Final Performance Report

Contract # **FA9550-08-C-0006**:

Hearing Protection for High-Noise Environments

Period of Performance: **Oct 01, 2007 – Nov 30, 2009**

Attachment 4

Parallelization of the Acoustic Integral-Equation Solver and Example Applications

Prepared by:

MONOPOLE RESEARCH
739 Calle Sequoia, Thousand Oaks, CA 91360
tel: (805) 375-0318 fax: (805) 499-9878

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
2009		**01-10-2007 to 30-11-2009**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Parallelization of the Acoustic Integral-Equation Solver and Example Applications	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **MONOPOLE RESEARCH,739 Calle Sequoia,Thousand Oaks,CA,91360**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**	**Same as Report (SAR)**	**41**	

Contents

1 Introduction **1**

2 The FFT-based matrix compression scheme **4**
2.1 Problem discretization and iterative solution 4
2.2 Compressed stiffness matrix representation 5
2.3 Fast matrix-vector multiplication 6

3 Parallel algorithm implementation **8**
3.1 Data partition 8
3.1.1 Distribution of the Cartesian vectors 8
3.1.2 Distribution of the geometry data 10
3.1.3 Distribution of the MoM unknowns 11
3.1.4 Distribution of the near-field matrix 11
3.1.5 Distribution of the far-field matrix data 11
3.2 Distribution of the computational work 12
3.2.1 Geometry processing 13
3.2.2 Matrix construction 13
3.2.3 Matrix-vector multiplication 15

4 Computational complexity estimates **17**

5 Storage estimates **17**

6 Examples of large-scale computations **22**
6.1 The geometrical models and material properties 22
6.2 Analysis of the accuracy of the far-field expansion in the problem of the human head helmet with cork filling 24
6.3 Pressure distributions in a human head model: dependence on the discretization 28
6.4 Pressure distributions in a human head model in the presence of an unattached helmet 28
6.5 Pressure distributions in a human head model in the presence and absence of a helmet with a cork lining 29
6.6 Scaling in parallel computations 31

A Handling of zero padding and communication in parallel implementation of FFTs **34**

References **38**

List of Figures

1 Partition of the Cartesian grid and the object geometry into disjoint z-slices 10
2 Model of the external head surface and the helmet 22
3 Averaged (over distance bins) relative far-field errors in the surface (S) and volume (V) problems for the range parameter $R_{\mathrm{N}}/h = 12$, plotted as functions of the distance between basis function supports, measured in units of the Cartesian grid spacing h. The plots represent output of the processor $p = 59$ with the assigned $N^{(p)} \simeq 60,000$ unknowns. 25
4 Convergence (the relative residual norm) of iterative solutions in the surface (S) and volume (V) problems for the indicated values of the near-field range parameter R_{N}/h. 26
5 Distributions of the real part of pressure on the external surface of the model of the human head, the helmet, and the cork filling, with $N \simeq 4,700,000$ unknowns. The distributions, plotted in linear scale, were obtained for the same values of the near-field range as in Fig. 4, i.e., (a) $R_{\mathrm{N}}/h = 6$, (b) $R_{\mathrm{N}}/h = 8$, (c) $R_{\mathrm{N}}/h = 10$, (d) $R_{\mathrm{N}}/h = 12$. 26
6 Distributions of the absolute value of pressure in the coronal section of the model of the human head, the helmet, and the cork filling, with $N \simeq 4,700,000$ unknowns. The distributions, plotted in linear scale, were computed for the same values of the near-field range as in the previous Figures, i.e., (a) $R_{\mathrm{N}}/h = 6$, (b) $R_{\mathrm{N}}/h = 8$, (c) $R_{\mathrm{N}}/h = 10$, (d) $R_{\mathrm{N}}/h = 12$. 27
7 Distributions of the real part of pressure in the coronal section of the model of the human head, the helmet, and the cork filling, with $N \simeq 4,700,000$ unknowns. The distributions, plotted in logarithmic scale, were obtained for for the same values of the near-field range before, i.e., (a) $R_{\mathrm{N}}/h = 6$, (b) $R_{\mathrm{N}}/h = 8$, (c) $R_{\mathrm{N}}/h = 10$, (d) $R_{\mathrm{N}}/h = 12$. 27
8 Distributions of the real part of the pressure on several section in the axial plane of the human head model 28
9 Distribution of the real part of the pressure on several sections in the axial plane of the human head model with an unattached helmet 29
10 Pressure distributions in the coronal plane for (i) the human head model and (ii) the system consisting of the human head and a steel helmet models . . 31
11 Matrix-vector multiplication time as a function of the number of Cartesian grid nodes 33
12 FFT in the (x, y) directions of the distributed data (a), followed by transposition and FFT in the z direction (b) 34
13 Original FFT data, their rearranged storage, and the result of their transposition by means of the MPI all-to-all communication routine 37

1 Introduction

We have recently developed a fast iterative volumetric integral-equation solver for acoustic scattering and propagation problems involving inhomogeneous media. The solver incorporates a Fast Fourier Transform (FFT)-based matrix compression technique (AIM) and the corresponding fast matrix-vector multiplication algorithm [1, 2]. The method consists, essentially, of the method-of-moments (MoM) computation of the near-field interactions involving physical sources and fields, and evaluation of the far-field by means of the FFTs operating on equivalent source and field distributions defined on nodes of a regular Cartesian grid. The FFT-based matrix compression is particularly well suited to volumetric discretization (with a fairly uniform discretization) and to sub-wavelength problems, i.e., spatial resolution scale much smaller than the wavelength. Both of these features are characteristic of realistic anatomical models and propagation problems in the biological applications we are considering.

For a problem with N unknowns (in this implementation, N tetrahedra in the volumetric mesh) the total storage is $\mathcal{O}(N)$ and the solution complexity as $\mathcal{O}(N \log N)$ times the number of iterations (which may be substantial, but, in practice, several orders of magnitude smaller than the number of unknowns). Thus, scaling of our solution method is similar to that of differential-equation (finite-element) solvers. We have also carefully optimized the choice of the matrix compression parameters, in order to minimize the total cost of near- and far-field computations.

Nevertheless, in application to large problems (millions of unknowns), parallelization of the solver is essential. We describe here such a parallel implementation developed for distributed-memory systems based on the message passing interface (MPI).

