Accelerating Bio-Inspired MAV Computations using GPUs

Amit Amritkar[1], Danesh Tafti[2], Paul Sathre[3], Kaixi Hou[4], Sriram Chivukula[5] and Wu-chun Feng[6]
Virginia Tech, Blacksburg, VA – 24061

In the paper, we discuss the use of general purpose GPUs for simulating the flapping flight of a fruit bat. The highly deformable prescribed wing motion is simulated using an Immersed Boundary Method (IBM). These computations are optimized for the GPU platform using CUDA Fortran to obtain about six times speed up over CPUs. Strong scaling study and code profiling is performed to understand code characteristics to develop a path for future improvements.

Introduction

Micro air vehicles (MAV) are a subset of unmanned aerial vehicles (UAV). MAVs are characterized by their restricted size as well as speed and are intended to be autonomous. They find applications in the area of defense and security as well as in the areas of emergency services such as debris navigation after earthquakes [1].

In recent years, the MAV community has started looking into bio-inspired (bird and insect) flight to achieve exquisite maneuverability in diverse operating conditions [2]. Though flapping flight MAV allows superior control in unpredictable environments, development of control systems for such a device is an ongoing challenge.

Bat flight could be considered an ideal platform to study flapping MAV flight dynamics because the bat flying speeds and weight fit the MAV criterion [3]. The bats fly at low speeds ($Re<10^5$) and the flight dynamics of such flight is an active area of research [4]. Measuring the dynamic flow field in the vicinity of a rapidly moving and deforming surface is very challenging and simulations play an important role in identifying the fundamental force producing mechanisms of flapping flight. Computational frameworks which can study such systems with rapid turnaround are desired [5].

In the current work, we utilize one such computational framework which is capable of performing flapping wing computations with high fidelity. The *in-house* computational fluid dynamics (CFD) program with Immersed Boundary Method (IBM) is used for simulating bat wing fluid dynamics. The objective of the work reported here is to evaluate the use of GPUs to accelerate the time to solution over the CPU version of the code.

Methodology

The GenIDLEST (Generalized Incompressible Direct and Large Eddy Simulation of Turbulence) program used in this study solves the incompressible time-dependent Navier-Stokes and energy equations in a generalized structured body-fitted multiblock framework. It is extensively used in propulsion, biological

[1] Postdoctoral researcher, Department of Mechanical Engineering and Department of Mathematics, amit@vt.edu
[2] Professor, Department of Mechanical Engineering, dtafti@vt.edu, AIAA member
[3] Research Associate, Department of Computer Science, sath6620@vt.edu
[4] Graduate student, Department of Computer Science, kaixihou@vt.edu
[5] Graduate student, Department of Computer Science, sriramc@vt.edu
[6] Professor, Department of Computer Science and Department of Electrical & Computer Engineering, wfeng@vt.edu

flows, and energy related applications which encounter complex multi-physics flows [6]. The program uses a finite volume formulation with fractional step algorithm using semi implicit Adams-Bashforth/Crank-Nicolson or a fully-implicit Crank-Nicolson method for a predictor step. The corrector step solves a pressure Poisson equation to satisfy mass continuity [7]. The computational algorithm used in GenIDLEST is depicted in Figure 1. The program uses state-of-the-art linear solvers based on Krylov subspace methods, CG, BiCGSTAB and GMRES, coupled with cache-aware sub-structured Additive Schwarz preconditioners, which are optimized to take maximum advantage of current day hierarchical memory, and parallel architectures mostly within the context of MPI and OpenMP [8]. The program spans over 200,000 lines and more than 500 subroutines [9].

