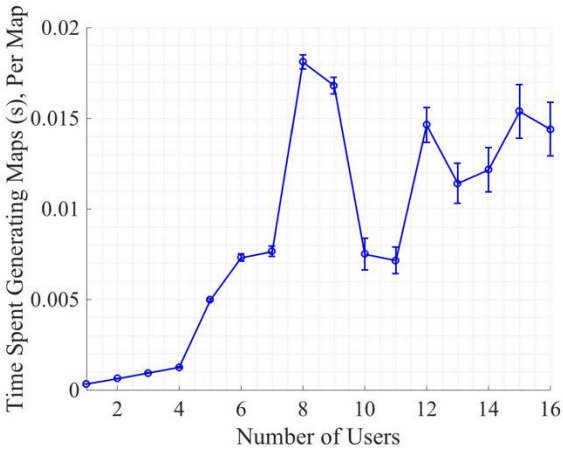


Figure 10. Access Map Generation Times for Simulated Users 1-16 on Wiggle Part

Wiggle low resolution accessibility map generation procedure.

Figure 11 shows the raw time results for the low-resolution access map simulation using the Candle Holder part. Similar to the raw results for the volumetric offset experiment, significant variation across the conducted trials can be observed. The access

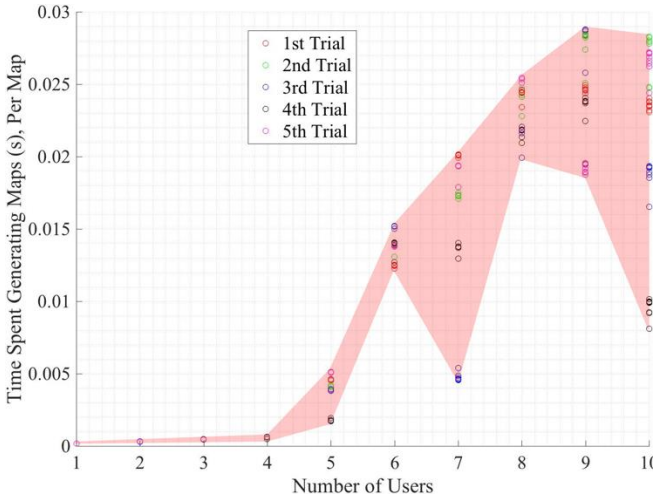


Figure 11. Full Raw Data for the Access Maps Operation on Candle Holder

map operation was determined to be linear to the number of CPUs on the machine, and there is very little variation across users or trials within this range.

DISCUSSION

For a successful deployment voxel-based CAM on a DaaS system, performance must be considered to ensure user experience acceptable. Compared to the results from Lynn, *et al*, the operational times within this work are significantly lower [5]; this reflects the choice of the more powerful M60 Tesla GPU within this system configuration over the older GRID K1 card. Given that GPU compute capability is advancing at a rapid rate, operational times for voxel-based CAM will become more

efficient as time progresses. For a low count of simulated users, the operational times for both offsetting and accessibility analysis follow a monotonically increasing linear trend with low variation. This promising result suggests that, if mass deployment is desired, the operator of the DaaS system should control user load to keep performance in the linear region for each operation to guarantee that each user will experience consistent and predicable performance when running CAM operations. As user count increases, however, variation in operation times rise for both offsetting and accessibility analysis. For a high enough user count, both the mean and range of the operational may be too unpredictable to provide acceptable user experience; for these user counts, variation across the separate trials is high, with some users in a trial performing the test operations in drastically less time than in other trials. Table 2

Table 2. Mean, Range and Sample Variance of Hi-Res Access Map Operational Times Across Simulated Users, Candle Holder Part

User Count	μ (s)	Range (s)	σ (s)
1	4.7425E-4		
2	9.3083E-4	2.8043E-7	1.9829E-7
3	1.3482E-3	1.8297E-5	9.6723E-6
4	1.6371E-3	8.9140E-6	3.9167E-6
5	1.9667E-2	1.4547E-4	6.4709E-5
6	1.1897E-2	1.0326E-3	4.9881E-4
7	7.0359E-2	4.4056E-3	1.4983E-3
8	8.8680E-2	1.1019E-3	3.0229E-4
9	7.7385E-2	3.0085E-3	9.1130E-4
10	6.9266E-2	3.7155E-3	1.1652E-3

demonstrates the mean, range, and standard deviation of normalized hi-res access map operational times for simultaneous users 1-10 using the Candle Holder part. Analysis of these results demonstrates that, as expected, average times increase as a greater number of users causes increased performance degradation. Of interest, however is that the standard deviation of operation time between users drastically increases as the number of simultaneous users increases. This produces undesirable inconsistencies in operation time caused by OS and hypervisor resource scheduling, and some users sharing the VM would see their operations require more time than other users. While the plots may show some user operations completing faster than others, in reality the total operation times always increase; the application only recorded the elapsed time between the beginning of the operation and its end, as opposed to recording the time at which the user requested the operation.

