Workload Distribution Framework for the Parallel Solution of Large Structural Models on Heterogeneous PC Clusters

Ozgur Kurc¹

Abstract: One of the main problems of substructure-based parallel solution methods is the imbalances in the condensation times of substructures when direct solvers are used. Such imbalances usually decrease the performance of the parallel solution. Thus, in this study, a workload distribution framework for such methods at heterogeneous computing environment is presented. The main idea behind this framework is to iteratively adjust the shapes of substructures so that the imbalance in their condensation times is minimized. Both generated and actual structural models were solved to illustrate the applicability and the efficiency of this approach. In these examples, a PC cluster having eight different computers was used.

DOI: 10.1061/(ASCE)CP.1943-5487.0000019

CE Database subject headings: Structural models; Substructures; Parallel processing; Load distribution.

Author keywords: Workload balancing; Partitioning; Repartitioning; Substructure; Parallel; Heterogeneous computing; PC cluster.

Introduction

Parallel computing techniques are being implemented in various structural analysis programs due to the increased size of structural models and the demand for performing more detailed and sophisticated analyses. Extensive research effort has been devoted to the parallel solution of finite-element problems over the last 20 years. Today, there are numerous parallel solution methods with iterative (Bitzarakis et al. 1997; Farhat et al. 2000) or direct solvers (Ashcraft and Grimes 1999; Amestoy et al. 2000) which may be based on global (Ashcraft et al. 1990; Heath and Raghavan 1994; Amestoy et al. 2000), element-by-element (Bitzarakis et al. 1997) or domain-by-domain (Farhat et al. 2000; Hsieh et al. 2002; Kurc and Will 2005) solution strategies. The performances of these solvers may, however, be limited depending on the type of analysis, parallel environment, and the structural properties of the system. Furthermore, many of these parallel solvers were designed for homogeneous computing environments. On the other hand, in many civil engineering design offices, the existing computer systems are usually heterogeneous clusters where each computer may have different computational characteristics. Thus, a parallel solution method which is suitable for civil engineering problems and considers the computational characteristics of such heterogeneous clusters will allow engineers to perform solutions faster without requiring the purchase of any additional hardware.

One type of parallel solution methods is the substructure-based approach (Hsieh et al. 2002; Kurc and Will 2005) which can be considered as a domain-by-domain solution strategy. In such methods, first the structure is partitioned into substructures. The number of substructures is usually taken as equal to the number of available computers. Then, each computer assembles its substructure’s stiffness matrix and force vectors and condenses the internal equations to the interfaces with other substructures, simultaneously (local solution). At this point, the solution is actually converted into the solution of dense system of equations at the substructure interfaces (interface stiffness matrix). Then, the interface stiffness matrix is assembled and solved to obtain displacement at the interface nodes (interface solution). After having computed the interface displacements, each computer calculates its substructure’s internal displacements, element forces, etc. Such methods offer several advantages since the partitioning is performed at the structural level (Hsieh et al. 2002). First of all, they allow parallelization of every step of the solution from the stiffness matrix generation to element stress and force computations. Moreover, the communication is minimized by requiring the data transfers during the interface solution only. On the other hand, one of the main challenges of such methods is the balanced distribution of computational loads (workload balancing) among computers especially when the computers’ computational speeds vary (Yang and Hsieh 2002; Kurc and Will 2007).

When direct solvers are used to condense the internal equations of substructures, none of the current partitioning approaches are capable of creating substructures having balanced condensation times (Hendrickson 2000). There are various studies (Fulton and Su 1992; Escaig et al. 1994) for homogeneous shared memory parallel computers that proposed assigning more processors to the substructures that were estimated to have more operation counts. For distributed memory parallel computers, Yang and Hsieh (2002) and Pinar and Hendrickson (2001) proposed an iterative repartitioning approach where initial partitioning was modified repeatedly until the cost function that represented the condensation time was minimized. Unfortunately, all of the studies mentioned above are for homogeneous systems.

The writer’s previous study focused on developing a parallel workload balancing framework for direct condensation of substructures at homogeneous PC clusters (Kurc and Will 2007; Kurc 2008). The framework investigated various approaches to balance the condensation times of substructures when the condensations were performed by the active-column (skyline) solvers only.
Although the performance of active-column solvers compete with the sparse solvers as the size of the substructures decreases (Farhat et al. 2000), in case of large structures the solution time and memory consumption of the active-column solvers are significantly larger (Hsieh et al. 2002). Hence, the performance of the workload balancing framework for condensations with sparse solvers is an important area that must be further investigated for the solution of large structures with small number of computers.

