The Potential of GPU Computing for Design in RotCFD

R. Ganesh Rajagopalan
Professor
Iowa State University
Ames, IA, USA

Jordan R. Thistle
Aerospace Engineer
Sukra Helitek Inc.
Ames, IA, USA

William J. Polzin
Aerospace Engineer
Sukra Helitek Inc.
Ames, IA, USA

ABSTRACT

Increasingly, the sophistication of rotorcraft flow simulation has become more complex, involving aerodynamically interacting environments such as a helicopter landing on a ship deck. These problems involve solving the Navier-Stokes equations on various grid types (structured, unstructured, or hybrid). As the complexity increases, the range of length and time scales may vary significantly within the problem domain, making simulations more demanding in terms of computational resources. To make RotCFD an affordable tool at the early stages of the design cycle, RotCFD's paradigm is to provide engineering solutions to complex problems in one computational environment. The key to obtaining an engineering solution is to employ one or more of the following techniques: (a) higher order schemes, (b) complex algorithms that exploit the physics intelligently, (c) error reduction techniques such as multigrid, (d) adaptation automation, and (e) parallel processing. This paper focuses on the development of parallel processing in RotCFD using Graphics Processing Units (GPUs) and the associated technical details, which show considerable promise for computational efficiency and design.

INTRODUCTION

RotCFD is an Integrated Design Environment (IDE) centered around a user-friendly Graphical User Interface (GUI) that facilitates engineering design using Computational Fluid Dynamics (CFD). CFD plays a significant and crucial role in the design of modern rotorcrafts. The complexity and sophistication of rotorcraft flow simulation has continually increased, requiring the simulation of environments such as a helicopter landing on a ship, formation flight of multiple vehicles, and brownout conditions due to landing on loose or sandy soil.

Making affordable engineering solutions to complex problems requires reduced computing resources, including computation time. Several techniques exist to reduce computation time, which can be broadly classified as: (a) higher order schemes, (b) complex algorithms that exploit the physics intelligently, (c) error reduction techniques such as multigrid, (d) adaptation and automation, and (e) parallel processing. These techniques are briefly introduced in the following sections, and the main focus of this paper is on the technique of parallel processing. Specifically, this paper focuses on the development of parallel processing in RotCFD using Graphics Processing Units (GPUs) and the associated technical details.

Higher Order Schemes

Finite volume simulations require the domain to be discretized as a set of contiguous smaller regions, called cells, that surround nodes about which the conservation equations are solved. The flow variables, such as velocity and pressure, are known at the nodes. However, their values also need to be known at the individual cell boundaries for flux computations, which are part of the solution process. The flux computation at the cell boundaries involve the nodes adjacent to the cell boundary, and the formulation can be mathematically thought of as interpolation stencils. Involving more nodes in the stencil may improve the order of accuracy at the cost of increased computation. This technique in general increases the computation per node, but decreases the overall number of nodes necessary to retain solution accuracy. This is a purely mathematical approach and is often used in CFD.

Intelligent Complex Algorithms

Rotor flows often involve incompressible, compressible, and viscous regions in the solution domain. The algebraic equations that result from the discretization of the conservation equations are non-linear and coupled. However, discrete equations may not be available for all the variables, such as in the case of incompressible flows, where pressure is available in the momentum equations but not in the continuity equation. The effect of pressure on velocity is of primary importance in incompressible flows. The continuity equation implicitly dictates the pressure field, yet pressure is not a variable of the mass conservation equation. This necessitates the manipulation of the algebraic equations into complex algorithms for obtaining a solution to the conservation equations. The SIMPLE family of algorithms (Ref. 1) has popularized the pressure-based schemes for incompressible flows. The pressure-velocity coupling is resolved in SIMPLE and its variants (SIMPLER, SIMPLEC) by obtaining an approximate

Presented at the AHS Technical Meeting on Aeromechanics Design for Transformative Vertical Flight, San Francisco, California, January 16-18, 2018.

pressure correction field, which is used iteratively to correct the velocity field and/or the pressure field to seek an overall satisfaction of the conservation equations. The approximate nature of the pressure correction equation is often attributed to convergence issues. Alternate algorithms that are more stable while still providing accurate solutions are essential for reducing the cost of design. One such approach for incompressible flows is to use the Runge-Kutta algorithm, which provides reliable, accurate, and fast solutions without the need for approximate equations. Such algorithms need to be extended to compressible and mixed regions for a complete solution of the rotor problem in all regimes.

Error Reduction Techniques

The conservation equations are coupled, non-linear partial differential equations. They are linearized during discretization and made into simultaneous algebraic equations for digital solution. The digital solutions require adjustments to correct for the errors introduced in converting and digitally solving the algebraic equations. The multigrid method, block correction method (Ref. 2), and generalized minimal residual method (GMRES) (Ref. 3) are some of the popular methods that have been successfully used in the past, and they can reduce the total time required to solve a problem using CFD.

Adaptation and Automation

Any CFD solution process starts with an initial set of values on a grid that is not necessarily optimized for the solution, since the solution is unknown *a priori*. Adaptation is a technique by which the grid is regenerated as the solution evolves using the gradient of the solution to adjust the density of the grids within the domain. Adaptation is particularly well suited to unstructured grids and is routinely used in the unstructured solvers in RotCFD. Adaptation can reduce the overall computation time despite the necessity to regenerate grids quite often. This process is successful only if the adaptation and solution processes are automated.

Parallel Processing

Modern computers have many computing resources in the form of Central Processing Units (CPUs) and/or Graphics Processing Units (GPUs). These units are capable of performing computations simultaneously. However, CFD algorithms generally solve either one node or one line of nodes at a time. Therefore, only a few of the processing unit cores are active at any given time, while the rest remain idle. Parallel processing is an approach to resolving this problem of idle cores by making them all compute simultaneously. From the CFD point of view, data dependency is the main obstacle to parallel processing using CPUs, GPUs, or both. In other words, solution algorithms need to be written in such a way that updating the value at a node does not depend on neighboring nodes, which also require simultaneous updating. Parallel processing using GPUs is the central theme of this paper, and the aforementioned ideas will be further elaborated later on.