The most essential part of the matrix compression and fast matrix-vector multiplication procedures is the FFT algorithm. Its distributed-memory parallelization is nontrivial, since the nature of the algorithm gives rise a large amount of inter-processor communication: even if the data are initially partitioned across the processors, the fact that Fourier transforms have to be evaluated in all the spatial coordinate directions requires a global rearrangement ("transposition") of the data. Computation of the near-field contributions, on the other hand, is to large extent local, and requires only a modest amount of communication.

For these reasons we build the parallelization scheme of the code around the parallel FFT algorithm. We take here advantage of the availability of a widely used FFT package, FFTW, which allows operations on FFT data distributed spatially (in the form of "slices") across the processors. Its MPI-parallelized implementation is available in both the current version 3 and the previous version 2. We opted for the more recent MPI alpha version 3.2alpha3 (recently, in November 2008, this version has been replaced by 3.3alpha1).

Most of the MPI-based operations are transparent to the package user. However, a literal application of the FFTW routines to carry out a global distributed FFT, would have been, in our case, rather inefficient. The main reason is as follows:

(i) The far-field computation amounts simply to a convolution of the given distribution of sources, defined on a Cartesian grid, (we describe them by a vector X) with a Green function G. This convolution can be evaluated in terms of FFTs. However, in order to avoid aliasing due to periodicity of the FFT (as a discrete Fourier transform), the "physical" region on which X is defined has to be enlarged by the factor of two in all directions, and padded

with zeros.

(ii) In a parallel implementation the padded data X are partitioned into slices (say, in the direction of the z axis) and distributed one slice per processor. Hence, about one-half of the processors handles only the padding region.

(iii) Such a distribution of data does not impair in any way execution of the FFTs. However, the vector multiplication by the compressed matrix includes also near-field operations which involve the physical sources (say, x) and field, which are coupled only to the "physical" part of the vector X, which resides only on one-half of the processors. Those near-field computations involve matrix blocks which have substantial sizes (especially in a volumetric problem). One possibility is to store those blocks only on that half of processors which handle the physical part of the vector X; in this case those processors are overloaded with storage, while the memory resources of the remaining processors are not fully used. Another possibility is to reorganize the near-field matrix blocks and distribute them evenly across the processors; this scheme, however, requires an additional (and computationally expensive) communication between practically all processors before and after near-field computation.

In view of the above difficulties, we devised an alternative distribution of FFT data, in which each processor stores equal parts of the physical part of the Cartesian vector X and the padding region. With this "interleaved" storage, the physical sources x and near-field matrix blocks can also be uniformly distributed across the processors, and no extra communication costs are incurred. Additional advantages of the "interleaved" data layout is a reduction, by a factor of about two, in the amount of communication in the parallel FFT (this aspect is discussed in Appendix A) and the fact that it allows us to directly utilize the collective MPI communication routine `MPI_Alltoallv`, which is often highly optimized for specific interconnect networks.

We also have to note here two potential drawbacks of the interleaved storage:

(a) Each processor has to store not only the "physical" part of its Cartesian grid, but also the padding area. In the alternative scheme (with separate physical grid and padding storage) the full (padded) grid can be distributed into twice as many processors.

(b) The FFTs cannot be computed entirely in-place, and hence an additional communication-related buffer is needed.

We address these problems in Section 5 and show that, in practice, they do not lead to difficulties.

Scaling of a parallel implementation of the solver for large problems requires distributing of practically ***all sizable data*** across the processors, with a minimal, if any, replication; at the same time, processors must have access to some non-local geometry data in order to evaluate near-field couplings between sources and field variables assigned to different processors. We solve the problem by (i) partitioning the geometry into non-overlapping slices corresponding to the Cartesian grid slices assigned to the processors, but (ii) in the matrix construction stage temporarily combining ("welding"), on each processor, several slices into a single "stack" geometry of thickness equal at least to the near-field range. This "welding" procedure, which can be realized in a straightforward way in the "interleaved" storage approach, allows us to treat the entire stack of slices as a conventional bona fide

complete geometry, without having to introduce any additional connectivity data relating adjacent or identical geometry elements in different slices (which would have been necessary if slices were treated as separate geometries).

We describe here the details of parallelization in the present code, which assumes basis functions associated with tetrahedra. However, these parallelization techniques are applicable, with only minor modifications, to various other types of geometry discretization, e.g., to basis functions associated with nodes of the mesh.

2 The FFT-based matrix compression scheme

We give here only a brief summary of the FFT-based compression method used in our solver. More details can be found in Refs. [2] and [3].

2.1 Problem discretization and iterative solution

We solve, iteratively (by using one of the minimum-residual algorithms, GMRES or GCR [4, 5]), the linear system

$$A\,x = b \tag{2.1}$$

obtained by Galerkin discretization of the acoustic Lippmann-Schwinger volumetric integral equation for the pressure p,

$$K\,p = p^{\mathrm{inc}}\ , \tag{2.2}$$

describing scattering of an incident pressure wave p^{inc} on a certain bounded region Ω filled with an inhomogeneous material characterized by a density $\rho(\mathbf{r})$ and the compressibility $\kappa(\mathbf{r})$. According to the nature of the L-S equations, the Green function appearing in the integral operators is always that of the background (surrounding) space – here air. Correspondingly, the equations involve ratios of the scatterer density and compressibility to those of the background medium.

In fact, in the case of large density contrasts, we use a modified integral-equation formulation described in Ref. [3]. It involves three different operators, associated with a surface problem (defined on high-contrast surfaces), with a coupling between that surface and the volume region, and with a volume problem. Explicit expressions for these operators are also given in Ref. [2].

The vector x consists of coefficient of expansion of the pressure p into a set of local basis functions ϕ_α, and b is a vector obtained by projecting the incident field p^{inc} on the same set of basis functions. The elements of the stiffness matrix A are given by the MoM expression

$$A_{\alpha\beta} = (\phi_\alpha\,,\,K\,\phi_\beta) = \int \mathrm{d}^3 r_1\,\mathrm{d}^3 r_2\,\phi_\alpha(\mathbf{r}_1)\,\hat{K}(\mathbf{r}_1,\mathbf{r}_2)\,\phi_\beta(\mathbf{r}_2)\ , \tag{2.3}$$

where $\hat{K}$ is one of the (non-symmetric) kernels arising in our system of the integral equations. Similarly, the elements of the r.h.s. vector b are

$$b_\alpha = (\phi_\alpha\,,\,p^{\mathrm{inc}}) = \int \mathrm{d}^3 r\,\phi_\alpha(\mathbf{r})\,p^{\mathrm{inc}}(\mathbf{r})\ . \tag{2.4}$$

The present implementation of the code assumes constant basis functions supported on tetrahedra, hence the indices α and β are tetrahedron numbers.[1] We assume that the material properties of the medium (its density ρ and compressibility κ) are constant on the individual tetrahedra.

