Interactively exploring the connection between nested dissection orderings for parallel Cholesky factorization and vertex separators

H. Martin Bücker
Institute for Computer Science
Friedrich Schiller University Jena

M. Ali Rostami
Institute for Computer Science
Friedrich Schiller University Jena

***Abstract*—Parallel computing is increasingly becoming a core component of undergraduate computing curricula. It is not only part of degree programs in computer science or computer engineering but also in other computational disciplines like computational science and engineering. The underlying concepts of parallel computing affect and interact with many areas of computer science including hardware, software, algorithms, and networking, to name a few. Parallel algorithms for problems arising from scientific computing often involve some combinatorial aspects too. In particular, there is an intimate connection between graph theory and sparse linear algebra. A seemingly serial formulation of a sparse Cholesky factorization can be transformed into a block matrix formulation exhibiting parallelism. This transformation is known as nested dissection ordering and can be expressed in terms of a vertex separator in a suitably defined graph. This paper presents the design and implementation of a novel interactive educational model illustrating this connection for classroom use. The educational module also includes some characteristics of serial games offering a different type of interactive learning in parallel computing education.**

I. Introduction

High-performance computing is increasingly becoming a major building block in many different degree programs, including computer science, computer engineering, as well as computational science and engineering. We strongly believe that, in these areas, parallel computing has to be an integral part of an undergraduate curriculum, rather than being taught as an advanced topic at the graduate level. Therefore, one of the authors actively took part in establishing different elements of parallel computing on the undergraduate level at RWTH Aachen University, Germany [1], [2], [3], [4]. In particular, this technically-oriented university is offering a course in high-performance computing, which is mandatory for the bachelor degree in computational engineering science awarded after 3.5 years. It is also an elective course for the three year bachelor degree in computer science.

The course in high-performance computing introduces students to performance issues on computer architectures with deep memory hierarchies, scalable parallel algorithms, parallel programming for shared and distributed memory, and graph partitioning. It also covers the direct solution of sparse systems of linear equations, highlighting the intimate connection between scientific computing and graph theory. To illustrate this connection, a set of interactive educational modules was first introduced in [5]. This teaching aid is called EXPLoring Algorithms INteractively (EXPLAIN) and is currently being developed further and integrated into a redesigned computational and data science master curriculum at Friedrich Schiller University Jena, Germany. The collection of interactive educational modules includes a module on sparse Cholesky factorization and its graph-theoretical formulation known as vertex elimination. This module is described in more detail in [5]. Later, another module was added illustrating certain linear combinations of columns of a sparse matrix arising from automatic differentiation. This particular scientific computing problem corresponds to the graph-theoretical problem of finding a vertex coloring of a suitably defined graph. The module was recently completed and published in [6].

The new contribution of the present paper is to develop a new module that addresses the way how to introduce parallelism in a Cholesky factorization of a sparse matrix. The main goal of this module is to illustrate the transformation of a given ordering of rows and columns into a new ordering called nested dissection ordering. The crucial issue of this reordering is that, while the original ordering does not provide any options for parallel computing, the nested dissection ordering introduces parallelism in the Cholesky factorization. The module helps to understand the connection between this row/column reordering and its graph-theoretic correspondence of finding a vertex separator of a graph. In addition to demonstrating this connection, we extend the module with some ideas from gamification, meaning that students learn this connection between scientific computing and graph theory by playing an educational game. Our aim in using gamification is to increase the engagement of the students in the learning process.

The outline of the article is as follows. After briefly sketching some related work in Sec. II, we present a specific problem from numerical linear algebra. More precisely, in Sec. III, we consider the problem of finding a nested dissection ordering of a sparse matrix such that carrying out a block Cholesky factorization of the reordered matrix exhibits parallelism. This scientific computing problem corresponds to a particular graph problem. The graph-theoretical for-

978-1-4799-4116-2/14 $31.00 © 2014 IEEE
DOI 10.1109/IPDPSW.2014.125

mulation consists of finding a vertex separator with certain properties as described in Sec. IV. The educational module highlighting the connections between scientific computing and graph theory is presented in Sec. V. First experiences with this module in classroom are summarized in Sec. VI. The implementation details and the conclusion are given in Sec. VII and Sec. VIII.

II. Related Work

Since graphs are ubiquitous in computer science, mathematics, and a variety of other scientific disciplines there are plenty of software tools for teaching graph-theoretical topics and graph algorithms. However, to the best of the authors' knowledge, there is no other software than EXPLAIN that provides the simultaneous visualization of a graph and a matrix next to each other. This overall layout of EXPLAIN is crucial to better understand the relationship between the graph problem and the corresponding matrix problem. These two different views of the same problem are critical for establishing an understanding of the problem at hand.

