Using Hierarchical Parallelism to Accelerate the Solution of Many Small Partial Differential Equations

Jacob Merson Department of Mechanical, Aerospace and Nuclear Engineering Rensselaer Polytechnic Institute 110 Eighth St. Troy, NY 12180 Email: mersoj@rpi.edu

Abstract—This paper presents efforts to improve the hierarchical parallelism of a two scale simulation code. Two methods to improve the GPU parallel performance were developed and compared. The first used the NVIDIA Multi-Process Service and the second moved the entire sub-problem loop into a single kernel using Kokkos hierarchical parallelism and a PackedView data structure. Both approaches improved parallel performance with the second method providing the greatest improvements.

1. Introduction

Hierarchical multiscale methods are commonly used to model engineering materials which exhibit complex micromechanical behavior that is not easily captured with standard constitutive modeling [1], [2], [3], [4]. This behavior is often caused by a material having a makeup of discrete constituents such as atoms, molecules, fibers, etc. For a two scale analysis the macroscale, or engineering scale, partial differential equations are often discretized by finite element methods. At each material point in the macroscale, a microscale sub-problem made from discrete components is solved to obtain the material constitutive properties at that point. This information passing scheme is often referred to as the FE2 scheme drawing from the fact that there are is a finite element analysis occurring on two scales [1].

In our multiscale implementation we use parallelism at multiple levels. On the macroscale, we use domain level parallelism to break up our finite element mesh. This is implemented using the SCOREC parallel unstructured meshing infrastructure (PUMI) [5]. Each microscale sub-problem is independent which leads to an "embarrassingly parallel" algorithm. To couple the scales, we use the Adaptive Multiscale Simulation Infrastructure which breaks processors on the target platform into an independent processor set for each scale [6]. In our previous work, the individual microscale sub-problems were not parallelized. Due to our increased understanding of the sub-scale physics in our problem of interest, we have increased the number of degrees of freedom in the microscale sub-problems by two to Mark S. Shephard Scientific Computation Research Center Rensselaer Polytechnic Institute 110 Eighth St. Troy, NY 12180 Email: shephard@rpi.edu

three orders of magnitude [7], [8]. This increase in problem size makes parallelization of the individual sub-problems essential to performing analyses with physical relevance.

Due to the increased computational cost associated with changes in the solution method described in section 2 and the increase in microscale problem size, the solution to the microscale problems became a performance bottleneck. Therefore, we ported our code to use GPU parallelism for the microscale problems. For our problem of interest, the microscale problems often have less than 100,000 degrees of freedom. This is not large enough to saturate the GPU with our current analysis methods which primarily consists of vector operations—similar to BLAS level 1 (See figure 1).

To achieve adequate GPU throughput, multiple microscale problems must be solved on each GPU at a time. This was accomplished using two methods. The first was using NVIDIA Multi-Process Service (MPS) which unintrusively allows multiple processes to launch GPU kernels at a time [9]. Although MPS gave a good speedup, our analysis was still limited by kernel launch overhead and by the limited number of processors on each node to launch GPU kernels from. Additionally, great care must be taken to use MPS in an environment with multiple GPUs per node because improper MPS setup causes a drastic reduction in the weak scalability. The second method for increasing GPU throughput was to pack multiple microscale problems into a single kernel launch using the Kokkos hierarchical parallelism construct [10]. A more thorough discussion of this method is given in section 3. To aid in understanding the selection of our parallelization strategy, a more comprehensive discussion of multiscale modeling techniques is presented below.

