

# Megapixel Topology Optimization on a Graphics Processing Unit\*

Eddie Wadbro<sup>†</sup>  
Martin Berggren<sup>‡</sup>

**Abstract.** We show how the computational power and programmability of modern graphics processing units (GPUs) can be used to efficiently solve large-scale pixel-based material distribution problems using a gradient-based optimality criterion method. To illustrate the principle, a so-called topology optimization problem that results in a constrained nonlinear programming problem with over 4 million decision variables is solved on a commodity GPU.

**Key words.** topology optimization, Poisson's equation, GPU, HPC

**AMS subject classifications.** 65K05, 65Y05, 65Y10, 90C06

**DOI.** 10.1137/070699822

**1. Introduction.** During the last few years, the computational power of graphics processing units (GPUs) has increased at a much higher rate than the corresponding rate for regular CPUs. Compared to a typical CPU, the GPU allocates more transistors to data processing and fewer to caching and flow control. Graphics hardware has evolved rapidly from fixed-function pipelines into general-function stream processors. Moreover, a graphics card is a standard computer component and is relatively inexpensive. Thus, a regular computer equipped with a high-end GPU is potentially a low budget “supercomputer.” Recent releases of GPU software development kits by the hardware vendors ATI and NVIDIA facilitate the programming of GPUs and make their computational power increasingly accessible for general computations.

The hardware architecture imposes an algorithmic constraint on the class of problems that benefit from GPU acceleration. Candidate problems need to be solvable by algorithms in which data-parallel computations dominate the computational effort. We will demonstrate that certain problems concerning optimization of material distributions are well suited for GPU computations. To the best of our knowledge, these kinds of problems have not been solved on a GPU before.

Problems for which material properties at each point of an object are to be determined from computations are of high and increasing engineering and scientific in-

---

\*Received by the editors August 10, 2007; accepted for publication (in revised form) September 16, 2008; published electronically November 6, 2009. Funding for this research has been provided by the Swedish Research Council.

<http://www.siam.org/journals/sirev/51-4/69982.html>

<sup>†</sup>Department of Information Technology, Uppsala University, Box 337, SE-751 05 Uppsala, Sweden. Current address: Department of Computing Science, Umeå University, SE-901 87 Umeå, Sweden (eddie.wadbro@cs.umu.se).

<sup>‡</sup>Department of Computing Science, Umeå University, SE-901 87 Umeå, Sweden (martin.berggren@cs.umu.se).

terest. A prime example is the problem of creating a load-carrying structure by freely distributing a fixed amount of material in an optimal way according to a criterion such as maximum stiffness. Bendsøe and Sigmund [3] extensively review this type of problem, usually called *topology optimization*. Topology optimization capabilities are included in commercial finite-element packages, for instance, those from Altair Engineering and FE-Design. These packages are increasingly used in the design of advanced mechanical components, for instance, in the aerospace and car industries. A problem similar in nature to topology optimization is the so-called inverse problem of determining material properties from a limited amount of measurements of an object's response to mechanical or electromagnetic forcing. In previous publications, we have applied the topology optimization concept to the design of devices involved in wave propagation problems [27, 28], and we have used similar ideas for microwave tomography, where material properties of tissue are reconstructed from scattered microwave radiation [29].

Material distribution problems are typically cast as large-scale nonlinear programming problems over the coefficients in a partial differential equation. The structural optimization community is a source of several successful optimization algorithms particularly tailored for such problems; we refer to the book by Bendsøe and Sigmund [3] for an overview and further references. The coefficient field is preferably represented in terms of large (for high resolutions) rectangular arrays of equally sized (to avoid a priori bias) pixels or voxels. GPUs are perfectly suited for the manipulations of such objects that are needed in the optimization algorithms. Most algorithms also require, at each iteration, an accurate numerical solution of the partial differential equation associated with the candidate coefficient field. For the success of a GPU implementation, it is therefore crucial that the partial differential equation can be efficiently solved in a data-parallel manner. We discuss the implementation on a GPU of a prototypical material distribution problem for which this requirement holds.

Our contribution is in line with a recent trend that utilizes GPUs to accelerate the computation for general problems. In a recent survey, Owens et al. [22] present a number of applications for which graphics hardware is used as a general-purpose compute device. The Norwegian strategic institute project "Graphics hardware as a high-end computational resource" focuses on image processing, partial differential equations, geometry, and linear algebra [7]. Harris et al. [15] simulate dynamic physical phenomena, such as boiling and reaction-diffusion, by modeling the complex global behavior with a set of simple local operations. Rumpf and Strzodka [23] use the graphics processor to track the propagation of a two-dimensional level-set model in order to perform segmentation of digital images. Their ideas were later generalized, by Lefohn et al. [19], into a GPU model supporting sparse and dynamic grids for volume deformation and visualization. Krüger and Westermann [18] develop and implement linear algebraic operators on a GPU and perform numerical simulations of physical phenomena of waves and fluids. Bolz et al. [4] use the GPU for a sparse-matrix conjugate gradient and multigrid solver and demonstrate the methods on geometric flow and fluid simulations. Goodnight et al. [12] present a multigrid solver for more general boundary value problems. Hillesland, Molinov, and Grzeszczuk [17] study a class of nonlinear optimization problems as a data streaming process and apply their methodology to solve image-based modeling problems. Until quite recently, one technical limitation was that GPUs natively offered only single precision accuracy. Hillesland and Lastra [16] and Da Graça and Defour [13] describe how to implement and perform computations in higher precision on the graphics device. Göddeke, Strzodka, and Turek [11] use a mixed precision approach to achieve higher accuracy

for finite-element simulations, illustrating benefits for several conjugate gradient and multigrid solvers. Recent releases of GPUs with native double precision support have finally overcome the single precision limitation.

**2. Problem Description.** A rectangular plate occupies a unit-size two-dimensional domain  $\Omega$ , as illustrated in Figure 2.1. The boundary  $\partial\Omega$  consists of two nonoverlapping parts  $\Gamma_D$  and  $\Gamma_N$ . The plate is insulated along boundary  $\Gamma_N$ , held at constant temperature at boundary  $\Gamma_D$ , and heated by sources uniformly distributed within the domain. We assume that the plate is made out of a solid material with heat conduction properties that are inhomogeneous (spatially varying) but isotropic (equal in each direction). At thermal equilibrium, the temperature field  $T$  satisfies

$$(2.1) \quad \begin{cases} -\nabla \cdot (\kappa \nabla T) = f & \text{in } \Omega, \\ T = 0 & \text{on } \Gamma_D, \\ (\kappa \nabla T) \cdot n = 0 & \text{on } \Gamma_N, \end{cases}$$

where  $\kappa$  is the scalar heat conduction coefficient,  $f$  the (constant) heat source density ( $f \equiv 1$  is used in the numerical experiments), and  $n$  the unit normal on  $\Gamma_N$ .

