EINTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN ENGINEERING
Int. J. Numer. Meth. Engng 2017; **112**:58–85
Published online 22 February 2017 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/nme.5506

Exploiting semantics of temporal multi-scale methods to optimize multi-level mesh partitioning

M. Hasan Jamal[1,3], Arun Prakash[2,*,†] and Milind Kulkarni[1]

[1]*School of Electrical and Computer Engineering, Purdue University, West Lafayette, 47907, IN, USA*
[2]*Lyles School of Civil Engineering, Purdue University, West Lafayette, 47907, IN, USA*
[3]*Department of Computer Science, COMSATS Institute of Information Technology, Lahore, Pakistan*

SUMMARY

Multi-scale problems are often solved by decomposing the problem domain into multiple subdomains, solving them independently using different levels of spatial and temporal refinement, and coupling the subdomain solutions back to obtain the global solution. Most commonly, finite elements are used for spatial discretization, and finite difference time stepping is used for time integration. Given a finite element mesh for the global problem domain, the number of possible decompositions into subdomains and the possible choices for associated time steps is exponentially large, and the computational costs associated with different decompositions can vary by orders of magnitude. The problem of finding an optimal decomposition and the associated time discretization that minimizes computational costs while maintaining accuracy is nontrivial. Existing mesh partitioning tools, such as METIS, overlook the constraints posed by multi-scale methods and lead to suboptimal partitions with a high performance penalty. We present a multi-level mesh partitioning approach that exploits domain-specific knowledge of multi-scale methods to produce nearly optimal mesh partitions and associated time steps automatically. Results show that for multi-scale problems, our approach produces decompositions that outperform those produced by state-of-the-art partitioners like METIS and even those that are manually constructed by domain experts. Copyright © 2017 John Wiley & Sons, Ltd.

Received 9 August 2016; Revised 27 December 2016; Accepted 3 January 2017

KEY WORDS: mesh partitioning; domain-specific optimization; multi-scale; load balancing; finite element methods; parallelization

1. INTRODUCTION

Multi-scale methods have seen significant development in the recent years for problems in science and engineering [1–3]. One class of these methods employs spatial domain decomposition [4–6] and multi-time-step integration [7, 8] to decompose a problem domain into smaller subdomains, solve them independently, and couple the subdomain solutions back to obtain the global solution. This approach allows a fine-grained simulation for areas of interest, at higher computational cost, while solving the remaining parts of the problem with a much coarser model to keep the overall cost down. This multi-scale method thus avoids incurring the cost of such fine-grained simulation throughout the problem domain while still maintaining the necessary level of resolution for the simulation. Figure 1 shows such a multi-scale problem of a beam with a notch, where the region around the notch is typically discretized with much smaller finite elements in comparison with the rest of the mesh and solved with correspondingly small time steps to capture the transient physical processes.

In this work, we focus on recursive domain decomposition [9–13] where a large problem domain is decomposed into two subdomains that are then further decomposed into two subdomains each,

*Correspondence to: Arun Prakash, Lyles School of Civil Engineering, Purdue University, West Lafayette, IN 47907, USA.
†Email: aprakas@purdue.edu

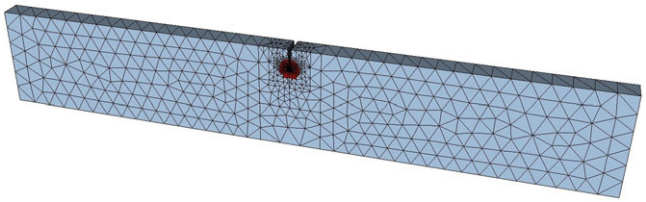


Figure 1. A sample physical system; beam mesh with a notch.

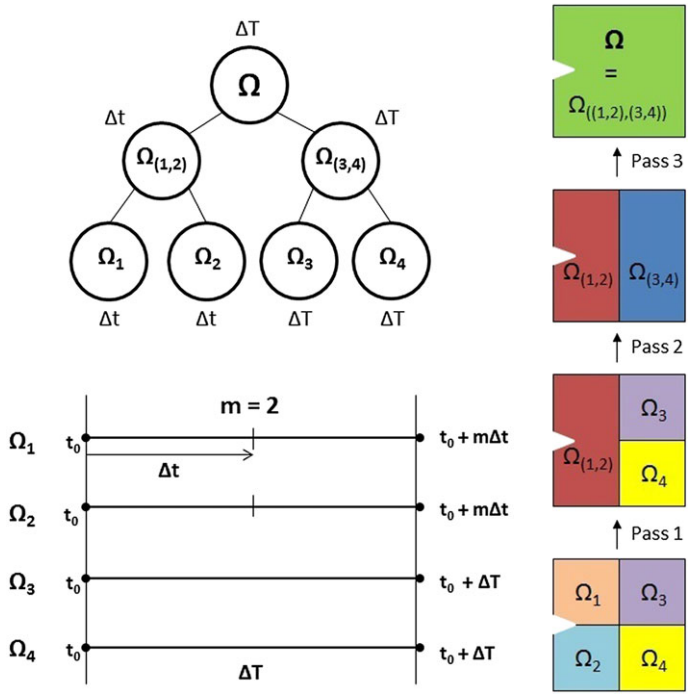


Figure 2. Recursive domain decomposition and coupling for multi-scale problems.

and the process is repeated successively to yield a decomposition that can be represented by a binary tree of loosely coupled subdomains. Figure 2 shows an example of this approach with individual subdomains and shared interfaces represented as leaf and interior nodes, respectively. The subdomains are solved independently with possibly different time steps and coupled together (two at a time) using Lagrange multipliers that enforce continuity of the solution at the interface between these subdomains. As an example, the problem domain shown in Figure 2 is decomposed into four subdomains such that the subdomains Ω_1 and Ω_2, adjoining the notch, are simulated with a smaller time step (Δt), and subdomains Ω_3 and Ω_4 are simulated with a larger time step (ΔT). For each large ΔT subdomain in the system, the subdomains at smaller Δt are solved m times, where $m = \Delta T/\Delta t$ is the integer time-step ratio between ΔT and Δt. After solving for each subdomain, individual subdomain solutions are then coupled together, in a recursive manner, to obtain the final solution. With this recursive approach, once two subdomains are coupled at a particular time step, they can be considered as a single larger subdomain (represented by an interior node in the tree) for the next coupling step higher up in the tree. After the final coupling operation at the root node of the tree, the global solution to the original problem is obtained. Further details of the coupling method are available in [8, 14]. This recursive coupling approach has also been shown to be computationally advantageous over coupling all the subdomains simultaneously [15].

DOI: 10.1002/nme

Despite the computational advantages of the recursive domain decomposition, some of the issues that users face while using this approach are finding an optimal mesh decomposition, selecting time steps for each subdomain and finding the optimal coupling order (tree) for the subdomains. When making the choice for number of subdomains and the coupling order, certain additional constraints on the choice of time steps also come into play. For large and complex problems, the number of finite elements in the mesh and the variation in sizes of these elements can span multiple orders of magnitude. For instance, in certain regions of interest within the problem domain, both the spatial and temporal discretizations are usually much finer in comparison with the rest of the mesh. The choice of time step for a subdomain is usually governed by stability, accuracy, and computational cost criteria, and often, it is closely linked to the size of the smallest element in the subdomain. In this work, for the purposes of partitioning the mesh, we begin by associating each individual element in the global mesh with its own ideal time step. Because the variations in element sizes (and consequently time-step sizes) can be quite large, elements with similar sizes (and similar time-step requirements) can be grouped together to obtain multiple discrete levels of time-step quantizations (Section 2.4). During the process of partitioning the mesh, the time step for a particular subdomain is chosen to be less than or equal to the time step for the smallest element in that subdomain. As an additional implementation issue, deciding the number of quantization levels of time steps and the time-step value at each quantization level is also not trivial. With these multiple constraints in play, assigning elements to different partitions and choosing the time-step quantizations for these partitions to obtain an optimal decomposition is a very challenging problem and especially hard for a user to tackle by hand.

For reasons both numerical and pragmatic, choosing an optimal mesh decomposition is crucial for obtaining good computational performance with multi-scale problems. In general, for a large finite element model, decisions regarding the number and choice of subdomains, their associated time steps and the coupling order of these subdomains may also depend upon the hardware platform on which it is implemented and how it is parallelized. In the current work however, we restrict our focus to minimizing the total computation cost of the recursive domain decomposition approach by identifying optimal decompositions and the associated time steps. By separating the issue of minimizing total computational cost of the method from its implementation, issues of parallelization and load balancing can largely be treated as a separate problem and will not be considered here.

The focus of the paper is on constructing an (almost) optimal partitioning from a given well-designed mesh that is based on physical and numerical considerations of accuracy and stability, rather to construct the mesh itself. We present an automated mesh-partitioning approach that exploits domain knowledge about the coupling method to produce mesh decompositions that optimize the total number of subdomains, allocation of elements to these subdomains, and the time steps assigned to these subdomains. We show that decomposed meshes generated by our approach perform as well as, if not better than, current state-of-the-art partitioners, like METIS [16], and even those that are manually constructed by domain scientists. Related work on mesh partitioning can be found in [17] and the references within.

2. BACKGROUND

Time-dependent problems in most computational science and engineering applications are governed by laws of physics that are expressed as partial differential equations. These partial differential equations are most commonly solved numerically by discretizing them in space and time using methods such as finite elements, finite volumes, and finite differences. This allows one to solve for the state $\boldsymbol{u}_{n+1} \approx \boldsymbol{u}(t_{n+1})$ of the system at some time t_{n+1} from a known state $\boldsymbol{u}_n \approx \boldsymbol{u}(t_n)$ by advancing through a small time step Δt where $t_{n+1} = t_n + \Delta t$. This process of time stepping can be represented as an algebraic system of equations:

$$\boldsymbol{M}\,\boldsymbol{u}_{n+1} = \boldsymbol{p}_{n+1} - \boldsymbol{N}\,\boldsymbol{u}_n, \tag{1}$$

DOI: 10.1002/nme

where $\boldsymbol{M}$ and $\boldsymbol{N}$ represent the discretized operators governing the behavior of the system and $\boldsymbol{p}$ represents a forcing vector that drives the dynamical system. This process can be repeated successively to advance the solution further in time. However, as pointed out earlier, solving Equation (1) as a whole is computationally inefficient for large multi-scale problems, and it is more advantageous to use recursive domain decomposition.

2.1. Recursive domain decomposition with uniform time discretization

Given a partitioned finite element mesh, a hierarchy of subdomains can be built by combining two subdomains at a time until the original undecomposed mesh is recreated as shown in the tree structure, in Figure 2. The original undecomposed structure Ω is represented by the root node, and the leaf nodes represent subdomains that are not further subdivided.