[1] The general structure of the code and the compression algorithm remains unchanged for other types of basis functions, e.g., piecewise-linear functions associated with vertices.

2.2 Compressed stiffness matrix representation

Since, in an iterative solver, most operations amount to matrix-vector multiplication, a fast solution can be achieved through matrix compression and a fast matrix-vector multiplication algorithm.

The FFT-compressed matrix A is stored as

$$A_{\alpha\beta} \simeq A^{\mathrm{Near}}_{\alpha\beta} + A^{\mathrm{Far}}_{\alpha\beta} \ . \tag{2.5}$$

The near-field part of the matrix, coupling tetrahedra up to the relative distance of some ***near-field range*** R_{N}, is computed by using the conventional MoM, i.e., Eq.(2.3) and, according to the usual AIM prescription [1, 2], subtracting the far-field contributions (as defined below). At larger distances the couplings are described by the far-field part of the matrix, having the general structure

$$A^{\mathrm{Far}}_{\alpha\beta} = \sum_{\substack{\mathbf{u}\in\mathcal{B}(\alpha)\\ \mathbf{v}\in\mathcal{B}(\beta)}\ \mathbf{u}\neq\mathbf{v}} V^{(1)}_{\alpha\mathbf{u}}\ G(\mathbf{u}-\mathbf{v})\, V^{(2)}_{\beta\mathbf{v}} \ , \tag{2.6}$$

where $\mathbf{u}$ and $\mathbf{v}$ are nodes of a global Cartesian grid $\mathcal{C}$ enclosing the object, and $\mathcal{B}(\alpha)$ is a cubic set of $(M+1)^3$ grid nodes at which equivalent sources associated with the basis function α are located; we call this set the ***expansion box*** of the basis function α (and similarly for $\mathcal{B}(\beta)$). The parameter M is the multipole expansion order of the far field; the typical value $M = 2$ corresponds to octupole expansion. For each tetrahedron α its expansion box $\mathcal{B}(\alpha)$ is defined as that set of $(M+1)\times(M+1)\times(M+1)$ grid nodes, whose center is closest to the centroid of the tetrahedron.

The expansion coefficients $V^{(1)}_{\alpha\mathbf{u}}$ and $V^{(2)}_{\beta\mathbf{v}}$ relate the original MoM basis functions and sets of the equivalent sources, and also depend on the structure of the integral operator in question. These coefficients are determined by requiring that the original basis functions and the corresponding sets of equivalent sources have the same multipole moments up to some expansion order M. Finally, $G(\mathbf{u}-\mathbf{v})$ are the tabulated values of the (scalar) Green function of the Helmholtz equation with the incident field wave number k (describing wave propagation in the background medium).

Our formulation for high-contrast problems involves matrix elements of two types, "monopole" (M) and "dipole" (D), for which the representation (2.6) has the form

$$A^{\mathrm{M\,Far}}_{\alpha\beta} = k^2 \sum_{\mathbf{u},\mathbf{v}} V^{(\mathrm{M})}_{\alpha\mathbf{u}}\ G(\mathbf{u}-\mathbf{v})\, V^{(\mathrm{M})}_{\beta\mathbf{v}} \left(\frac{\rho_0}{\rho_\beta} - \frac{\kappa_\beta}{\kappa_0}\right) , \tag{2.7a}$$

$$A^{\mathrm{D\,Far}}_{\alpha\beta} = -\sum_{\mathbf{u},\mathbf{v}} \mathbf{V}^{(\mathrm{D})}_{\alpha\mathbf{u}} \cdot\ G(\mathbf{u}-\mathbf{v})\, \mathbf{V}^{(\Lambda)}_{\beta\mathbf{v}} \ , \tag{2.7b}$$

(Eqs. (21) and (24) of Ref. [2]). It involves three different types of expansion coefficients, with one of them, $\mathbf{V}^{(\Lambda)}_{\beta\mathbf{v}}$, depending on the discontinuities Λ of the density across facets of the tetrahedron β; similarly, ρ_β and κ_β are the density and compressibility of the tetrahedron β, and ρ_0 and κ_0 the corresponding parameter values in the background space (air). The vector-valued expansion coefficients, $\mathbf{V}^{(\mathrm{D})}_{\beta\mathbf{v}}$ and $\mathbf{V}^{(\Lambda)}_{\beta\mathbf{v}}$, represent the gradient of the pressure.

Having specified the far-field representations (2.6) or (2.7) of the matrix elements, we define the near-field component of the matrix as a difference between the actual (MoM) matrix elements and their far-field approximations,

$$A^{\rm Near}_{\alpha\beta} = \begin{cases} A_{\alpha\beta} - A^{\rm Far}_{\alpha\beta} & \text{for } |\mathbf{c}_\alpha - \mathbf{c}_\beta| \le R_{\rm N} \ , \\ 0 & \text{otherwise} \ , \end{cases} \tag{2.8}$$

where $\mathbf{c}_\alpha$ and $\mathbf{c}_\beta$ are centroids of the supports of the basis functions ϕ_α and ϕ_β, and $R_{\rm N}$ is a pre-defined near-field range. Subtraction of the far-field matrix element $A^{\rm Far}_{\alpha\beta}$ (mentioned following Eq.(2.5)) is necessary in view of the fact that, in matrix-vector multiplication, the far-field contributions are being added irrespectively of the separation between the basis functions α and β (this is required by the Toeplitz property of the compressed matrix) and, clearly, $A^{\rm Far}_{\alpha\beta}$ is not a good approximation to $A_{\alpha\beta}$, hence the subtraction compensates for the resulting error.