ROTCFD: A DESIGN TOOL

Rajagopalan et. al. have previously introduced the paradigm of RotCFD (Refs. 4–6), along with validations of the tool's capabilities. A schematic of the RotCFD IDE architecture is shown in Figure 1. A brief introduction to the principal functional units of RotCFD are presented in the following sections.

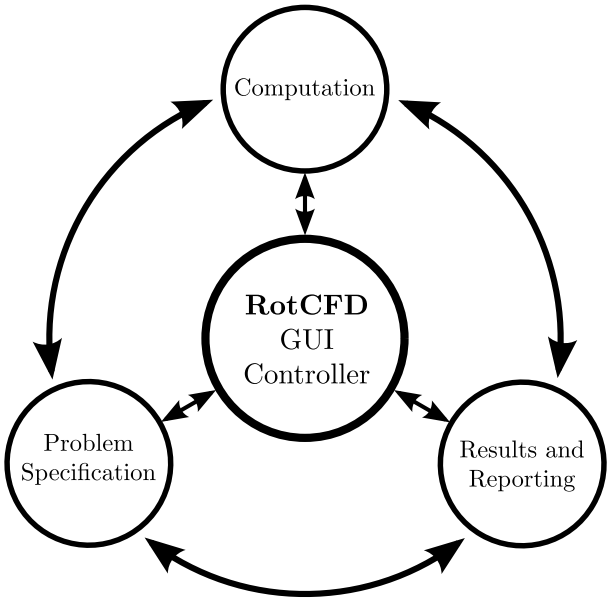


Fig. 1. RotCFD Architecture and Component Interaction

Graphical User Interface (GUI)

The RotCFD GUI allows the user to collect information and assemble the problem. The RotCFD work space is shown in Figure 2.

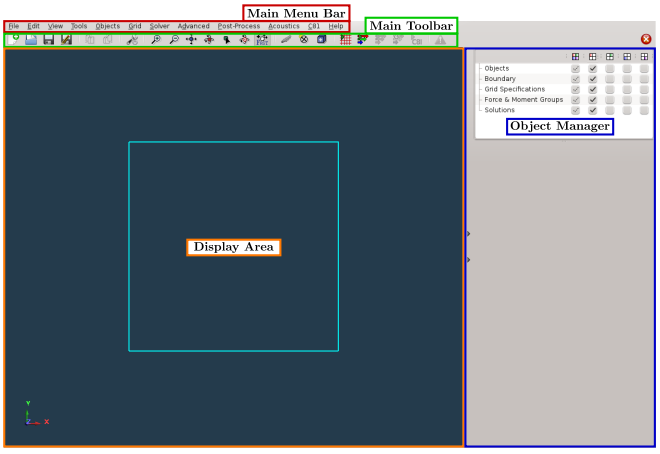


Fig. 2. RotCFD Work Space

Controller

The controller in RotCFD is comparable to the brain of a living system. The controller dictates the sequence of the various

steps that RotCFD takes to obtain a solution. For example, the RotCFD controller is used to call the various grid generators and flow solvers from the GUI. For problems with grid adaptation, the controller periodically calls the grid generator as the flow field develops. For an unsteady inflow the controller is used to reset the boundary conditions of the flow. It can be used to batch a finite number of cases with parametric variation, such as a collective pitch sweep. It can also be potentially used in the future to make function calls for an optimizer, where an unknown arbitrary number of cases need to be run based on the solution as it evolves.

Problem Specification

Problem specification involves all pre-processing, such as describing the geometries and performance specifications for the problem. Geometries include scene objects, which are generally environment objects such as buildings, and configuration geometry, which include the rotors, fuselage, control surfaces, etc. Performance specifications include the set controls and equilibrium conditions. Figure 3 shows how a configuration can be copied and manipulated in RotCFD.

Computation

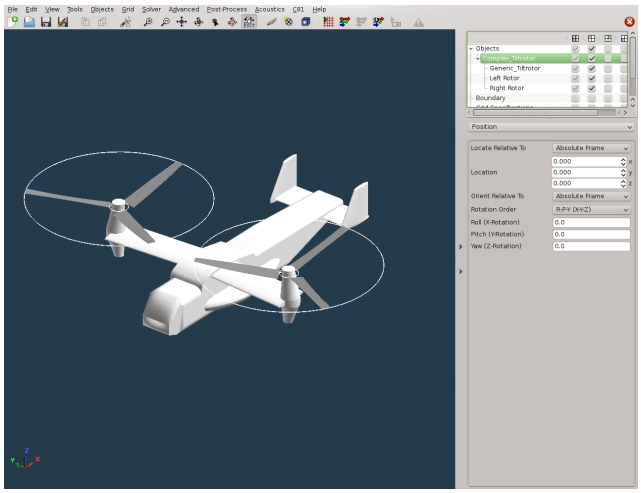
The flow computation involves several steps. A computational grid is generated based on the geometry of the problem. Several grid types are available in RotCFD, each of which is suited to a different type of problem. These grid types are shown in Figures 4-8. Next, the flow is initialized within the domain based on the specified flight condition and boundary conditions. Finally, the flow solver is run to obtain a solution on the grid.

