COBEM2015

ABCM Associação Brasileira de Engenharia e Ciências Mecânicas

23rd ABCM International Congress of Mechanical Engineering
December 6-11, 2015, Rio de Janeiro, RJ, Brazil

A SCALABLE IMPLEMENTATION OF THE GROUND STRUCTURE METHOD FOR SOLVING LARGE SCALE TOPOLOGY OPTIMIZATION PROBLEMS IN GPU

Arturo Elí Cubas Rodriguez
Ivan Fabio Mota de Menezes
Pontifical Catholic University of Rio de Janeiro
acubasr@yahoo.com, ivan@puc-rio.br

Abstract. *Topology optimization aims to find the most efficient distribution of material in a specified domain without violating user-defined design constraints. When applied to continuum structures, topology optimization is usually performed by means of well-known density methods. In this study, we focus on the application of its discrete formulation in which a given domain is discretized into a ground structure, i.e., a finite spatial distribution of nodes that are connected using truss members. The ground structure method provides an approximation to optimal Michell-type structures, which are composed of an infinite number of members, using a reduced number of truss members. The optimal least-weight truss with a single load under linear elastic conditions and subjected to stress constraints can be posed as a linear programming problem. The aim of this work is to provide a scalable implementation for the optimization of least-weight trusses embedded in a domain with any type of geometry. The method removes unnecessary members from a truss that has a user-defined degree of connectivity while keeping the locations of nodes fixed. We detail the scalable implementation of the ground structure method using an efficient and robust interior point algorithm in a parallel computing environment that involves graphics processing units (GPUs). The capabilities of the proposed implementation are illustrated by means of large-scale applications to practical problems with millions of members in both 2D and 3D structures.*

Keywords: *topology optimization, interior point algorithm, ground structure method, parallel computing, GPU*

1. INTRODUCTION

Worldwide trends toward reduced costs and time in structural designs are significant forces that push research toward designs that are better in terms of cost and performance. The emerging tool to help manual and computer-based designs meet these needs is topology optimization, which is a mathematical technique that finds an optimal layout in terms of weight, volume or another objective quantity for a given domain, set of boundary conditions and set of loads. The availability of powerful, cheap computers and mature mathematical algorithms has paved the way for the development of software for topology optimization in research and industrial fields. Moreover, graphical processor units (GPUs) are reshaping the way that numerically intensive applications are coded due to their high numerical throughput. In this paper, we propose a scalable implementation of the ground structure method using an efficient and robust interior point algorithm for solving large scale topology optimization problems in a parallel computing environment.

2. RELATED WORK AND PROBLEM STATEMENT

In the mid-60s, Dorn *et al.* (1964) developed the ground structure method, which consists of a discrete representation of a given domain by means of bars connecting a predefined set of nodes. The areas of the bars are the design variables. Not all of the bars are necessary, and some of them can be removed during the optimization process, but this cannot be predicted in advance. This is illustrated in Fig. 1(a) and (f). Schmit (1960) used mathematical programming techniques to solve the nonlinear-inequality-constrained problem of designing elastic structures under multiple loading conditions (Kirsch (1993)). Michell (1904) derived the conditions for an optimal least-volume truss subject to static loads. If $\sigma_T > 0$ and $\sigma_C > 0$ are the traction and compressive stress limits, respectively, for each bar, and $\sigma_0 = (\sigma_T + \sigma_C)/2$, with a positive constant, ϵ_0, then the Michell conditions for a minimum volume are:

1. The resulting truss satisfies the specified static loads;
2. Each bar has a stress equal to σ_T or $-\sigma_C$;
3. There exists a deformation compatible with the strains that is equal to $\sigma_0\epsilon_0/\sigma_T$ for members in tension, and

$\sigma_0\epsilon_0/\sigma_C$ for those in compression.

Optimized trusses obtained in this way are called Michell trusses. Ideal Michell trusses encompass an infinite number of bars, and some analytical results were derived by Lewiński and Rozvany (2007). There are at least two formulations for optimizing the layout. The elastic formulation uses the truss stiffness matrix explicitly, and the displacements must satisfy the compatibility equations. In the plastic formulation, the forces must be statically admissible, and the resulting problem can be solved using linear optimization algorithms. In this study, the plastic formulation is adopted.

2.1 The Plastic Formulation

A minimum-volume truss with stress constraints and a single load condition can be expressed as:

$$\begin{aligned} \min_{\boldsymbol{a}} \quad & V = \boldsymbol{l}^T \boldsymbol{a} \\ \text{s.t.} \quad & \boldsymbol{B}\boldsymbol{n} = \boldsymbol{f} \\ & -\sigma_C a_i \leq n_i \leq \sigma_T a_i \qquad i = 1, 2 \ldots N_b, \end{aligned} \tag{1}$$

where $\boldsymbol{l}$ is the $N_b \times 1$ vector of bar lengths, $\boldsymbol{a}$ is the $N_b \times 1$ vector of bar areas, $\boldsymbol{B}$ is the $N_{dof} \times N_b$ geometrical matrix created from the bar direction cosines, and $\boldsymbol{n}$ is the $N_b \times 1$ vector of bar internal forces. Through a series of transformations, the original formulation can be rewritten as the following linear programming problem (Zegard (2014)):