The current research on workload balancing at heterogeneous clusters has a more general approach which can be classified in three groups. The first group of studies focused on developing scheduling schemes (Qin and Jiang 2005) in such a way that subtasks are assigned to the computers and ordered for execution to minimize the overall execution time. The second group of studies targeted to execute the parallel applications developed for homogeneous clusters efficiently on heterogeneous clusters without rewriting them (Cuenca et al. 2005; Kishimoto and Ichikawa 2005). For that purpose, multiple processes were invoked on faster processors to balance the solution times. The last group of studies investigated new data distribution schemes which assigned more workload to the faster processors in a heterogeneous computing environment (Beaumont et al. 2001; Kalinov and Lastovetsky 2001; Bohn and Lamont 2002). Bohn and Lamant (2002) used column-wise block-stripping partitioning to distribute the elements of a matrix among processors and performed NAS Parallel Benchmark LU simulation to test the efficiency of their asymmetric load balancing strategy.

This study presents an enhanced version of the writer’s previous study (Kurc and Will 2007) which was for homogeneous computing environments only and designed for active-column solvers. The workload distribution framework presented in this paper, however, focuses on improving the performance of a substructure-based parallel solution approach at heterogeneous computing environments where the condensations are performed by sparse solvers. The main idea behind the workload distribution framework is to iteratively adjust the shapes of substructures so that each computer receives computational loads according to its computing speed. The target parallel computing environment is cheap clusters composed of ordinary PCs connected with network switches. The performance of the framework was tested on large size actual structural models which cannot be solved by a single computer without a special out-of-core solver.

The rest of the paper is organized as follows: the Workload Distribution Framework section presents the details of the workload distribution framework. First a general overview of the main steps of the framework is presented and then in-depth explanations of the important steps are given. A brief description of the substructure-based parallel solution framework is explained in General Structure section. The Cluster Recognition section focuses on the performance of the presented approach on a midsize illustrative model and two actual large-size building models. The final section summarizes important results and conclusions of this work.

Workload Distribution Framework

The workload distribution framework was designed as a separate program that prepared an input file for a substructure-based parallel solution method. The program first takes a typical structural data, i.e., nodal coordinates, element connectivity, loadings, etc. as an input and performs partitioning, workload balancing, and equation numbering which will be discussed in detail in the following sections. The framework was developed with C++ programming language and implemented under the Windows operating system. MPICH Library (2005) message passing library was used for parallelization.

Substructure-Based Parallel Solution Method

Main steps of the workload distribution framework are presented in Fig. 1. First, the computational characteristics of each computer at a heterogeneous PC cluster must be determined. In other words, the computing speeds of each computer must be determined so that their condensation times can be estimated. This step is called cluster recognition and performed once for each cluster. The rest of the computations utilize these speed values while creating the substructures for the parallel solution.

The next step is to divide the structure into substructures by using a partitioning algorithm. The objective of many existing partitioning algorithms (Hendrickson and Kolda 2000) is to minimize the communication volume while keeping the computational load of each partition balanced. The computational loads are considered balanced if the sum of the predefined weights of nodes or elements of each substructure is equal to each other. This approach works fine if the computational cost can be represented by a single weight value assigned to a node or an element. Conversely, when a substructure’s stiffness matrix is condensed with a direct solver, such weight definitions are insufficient to provide a balanced distribution of the computational load (Hendrickson 2000; Kurc and Will 2007). There are many variables that affect the condensation time which can only be known after partitioning, such as the number of internal and interface nodes, the amount and distribution of the nonzero elements of the stiffness matrix, etc. Because of this reason, if the substructures are created by using a partitioning algorithm only, it is very probable to obtain substructures with unbalanced condensation times. Such imbalances decrease the efficiency of the parallel solution because the time spent during the condensation step is governed by the substructure with the slowest condensation time.