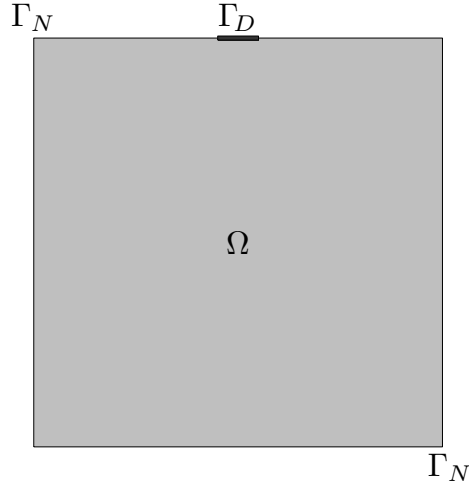
Now assume that we have a limited amount of a high conductivity material ( $\kappa = \bar{\kappa}$ ) and an unlimited amount of a low conductivity material ( $\kappa = \underline{\kappa}$ ); we used  $\bar{\kappa} = 1$  and  $\underline{\kappa} = 0.001$  in the numerical experiments. We parameterize the heat conduction coefficient at each point in the domain as

$$(2.2) \quad \kappa(x) = \underline{\kappa} + (\bar{\kappa} - \underline{\kappa})\alpha(x),$$

where  $\alpha$  belongs to the set of admissible design variables

$$\mathcal{U} = \{ \alpha \in L^\infty(\Omega) \mid \alpha(x) \in \{0, 1\} \text{ a.e. in } \Omega \}.$$

We seek to distribute these two materials in order to obtain a temperature field that is as “even” as possible, a requirement that we formulate as the minimization of



**Fig. 2.1** The problems consist of finding the distribution within  $\Omega$  of two materials with different heat conduction properties in order to obtain a temperature field that is as even as possible.

the objective function

$$(2.3) \quad J(\alpha) = \int_{\Omega} T f \, dx,$$

where  $T$  is obtained by solving equation (2.1), in which  $\kappa$  is defined by expression (2.2). Note that we are minimizing the *average* temperature, since  $f$  is constant. Our material distribution problem can then be formulated as the optimization problem

$$(2.4) \quad \begin{aligned} & \min_{\alpha \in \mathcal{U}} J(\alpha) \\ & \text{subject to } \int_{\Omega} \alpha \, dx \leq V, \end{aligned}$$

where  $V$  corresponds to the available amount of the high conductivity material.

Equation (2.1), or others that are very similar, also models many other physical phenomena, such as groundwater flow, elastic torsion, and electrostatics. Variations of optimization problem (2.4) can be formulated for many of these models, as Donoso and Sigmund [8] comprehensively describe. The structure of optimization problem (2.4) is similar to other more complicated topology optimization problems, such as the material distribution problem of linear elasticity; we refer to Bendsøe and Sigmund's book [3] for more examples and applications. The problem can therefore act as an appropriate model problem in the development of new algorithms, such as Gersborg-Hansen, Bendsøe, and Sigmund's [10] investigation of the use of finite-volume methods in the context of topology optimization.

Optimization problem (2.4) is an example from a class of problems that Bejan [1] calls area-to-point flow problems. As we will see, the picture of the optimal conductivity distribution resembles the root of a plant. Bejan argues that this and similar geometric shapes, ubiquitous in engineering and the natural world, originate in optimization principles. Our experiments demonstrate that finer and finer feeder roots emerge as the discretization is refined. This is a manifestation of the fact that optimization problem (2.4) is ill-posed in the sense that the problem lacks solutions within the set  $\mathcal{U}$  of feasible designs: there exist nonconvergent minimizing sequences of elements from  $\mathcal{U}$ . For real life problems there are often some external requirements, such as a minimal width of the structural members. Imposing such a constraint assures the existence of a minimizer within the set  $\mathcal{U}$ .

The requirement that  $\mathcal{U}$  should be binary-valued leads in the discrete case to a large-scale nonlinear integer programming problem. Generally, problems of this type are computationally expensive to solve. Therefore, we choose to attack the problem using a gradient-based method in combination with a penalization technique. In order to use a gradient-based algorithm, we relax the range of the design variable to the continuum  $[0, 1]$ ; that is, we replace the space of admissible designs  $\mathcal{U}$  with

$$\widehat{\mathcal{U}} = \{ \alpha \in L^{\infty}(\Omega) \mid \alpha(x) \in [0, 1] \text{ a.e. in } \Omega \}.$$

Replacing  $\mathcal{U}$  with  $\widehat{\mathcal{U}}$  turns problem (2.4) into a convex optimization problem and assures the existence of a unique solution. However, this change will likely yield nonbinary optimal designs, that is, the optimal design does not satisfy  $\alpha \in \{0, 1\}$  almost everywhere. In other words, the computed design is not a solution to the original problem of how to distribute the two materials at hand. In order to promote the values 0 and 1 and suppress the intermediate values, we use the so-called SIMP

(solid isotropic material with penalization) approach [2], in which the problem of finding the distribution of the two materials is approximated by the problem

$$(2.5) \quad \begin{aligned} & \min_{\alpha \in \hat{\mathcal{U}}} J_p(\alpha) \\ & \text{subject to } \int_{\Omega} \alpha \, dx \leq V, \end{aligned}$$

where  $J_p(\alpha) = J(\alpha^p)$  is the penalized objective function and  $p$  is a penalty parameter. Note that optimization problem (2.5) is obtained by replacing definition (2.2) of the heat conduction coefficient  $\kappa$  by the relation  $\kappa = \underline{\kappa} + (\bar{\kappa} - \underline{\kappa})\alpha^p$ . SIMP suppresses intermediate values  $0 < \alpha < 1$  when  $p \gg 1$  since intermediate values make much larger contributions to the volume constraint than to the conductivity distribution: intermediate values give a small gain compared to their cost.