$$\begin{aligned} \min_{\boldsymbol{s}^+, \boldsymbol{s}^-} \quad & V = \begin{bmatrix} \frac{\boldsymbol{l}^T}{\sigma_T} & \frac{\boldsymbol{l}^T}{\sigma_C} \end{bmatrix} \begin{bmatrix} \boldsymbol{s}^+ \\ \boldsymbol{s}^- \end{bmatrix} \\ \text{s.t.} \quad & \begin{bmatrix} \boldsymbol{B} & -\boldsymbol{B} \end{bmatrix} \begin{bmatrix} \boldsymbol{s}^+ \\ \boldsymbol{s}^- \end{bmatrix} = \boldsymbol{f} \\ & s_i^+, s_i^- \geq 0 \, . \end{aligned} \tag{2}$$

where $\boldsymbol{n} = \boldsymbol{s}^+ - \boldsymbol{s}^-$. This formulation is used in this paper and includes $2N_b$ variables and N_b restrictions. Although the number of variables has doubled, the problem can be efficiently solved using appropriate linear programming algorithms.

2.2 The Ground Structure Method

The ground structure method provides a way to generate the truss mesh by means of the following steps: (i) a domain is defined; (ii) a node map is defined inside the domain; (iii) the nodes are connected by bars (connections can be made between pairs of nodes according to a connectivity level, see Fig. 1(b) to Fig. 1(e) or by connecting every pair of nodes in the domain); (iv) the area of each bar is modified iteratively using an optimization algorithm until it reaches an optimal structure, as shown in Fig. 1(f).

3. THE INTERIOR POINT METHOD (IPM)

The linear programming problem can be expressed in its primal or dual form, as shown in Eq. (3) and Eq. (4) respectively:

$$\min_{\boldsymbol{x}} \quad \boldsymbol{c}^T \boldsymbol{x}, \quad \text{s.t.} \quad \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \quad \boldsymbol{x} \geq \boldsymbol{0} \, , \tag{3}$$

$$\max_{\boldsymbol{y}} \quad \boldsymbol{b}^T \boldsymbol{y}, \quad \text{s.t.} \quad \boldsymbol{A}^T \boldsymbol{y} + \boldsymbol{z} = \boldsymbol{c} \quad \boldsymbol{z} \geq \boldsymbol{0} \, , \tag{4}$$

where $\boldsymbol{c}$, $\boldsymbol{x} \in \mathbb{R}^n$, $\boldsymbol{b} \in \mathbb{R}^m$, $\boldsymbol{A}$ is an $m \times n$ matrix with full row rank, $\boldsymbol{y} \in \mathbb{R}^m$, and $\boldsymbol{z} \in \mathbb{R}^n$.

A class of linear programming algorithms, known as interior point methods (IPM), was presented by Karmarkar (1984). He proposed an algorithm that starts at a given initial point $\boldsymbol{x}_0$ and reaches the optimal solution $\boldsymbol{x}^*$ through a series of iterations inside the feasible region, which gave the method the name "interior." These iterations computes steps $\triangle\boldsymbol{x}$, $\triangle\boldsymbol{y}$, and $\triangle\boldsymbol{z}$ by solving a linear system of equations. This process is shown in Fig. 2 for the variable $\boldsymbol{x}$. There are some variants of the IPM. We use the long-step variant because of its ease of implementation and parallelization. Algorithm 1 shows the details.

3.1 Using the IPM in Truss Optimization Problems

The most computationally demanding part of Algorithm 1 is step 5. This step can be completed by solving $\boldsymbol{A}\boldsymbol{D}_{xz}^2\boldsymbol{A}^T\triangle\boldsymbol{y}^k = \boldsymbol{r}_y$ for $\triangle\boldsymbol{y}^k$ and making simple substitutions to find $\triangle\boldsymbol{x}^k$ and $\triangle\boldsymbol{z}^k$. The following assignments are made: $\boldsymbol{A} =$