Though there is no previous work directly related to that area, we shortly mention the Gato/CATBox [7] software whose focus is on animation of graph algorithms. Similarly, the CABRI-Graph [8] software is mainly used for algorithm visualization. There are also many software tools in the area of information visualization like Tulip [9], [10] that visualize and analyze graphs. However, all these graph software tools do not involve any aspects of scientific computing. On the other hand, existing tools with a focus on scientific computing do not involve any aspects from graph theory. Examples include the interactive Java applets devoted to the textbook by Heath [11] and the NCM software to be used in conjunction with the textbook by Moler [12].

Though our primary aim is to design an educational model for illustrating the connection between scientific computing and graph theory, we also experience with ideas from gamification [13], [14]. The use of elements from game design in the context of computer science education is not new. In particular, programming assignments can involve implementations of games. In [15], for instance, an introductory programming course is taught under the common umbrella of two-dimensional game development. Similarly, a game project is used in a course on software architecture [16]. Programming assignment can also involve pieces of software that act as a player in an existing game. However, throughout the present paper, our focus of serious games is different. Rather than implementing a game, we are interested in situations where students learn by playing a game. Surprisingly, there are only a few publications addressing this aspect of gamification. An example is given in [17] where game-based learning is used to teach a course in data structures and algorithms. A collaborative game is described in [18] that aims at improving the teaching quality in a course on mathematical logic.

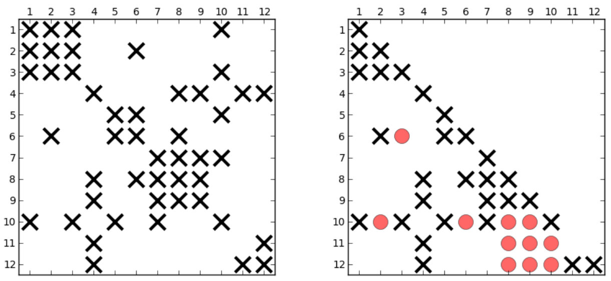


Figure 1. Nonzero pattern of a given matrix A (left) and it's Cholesky factor L with fill-in elements generated during the factorization (right).

III. Nested Dissection

Throughout this paper, we consider the following problem from scientific computing. Given a sparse positive definite system of linear equations, find a way to increase the level of parallelism when using the Cholesky factorization. In this numerical technique, the coefficient matrix A is decomposed into the product of the form

$$A = LL^T,$$

where the matrix L is lower triangular. The factorization is typically carried out "in-place," meaning that the same storage is used for the input A and the output L. That is, values of A are overwritten by values of L. It is known for a long time [19] that, for sparse matrices A, there are matrix positions (i, j) where the factorization process generates nonzero values in L where there were zeros in A. These additional nonzero elements in L are called fill-in elements. An example of a matrix A of order 12 and its Cholesky factor is depicted in Fig. 1. Here, a nonzero is indicated by the symbol $\times$ and a zero is represented by the empty space. The fill-in elements in L are marked by red bullets.

Now consider a reordering of the rows and columns of the matrix A into a matrix A'. In formulae, this reordering is nothing but a symmetric permutation $A' = PAP^T$ of the matrix A where P is a permutation matrix. Since symmetric permutations preserve the symmetry and positive definiteness, the Cholesky factorization can also be applied to the reordered matrix, i.e,

$$A' = L'(L')^T.$$

It is well known [20] that the reordering has an effect on the number and positions of the fill-in elements. Therefore, one is interested in finding a permutation P that is advantageous in some sense. In the following, we try to find a reordering of the form

$$A' = \begin{bmatrix} A_1 & 0 & B_1^T \\ 0 & A_2 & B_2^T \\ B_1 & B_2 & C \end{bmatrix}, \quad (1)$$

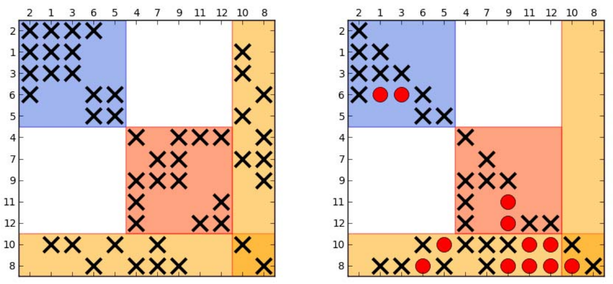


Figure 2. Nonzero pattern of the matrix A' obtained from a nested dissection ordering of A given in Fig. 1 (left) and it's Cholesky factor L' with fill-in elements generated during the factorization (right).

whose nonzero blocks are schematically depicted by color in the left of Fig. 2. We will now describe why this reordering allows to execute two tasks in parallel.