The penalty destroys the convexity of the relaxed problem. A common approach to regularizing the SIMP formulation to ensure existence of solutions is the filtering strategy suggested by Bruns and Tortorelli [6], in which the conductivity  $\kappa$  is defined through the penalized convolution

$$(2.6) \quad \kappa(x) = \underline{\kappa} + (\bar{\kappa} - \underline{\kappa}) \left( \int_{\Omega} \sigma(x) \max \left( 0, 1 - \frac{|x-y|}{\tau} \right) \alpha(y) \, dy \right)^p.$$

Parameter  $\tau$  is the filter radius and  $\sigma(x)$  is a function such that

$$\int_{\Omega} \sigma(x) \max \left( 0, 1 - \frac{|x-y|}{\tau} \right) \, dy \equiv 1.$$

The filter introduces a local averaging within a circle of radius  $\tau$  around each point. Moreover, when used together with a penalization approach, the filter radius  $\tau$  limits the minimal width of the structural members [5]. The filter (2.6) is also useful for a different reason, namely, to stabilize the numerical procedure, as we discuss in section 3.1.

### 3. Numerical Approach and Implementation.

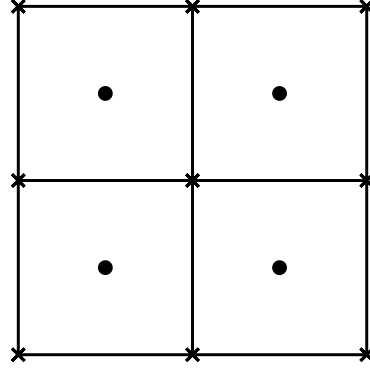
**3.1. Discretization.** We partition the domain  $\Omega$  into  $N = n^2$  squares, where  $n$  is the number of panels in each direction, and use the finite-element method with bilinear elements to solve equation (2.1) numerically. Let  $\mathbf{t} = (t_1, t_2, \dots, t_M)^T$ , where  $M = (n+1)^2$ , be the vector of degrees of freedom, associated with the vertices of the squares, for the temperature field (Figure 3.1). Interpolation with the finite-element shape functions  $\varphi_i$  yields the continuous, piecewise bilinear approximate temperature field

$$T_h(x) = \sum_{i=1}^M t_i \varphi_i(x)$$

at any point  $x \in \Omega$ . We approximate the conductivity  $\kappa$  as well as the design variable  $\alpha$  with functions  $\kappa_h, \alpha_h$  that are constant on each element. Letting  $\mathbf{k} = (k_1, k_2, \dots, k_N)^T$  and  $\mathbf{a} = (a_1, a_2, \dots, a_N)^T$  denote the degrees of freedom for  $\kappa_h$  and  $\alpha_h$  (Figure 3.1), the discrete version of the penalized convolution (2.6) can be written as

$$(3.1) \quad \mathbf{k} = \underline{\kappa} + (\bar{\kappa} - \underline{\kappa}) (\mathbf{F}\mathbf{a})^p,$$

where  $\mathbf{F}$  is a symmetric matrix, with components  $F_{ij}$ , corresponding to the integral in expression (2.6), and the exponentiation of the vector  $\mathbf{F}\mathbf{a}$  is an elementwise operation. The filter radius  $\tau$  is chosen as a fixed multiple of the element size so that



**Fig. 3.1** The crosses indicate the degrees of freedom for the temperature field  $T_h$  and the dots the degrees of freedom for the conductivity  $\kappa_h$  and the design variable  $\alpha_h$

the averaging involves only nearest neighbors. Note that this filter does *not* impose a smallest geometry scale; to impose such a constraint, the filter radius would need to be fixed independently of the element size. Here, we *wish* to capture the increasing scales in the conductivity pattern as the discretization is refined. Our filter acts only as a stabilization to combat an artifact of purely numerical origin, the so-called checkerboard phenomenon that appears when using this choice of finite elements. See section 1.3.2 in the book by Bendsøe and Sigmund [3] for more information about the checkerboard problem.

The linear system resulting from the finite-element discretization of problem (2.1) can be written

$$(3.2) \quad \mathbf{K}(\mathbf{k})\mathbf{t} = \mathbf{f},$$

where matrix  $\mathbf{K}(\mathbf{k})$  and vector  $\mathbf{f}$  have elements

$$K_{ij} = \int_{\Omega} \kappa_h \nabla \varphi_i \cdot \nabla \varphi_j \, dx \quad \text{and} \quad f_i = \int_{\Omega} \varphi_i f \, dx, \quad i, j = 1, \dots, M.$$

To achieve good performance on the GPU it is necessary to keep the computations localized as well as to reduce the number of memory operations. The system matrix  $\mathbf{K}$  is symmetric and positive definite, which admits the use of the preconditioned conjugate gradient (PCG) method to solve linear system (3.2). The PCG algorithm complies well with the requirements of the GPU as long as the preconditioner is well chosen. We use a diagonal preconditioner based on the sum of the conductivities in the elements surrounding each temperature node. The preconditioner compensates for the varying coefficient in the system matrix. The PCG algorithm requires only vector operations, inner products, and the results of matrix–vector products. Moreover, the system matrix is sparse and acts only locally on the immediately surrounding nodes. There is thus no need to explicitly form the system matrix: the elements are computed on the fly while computing the action of the matrix on a vector.

The objective function (2.3) is discretized as

$$J_h(\mathbf{a}) = \mathbf{f}^T \mathbf{t},$$

where  $\mathbf{t}$  solves equation (3.2) with  $\mathbf{k}$  defined as in expression (3.1). Optimization

problem (2.5) is discretized as

$$(3.3) \quad \begin{aligned} & \min_{0 \leq \mathbf{a} \leq 1} J_h(\mathbf{a}) \\ & \text{subject to} \quad \mathbf{a}^T \mathbf{e} \leq NV, \end{aligned}$$

where  $\mathbf{e} = (1, 1, \dots, 1)^T$  is the vector of ones. The gradient of the mapping  $\mathbf{k} \mapsto \mathbf{f}^T \mathbf{t} = j_h(\mathbf{k})$  is locally computable from the solution of equation (3.2) through the expression

$$(3.4) \quad \frac{\partial j_h}{\partial k_n} = -\mathbf{t}^T \mathbf{K}^{(n)} \mathbf{t} \quad \text{with} \quad K_{ij}^{(n)} = \int_{E_n} \nabla \phi_i \cdot \nabla \phi_j \, dx,$$