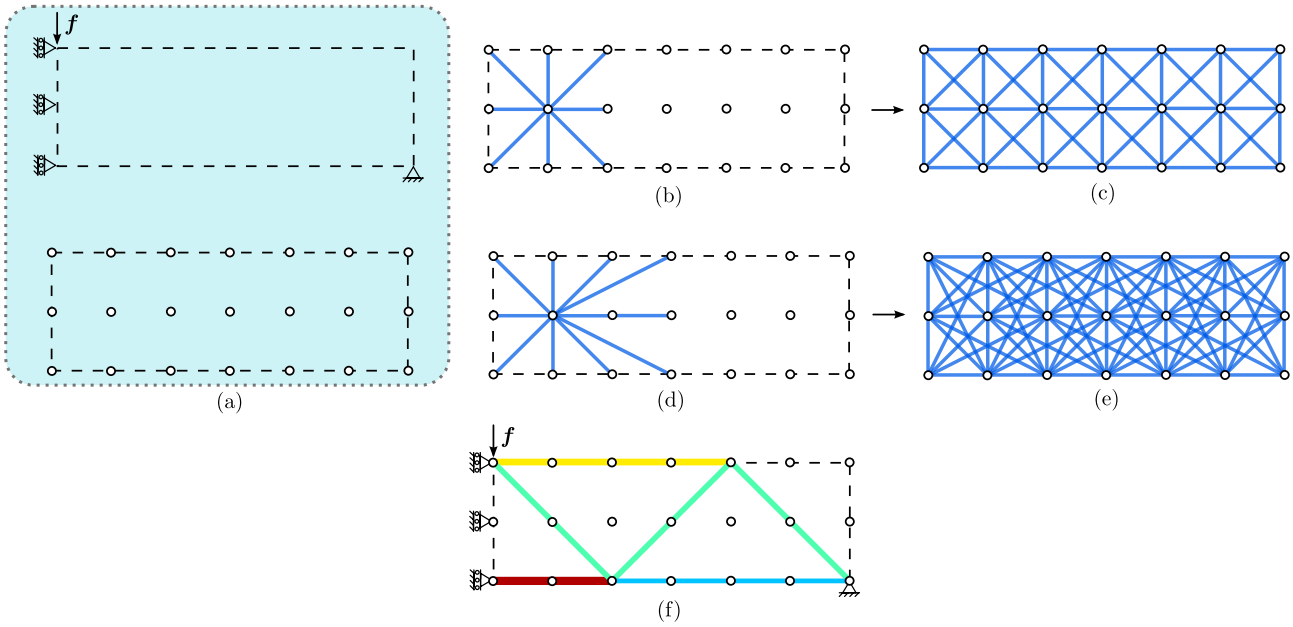

Figure 1. Ground structure generation for a structured base mesh and truss optimization sequence: (a) domain and a generated set of nodes; (b) node with connectivity level 1; (c) domain with connectivity level 1; (d) node with connectivity level 2; (c) domain with connectivity level 2; (f) optimized truss.

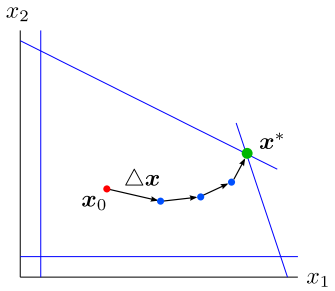

Figure 2. The sequence of the interior point method.

$\begin{bmatrix} \boldsymbol{B} & -\boldsymbol{B} \end{bmatrix}$, $\boldsymbol{b} = \boldsymbol{f}$, and $\boldsymbol{c} = \begin{bmatrix} l/\sigma_T & l/\sigma_C \end{bmatrix}^T$. Then, the matrix $\boldsymbol{A}\boldsymbol{D}_{xz}^2\boldsymbol{A}^T$ becomes $\boldsymbol{M} = \boldsymbol{B}\boldsymbol{D}_{ipm}\boldsymbol{B}^T$, where $\boldsymbol{D}_{ipm}$ is the result of partitioning $\boldsymbol{D}_{xz}^2$ into four equally-sized matrices and adding their diagonals. It is notable that $\boldsymbol{B}$ is sparse and constant, while $\boldsymbol{D}_{ipm}$ changes in each iteration of the IPM. As the iterations proceed, this matrix presents both small and large numbers and $\boldsymbol{B}\boldsymbol{D}_{ipm}\boldsymbol{B}^T$ can become ill-conditioned. This causes numerical problems for direct and iterative solvers but affects iterative solvers are more strongly. When iterative solvers are used it is not necessary to explicitly assemble $\boldsymbol{B}\boldsymbol{D}_{ipm}\boldsymbol{B}^T$ because such solvers rely on matrix vector products of the form $\boldsymbol{B}\boldsymbol{D}_{ipm}\boldsymbol{B}^T\boldsymbol{v}$, where $\boldsymbol{v}$ is a given vector.

Each iteration of the IPM involves the solution of a linear system of equations with a different $\boldsymbol{D}_{ipm}$ matrix and, consequently, a different $\boldsymbol{M}$. Therefore, the single bottleneck of the IPM is solving the linear system $\boldsymbol{M}\triangle\boldsymbol{y} = \boldsymbol{r}_y$. In this study, we use the well-known iterative PCG (preconditioned conjugate gradient) solver, which has a time complexity of $\mathcal{O}(nnz\sqrt{\kappa})$, where nnz is the number of non-zero entries in $\boldsymbol{M}$ and κ is its condition number. For trusses, $nnz = DIM^2(2N_b + N_n)$, where DIM is the dimension of the problem (2 or 3), N_b is the number of bars and N_n is the number of truss nodes. Usually $N_b \ggg N_n$; therefore, the PCG solvers time complexity for trusses is $\mathcal{O}(N_b\sqrt{\kappa})$. This reveals that using parallel computing is an affordable way to solve larger problems and perform better preconditioning to overcome the ever-increasing values of κ as the IPM iterates.

4. THE PROPOSED APPROACH

4.1 Outline

Zegard (2014) developed GRAND (ground structure analysis and design), which is a tool for optimizing least–weight trusses embedded in non-Cartesian unstructured domains. It generates and optimizes truss meshes using MATLAB®. We present GRAND++, which extends GRAND by solving the optimization problem in C++ and CUDA for faster optimiza-