It is known that, independent from the nonzero pattern within the colored parts of A', fill-in elements in L' will not be generated in the zero block. In other words, we have the following block form of a Cholesky decomposition:

$$\begin{bmatrix} A_1 & 0 & B_1^T \\ 0 & A_2 & B_2^T \\ B_1 & B_2 & C \end{bmatrix} = \begin{bmatrix} L_1 & 0 & 0 \\ 0 & L_2 & 0 \\ E_1 & E_2 & D \end{bmatrix} \cdot \begin{bmatrix} L_1^T & 0 & E_1^T \\ 0 & L_2^T & E_2^T \\ 0 & 0 & D^T \end{bmatrix}.$$

Thus, given the blocks A_1, A_2, B_1, B_2 and C in the matrix A', we immediately find the blocks L_1, L_2, E_1, E_2 and D in its Cholesky factor L' as follows:

$$A_1 = L_1 L_1^T, \tag{2}$$

$$A_2 = L_2 L_2^T, \tag{3}$$

$$L_1 E_1^T = B_1^T, \tag{4}$$

$$L_2 E_2^T = B_2^T, \tag{5}$$

$$C = E_1 E_1^T + E_2 E_2^T + DD^T. \tag{6}$$

That is, from (2) and (3), the blocks L_1 and L_2 are given by Cholesky factorizations of the blocks A_1 and A_2, respectively. Once these partial results are available, the next task is to solve the two linear systems with multiple right-hand sides (4) and (5) for the blocks E_1 and E_2, respectively. Finally, by rearranging (6) into

$$C - E_1 E_1^T - E_2 E_2^T = DD^T, \tag{7}$$

the block D is given by another Cholesky factorization of the block $C - E_1 E_1^T - E_2 E_2^T$.

The crucial issue here is that the task of computing the Cholesky factorization in (2) followed by a solution of (4) is independent from the corresponding task involving (3) followed by (5). The only serial part consists of the Cholesky factorization in (7). So, having two independent processes, we would like to minimize the computational work associated with the serial bottleneck (7) while satisfying the constraint that the computational work associated with (2) and (4) is (almost) the same as the computational work associated with (3) and (5). This is formally stated in the following problem.

Problem 1 (NESTED DISSECTION ORDERING): Given a sparse symmetric positive definite matrix A, find a symmetric permutation P^TAP of A in the form of (1) such that the size of the block C is minimized while the sizes of the blocks A_1 and A_2 are balanced.

This reordering approach can be applied recursively to the blocks A_1 and A_2, which explains the name "nested dissection ordering." Finally, we stress that this approach easily generalizes to $p \geq 2$ processes in which case the nested dissection ordering of the matrix is given by

$$A' = \begin{bmatrix} A_1 & 0 & \cdots & 0 & B_1^T \\ 0 & A_2 & \cdots & 0 & B_2^T \\ \vdots & \vdots & \ddots & 0 & \vdots \\ 0 & 0 & \cdots & A_p & B_p^T \\ B_1 & B_2 & \cdots & B_p & C \end{bmatrix}.$$

IV. FINDING A SMALL VERTEX SEPARATOR

The problem NESTED DISSECTION ORDERING is now reformulated in terms of an undirected graph. To this end, a graph $G = (V, E)$ is associated with a sparse matrix A with a given symmetric nonzero pattern as follows. There is a vertex $v_i \in V$ for each row i of the matrix A. There is an edge $(v_i, v_j) \in E$ if and only if the matrix element $A(i,j) \neq 0$ and $i > j$. Here, the inequality condition prevents the addition of two edges between two vertices in G. In other words, the adjacency matrix [21] of G has the same nonzero pattern as the matrix A. An example is presented in Fig. 3. In this figure, the first row and the first column of the matrix are completely filled with nonzero elements, which corresponds to the first vertex being connected to all other vertices.

A nested dissection ordering is then obtained from decomposing the set of vertices into three disjoint sets

$$V = V_1 \cup V_2 \cup S$$

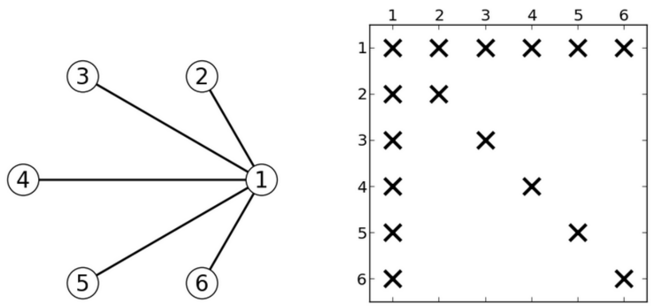


Figure 3. Graph G (left) associated with a sparse matrix A (right).

where S is a vertex separator. A subset of vertices S is called a vertex separator of G if the subgraph induced by removing from G all vertices in S, including their incident edges, separates G into two disconnected components V_1 and V_2. A nested dissection ordering results from numbering the vertices in V_1 first, then the vertices in V_2, and those in the separator S last. Within each of the three sets V_1, V_2 and S, the vertices are numbered in any order.