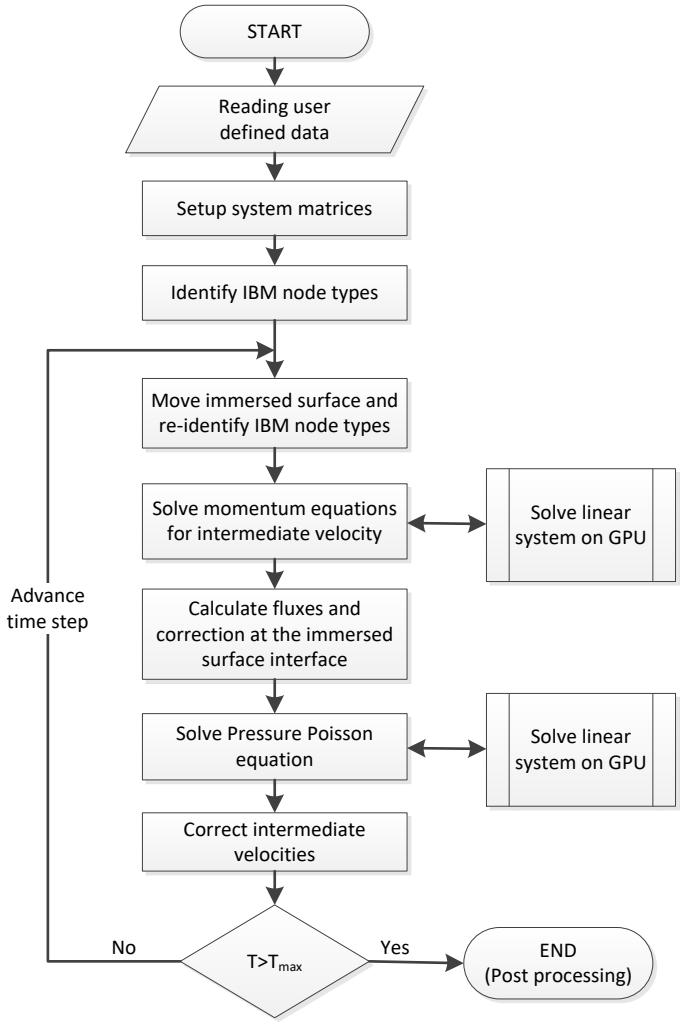

Figure 1 Flow chart of computations in GenIDLEST

GPU parallelization

Only the linear solvers are ported to GPUs mainly because they account for most of the computational time – over 90% on CPUs in the current case. Additionally, the GPU has limited Graphic Dynamic Random Access Memory (GDRAM) available which precludes porting the entire application to the GPU. This is because along with the large number of arrays required for computation, the field arrays would need to be copied to the GPU. This would have an undesired effect in that it would restrict the problem size that could be solved per GPU.

The Central Processing Unit (CPU) parallelism of the program has been optimized over the years. The multilevel parallelism available in GenIDLEST at the mesh block level and at the sub-structured cache block level in the preconditioner, are used for accelerating the linear solvers on GPUs. The program uses mesh blocks for domain decomposition to parallelize the computation across multiple CPU cores which is mapped to multi-GPU computations using MPI. The smaller sub-structured cache blocks are used to efficiently utilize the hierarchical memory architecture available on CPUs. These cache blocks are trimmed to fit the block structure on GPUs.

Immersed Boundary Method

The IBM scheme used in this work is based on the indirect forcing sharp interface method to handle complex 3D geometries [10]. The scheme has been suitably modified to fit the non-staggered framework of GenIDLEST.

The steps involved in IBM are,

1. Identification of node type: This step categorizes nodes as fluid, solid or immersed boundary (IB) nodes. No fluid phase calculations are performed on solid and IB nodes.
2. Boundary treatment for IB nodes: IB nodes are treated separately to enforce immersed surface boundary conditions. Each IB node has a probe associated with it which is used to impose suitable boundary conditions. At every time step, the immersed surface is moved based on the prescribed motion and the above process is repeated.

Since the computations pertaining to IBM take less than 2% of the overall computational time, these calculations are performed on the CPU [11].

Performance optimization

After the initially porting of GenIDLEST to GPUs using CUDA Fortran, additional optimizations were applied to accelerate the computations. This process was done in three major steps [12].

1. The program was profiled to identify performance bottlenecks.
2. Accelerator architecture aware optimizations including coalesced memory access, shared memory usage, double buffering and pointer swapping and removal of redundant synchronizations were applied.
3. Algorithmic modifications were performed to suit the GPUs. This involved optimizations such as elimination of trivial memory accesses by performing in situ computations and dot products plus reductions which are ghost cell aware.

