Exploiting Domain Knowledge to Optimize Mesh Partitioning for Multi-Scale Methods

M. Hasan Jamal[1], Milind Kulkarni[1] and Arun Prakash[2]
[1]School of Electrical and Computer Engineering
[2]Lyles School of Civil Engineering
Purdue University, West Lafayette, IN, USA
Email: {jamal0, milind, aprakas}@purdue.edu

I. Introduction

Multi-scale methods are widely used in complex scientific computing problems [1], [2] which allow problem domains to be decomposed into smaller parts and simulated with different spatial and temporal scales. This way, "the area of interest" is simulated at a finer granularity and higher computation cost while the rest of the system is approximated with a much coarser model. Recursive domain-decomposition [3], [4] is an attractive approach for multi-scale simulation where a large problem is decomposed into group of loosely-coupled subdomains. These subdomains are solved independently and the solutions of individual subdomains are coupled back to get the desired solution. This is illustrated as tree-like algorithmic approach with individual subdomains and shared interfaces are represented as leaf and interior nodes respectively. A recursive TreeSolve function call on root node gives the final solution for a single timestep iteration. Subdomains at smaller timescale will execute multiple times in each iteration.

II. Motivation

A critical issue when using the recursive approach is finding a good decomposition into subdomains. For a given large problem domain, the search space of possible decompositions is exponentially large. The performance of decomposed multi-scale problems is influenced by numerous parameters ranging from the size and quality of the mesh to choice of subdomains and their timescales. Optimizing for so many parameters to produce an optimal decomposition is non-trivial. A substantial amount of domain knowledge is required when choosing the number and size of subdomains during mesh decomposition. Existing partitioners like METIS [5] do not incorporate the necessary domain knowledge for multi-scale problems, resulting in poorly performing decompositions. To circumvent this issue, domain scientists often make intuitive judgments and partition the mesh manually for multi-scale problems. However, for complex problems, deciding the number of timescales, and the timescale ratio between them is challenging, even for the most adept domain scientist. This poster presents a domain specific partitioner that exploits domain knowledge and produces mesh decomposition, automatically, which is optimized for total number subdomains, ratio between timescales, and the number of subdomains assigned to each timescale.

III. Multi-level Domain Specific Partitioner

METIS [5], a state-of-the-art Graph/Mesh partitioner, uses a multi-level bisection algorithm with three phases; 1) Graph Coarsening; 2) Initial Partitioning; and 3) Graph Uncoarsening/Refinement. METIS has limitations when used on multi-scale problems. Multi-scale problems are sensitive to element sizes which dictates timescale value and elements/partitions at smaller timescale do more work. METIS being insensitive to element size produces partitions with varying element sizes, hence forcing them to run at a smaller timescale at higher computational cost. Additionally, the number of partitions required is provided as input parameter and determining the optimal number of partitions is based on guess work. Choosing the wrong number of partitions can have a severe negative effect on performance. Achieving optimized decompositions for a problem domain requires incorporating some sort of domain knowledge at each level of the partitioner and determining the right partitioner phase to apply required domain knowledge is key to performance. Figure 1 shows various phases of our domain specific multi-level partitioner DSMLP.

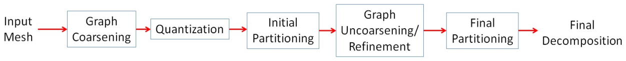


Fig. 1. Partitioning phases in DSMLP

The goals of DSMLP are to i) group elements into partitions based on their size (coarsening); ii) determine the correct timescale for each partition (initial partitioning); iii) move elements between timescales to reduce computational cost without violating accuracy and numerical stability (uncoarsening/Refinement); iv) and determine optimal number of partitions automatically (final partitioning); by applying the required domain knowledge at the right phase. This cannot be achieved by assigning initial weights to METIS. We adopt a simple domain specific cost model to quantify the cost of required operations; (i) subdomain solve; and (ii) interface solve (coupling between subdomains); by analyzing the input problem to determine its properties, namely the interface size and subdomain sizes. Using this information, we can approximate the size of matrices for all operations to estimate the runtime of actual TreeSolve.

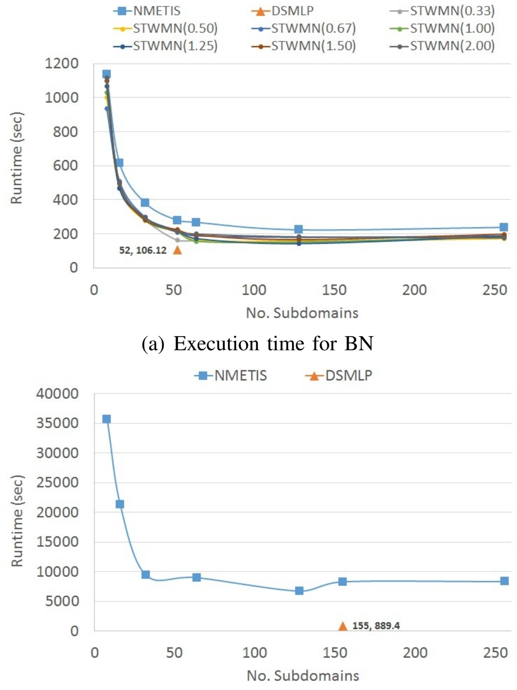
In our approach, the input mesh is represented as an element connectivity graph. Initial graph weight is assigned based on required workload (number of equations to be solved). Each vertex is assigned an approximate theoretical timescale value based on its size. In the coarsening phase, our cost model helps in grouping the elements based on size by merging elements only if the cost is reduced. Additionally, elements are merged only if the difference in their theoretical timescale is smaller than a defined threshold. In the partitioning phase, our cost model helps determine optimized workload distribution between timescales and timescale values (2 quantization levels in this study). To reduce the resulting large interface (high coupling cost) between timescales, elements are moved between timescales in the refinement phase, ensuring accuracy and numerical stability using domain knowledge.