This way, the vertices in V_1 represent rows and columns in the block A_1. The edges connecting vertices in V_1 represent the nonzero elements in A_1. In the same manner, the vertices and edges in V_2 represent the block A_2 and its nonzeros. The vertex separator corresponds to the rows and columns in the blocks B_1, B_2 and C. So, Problem 1 is equivalent to finding a small vertex separator S with balanced components V_1 and V_2. This is summarized in the following problem.

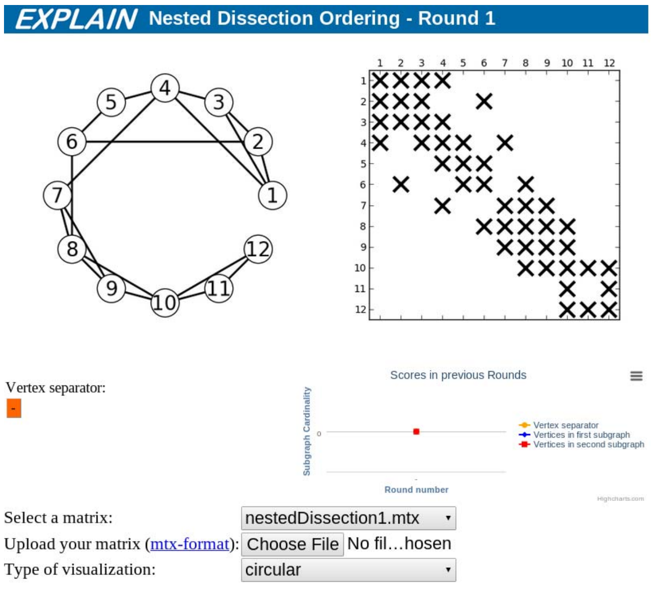
Problem 2 (SMALL VERTEX SEPARATOR): Given the graph G associated with a sparse matrix A, find a disjoint decomposition of the vertices $V = V_1 \cup V_2 \cup S$ with a vertex separator S such that the size of the vertex separator, $|S|$, is minimized while the sizes of the two remaining components, $|V_1|$ and $|V_2|$, are balanced.

More details on the connection between Cholesky factorization and graphs are given in the textbook [22]. In the next section, we introduce a novel educational module to examine the equivalence of Problems 1 and 2.

V. NESTED DISSECTION ORDERING MODULE

In [6], EXPLAIN is defined as an extensible collection of modules for classroom use. We extended the software with a new module that considers the nested dissection ordering of a given matrix and the equivalent graph problem, i.e., finding a vertex separator. The overall layout of this module with a preloaded matrix is depicted in Fig. 4. In the top part of this layout, the graph and the matrix are shown next to each other. In the bottom part, there is room for displaying the history of choosing the vertices of the vertex separator in a specific round, the scores of previous rounds, and panels to control the choice of the matrix/graph instance. The term "round" refers to the process of solving a single instance of a given problem. In this module, a round consists of finding a vertex separator.

In Fig. 4, the graph is visualized in a circular layout, which is the default. However, we integrated the new feature to select various graph layouts. The layouts are the ones supported by the library *matplotlib* [23]. This new functionality can be used also in other modules but it is particularly important in the nested dissection ordering. A "good" graph layout reveals the vertex separator. For instance, a vertex separator is not immediately obvious from visual inspection of the circular layout given in Fig. 5 (left). However, a vertex separator is observable from a force-directed (spring) layout [24] of the same graph depicted on the right of this figure.


EXPLAIN Nested Dissection Ordering - Round 1

Vertex separator:

Select a matrix: nestedDissection1.mtx

Upload your matrix (mtx-format): Choose File No fil…hosen

Type of visualization: circular

Figure 4. Overall layout of the nested dissection module.

In similar modules of previous versions of EXPLAIN, the students did not get any feedback when a round is completed. Also, there was no indication of the results of previous rounds. As a new feature, we use gamification to engage the students more in the process of teaching and to provide more feedback to them. According to [14], gamification is the idea of using game design elements in non-game contexts to motivate and increase user activity. The concept of a round and completion of a round is added to the software. EXPLAIN shows the round number at the top of the page; see Fig. 4. After completion of an algorithm, the text "Round x is completed" signals to the student the end of this round.