Algorithm 1 Long-Step IPM

1: **Initialization**
2: Given $\sigma \in [0,1]$ and $\left(\boldsymbol{x}^0, \boldsymbol{y}^0, \boldsymbol{z}^0\right)$
3: **for** $k = 0, 1, 2, \ldots$ **do**
4: Set $\mu_k = (\boldsymbol{x}^k)^T \boldsymbol{z}^k / n$
5: Solve to obtain $\left(\triangle \boldsymbol{x}^k, \triangle \boldsymbol{y}^k, \triangle \boldsymbol{z}^k\right)$

$$\begin{bmatrix} \boldsymbol{0} & \boldsymbol{A}^T & \boldsymbol{I} \\ \boldsymbol{A} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{Z} & \boldsymbol{0} & \boldsymbol{X} \end{bmatrix} \begin{bmatrix} \triangle \boldsymbol{x}^k \\ \triangle \boldsymbol{y}^k \\ \triangle \boldsymbol{z}^k \end{bmatrix} = \begin{bmatrix} -\boldsymbol{r}_{\boldsymbol{c}}^{\boldsymbol{k}} \\ -\boldsymbol{r}_{\boldsymbol{b}}^{\boldsymbol{k}} \\ -\boldsymbol{X}\boldsymbol{Z}\boldsymbol{e} + \sigma\mu\boldsymbol{e} \end{bmatrix} \tag{5}$$

6: Choose α_k^{pri} and α_k^{dual} such that

$$\begin{aligned} \alpha_{k,max}^{pri} &= \min_{\mathrm{i}:\triangle \boldsymbol{x}_{\mathrm{i}}^{\mathrm{k}} < \boldsymbol{0}} -\frac{\boldsymbol{x}_{\mathrm{i}}^{\mathrm{k}}}{\triangle \boldsymbol{x}_{\mathrm{i}}^{\mathrm{k}}}, \quad \alpha_{\mathrm{k,max}}^{\mathrm{dual}} = \min_{\mathrm{i}:\triangle \boldsymbol{z}_{\mathrm{i}}^{\mathrm{k}} < \boldsymbol{0}} -\frac{\boldsymbol{z}_{\mathrm{i}}^{\mathrm{k}}}{\triangle \boldsymbol{z}_{\mathrm{i}}^{\mathrm{k}}} \\ \alpha_k^{pri} &= \min(1, \eta_{\mathrm{k}} \alpha_{\mathrm{k,max}}^{\mathrm{pri}}), \quad \alpha_{\mathrm{k}}^{\mathrm{dual}} = \min(1, \eta_{\mathrm{k}} \alpha_{\mathrm{k,max}}^{\mathrm{dual}}) \end{aligned} \tag{6}$$

7: Set

$$\begin{aligned} \boldsymbol{x}^{k+1} &= \boldsymbol{x}^k + \alpha_k^{pri} \triangle \boldsymbol{x}^k \\ \left(\boldsymbol{y}^{k+1}, \boldsymbol{z}^{k+1}\right) &= \left(\boldsymbol{y}^k, \boldsymbol{z}^k\right) + \alpha_k^{dual} \left(\triangle \boldsymbol{y}^k, \triangle \boldsymbol{z}^k\right) \end{aligned} \tag{7}$$

8: **end for**

tion and better memory management, allowing the number of bars to be increased. Figure 3 shows the general diagram; the blue boxes represent the new code, and the yellow ones represent the original GRAND code. We implemented *iterative* solvers because they require less memory, which is an essential feature for optimization in large domains. We used the PCG algorithm with the Jacobi preconditioner. The PCG algorithm contains an expensive computational step, which is a sparse matrix vector multiplication (SpMV): $\boldsymbol{q} = \boldsymbol{M}\boldsymbol{p}$. We implemented two variants that differ from each other in the way the matrix-vector product is calculated. The variants are the so called *element by element* PCG solver and the *blocked diagonal* solver; both were implemented for CPUs and GPUs for the purpose of comparison.

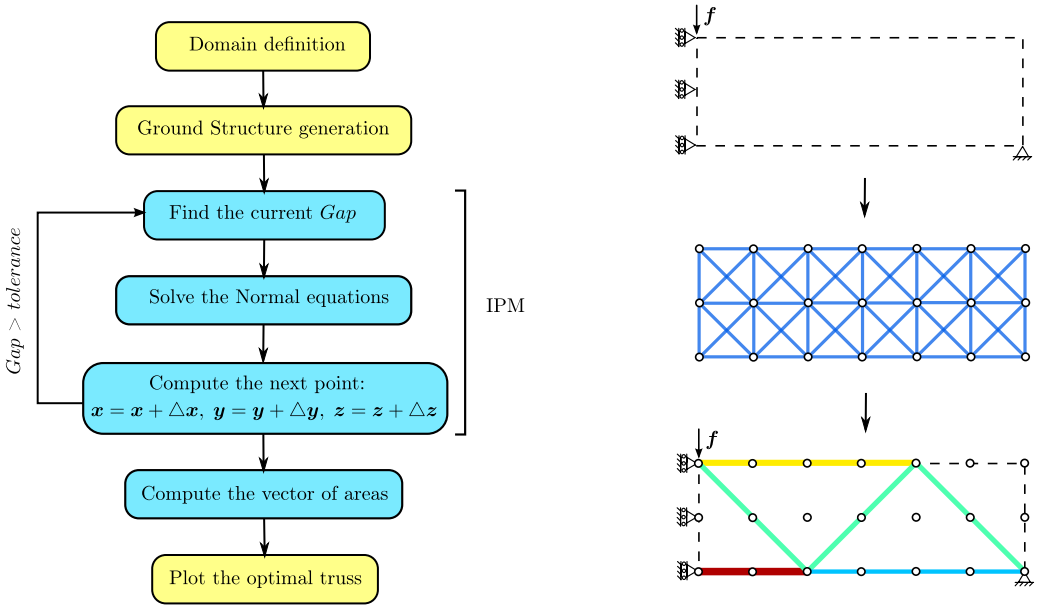

Figure 3. A truss topology optimization diagram.