According to Eq.(2.8), for distances exceeding $R_{\rm N}$, we neglect the difference between the matrix element $A_{\alpha\beta}$ and its far-field approximation $A^{\rm Far}_{\alpha\beta}$. Therefore, the near-field range has to be large enough to ensure an acceptably small relative error of the approximation,

$$\mathcal{E}_{\alpha\beta} := \frac{|A_{\alpha\beta} - A^{\rm Far}_{\alpha\beta}|}{|A_{\alpha\beta}|} \ , \tag{2.9}$$

for $|\mathbf{c}_\alpha - \mathbf{c}_\beta| > R_{\rm N}$. Usually, an error of 1 % to 2 % can be considered adequate.

In the following we denote the average tetrahedron size (edge length) by a, and the spacing of the Cartesian grid by h. Our experience with optimizing compression parameters (the main results of which were reported in Ref. [2]) has shown that preferred values of the grid spacing is relatively small, $h \lesssim a/2$, the optimal expansion order is $M = 2$, and the near-field range can be chosen as $R_{\rm N} \simeq 3\,h$ for the monopole matrix elements and $R_{\rm N} \simeq 6\,h$ for the dipole elements (Eqs. (2.7)). With these parameters the relative error in the far-field matrix elements should be on the level of about 1 %.

Eqs. (2.5) and (2.6) can be interpreted as providing a compressed matrix representation as a sum of the near-field matrix (which, by construction, is sparse) and a far-field matrix given as a product of two sparse matrices V and a Toeplitz matrix G. For N unknowns and $N_{\mathcal{C}} \sim N$, and for a fixed near-field range $R_{\rm N}$, all these matrices require storage proportional to N; hence the entire storage is also $\mathcal{O}(N)$.

2.3 Fast matrix-vector multiplication

With the compressed matrix representation of Eqs. (2.5) and (2.6), matrix-vector multiplication $y = A\,x$ amounts to computing

$$y \equiv y^{\,\rm Near} + y^{\,\rm Far} = A^{\rm Near}\,x + V\,G\,V^{\rm T}\,x \ . \tag{2.10}$$

While the near-field evaluation involves simply a vector multiplication by a sparse matrix, the far-field contribution is obtained by

1. Accumulating contributions of basis functions, weighted by the components x_β of the vector x, to the equivalent sources X on the Cartesian grid nodes $\mathbf{v} \in \mathcal{C}$,

$$X_{\mathbf{v}} \mathrel{+}= V^{\mathrm{T}}_{\mathbf{v}\beta}\, x_\beta \qquad \text{for } \mathbf{v} \in \mathcal{B}(\beta)\ . \tag{2.11}$$

   This operation is carried out in a loop over β, and in each step contributions to all relevant grid nodes $\mathbf{v}$ are computed and added. As indicated in Eq.(2.7), the vector X and the coefficients V have, in our case, four components, describing the pressure and its gradient.

2. Taking its Fourier transform

$$\widetilde{X}_{\mathbf{q}} = \sum_{\mathbf{v}\in\mathcal{C}} \mathcal{F}_{\mathbf{qv}}\ X_{\mathbf{v}}\ . \tag{2.12}$$

3. Multiplying $\widetilde{X}$ by the Fourier transform of the Green function,

$$\widetilde{Y}_{\mathbf{q}} = \widetilde{G}_{\mathbf{q}}\, \widetilde{X}_{\mathbf{q}}\ , \tag{2.13}$$

   for all points $\mathbf{q}$ of the dual grid $\widetilde{\mathcal{C}}$.

4. Taking the inverse Fourier transform

$$Y_{\mathbf{u}} = \sum_{\mathbf{q}\in\widetilde{\mathcal{C}}} \mathcal{F}^{-1}_{\mathbf{uq}}\ \widetilde{Y}_{\mathbf{q}}\ . \tag{2.14}$$

5. Finally, converting back the Cartesian-grid field distribution Y to its MoM representation y,

$$y^{\mathrm{Far}}_\alpha = \sum_{\mathbf{u}\in\mathcal{B}(\alpha)} V_{\alpha\mathbf{u}}\, Y_{\mathbf{u}}\ . \tag{2.15}$$

Physically, X in the above expressions represents the equivalent sources on the Cartesian grid $\mathcal{C}$, and Y is the field generated on the same Cartesian grid by the sources X. We recall that the vectors X and Y have four components, while G is a scalar.

In the following we refer to the sets of equivalent sources, fields, or the Green function values, and to their discrete Fourier transforms, i.e., to the vectors X, Y, G, $\widetilde{X}$, $\widetilde{Y}$, or $\widetilde{G}$, as ***Cartesian vectors*** (all of them are denoted by capital letters). We note that in our implementation FFT operations are executed in-place,[2] hence the vectors X, Y, G, $\widetilde{X}$, and $\widetilde{Y}$ share the same storage area.

[2]However, as we mentioned in the Introduction and discuss later, communication in the parallel FFT requires an additional storage buffer. Its size is one-half of the vector X, since zero-padding in one direction is not needed.

3 Parallel algorithm implementation

A parallel implementation of the solver for large problems requires distributing of practically ***all data*** across the processors, with a minimal, if any, replication. Similarly, the computational work should be distributed, as evenly as possible, across all the processors. Our implementation achieves this goal and ensures scaling, in the sense that no processor stores an amount of data, or executes an amount of work, growing with the total problem size. In other words, the storage and computational work per processor may remain bounded if the number of processors grows, in an appropriate way, with the problem size.

We discuss in the following, separately, the aspects of parallelization related to data partition and distribution of the computational work.

3.1 Data partition

The data to be distributed across the processors include:

1. the object geometry $\mathcal{G}$,
2. the MoM unknowns x and the right-hand-side vector b,
3. the equivalent Cartesian representations of the unknown vector and the Green function, i.e, the Cartesian vectors X, Y, $\widetilde{X}$, and $\widetilde{Y}$ (sharing common storage), defined on points of the Cartesian grid $\mathcal{C}$ covering the object,
4. the elements of the compressed representation (2.6) of the stiffness matrix A:
   (a) the near-field part A^{Near} of the matrix,
   (b) the expansion coefficients V mapping MoM into Cartesian vector representations,
   (c) the tabulated Green function G, or rather its Fourier transform $\widetilde{G}$ appearing in Eq.(2.13).