In general, the problem of solving any interior node $\Omega_{(A,B)}$ in the tree is substituted with two coupled subproblems for Ω_A and Ω_B replacing Equation (1) with

$$\left[\begin{array}{cc|c} \boldsymbol{M}^A & & \boldsymbol{C}^A \\ & \boldsymbol{M}^B & \boldsymbol{C}^B \\ \hline \boldsymbol{B}^A & \boldsymbol{B}^B & \boldsymbol{0} \end{array}\right] \left[\begin{array}{c} \boldsymbol{u}^A_{n+1} \\ \boldsymbol{u}^B_{n+1} \\ \hline \boldsymbol{\lambda}^{(A,B)}_{n+1} \end{array}\right] = \left[\begin{array}{c} \boldsymbol{p}^A_{n+1} - \boldsymbol{N}^A \boldsymbol{u}^A_n \\ \boldsymbol{p}^B_{n+1} - \boldsymbol{N}^B \boldsymbol{u}^B_n \\ \hline \boldsymbol{0} \end{array}\right], \tag{2}$$

where $\boldsymbol{C}$ and $\boldsymbol{B}$ represent connectivity matrices between subdomains Ω_A and Ω_B and $\boldsymbol{\lambda}^{(A,B)}_{n+1}$ represents the Lagrange multiplier that enforces continuity of the solution across the interface between Ω_A and Ω_B. Starting at the root node, the entire system can be solved recursively in a decomposed manner, using the following sequence of operations at each time step:

(i) Solve the smaller uncoupled subdomain problems for $s = \{A,B\}$.

$$\boldsymbol{M}^s \ \boldsymbol{v}^s_{n+1} = \boldsymbol{p}^s_{n+1} - \boldsymbol{N}^s \boldsymbol{u}^s_n \tag{3}$$

(ii) Solve for the interface variable $\boldsymbol{\lambda}^{(A,B)}_{n+1}$ from

$$\boldsymbol{H}^{(A,B)} \ \boldsymbol{\lambda}^{(A,B)}_{n+1} = \boldsymbol{f}^{(A,B)}_{n+1}, \tag{4}$$

where the matrix $\boldsymbol{H}^{(A,B)} = \boldsymbol{B}^A \boldsymbol{Y}^A + \boldsymbol{B}^B \boldsymbol{Y}^B$ represents the interface operator required to enforce continuity of the solutions between subdomains Ω_A and Ω_B and the vector $\boldsymbol{f}^{(A,B)}_{n+1}$ is obtained from the uncoupled solution in step (i) earlier as $\boldsymbol{f}^{(A,B)}_{n+1} = \boldsymbol{C}^A \boldsymbol{v}^A_{n+1} + \boldsymbol{C}^B \boldsymbol{v}^B_{n+1}$.

(iii) Update the individual solutions for Ω_A and Ω_B.

$$\boldsymbol{u}^s_{n+1} = \boldsymbol{v}^s_{n+1} - \boldsymbol{Y}^s \boldsymbol{\lambda}^{(A,B)}_{n+1}. \tag{5}$$

We refer to the earlier solution procedure as $\mathsf{TreeSolve}(\Omega_{(A,B)})$. Calling $\mathsf{TreeSolve}(\Omega)$ at the root node and recursively solving the entire tree advances the state of the original problem from $\boldsymbol{u}_n$ to $\boldsymbol{u}_{n+1}$. Further details of the coupling method are available in [8, 14].

A key semantic property of this recursive domain decomposition method is that all nonoverlapping decompositions of a given mesh into subdomains are admissible and that it also allows for these subdomains to be coupled back in any order to yield a solution to original global problem [15]. These properties result in a very large number of possible tree representations of the coupled solution procedure for a given set of subdomains resulting from a specific decomposition. For instance, a problem decomposed into three subdomains can be coupled in three different ways and correspond to 3 distinct trees, 4 subdomains result in 15 distinct trees, 8 subdomains yield 135,135 trees, 16

DOI: 10.1002/nme

subdomains have about 6.19×10^{15} trees, and so on. Thus, in the extreme case, where one may choose to represent each element within a mesh as an individual subdomain, the number of possible trees would be enormously large. Picking a particular tree that results in good performance out of this huge space of possible trees is not trivial.

2.2. Recursive domain decomposition with multiple timescales

In Section 2.1, we discussed a method that utilizes the same time step for discretizing all the different subdomains in a tree. For large multi-scale problems, certain regions that contain physically interesting features (e.g., cracks in a beam) need to be simulated at a finer granularity compared with the rest of the model. Solving these types of problems using a single, very small, time step (i.e., governed by the fastest dynamics around the crack) is impractical and computationally unnecessary because we want this level of resolution only in some parts of the problem domain. To overcome this issue, such problems are solved using multiple time steps, smaller (finer) time steps for the subdomains in and around the region of interest, and larger (coarser) time steps for the remaining subdomains.

The approach described in Section 2.1 is most beneficial when different subdomains are solved at different time steps that can be modified as needed to account for multiple timescales in the problem. In step (i) of the TreeSolve procedure, if the daughter nodes Ω_A and Ω_B are assigned different time steps Δt_A and Δt_B with an integer time-step ratio $m = \Delta t_A / \Delta t_B$ between them, then subdomain Ω_B is simply solved m times for every time that subdomain Ω_A is solved. As an example, consider the beam problem in Figure 2 with a time-step ratio of $m = 2$. To study the notch at a finer granularity, subdomains Ω_1 and Ω_2 will be simulated at smaller (finer) time step Δt while the remaining two subdomains are simulated at a larger (coarser) time step ΔT. Thus, subdomains Ω_1 and Ω_2 are solved twice for each solve of subdomains Ω_3 and Ω_4. Coupling between the different problem timescales in step (ii) of the TreeSolve procedure is performed only once, at the larger time step Δt_A. Finally, the update in step (iii) requires that the solutions for subdomains Ω_A and Ω_B be updated to synchronize the solution and enforce its continuity across the subdomain interfaces. Although the multi-time-step method is valid for any number of timescales in the problem, in this study, we focus on the most common scenario, that is, problems with two different timescales. Finer details of the multi-time-step method can be obtained from the references [14, 18].

As pointed out in Section 2.1, the space of possible decompositions and the number of corresponding trees is very large, even without the presence of multiple timescales. The presence of different timescales associated with each subdomain adds a whole new dimension to this space. Now, because every node of every tree from Section 2.1 can be associated with its own timescale, the number of possible ways to choose these timescales is practically infinite. However, there are two constraints that the multi-time-step method imposes on the choice of time steps [18]. For any interior (parent) tree node, all child nodes must be solved using a time step that is either smaller than or identical to the time step of the parent node. In addition, subdomains at the same time-step quantization level must be coupled prior to coupling them with subdomains at a different time-step level. Despite this restriction, the number of possible decompositions in the search space is infinite.

2.3. Computational cost modeling

As described in Section 2.1, for a given decomposed mesh, the computation involves three main operations: (1) uncoupled subdomain solve, (2) interface solve for coupling, and (3) update. We adopt a simple model to quantify the computational cost of these operations. This is performed by analyzing the input problem to determine its properties, namely, the subdomain sizes (n_A and n_B) and the interface size ($n_{(A,B)}$). With this information, we can approximate the matrix sizes for all operations to estimate the time it will take to run during actual execution. There are several different cost components that contribute to the TreeSolve cost. Most of them are matrix–matrix or matrix–vector multiplications as well as a few matrix–matrix and matrix–vector solves. A detailed description of these costs is given in [15]. However, certain operations dominate the computational cost. To simplify our cost model, we approximate the cost of subdomain solves (against prefactorized matrices) to be order n_A^2 and n_B^2, the coupling cost to be order $n_{(A,B)}^2$, and the cost of updating the subdomain solutions is $n_A n_{(A,B)} + n_B n_{(A,B)}$. For problems with uniform timescales, this simplified cost model

DOI: 10.1002/nme

was found to work well for determining the cost associated with a given coupling configuration. For the multi-time-step method, one simply scales the costs of the subdomain running at the smaller time step by the time-step ratio m. Thus, the total computational cost of running a problem with the multi-step-method is $n_A^2 + m\, n_B^2 + n_{(A,B)}^2 + n_A\ n_{(A,B)} + m\ n_B\ n_{(A,B)}$.

2.4. *Time-step values and quantization*

In order to determine an appropriate time step for each subdomain, first, each element in the global mesh is assigned a default time step based upon an approximated CFL condition [19]. For problems in solid mechanics, this can be approximated as time taken by a wave to propagate through the element:

$$\Delta t_e \leqslant \frac{l_e}{c}, \tag{6}$$

where c is the wave speed calculated as $\sqrt{E/\rho}$, E is Young's modulus and ρ is the density of the material, and l_e is the shortest distance between any two nodes of the element. Thus, smaller elements are associated with smaller (finer) time steps. From purely a stability point of view, it is possible to run elements at a time step greater than that given by the CFL condition – using an implicit time-integration scheme, for instance. However, we postulate that, in a well-designed mesh, the time step required for accuracy of the solution (in capturing the transients) would be close to the CFL time-step criterion. Thus, in this work, because of stability and accuracy considerations, we assign the default time step for each element based on the approximate CFL condition earlier.

Usually for complex problems, the number of elements in a given mesh and the variation in size of these elements (and the corresponding time steps) is quite large. For instance, Figure 3 shows a distribution of the workload associated with individual elements for the beam problem shown in Figure 1 in ascending order of their sizes. There are 36,552 elements in the mesh with the ratio of the size of the largest element to size of the smallest element around 100. We assign the element with the largest size (and consequently the largest time step, ΔT) in the mesh, a workload of 1. Workloads for the other elements in the mesh are approximated by taking the inverse of the ratio of their time steps Δt_e to the maximum time step ΔT in the mesh.

Rather than run each element in the mesh with a different time step, we quantize the entire range of time steps into multiple discrete levels for computational efficiency, as shown in Figure 3. Because individual elements can be assigned any time-step value smaller than the CFL limit, we round the time step of each element down to the immediately lower discrete quantization level in the mesh (causing its workload to be rounded up as shown by the dashed lines in Figure 3). Once all the elements are assigned to discrete levels of time steps, the time step for subdomains can be determined. To ensure numerical stability and accuracy of the resulting solution, the maximum value of time step that can be assigned to a particular subdomain is equal to the smallest time step among all the

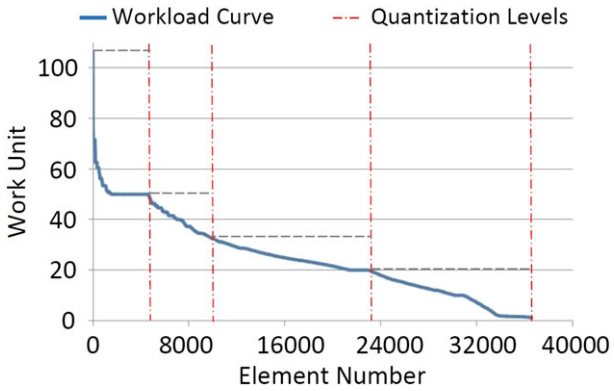


Figure 3. Timescale quantization.

DOI: 10.1002/nme

elements in that subdomain. A value higher than this maximum may be chosen for implicit time-stepping methods, but that may result in loss of accuracy. Any value lower than this maximum can indeed be assigned to an element or subdomain, however, that will result in higher computational cost. In this work, we restrict our attention to two quantization levels, that is, one large time-step level and one small time-step level. We define the point at which the workload curve is split as (workload of the small time-step level)/(workload of the large time-step level) and call it the split ratio (SR).

3. MULTI-LEVEL PARTITIONERS

Using the cost model described in Section 2.3, we can determine if it is beneficial to decompose a given subdomain by comparing the cost of solving the subdomain as a whole versus the cost of the prospective decomposed subdomains. Further, if it is worthwhile to decompose a given subdomain, then the choice of an optimal time-step ratio between the resulting decomposed subdomains can also be made using this cost model. In order to facilitate such decisions, the decomposed mesh is abstracted as an undirected graph, as shown in Figure 4. In the graph, each node (or vertex) represents an individual subdomain, and edges represent shared interfaces between them. In essence, the graph represents the topology of the subdomains. The graph nodes and edges are assigned weights based on the cost of their subdomain solves and coupling operations, respectively.

Mesh and graph partitioning to achieve minimum edge cuts (communication) and optimal node balance between partitions (computation) is an NP-complete problem. However, state-of-the-art tools like METIS [16] produce quality partition inexpensively. In the following sections, we provide an overview of METIS and its limitations for multi-scale methods and describe the search space of possible decompositions for a given finite element mesh.

3.1. Phases of multi-level partitioners

Figure 5 shows the partitioning strategy employed by METIS. METIS uses a multi-level bisection algorithm, where a large graph is coarsened down to a few hundred vertices, which is then bisected

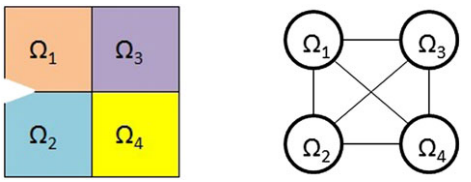


Figure 4. Graph representing a four subdomain system.