where  $E_n$  is the  $n$ th element. The gradient of  $J_h$  (that is, the gradient with respect to changes in the design variable  $\mathbf{a}$ ) follows from expression (3.4) and the chain rule applied to relation (3.1):

$$\frac{\partial J_h(\mathbf{a})}{\partial a_n} = \sum_i \frac{\partial j_h(\mathbf{k})}{\partial k_i} (\bar{\kappa} - \underline{\kappa}) p(\mathbf{Fa})_i^{p-1} F_{in} = p(\bar{\kappa} - \underline{\kappa}) \sum_i F_{ni} \frac{\partial j_h(\mathbf{k})}{\partial k_i} (\mathbf{Fa})_i^{p-1},$$

that is,

$$\nabla J_h(\mathbf{a}) = p(\bar{\kappa} - \underline{\kappa}) \mathbf{F} \mathbf{b}, \quad \text{where } \mathbf{b}_n = -\mathbf{t}^T \mathbf{K}^{(n)} \mathbf{t} (\mathbf{Fa})_n^{p-1}, \quad n = 1, \dots, N.$$

We use the so-called optimality criterion method to solve optimization problem (3.3). (For a more detailed exposition of the algorithm, see, for instance, section 1.2 in Bendsøe and Sigmund's book [3].) From the necessary condition of optimality for problem (3.3), it follows that there is a  $\Lambda \geq 0$  such that, for those  $i$  where  $0 < a_i < 1$ ,

$$\frac{\partial J_h}{\partial a_i} + \Lambda = 0;$$

$\Lambda$  is the Lagrange multiplier corresponding to the volume constraint  $\mathbf{a}^T \mathbf{e} \leq NV$ . Moreover, for the current problem, it holds that the volume constraint is active and  $\Lambda$  is strictly positive at optimum.

Each step of the optimality criterion algorithm updates the design variable  $\mathbf{a}$  in order to decrease the objective function while keeping the box and volume constraints satisfied. The update starts by guessing the value of  $\Lambda > 0$ . Then the quantity

$$B_i = -\frac{\partial J_h / \partial a_i}{\Lambda}$$

is checked at each element. From physical reasoning, it can be argued that the value  $B_i$  indicates the relative gain of changing the conductivity at the corresponding element. Loosely speaking, the algorithm attempts to increase  $a_i$  when  $B_i > 1$  and decrease it when  $B_i < 1$ , but only if the box constraints will not be violated. In precise terms, the optimality criterion method applies the following update:

$$(3.5) \quad a_i^{\text{new}} = \begin{cases} \max\{a_i^{\text{old}} - \zeta, 0\} & \text{if } a_i^{\text{old}} B_i^\eta \leq \max\{a_i^{\text{old}} - \zeta, 0\}, \\ \min\{a_i^{\text{old}} + \zeta, 1\} & \text{if } a_i^{\text{old}} B_i^\eta \geq \min\{a_i^{\text{old}} + \zeta, 1\}, \\ a_i^{\text{old}} B_i^\eta & \text{otherwise,} \end{cases}$$

where  $\eta$  is a tuning parameter (to soften the amount of update) and  $\zeta$  is a move limit (see below). The numerical experiments use  $\eta = 0.5$  and  $\zeta = 0.2$ . The first two cases

in (3.5) enforce the box constraint  $0 \leq \mathbf{a} \leq 1$  and impose through  $\zeta$  a limit on the size of the update at each design element. After update (3.5), the volume constraint is checked and the value of the Lagrange multiplier  $\Lambda$  is adjusted with a bisection method: if the volume is too large (small), the Lagrange multiplier is increased (decreased). The update scheme (3.5) and the Lagrange multiplier adjustment alternate until the volume constraint is actively satisfied.

The above updating strategy typically works well when there is only one global constraint on the design variable (the volume constraint). However, the update strategy as motivated above is a heuristic that is not so easily extended to more general problems. For material distribution problems, there are typically far fewer global constraints than there are design variables. In such cases, the use of so-called separable convex approximations together with dual solution methods offers a more systematic approach [9, 25]. The above optimality criterion approach can be viewed as the simplest version of these more systematic approaches [14]. For most of these methods, including the optimality criterion method, there is no guarantee that method will converge; however, some methods (for example, GCMMA [26]) are globally convergent.

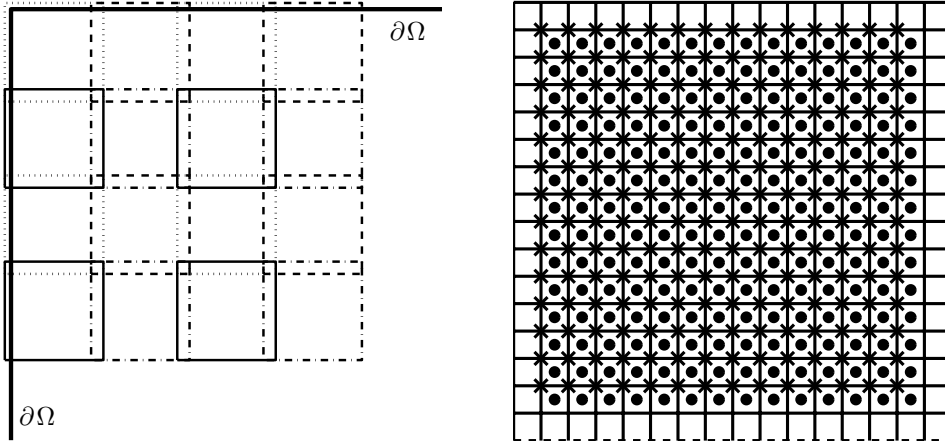
Below we state the main steps of the optimization algorithm. The following section contains a more detailed description of the implementation. In order not to impose any a priori bias, we initially set each component of  $\mathbf{a}$  equal to  $V$  throughout the domain. The optimization problem is then solved numerically by repeating the following steps.