Recall from the previous section that the goal of nested dissection ordering is to find three subgraphs. We assign a color to each subgraph: Orange for the vertex separator S, blue for the component V_1, and red for the component V_2. The same colors are used for the corresponding matrix blocks. These colors help students to have a unique view

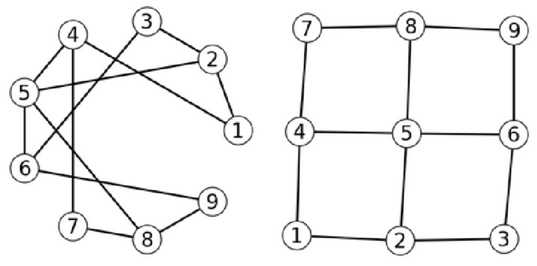


Figure 5. The same graph in circular and force-directed layout.

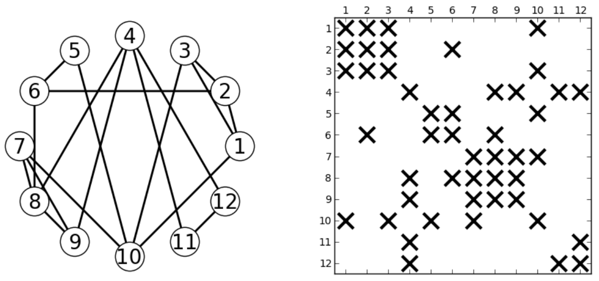


Figure 6. Two equivalent representations in terms of a graph (left) and a matrix (right).

of the equivalence of the graph and the matrix view of that problem.

In addition, a score diagram keeps the information from the previous rounds. This diagram provides the opportunity to assess the improvement of the result over multiple rounds. The score diagram visualizes the cardinality of the three subgraphs generated from the nested dissection ordering. The same colors for the vertex sets V_1, V_2 and S are used in the graph view, the matrix view, and also for the corresponding curves in the score diagram, see Fig. 10.

In each step of a round, the student clicks on a vertex in the graph view. By that, the selected vertex is assigned to the vertex separator. As a result, the selected vertex and the corresponding column will be colored orange and the column will be moved to the right of the matrix. Also, this vertex number is added to the vertex separator list, which saves the history of the selections as well as the possibility to return to that step. We remove all edges incident to the vertices in the vertex separator to show the effect of forming the three components. However, the nonzeros corresponding to these edges removed in the graph view are still depicted in the matrix view. The two components V_1 and V_2, together with their corresponding matrix blocks A_1 and A_2, are colored only if a vertex separator is determined.

As an example, consider a scenario where one is interested in finding a small vertex separator for the matrix shown in Fig. 6. Suppose that the student first selects the vertex 10. As a result the row and column number 10 is brought to the right of the matrix and they will be colored as orange; see Fig. 7. Notice that, now, all edges incident to the vertex 10 are eliminated in the graph view, while the corresponding nonzeros are still shown in the matrix view. Then, the student needs to click on vertices repeatedly, until a vertex separator is found. The size of the vertex separator depends on the student's choice of the vertices. It is not always possible to find a separator with minimal size in the first try. Fig. 8 represents the result in which the student has clicked first on vertex 10 and then on 4. This result is bad since the number of vertices in V_1 and V_2, corresponding to the sizes

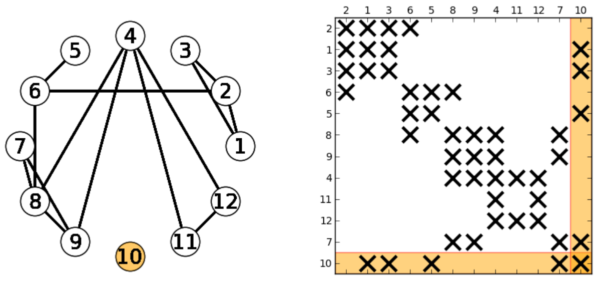


Figure 7. Graph and matrix view after selecting the vertex number 10. The decomposition into two blocks is still not shown as the graph is not yet decomposed into two disconnected components.

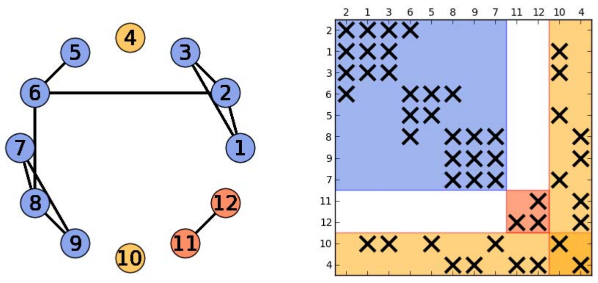


Figure 8. Graph and matrix view after selecting the vertices number 10 and then 4. The selection is not adequate as the sizes of blocks are not balanced.

of the blocks, is not balanced. A better result for this problem instance could be achieved with more effort by clicking on vertices 8 and 10. As shown in Fig. 9, the size of the vertex separator is given by 2 and the sizes of the diagonal blocks are balanced.