Computational Details

The bat wing simulations are carried out on a CPU+GPU cluster, HokieSpeed, at Virginia Tech. Each node has two six-core Xeon E5645 CPUs with 24 Gbytes memory and two Nvidia M2050 GPUs.

For linear solvers, the BiCGSTAB algorithm with additive Schwarz block pre-conditioner and Jacobi point iterative smoothing is used. In order to utilize the shared memory on GPUs, each of the cache blocks is restricted in size to about 400 grid points.

Bat wing configuration

The bat wing flapping motion is prescribed based on the data from experimental measurements performed at Brown University [13, 14]. The current simulations use flight data which is reconstructed from forward flight data [4, 15] of a bat ascending at a 21 degree angle. The results presented are for the left wing of the bat flapping at a characteristic Strouhal number of 2.86.

The Figure 2 shows the boundary conditions along with the bat wing as the immersed surface. The background fluid mesh consists of about 24 million grid cells and the immersed bat wing surface consist of 9400 surface elements. The computational domain is 6 chord lengths long in up and down stream directions which have inlet and outlet boundary conditions, respectively. The symmetry boundary condition is used at the fixed end of the wing base. Far field boundary condition is applied on the remainder of sides assuming that the effect of the flapping does not reach these boundaries.

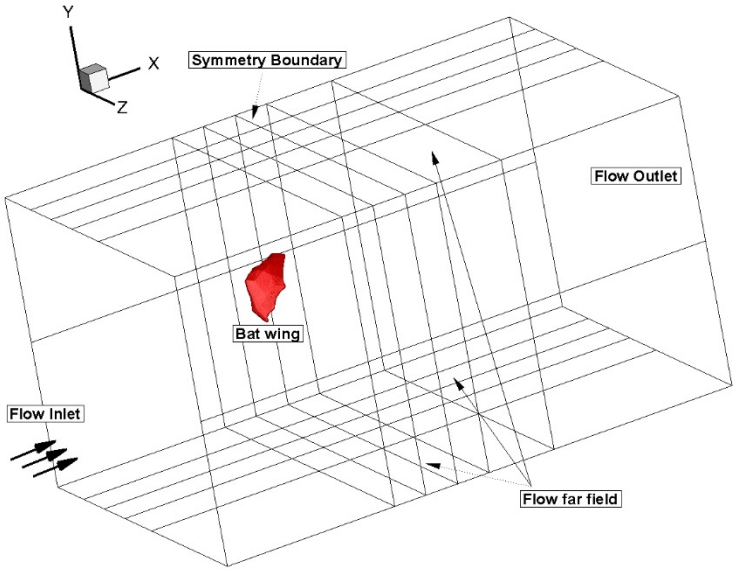

Figure 2 Immersed surface (bat wing) with boundary conditions.

Results and Discussion

Wing Dynamics and Force Production

During the flapping cycle, the wing undergoes complex deformations. The wing conformation is shown in Figure 3 at six different instances during a flapping cycle. During the flapping cycle the wing shape changes continuously, causing the camber and area of the wing to change.

At the beginning of the upstroke, the bat folds the wing close to the body. For most flapping mechanisms, upstroke is required only to get back to the force generating downstroke. Thus, during the upstroke the wing area is reduced as shown in Figure 3 (a-b) to minimize the generated negative lift. The wing surface area is increased as shown in Figure 3 (d-e) during the downstroke and remains almost fully extended all through the downstroke for maximizing lift and thrust force production. This capability to adapt the wing area along with camber and angle of attack gives the bat a high degree of control over the surface pressure distribution. The most prominent force producing mechanism is the attached leading edge vortex (LEV) during the downstroke. The instantaneous coefficients of thrust, lift, and z-force are also shown in Figure 3. Most of the force production occurs during the downstroke when the LEV is attached to the leading edge of the wing (Figure 3 d&e). Another interesting feature is the generation of positive lift force toward the end of the upstroke when the wing is still moving upward. The cycle averaged coefficients of lift and thrust normalized based on the forward flight velocity and planform area are found to be 10 and 7.5, respectively.