1. COMPUTE THE FILTERED DESIGN  
[Set  $\mathbf{a}_f = \mathbf{F}\mathbf{a}$ ]
2. FIND THE TEMPERATURE DISTRIBUTION  
[Compute  $\mathbf{k} = \underline{\kappa} + (\bar{\kappa} - \underline{\kappa})(\mathbf{a}_f)^p$  (elementwise exponentiation)]  
[Solve  $\mathbf{K}(\mathbf{k})\mathbf{t} = \mathbf{f}$  using the PCG method]
3. COMPUTE THE GRADIENT  
[Compute  $\mathbf{d}\mathbf{a} (= \partial J / \partial \mathbf{a})$  using the values of  $\mathbf{t}$  and  $\mathbf{a}_f$ ]
4. UPDATE THE DESIGN  
[Apply the optimality criterion scheme (3.5)]

The above four steps are iterated until the maximum change over all elements of the design is less than a prespecified tolerance; the numerical experiments used 0.001. For the optimality criterion scheme, this condition is equivalent to requiring that the residual of the first-order necessary optimality conditions (the KKT conditions) is sufficiently small. If two consecutive steps give the same design, then  $B_i = 1$ ; that is,  $\Lambda = -\partial J_h / \partial a_i$  for all  $i$  such that  $0 < a_i < 1$ , and the KKT conditions are satisfied.

**3.2. Implementation.** The numerical experiments presented in the next section are executed using a regular personal computer equipped with an Intel Core 2 Duo processor running at 1.86 GHz and an NVIDIA 8800 GTX-based graphics card. This GPU consists of 16 multiprocessors, each containing 8 processors. Each multiprocessor has a SIMD (single instruction multiple data) architecture, that is, each processor on the multiprocessor executes the same instruction but works on different data. The device memory is mainly divided into a global memory accessible from all multiprocessors, a 16 KB shared memory local on each multiprocessor, and a number of local registers for each processor.

The algorithm presented in the previous section is implemented using NVIDIA's Compute Unified Device Architecture (CUDA) [21], utilizing the level 1 CUBLAS [20] (Basic Linear Algebra Subprograms (BLAS) implemented on the graphics device) routines for the inner products in the PCG algorithm. The CUDA programming



**Fig. 3.2** Left: the domain is divided into a number of overlapping blocks. Right: illustration of a single block.

interface consists of a set of extensions to the C programming language and a runtime library. The CUDA programming model is based on dividing the computational problem into a number of blocks, where each block consists of a number of threads. When the program is executed, each block is processed by one multiprocessor only, which enables fast memory access through the shared memory. Each multiprocessor typically processes a number of blocks simultaneously by time slicing; these blocks jointly utilize the registers and the shared memory on the multiprocessor. On the multiprocessor, the blocks are divided into groups of 32 threads on which the SIMD operations are used. For more information on the programming model, how to set up memory access patterns to minimize latency and avoid bank conflicts, etc., we refer to the programming guide [21].

Below we describe an implementation of the algorithm on the GPU focusing on the main ideas. The computational problem is divided into blocks, as illustrated in Figure 3.2. Each block is responsible for a specific part of the computational domain  $\Omega$ . The left diagram in Figure 3.2 shows a collection of 16 blocks covering the upper left part of the domain. This diagram also illustrates the small overlap between the blocks due to the fact that when computing the matrix–vector product and the filtering, the nodes and elements require information from their nearest neighborhood. The right diagram shows a single block; this block is responsible for updating the elements marked with  $\bullet$  and the nodes marked  $\times$ . Each block also has a number of ghost elements (the unmarked elements) and ghost nodes (the unmarked nodes not on the dashed lines), whose values are required for the update of the values the block is responsible for. The block reads data from all elements and nodes, including the ghosts, but computes and writes the results for the internal elements and nodes only. The blocks bordering on the bottom or right boundary of the domain are the exceptions to this scheme. Within these blocks, data are also written to corresponding boundary nodes. Since each block updates  $14 \times 14$  elements, the discretization size  $N$  is chosen so that a square grid of blocks fills the domain, that is,  $N = (14n)^2$ , where  $n$  is the number of blocks in each direction.

Each block consists of 256 elements and 256 nodes, and is handled by 256 threads. Each thread is assigned one element and the node at its upper left corner (right diagram in Figure 3.2). Whenever computations are performed on a specific block,

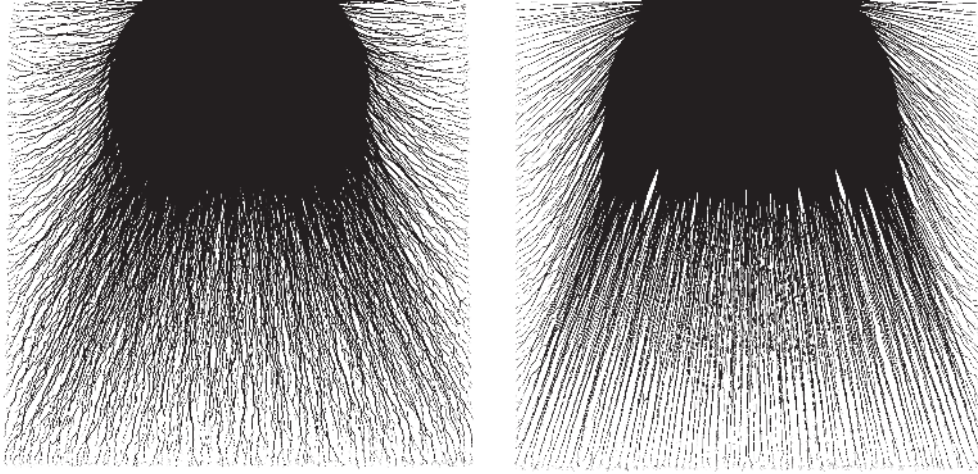
the required nodal and element values are read into the fast shared memory portion of the multiprocessor on which the block is active. The computations are massively parallel; however, the movement of data between the global and shared memory spaces takes a significant part of the execution time, since, in general, only a small fraction of each variable fits into the available fast shared memory. By allocating threads to the ghost elements and ghost nodes, the costly global memory operations can be handled in a perfectly parallel manner at the cost of the threads associated with the ghost cells idling during the computations. An alternative viewpoint is to consider the shared memory as the on-chip cache. In this view, these noncomputational threads are devoted to improving the data-caching efficiency.

One important implementation aspect comes from the combination of the lack of native double precision support on the GPU that we employ, which limits the implementation to single precision floats, and the fact that the conjugate gradient method is notoriously sensitive to roundoff errors. We noted that a naive implementation of the inner product summations in single precision on the CPU is not accurate enough for the PCG method to converge. Fortunately, the fast CUBLAS routines provide sufficiently accurate inner product computations to ensure convergence of the conjugate gradient solver. Note that more recent generations of GPUs, released after completion of the present study, offer native double precision support.