Results and Reporting

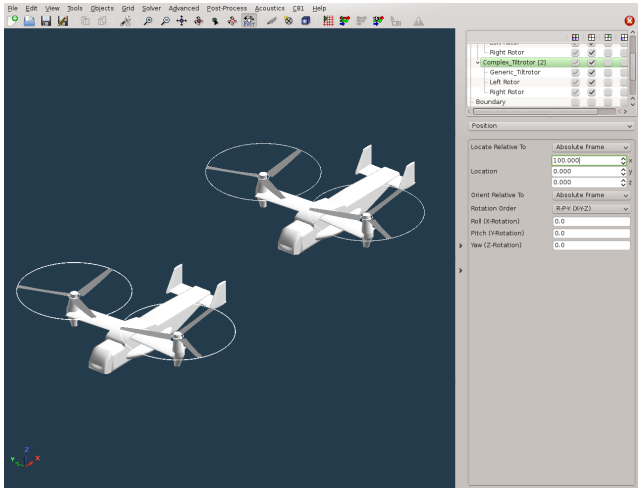
As the flow solver is running, real-time analysis and intermediate results are given. Once the flow simulation is complete, a report is generated and final solutions are available to analyze in the GUI.

PARALLELIZATION

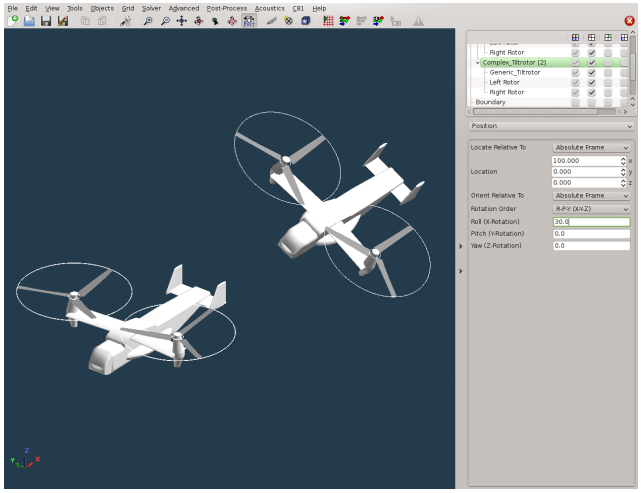
Parallelization involves modifying a serial solver to suit the hardware being used. The current version of RotCFD uses the hardware generally available in the workstation class of computer systems in the commodity market. Therefore, RotCFD uses OpenMP to utilize multiple CPU cores and OpenCL to use the cores available in the system's GPU. In order to provide the maximum performance on all desktop platforms, RotCFD allows the user the option to run in parallel on the CPU or GPU. If the machine contains multiple CPU cores, then CPU-based OpenMP parallelization can be used. If the machine contains a compatible graphics card, then GPU-based OpenCL (Ref. 7) parallelization can be used. By default, RotCFD uses the fastest available run option. In the following sections important aspects of parallel algorithms in RotCFD and GPU related hardware characteristics are discussed.



(a) Single tiltrotor configuration



(b) Copied tiltrotor configuration



(c) Rotated tiltrotor configuration

Fig. 3. Tiltrotor Configuration Manipulation

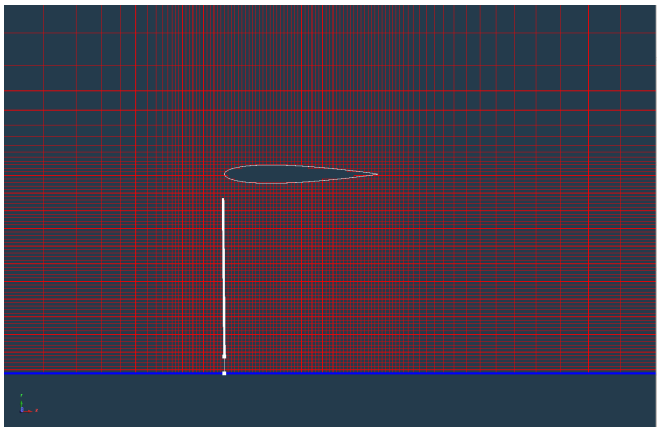


Fig. 4. RotAXC Grid

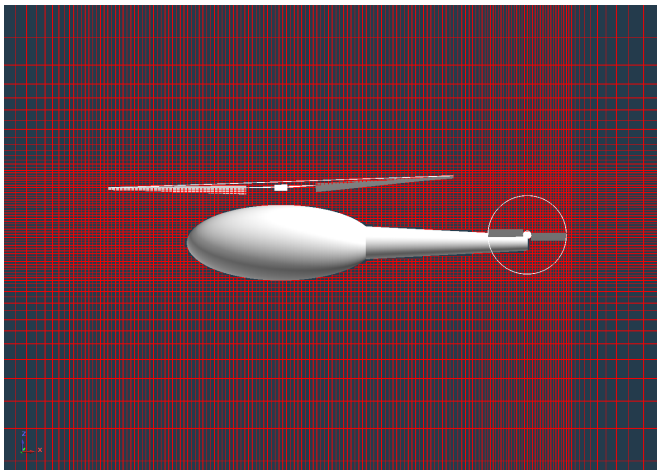


Fig. 5. Rot3DC Grid

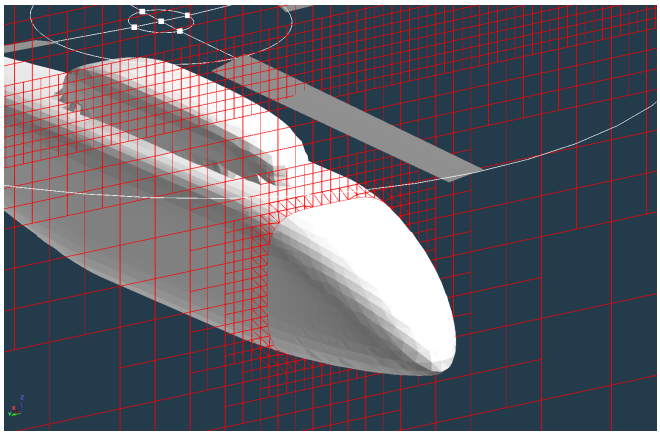


Fig. 6. RotUNS Grid (front)

Parallel Algorithms

One of the important characteristics of the code modification for parallelization is to eliminate data dependency, which requires special algorithm modifications to the flow solver. The parallel algorithms used in RotCFD are discussed in the following paragraphs.