In the following we assume, for definiteness, that the geometry and the Cartesian grid are partitioned into ***"z-slices"*** specified by the z-coordinate. Below we describe how the data elements, listed above, are distributed across a selected number, P, of processors.

3.1.1 Distribution of the Cartesian vectors

As we mentioned before, the matrix compression algorithm and its parallel version are designed around the operations on the Cartesian grid data.

The global Cartesian grid. Suppose the bounding box of the geometry $\mathcal{G}$ has sizes $B_x \times B_y \times B_z$, the Cartesian grid spacing (assumed to be identical in all three directions) is h, and the far-field expansion order is M. The global Cartesian grid $\mathcal{C}$ is then defined as the set of $K_x \times K_y \times K_z$ grid nodes such that it extends outside the bounding box by at least $(M+1)/2$ grid spacings in each direction; this condition is necessary in order to ensure that the expansion boxes $\mathcal{B}(\alpha)$ of all the tetrahedra α are contained in the grid $\mathcal{G}$. The physical sizes of the Cartesian grid are then $(K_x - 1)\,h \times (K_x - 1)\,h \times (K_x - 1)\,h$. As

we discuss in Sections 4 and 5, the optimal (from the point of view of the Cartesian data manipulations) orientation of the object is such that

$$K_x \le K_y \le K_z \ ; \tag{3.1}$$

we always assume this orientation in the following.

We recall that, before executing the Fourier transform of Eq.(2.12), the Cartesian-grid data X have to be extended to twice their "physical" size by padding them with zeros, in order to avoid aliasing (due to periodicity of discrete Fourier transforms) in their subsequent convolution (2.13) with the Green function. After the backward Fourier transformation to the coordinate space (Eq.(2.14)), only the physical part of the output vector Y is used. With padding, the numbers of Cartesian grid nodes are $N_i = 2\,K_i$, $i = x, y, z$.

Data layout in the parallel FFT algorithm. Before discussing Cartesian data partitioning we recall that the FFTW parallel implementation of the FFT, say the Fourier transform (2.12), involves

1. Distributing the vector X across the processors according to the z-coordinates of the grid points.
2. Executing, in parallel, the two-dimensional FFT of X in the (x, y) directions.
3. Transposing the y and z indices in the data, i.e., redistributing the data such that they partitioned according to their y indices. This operation is crucial from the point of view of the parallel algorithm implementation, since it involves global all-to-all communication between the processors.
4. Executing, in parallel, the one-dimensional FFT of X in the z direction.

After the transformation the data are arranged in a different than the initial order, but, in our application, there is no need to rearrange them (the operations on the Fourier transformed Cartesian vectors may be executed on the reshuffled data as well). In the inverse FFT the steps 1 to 4 above are carried out in the reversed order.

An important remark is in order here. We mentioned before that, in our acoustic problem, the Cartesian vector X has four components, describing the pressure and three components of its gradient. However, in our implementation we do not store all the components of that vector simultaneously, but rather reuse a single-component vector. In other words, we evaluate the far-field matrix-vector product of Eq.(2.10) as a sequentially computed sum of four terms,

$$\begin{aligned} y_\alpha^{\mathrm{Far}} = k^2 \sum_\beta \sum_{\mathbf{u},\mathbf{v}} V_{\alpha\mathbf{u}}^{(\mathrm{M})}\, G(\mathbf{u}-\mathbf{v})\, V_{\beta\mathbf{v}}^{(\mathrm{M})} \left(\frac{\rho_0}{\rho_\beta} - \frac{\kappa_\beta}{\kappa_0}\right) x_\beta \\ - \sum_{j=x,y,z} \sum_\beta \sum_{\mathbf{u},\mathbf{v}} V_{\alpha\mathbf{u}}^{(\mathrm{D})\,j}\, G(\mathbf{u}-\mathbf{v})\, V_{\beta\mathbf{v}}^{(\Lambda)\,j}\, x_\beta \ , \end{aligned} \tag{3.2}$$

where we used Eqs. (2.7) and labeled with the index j the vector components of $\mathbf{V}_{\beta\mathbf{v}}^{(\mathrm{D})}$ and $\mathbf{V}_{\beta\mathbf{v}}^{(\Lambda)}$. Such an organization of the data results in a very significant reduction of storage and, in practice, no loss in the computation time; we discuss this point further in Section 3.2.3, 4, and 5.

Partition of the Cartesian grid. According to the requirements of the parallel FFT algorithm, the Cartesian grid $\mathcal{G}$ is distributed across the P processors by partitioning the set of K_z *unpadded* grid nodes into P z-slices, i.e., P subsets of $K_z^{(p)}$ points, such that

$$K_z^{(0)} + K_z^{(1)} + \cdots + K_z^{(P-1)} = K_z \ . \tag{3.3}$$

The partition task is carried out by the FFTW routine `fftw_mpi_local_size_many_transposed`. The same routine supplies a preferred distribution of the Cartesian vector X into y-slices used in the z-direction Fourier transform (step 4 above), i.e., a partition

$$N_y^{(0)} + N_y^{(1)} + \cdots + N_y^{(P-1)} = N_y \ , \tag{3.4}$$

in this case specified for the padded grid sizes. in practice, most of the numbers $K_z^{(p)}$ and $N_y^{(p)}$ are equal, but some (those assigned to the last processors) may be zero.

In the following we denote by $\mathcal{C}^{(p)}$ the z-slices of the physical Cartesian grid $\mathcal{C}$ associated with the Cartesian data slices.

3.1.2 Distribution of the geometry data

Since the Cartesian vector data are obtained from the MoM data, and those are related to the object geometry $\mathcal{G}$, we partition the geometry in a way closely related to the Cartesian data slices. Specifically, we define the geometry slice $\mathcal{G}$ assigned to the processor p is defined as the collection of all tetrahedra whose centroids $\mathbf{c}$ satisfy the condition

$$z_p \le c_z < z_{p+1} \ , \tag{3.5}$$

where $z_0, \ldots, z_p$ are the z-coordinates of planes located half-way between the nodes of the adjacent grid slices $\mathcal{C}^{(p)}$. The resulting partition is illustrated in Fig. 1.