2. Multiscale Modeling of Biological Tissues

One particular application of these methods is to model biological tissues which are made of constituent collagen fiber networks. Typically, modeling biological tissues requires large strain analysis because they are soft and physiological strains can often exceed 50%. The FE2 method has been extended to allow for large strains; however, the methods discussed in the literature utilize implicit finite element methods for both analysis scales [2], [11]. Unfortunately, the deformation of fiber networks is highly nonlinear and the network can go through bifurcation points, or may not be isostatic, i.e. the tangent stiffness matrix can be singular during an analysis [12]. As a result, athermal fiber networks, such as collagen networks, are typically modeled with explicit finite element methods [13], [14]. The use of a purely explicit analysis for the sub-scale problems in a multiscale analysis is problematic because kinetic energy is lost in the microscale-to-macroscale coupling. Additionally, inertial effects can change the microscale material properties. To work around these issues, a dynamic relaxation method is used for the microscale problems. Dynamic relaxation works by mapping a static analysis to a damped dynamic explicit one where the system residual is monitored for convergence [15].

The use of the dynamic relaxation method at the microscale greatly improves the global strains which the multiscale method can achieve for fibrous materials, however it imposes a significant computational cost. Our previous studies have shown the multiscale analysis of biological tissues at scale using homogeneous computing technologies with MPI based parallelism [6], [16]. Due to the increased computational cost of dynamic relaxation, physiologically relevant problems are no longer accessible through MPI based parallelism alone. Therefore, the microscale portions of our analysis code have been ported to use GPU accelerators.

The variability in GPU programming environments across hardware vendors poses a significant challenge to the maintainability of a GPU accelerated code which must run on a variety of systems. Therefore, we chose to use the Kokkos C++ library for GPU support. Kokkos is a C++ programming model which is designed to enable performance portability [10], [17]. The Kokkos team has committed to maintaining support for all of the vendors who are providing accelerators for the Department of Energy leadership class computing resources (AMD, Intel, NVIDIA). This support allows for writing a single version of the analysis code that will run across most of the easily accessible GPU accelerators.

3. Parallel Implementation

Figure 1 gives the basic dynamic relaxation algorithm we used for the microscale sub-problems. This algorithm is identical to a two step central difference method found in any finite element text book, with the exception that the convergence criteria is based on a force residual measurement rather than time. Note that each sub-scale problem converges at a different rate which can lead to load imbalance.

In the naive approach, this algorithm was carried out using fused kernels for any subsequent operations with the same loop characteristics. The benefits of kernel fusion have been discussed extensively in the literature both for the case Figure 1. Dynamic relaxation algorithm

- 1: Load mesh and compute edge connectivity
- 2: Compute the mesh connectivity array
- 3: Transfer the connectivity array to the device (GPU)
- 4: Set displacement boundary conditions on fixed nodes (fixed dof vector operation)
- 5: Compute mass matrix (finite element integration)
- 6: GETINTERNALFORCES(u)
- 7: update accelerations (free dof vector operation)
- 8: repeat
 - 9: Compute next time step
- 10: Partial velocity update (free dof vector operation)
- 11: Update displacements (free dof vector operation)
- 12: GETINTERNALFORCES(**u**)
- 13: Compute damping force (free dof vector operation)
- 14: Compute force residual (free dof vector reduction)
- 15: Update Accelerations (free dof vector operation)
- 16: Partial velocity update (free dof vector operation)
- 17: Optionally Check Energy Balance (3 vector reductions)
- 18: Update iteration count
- 19: until Force residual converged

of explicit ODEs, and general GPU computations [18], [19], [20]. The GetInternalForces subroutine accounts for two Kernel launches: the first to zero the internal force vector, and the second to scatter the elemental internal forces to the nodes. The current implementation uses atomic operations to scatter the forces.

In this naive approach, a number of microscale subproblems were assigned to each MPI rank, and were executed serially with respect to each other within each rank. Despite the use of GPU acceleration for the vector operations, this approach had poor performance for the subscale problems with small numbers of degrees of freedom when compared with a CPU-only implementation with serial vector operations. To unintrusively improve this naive approach, NVIDIA MPS was used to allow kernels from multiple MPI ranks to run concurrently. The use of MPS led to significant performance improvements for small DOF problems compared with the naive case. Problem size and number of simultaneous MPI ranks used with MPS can have a drastic effect on performance. All MPS results presented in section 4 use 32 MPI ranks per GPU which gives the best performance in the range of problem sizes discussed here.