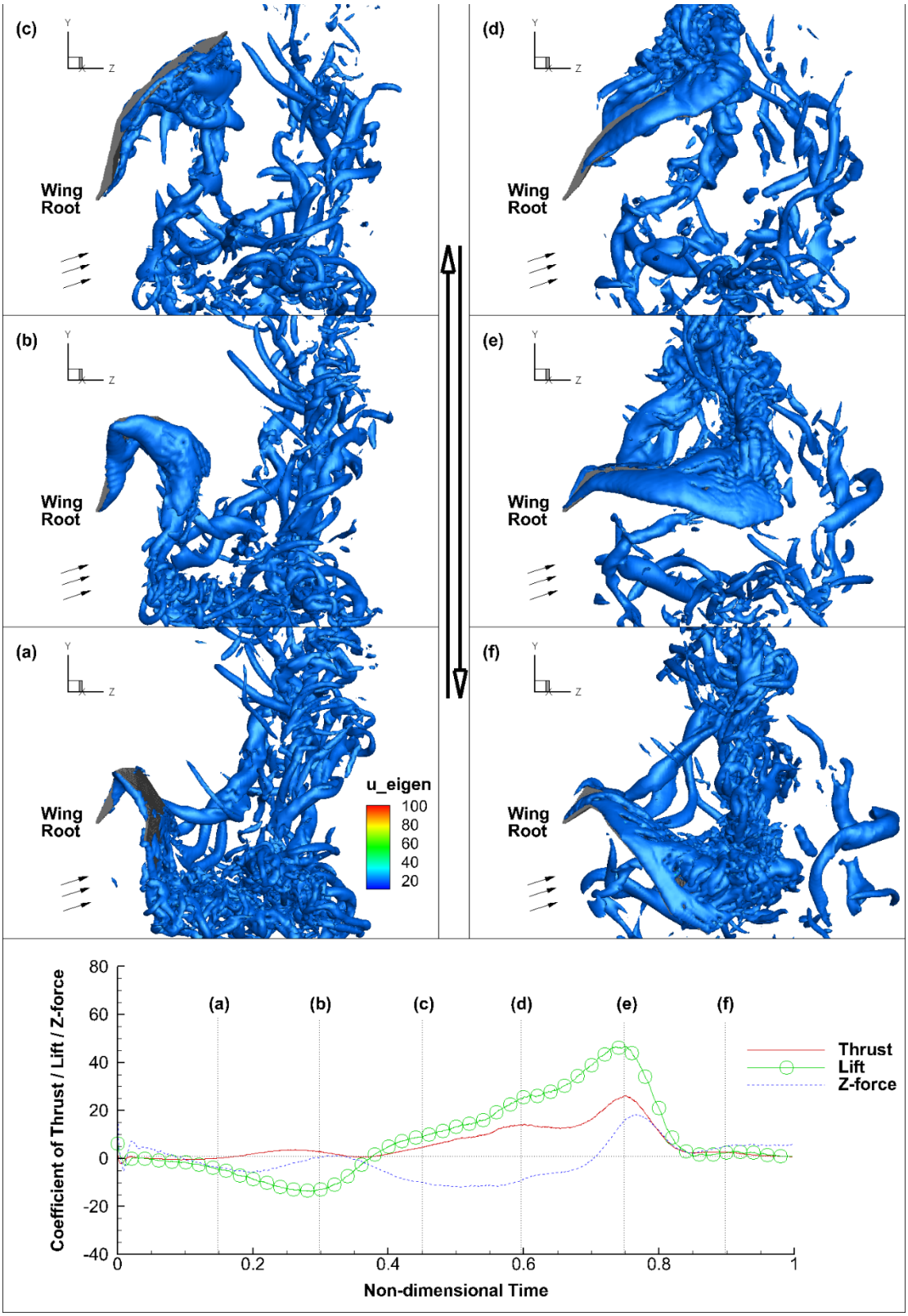

Figure 3 Variation in coherent vortex formation around the bat wing with time. Corresponding lift, drag and z-force locations are mapped on the bottom graph

Parallel Performance

The comparison of thrust and lift coefficients between the CPU and GPU implementation is compared in Figure 4 for correctness of the GPU implementation. Values within 0.1% of each other are obtained.