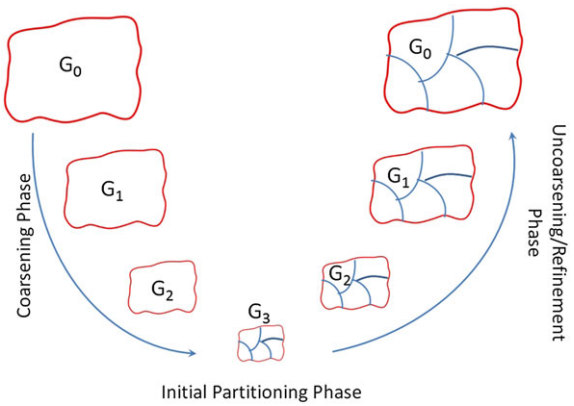


Figure 5. Illustration of multi-level graph partitioning algorithm.

DOI: 10.1002/nme

and the partitioning is reflected back to the original graph. METIS uses a three-phase partitioning method: (1) graph coarsening, (2) initial partitioning, and (3) graph uncoarsening/refinement.

3.1.1. Graph coarsening. In this phase, a sequence of smaller graphs are constructed from the original graph by collapsing edges and merging vertices. At each step of this sequence, a maximal matching of the graph is obtained, and all the edges connecting matched vertices are collapsed by merging these vertices. Three methods are commonly used to achieve this maximal matching: (1) random matching (RM), (2) heavy edge matching (HEM), and (3) modified HEM. In the RM method, vertices are chosen randomly and if it is not already matched, then it is matched with one of its adjacent vertices, which is also chosen randomly. In the HEM matching, like RM, the vertices are visited in random order but the adjacent vertex is picked that has the highest edge weight among all the adjacent vertices. Modified HEM is similar to HEM but if there exist more than one edge with the highest weight, then the edge that reduces the overall degree of the graph is matched.

Upon merging, the weight of the merged vertex is taken as the sum of the weight of two vertices that are to be merged. If a third vertex is adjacent to these two vertices, then the edge weight of the merged vertex to this third vertex is obtained as the sum of the two merged edge weights. This approach coarsens the graph, and the process is repeated until a few hundred vertices remain in the resulting graph (the actual number depends on the size and topology of the original graph). The coarsening method ensures that the edge cut of a partition in the coarser graph is equal to the edge cut of the same partition in the finer graph, and balancing the partitions in the coarser graph lead to balanced partitions in the finer graph.

3.1.2. Initial partitioning. In this phase, a k-way partitioning of a coarse graph is computed such that each partition contains roughly v_0/k vertices, where v_0 is the total number of vertices in the original graph. This k-way partitioning is usually computed using a multi-level bisection algorithm [16, 20].

3.1.3. Graph uncoarsening/refinement. In this phase, the partitioned coarse graph is projected back to the original graph by going through all levels of coarser graphs that were generated in the coarsening phase. Note that even though the partitioned coarse graph obtained in the initial partitioning phase is a local minima, the projections of this partitioning to the upper-level graphs need not be their local minima. Therefore, a simplified version of Kernighan–Lin algorithm [16] is used to refine each level of coarse graphs to achieve local minima. Figure 5 shows the three phases of the multi-level graph partitioning algorithm.

3.2. Limitations of multi-level partitioners

METIS is widely used for graph partitioning in science and engineering problems. In most applications, initial weights are assigned to vertices and edges of a graph, which is then passed on to METIS for partitioning. The required number of partitions are either specified by the user or determined a priori based on application-specific requirements. The resulting partitions are assigned a particular time step based on the quantization process described in Section 2.4. We call this approach Naive METIS (NMETIS), and it represents the baseline decomposition for a problem in our simulations in Section 5.

One drawback of using the NMETIS approach is that there is no efficient way for determining the optimal number of partitions needed for best performance. Essentially, one has to try out a range of different number of partitions and check the performance of the resulting decomposition. Our results show that not picking a close to optimal number of partitions has a severely adverse affect on performance.

A limitation of METIS itself, is that its partitioning approach does not account for the element size and associated physics of the problem, as described in Section 2.4. Multi-scale methods, in particular, are very sensitive to size of the elements. For optimal performance, smaller and larger elements should be in different partitions. Grouping them together will result in forcing the larger elements to be run at a lower time step leading to unnecessarily higher computational cost. However, METIS tries to balance number of elements between subdomains, which often leads to a mixture

DOI: 10.1002/nme

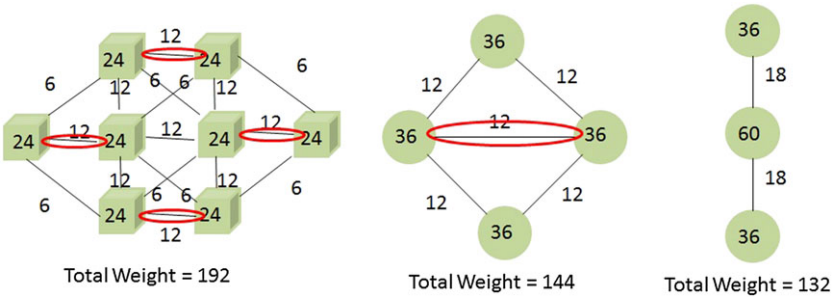


Figure 6. Illustration of coarsening phase exploiting domain knowledge.

of small and large elements in the same partition. In the extreme case, if the number of partitions required is small or if the regions of interest (with small elements) are scattered in multiple locations in the global mesh, then it is highly likely that all partitions will have a mixture of small and large elements. This will lead to running the entire mesh at a small time step, which is computationally very inefficient.

Finally, METIS also does not account for domain-specific nuances associated with the particular application. For the multi-time-step recursive domain decomposition method employed in this study, the weights of the graph nodes and edges in the multi-level partitioning cannot be simply added together during merging operations. Instead, one needs to account for the computational cost of the partitioned subdomains and the coupling costs in accordance with Section 2.3. For example, consider the case of eight hexahedral elements shown in Figure 6 that need to be solved as eight separate subdomains. The node weights in the corresponding graph are based on the fact that each subdomain (or element, in this case) requires 24 equations to be solved, and the edge weights are based on the number of nodes being shared between the subdomains. During the coarsening phase, when the edges marked red are collapsed, the total weight should be determined by accounting for common FEM nodes. However, because METIS simply sums up the weights for vertices and edges, there is no proper mechanism to account for this domain-specific artifact. For complex meshes with large number of elements, the weights on the coarsest graph produced by METIS can be very different from what the problem demands, potentially leading to suboptimal partitions.

3.3. Mesh decomposition search space

For multi-scale problems in solid mechanics solved with the multi-time-step recursive domain decomposition method, there are several parameters that influence performance. These include (1) total number of elements in the original mesh, (2) mesh topology, (3) variation in element sizes, (4) timescales associated with each element, (5) number of subdomains in the decomposition, (6) number and choice of time-step quantization levels, (7) assignment of individual elements to subdomains, and (8) assignment of subdomains to different time steps. Parameters (1)–(4) are related to mesh quality, whereas parameters (5)–(8) pertain to mesh decomposition and are tightly coupled to each other. Any of these parameters can have a considerable impact on performance. For instance, for the multi-time-step approach itself to be beneficial, the variation of element sizes needs to be sufficiently large, and the mesh needs to be designed and graded in a manner that captures the spatial and temporal scales of the problem. Similarly, for the same global mesh, even small changes in the number of subdomains and/or timescale quantization levels may lead to widely different performance.

In the presence of all the earlier parameters, the space of possible decompositions is enormous, and finding an optimal decomposition from this space is very difficult. The goal of this study is to develop a domain-specific multi-level partitioner that exploits domain knowledge to produce close to optimal mesh partitions automatically without the need to search through the entire space. Assuming that a good quality global finite element mesh is provided as input, the parameters we optimize for are the mesh decomposition parameters (5) to (8) earlier. For this study, we restrict ourselves to only two quantization levels for the timescales, which still covers a large number of multi-scale problems.

DOI: 10.1002/nme

3.4. Hand-tuned partitioning for multiple timescales

To overcome the limitations of the NMETIS approach, one way to avoid mixture of large and small elements in the same partition is to quantize – select the time steps for each element – before partitioning the graph. For example, a global mesh can be approximately partitioned (manually) into two (or more) subdomains by grouping elements in certain regions of interest into one subdomain, which can be run at a smaller time step. The remaining elements form another subdomain, which can be run at a larger time step. For the problem shown in Figure 1, elements within a certain radius around the notch-tip (shown in red) are picked to be run at a small time step, and the remaining elements are run at a large time step. The subgraphs corresponding to these subdomains are then passed to METIS separately for further partitioning where, once again, one would need to guess the number of subdomains at each time-step level. Domain scientists usually follow this approach, which we call the strawman heuristic (STWMN) in our simulations in Section 5.

Despite separation of timescales by quantization, the STWMN approach also suffers from some of the same limitation as that of NMETIS, that is, picking the optimal number of partitions and the number of partitions for each timescale is not easy. Another drawback with this approach is that the optimal value of the radius around the notch tip also needs to be determined by trial and error. In this approach, with the radius being another geometric parameter to optimize for, finding the optimal decomposition is even more challenging. Further, note that with this approach, the time steps are quantized based on the radius as well. A large radius results in a large time-step ratio between the subdomains, causing more elements to be run at the smaller time step, and therefore worsening performance. Alternatively, a small radius may result in a small time-step ratio, which leads to more TreeSolve executions for the subdomain with the larger time step, again leading to performance loss. In addition, this approach may lead to a complicated irregular interface between the subdomains at the two time-step levels and that can be an additional bottleneck.

As an alternative to manual quantization using an *ad hoc* radius, one may quantize timescales based on the workload associated with the resulting subdomains by utilizing domain knowledge about the computational cost of the method in a manner similar to that shown in Figure 3. For instance, with two quantization levels, one could either pick a point on the workload curve that splits the work evenly between the large time step and the small time step or pick a particular workload split that leads to a minimum total computational cost. However, irrespective of the criteria that one uses to quantize, the problem with conducting quantization at the beginning of partitioning process is that at this stage, the computational costs are based on the timescales of individual elements, whereas the actual costs of the subdomains that finally result from the partitioning can be quite different. Thus, both quantization at the beginning of partitioning (as with the STWMN approach), or at the end of partitioning (as with the NMETIS approach), lead to suboptimal performance.

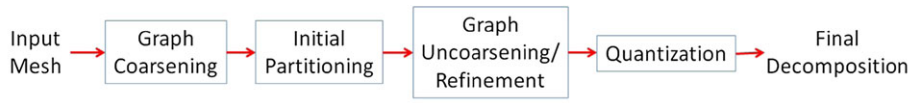
4. DOMAIN-SPECIFIC MULTI-LEVEL PARTITIONER

In this section, we explore possible ways to incorporate domain knowledge in the partitioning process. We also present our domain-specific multi-level partitioner that exploits this domain knowledge in each phase of partitioning and utilizes the domain-specific cost-model to produce an optimized mesh decomposition automatically. In contrast to the NMETIS and STWMN approaches, which conduct quantization of the problem timescales in specific phases of the partitioning process (consequently treating it as an invariable parameter outside of those specific phases), we adopt an approach where we utilize knowledge of the timescales (and other problem parameters) throughout the partitioning process. Figure 7 shows a comparison of the partitioning process for different approaches discussed in this work. Figure 7(a) represents NMETIS, (b) represents STWMN, and (c) represents our domain-specific multi-level partitioning approach DSMLP. The following subsections detail our approach.