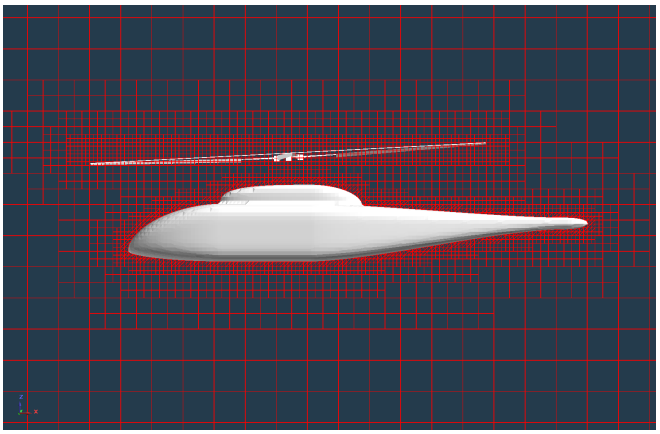


Fig. 7. RotUNS Grid (side)

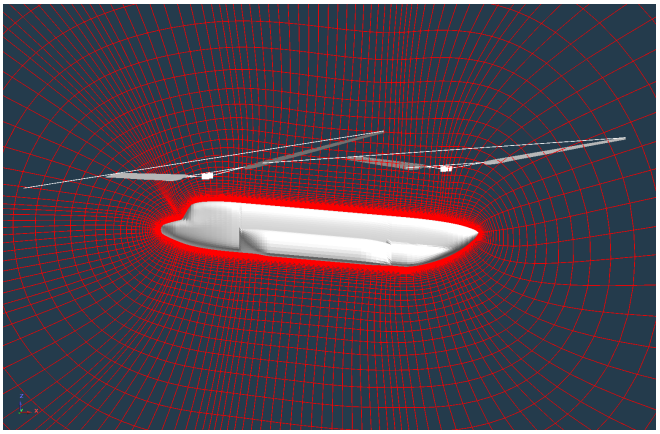


Fig. 8. RotVIS Grid

Greedy Coloring Algorithm The Greedy coloring algorithm (Ref. 8) was implemented in RotCFD for both cells and faces. The coloring algorithm uses various cell and face queues to generate the coloring. It then automatically chooses the coloring layout that should produce the best solver performance. This allows the solver routines to operate in parallel for both OpenMP and OpenCL. Once colors are assigned to all the cells and faces, the cells/faces within a given color can be computed independently from each other in parallel. A sample coloring is shown in Figure 9.

Solver Algorithms Some algorithms, such as the Gauss-Seidel and tridiagonal solver, cannot simply be calculated using the coloring approach since they require updated values from neighboring cells. The tridiagonal solver was replaced with a Parallel Cyclic Reduction (PCR) solver (Ref. 9). The PCR algorithm decomposes a tridiagonal matrix into two halves, each of which maintains a tridiagonal form. This process is repeated on the smaller matrices until n sub-matrices are obtained. The reductions require additional computation $O(n \log_2 n)$, but allows for n-parallelism. As a result, a PCR tridiagonal solve can produce computation times proportional to $\log_2 n$.

The Gauss-Seidel solver was reformulated to compute each color and then use the updated color groups to compute the

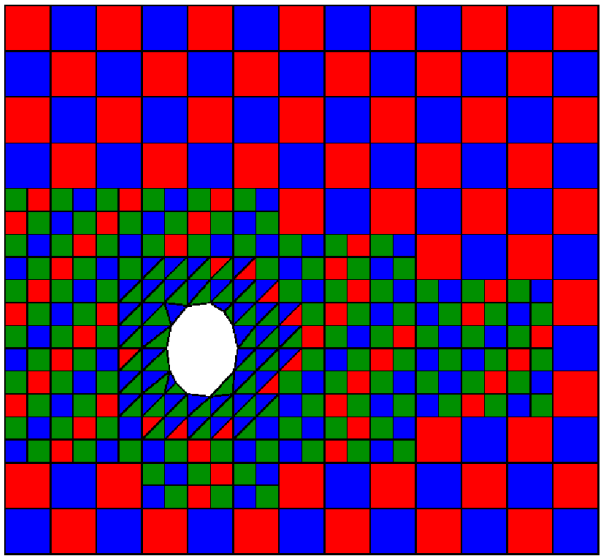


Fig. 9. Greedy coloring scheme

next color instead of individual cell values. The Alternating Color Direction technique is similar to the ADI technique previously used, and it is formulated as follows:

1. Each cell within a color is computed using the neighboring cells.

2. Each cell within the remaining colors is updated.

3. The cells within the original color are updated again using the newly updated values.

4. The colors are swept from 1 to N (where N refers to the maximum number of colors in a domain) and then in the reverse order from N to 1 for each solver iteration, and all cells within a color are computed in parallel.

HARDWARE CONSIDERATIONS

In this section, some important hardware considerations for GPU computing are presented. RotCFD is a tool intended to be used for preliminary design using commodity hardware, for which a sizable number of choices exist. These choices may limit the performance of GPUs, and it may be difficult or expensive to retrofit the machine after the initial purchase if the user discovers that the GPU performance is limited. For example, modern GPU units can require significant amounts of power, and the motherboard and power supply in the system must be capable of providing the GPU's power requirement at peak performance. When multiple GPUs are used, adequate cooling and airflow through the system have to be considered so as not to limit the peak performance of the GPUs. The following sections discuss additional important factors for GPU computing.

GPU-Motherboard Interaction

One important factor in GPU computing is the PCI Express (PCIe) bandwidth and the number of lanes of communication available on the motherboard between CPU and GPU. Lanes are parallel data pathways that are used for sending data and commands to and from the GPU. The number of lanes also affects the performance of the graphics card.