4.2 Element by Element PCG

4.2.1 CPU version

The **Element by Element PCG in CPU (EbeCPU)** has two key features:

1. To reduce the amount of memory required, the complete matrix $\boldsymbol{M}$ is not assembled. Instead, the matrix-vector product is evaluated as the following sum of products *per bar*:

$$\boldsymbol{q} = \sum_{r=1}^{Nb} \boldsymbol{M}_r \boldsymbol{p} \; ; \tag{8}$$

2. To reduce the time required to solve the system, parallel computing is used to calculate matrix-vector products efficiently.

The SpMV operation is performed using Eq. 8; each CPU thread takes the $\boldsymbol{M}_r$ and $\boldsymbol{p}$ data and updates the vector $\boldsymbol{q}$ in the corresponding degrees of freedom for each bar. The $\boldsymbol{q}_r = \boldsymbol{M}_r \boldsymbol{p}$ operation for each bar requires:

1. 2 elements (i,j) of matrix $\boldsymbol{BARS}$ to define the nodes;
2. 4 elements of $\boldsymbol{p}$ (6 for a 3D problem) based on (i,j);
3. 2 direction cosines (3 for a 3D problem)
4. a single D_{ipm} value that corresponds to the bar
5. 4 elements of $\boldsymbol{q}$ (6 for a 3D problem).

Race condition. If two bars share a node, they will have some non-zeros elements in the same positions in $\boldsymbol{M}$. This means that different threads will attempt to access the same memory locations in vector $\boldsymbol{q}$, which will result in undefined behavior. This is called a *race condition* and should be avoided to ensure numerical accuracy. One way to overcome this problem is to use *coloring*, (Gebremedhin *et al.* (2005)). A coloring algorithm creates groups of elements that do not share any node, allowing threads to process the bars with the same color at the same time. Once the bars of one color have been processed, the bars of the next color are processed, and so on. A colored 2D truss is shown in Fig. 4.

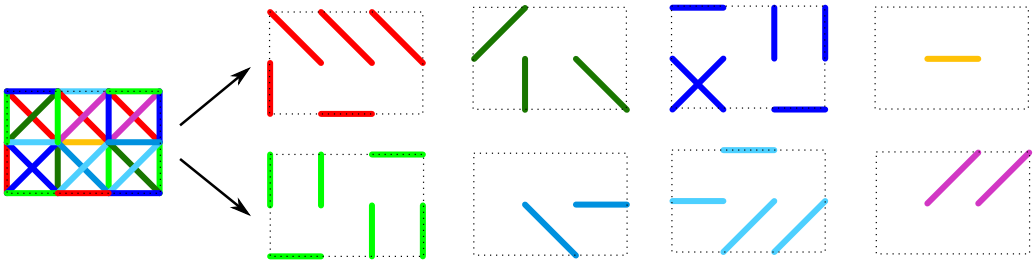

Figure 4. Bar coloring and groups by color.

4.2.2 GPU Versions

The number of mathematical operations for large meshes is enormous, and, to increase its computational efficiency, the linear system solver was implemented on a GPU. The PCG algorithm was ported to the GPU using CUDA programming, and two versions were developed.

GPU By Element (EbeGPU). The bars are colored and reordered to avoid attempting to access invalid memory addresses. Moreover, the bar vectors $\boldsymbol{i}$,$\boldsymbol{j}$,$\boldsymbol{t}_x$,$\boldsymbol{t}_y$ (and $\boldsymbol{t}_z$ in 3D problems) are *padded*, which means that dummy values are appended to ensure an appropriate length and to prevent addressing invalid memory locations by making the vector length a multiple of the GPUs block size. For colored bars, each bar vector is padded with 0s or -1s to ensure that every group of vectors of one color has a number of elements that is a multiple of a constant called `PCGTHREADS`, as shown in Fig. 5.

The element by element approach has the disadvantage that requires non-coalesced memory access to vectors $\boldsymbol{p}$ and $\boldsymbol{q}$ due to the varying indexes i and j. The next solver attempts to overcome this problem.

Blocked Diagonal in GPU (BdiaGPU). The diagonal format (DIA) is an efficient way to represent a matrix if it is sparse and its elements are grouped along its diagonals. Figure 6 shows the DIA matrix data and offsets for a matrix $\boldsymbol{A}$. The diagonals of $\boldsymbol{A}$ are stored as matrix `data` with elements that may be padded marked by $*$. The vector `offsets` stores the diagonal offset values. The main diagonal has an offset of zero, the diagonals below the main diagonal have negative offsets and the upper diagonals have positive offsets. This format is efficient not only for storing a matrix that has values grouped along its diagonals but also for accessing the elements in memory. The latter is essential for GPU programming.