Thus, after the structure is initially partitioned into substructures, the workload balancing step is initiated whose aim is to adjust the estimated imbalance of substructures by iteratively transferring nodes from the substructures with slower estimated condensation times to the substructures with faster estimated condensation times. By doing this, it is expected that as the condensation times of substructures approach to each other, the governing condensation time will decrease. The workload balancing step is performed in parallel by using all the computers that will also be used for the parallel solution.
The next step is the preparation of the structural data where the node and element definition of each substructure are created from the partitioning information obtained after the workload balancing step. When the nodal graph is partitioned or repartitioned, the results indicate to which substructure a node belongs, but there will be no information about the elements at the interfaces. Such elements, called the interface elements, have nodes on two or more substructures. Hence a decision must be made for determining to which substructure the corresponding interface element should be assigned. In this framework, the interface elements are assigned to one of their adjacent substructures based on two criteria: keeping the number of interface nodes of each substructure as low as possible and assigning more elements to the substructures having lower condensation time estimations.

Once the substructures are created, the shuffling step is initiated where the substructures may be assigned to computers other than their initial computers for further reducing the imbalances in the condensation times. The final step is the interface nodes ordering using the bandwidth minimization algorithm. The results are written into the file which will be used by the parallel solution program.

**Results and Discussion**

The current version of the framework was designed for the substructure-based parallel solution method where substructures’ stiffness matrices were condensed with a sparse solver. The current sparse solver computes LDLᵀ factorization of symmetric matrices in row-wise fashion whose governing operations are scalar-vector multiplication followed by a vector-vector subtraction (Stewart 2003). Moreover, only the nonzero elements of the lower-triangular part of the stiffness matrix are stored in compressed column form that requires random memory access during factorization. Thus, to estimate the condensation speed of computers, various sample structures were factorized with the current sparse solver at each computer and the average of the factorization time-operation count ratios was considered to be the condensation speed of the corresponding computer. The sample structures were square plate model composed of quadrilateral shell elements, cube model composed of solid elements and a midsize building model composed of both frame and shell elements. During the condensations, no communication is required between the computers. Data transfer is only required during the interface solution. Because of this reason, the possible differences at the communication speeds of different computers were not considered at this step.

**Initial Partitioning**

The partitioning algorithms divide a structure into desired number of substructures by using various approaches such as geometric, topological, and graph methods (Farhat 1988; Pothen et al. 1990; Jones and Plassman 1994; Karypis and Kumar 1998). The graph methods work with the graph representation of a structure which describes a structure in terms of vertices and edges with predefined weights. Each vertex can be considered as a solution point and its weight represents the computational cost of that point. An edge is used to define the interactions between the vertices. Therefore, a graph partitioning algorithm attempts to keep the sum of vertex weights balanced in each partition while keeping the number of edges at the domain interfaces as small as possible. Multilevel graph partitioning methods (Karypis and Kumar 1998) target obtaining a reasonably good solution very quickly instead of the best one by first reducing the size of the graph (coarsening) and performing partitioning on a relatively smaller graph. Although the partition quality decreases due to coarsening, the partitioning time drops considerably. In this study, the multilevel graph partitioning library METIS (Karypis and Kumar 1998) was used to create the initial substructures.

Usually three types of graph representations were considered for describing finite-element meshes: nodal, dual, and communication graphs (Venkatakrishnan et al. 1992; Hsieh et al. 1995; Topping and Ivanyi 2001). In a nodal graph, each node corresponds to a vertex in the graph. The vertices are joined with an edge if the corresponding nodes are connected by an element. Dual graphs define connectivity of the finite elements at their boundaries. Because of this reason, the dual graph definition of finite-element models consisting of one-dimensional (1D) elements is not clear since 1D elements connect to other elements through zero-dimensional nodes (Topping and Ivanyi 2001). Communication graph proposed by Venkatakrishnan et al. (1992) better described the interprocessor communication by representing the elements by vertices and joining adjacent finite elements that share a common node with an edge. In this framework, nodal graph representation was used during initial partitioning and repartitioning since it can handle structural models having both 1D and two-dimensional elements and is similar to adjacency graph used by equation ordering algorithms.

The first step of the initial partitioning step is the conversion of the structural information, i.e., element connectivity, into the nodal graph representation. Then, the initial substructures are created where the number of substructures is taken equal to the number of computers that will be used for solution. For this purpose, the unequal size k-way partitioning algorithm of METIS (Karypis and Kumar 1998) is used which allows partitioning with prescribed partition weights. The weight of each substructure for initial partitioning is determined by the following equation:

\[ W_p(j) = \frac{S_j}{\sum_{i=1}^{p} S_i} \]  

where \( W_p(j) \) = partition weight of the \( j \)th substructure; \( S_j \) = condensation speed of the \( j \)th computer; and \( p \) = number of computers or substructures. The above equation forces the partitioning algorithm to determine the total number of nodes belonging to a substructure according to the condensation speed of the corresponding computer.