A schematic for PCIe lanes is shown in Figure 10. Most newer graphics cards support PCIe v3.0 and have 16 lanes. However, this is only part of the equation. For maximum performance, the motherboard and CPU have to be able to support the latest PCI version and lanes. PCIe v1.0 has a maximum throughput of 250 MB/s per lane, v2.0 is 500MB/s, and v3.0 is 1GB/s. PCIe versions are backwards compatible. As a result, the graphics card, motherboard, and CPU will use the highest available version common to all the components. For example, if the GPU and CPU support PCIe 3.0, but the motherboard supports PCIe 2.0, all devices will run at PCIe 2.0 speed. This has a significant effect on the data transfer speed and kernel queuing speed. If multiple GPUs are being used, it is important that the CPU supports enough total PCIe lanes to make effective use of the GPUs. Intel's IvyBridge based CPUs only support 16 PCIe lanes, allowing you to run one GPU at full speed. The updated Intel IvyBridge-E allows 40 PCIe lanes, allowing two full speed GPUs and one half-speed GPU. The current work primarily used Dell 5810 computers, the motherboard of which is shown in Figure 11, which allows for up to 40 PCIe lanes.

Graphics Card Specifications

Other important factors in GPU computing are the graphics card specifications: available memory, core count, and speed. Graphics card memory size is typically the largest limiting factor in the practical size of cases that can be run. The available GPU memory has been steadily increasing over the past years, allowing for larger cases to be run on the GPU. The latest generation of Nvidia graphics cards suitable for GPU computation have 2-24 GB of memory. By utilizing single precision numbers and only transferring the required data to the GPU, RotCFD is capable of running cases with millions of cells on a typical desktop. This capability will continue to allow larger cases and faster run times as the hardware improves. However, many CFD problems can require grids larger than what can be run on current GPUs. The motherboards of many commodity systems can support around 10 times the memory of the most expensive Nvidia cards. Therefore, problems that require very large grids currently need to be run on the CPU using OpenMP, which is an option available in RotCFD. Future development could help with this issue by utilizing the memory of multiple GPU units.

For most graphics cards, core count and speed can be summarized by the cards' maximum floating-point operations per second (FLOPS). Most computational problems are limited by the speed of calculations. As a result, the available FLOPS

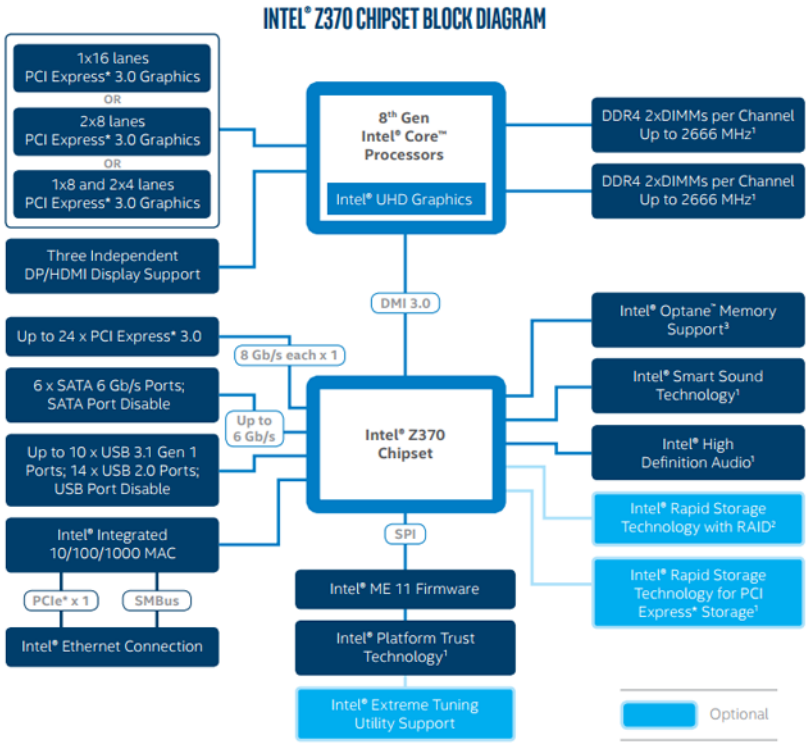
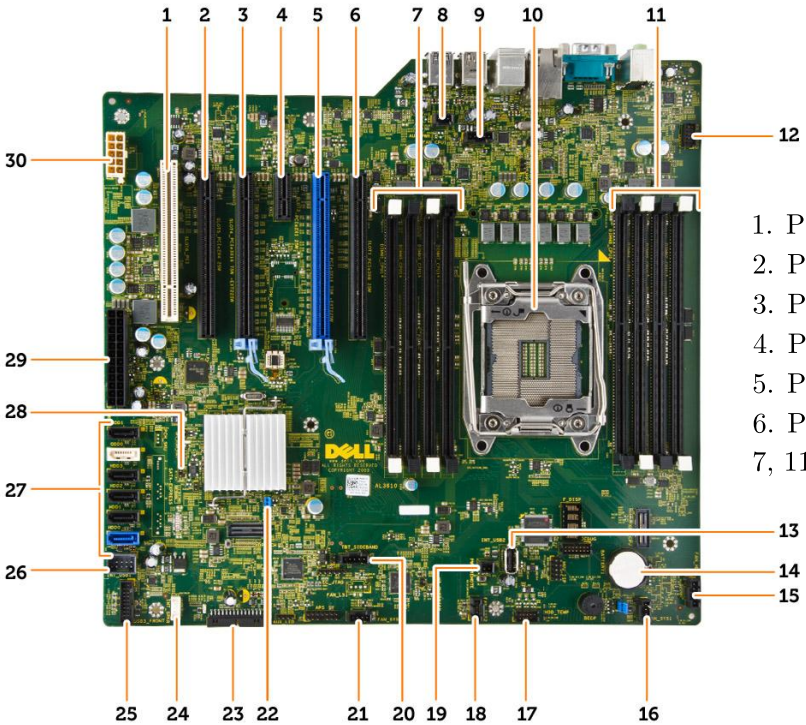