A resulting score diagram from different rounds looks like Fig. 10. Here, the blue and red curves should have values close to each other since this indicates the balancing

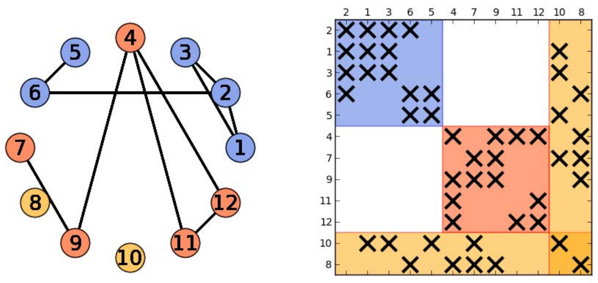


Figure 9. Graph and matrix view after selecting the vertices number 10 and then 8. The block sizes are balanced and the separator size is minimized.

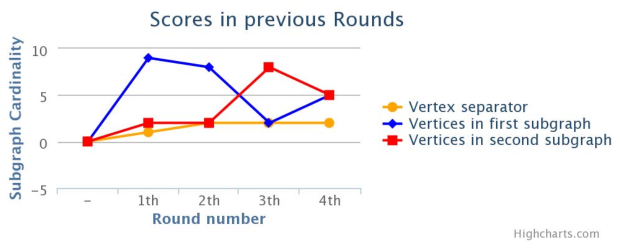


Figure 10. Score diagram resulting from four different rounds.

condition. The orange curve should be as low as possible because it represents the minimization of the separator size. In this figure, the separator size is indeed minimal and the balance condition is satisfied in the last round.

VI. First Experience in Classroom

We used this educational module in the course "High-Performance Computing" at the University of Jena, Germany. This course was offered in winter 2013/14 within the master curriculum of computer science, with a few students from other master and (upper-level) bachelor curricula including bioinformatics and physics.

During this course the Cholesky factorization was thoroughly treated with an emphasis on sparse matrices. After this introduction to sparse Cholesky factorization, the idea of nested dissection was formally explained in classroom without the module. In the following week, the teacher started with a short recapitulation of nested dissection, now introducing the new module. The module was presented with specific problem instances and the students were asked in classroom to decide which vertex is selected for the vertex separator in the next step. The students suggested different orders of selection to get to the optimum result. They were then asked why some of the results are not optimal. Finally, they could find the optimal solution only when we show the graph by the force-directed layout.

A preliminary evaluation of the module was carried out immediately after using this module for the first time in classroom. A short summary of this interactive feedback by the students is as follows:

- The concept of a round was clear to them. Hence, they did compare their new results with those from the previous rounds. However, they mentioned that the functionality of returning to a previous round is missing.
- Since the matrix columns and rows are reordered using a depth-first search algorithm on the graph, their numbering within a block is unsorted. This confused the students when taking a first glance. The vertex separator is clear enough as it moves to the right of the matrix and is also colored orange.
- The students found the coloring useful and clear. In particular, the use of the same color for the graph, the matrix, and the score diagram was considered to be convincing.
- After removing the vertex separator, the remaining two components are not necessarily connected. Thus, if one or both of the components V_1 and V_2 are disconnected, there are different ways to form the two components V_1 and V_2. This was an issue that the students suggested to clarify in future versions of this module.

VII. Implementation Details

As described in [6], the following libraries are used to implement EXPLAIN: *NetworkX* [25] for the graph data structure, *matplotlib* [23] for the visualization aspects, *Scipy* [26] for the sparse matrix computation, and *Mod_python* [27] for the web based version. In this section, we explain the new features and how they are added to EXPLAIN.

In any step of the algorithm, the student clicks on a vertex v. This choice is immediately visualized by displaying this vertex as well as the corresponding column and row in the color orange. Also, the edges incident to v are removed and the program searches for components by the depth-first search (DFS) algorithm. If two different components are found, those will be colored blue and red. The following pseudo-code shows the algorithm:

```
# remove the edges of v
for u in G.neighbors(v):
     G.remove_edge(u,v)

# color the vertex v and its
# corresponding row and column

# add v to the vertex separator
sep.append(v)

# find the set of non-separator vertices
non_sep=set(G.nodes()).difference(sep)

# depth-first search
dfs_nodes = nx.dfs_preorder_nodes(G, non_sep)
first_component =
list(it for it in dfs_nodes)

second_component =
list(set(non_sep).difference(first_component))

if len(second_component)!=0:
  # color the matrix blocks
```

To color the matrix blocks, the rows and columns need to be grouped corresponding to the vertices in the connected components V_1 and V_2 and in the vertex separator S. Any numbering of the vertices within each of these three groups of vertices is valid. Let *order* denote the reordering of all vertices obtained from appending the orderings of the three groups V_1, V_2 and S, in that order. Given the reordering *order* we can easily construct the permutation matrix P representing the reordered matrix $A' = P^T AP$. The following pseudo-code shows the implementation:

```
order = first_component + second_component
        + separator

# create permutation matrix
P = scipy.sparse.csr_matrix(A.shape)
for i,j in enumerate(order):
  P[j,i] = 1.

A_prime = P.transpose()*A*P
```

Finally, we used the Javascript library Highcharts [28] to include the score diagram of the previous rounds. This library draws the three colored curves. The generation of the diagram is fast since it is written completely in Javascript.