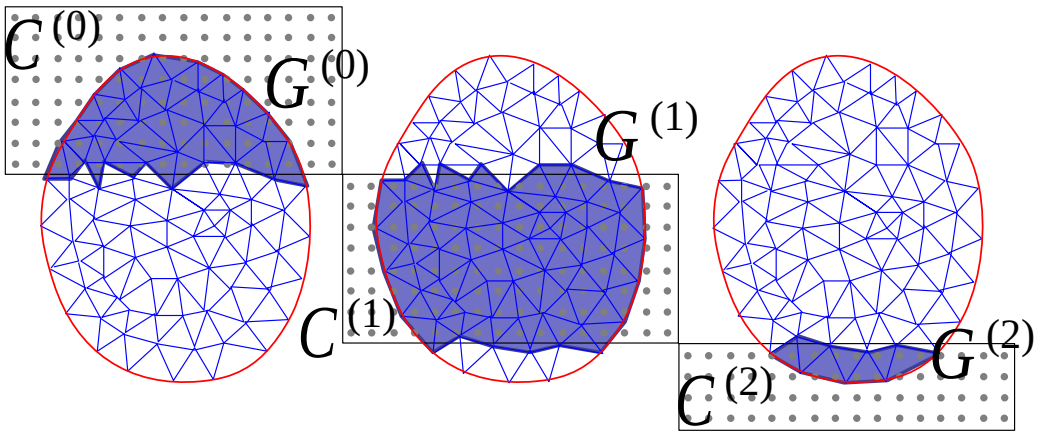


Figure 1: Partition (for $P = 3$ processors) of the Cartesian grid $\mathcal{C}$ and the object geometry $\mathcal{G}$ into disjoint z-slices $\mathcal{C}^{(p)}$ and $\mathcal{G}^{(p)}$. In order to match the usual vector and matrix notation, we directed the z axis *downwards*.

3.1.3 Distribution of the MoM unknowns

Geometry partition determines directly the distribution of the vector of the MoM unknowns x. It is imply split into P blocks,

$$x = \begin{bmatrix} x^{(0)} \\ x^{(1)} \\ \vdots \\ x^{(P-1)} \end{bmatrix} \tag{3.6}$$

where the block $x^{(p)}$ corresponds to the tetrahedra in the geometry slice $\mathcal{G}^{(p)}$.

3.1.4 Distribution of the near-field matrix

The near-field matrix A^{Near} (the item 4(a) on the list in Section 3.1) must include couplings between adjacent geometry slices, up to the near-field range R_{N}. Thus, generally, A^{Near} has the block structure

$$A^{\text{Near}} = \begin{bmatrix} A^{(0,0)} & A^{(0,1)} & \dots & A^{(0,\sigma)} & 0 & \cdots & 0 \\ A^{(1,0)} & A^{(1,1)} & \dots & A^{(1,\sigma)} & A^{(1,\sigma+1)} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ A^{(\sigma,0)} & A^{(\sigma,1)} & \dots & A^{(\sigma,\sigma)} & A^{(\sigma,\sigma+1)} & \cdots & 0 \\ 0 & A^{(\sigma+1,0)} & \dots & A^{(\sigma+1,\sigma)} & A^{(\sigma+1,\sigma+1)} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & 0 & \cdots & A^{(P-1,P-1)} \end{bmatrix} . \tag{3.7}$$

In order to simplify the notation, we assumed here that each geometry slice couples to σ of its nearest neighbors, i.e., each row and column in the matrix consists of up to $2\sigma + 1$ blocks. Since slices may have, in general, different thickness, the number of nonzero blocks in tows and columns may vary. In the following σ will denote the maximum number of coupled slices. We note that in our problem the matrix A^{Near} is not symmetric.

In distributing the matrix blocks we simply assign all the blocks of the matrix in its p-th row to the processor number p (assignment of blocks by columns would merely amount to changing the order of some operations in matrix-vector multiplication). In other words, the p-th processor stores the set of blocks

$$A^{[p,\sigma]} = \begin{bmatrix} A^{(p,p-\sigma)} & \dots & A^{(p,p-1)} & A^{(p,p)} & A^{(p,p+1)} & \dots & A^{(p,p+\sigma)} \end{bmatrix} . \tag{3.8}$$

3.1.5 Distribution of the far-field matrix data

The MoM-to-Cartesian mapping coefficients V. Partition of the expansion coefficients V (the item 4(b) listed in Section 3.1) deserves special care, since their storage requires a significant amount of memory (we give estimates in Section 5).

As in the case of the near-field matrix, the coefficients $V_{\alpha\mathbf{u}}$ may couple elements assigned to different slices; i.e., a tetrahedron α located in the geometry slice $\mathcal{G}^{(p)}$ may contribute to a Cartesian grid node $\mathbf{u}$ belonging to a grid slice $\mathcal{C}^{(p+\tau)}$, $\tau \neq 0$. Therefore, a given processor p must store some V data associated with either tetrahedra or Cartesian grid nodes assigned to other processors.

For example, a processor p could store sets of coefficients $V_{\alpha\mathbf{u}}$ for all tetrahedra α such that at least one of the nodes $\mathbf{u}$ belongs to the expansion box of the tetrahedron, $\mathbf{u} \in \mathcal{B}(\alpha)$ is located in the grid slice $\mathcal{C}^{(p)}$. This condition implies that the processor may have to store coefficients V for tetrahedra whose centroids are up to the distance Mh outside the home slice of the processor. Such a solution might be acceptable for "thick" slices (of thickness large compared to h). However, for thin slices (and a grid slice may contain just a single plane of the grid), the additional (and replicated) storage could easily exceed the storage for the slice itself.

In order to eliminate replication in the storage, and, at the same time, minimize the amount of communication, we introduced a "sparse" data structure for the representation of the mapping coefficients V: It includes, for each tetrahedron α, a list of nodes $\mathbf{u}$ (numbered within the expansion box $\mathcal{B}(\alpha)$) for which the coefficients are stored. Thus, each processors stores the coefficients for all tetrahedra α whose expansion boxes overlap the the grid slice $\mathcal{C}^{(p)}$; however, it stores coefficients only for the nodes $\mathbf{u} \in \mathcal{C}^{(p)}$. In this way, expansion coefficients for each α and $\mathbf{u}$ are stored only once, and the additional lists amount to only a very minor overhead (more detailed estimates are given in Section 5).