**Workload Balancing**

The flowchart of the workload balancing step is presented in Fig. 2. More detailed information on the workload balancing algorithm is presented in Kurc and Will (2007). Only the major steps, modifications, and enhancements are mentioned in this section. As an initial step, the master computer prepares the nodal graph and performs initial partitioning. Then, the master computer sends the nodal graph and initial partitioning information to all other computers in the cluster. The initial partitioning information is an integer vector that describes how the structure is divided into substructures. The \( j \)th element of the initial partitioning information describes to which substructure the \( j \)th node belongs. Once the computers have finished receiving the data, the first iteration starts. All computers extract their assigned substructure’s subgraphs from the nodal graph by using the partitioning information. The subgraphs can be pictured as a nodal graph of a
Substructure with links to the adjacent substructures at interfaces. These links are represented by the interface edges that keep the adjacency information among the interface nodes.

Multistage minimum degree (MSMD) ordering algorithm (Liu 1989) was chosen to number the equations of each substructure because MSMD algorithm numbers the vertices by stages, in other words, the vertices belonging to stage $i$ are numbered before the vertices belonging stage $i+1$. This property makes the MSMD algorithm very suitable for condensations with sparse solvers since the condensation algorithms require the interface equations to be assembled after the internal equations.

Thus, after having the subgraphs prepared, each computer creates the adjacency graph for MSMD ordering. During this process, a new graph is extracted from the subgraph by ignoring the interface edges and defining the internal and interface vertices as stages 0 and 1, respectively. Then, each computer performs MSMD numbering and calculates the operation count for condensation.

Next, the master computer collects the condensation time estimation of each substructure that is calculated by dividing the operation count with the condensation speed of the corresponding computer. Then, the master computer checks whether the condensation times are balanced or the maximum number of iterations has been reached. The condensation times are considered balanced if no further change is obtained in the shape of substructures after repartitioning. If so, the iterations are finalized and the condensation times are not obtained at the end of the iterations, it is possible to decrease the governing condensation time estimations. The repartitioning is performed by using either diffusion (Hu et al. 1998) or scratch-remap (Oliker and Biswas 1998) type repartitioning algorithm of PARMETIS (Karypis et al. 2003) library which is a parallel multilevel graph partitioning and repartitioning library.

Once the repartitioning step finalizes, the new partitioning information is shared between the computers and the next iteration initiates. The computers create new subgraphs using the new partitioning information. It should be noted that no vertex migration occurs as the structure always remains the same during the iterations. Since the nodal graph of the structure was initially distributed to all computers, the only information which must be distributed to create the new substructures is the new partitioning information. As a result, the communication overhead is minimized.

### Shuffling

Throughout the workload balancing iterations, the substructures were always assigned to their initial computers. If balanced condensation times are not obtained at the end of the iterations, it is possible to obtain more balanced condensation times by assigning different computers to substructures. This way, it is further possible to decrease the governing condensation time. For this purpose, the computers and the substructures are sorted according to their condensation speeds and operation count for condensations, respectively. Then, the substructure having the maximum operation count is assigned to the fastest computer. The substructure having the second maximum operation count is assigned to the second fastest computer. This procedure is repeated for all remaining substructures.

### Substructure-Based Parallel Solution Method

The parallel solution begins by creating separate object-oriented data structures at each computer. The master computer reads the input file prepared by the data distribution framework and sends the nodal, element connectivity, and loading information of each substructure to the corresponding computer. Then, each computer creates substructure data structure in their local memory.

After the creation of data structures, the local solution initiates.

\[ W_v(j) = \frac{R_v(j)/n(j)}{\sum_{i=1}^{p} R_v(i)/n(i)} \]

In the above equation, $n(j)$ represents the number of vertices in the $j$th substructure and $W_v(j)$ represents the new vertex weight for the $j$th substructure. Eq. (3) calculates the vertex weights in such a way that the relative ratios of the sum of the vertex weights of substructures are equal to the relative ratios of the condensation time estimations.