VIII. Conclusion

We consider the problem of teaching how parallelism is introduced to a sparse Cholesky factorization via a nested dissection ordering. The overall idea of our teaching approach is to demonstrate the intimate connection of finding a nested dissection ordering of a matrix and finding a vertex separator in the graph associated to that matrix. To this end, we design and implement a new interactive educational module with an emphasis to visualize this connection clearly. We also report on preliminary classroom experiences when using this web-based module in a course on high-performance computing. It turns out that this module enhances the students' understanding of the topic. This module is integrated into a collection of educational module called EXPLAIN.

To engage the students more in the teaching process, we improved EXPLAIN such that the students get more feedback from the software. This is done by gamification of the software by interpreting each solution to a problem instance as a round. The score diagram reporting the results of previous rounds also provides another feedback. The idea of gamification is used to solve a combinatorial minimization problem consisting of minimizing the size of the vertex separator while, at the same time, balancing the size of the remaining components. The consistent use of colors in the graph view, the matrix view, and in the score diagram makes it easier for the student to understand minimizing a serial bottleneck in the Cholesky factorization while balancing the computational load.

There is still room for improvement and extension of the educational module. For instance, nested dissection is typically applied recursively. However, the current version of the module is capable of determining a single vertex separator. That is, EXPLAIN is actually addressing a bisection problem rather than a nested dissection approach. In the future, one might therefore consider the extension to nested dissection where multiple vertex separators, which are computed recursively, can be explored interactively. This would require a major change in the software design because the focus of the current design is on small graphs and matrices. Any recursive approach would require to support larger graphs and matrices.

Acknowledgment

The authors thank Michael Lülfesmann for his major role in designing previous versions of the EXPLAIN software. This work is partially supported by the German Federal Ministry for the Environment, Nature Conservation, Building and Nuclear Safety (BMUB) within the project MeProRisk II, contract number 0325389F.

References

[1] H. M. Bücker, B. Lang, and C. H. Bischof, "Teaching Different Parallel Programming Paradigms Using Java," in *Proceedings of the 3rd Annual Workshop on Java for High Performance Computing, Sorrento, Italy, June 17, 2001*, 2001, pp. 73–81.

[2] C. H. Bischof, H. M. Bücker, J. Henrichs, and B. Lang, "Hands-On Training for Undergraduates in High-Performance Computing Using Java," in *Applied Parallel Computing: New Paradigms for HPC in Industry and Academia, Proceedings of the 5th International Workshop, PARA 2000, Bergen, Norway, June 2000*, ser. Lecture Notes in Computer Science, T. Sørevik, F. Manne, R. Moe, and A. H. Gebremedhin, Eds., vol. 1947. Berlin: Springer, 2001, pp. 306–315.

[3] H. M. Bücker, B. Lang, and C. H. Bischof, "Parallel programming in computational science: an introductory practical training course for computer science undergraduates at Aachen University," *Future Generation Computer Systems*, vol. 19, no. 8, pp. 1309–1319, 2003.

[4] H. M. Bücker, B. Lang, H.-J. Pflug, and A. Vehreschild, "Threads in an Undergraduate Course: A Java Example Illuminating Different Multithreading Approaches," in *Computational Science and Its Applications – ICCSA 2004, Proceedings of the International Conference on Computational Science and its Applications, Assisi, Italy, May 14–17, 2004. Part II*, ser. Lecture Notes in Computer Science, A. Laganà, M. L. Gavrilova, V. Kumar, Y. Mun, C. J. K. Tan, and O. Gervasi, Eds., vol. 3044. Berlin: Springer, 2004, pp. 882–891.

[5] M. Lülfesmann, S. R. Lessenich, and H. M. Bücker, "Interactively exploring elimination ordering in symbolic sparse cholesky factorization," *International Conference on Computational Science, ICCS 2010*, vol. 1, pp. 867–874, 2010.

[6] H. M. Bücker, M. A. Rostami, and M. Lülfesmann, "An Interactive Educational Module Illustrating Sparse Matrix Compression via Graph Coloring," in *2013 International Conference on Interactive Collaborative Learning (ICL), Proceedings of the 16th International Conference on Interactive Collaborative Learning, Kazan, Russia, September 25–27, 2013.* IEEE, 2013, pp. 330–335.