The Green function. The last item, 4(c), on the list in Section 3.1 is the Green function tabulated at the nodes of the Cartesian grid. Its multi-processor distribution is obtained in a straightforward way: (i) each processor p computes and stores data G for its own grid slice $\mathcal{C}^{(p)}$, and (ii) a global FFT transform is applies to these distributed data. The result is the Fourier-transformed $\widetilde{g}$ distributed according to the y-slices of the grid, which automatically matches the layout of the Fourier transform $\widetilde{X}$, as described in Section 3.1.1.

3.2 Distribution of the computational work

Parallelization of the operations in the code is constructed around the multi-processor partition of the data. An important feature of the implementation is that each processor handles only the minimal part of the object geometry, necessary in order to construct the data assigned to the processor. In the matrix construction stage we also make use, rather freely, of disk storage of geometry data and the matrix blocks, in order to avoid storing too much data at the same time. In the iterative solution stage, however, we do not utilize any out-of-core storage.

In addition to few parameters specifying matrix compression, the incident wave, etc., the input data consists of a single geometry file containing vertex coordinates of the tetrahedral mesh and definitions of tetrahedra as quadruplets of vertex numbers, and a file listing material parameters for all the tetrahedra. With this input, the computational procedure involves steps described in the following.

3.2.1 Geometry processing

The main operations performed here are as follows:

1. One processor extracts from the input geometry file the minimal amount of the global geometry information, required in order to determine geometry and other data partition.

2. Each processor p extracts from the input file its own geometry slice $\mathcal{G}^{(p)}$ and stores it as a separate file. It then extracts the corresponding material parameter data and also stores it as a file.

3. Each processor p reads geometry and material data for up to $2\sigma + 1$ geometry slices, $\mathcal{G}^{(p-\sigma)}$ to $\mathcal{G}^{(p+\sigma)}$, where σ is the number of nearest coupled slices (cf. Eq.(3.7)). In the following we refer to the slice $\mathcal{G}^{(p)}$ originally assigned to the processor p as its "home slice".

   The processor then combines the set of slices into a "welded stack" $\mathcal{W}^{[p,\sigma]}$. We use the expression "welded" to indicate that common elements of adjacent slices (vertices and facets) have been eliminated, and the resulting geometry can be treated as a conventional bona fide complete geometry. The processor also concatenates the corresponding material data into a single block.

The above operations provide, for each processor, the input data needed in the matrix computation.

3.2.2 Matrix construction

The near-field matrix. Each processor computes then the near-field part of the matrix (Eq.(2.5)) by using a pair of geometries,[3] $\mathcal{G}^{(p)}$ and $\mathcal{W}^{[p,\sigma]}$. An important advantage of this procedure is that any matrix-fill algorithm developed for a *serial* code may be applied here; in particular, there is no need to introduce any additional connectivity data relating adjacent or identical geometry elements in different slices. The resulting rectangular matrix block constitutes the set of blocks in the p-th row of the matrix of Eq.(3.7), denoted by $A^{[p,\sigma]}$ in Eq.(3.8). However, the columns of this block matrix are still indexed by tetrahedra in the welded stack geometry $\mathcal{W}^{[p,\sigma]}$.

Far-field subtractions. In the next step, each processor p modifies its matrix block $A^{[p,\sigma]}$ by subtracting far-field contributions evaluated according to Eq.(2.6). More precisely, the subtraction is implemented as

$$A^{[p,\sigma]}_{\alpha\beta} \rightsquigarrow A^{[p,\sigma]}_{\alpha\beta} - \sum_{\mathbf{u}\in\mathcal{B}(\alpha)} \sum_{\mathbf{v}\in\mathcal{B}(\beta)} V^{(1)}_{\alpha\mathbf{u}}\, G(\mathbf{u}-\mathbf{v})\, V^{(2)}_{\beta\mathbf{v}} \ . \tag{3.9}$$

In more detail:

[3] Here and in similar operations the geometries are always complemented with their corresponding material data.

1. We tabulate the set of all values of $G(\mathbf{u})$ needed in Eq.(3.9). This procedure saves a considerable amount of time by eliminating repeated computation of exponents; the required storage for G is minor, since padding is not needed and one can take advantage of the symmetries of the Green function.

2. The full set of the coefficients $V^{(1)}_{\alpha\mathbf{u}}$ is computed and stored, for all tetrahedra α in the geometry slice $\mathcal{G}^{(p)}$ and for all nodes in their expansion boxes $\mathcal{B}(\alpha)$.

3. A loop over nonempty columns β of the matrix block $A^{[p,\sigma]}_{\alpha\beta}$ is executed. For each tetrahedron β:

   (a) The set of coefficients $V^{(2)}_{\beta\mathbf{v}}$ is evaluated and stored (in a small work array) for all nodes $\mathbf{v}$ in the expansion box $\mathcal{B}(\beta)$.

   (b) A loop is executed over all nonzero elements α in the column β of the matrix block $A^{[p,\sigma]}_{\alpha\beta}$. For each element, associated with a tetrahedron α in the geometry slice $\mathcal{G}^{(p)}$, the far-field value of the matrix element is evaluated according to Eq.(3.9), in terms of the previously computed and stored coefficients $V^{(1)}_{\alpha\mathbf{u}}$ and the computed on-the-fly and temporarily stored coefficients $V^{(2)}_{\beta\mathbf{v}}$; the Green function values $G(\mathbf{u}-\mathbf{v})$ are extracted from the table precomputed in step 1 above.

After the far-field subtraction is completed, the set of mapping coefficients $V^{(1)}$ and the tabulated G are deleted.

The advantage of the above procedure is that it requires only storing the mapping coefficients $V^{(1)}_{\alpha\mathbf{u}}$ for tetrahedra in the home slice, and thus avoids any replication of the data on different processors; since the coefficients $V^{(2)}_{\beta\mathbf{v}}$ are computed on-the-fly, they incur no storage whatsoever. We note here that, for thin slices (which may often be the case), the near-field range may be significantly larger than the slice thickness; hence, storing the coefficients $V^{(2)}_{\beta\mathbf{v}}$ for βs in the entire near-field range would require an excessive amount of memory.