Then, the governing condensation time of the current partitioning is compared with the fastest governing condensation time obtained up to the current iteration. If the current partitioning produces the fastest condensations, it will be chosen for solution and stored. After that, repartitioning is initiated. In this framework, repartitioning algorithms are used to modify the shapes of the substructures at the current iteration according to their new vertex weights. This way, the vertices are transferred from the substructures having slower condensation time estimations to the substructures having faster condensation time estimations. The repartitioning is performed by using either diffusion (Hu et al. 1998) or scratch-remap (Oliker and Biswas 1998) type repartitioning algorithm of PARMETIS (Karypis et al. 2003) library which is a parallel multilevel graph partitioning and repartitioning library.
Each computer assigns the local degrees of freedom to the nodes of their substructures simultaneously. The nodes of each substructure were written into the input file according to their optimized order and were placed into the database in the same order. Hence, during the local equation numbering process, each node is visited one by one and the nodes’ active degrees of freedom are numbered consecutively.

The next step is the computation and assembly of the substructure stiffness matrix and force vectors. Each computer computes and assembles the stiffness matrices of their elements simultaneously. The force vectors are constructed in the same manner. Then condensations are initiated. The condensations are performed by using a sparse solver which performs LDLT factorization of symmetric matrices in row-wise fashion (Stewart 2003). Only the nonzero elements of the lower-triangular part of the stiffness matrix are stored in the compressed column format to minimize the memory usage. Up to this point, neither communication nor synchronization among computers is required.

The interface solution does not start until all computers finalize the condensations since their results will be used during the interface solution. The first step of interface solution is the numbering and assembly of the interface stiffness matrix. The optimum ordering of the interface equations were computed and written to the input file by the first program for the variable band solver. Thus, the active degrees of freedom of the interface nodes are numbered according to that ordering. The interface displacements are computed by using the parallel variable band solver (Kurc and Will 2005; Kurc 2008) which is actually the row-wise implementation of LU decomposition method. The parallel variable band solver requires the interface stiffness matrix to be distributed to the computers in cyclic row-wise fashion. Thus, to minimize the communication and synchronization overhead during the assembly, first each computer prepares a data distribution scheme that describes which portion of the interface stiffness matrix will be stored by each computer. Then each computer prepares data buffers that involve the portion of the interface stiffness matrix that will be sent to a particular computer. Then, data transfers are initiated where the computers send and receive data in pairs simultaneously. New computer pairs are formed among the idle computers until all computers send and receive data from each other.

Once the interface solution is finalized, each computer has the displacements of their assigned rows. However, other computers require the displacements at their interface nodes to recover the internal displacements. Thus, the interface displacements are distributed to the corresponding computers as a final step of the interface solution.

The last step of the parallel solution is the recovery phase where each computer computes its internal displacements for each loading condition simultaneously. Finally, the element forces and stresses are computed and the results are written to a file for postprocessing.

**Results and Discussions**

The efficiency of the presented framework was tested on the heterogeneous PC cluster composed of eight computers with various processor speeds and computational characteristics as shown in Table 1. The condensation speed of each computer was computed using the method explained in the previous section. All computers were running Windows XP Professional and were connected with a 1-Gb network switch. The blocking send and receive speed of this network was computed as 588.24 Mb/s and this value was not influenced by the differences in computers’ hardware.

As an illustrative example, the half-disk model was solved by using the first six computers of the heterogeneous cluster and the results for every step of the data distribution framework were presented for both the initial substructures and the substructures created after the workload distribution (final substructures). The half-disk model has 28,128 brick elements with 36,773 nodes and 110,319 equations. The shape of the structure, the initial and new substructures obtained just after the workload balancing iterations were shown in Fig. 3. The new substructures were obtained by using both the scratch-remap and diffusion repartitioning algorithm. The “SSRatio” shown in Fig. 3 illustrates how balanced the condensation times of substructures are and is calculated by dividing the difference between the maximum and minimum condensation time estimation with the minimum condensation time estimation [(max-min)/min]. A smaller SSRatio indicates that the condensation times of substructures are better balanced. The edgemax value illustrates the total number of edges at the substructure interfaces and can be considered as an indication for the interface problem size.

![Fig. 3. Half-disk model and its substructures](image-url)
As indicated by high SSRatio in Fig. 3(b) (2.21), there was a significant imbalance in the condensation times of the initial substructures. For example, the condensation time of the third substructure was estimated as 26.6 s whereas the condensation of the second substructure would only consume 8.27 s. After the workload balancing step, the condensation times of the new substructures were much more balanced. In addition to this, the governing condensation times were also decreased (16.9 s with diffusion algorithm and 17.1 s with scratch-remap algorithm). The workload balancing step continued until the maximum number of iterations (10) for both cases but the fastest partitioning was obtained at the fifth and eighth iterations with scratch-remap and diffusion algorithms, respectively.