Fig. 10. PCI Lanes



1. PCI slot (slot 6)
2. PCIe x16 slot (PCIe 2.0 wired as x4) (slot 5)
3. PCIe 3.0 x16 slot (slot 4)
4. PCIe 2.0 x1 slot (slot 3)
5. PCIe 3.0 x16 slot (slot 2)
6. PCIe x16 slot (PCIe 3.0 wired as x8) (slot 1)

7, 11. DIMM slots

Fig. 11. Dell 5810 Motherboard

will be a primary factor in the performance. However, computational problems such as CFD can also be memory intensive. In these cases, performance can be bottlenecked by either the memory bandwidth of the GPU or the transfer speeds of the PCIe lanes.

RESULTS

The results presented in this section show the performance on various graphics cards for an isolated rotor case using RotUNS, the RotCFD unstructured flow solver. The test grid of 1,330,000 cells was run for 700 time steps. Unless otherwise specified, the test machine contained a 6 core Intel® Xeon® E5-1654 V4 3.6 GHz processor. All GPU computations were performed in single precision. Each GPU/CPU setup was run twice, and the average time was taken to determine the run time. Tables 1 and 2 contain the specifications for the graphics cards used for the current work that were manufactured by Nvidia, and Tables 3 and 4 contain the specifications for the graphics cards that were manufactured by EVGA.

Table 1. Nvidia GPU Card Specifications

GPU Card	CUDA Cores	Memory	Cost
GTX 1060	1280	6 GB	$249
GTX 1070	1920	8 GB	$379
GTX 1080 Ti	3584	11 GB	$699
GTX Titan X	3584	12 GB	$1200

Table 2. Nvidia GPU Card Specifications (cont.)

GPU Card	Single Precision TFLOPS	Memory Bandwidth (GB/s)
GTX 1060	3.855	192
GTX 1070	5.783	256
GTX 1080 Ti	10.609	484
GTX Titan X	10.157	480

Table 3. EVGA GPU Card Specifications

GPU Card	CUDA Cores	Memory	Cost
GTX 1050 Ti	768	4 GB	$169
GTX 1060	1280	6 GB	$269
GTX 1070	1920	8 GB	$439
GTX 1080	2560	8 GB	$559
GTX 1080 Ti	3584	11 GB	$749

GPU Manufacturer Comparison

The first comparison made was between similar cards made by different manufacturers. In particular, the standard Nvidia card models GTX 1060, GTX 1070, and GTX 1080 Ti were compared against the superclocked versions manufactured by EVGA. Figure 12 shows the run times, and Figure 13 shows the speed-up relative to serial. The superclocked EVGA cards

Table 4. EVGA GPU Card Specifications (cont.)

GPU Card	Single Precision TFLOPS	Memory Bandwidth (GB/s)
GTX 1050 Ti	1.981	112.16
GTX 1060	3.855	192
GTX 1070	5.783	256.3
GTX 1080	8.228	320
GTX 1080 Ti	10.609	484

were slightly faster than the standard Nvidia cards, but only by a small margin of 1-2%.

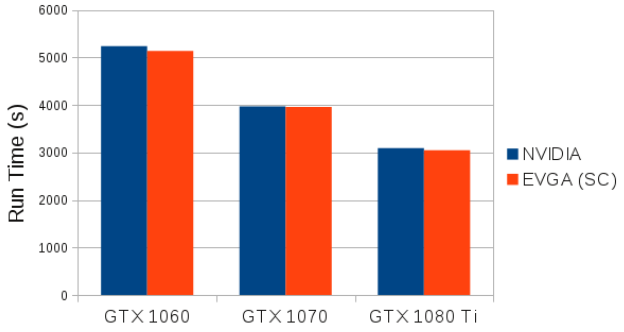


Fig. 12. Run time comparison between NVIDIA and EVGA

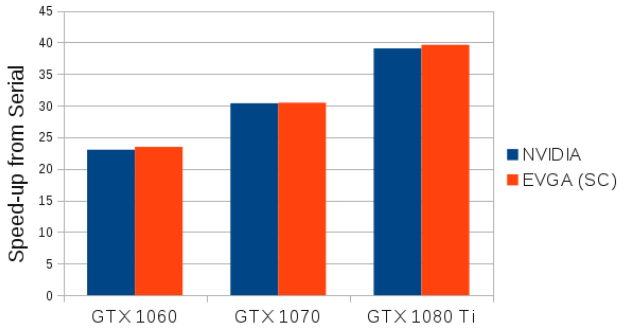


Fig. 13. Speed-up comparison between NVIDIA and EVGA

Effect of CPU in GPU Calculations

To determine the effect of CPU speed on GPU calculations, cases were tested on two other CPU configurations. The first alternative machine contained a 10 core Intel® Xeon® E5-2687W V3 3.1 GHz processor. Cases were run on the EVGA GTX 1070. The average run time on the alternative machine was 3935 seconds. Compared to the base run time of 3957 seconds, there was a speed increase of only 1.006x.

The second alternative machine contained an 18 core Intel® Xeon® E5-2697 V4 2.3 GHz processor. Cases were run on the Nvidia GTX 1070. The average run time on the alternative machine was 3924 seconds. Compared to the base run time

of 3969 seconds, there was a speed increase of only 1.011x. Therefore, the number of cores in the machine's CPU configuration has very little bearing on the final run time if the case is to be run on the GPU.

GPU Card Comparison

Speed-up Comparison This section describes the difference in speed-up between all of the graphics cards used in the current work. Figure 14 shows the run times for each of the graphics card models tested, and Figure 15 shows the speed-up relative to serial.