Since the loop in algorithm 1 executes millions of times per macroscale simulation step, we observed that this approach had significant kernel launch overhead. To overcome this, we moved the entire loop into a single kernel. This was done using Kokkos hierarchical parallelism which uses teams of threads to enable a 2D map to the hardware. The CUDA reciprocal to this mechanism is launching a 1D grid of 1D blocks. Since our sub-scale problems each have less than 10,000 free degrees of freedom, we found that good performance could be achieved by assigning one thread team to each sub-scale problem. Here, we juxtapose the free degrees of freedom which are those without any Dirichlet constraints, to what we call degrees of freedom which are all potential degrees of freedom. Unlike an implicit FEM method, the constrained degrees of freedom cannot be completely eliminated as they are needed for the internal force computation. Reordering the fixed degrees of freedom to a contiguous block at the end of the displacement array allows most of the update algorithm to only operate on the smaller proportion of free degrees of freedom (figure 1).

The choice of number of threads per team had a strong effect on performance. The ideal number of threads per team is a function of the microscale problem size. All presented results use 512 threads per team, which provided a good compromise for the performance of the smallest and largest systems we tested.

A PackedView data structure which has similar semantics to a Kokkos DualView was used to allow effective access to N-D vector data within each thread team [21]. This data structure uses a row vector and value vector. similar to those from compressed row storage (CRS), to store the data associated with all sub-scale problems on the current MPI rank in a contiguous array in memory. Each sub-scale problem gains access to the correct portion of memory through a Kokkos Subview. In some ways, this structure is similar to a Kokkos View of Views. However, with the current implementation the PackedView can not be resized after initialization. A comprehensive performance comparison between the PackedView data structure, and View of Views has not been performed to date. This differs from the StaticCrsGraph in Kokkos which cannot handle non-integral datatypes, and does not have DualView semantics.

Although moving the analysis loop inside of a single kernel launch was effective for our problems of interest, it can easily succumb to low performance from high register pressure. Significant effort had to be made to reduce the register pressure and ensure that multiple warps could be concurrently scheduled. One mechanism we used to reduce register pressure was to move some of the variables which are carried across loop iterations such as the pseudo-time and the loop iteration count into shared memory. We found that performance gain from the reduction in register pressure outweighed the loss in bandwidth from moving these variables to shared memory. The need to reduce register usage in this single kernel implementation led us to favor a strippeddown version of our algorithm which was specific to the physical system at hand. In other words, flexibility of our code had to be sacrificed to obtain improved performance characteristics.

4. Results

The performance results presented here, are all computed on a single Volta V100 GPU—part of an IBM AC922 node. Each AC922 node contains 2, 20 core IBM power 9 processors clocked at 3.15GHz, 512 GiB of RAM, and 6 Volta V100 GPUs. The code is compiled with version 16.1.0 of IBM's XL compiler for host code, version 10.1 of Cuda,



Figure 2. Speedup of the naive loop based analysis normalized by the number of concurrent sub-problems compared with the loop based single sub-problem. A flat line corresponds to a linear increase in runtime. Each data point is the mean of three analysis runs.



Figure 3. Speedup of the thread team based analysis normalized by the number of concurrent sub-problems compared with the thread team based single sub-problem. Each data point is the mean of three analysis runs.

and version 3.1 of Kokkos. The MPS results make use of Spectrum MPI version 10.3.