When the shapes of the substructures were examined, it was observed that some of the substructures moved to a completely different location after the workload balancing step. For example, the second substructure was on the left-hand side after the initial partitioning [Fig. 3(b)] but moved to the right-hand side after continuous repartitioning [Figs. 3(c) and (d)]. Moreover, when the workload balancing step was performed using the diffusion algorithm, two distinct structures in a single partition were obtained which was the case for the first and second substructures as shown in Fig. 3(c). Because of this reason, the size of the interface of substructures in Fig. 3(c) was much larger than the size of the interface of initial substructures as indicated by a larger edge-cut value.

As the new substructures were obtained after the workload balancing iterations, the interface elements were created at each substructure. During this process, the interface elements were assigned to one of their adjacent substructures which altered the condensation time of substructures. Table 2 presents the operation counts and estimated condensation times of the final substructures of Fig. 3(d) before and after the interface element assignment. While estimating the condensation times of each substructure, it was assumed that the substructures would be solved by their initial computers. In other words, the substructure one (SS1) was assumed to be condensed by computer one (C1), etc. The last column of Table 2 illustrates the additional nodes created at each substructure.

As can be seen from Table 2, the presented interface element assignment approach usually assigned more interface elements to the substructures having smaller operation counts. The sixth substructure received 294 additional nodes since it had the smallest operation count for condensation. Similarly, the fourth substructure received the least number of additional nodes due to having the slowest condensation time estimation. The number of additional nodes that a substructure will receive also depends on the substructure’s adjacent substructures. For example, although the first substructure had slower condensation time estimation than the second substructure, first substructure received more additional nodes because first substructure can receive interface elements from the third and sixth substructure whereas the second substructure can only receive interface elements from a single substructure, i.e., the fourth substructure. When the first substructure was numbered again with the additional nodes, the MSMD algorithm produced a better numbering and that’s why the operation count with interface elements were smaller than the operation count without interface elements. The increase in the condensation time of other substructures was remained around 10%.

After estimating the new condensation times of substructures with interface elements the shuffling step was initiated. When different computers were assigned to condense the substructures of the half-disk model, the SS ratio decreased to 0.14 and 0.18 for the ones substructures obtained with diffusion and scratch-remap algorithms, respectively.

Table 3 illustrates the operation counts, the estimated and actual condensation times of substructures before and after interface element assignment.

| Table 2. Operation Counts and Estimated Condensation Times of Substructures before and after Interface Element Assignment for Half-Disk Model |
|----------------|----------------|----------------|----------------|
| ID | Operation count | Estimated condensation time | Estimated condensation time | Additional nodes |
| SS1 | $3.958 \times 10^9$ | 14.8 | $3.833 \times 10^9$ | 14.4 | 239 |
| SS2 | $4.670 \times 10^9$ | 13.1 | $5.255 \times 10^9$ | 14.8 | 138 |
| SS3 | $4.968 \times 10^9$ | 13.9 | $5.890 \times 10^9$ | 16.4 | 171 |
| SS4 | $4.957 \times 10^9$ | 16.8 | $5.078 \times 10^9$ | 17.2 | 75 |
| SS5 | $5.076 \times 10^9$ | 14.1 | $5.339 \times 10^9$ | 14.8 | 115 |
| SS6 | $3.653 \times 10^9$ | 10.1 | $4.011 \times 10^9$ | 11.1 | 294 |

| Table 3. Operation Counts, Estimated, and Actual Condensation Times of Substructures (Half-Disk Model) |
|----------------|----------------|----------------|----------------|
| Initial substructures (a) | Final substructures (b) |
| Computer ID | Operation count | Estimated condensation time | Actual condensation time | Substructure ID | Operation count | Estimated condensation time | Actual condensation time |
| C1 | $2.778 \times 10^9$ | 10.41 | 11.85 | SS1 | $3.833 \times 10^9$ | 14.37 | 15.84 |
| C2 | $2.969 \times 10^9$ | 8.34 | 9.52 | SS4 | $5.078 \times 10^9$ | 14.26 | 15.84 |
| C4 | $2.533 \times 10^9$ | 8.59 | 10.04 | SS6 | $4.011 \times 10^9$ | 13.60 | 15.43 |
| C5 | $3.526 \times 10^9$ | 9.78 | 11.25 | SS5 | $5.339 \times 10^9$ | 14.81 | 16.64 |
| C6 | $8.331 \times 10^9$ | 23.05 | 24.87 | SS3 | $5.890 \times 10^9$ | 16.29 | 17.38 |

Note: Boldface font denotes governing condensation time.
tual condensation times of the initial and final substructures. The initial substructures were condensed by their initial computers whereas different computers condensed the substructures according to the results obtained after the shuffling step. The ID of the substructure that was condensed by each computer was presented at the fifth column.