**4. Results.** Figure 4.1 shows two conductivity distributions optimized using a discretization of  $N = 70^2$  elements and allowing high conductivity material to fill half the plate, that is,  $V = 0.5$ . The left image is optimized using a single penalization of  $p = 3$ , while the right image is optimized using a continuation approach for the penalization; that is, the problem is first solved without penalization ( $p = 1$ ), then  $p$  is increased and the optimization problem is solved again using the previously computed solution as a starting guess. In the experiments  $p$  is increased by 0.5 between the optimization rounds and the procedure is repeated until the problem is solved with  $p = 3$ . These two conductivity distributions both successfully even out



**Fig. 4.1** *Material distributions optimized on a discretization of  $N = 70^2$  elements allowing high conductivity material (black) to fill a relative volume fraction  $V = 0.5$  of the plate. Left: using a single penalization level  $p = 3$ . Right: using a continuation approach for the penalization.*

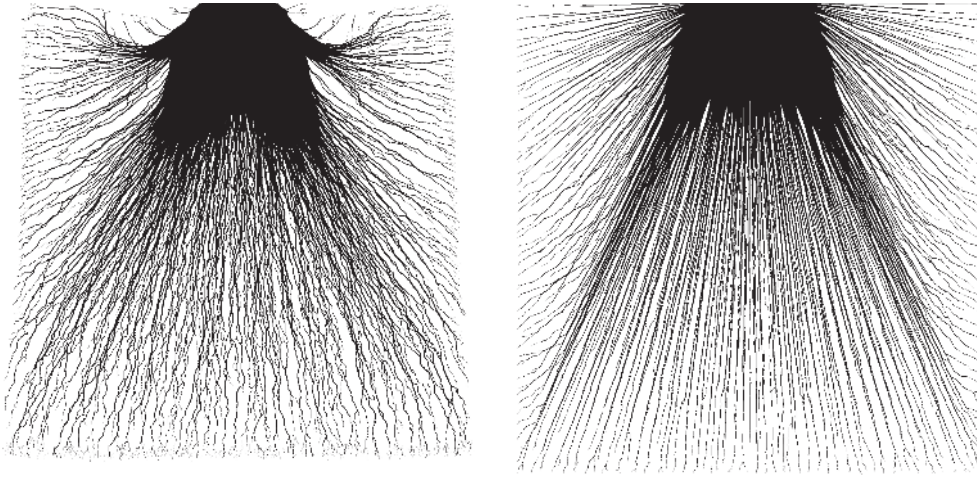


**Fig. 4.2** *Material distributions optimized on a discretization of  $N = 1050^2$  elements allowing high conductivity material (black) to fill a relative volume fraction  $V = 0.5$  of the plate. Left: using a single penalization level  $p = 3$ . Right: using a continuation approach for the penalization.*

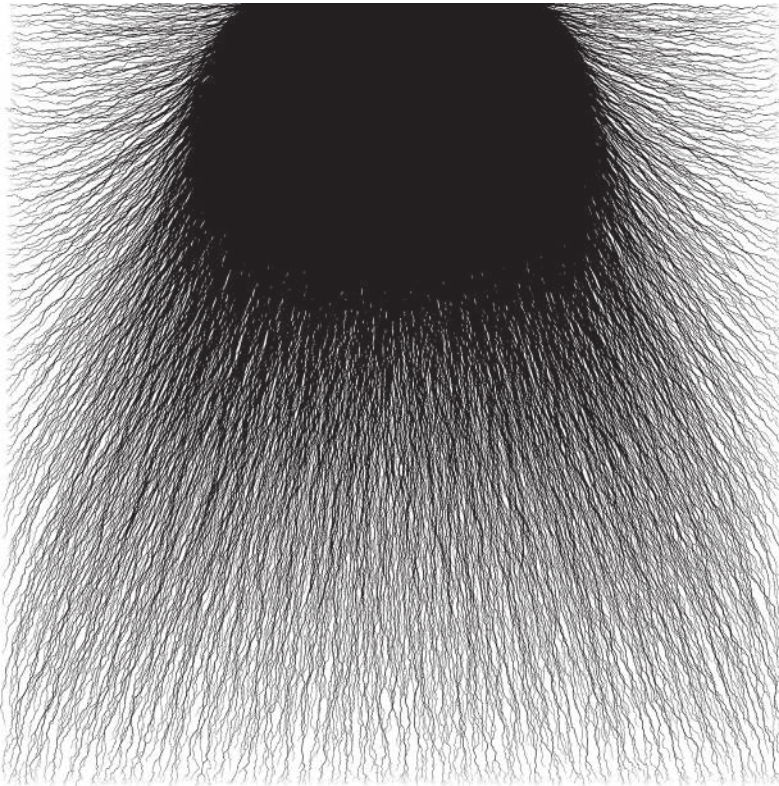
the temperature field and jointly illustrate that the optimization problem does not have a unique solution. Similar results can also be obtained by modifying Sigmund's 99-line Matlab code [24] following the description in the corresponding paper.

Note that there is nothing in the problem formulation stipulating that the optimal design is symmetric. One of the interesting aspects of this nonlinear problem is that the detailed shapes are sensitive to disturbances, and it can be seen that the results are not perfectly symmetric. The nonsymmetry is imposed by the properties of finite-precision arithmetic. That is, even if the conductivity field is perfectly symmetric, the numerically computed temperature field will generally not be symmetric.