The final partitioning phase, recursively bisects each of the two partitions using our cost model to base its decision to bisect. This phase determines the optimized total number of subdomains and subdomains at each timescale, automatically, and is the final optimized decomposition.

IV. Evaluation

Decompositions produced by DSMLP are evaluated based on runtime of TreeSolve for that decomposition. Two finite element meshes of tetrahedral elements are used as test inputs: 1) a beam mesh with a notch (BN) with 36,552 elements; and 2) a cuboid with two needle and one disk shaped fractures (CF) with 100,720 elements. Decompositions produced from these systems are simulated using scheduler and multi-scale solver from our previous work [6] running on an Intel Xeon E5-4650 system configured with four 8-core processors (total of 32 cores) running at 2.7 GHz. The simulations are run up to 100 microseconds. This is on the order of 10 to 100 TreeSolve iterations, depending on the timescale chosen by the partitioner. For example, if a mesh has a higher timescale of one microsecond, there are 100 TreeSolve iterations. For parallel implementation, we use Cilk [7] runtime system. Figure 2 compares DSMLP against decompositions that a domain scientist is most likely to pick. DSMLP chooses the number of partitions automatically while other decompositions are obtained by providing a pre-selected number of partitions. Our baseline is Naive METIS (NMETIS) with quantization phase after partitioning. STWMN is only used for simpler BN input as for complex problems, it is not feasible for domain scientists to partition the mesh manually.

Decompositions produced by DSMLP shows a performance improvement of 35% and 111% to 656% in execution time for BN and FC respectively, over best performing decomposition produced by a domain scientist and baseline decomposition. Decompositions produced by DSMLP also delivers noticeably better parallel TreeSolve execution times and achieves best speedups in single and multiple threaded execution over other decompositions. DSMLP optimizes for work minimization hence yields better runtime performance. Secondly, despite DSMLP focusing on work minimization sequentially, it yields effective parallelism. The parallel plots and profiling results are presented in the poster.



(a) Execution time for BN

(b) Execution time for FC

Fig. 2. Execution time for solving decompositions produced with various partitioners

A. Mesh decomposition overhead

The mesh partitioning phase for DSMLP, which consists of reading the input mesh, coarsening, quantizing and refining it and reflecting back the decomposition to the original mesh, took 380 and 1170 seconds for the BN and FC meshes respectively. Mesh decomposition cost does not amortize over the 10 to 100 TreeSolve iterations our simulation runs. However, long term simulation of physical models may require running thousands or millions of timesteps, which amortizes mesh decomposition cost.

References

[1] John Michopoulos, Charbel Farhat, and Jacob Fish. Modeling and simulation of multiphysics systems. *Journal of Computing and Information Science in Engineering*, 5:198, 2005.

[2] Jacob Fish. Bridging the scales in nano engineering and science. *Journal of Nanoparticle Research*, 8(5):577–594, 2006.

[3] Jeffrey K. Bennighof and Richard B. Lehoucq. An automated multilevel substructuring method for eigenspace computation in linear elastodynamics. *SIAM Journal on Scientific Computing*, 25:2084–2106, 2004.

[4] Peter Arbenz, Ulrich L. Hetmaniuk, Richard B. Lehoucq, and Raymond S. Tuminaro. A comparison of eigensolvers for large-scale 3D modal analysis using AMG-preconditioned iterative methods. *International Journal for Numerical Methods in Engineering*, 64:204–236, 2005.

[5] George Karypis and Vipin Kumar. Metis - unstructured graph partitioning and sparse matrix ordering system, version 2.0. Technical report, Department of Computer Science, University of Minnesota, 1995.

[6] Chenyang Liu, Muhammad Hasan Jamal, Milind Kulkarni, Arun Prakash, and Vijay Pai. Exploiting domain knowledge to optimize parallel computational mechanics codes. In *Proceedings of the 27th International ACM Conference on International Conference on Supercomputing*, ICS '13, pages 25–36, New York, NY, USA, 2013. ACM.

[7] Robert D. Blumofe, Christopher F. Joerg, Bradley C. Kuszmaul, Charles E. Leiserson, Keith H. Randall, and Yuli Zhou. Cilk: An efficient multithreaded runtime system. In *Proceedings of the 5th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPOPP '95, pages 207–216, 1995.