The imbalance in the condensation times of the initial substructures can also be observed from the actual condensation times. The governing condensation time was equal to 28.8 s but when the condensations were performed with the final substructures, the time spent during the condensation step was decreased to 17.4 s. The workload distribution step, involving graph distribution and initial partitioning consumed 6.8 s. The time spent per iteration was around 0.5 s and half of this time was spent during the MSMD ordering. The condensation time estimations were faster than the actual times but they represented the relative condensation speed differences among computers. The difference between the estimated and actual times ranged between 6% and 14% for this small size illustrative model.

The influence of the workload balancing algorithm on the interface solution time is also an important aspect of the efficiency of the presented framework. Table 4 illustrates the edge-cut value, number interface equations, average bandwidth, and the interface solution time of the three cases; initial substructures, final substructures created using both the diffusion and scratch-remap algorithm.

When the repartitioning was performed with the diffusion algorithm, the size of the interface stiffness matrix, hence the interface solution time significantly increased. On the other hand, when scratch remap algorithm was used, there was a slight increase in the number of interface equations but the average bandwidth decreased significantly when compared with the initial substructures. As a result, for this specific case, the workload distribution step with scratch-remap algorithm not only reduced the governing condensation time but also the interface solution time.

As a second step, the presented workload distribution framework was tested on two actual building models. Both models are for mixed use hotel and residential buildings composed of flat plate slab systems with shear walls and columns. The Building I model as shown in Fig. 4(a), was composed of 6,641 frame and 93,150 shell elements with 87,075 nodes and 518,580 equations. The second model as shown in Fig. 4(b), was composed of 2,305 frame and 133,356 shell elements with 136,844 nodes with 817,992 equations. The solutions were performed by using the first two, four, six, and eight computers of the PC cluster and the condensation and total solution times of the initial substructures and the substructures created using both the diffusion and scratch-remap algorithm were compared.

The condensation and total solution times and the SSRatio of all building models are presented in Fig. 5 and Table 5, respectively. When the parallel solutions were performed for Building I and Building II models with two and four computers, the available in-core memory was exceeded and Windows virtual memory was used. Because of this reason, the condensation time results for two and four computers presented in Fig. 5 were the estimates to provide a better comparison. The condensation time estimations were calculated by dividing the operation count for condensation with the speed of the corresponding computer. The total solution times involve the overhead due to the utilization of virtual memory.

As illustrated in Table 5, there were significant imbalances in the condensation times of initial substructures. The best case for Building I model was the solution with two computers where the SSRatio was equal to 2.47. After performing the workload distribution steps using both the scratch-remap and diffusion

<table>
<thead>
<tr>
<th>Case</th>
<th>Edge-cut</th>
<th>Number of equations</th>
<th>Average bandwidth</th>
<th>Assembly time (s)</th>
<th>Interface solution time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial</td>
<td>6,643</td>
<td>2,919</td>
<td>1,186.4</td>
<td>0.39</td>
<td>7.13</td>
</tr>
<tr>
<td>Final (diffusion)</td>
<td>9,393</td>
<td>4,302</td>
<td>1,524.4</td>
<td>0.85</td>
<td>14.85</td>
</tr>
<tr>
<td>Final (scratch-remap)</td>
<td>6,751</td>
<td>3,063</td>
<td>885.3</td>
<td>0.45</td>
<td>3.40</td>
</tr>
</tbody>
</table>

Fig. 4. Example structural models
algorithms, substructures had much more balanced condensation times for all cases. For most of the cases scratch-remap algorithm produced more balanced substructures than the diffusion algorithm.