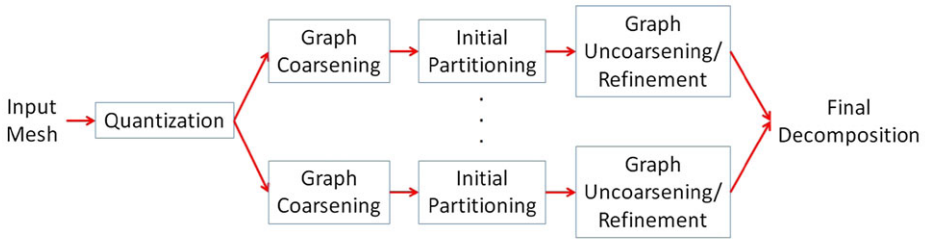
4.1. Initial representation

As with METIS, we start by representing the finite element mesh as a graph as shown in Figure 4, where each vertex represents a single element and the edges represent the shared interface between

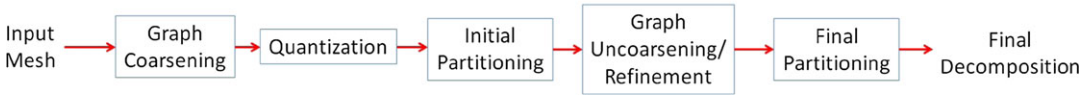
DOI: 10.1002/nme



(a) Partitioning phases for NMETIS



(b) Partitioning phases for STWMN



(c) Partitioning phases for DSMLP

Figure 7. Partitioning phases in various decomposition approaches. (a) Partitioning phases for NMETIS, (b) partitioning phases for STWMN, and (c) partitioning phases for DSMLP.

elements. The vertices and edges are assigned initial weights in accordance with the number of equations (degrees of freedom) of the element and its interface with adjacent elements. In addition to this basic representation, each vertex is also assigned an approximate time-step value determined by Equation (6). The range of time steps across all elements, ranging from the minimum (Δt) to the maximum (ΔT), is also recorded.

4.2. *Graph coarsening*

In this phase, like METIS, a sequence of smaller graphs are constructed from the original graph by collapsing edges and merging vertices. At each coarsening level, a maximal matching of the graph is obtained, and all the matched edges are collapsed. However, unlike METIS, we maintain proper weights by taking into account the common degrees of freedom when merging edges and vertices. As shown in Figure 6, when merging two vertices or edges, the weight of common equations is subtracted. Further, the time step of the merged vertex is chosen to be the smaller of its two component vertices.

To construct the maximal matching, we modify the HEM method from METIS. In our modified method, which we call NHEM, the edges are sorted in a descending order based on their weights, and the edge with the maximum weight is collapsed first. At a particular coarsening level, if both adjacent vertices of a matched edge have already been matched, then that edge is skipped until all the vertices in the graph have been matched (except possibly a few vertices that may be left over after the edge-matching process). This creates roughly balanced partitions with similar number of elements in each partition. With NHEM, in addition to achieving balanced partitions, our goal is to reduce the overall interface size (communication cost) and by merging heavy weight edges, we achieve this. We also implemented HEM from METIS and found that it produced similar decompositions as those from our NHEM approach.

When collapsing edges with NHEM, we also utilize domain knowledge of timescales associated with individual elements. To separate elements of dissimilar sizes (and dissimilar timescales) into different quantization levels, we incorporate additional restrictions to the vertex merging and edge collapsing process in NHEM. We merge two vertices only if the ratio of the difference of their time steps to the mean of their time steps is small (less than one-half in our case). While this ensures that elements with dissimilar timescales are not merged, we observed that in some cases, there are

DOI: 10.1002/nme

outlier vertices (with large time-step values), that do not get merged with any other vertices in the graph, resulting in some partitions with only a few vertices. To overcome this issue, we determine the mean time step among all the elements ($\overline{\Delta t}$), and vertices with time steps greater than $2\overline{\Delta t} - \Delta t$ are then merged without the earlier restriction on time-step ratio. Finally, we utilize the domain-specific cost model, described in Section 2.3, to determine the change in overall cost by merging vertices. We collapse an edge only if the cost of merged vertices is less than the cost of each vertex plus their interface.

4.3. Initial partitioning

In this phase, the coarsest graph produced in the coarsening phase is partitioned and quantized. However, unlike METIS, which computes a k-way partitioning of the coarsest graph, we construct only two partitions, (one for each of the two timescale quantization levels). This graph bisection is based on a workload split that minimizes total computational cost in a manner similar to that shown in Figure 3, with the restriction that only two timescale quantization levels are used. We run through different workload split combinations and use the cost model to select the particular workload distribution that has the least cost. This creates two partitions Ω_A and Ω_B with time steps Δt_A and Δt_B, respectively. Note that at this stage, this workload distribution is just an estimate because the refinement phase may shift several elements between the two partitions leading to a changed workload distribution. At this stage, like STWMN, the interface between the two partitions is also large and irregular.

4.4. Graph uncoarsening/refinement

Once we have obtained initial partitioning, we reflect this partitioning back to the original graph by going through all levels of the coarsened graphs. To reduce and smoothen the interface between timescales (achieve local minima), we use the following refinement method to move vertices between the two partitions. Searching across all vertices on the edge of the partition, a vertex is moved from one partition to the other if the weight of the edge cuts between partitions is lowered by moving it. In our approach, any vertex from the larger time-step partition (Ω_A) can be shifted to the smaller time-step partition (Ω_B) as long as the edge cut is lowered. Theoretically, a vertex cannot move from Ω_B to Ω_A in order to preserve accuracy and stability. However, with our quantization approach, there is a possibility that a vertex has been assigned to Ω_B, while its actual timescale is much higher, potentially at the level of Ω_A. We allow such vertices to move to Ω_A, as long as it does not violate accuracy and stability requirements. This refinement process is applied at each level of coarsened graph up to the original graph, and it slightly changes the estimated workload distribution performed at the initial partitioning phase. There is a trade-off between reducing interface size and moving more elements to lower timescale. While reducing the interface between Ω_A and Ω_B usually leads to lower computational cost, having more elements in Ω_B increases it. We have found that often it pays off to move more elements at lower timescale at the expense of a larger interface.

4.5. Final partitioning

In the final partitioning phase, the total number of subdomains required at each timescale is determined. After graph uncoarsening/refinement phase, we now have two partitions (one for each timescale) and a final assignment of elements to these two partitions. We begin by representing this partition as a tree containing only three nodes, two leaf nodes for each of the two subdomains at the two timescales, and their parent node representing the entire problem domain. On each leaf node, we use the cost model to determine if splitting a given subdomain helps lower total cost. If the cost is reduced by splitting a node, then it is bisected (using METIS) to create two new leaf nodes. Each leaf node is recursively bisected until the cost cannot be reduced any further. With this approach, we now have determined the total number of subdomains, as well as, subdomains for each timescale automatically and this is our final decomposition. This approach helps us avoid guessing the number of subdomains for each timescale and finds an optimal decomposition out of the entire search space automatically.

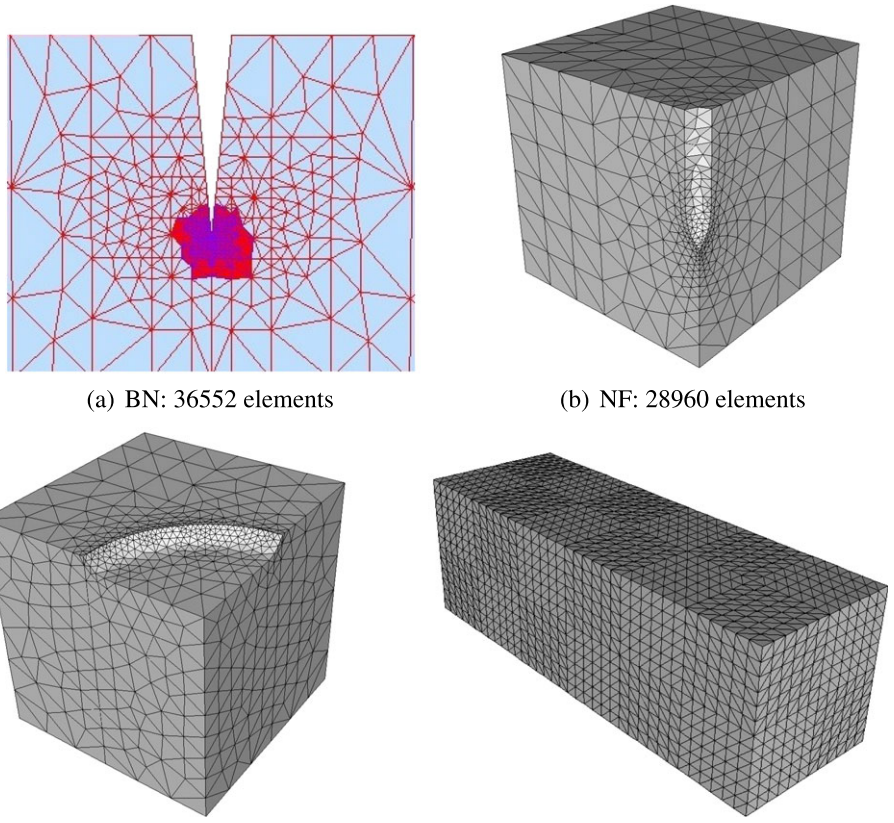
DOI: 10.1002/nme

5. EVALUATION

We evaluate the performance of our domain-specific partitioning approach DSMLP and compare it against NMETIS and STWMN described previously. We focus on the results from four physical systems (shown in Figure 8), represented as finite element meshes of tetrahedral elements, as testing systems: (1) beam mesh with a notch (BN) in Figure 1 with 36,552 elements; (2) cube mesh containing a needle-shaped inclusion (NF) with 28,960 elements; (3) cube mesh with a disk-shaped inclusion (DF) with 42,800 elements; and (4) cuboid with two needle-shaped and one disk-shaped inclusions (CF) with 100,720 elements. STWMN is evaluated for only BN problem because it is not feasible to apply the simple STWMN approach (described in Section 3.4) to identify suitable areas of interest for the other test problems. These problems are simulated using the multi-scale solver running on an Intel Xeon E5-4650 system configured with four 8-core processors (total of 32 cores) running at 2.7 GHz. The simulation is run up to 1 ms, which is on the order of 100 to 1000 TreeSolve time steps, depending upon the actual timescale chosen by the partitioner.

5.1. Performance comparisons

We compare the execution times for running our heuristic, DSMLP, with NMETIS and STWMN, on the four test inputs. In order to explore the search space of mesh decompositions, we run NMETIS

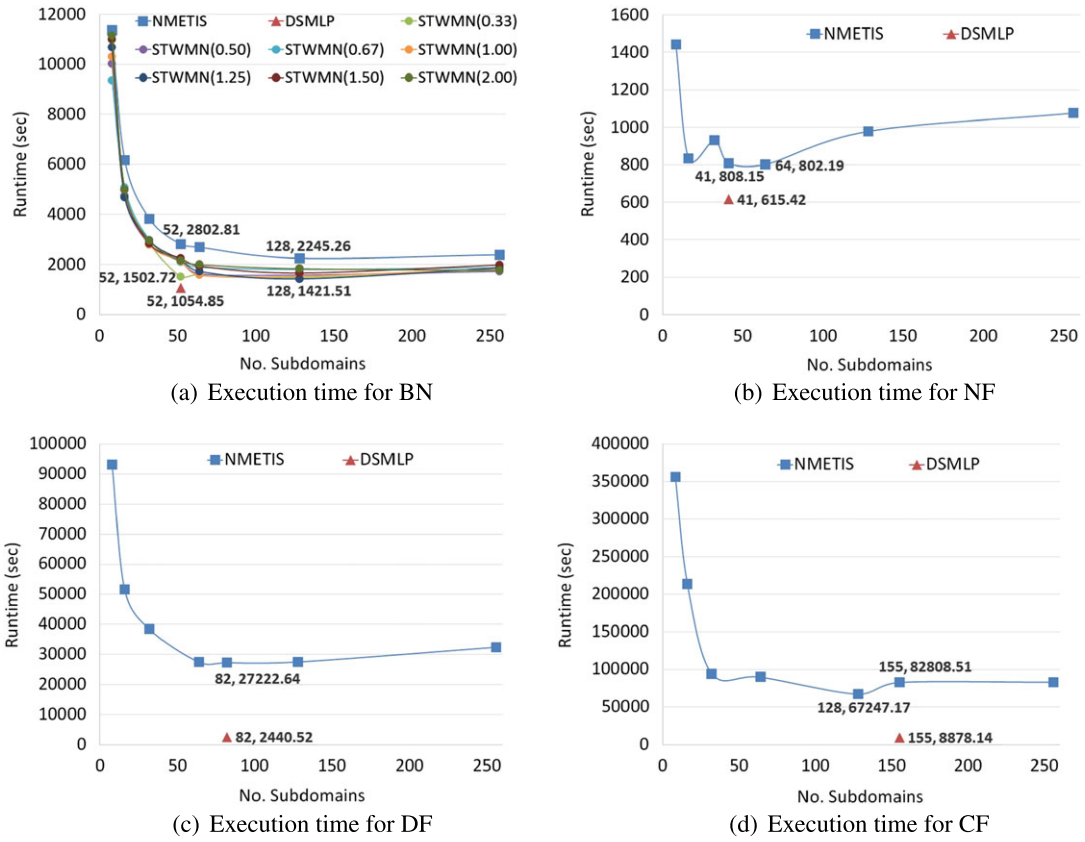


(a) BN: 36552 elements (b) NF: 28960 elements

(c) DF: 42800 elements (d) CF: 100720 elements

Figure 8. Input meshes used for evaluation: (a) BN: beam with a notch (same as Figure 1 zoomed in at the notch where the red colored elements represent the small time-step domain) (36,552 elements); (b) NF: one-eighth symmetric section of a cube with a needle-shaped inclusion (28,960 elements); (c) DF: one-eighth symmetric section of a cube with a disk-shaped inclusion (42,800 elements); (d) CF: cuboid constructed by joining three cubes end to end, containing two needle-shaped and one disk-shaped inclusions (100,720 elements).