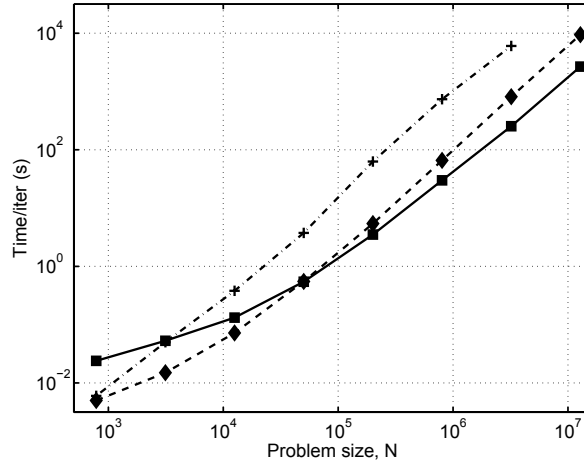
Increasing the resolution to  $N = 1050^2$  results in the distributions depicted in Figure 4.2. The final values of the objective function for the two distributions are essentially the same. Moreover, the two distributions have the same main characteristics. Next to the heat sink both have a large core consisting of high conductivity material, and the amount of high conductivity material decays as the distance to the heat sink increases. Attached to the core are small wires or roots of high conductivity material, extending out to the rest of the domain. The way these roots stretch out from the core differs between the two optimized distributions. The roots in the left image, obtained when using a single penalization of  $p = 3$ , have a wiggly form, while the roots in the right image, obtained using the continuation approach for the optimization, extend straight out toward the boundary. Changing the amount of available high conductivity material to be 30% of the plate area ( $V = 0.3$ ) and still using a discretization of  $N = 1050^2$  elements results in the optimized distributions depicted in Figure 4.3. These distributions show the same main characteristics as the distributions optimized using  $V = 0.5$  (Figure 4.2). As a final example, the image in Figure 4.4 shows the resulting material distribution for  $N = 2100^2$ ,  $V = 0.5$ , and a single penalization level of  $p = 3$ . In this case, the nonlinear optimization problem has over 4 million design variables.



**Fig. 4.3** *Material distributions optimized on a discretization of  $N = 1050^2$  elements allowing high conductivity material (black) to fill a relative volume fraction  $V = 0.3$  of the plate. Left: using a single penalization level  $p = 3$ . Right: using a continuation approach for the penalization.*



**Fig. 4.4** *Material distribution optimized using a single penalization level  $p = 3$  on a discretization of  $N = 2100^2$  elements allowing high conductivity material (black) to fill a relative volume fraction  $V = 0.5$  of the plate.*



**Fig. 5.1** Iteration time in seconds as a function of problem size for the different experimental setups. The solid line illustrates the computational time on the GPU. The dotted line presents the computational time on a single core of the 1.86 GHz Intel Core 2 Duo processor. The dashed line shows the computational time on four cores of a 2.80 GHz AMD Opteron 2220 based HPC cluster.

**5. Discussion.** For small problem sizes, the fastest way of solving the linear system (3.2) on the CPU would be to use a banded direct method requiring  $O(N^2)$  operations and  $O(N^{3/2})$  memory. Direct methods become increasingly costly as the problem grows. For example, the solution of the linear system corresponding to the optimization using  $N = 2100^2$  elements (Figure 4.4) would require 35 GB of memory. On the other hand, the memory requirements of the conjugate gradient solver grow as  $O(N)$ , and the expected running time is  $O(N^{3/2})$ . Each step requires  $O(N)$  operations, and the number of iterations required is expected to grow as  $O(N^{1/2})$  due to the increase in the condition number of the system. That is, the conjugate gradient method is both faster and less memory consuming for large problems.

To compare the running time for the algorithm on the GPU with typical CPU-based implementations, we also implement a serial version of the algorithm as well as an OpenMP parallelized version. We run the GPU and the single core CPU versions of the algorithm on a workstation equipped with an 1.86 GHz Intel Core 2 Duo processor and an NVIDIA 8800 GTX-based graphics card. The parallel version of the algorithm is executed on an AMD Opteron 2220 (2.80 GHz) based HPC machine. Figure 5.1 shows the time per iteration for the three experimental setups described above. This estimate compares the running time for the first 10 iterations of the main optimization loop using the implementation discussed in section 3.2 for the GPU and an unblocked implementation in single precision—except for the summations in the conjugate gradient method, which are performed in double precision—of the same algorithm for the CPUs.

The GPU-based version of the code starts to perform faster than the CPU-based version at problem sizes larger than  $N = 56^2$ , and is faster than the parallel version at problem sizes larger than  $N = 224^2$ . A comparison, for the larger problems, between the running time of the algorithm using the GPU and using a single core of the CPU shows that the implementation using the GPU is about 20 times faster than the CPU implementation and about 3 times faster than the parallelized version running on the HPC machine.

A closer look at the asymptotics for the time per iteration reveals that the single core CPU follows the asymptotic growth of  $O(N^{3/2})$ . The parallel CPU-based version attains this asymptotic growth once the additional overheads, such as startup latency and communications, are small relative to the computation cost. In contrast, the time per iteration for the GPU follows a dogleg-shaped curve. The smallest problem consists of only four blocks, which leaves most of the multiprocessors on the GPU idle. The next problem size also fits on the GPU; thus, the same number of operations are required to compute each matrix–vector product in the two smallest problems. The increase in computational time for the smaller problems is mostly due to the  $O(N^{1/2})$  growth of the conjugate gradient iterations. As the problems get larger, the time per iteration on the GPU follows the expected  $O(N^{3/2})$  growth.

**6. Conclusions.** Material distribution problems are often posed on uniform meshes, which easily map onto the GPU, to avoid a priori bias between different parts of the domain. To allow the use of the conjugate gradient algorithm, the system matrix needs to be symmetric and at least positive semidefinite. There are several other problems of scientific and engineering interest that share that particular problem structure, most notably topology optimization problems for linearly elastic structures [3]. However, material distribution problems for other applications, for instance, in the context of wave propagation and fluid flow, do not yield symmetric and positive definite system matrices. In these cases, the availability of suitable solution algorithms for the governing equations is a crucial issue when determining whether a material distribution problem would benefit from a GPU acceleration.

The GPU can provide high computing performance per price and energy unit for material distribution problems. To utilize the computational power of the GPU the problem needs to be formulated in terms of data-parallel tasks. Structural topology optimization problems are amenable to data parallelism and can be treated analogously to the heat conduction problem of this paper. The acoustics problem is much more difficult; the challenge here consists of finding efficient data-parallel solvers for the Helmholtz equation.

**Acknowledgments.** The authors wish to thank Bo Kågström and the anonymous reviewers for valuable comments on the manuscript.