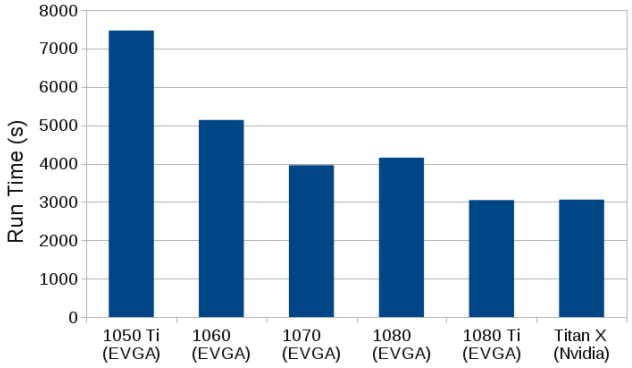


Fig. 14. Run times for various graphics cards

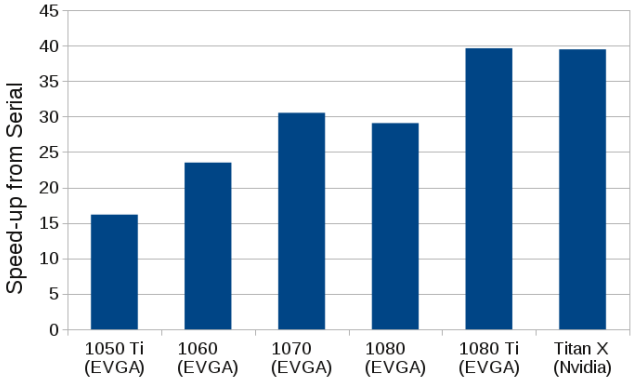


Fig. 15. Speed-up comparison for various graphics cards

Figures 16-19 show the speed-up plotted against various card specifications. Figure 16 shows the speed-up plotted against the number of CUDA cores. Figure 17 shows the speed-up plotted against the single precision GPU speed. Figure 18 shows the speed-up plotted against the GPU memory bandwidth. Figure 19 shows the run time plotted against the cost. Figure 20 shows the speed-up plotted against the cost of each graphics card, and Except for the EVGA 1080, which is an outlier for all of the metrics shown, the general trend for speed-up is logarithmic growth or decay in nature.