DOI: 10.1002/nme



(a) Execution time for BN

(b) Execution time for NF

(c) Execution time for DF

(d) Execution time for CF

Figure 9. Runtime for solving decompositions produced with various partitioners. (a) Execution time for BN, (b) execution time for NF, (c) execution time for DF, and (d) execution time for CF.

and STWMN with different numbers of partitions, chosen from $\{1, 2, 4, \ldots, 256\}$ and record their respective sequential runtimes in Figure 9. Recall that NMETIS and STWMN require a preselected number of partitions, while DSMLP chooses the number of partitions automatically. The results shown here start from eight subdomains because with subdomains less than that, the runtimes are too high. For assigning the elements to higher and lower timescales for STWMN, we use the radii of lengths 0.33, 0.5, 0.67, 1.0, 1.25, 1.5, and 2.0 m from the notch tip. NMETIS serves as our baseline for comparison. Note that runtimes of our DSMLP approach are consistently smaller in comparison with the other two approaches, including the hand-tuned STWMN approach for the BN problem. We also partition NMETIS and STWMN into 41, 52, 82, and 155 subdomains for NF, BN, DF, and CF meshes, respectively, because our DSMLP automatically determines that these are optimized number of subdomains for their respective meshes.

There are several takeaways from the plots in Figure 9. Firstly, the search space for mesh decomposition is very large with a huge variation in performance between various decompositions, and finding the optimal decomposition is not easy. Secondly, as the number of subdomains increase the runtime decreases up to a certain limit after which it starts to increase again. This is because the cost of solving a subdomain is of order $\boldsymbol{n}^2$ and with large subdomain size (fewer number of subdomains), the runtime is very high. On the other hand, with small subdomains, the resulting interface size between the subdomains is comparable with the size of the subdomains themselves, which leads to more time spent on coupling operations at the interface, hence overall poor performance.

Another factor that controls the runtimes is the timescale ratio between the higher and lower timescales. The relationship between number of subdomains and timescale ratio is very unpredictable. This can be observed from sudden increases and decreases in runtimes, especially in the NMETIS case. For example, in the case of NMETIS with less number of subdomains, small-sized

DOI: 10.1002/nme

Table I. Performance of different partitioning approaches for the BN problem.

	NMETIS		STWMN (0.33)		DSMLP	
	SDT	LDT	SDT	LDT	SDT	LDT
NSUB	48	4	32	20	14	38
DTR	36		2		2	
CPT	1.22		194.10		389.02	
SI	111		2421		1899	
NEL	33,797	2755	22,707	13,845	12,108	24,444
NEQ	31,113	2517	20,697	12,768	10,449	22,680
LS	1446.08	3.00	763.66	228.87	125.08	138.88
CP	1350.63	0.07	279.60	35.83	153.29	248.24
RUN	2802.81		1502.72		1054.85	

elements end up in each of the subdomains, resulting in timescale ratio of 1, that is, running the entire problem at the lower timescale. With large number of subdomains, the timescale ratio is high, leading to fewer TreeSolve calls but also a large number of elements at the lower timescale. Because lower timescale subdomains have to be solved multiple times (depending on the timescale ratio m) for each call of TreeSolve, the runtime is usually high. However, in some cases, further increasing the number of subdomains does not significantly increase the timescale ratio but reduces the runtime at lower timescales through addition of more subdomains. With this erratic behavior, it is very difficult to determine the number of subdomains for optimal decomposition with the NMETIS approach.

We observe that our DSMLP partitioner (with a runtime of 1054.85 s at 52 subdomains) shows speedup of 1.35 and 2.13 for the BN problem over the best produced decompositions by STWMN (1421.51 s at 128 subdomains) and NMETIS (2245.26 s at 128 subdomains), respectively. Similarly, compared with NMETIS, DSMLP shows speedup of 1.30, 11.15, and 7.57 for the NF, DF, and CF meshes, respectively. Note that while the speedup achieved by DSMLP over decompositions produced by NMETIS and STWMN is moderate for small meshes, for larger meshes, it reduces the runtime of a problem by an order of magnitude. These results show that incorporating domain-specific knowledge into the partitioning process effectively helps us in achieving almost optimal partitioning automatically.

To compare the quality of mesh decompositions produced by NMETIS, STWMN, and DSMLP, we decompose the meshes with the same number of subdomains for NMETIS and STWMN as what is determined automatically by DSMLP. The BN mesh is decomposed with NMETIS and STWMN using 52 subdomains – the same number of subdomains as that determined by DSMLP automatically. With 52 subdomains, STWMN works best with radius of 0.33 m so we use results obtained by this radius. NF, DF, and CF meshes were decomposed with 41, 82, and 155 subdomains, respectively, with NMETIS.

Tables I–IV show numbers distributed between lower timescale (SDT) and higher timescale (LDT) for all decompositions. In these tables, NSUB represents the number of subdomains at the lower and higher timescales, DTR represents the timescale ratio, CPT the coupling cost (in seconds) between the timescales at the top (root) node of the tree, and SI represents the size of the shared interface between lower and higher timescales. Further, NEL represent number of elements, NEQ represents the sum of number of equations to be solved, LS represents the sum of the runtimes of all leaf solves (total cost of solving a subdomain), and CP represents the sum of all coupling costs at each timescale. Finally, RUN gives the total runtime (in seconds) for the problem and is the sum of CPT, LS, CP times, and a small overhead because of calls to the TreeSolve subroutine.

These tables confirm that NMETIS, without domain knowledge about the timescales, tries to balance number of elements per subdomain (given by the ratio NEL/NSUB). This leads to poor performance because it causes a majority of the elements to be assigned to the lower timescale and

DOI: 10.1002/nme

Table II. Performance of different partitioning approaches for the NF problem.

	NMETIS		DSMLP	
	SDT	LDT	SDT	LDT
NSUB	22	19	10	31
DTR	8		4	
CPT	104.41		136.06	
SI	1709		1308	
NEL	15,437	13,523	6772	22,188
NEQ	14,502	11,859	5844	20,133
LS	204.30	17.75	114.44	81.58
CP	446.31	35.23	57.49	225.56
RUN	808.15		615.42	

Table III. Performance of different partitioning approaches for the DF problem.

	NMETIS		DSMLP	
	SDT	LDT	SDT	LDT
NSUB	40	42	17	65
DTR	188		53	
CPT	373.62		563.49	
SI	3036		1491	
NEL	20,765	22,035	1352	41,448
NEQ	20,994	20,992	2286	39,711
LS	5614.75	28.96	69.79	318.21
CP	21,081.95	123.22	11.18	1476.65
RUN	27,222.64		2440.52	

Table IV. Performance of different partitioning approaches for the CF problem.

	NMETIS		DSMLP	
	SDT	LDT	SDT	LDT
NSUB	79	76	36	119
DTR	197		61	
CPT	2013.33		2684.15	
SI	7137		3564	
NEL	51,284	49,436	7362	93,358
NEQ	48,927	46,311	8316	86,544
LS	17,935.42	84.83	1113.17	615.57
CP	62,372.40	402.50	1051.45	3413.67
RUN	82,808.51		8878.14	

produces a large timescale ratio. For the BN problem, STWMN performs better than NMETIS by having comparatively fewer elements at the lower timescale. However, as can be seen from leaf solve times (LS), there is still a large workload imbalance between the lower and higher timescale with STWMN. One reason for this is that there are more number of elements at the lower timescale, and

DOI: 10.1002/nme

consequently, there is more work at the lower timescale. In addition, however, the number of subdomains at both the timescales is largely determined by the number of elements at these timescales and not by the actual computational cost of the resulting tree. This results in more number of subdomains at the higher timescale (overdecomposed) than it actually should have for optimal performance. As is confirmed by Figure 9, decomposing a given domain into the correct number of subdomains is important for good performance, and having either lesser or greater number of subdomains than the optimal number degrades performance.

The degrading effect of not having an optimal number of subdomains can also be observed in Tables I–IV. For the BN problem, even though the total number of subdomains is 52 for all three approaches, DSMLP performs best because the distribution of subdomains within each timescale is suboptimal for the NMETIS and STWMN approaches. DSMLP assigns fewer elements at the lower timescale with a lower timescale ratio, reducing the total amount of work. Additionally, by exploiting domain knowledge of computational cost, DSMLP is able to decompose the mesh into an optimal number of subdomains for each timescale, thereby balancing the workload between timescales and lower overall leaf solve runtimes. However, because of fewer elements at the lower timescale, the interface between timescales becomes large and irregular as shown in Figure 8(a). Nevertheless, DSMLP balances the leaf solve costs, the coupling costs within each timescale, and the coupling cost between timescales yielding an almost optimal decomposition that performs as well as, or better than the decompositions produced by other approaches.

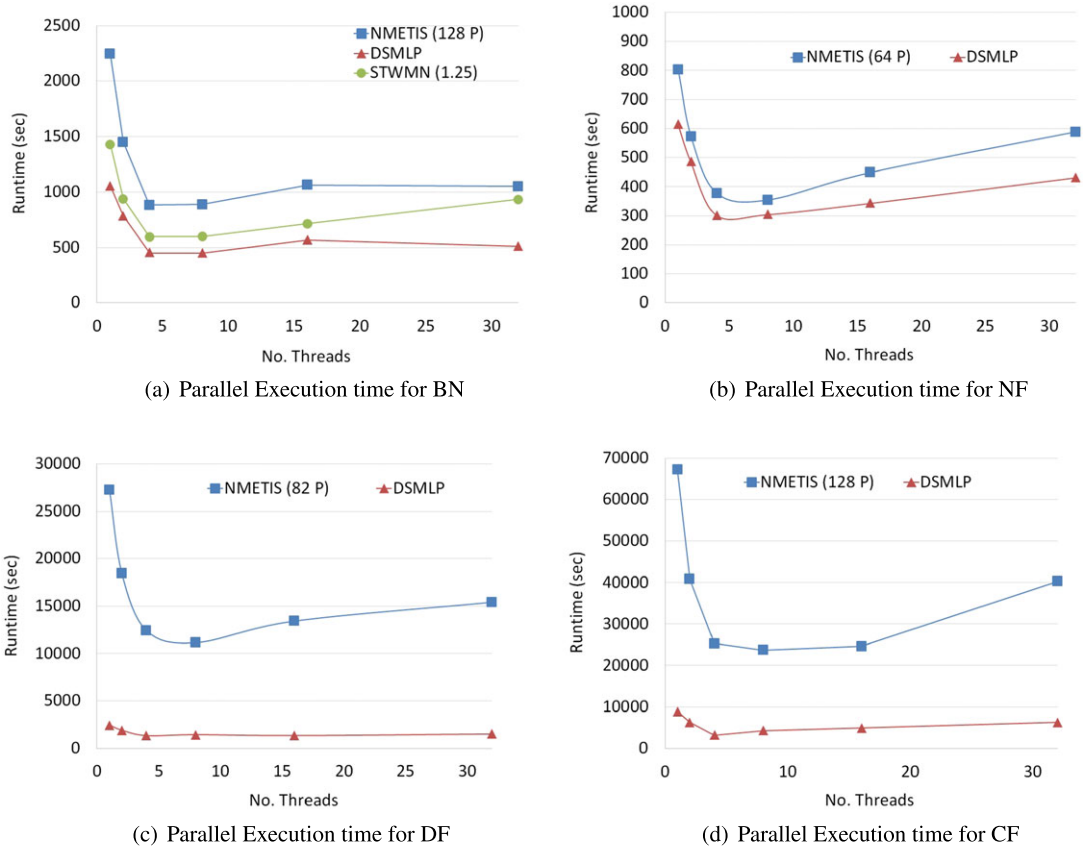
5.2. Parallel performance

A parallel implementation of the multi-time-step coupling algorithm is developed using Cilk [21] – an algorithmic multithreaded language used to optimize parallel programs. We chose to use Cilk because it is a minimally intrusive parallelization tool that deals with irregular codes without introducing large synchronization overheads. The parallel TreeSolve calls the Cilk multithreaded runtime system using fork-join parallelization at each tree node, where both child nodes of an interior node are solved on separate threads concurrently, synchronized, and then coupled with each other. This allows each subtree path to run concurrently, while Cilk's work-stealing mechanism effectively handles any load balance issues that arises because of complexities of the recursive algorithm and multiple timescales. Further details of this approach are given in [15].