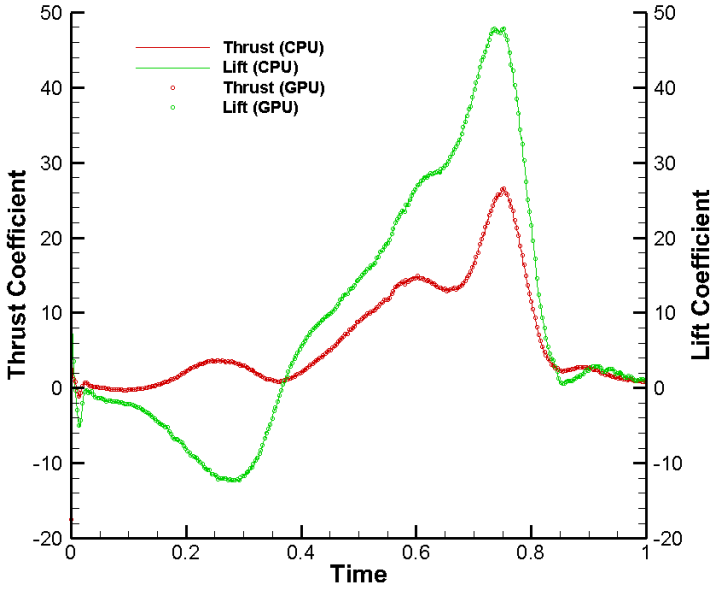

Figure 4 Comparison of coefficients of Thrust and Lift on CPU and GPU

A speed up of approximately six is achieved on 60 GPUs versus 60 CPU cores for computations using double precision. The wall clock time required is shown in Table 1. Thus, for the HokieSpeed architecture, a single CPU socket (6 CPU cores) is equivalent to a single GPU in terms of performance. The computations are run using 30 nodes and 2 CPU cores or GPUs per node which ensured that the memory bandwidth available on the CPU side remained the same for both CPU and GPU runs.

Table 1 : Wall clock time for one cycle of bat wing flapping

CPU time	GPU time	Speedup
580 minutes	100 minutes	5.8 fold

The strong scaling study for the bat case using CPU and GPU versions of the code is shown in Figure 5. As mentioned before, CPU memory bandwidth consistency is maintained in the scaling study. Two different mesh block configurations are tested. One is with constant number of mesh blocks (256) and the other with one mesh block per CPU or GPU, keeping the total number of computational cells at 24 million in all cases. The computational work load per CPU core or GPU reduces as more CPUs or GPUs are used and ideally the time to solution should decrease. In this case, it can clearly be seen that the GPU code doesn't scale well. This is because the workload per GPU becomes significantly small and the high concurrency available in the GPU doesn't get fully utilized. This indicates that there is a minimum workload threshold in order to realize performance benefits from the GPUs. Also with increasing number of GPUs, increase in communication cost becomes a factor in poor performance scaling.

The 256 mesh block case is noticeably slower than the 32 mesh block case on 32 GPUs and slightly slower with 64 GPUs. In the 256 mesh block case there are more inter-block boundaries as compared to the single mesh block per CPU or GPU case. Thus there is need for additional data synchronization at the mesh block

boundaries for 256 mesh block case. This overhead is small on the CPU but for GPUs it is pronounced. In order to further understand the difference in the performance of the two configurations, profiling of the code is performed with 32 GPUs as shown in Figure 6.