## REFERENCES

- [1] A. BEJAN, *Shape and Structure, from Engineering to Nature*, Cambridge University Press, Cambridge, UK, 2000.
- [2] M. P. BENDSØE, *Optimal shape design as a material distribution problem*, Struct. Optim., 1 (1989), pp. 193–202.
- [3] M. P. BENDSØE AND O. SIGMUND, *Topology Optimization. Theory, Methods, and Applications*, Springer, Berlin, 2003.
- [4] J. BOLZ, I. FARMER, E. GRINSUN, AND P. SCHRÖDER, *Sparse matrix solvers on the GPU: Conjugate gradients and multigrid*, ACM Trans. Graphics, 22 (2003), pp. 917–924.
- [5] T. BORRVAL, *Topology optimization of elastic continua using restriction*, Arch. Comput. Methods Engrg., 8 (2001), pp. 351–385.
- [6] T. E. BRUNS AND D. A. TORTORELLI, *Topology optimization of non-linear elastic structures and compliant mechanisms*, Comput. Methods Appl. Mech. Engrg., 190 (2001), pp. 3443–3459.
- [7] T. DOKKEN, T. R. HAGEN, AND J. M. HJELMERVI, *The GPU as a high performance computational resource*, in Proceedings of the 21st Spring Conference on Computer Graphics (Budmerice, Slovakia), B. Jüttler, ed., ACM Press, New York, 2005, pp. 21–26.
- [8] A. DONOSO AND O. SIGMUND, *Topology optimization of multiple physics problems modelled by Poisson’s equation*, Latin Amer. J. Solids Structures, 1 (2004), pp. 169–184.
- [9] C. FLEURY, *CONLIN: An efficient dual optimizer based on convex approximation concepts*, Struct. Optim., 1 (1989), pp. 81–89.

- [10] A. GERSBORG-HANSEN, M. P. BENDSØE, AND O. SIGMUND, *Topology optimization of heat conduction problems using the finite volume method*, Struct. Multidiscip. Optim., 31 (2006), pp. 251–259.
- [11] D. GÖDDEKE, R. STRZODKA, AND S. TUREK, *Performance and accuracy of hardware-oriented native-, emulated- and mixed-precision solvers in FEM simulations*, Int. J. Parallel Emergent Distrib. Syst., 22 (2007), pp. 221–256.
- [12] N. GOODNIGHT, C. WOOLLEY, G. LEWIN, D. LUEBKE, AND G. HUMPHREYS, *A multigrid solver for boundary value problems using programmable graphics hardware*, in HWWS '03: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware, M. Doggett, W. Heidrich, W. Mark, and A. Schilling, eds., Eurographics Association, Aire-la-Ville, Switzerland, 2003, pp. 102–111.
- [13] G. D. GRAÇA AND D. DEFOUR, *Implementation of float-float operations on graphics hardware*, in Proceedings of the 7th Conference on Real Numbers and Computers (RNC7), G. Hanrot and P. Zimmermann, eds., 2006, pp. 23–32.
- [14] A. A. GROENWOLD AND L. F. P. ETMAN, *On the equivalence of optimality criterion and sequential approximate optimization methods in the classical topology layout problem*, Internat. J. Numer. Methods Engrg., 73 (2008), pp. 297–316.
- [15] M. HARRIS, G. COOMBE, T. SCHEUERMANN, AND A. LASTRA, *Physically-based visual simulation on graphics hardware*, in HWWS '02: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware, Eurographics Association, Aire-la-Ville, Switzerland, 2002, pp. 109–118.
- [16] K. HILLESLAND AND A. LASTRA, *GPU floating-point paranoia*, in GP2 2004 ACM Workshop on General Purpose Computing on Graphics Processors, A. Lastra, M. Lin, and D. Manocha, eds., 2004, pp. C–8.
- [17] K. E. HILLESLAND, S. MOLINOV, AND R. GRZESZCZUK, *Nonlinear optimization framework for image-based modeling on programmable graphics hardware*, ACM Trans. Graphics, 22 (2003), pp. 925–934.
- [18] J. KRÜGER AND R. WESTERMANN, *Linear algebra operators for GPU implementation of numerical algorithms*, ACM Trans. Graphics, 22 (2003), pp. 908–916.
- [19] A. LEFOHN, J. KNISS, C. HANSEN, AND R. WHITAKER, *Interactive deformation and visualization of level set surfaces using graphics hardware*, in Proceedings of the 14th IEEE Visualization Conference (VIS'03), IEEE Computer Society Press, Los Alamitos, CA, 2003, pp. 75–82.
- [20] NVIDIA CORP., *CUDA CUBLAS Library*, Ver. 1.1, NVIDIA Corp., Santa Clara, CA, 2007. Available online at [http://www.nvidia.com/object/cuda\\_develop.html](http://www.nvidia.com/object/cuda_develop.html).
- [21] NVIDIA CORP., *NVIDIA CUDA Compute Unified Device Architecture Programming Guide*, Ver. 1.1, NVIDIA Corp., Santa Clara, CA, 2007. Available online at [http://www.nvidia.com/object/cuda\\_develop.html](http://www.nvidia.com/object/cuda_develop.html).
- [22] J. D. OWENS, D. LUEBKE, N. GOVINDARAJU, M. HARRIS, J. KRÜGER, A. E. LEFOHN, AND T. J. PURCELL, *A survey of general-purpose computation on graphics hardware*, Comput. Graphics Forum, 26 (2007), pp. 80–113.
- [23] M. RUMPF AND R. STRZODKA, *Level set segmentation in graphics hardware*, in Proceedings of the 2001 International Conference on Image Processing (ICIP'01), Vol. 3, IEEE, 2001, pp. 1103–1106.
- [24] O. SIGMUND, *A 99 line topology optimization code written in MATLAB*, Struct. Multidiscip. Optim., 21 (2001), pp. 120–127.
- [25] K. SVANBERG, *The method of moving asymptotes—a new method for structural optimization*, Internat. J. Numer. Methods Engrg., 24 (1987), pp. 359–373.
- [26] K. SVANBERG, *A class of globally convergent optimization methods based on conservative convex separable approximations*, SIAM J. Optim., 12 (2002), pp. 555–573.
- [27] E. WADBRO AND M. BERGGREN, *Topology optimization of an acoustic horn*, Comput. Methods Appl. Mech. Engrg., 196 (2006), pp. 420–436.
- [28] E. WADBRO AND M. BERGGREN, *Topology optimization of wave transducers*, in IUTAM Symposium on Topological Design Optimization of Structures, Machines and Materials, M. P. Bendsøe, N. Olhoff, and O. Sigmund, eds., Springer, Dordrecht, The Netherlands, 2006, pp. 301–310.
- [29] E. WADBRO AND M. BERGGREN, *Microwave tomography using topology optimization techniques*, SIAM J. Sci. Comput., 30 (2008), pp. 1613–1633.