Figure 10 compares the execution times for each partitioning approach across different number of threads for both input meshes. We compare DSMLP with best performing decompositions from NMETIS and STWMN in the serial case. Referring to Figure 9, the BN, NF, DF, and CF problems are decomposed into 128, 64, 82, and 128 subdomains, respectively, using NMETIS and STWMN (in case of BN mesh). For STWMN, we pick a radius of 1.25 m with 128 subdomains as it was the best performing decomposition among all STWMN decompositions. We note that in all cases, the DSMLP approach delivers much better parallel performance than METIS and STWMN, especially with larger meshes where we see a significant performance improvement. Table V shows the speedup of STWMN (for the BN problem) and DSMLP (for all problems) over the baseline NMETIS for both one and eight threads. We also observe that while DSMLP performs the best in single-threaded and multi-threaded execution, the relative speedup over the baseline NMETIS may or may not increase at higher thread counts.

For all decompositions, performance scales up to four threads and speedup is obtained when increasing the number of threads. For eight threads and beyond, the results do not scale well. Using the Cilk profiling tool [21], we found that for the input meshes considered here, past four cores, the runtime of the critical path approaches the total runtime for the problem. This indicates that the parallelism saturates around four to eight threads and not much more parallelism can be exploited. The lack of scalability is explained by the structure of the inputs that leads to inherently imbalanced decompositions as can be seen in Tables I–IV. For DSMLP, solving for the interface between timescales (CPT at the root node) is 20% to 40% of total workload and cannot be parallelized. Nevertheless, because DSMLP produces less work (because it optimizes for work load), its overall runtime is still faster than NMETIS, and hence, it performs the best.

DOI: 10.1002/nme



(a) Parallel Execution time for BN

(b) Parallel Execution time for NF

(c) Parallel Execution time for DF

(d) Parallel Execution time for CF

Figure 10. Execution times across different number of threads. (a) Parallel execution time for BN, (b) parallel execution time for NF, (c) parallel execution time for DF, and (d) parallel execution time for CF.

Table V. Speedups of STWMN and DSMLP schedules over baseline NMETIS.

	No. of threads = 1		No. of threads = 8	
	STWMN	DSMLP	STWMN	DSMLP
BN	1.57	2.13	1.48	1.97
NF	N/A	1.30	N/A	1.16
DF	N/A	11.15	N/A	7.77
CF	N/A	7.57	N/A	5.53

Table VI uses Cilk to profile the total work, critical path length (span), and amount of parallelism (work/span) afforded by each decomposition across all of the inputs. There are two key takeaways from these results. First, as expected, DSMLP, which incorporates domain-specific semantic and cost information, yields the best single-threaded performance (because it creates the least work), often by a significant margin.

Second, despite DSMLP focusing on work minimization for sequential execution, it still yields effective parallelism. DSMLP performance appears to be less scalable than NMETIS and STWMN because it optimizes for work, whereas NMETIS generates unnecessary work. Note that DSMLP schedule is still the fastest at higher core counts.

DOI: 10.1002/nme

Table VI. Inherent parallelism in different decompositions.

Input	Schedule	Work (s)	Critical path (s)	Parallelism
BN	METIS	2245.66	876.98	2.56
	STWMN(1.25)	1422.05	626.80	2.27
	DSMLP	1056.42	547.45	1.93
NF	METIS	803.50	402.48	2.00
	DSMLP	618.08	293.42	2.11
DF	METIS	27,223.19	13,357.13	2.04
	DSMLP	2452.20	1511.88	1.62
CF	METIS	67,251.82	26,717.44	2.52
	DSMLP	8893.80	4505.77	1.97

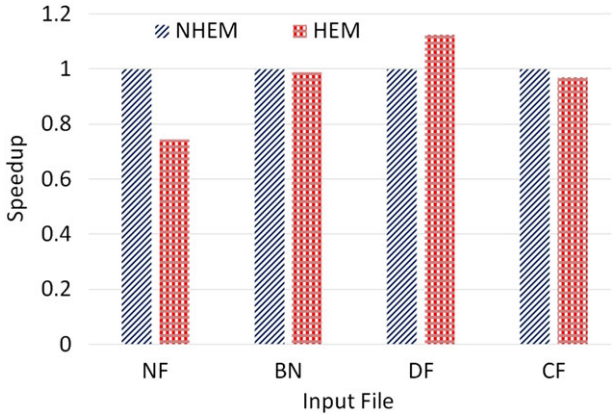


Figure 11. Speedup comparison of NHEM with respect to heavy edge matching (HEM).

5.3. *Sensitivity analysis of DSMLP*

In order to study the impact of each partitioning phase, the overall optimized decomposition, we perform a sensitivity analysis of the decomposition choices made by DSMLP. We also justify our decisions of using domain knowledge in each phase of the partitioner.

5.3.1. Coarsening phase. To analyze the sensitivity of the decomposition that is finally produced by DSMLP to the coarsening phase of the partitioning, we compare the two edge-matching schemes: (1) our NHEM and (2) HEM from METIS. Figure 11 shows the speedup of solving the decomposition produced using HEM over that produced by NHEM, computed as (runtime for NHEM)/(runtime for HEM). Note that for NF, NHEM performs better while for DF, HEM has the edge. For BN and CF meshes, both schemes produce similar results. Overall, the choice of edge-matching scheme used in the coarsening phase seems to have a moderate impact on the performance of the final decomposition produced, but not so much as to affect it by an order of magnitude.

To further explore the impact of each scheme, Tables VII–X shows the distribution of runtimes between lower timescale (SDT) and higher timescale (LDT) for the two edge-matching schemes using DSMLP. SR represents the workload SR between timescales (as shown in Figure 3), determined automatically later in the initial partitioning phase. Because NHEM segregates elements with different timescales into different subdomains, the timescale ratios (DTR) for NHEM are higher than HEM, and the number of elements at the smaller timescale (SDT) are also higher, in general. Further, because HEM does not distinguish between elements of different sizes, the subdomains in

DOI: 10.1002/nme

Table VII. Performance numbers for analysis of sensitivity to coarsening phase for the BN input.

	NHEM		HEM	
	SDT	LDT	SDT	LDT
NSUB	14	38	14	33
DTR	2		2	
CPT	389.02		390.38	
SI	1899		1968	
SR	0.45		0.5	
NEL	12,108	24,444	13,384	23,168
NEQ	10,449	22,680	11,610	20,664
LS	125.08	138.88	145.78	159.25
CP	153.29	248.24	160.72	202.53
RUN	1054.85		1058.66	

Table VIII. Performance numbers for analysis of sensitivity to coarsening phase for the NF input.

	NHEM		HEM	
	SDT	LDT	SDT	LDT
NSUB	10	31	3	27
DTR	4		3	
CPT	136.06		184.45	
SI	1308		1365	
SR	0.55		0.55	
NEL	6772	22,188	6089	22,871
NEQ	5844	20,133	4599	20,184
LS	114.44	81.58	184.87	138.76
CP	57.49	225.56	15.20	296.04
RUN	615.42		819.32	

Table IX. Performance numbers for analysis of sensitivity to coarsening phase for the DF input.

	NHEM		HEM	
	SDT	LDT	SDT	LDT
NSUB	17	65	7	49
DTR	53		34	
CPT	563.49		229.37	
SI	1491		687	
SR	0.35		0.6	
NEL	1352	41,448	895	41,905
NEQ	2286	39,711	1266	37,770
LS	69.79	318.21	52.16	499.02
CP	11.18	1476.65	12.35	1361.05
RUN	2440.52		2153.95	

DOI: 10.1002/nme

Table X. Performance numbers for analysis of sensitivity to coarsening phase for the CF input.

	NHEM		HEM	
	SDT	LDT	SDT	LDT
NSUB	36	119	26	130
DTR	61		51	
CPT	2684.15		2653.47	
SI	3564		3063	
SR	0.35		0.45	
NEL	7362	93,358	4651	96,069
NEQ	8316	86,544	5604	88,545
LS	1113.17	615.57	971.86	1095.68
CP	1051.45	3413.67	257.76	4153.92
RUN	8878.14		9132.69	

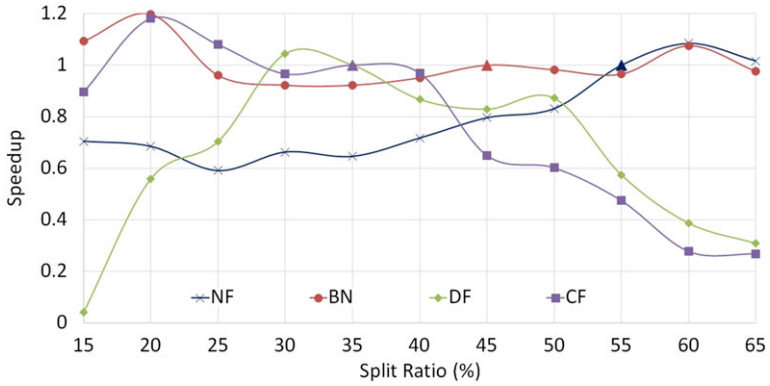


Figure 12. Performance comparison of various split ratios in initial partitioning phase.

the coarsened graph it produces have a greater mix of elements of different sizes, leading to higher SR values. Despite appreciable differences in the decompositions produced by the two approaches, these differences do not affect the final performance in Figure 11 as much because, for HEM, the next three phases of the partitioner adjust accordingly to produce an optimized decomposition.

5.3.2. Initial partitioning phase. To analyze the sensitivity of the initial partitioning phase, we split the estimated workload between timescales using different SRs. The SRs we use are 0.15, 0.2, 0.25 … 0.65. Figure 12 shows the speedup plot of various SRs for all input files. The speedup values are calculated by dividing the actual execution time taken to solve a decomposition produced by automatically selecting the SR using DSMLP by the execution time taken to solve a decomposition produced by a different SR. Triangular symbols in Figure 12 represent the execution time using DSMLP with a speedup value of 1. Although it is very difficult to determine the actual workload at initial partitioning stage because of many factors (i.e., timescale ratio and interface size between timescales), our DSMLP approach is able to determine a close to optimal SR, if not the best, for all inputs.

5.3.3. Uncoarsening/refinement phase. To determine the impact of the refinement process on the overall decomposition, using DSMLP, we compare the decompositions produced with refinement phase enabled (REF) and disabled (NREF). Figure 13 clearly shows the advantage of refinement process.