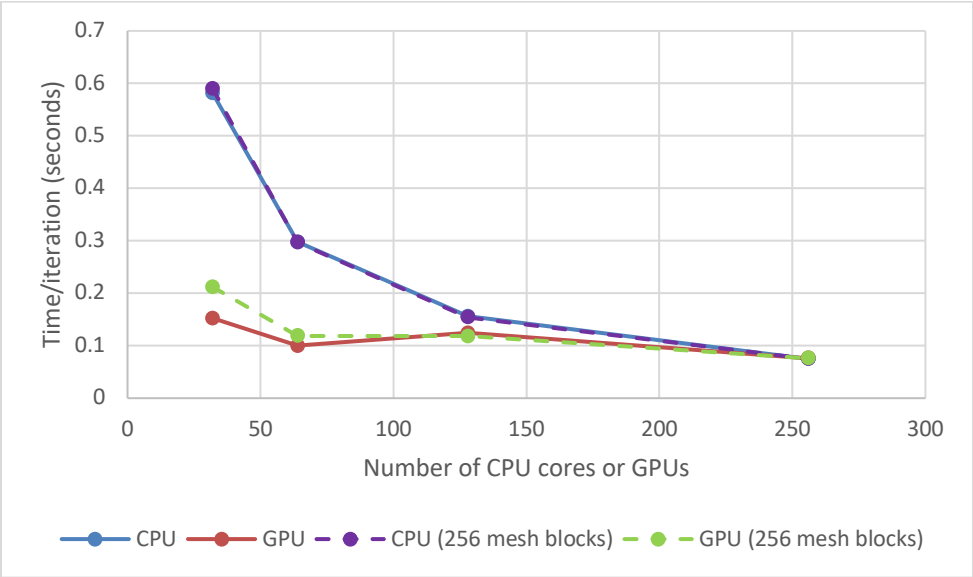

Figure 5 Strong scaling study with one mesh block per processor (CPU or GPU) and 256 mesh blocks (constant)

The time taken in various subroutines is measured using the TAU profiler. The top 21 subroutines which account for about 90% of the total run time are shown. The subroutine names with *gpu* are run on GPUs, names with *mpi* are message passing routines and with *pc* are for preconditioner in the linear solver. The top five time consuming subroutines are all related to data communication except the preconditioner.

For the 256 mesh block case, the additional time is only observed in data communication subroutines used for synchronization across multiple mesh blocks on a GPU. Thus local data copy of ghost cell values between adjoining blocks is observed to be an expensive operation on the GPU as compared to CPU and should be avoided for better scalability.

Table 2 The percentage time of total time spent in communications and GPU/CPU calculations for 32 GPU case

	256 Mesh block	32 Mesh block
Communication	61	40
GPU calculations	22	35
CPU calculations	8	15

The percentage time spent on various aspects of the computation is calculated by combining the time taken by the subroutines shown in Figure 6 and is listed in Table 2. It can be observed that the dominant cost of the whole simulation is the cost of communication. The communication cost can potentially be reduced by using RDMA aware MPI implementation (GPUDirect v3). The CPU calculations are mostly related to pre and post processing of data including I/O costs.