The structure of matrix $\boldsymbol{M}$, associated with the truss in Fig. 4, is shown in Fig. 7(a), where each blue box represents 2×2 entries (3×3 for 3D problems). As in the element by element approach, instead of storing these four terms for each bar, we store only their direction cosines. Therefore, we save memory space, especially for 3D problems. In addition,

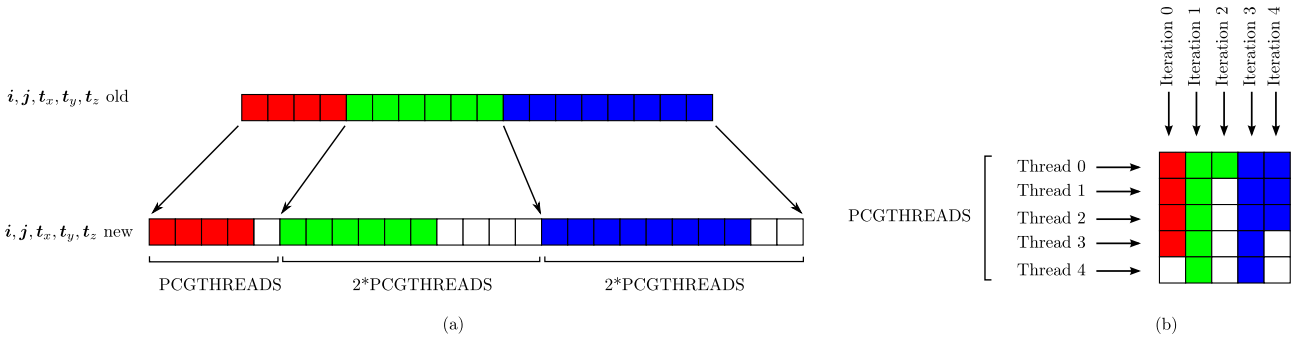

Figure 5. The kernel launching parameters. (a) Padding for a colored bar. (b) The data access pattern of the GPU.

$$\boldsymbol{A} = \begin{bmatrix} 2 & 5 & 0 & 0 \\ 0 & 4 & 9 & 0 \\ 8 & 0 & 1 & 7 \\ 0 & 9 & 0 & 4 \end{bmatrix} \quad \text{data} = \begin{bmatrix} * & 2 & 5 \\ * & 4 & 9 \\ 8 & 1 & 7 \\ 9 & 4 & * \end{bmatrix} \quad \text{offsets} = \begin{bmatrix} -2 & 0 & 1 \end{bmatrix}$$

Figure 6. The DIA format representation of matrix $\boldsymbol{A}$.

due to symmetry, we need to store only the upper triangular matrix and the main diagonal. The steps of the SpMV are as follows: first diagonal 1, shown in Fig. 7 (b), is read and multiplied by vector $\boldsymbol{p}$ in parallel, as shown in Fig. 7(c), and then the results are accumulated in vector $\boldsymbol{q}$. We repeat this process for the other diagonals (2 to 9) until the product is complete.

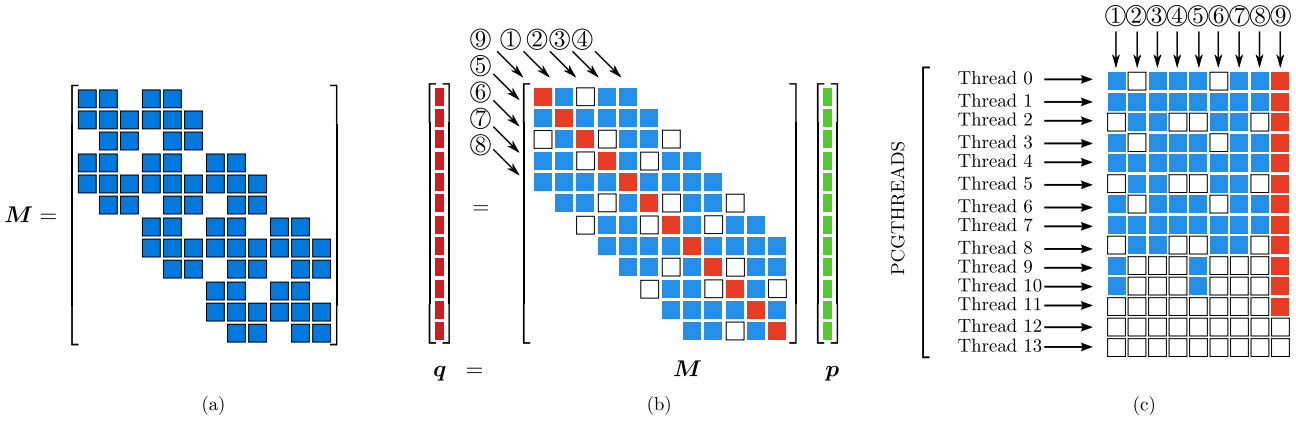

Figure 7. Parallel data access patterns. (a) The structure of $\boldsymbol{M}$, (b) Padding and SpMV in BdiaGPU. (c) The data access pattern for the numbered diagonals.

5. SIMULATIONS AND RESULTS

In this section, the results obtained using GRAND++ are presented for several meshes and solvers. All of the tests were performed using a computer with an Intel Core i7-4770 CPU at 3.40 GHz, 16.0 GB of RAM, an NVIDIA GeForce GTX 760 GPU with 2 GB of VRAM and the Windows 7 64-bit operating system. The parameters used to test the program were: stress limits $\sigma_T = \sigma_C = 10^6$, applied force $f = 1$, IPM tolerance $tol_{IPM} = 10^{-8}$, and PCG tolerance $tol_{PCG} = 10^{-10}$, which is used as the stopping criterion for the PCG loop. Optimization problems for the following three domains were solved: a structured bridge (Fig. 8 a), a non-structured hook (Fig. 8 b), and a 3D structured cantilever (Fig. 8 c). All of the domains were generated using GRAND (Zegard (2014)) and optimized using the proposed code, GRAND++. The times for the IPMs first iteration are shown in Fig. 9. We compare our solvers with PARDISO, which is a fast direct solver. Although PARDISO cannot optimize more than 15 M bars because it runs out of memory, the iterative solvers can optimize up to 30 M bars. The increases in speed are shown in Fig. 10 by comparing each solver with its CPU-based implementation.