The workload distribution step with scratch-remap algorithm required around 20 s for Buildings I and II models for all cases and most of this time was spent during equation numbering and repartitioning. As the number of computers increased, the time spent for equation numbering decreased but this time repartitioning step consumed more time due to increased communication and complexity. For all solution cases, the reduction obtained in the governing condensation times was more than the time spent for workload balancing iterations. For both Buildings I and II models, the condensation time estimations represented the actual condensation times within 5% difference for most of the cases for solutions with six and eight computers.

When the governing condensation times of the initial and final substructures were compared (Fig. 5), a considerable reduction

![Condensation Times Building I](image1)

![Total Solution Times Building I](image2)

![Condensation Times Building II](image3)

![Total Solution Times Building II](image4)

**Fig. 5.** Condensation and total solution times of building models

<table>
<thead>
<tr>
<th>Number of computers</th>
<th>Building I</th>
<th></th>
<th>Building II</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Initial</td>
<td>Final (diffusion)</td>
<td>Final (remap)</td>
<td>Initial</td>
</tr>
<tr>
<td>2</td>
<td>2.47</td>
<td>0.08</td>
<td>0.22</td>
<td>0.89</td>
</tr>
<tr>
<td>4</td>
<td>4.14</td>
<td>0.29</td>
<td>0.19</td>
<td>1.98</td>
</tr>
<tr>
<td>6</td>
<td>4.26</td>
<td>2.73</td>
<td>0.51</td>
<td>2.46</td>
</tr>
<tr>
<td>8</td>
<td>3.04</td>
<td>2.57</td>
<td>1.12</td>
<td>2.65</td>
</tr>
</tbody>
</table>

**Table 5.** SSRatio Values of the Building Models
was observed for solutions with two, four, and six computers. For example, the SSRatio of the initial substructures of the Building I model was equal to 4.14 for the solution with four computers. After the workload distribution step, the SSRatio became equal to 0.19 with the scratch-remap algorithm which almost halved the governing condensation time. The reduction at the eight computer solution was, however, insignificant. For almost every case, the scratch-remap algorithm produced not only more balanced but also faster condensation times than the diffusion algorithm. A similar trend was also noticed in the total solution times; the final substructures were faster than the initial ones for most of the cases.

When two computers were used for the solution, there was a significant imbalance in the condensation times of substructures. In Building I model, one of the substructures of the initial partitioning had to utilize the virtual memory during condensation and required 800.2 s. The condensation of the other substructure required only 168.9 s. As the condensation times were balanced by the workload balancing iterations with scratch-remap algorithm, the amount of memory required during condensation was reduced and the condensations of the two substructures were finalized in 447.9 and 431.3 s, respectively. In a similar manner, as four computers were used, one of the substructures of the initial partitioning required virtual memory and spent 410.7 s during condensation. As the condensation times are balanced, all condensations were performed without the need of virtual memory and the slowest one was finalized in 167.1 s.

The total solution times of the final substructures with eight computers did not result in any considerable speed-up, even slower than the solution with initial substructures for some cases. This situation was due to having larger interface problem size at final substructures. Since the reduction in the governing condensation time considering the additional time spent during workload balancing iterations was less than the additional time spent during the interface solution, no improvement was observed in the total solution time.

**Conclusions**

In this study, a workload distribution framework for a substructure-based parallel solution method at heterogeneous computing environments was presented. When a structure is partitioned into substructures using a partitioning algorithm only, a significant imbalance was observed in the condensation times of the substructures. Such imbalances considerably reduced the performance of the parallel solution. As the condensation time imbalance of substructures was diffused by iteratively modifying the shapes of substructures according to the computational characteristics of each computer in the cluster, a considerable speed-up in the parallel solution times was obtained. The speed-up, however, was very limited and even worse than the unbalanced case when eight computers were used to solve the structural models considered herein. This situation was due to obtaining larger interface problem size after workload balancing and additional time spent during iterations.

Two different repartitioning algorithms were used during the workload balancing iterations; diffusion and scratch-remap. The shape of the substructures differed according to the type of the repartitioning algorithm. Both algorithms created substructures having more balanced condensation times and for most of the cases, the governing condensation time also decreased. Scratch-remap algorithm, on the other hand, created substructures having faster and more balanced condensation times and produced very close or faster total solution times when compared to the diffusion algorithm. Thus, the scratch-remap algorithm is a better choice for the workload balancing iterations.

As a result, the presented workload distribution approach can be used to improve the performance of a substructure-based parallel solution methods especially when few computers were used for the solution. This way, large size structural models can be solved on cheap and ordinary PC clusters of any type.

**References**