[7] A. Schliep and W. Hochstättler, "Developing Gato and CATBox with Python: Teaching graph algorithms through visualization and experimentation," *Multimedia Tools for Communicating Mathematics*, pp. 291–310, 2002.

[8] Y. Carbonneaux, J.-M. Laborde, and R. Madani, *CABRI-Graph: A tool for research and teaching in graph theory*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 1996, vol. 1027.

[9] D. Auber, D. Archambault, R. Bourqui, A. Lambert, M. Mathiaut, P. Mary, M. Delest, J. Dubois, and G. Melançon, "The Tulip 3 Framework: A Scalable Software Library for Information Visualization Applications Based on Relational Data," INRIA, Research Report RR-7860, Jan. 2012. [Online]. Available: http://hal.archives-ouvertes.fr/hal-00659880

[10] A. Lambert and D. Auber, "Graph analysis and visualization with Tulip-Python," in *EuroSciPy 2012 – 5th European meeting on Python in Science*, Bruxelles, Belgique, 2012-08. [Online]. Available: http://hal.archives-ouvertes.fr/hal-00744969

[11] M. T. Heath, *Scientific Computing: An Introductory Survey*, 2nd ed. McGraw-Hill, 2002.

[12] C. B. Moler, *Numerical Computing with MATLAB*. Philadelphia, PA, USA: SIAM, 2004.

[13] S. Deterding, M. Sicart, L. Nacke, K. O'Hara, and D. Dixon, "Gamification: Using game-design elements in non-gaming contexts," in *CHI '11 Extended Abstracts on Human Factors in Computing Systems*, ser. CHI EA '11. New York, NY, USA: ACM, 2011, pp. 2425–2428. [Online]. Available: http://doi.acm.org/10.1145/1979742.1979575

[14] S. Deterding, D. Dixon, R. Khaled, and L. Nacke, "From game design elements to gamefulness: Defining "gamification"," in *Proceedings of the 15th International Academic MindTrek Conference: Envisioning Future Media Environments*, ser. MindTrek '11. New York, NY, USA: ACM, 2011, pp. 9–15.

[15] S. Leutenegger and J. Edgington, "A games first approach to teaching introductory programming," in *Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education*, ser. SIGCSE '07. New York, NY, USA: ACM, 2007, pp. 115–118. [Online]. Available: http://doi.acm.org/10.1145/1227310.1227352

[16] A. I. Wang, "Extensive evaluation of using a game project in a software architecture course," *Trans. Comput. Educ.*, vol. 11, no. 1, pp. 5:1–5:28, Feb. 2011. [Online]. Available: http://doi.acm.org/1921607.1921612

[17] L. Hakulinen, "Using serious games in computer science education," in *Proceedings of the 11th Koli Calling International Conference on Computing Education Research*, ser. Koli Calling '11. New York, NY, USA: ACM, 2011, pp. 83–88. [Online]. Available: http://doi.acm.org/10.1145/2094131.2094147

[18] A. Schäfer, J. Holz, T. Leonhardt, U. Schroeder, P. Brauner, and M. Ziefle, "From boring to scoring – a collaborative serious game for learning and practicing mathematical logic for computer science education," *Computer Science Education*, vol. 23, no. 2, pp. 87–111, 2013.

[19] S. Parter, "The use of linear graphs in Gauss elimination," *SIAM Review*, vol. 3, no. 2, pp. pp. 119–130, 1961.

[20] D. J. Rose, "A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations," in *Graph Theory and Computing*, R. C. Read, Ed. New York: Academic Press, 1972, pp. 183–217.

[21] A. Bondy and U. Murty, *Graph Theory*, ser. Graduate Texts in Mathematics. Springer, 2008.

[22] T. A. Davis, *Direct Methods for Sparse Linear Systems*, ser. Fundamentals of Algorithms. SIAM, Philadelphia, 2006.

[23] J. Hunter, "Matplotlib: A 2D graphics environment," *Computing in Science Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[24] S. G. Kobourov, "Spring embedders and force directed graph drawing algorithms," Computing Research Repository (CoRR), arXiv preprint arXiv:1201.3011, 2012.

[25] A. A. Hagberg, D. A. Schult, and P. J. Swart, "Exploring network structure, dynamics, and function using NetworkX," in *Proceedings of the 7th Python in Science Conference*, G. Varoquaux, T. Vaught, and J. Millman, Eds., Pasadena, CA USA, 2008, pp. 11–15.

[26] E. Jones *et al.*, "SciPy: Open source scientific tools for Python," 2001–. [Online]. Available: http://www.scipy.org

[27] A. S. Foundation, "Mod_python module," Jan. 2013. [Online]. Available: http://www.modpython.org

[28] J. Kuan, *Learning Highcharts*. Packt Publishing, 2012.