DOI: 10.1002/nme

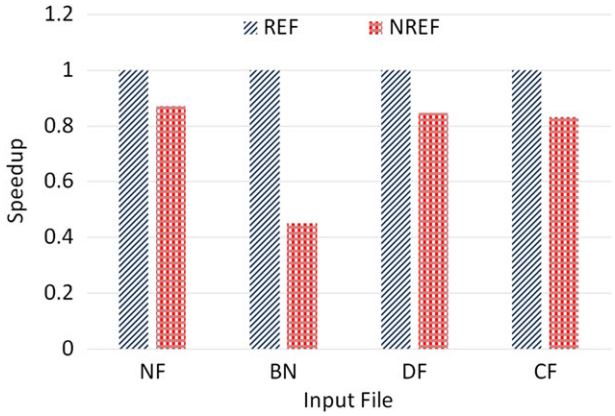


Figure 13. Performance comparison of DSMLP with and without refinement step.

Table XI. Performance numbers for analysis of sensitivity to refinement phase in the BN input.

	REF		NREF	
	SDT	LDT	SDT	LDT
NSUB	14	38	19	73
DTR	2		2	
CPT	389.02		1537.72	
SI	1899		5631	
NEL	12,108	24,444	10,301	26,251
NEQ	10,449	22,680	11,184	28,356
LS	125.08	138.88	109.94	150.40
CP	153.29	248.24	168.49	371.33
RUN	1054.85		2337.88	

Table XII. Performance numbers for analysis of sensitivity to refinement phase in the NF input.

	REF		NREF	
	SDT	LDT	SDT	LDT
NSUB	10	31	15	26
DTR	4		4	
CPT	136.06		260.45	
SI	1308		2235	
NEL	6772	22,188	6548	22,412
NEQ	5844	20,133	6618	19,464
LS	114.44	81.58	105.90	87.45
CP	57.49	225.56	82.7	167.9
RUN	615.42		704.4	

Tables XI–XIV show numbers distributed between lower timescale (SDT) and higher timescale (LDT) for decompositions with the refinement process enabled and disabled. We notice a significant drop in the size of the interface (SI) between the timescales when refinement process is enabled. This is due to shifting of elements between timescales to reduce the final coupling cost between timescales

DOI: 10.1002/nme

Table XIII. Performance numbers for analysis of sensitivity to refinement phase in the DF input.

	REF		NREF	
	SDT	LDT	SDT	LDT
NSUB	17	65	26	63
DTR	53		53	
CPT	563.49		963.19	
SI	1491		2463	
NEL	1352	41,448	1963	40,837
NEQ	2286	39,711	3450	38,949
LS	69.79	318.21	90.79	374.93
CP	11.18	1476.65	23.85	1417.41
RUN	2440.52		2870.17	

Table XIV. Performance numbers for analysis of sensitivity to refinement phase in the CF input.

	REF		NREF	
	SDT	LDT	SDT	LDT
NSUB	36	119	40	135
DTR	61		61	
CPT	2684.15		3816.02	
SI	3564		5415	
NEL	7362	93,358	8061	92,659
NEQ	8316	86,544	10,173	87,633
LS	1113.17	615.57	1922.89	690.09
CP	1051.45	3413.67	1076.12	3162.04
RUN	8878.14		10,667.16	

at the expense of individual subdomain solve cost. BN mesh has the highest drop in interface size by enabling the refinement process. This is also reflected by the number of subdomains (NSUB) at higher timescale. As the interface size is reduced, the coupling costs and the number of equations associated with each subdomain also reduce significantly. Hence, with lower overall cost, the final partitioning phase requires less number of subdomains. An interesting point to note is that for DF and CF meshes (larger inputs), by enabling the refinement process, more elements are shifted from lower to higher timescale, hence reducing the number of elements at smaller timescale. This is perfectly legal as our approach ensures numerical stability and accuracy by allowing only those elements to move that have theoretical timescale value less than the quantized higher timescale value.

5.3.4. Final partitioning phase. To study the impact that choice of number of subdomains have on overall decomposition, we decomposed the refined partitions produced by DSMLP at the end of the uncoarsening/refinement phase into different number of subdomains. The choice of number of subdomains that each input mesh was decomposed into, was less than the total number of subdomains determined by DSMLP automatically, because that is the maximum number of subdomains our approach allows based on the domain-specific cost model. Figure 14 shows normalized runtime for various decompositions. The normalized runtime for each input mesh is calculated by dividing the runtime for solving a particular decomposition by runtime for solving the decomposition produced automatically by DSMLP. We observe that decompositions with lesser number of subdomains take

DOI: 10.1002/nme

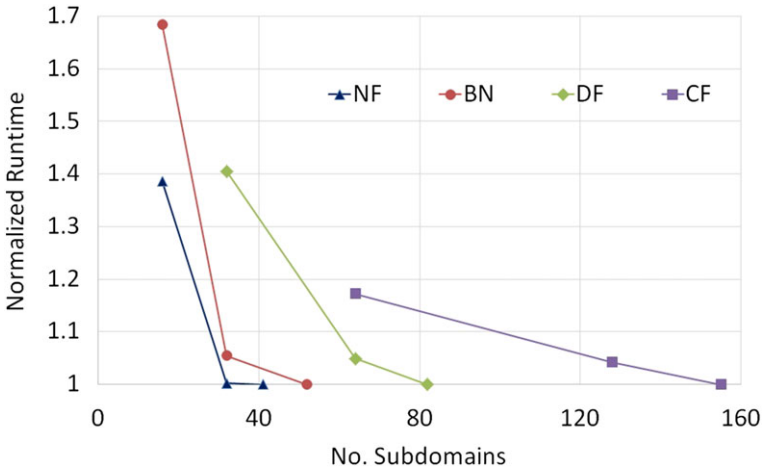


Figure 14. Performance comparison of DSMLP with varying number of subdomains.

longer to execute. Additionally, with DSMLP, we cannot further decompose the subdomains as the cost will increase. Hence, our approach determines a near-optimal number of subdomains that a mesh needs to be decomposed into.

5.4. Mesh decomposition overhead

In this section, we temper the significant performance improvements achieved by our DSMLP approach with the overhead incurred because of the partitioning process itself. We compare the cost of using DSMLP with NMETIS because only these two approaches are automated, whereas STWMN requires manual input by a domain expert, thereby making it the approach with the highest associated overhead.

For DSMLP, the mesh partitioning process, which consists of reading the input mesh, coarsening, quantizing, refining, final partitioning, and reflecting it back to the original mesh, took 254.39, 378.35, 413.65, and 1037.26 s for the NF, BN, DF, and CF meshes, respectively. In stark contrast, NMETIS takes only 0.8 s for the NF mesh to 3.8 s for the CF mesh. The primary reason that for DSMLP takes much longer to decompose the mesh as compared with NMETIS is that DSMLP does many additional work at each partitioning phase, utilizing domain knowledge and performing additional bookkeeping to come up with an almost optimal decomposition. In the coarsening phase, NMETIS just sums up the weights, whereas DSMLP first searches for common nodes and then subtracts the associated cost between merging vertices. Additionally, if an edge is marked in the maximal matching but the difference in element size is large, that edge is discarded, hence in some cases, more time is required to coarsen the graph with DSMLP than with NMETIS. Finally, METIS, the underlying driver of NMETIS, is very mature and each one of its partitioning phases has been highly optimized for high performance. We anticipate that with time, our DSMLP approach will also undergo significant refinement and be comparable in terms of overhead to NMETIS.

Table XV presents the percentage of total time taken by each phase of DSMLP. It is observed that about 90% of the time is spent in the initial partitioning phase. This is because DSMLP searches for the best SR from among 10 different workload SRs using the cost model. For each SR, the cost of the subsequent refinement process is also included, which adds even more time to the initial partitioning phase. Table XVI provides the minimum, maximum, mean, and median costs of decomposition for different SR cases.

Figure 15 shows how the times for different partitioning phases of DSMLP vary as the input size increases. The timing results are normalized to the smallest NF mesh. The increase in time for all partitioning phases and for total partitioning appears to be proportional to $n\log n$, as input size is increased. However, the time for the final partitioning phase increases at a faster rate as the input size increases because more number of subdomains are required for larger inputs, and it takes more time

DOI: 10.1002/nme

Table XV. Percentage of total partitioning time taken by each DSMLP phase.

	Coarsening	Initial partitioning	Refinement	Final partitioning
NF	1.52	92.15	5.66	0.67
BN	1.41	91.34	6.48	0.77
DF	1.48	90.17	7.09	1.26
CF	1.51	90.77	5.34	2.38

Table XVI. Per iteration timing information for initial partitioning phase in seconds.

	Minimum	Maximum	Mean	Median
NF	20.77	27.24	23.41	23.33
BN	29.00	39.66	34.52	34.78
DF	24.87	45.41	37.24	36.95
CF	73.92	108.51	94.03	94.64

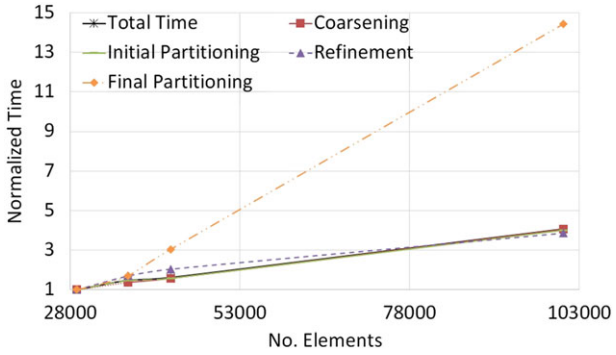


Figure 15. Normalized time to partition each mesh with increasing input size.

to determine if a subdomain needs to be split, based on cost. While it looks like the final partitioning phase does not scale well with input size, it is still a very small fraction of the overall runtime, and therefore, the overall runtime does not increase at the same rate with input size.

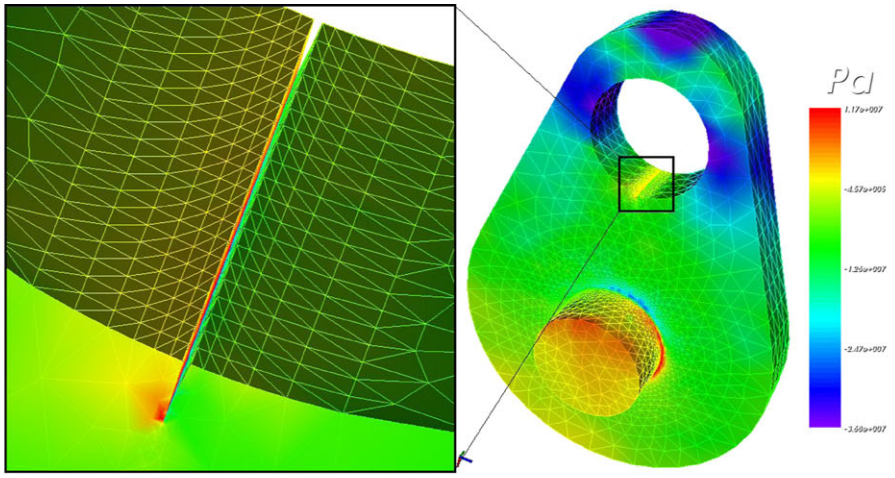
Because long-term simulations of physical models may require running thousands or even millions of time steps, the mesh decomposition overhead of using DSMLP is easily amortized over these time steps. With its overhead amortized, DSMLP more than makes up for its cost by providing almost an order of magnitude improvement in performance.