Figures 2 and 3 show the runtime of a single subproblem divided by runtime normalized by the number of concurrent sub-problems. This gives a measure of the speedup of a single sub-problem when computed in a concurrent batch. Since we are using the analysis technique's own single sub-problem runtime as a baseline for the speedup, we call this the "self speedup". For the naive loop based case (figure 2), we see the self speedup is very flat which indicates the expected linear increase in runtime. The smallest problem size sees a slight self speedup. When thread team based parallelization is used, a significant self speedup is observed (figure 3). Here we see a initial regime of linear self speedup and a plateau regime for large numbers of concurrent sub-problems. In this initial linear scaling regime, the runtime remains flat since the numerical workload is not large enough to overcome the



Figure 4. Speedup of the team based (solid line) and MPS based (dashed line) analysis over the naive approach. The MPS lines correspond to launching GPU kernels from 32 MPI ranks simultaneously. Each data point is the mean of three analysis runs.

kernel launch latency. Interestingly, the initial self speedup is almost identical for each of the problem sizes we tried. The plateau region show that as the number of degrees of freedom in the problem increase, the self speedup decreases.

Figure 4 shows the speedup of the thread team based and MPS based analysis methods over the naive loop based approach. In this plot, we see an initial linear scaling and a plateau region. We observe that as the problem size increases, the speedup obtained from the team thread based method decreases. This is likely due to a reduction in percentage of the problem which resides in the cache. We also observe that MPS based parallelism does provide some speedup over the naive approach, but it is not as effective as the thread team based approach. Also, for MPS, the speedup plateau does not depend on the problem size. One way to interpret these results is that for maximum efficiency, at least 80 concurrent sub-problems should be run on each GPU. Since the V100 has 80 SMs (streaming multiprocessors), this is consistent with each Kokkos team (CUDA block) occupying a single SM.

The speedups achieved using thread team parallelism tell a compelling story that moving an analysis loop inside of a single heavy weight kernel can be an effective optimization mechanism for problems that need to solve many problems which cannot saturate the GPU on their own. Although MPS seemed like it might be a reasonable solution, it suffered from still incurring a high kernel call latency due to the many kernels which were being called inside of a hot loop. Additionally, the MPS solution was not able to make as effective use of the cache since many different sub-scale problems were competing to be scheduled simultaneously, and each sub-scale problem that was scheduled in an interleaved fashion would cause cache misses.

Acknowledgments

This work was supported in part by the National Institutes of Health (NIH) through Grant No. U01 AT01032606. Also, this material is based upon work supported by the National Science Foundation Graduate Research Fellowship under Grant No. DGE-1744655.