6. CONCLUSIONS

The proposed solvers performed well for well-conditioned systems; however, for illconditioned, systems the PCG solver needs a better preconditioner. In addition, this solver requires a large system to minimize the effect of padding and

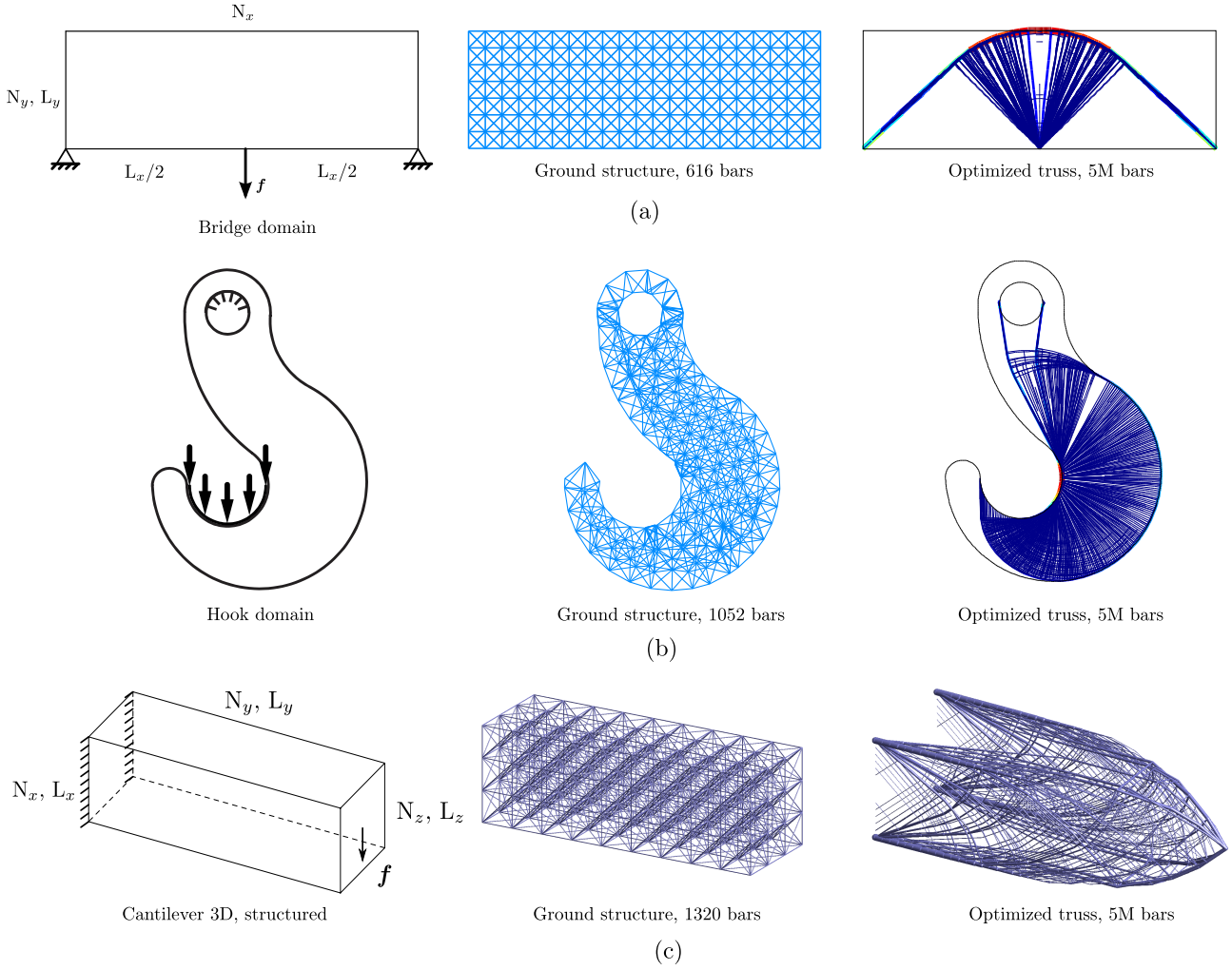

Figure 8. The domains, ground structures and optimized trusses for the (a) bridge domain, (b) hook domain and (c) cantilever domain.