5.5. Application

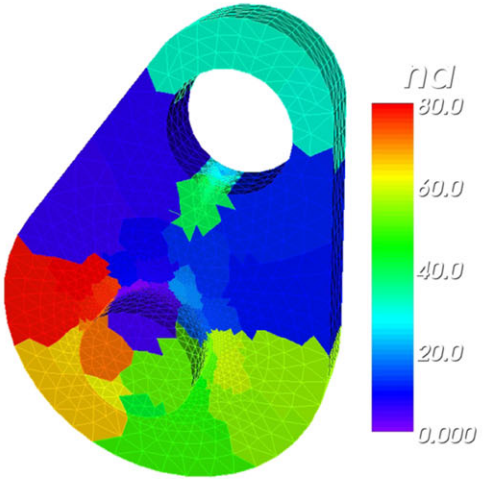
In this section, we compare the performance of DSMLP and NMETIS approaches for a realistic example. We consider a cracked cam, a common mechanical part, as shown in Figure 16(a). The mesh is highly refined in the vicinity of the crack and in the neighborhood of the circular edge where the cylindrical shaft meets the cam. The mesh has 11,503 nodes and 57,326 tetrahedral elements. The physical problem being simulated involves stress wave propagation in the cracked cam and capturing the dynamics of these stress concentrations at the crack-tip and their interactions with the stress concentrations at the circular edge of the shaft and the cam.

The problem is solved first using no mesh partitioning and with a uniformly small time step of 10^{-8} s. Displacements, velocities, accelerations, and stresses throughout the mesh are recorded as a base reference solution. Next, the problem is solved using the NMETIS and DSMLP approaches

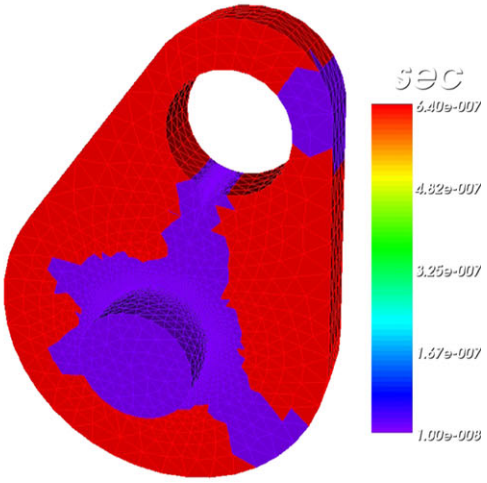
DOI: 10.1002/nme



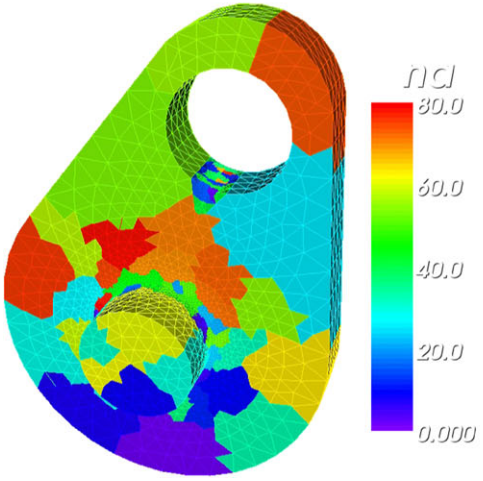
(a) Stress wave in a cracked cam



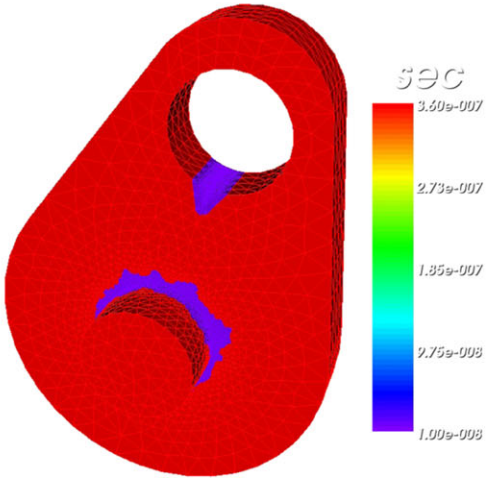
(b) NMETIS with 81 subdomains



(c) NMETIS time-step distribution



(d) DSMLP with 81 subdomains



(e) DSMLP time-step distribution

Figure 16. Comparison of the DSMLP and NMETIS approaches for wave propagation in a cracked cam. (a) Stress wave in a cracked cam, (b) NMETIS with 81 subdomains, (c) NMETIS time-step distribution, (d) DSMLP with 81 subdomains, and (e) DSMLP time-step distribution.

DOI: 10.1002/nme

Table XVII. Comparison of the perfomance of NMETISand DSMLPapproaches for a realistic application.

	Base	NMETIS	DSMLP
Small Δt (s)	10^{-8}	10^{-8}	10^{-8}
Time-step ratio $\Delta T/\Delta t$	—	64	36
Subdomains (and elements) at small Δt	1 (57,326)	68 (48,085)	42 (26,208)
Subdomains (and elements) at large ΔT	—	13 (9241)	39 (31,118)
Total runtime (s)	239,304.59	8313.7	3502.98
Error in displacements	—	0.71 %	0.77 %
Error in velocities	—	0.28 %	0.27 %
Error in accelerations	—	0.23 %	0.24 %

with automatic multi-level partitioning as described in the previous sections. Table XVII compares the decomposition parameters, runtimes, and solution errors for the two approaches. Errors in the NMETIS and DSMLP solutions are measured with respect to the base solution by taking the means of the absolute differences in displacements, velocities, and accelerations across all degrees of freedom at all time steps and normalizing them against the difference between their maximum and minimum values.

Table XVII shows that both NMETIS and DSMLP approaches lead to large reductions in runtimes in comparison with the base case while maintaining relatively small levels of error (less than 1%). Further, as shown in Figure 16, our DSMLP approach is able to correctly capture the regions of the mesh that require small time steps and therefore is able to optimizes the partitioning and timescale quantization to minimize the total runtime. Note that even though NMETIS leads to a greater time-step ratio (64) in comparison with DSMLP (36), the distribution of the subdomains and elements at the two timescales is better for DSMLP and that leads to a much smaller overall runtime.

Finally, we make some comments on the problem of dynamic performance optimization. In the context of the application presented in this paper, the problem of dynamic performance optimization can be approached in two ways: one that is driven by the physics of the problem itself, and the other that is dictated by the hardware and operating system platform at runtime. The first of these situations occurs when the multi-scale problem itself is dynamically changing. For instance, in the case of a dynamically propagating crack, the computation may become highly irregular in time. Techniques similar to adaptive mesh refinement (accompanied with adaptive time-step refinement and coarsening) can be used to address such situations. The second scenario occurs when the computational resources of the machine are shared among multiple applications, and with varying loads because of different applications, there may be a need to dynamically adapt the computation to available resources. We note that the same parallelization strategy adopted in Cilk [21] to maintain load balance in parallel execution can help account for different hardware availability: if one processor/system is overloaded and hence takes longer to complete its computation, this looks identical to load imbalance to the Cilk scheduler, and its work-stealing mechanisms can help compensate for this.

6. CONCLUSIONS

This paper presents a domain-specific multi-level mesh partitioning approach that exploits domain knowledge to automatically produce an optimized decomposition for temporal multi-scale problems in solid mechanics and similar other computational domains. Decomposing a mesh comprises assigning elements to timescales, determining an optimal total number of subdomains, choosing appropriate timescales for the subdomains, and determining the number of subdomains at each timescale. Each of these factors affects the final performance and optimizing for one often affects others negatively. We show that the number of possible mesh decompositions is exponentially large

DOI: 10.1002/nme

for meshes even with a moderate number of elements. Among the various decompositions, many do not perform well, and it is not obvious which decomposition leads to good performance.

We demonstrate for real problems that our mesh decomposition technique can generate near-optimal decomposition that can be solved up to 11 times faster than the current state-of-the-art partitioners like METIS or hand-picked decompositions by domain experts. We show that by exploiting domain knowledge and cost models, we can determine the optimal number of subdomains required per timescale, the timescale ratio, and the number of elements at each timescale, automatically. Finally, we show that our decomposition is able to deliver scalable performance on multiple cores as long as there is parallelism to be exploited.

ACKNOWLEDGEMENTS

This research is supported by the Department of Energy under contract DE-FC02-12ER26104. M. Hasan Jamal is supported by Fullbright PhD Fellowship.

REFERENCES

1. Michopoulos J, Farhat C, Fish J. Modeling and simulation of multiphysics systems. *Journal of Computing and Information Science in Engineering* 2005; **5**(3):198–213.
2. Fish J. Bridging the scales in nano engineering and science. *Journal of Nanoparticle Research* 2006; **8**(5):577–594.
3. Gravemeier V, Lenz S, Wall WA. Towards a taxonomy for multiscale methods in computational mechanics: building blocks of existing methods. *Computational Mechanics* 2008; **41**(2):279–291.
4. Toselli A, Widlund O. *Domain Decomposition Methods – Algorithms and Theory*. Springer: Berlin, Germany, 2005.
5. Fragakis Y, Papadrakakis M. The mosaic of high performance domain decomposition methods for structural mechanics: formulation, interrelation and numerical efficiency of primal and dual methods. *Computer Methods in Applied Mechanics and Engineering* 2003; **192**:3799–3830.
6. Xu J, Zou J. Some nonoverlapping domain decomposition methods. *SIAM Review* 1998; **40**:857–914.
7. Gravouil A, Combescure A, Brun M. Heterogeneous asynchronous time integrators for computational structural dynamics. *International Journal for Numerical Methods in Engineering* 2015; **102**(3–4):202–232.
8. Prakash A, Hjelmstad KD. A FETI based multi-time-step coupling method for Newmark schemes in structural dynamics. *International Journal for Numerical Methods in Engineering* 2004; **61**:2183–2204.
9. Hackbush W. On the computation of approximate eigenvalues and eigenfunctions of elliptic operators by means of a multigrid method. *SIAM Journal on Numerical Analysis* 1979; **16**:201–215.
10. Bennighof JK, Lehoucq RB. An automated multilevel substructuring method for eigenspace computation in linear elastodynamics. *SIAM Journal on Scientific Computing* 2004; **25**:2084–2106.
11. Arbenz P, Hetmaniuk UL, Lehoucq RB, Tuminaro RS. A comparison of eigensolvers for large-scale 3D modal analysis using AMG-preconditioned iterative methods. *International Journal for Numerical Methods in Engineering* 2005; **64**:204–236.
12. Saad Y, Zhang J. BILUTM: A domain-based multilevel block ILUT preconditioner for general sparse matrices. *SIAM Journal on Matrix Analysis and Applications* 1999; **21**:279–299.
13. Li Z, Saad Y, Sosonkina M. pARMS: a parallel version of the algebraic recursive multilevel solver. *Numerical Linear Algebra with Applications* 2003; **10**:485–509.
14. Prakash A, Taciroglu E, Hjelmstad KD. Computationally efficient multi-time-step method for partitioned time integration of highly nonlinear structural dynamics. *Computers & Structures* 2014; **133**:51–63.
15. Liu C, Jamal MH, Kulkarni M, Prakash A, Pai V. Exploiting domain knowledge to optimize parallel computational mechanics codes. In *Proceedings of the 27th International ACM Conference on International Conference on Supercomputing*, ICS '13. ACM: New York, NY, USA, 2013; 25–36. http://doi.acm.org/10.1145/2464996.2464998.
16. Karypis G, Kumar V. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing* 1998; **20**(1):359–392.
17. Liu P, Wang C-F. A bubble-inspired algorithm for finite element mesh partitioning. *International Journal for Numerical Methods in Engineering* 2013; **93**(7):770–794.
18. Prakash A, Hjelmstad KD. A multi-time-step coupling method for Newmark schemes in structural dynamics. In *Proceedings of Mcmat2005: Joint ASME/ASCE/SES Conference on Mechanics and Materials*, Baton Rouge, Louisiana, USA, 2005.
19. Courant R, Friedrichs K, Lewy H. Über die partiellen differenzengleichungen der mathematischen physik. *Mathematische Annalen* 1928; **100**(1):32–74.
20. Hendrickson B, Leland R. A multi-level algorithm for partitioning graphs. In *Proceedings of the IEEE/ACM SC95 Conference Supercomputing, 1995*, San Diego, California, USA, 1995; 28–28.
21. Blumofe RD, Joerg CF, Kuszmaul BC, Leiserson CE, Randall KH, Zhou Y. Cilk: An efficient multithreaded runtime system. In *Proceedings of the 5th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPOPP '95, Santa Barbara, California, USA, 1995; 207–216.

DOI: 10.1002/nme