References

- F. Feyel and J.-L. Chaboche, "FE2 multiscale approach for modelling the elastoviscoplastic behaviour of long fibre SiC/Ti composite materials," *Computer Methods in Applied Mechanics and Engineering*, vol. 183, no. 3, pp. 309–330, Mar. 2000. [Online]. Available: http://www.sciencedirect.com/science/article/pii/ S0045782599002248
- [2] R. Smit, W. Brekelmans, and H. Meijer, "Prediction of the mechanical behavior of nonlinear heterogeneous systems by multilevel finite element modeling," *Computer Methods in Applied Mechanics and Engineering*, vol. 155, no. 1-2, pp. 181–192, Mar. 1998. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/ S0045782597001394
- [3] C. Miehe, "Computational micro-to-macro transitions for discretized micro-structures of heterogeneous materials at finite strains based on the minimization of averaged incremental energy q," *Comput. Methods Appl. Mech. Engrg.*, p. 33, 2003.
- [4] P. Kanouté, D. P. Boso, J. L. Chaboche, and B. A. Schrefler, "Multiscale Methods for Composites: A Review," Archives of Computational Methods in Engineering, vol. 16, no. 1, pp. 31–75, Mar. 2009. [Online]. Available: http://link.springer.com/10.1007/ s11831-008-9028-8
- [5] D. A. Ibanez, E. S. Seol, C. W. Smith, and M. S. Shephard, "PUMI: Parallel Unstructured Mesh Infrastructure," ACM Transactions on Mathematical Software, vol. 42, no. 3, pp. 1–28, May 2016. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2935754.2814935
- [6] W. R. Tobin, "The Adaptive Multiscale Simulation Infrastructure," Ph.D. dissertation, Rensselaer Polytechnic Institute, Troy, Ny, Jul. 2018.
- [7] A. S. Shahsavari and R. C. Picu, "Size effect on mechanical behavior of random fiber networks," *International Journal of Solids* and Structures, vol. 50, no. 20, pp. 3332–3338, Oct. 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/ S0020768313002382
- [8] J. Merson and R. Picu, "Size Effects in Random Fiber Networks Controlled by the Use of Generalized Boundary Conditions," *Under Review*, 2020.
- [9] NVIDIA, "Multi-Process Service," NVIDIA Corporation, Tech. Rep., Aug. 2019. [Online]. Available: https://docs.nvidia.com/deploy/pdf/ CUDA_Multi_Process_Service_Overview.pdf
- [10] H. Carter Edwards, C. R. Trott, and D. Sunderland, "Kokkos: Enabling manycore performance portability through polymorphic memory access patterns," *Journal of Parallel and Distributed Computing*, vol. 74, no. 12, pp. 3202–3216, Dec. 2014. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0743731514001257
- [11] C. Miehe, "Numerical computation of algorithmic (consistent) tangent moduli in large-strain computational inelasticity," *Computer Methods in Applied Mechanics and Engineering*, vol. 134, no. 3-4, pp. 223–240, Aug. 1996. [Online]. Available: https: //linkinghub.elsevier.com/retrieve/pii/0045782596010195
- [12] A. J. Licup, A. Sharma, and F. C. MacKintosh, "Elastic regimes of subisostatic athermal fiber networks," *Physical Review E*, vol. 93, no. 1, Jan. 2016. [Online]. Available: https://link.aps.org/doi/10. 1103/PhysRevE.93.012407
- [13] M. R. Islam, G. Tudryn, R. Bucinell, L. Schadler, and R. C. Picu, "Stochastic continuum model for mycelium-based biofoam," *Materials & Design*, vol. 160, pp. 549–556, Dec. 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/ S0264127518307482

- [14] S. Deogekar and R. C. Picu, "On the strength of random fiber networks," *Journal of the Mechanics and Physics of Solids*, vol. 116, pp. 1–16, Jul. 2018. [Online]. Available: http: //www.sciencedirect.com/science/article/pii/S0022509618301285
- [15] P. Underwood, "Dynamic Relaxation," in *Computational Method for Transient Analysis*, 1986, vol. 1, pp. 245–263. [Online]. Available: https://ci.nii.ac.jp/naid/10009833325/
- [16] V. W. L. Chan, W. R. Tobin, S. Zhang, B. A. Winkelstein, V. H. Barocas, M. S. Shephard, and C. R. Picu, "Image-based multi-scale mechanical analysis of strain amplification in neurons embedded in collagen gel," *Computer Methods in Biomechanics and Biomedical Engineering*, pp. 1–16, Nov. 2018. [Online]. Available: https: //www.tandfonline.com/doi/full/10.1080/10255842.2018.1538414
- [17] "Kokkos C++ Performance Portability Programming EcoSystem: The Programming Model: Parallel Execution and Memory Abstraction - kokkos/kokkos," Kokkos. [Online]. Available: https://github.com/ kokkos/kokkos
- [18] M. Korch and T. Werner, "Accelerating explicit ODE methods on GPUs by kernel fusion," *Concurrency and Computation: Practice and Experience*, vol. 30, no. 18, p. e4470, 2018. [Online]. Available: http://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.4470
- [19] G. Wang, Y. Lin, and W. Yi, "Kernel Fusion: An Effective Method for Better Power Efficiency on Multithreaded GPU," in 2010 IEEE/ACM Int'l Conference on Green Computing and Communications Int'l Conference on Cyber, Physical and Social Computing, Dec. 2010, pp. 344–350.
- [20] M. Wahib and N. Maruyama, "Scalable Kernel Fusion for Memory-Bound GPU Applications," in SC '14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, Nov. 2014, pp. 191–202.
- [21] J. Merson, "Kokkos-packed-data." [Online]. Available: https://github. com/jacobmerson/kokkos-packed-data