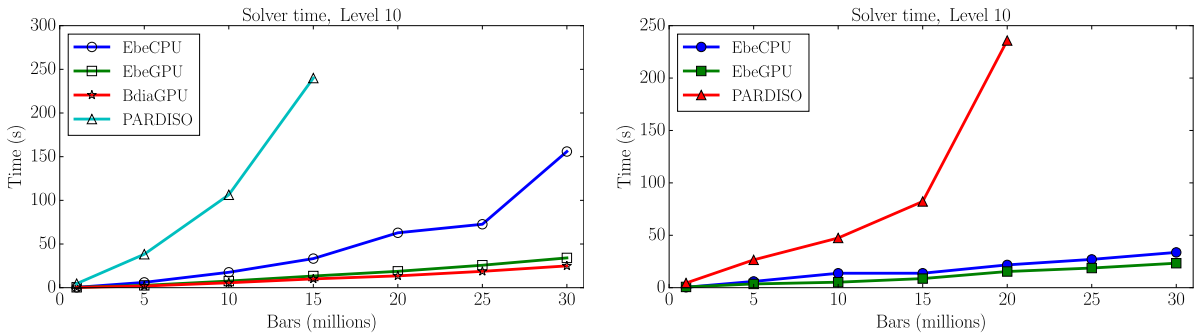

Figure 9. Solver times for the first iteration at connectivity level 10 for (a) structured meshes, (b) non-structured meshes.

allow an appropriate number of threads to be launched. If the system is small (fewer than 1 M bars), the solvers tend to waste computing time and memory. The BdiaGPU solver does not need coloring to avoid race conditions, but after every row or column is processed, the kernel needs to be launched because the GPU does not have a global sync. Only kernel launching can be used for this purpose, and it can waste a noticeable amount of time if numerous launches are made with little computation, as occurs for trusses with 1 million bars or fewer. The solver described in this work can be used not only for the topology optimization of trusses but also for applying the finite element method to trusses because of the similarity between matrix $\boldsymbol{M}$ and the stiffness matrix. Finally, the ideas presented here can be used by the community to develop solvers with other applications that require parallel computing.

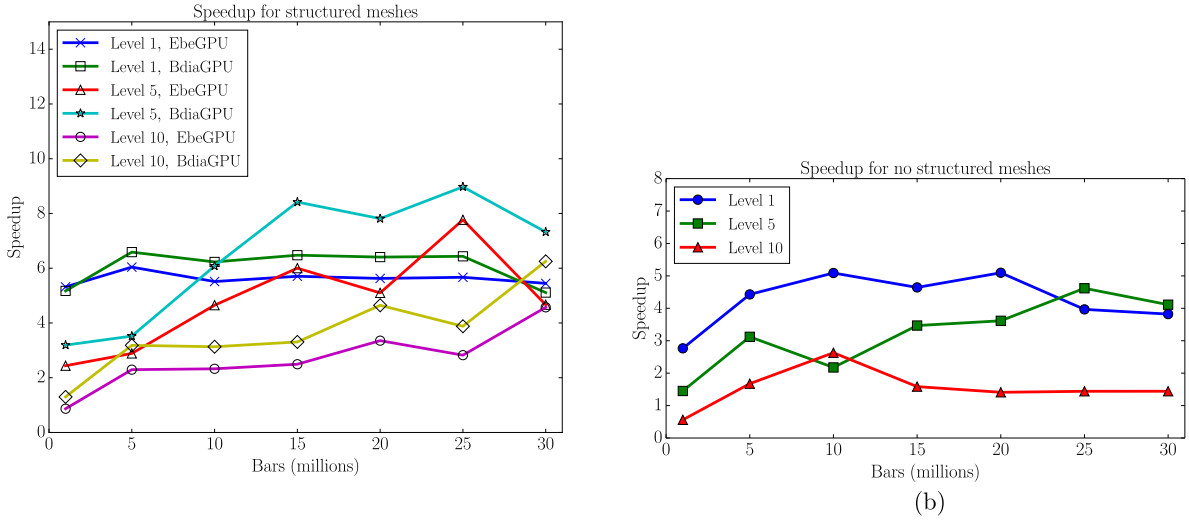

Figure 10. Speedups for connectivity levels 1, 5 and 10 for (a) structured meshes, (b) non-structured meshes.

7. ACKNOWLEDGEMENTS

The authors acknowledge the support provided by Tecgraf/PUC-Rio.

8. REFERENCES

Dorn, W.S., Gomory, R.E. and Greenberg, H.J., 1964. "Automatic design of optimal structures". *Journal de Mecanique, 3(1):25-52*.

Gebremedhin, A.H., Manne, F. and Pothen, A., 2005. "What color is your jacobian? graph coloring for computing derivatives". *SIAM REV*, Vol. 47, pp. 629–705.

Karmarkar, N., 1984. "A new polynomial time algorithm for linear programming". *Combinatorica 4:373-395*.

Kirsch, U., 1993. *Structural optimization: fundamentals and applications*. Springer-Verlag. ISBN 9783540559191.

Lewiński, T. and Rozvany, G., 2007. "Exact analytical solutions for some popular benchmark problems in topology optimization ii: three-sided polygonal supports". *Structural and Multidisciplinary Optimization*, Vol. 33, No. 4-5, pp. 337–349. ISSN 1615-147X. doi:10.1007/s00158-007-0093-7. URL `http://dx.doi.org/10.1007/s00158-007-0093-7`.

Michell, A., 1904. "The limits of economy of material in frame-structures". *Philosophical Magazine Series 6*, Vol. 8, No. 47, pp. 589–597. ISSN 1941-5982. doi:10.1080/14786440409463229.

Schmit, L., 1960. "Structural design by systematic synthesis". *Proceedings of the 2nd Conference on Electronic Computation. New York: American Society of Civil Engineering 1960, pp. 105-122*.

Zegard, T., 2014. *Methods and Tools for Discrete and Continuum Structural Optimization*. Ph.D. thesis, University of Illionis at Urbana-Champaign.

9. RESPONSIBILITY NOTICE

The authors are the only responsible for the printed material included in this paper.