From the access map Figures, it is apparent that there exists some number of simultaneous users past which significant resource scheduling occurs, as characterized by a nonlinear trend in operation time. If voxel-based CAM is to be deployed on a mass-user scale, the number of users should not exceed this critical number to ensure consistent performance. For the offset operation, the entire trend was linear, so the count of ten users was below this critical number. The offset procedure is heavily dependent on the GPU, whose driver is able to delegate tasks to be done in parallel; this difference in scheduling explains why the trend for the offset operation is linear [25]. For the access map operation, however, the trend was linear for only the first four users. The explanation for this lower number lies in the experimental setup for this procedure: the VM environment has four CPU cores available with which to run CAM operations; because the Python application used a Multiprocessing module to spawn the user processes, the application assigned one user process to each core [23]. Past that, scheduling commenced as multiple users were assigned to the same core, leading to nonlinearly increasing operational times.

CONCLUSIONS

This research presented experimental simulation of simultaneous users running voxel-based CAM software in a DaaS VM environment. An application used to automate the experimental procedure was developed and presented; the resulting time results for two important CAM operations were measured under variable user load; and the resulting data were plotted and analyzed to assess the performance impact that simultaneous user load has on overall operation performance. The modular automated analysis method developed in this work can be used to benchmark performance of other toolpath planning operations as virtualized CAM gains more prominence, and the results of this research can be used to guide designs of new DaaS systems for HPC-accelerated CAM delivery.

Future continuation of this work will focus on improving the DaaS system deployed at Georgia Tech to incorporate more powerful hardware, such as Pascal and Volta GPUs. Additionally, the number of physical CPU cores installed in the system will be increased to improve accessibility analysis performance and to determine if the linear range of operation can be extended. While the experimental protocol described in this work simulated identical operations on the same parts, future protocols will involve different simultaneous operations on different parts. Other essential voxel-based CAM operations will be evaluated with the Python application to determine how their performance is affected by user load. Upon further data collection, models will be developed to enable the prediction of operation performance given a number of users and a part of given complexity.

ACKNOWLEDGMENTS

This work was supported by NSF grants IIP-1631803, CMMI-1646013 and DGE-1650044.

REFERENCES

[1] D. Jang, K. Kim, and J. Jung, “Voxel-Based Virtual Multi-Axis Machining,” *The International Journal of Advanced Manufacturing Technology*, vol. 16, no. 10, pp. 709–713, 2000.

[2] M. M. Hossain, T. M. Tucker, T. R. Kurfess, and R. W. Vuduc, “A GPU-parallel construction of volumetric tree,” *Proceedings of the 5th Workshop on Irregular Applications: Architectures and Algorithms*. ACM, Austin, Texas, pp. 1–4, 2015.

[3] M. Gottschlag, M. Hillenbrand, J. Kehne, J. Stoess, and F.

Bellosa, "LoGV: Low-Overhead GPGPU Virtualization," in *2013 IEEE 10th International Conference on High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing (HPCC_EUC)*, 2013, pp. 1721–1726.

[4] R. Lynn, D. Contis, M. Hossain, N. Huang, T. Tucker, and T. Kurfess, "Extending Access to HPC Manufacturability Feedback Software through Hardware-Accelerated Virtualized Workstations," in *International Symposium on Flexible Automation (ISFA 2016)*, 2016.

[5] R. Lynn, D. Contis, M. M. Hossain, N. Huang, T. M. Tucker, and T. R. Kurfess, "Voxel Model Surface Offsetting for Computer-Aided Manufacturing using Virtualized High-Performance Computing," *Journal Of Manufacturing Systems*, vol. 43, pp. 296–304, 2016.

[6] Tucker Innovations Inc, "SculptPrint - The Subtractive 3D Printing Application." www.sculptprint3d.com.

[7] M. M. Hossain, T. M. Tucker, T. R. Kurfess, and R. W. Vuduc, "Hybrid Dynamic Trees for Extreme-Resolution 3D Sparse Data Modeling," in *2016 IEEE 30th International Parallel and Distributed Processing Symposium, IPDPS 2016*, 2016, pp. 132–141.