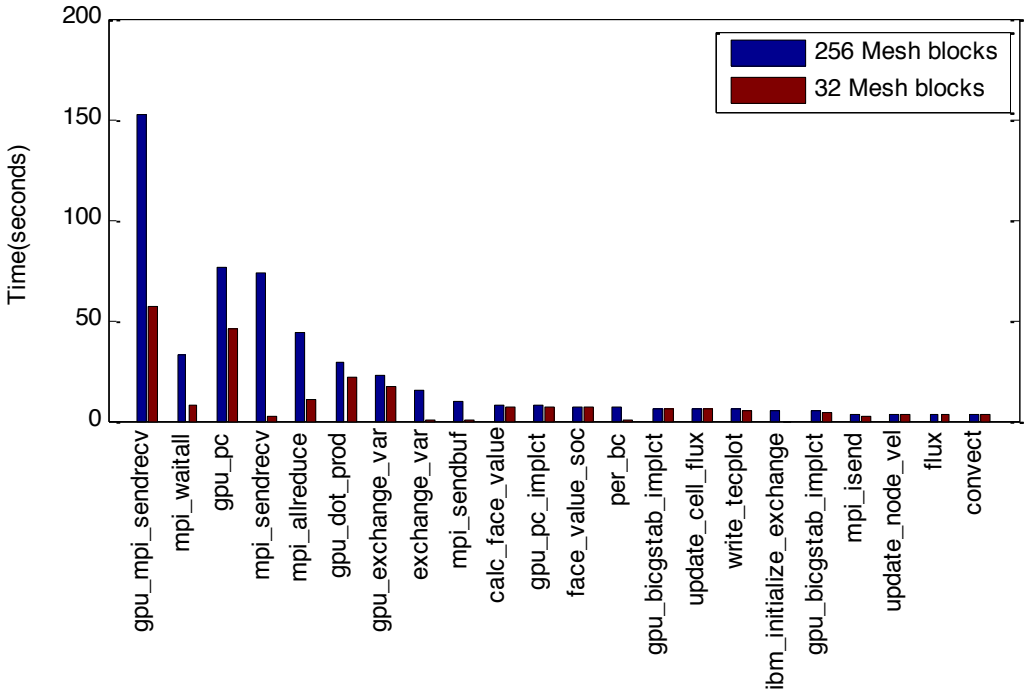

Figure 6 Average time in major subroutines with 32 GPUs measured using TAU profiling

Conclusion and Future work

Bat flight based on prescribed motion was simulated using GPGPUs. An average coefficient of lift of 10 and coefficient of thrust of 7.5 was obtained. An acceleration of about 5.8 times was achieved using multiple GPUs. Strong scaling study and code profiling indicated that MPI communications take majority of the execution time on GPUs.

Currently, further improvement in performance is underway. This includes data marshalling by means of GPU aware MPI implementation. Furthermore, utilizing the idle CPUs on the GPU nodes to perform hybrid GPU+CPU simulation is planned. Using such a framework, it is desired that detailed MAV flight dynamics studies can be performed with a rapid turnaround.

Acknowledgement

This work was funded in part by the Air Force Office of Scientific Research (AFOSR) Computational Mathematics Program via Grant No. FA9550-12-1-0442.

References

1. Pines, D.J. and F. Bohorquez, *Challenges facing future micro-air-vehicle development.* Journal of aircraft, 2006. **43**(2): p. 290-305.
2. Floreano, D., *Flying insects and robots*. 2009: Springer.
3. Bunget, G. and S. Seelecke. *BATMAV: a biologically inspired micro air vehicle for flapping flight: kinematic modeling*. 2008.
4. Viswanath, K., et al., *Straight-Line Climbing Flight Aerodynamics of a Fruit Bat.* Physics of Fluids, 2014. **26**(2).
5. Willis, D.J., et al. *A computational framework for fluid structure interaction in biologically inspired flapping flight*. in *AIAA applied aerodynamics meeting*. 2007.

6. Tafti, D.K. *GenIDLEST - A scalable parallel computational tool for simulating complex turbulent flows*. in *ASME-IMECE*. 2001. New York, NY 10016-5990, United States: American Society of Mechanical Engineers.
7. Tafti, D.K., *Time-accurate techniques for turbulent heat transfer analysis in complex geometries*, in *Computational Fluid Dynamics and Heat Transfer*, R. Amano and B. Sunden, Editors. 2011, WIT PRESS: Southampton, UK. p. 217-264.
8. Amritkar, A., et al., *OpenMP parallelism for fluid and fluid-particulate systems*. Parallel Computing, 2012. **38**(9): p. 501-517.
9. Amritkar, A., S. Deb, and D. Tafti, *Efficient parallel CFD-DEM simulations using OpenMP*. Journal of Computational Physics, 2014. **256**(0): p. 501-519.
10. Nagendra, K. and D.K. Tafti, *Flows Through Reconstructed Porous Media Using Immersed Boundary Methods.* Journal of Fluids Engineering, 2014. **136**(4): p. 040908-040908.
11. Nagendra, K., D. Tafti, and K. Viswanath, *A new approach for conjugate heat transfer problems using immersed boundary method for curvilinear grid based solvers.* Journal of Computational Physics, 2014. **267**: p. 225-246.
12. Sathre, P., et al., *A Methodology for Concurrent Co-Design of Accelerator-Aware Applications*, in *to be submitted*.
13. Riskin, D.K., et al., *Quantifying the complexity of bat wing kinematics.* Journal of Theoretical Biology, 2008. **254**(3): p. 604-615.
14. Hubel, T., et al., *Time-resolved wake structure and kinematics of bat flight.* Experiments in Fluids, 2009. **46**(5): p. 933-943.
15. Viswanath, K., K. Nagendra, and D. Tafti. *Climbing Flight of a Fruit Bat Deconstructed*. in *52nd Aerospace Sciences Meeting*. 2014. Maryland: AIAA.