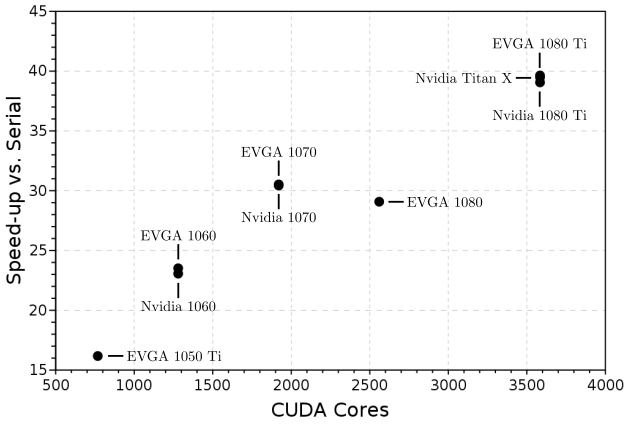


Fig. 16. Speed-up vs. CUDA Cores

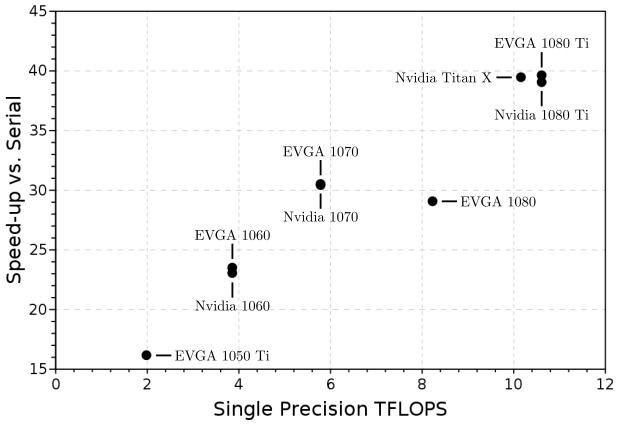


Fig. 17. Speed-up vs. TFLOPS

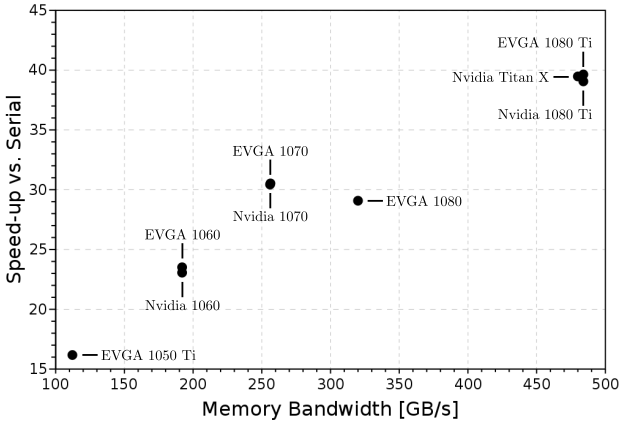


Fig. 18. Speed-up vs. Memory Bandwidth

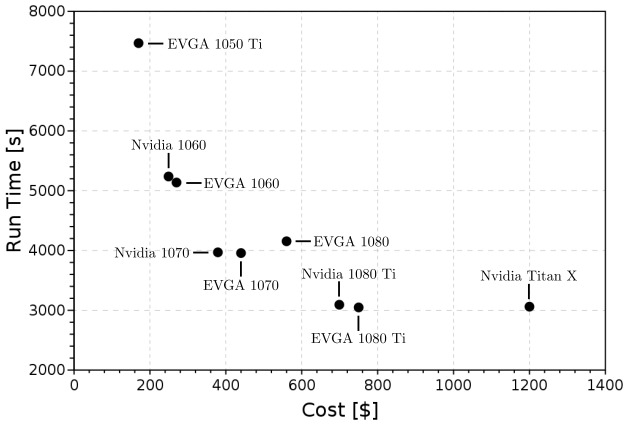


Fig. 19. Run time vs. Cost

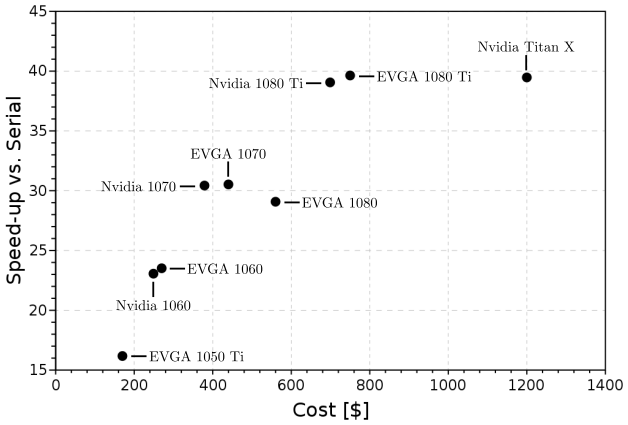


Fig. 20. Speed-up vs. Cost

Maximum Grid Size Comparison The amount of memory available on the graphics card limits the maximum grid size that can be run. This maximum grid size for an isolated rotor case using RotCFD was found for each card. Figure 21 shows the maximum grid size plotted against the cost of each graphics card.

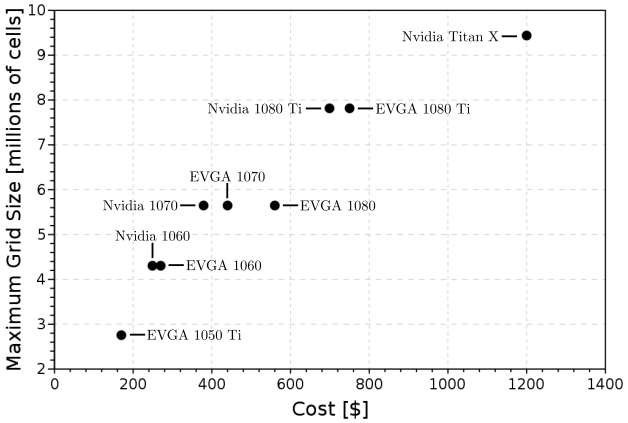


Fig. 21. Maximum grid size vs. Cost

CONCLUSIONS

When using GPU computation for RotCFD, the most important factors for overall performance were the number of CUDA cores, GPU computational speed, and GPU memory bandwidth. The CPU's clock speed and the number of CPU cores available generally showed very little influence on the overall performance. Superclocked GPUs showed only marginal improvements on speed. Currently, the most limiting factor for GPU computations is the available graphics memory, limiting the size of the computation.

Although the CPU does not have much influence on the performance, several hardware factors still need to be considered when running GPU simulations. All hardware should be carefully paired to support the same PCIe bus version. Otherwise, the transfer speed to the GPU will be limited by the lowest supported version. Additionally, the CPU should have enough available power to run the graphics card. The power supply on the motherboard required for the graphics cards used in the current work ranged from 300-600 watts.

In general, GPU computations provide an optimal and affordable alternative to high performance computing if the size of the problem can be accommodated within the GPU's memory. Further improvements could be made by using shared memory and MPI technology, which would allow RotCFD to be designed to use multiple GPUs to facilitate faster design cycles.

AUTHORS CONTACT

R. Ganesh Rajagopalan, rajagopa@iastate.edu; Jordan R. Thistle, jordan.r.thistle@sukra-helitek.com; William J. Polzin, william.j.polzin@sukra-helitek.com

ACKNOWLEDGMENTS

The authors acknowledge the continued financial and technical support of NASA, ARMY, and the US helicopter industry in developing this software for the last three decades.

REFERENCES

[1]Patankar, S. V., *Numerical Heat Transfer and Fluid Flow*, Hemisphere Publishing Corp., New York, NY, 1980.

[2]Settari, A. and Aziz, K., "A Generalization of the Additive Correction Methods for the Iterative Solution of Matrix Equations," *SIAM Journal on Numerical Analysis*, Vol. 10, 1973, pp. 506–521.

[3]Saad, Y. and Schultz, M. H., "GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems," *SIAM Journal on Scientific and Statistical Computing*, Vol. 7, (3), 1986, pp. 856–869.

[4]Rajagopalan, R. G., Baskaran, V., Hollingsworth, A., Lestari, A., Garrick, D., Solis, E., and Hagerty, B., "RotCFD A Tool for Aerodynamic Interference of Rotors: Validation and Capabilities," American Helicopter Society Future Vertical Lift Aircraft Design Conference, San Francisco, CA, January 18–20, 2012.

[5]Novak, L. A., Guntupalli, K., and Rajagopalan, R. G., "RotCFD: Advancements in Rotorcraft Modeling and Simulation," 4^{th} Asian/Australian Rotorcraft Forum, IISc, India, November 16–18, 2015.

[6]Guntupalli, K., Novak, L. A., and Rajagopalan, R. G., "RotCFD: An Integrated Design Environment for Rotorcraft," American Helicopter Society Specialists' Conference on Aeromechanics Design for Vertical Lift, San Fransisco, CA, January 20–22, 2016.

[7]Khronos OpenCL Working Group, *The OpenCL Specification*, 2013.

[8]Cormen, T. H., Leiseron, C. E., Rivest, R. L., and Stein, C., *Introduction to Algorithms*, The MIT Press, Cambridge, Massachusetts, 2009.

[9]Hockney, R., *Parallel Computers*, McGraw-Hill, New York, 1983.