[8] D. Konobrytskyi, M. M. Hossain, T. M. Tucker, J. A. Tarbutton, and T. R. Kurfess, "5-Axis Tool Path Planning Based On Highly Parallel Discrete Volumetric Geometry Representation: Part I Contact Point Generation," *Computer-Aided Design and Applications*, pp. 1–14, 2017.

[9] J. A. Tarbutton, T. R. Kurfess, T. Tucker, and D. Konobrytskyi, "Gouge-free Voxel-Based Machining for Parallel Processors," *The International Journal of Advanced Manufacturing Technology*, vol. 69, no. 9, pp. 1941–1953, 2013.

[10] R. Lynn, M. Dinar, N. Huang, J. Yu, J. Collins, C. Greer, T. Tucker, and T. Kurfess, "Direct Digital Subtractive Manufacturing of Functional Assemblies Using Voxel-Based Models (in Press)," *ASME Journal of Manufacturing Science and Engineering*, 2017.

[11] D. Konobrytskyi, "Automated CNC Tool Path Planning and Machining Simulation on Highly Parallel Computing Architectures," Clemson University, 2013.

[12] N. Wang and K. Tang, "Five-axis tool path generation for a flat-end tool based on iso-conic partitioning," *Computer-Aided Design*, vol. 40, no. 12, pp. 1067–1079, 2008.

[13] M. S. Vinaya, N. Vydyanathan, and M. Gajjar, "An evaluation of CUDA-enabled virtualization solutions," in *2012 2nd IEEE International Conference on Parallel, Distributed and Grid Computing*, 2012, pp. 621–626.

[14] V. Gupta, A. Gavrilovska, K. Schwan, H. Kharche, N. Tolia, V. Talwar, and P. Ranganathan, "GViM: GPU-accelerated virtual machines," *Proceedings of the 3rd ACM Workshop on System-level Virtualization for High Performance Computing*. ACM, Nuremburg, Germany, pp. 17–24, 2009.

[15] S. Xiao, P. Balaji, Q. Zhu, R. Thakur, S. Coghlan, H. Lin, G. Wen, J. Hong, and W.-C. Feng, "VOCL: An Optimized Environment for Transparent Virtualization of Graphics Processing Units."

[16] R. Lynn, K. W. Jablokow, N. Reddy, C. Saldana, T. Tucker, T. W. Simpson, T. Kurfess, and C. Williams, "Using Rapid Manufacturability Analysis Tools to Enhance Design-for-Manufacturing Training in Engineering Education," in *ASME 2016 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference (IDETC/CIE 2016)*, 2016.

[17] R. Lynn, C. Saldana, T. Kurfess, S. N. R. Kantareddy, T. Simpson, K. Jablokow, T. Tucker, S. Tedia, and C. Williams, "Toward Rapid Manufacturability Analysis Tools for Engineering Design Education," *Procedia Manufacturing*, vol. 5, no. 44th SME North American Manufacturing Research Conference (NAMRC 44), pp. 1183–1196, 2016.

[18] R. Lynn, K. Jablokow, C. Saldana, T. Tucker, and T. Kurfess, "Enhancing Undergraduate Understanding of Subtractive Manufacturing through Virtualized Simulation of CNC Machining," in *2017 ASEE Annual Conference and Exposition*, 2017.

[19] J. Kouril and P. Lambertova, "Performance analysis and comparison of virtualization protocols , RDP and PCoIP," in *ICCOMP'10 Proceedings of the 14th WSEAS international conference on Computers: part of the 14th WSEAS CSCC multiconference*, 2010, vol. II, pp. 782–787.

[20] J. Li, Q. Wang, D. Jayasinghe, J. Park, T. Zhu, and C. Pu, "Performance overhead among three hypervisors: An experimental study using hadoop benchmarks," in *Proceedings - 2013 IEEE International Congress on Big Data, BigData 2013*, 2013, pp. 9–16.

[21] "GPU Acceleration for Windows Server OS," *Citrix Product Documentation*, 2017.

[22] "NVIDIA and Citrix: Graphics-Accelerated Virtual Desktops and Applications." NVIDIA Whitepaper, 2014.

[23] "Python Multiprocessing Module," *Python Software Foundation*, 2017. .

[24] A. Gupta, A. Tucker, and S. Urushibara, "The Impact of Operating System Scheduling Policies and Synchronization Methods on the Performance of Parallel Applications," *Proceedings of SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, vol. 19, no. 1, pp. 120–132, 1991.

[25] P. Micikevicius, "Performance Optimization: Programming Guidelines and GPU Architecture Reasons Behind Them," in *NVIDIA GPU Technology Conference (GTC)*, 2013.