Far-field data in the compressed matrix. Now the processor p continues to utilize the availability of its welded stack geometry $\mathcal{W}^{[p,\sigma]}$ to (re)compute[4] the final sets of the mapping coefficients $V^{(1)}_{\alpha\mathbf{u}}$ and $V^{(2)}_{\beta\mathbf{u}}$. This time the coefficients are stored in the sparse representation, as defined in Section 3.1.5; i.e., they are computed only for nodes $\mathbf{u}$ in the home grid slice $\mathcal{C}^{(p)}$, and for all necessary tetrahedra α, either in the home geometry slice or in one of the neighboring slices. After the procedure is completed, the welded stack geometry $\mathcal{W}^{[p,\sigma]}$ is deleted from the processor's memory (together with the corresponding material data).

Near-field data in the compressed matrix. Finally, in order to generate the near-field matrix in the form of Eq.(3.7), each processor p splits its rectangular (and far-field subtracted) matrix block $A^{[p,\sigma]}$ into separate blocks, $A^{(p,p-\sigma)}$ to $A^{(p,p+\sigma)}$, with columns

[4]Computation of the mapping coefficients incurs a minor cost, hence it is not necessary to store them or transmit them between the processors.

indexed by unknowns in the geometry slices $\mathcal{G}^{(p-\sigma)}$ to $\mathcal{G}^{(p+\sigma)}$. It then deletes the original block $A^{[p,\sigma]}$.

Communication data in the compressed matrix. The compressed matrix components constructed as described above have to be complemented with information on how to exchange data between the processors in matrix-vector multiplications executed in the iterative solution stage. In computing both near- and far-field contributions only elements of the MoM vectors, such as x of Eq.(3.6), are being exchanged. Lists of such elements ("communication tables") are constructed in the preceding steps of the algorithm. For example, each processor p constructs its communication tables for near-field couplings based on the structure of the rectangular near-field matrix block $A^{[p,\sigma]}$: elements of the vector blocks $x^{(p+\tau)}$, $\tau \neq 0$, that have to be received from other processors are numbers β of those nonzero columns of $A^{[p,\sigma]}$ which reside on other processors. Similarly, far-field communication tables are built on the basis of the structure of the sets of the expansion coefficients V constructed by the processor.

3.2.3 Matrix-vector multiplication

Near-field contributions. With the data organization described in Section 3.1.4, each processor p stores a row (3.8) of near-field matrix blocks; it has to be realized that the off-diagonal blocks $A^{(p,p+\tau)}$, $\tau \neq 0$, may only contain small numbers of rows and columns. We evaluate then the p-th block of the near-field matrix-vector product as

$$y^{\mathrm{Near}(p)} := A^{[p,\sigma]}\, x = \sum_{\tau=-\sigma}^{\sigma} A^{(p,p+\tau)}\, x^{(p+\tau)} \tag{3.10}$$

(we also recall here that σ has been defined as the *maximum* number of neighboring coupled slices; the actual number of nonzero blocks in the set (3.8) may be smaller than $2\sigma + 1$).

In order to compute the matrix block (3.10), the p-th processor has to receive subsets of the vectors $x^{(p+\tau)}$ from the other processors, $(p+\tau)$. It also has to send to those processors sub-vectors of its own input vector block $x^{(p)}$. These operations are governed by the near-field communication tables constituting a part of the compressed matrix. As indicated in Eq.(3.10), they are executed in a loop through the index τ, and in each step the MPI function `MPI_Sendrecv` is called to exchange subsets of the vector $x^{(p+\tau)}$; those elements that are being sent by the processor p have been previously copied to and stored in a work buffer.

Far-field contributions. This part of the algorithm involves the operations described by Eqs. (2.11) to (2.15).

(i) In the first step, each processor p converts its MoM vector block $x^{(p)}$ to the block of the Cartesian representation vector X associated with its home slice $\mathcal{C}^{(p)}$ of the Cartesian grid. It is implemented as

$$X_{\mathbf{v}}^{(p)} = \sum_{\tau=-\sigma}^{\sigma} \sum_{\beta \in \mathcal{G}^{(p+\tau)}} V_{\beta\mathbf{v}}^{(p+\tau,p)}\, x_{\beta}^{(p+\tau)} \qquad \text{for all } \mathbf{v} \in \mathcal{C}^{(p)}\ . \tag{3.11}$$

The β sum runs over the basis functions (unknowns) in the geometry slices $\mathcal{G}^{(p+\tau)}$ stored on the processor p itself and on the neighboring processors $(p+\tau)$. The expansion coefficients $V^{(p+\tau,p)}$ are stored locally on the processor p, but in a sparse representation (Section 3.1.5), i.e., only for the nodes $\mathbf{v}$ located in the home Cartesian grid slice $\mathcal{C}^{(p)}$. As in the near-field matrix-vector multiplication, some elements of the vector blocks have to be exchanged between the processors, also in a procedure controlled by communication lists and utilizing the routine `MPI_sendrecv`.

(ii) The next steps involve operations (2.12), (2.13), and (2.14), on the Cartesian vector X.

The forward and backward FFTs involve global data transposition (mentioned in the FFT algorithm description in Section 3.1.1), i.e., global communication between all the processors. Therefore, its implementation is crucial for the parallel performance of the solution algorithm.

In our solver the forward FFT (Eq.(2.12)) is carried out with the help of FFTW routines, but with a communication scheme reducing the amount of the transferred data (relative to a literal application of the parallel FFTW implementation of the global FFT). Our algorithm (described in Appendix A) carries out communication by directly executing the MPI vector all-to-all communication routine `MPI_Alltoallv`.

Next (Eq.(2.13)), each processor p multiplies its segment of the Fourier-transformed Cartesian data $\widetilde{X}$ by the FFT of the Green function:

$$\begin{aligned} \widetilde{X}^{(p)}_{n_x\, n_y\, n_z} &= \widetilde{G}^{(p)}_{n_x\, n_y\, n_z}\, \widetilde{X}^{(p)}_{n_x\, n_y\, n_z} \\ &\qquad \text{for } 0 \le n_x < N_x \ ,\ 0 \le n_y < N_y^{(p)} \ ,\ 0 \le n_z < N_z \ . \end{aligned} \tag{3.12}$$

We exhibited here explicitly operations on the indices to emphasize the fact that the vector $\widetilde{X}$ includes padding and that its is stored in the transposed representation, and thus distributed across the processors according to the y-slices (Eq.(3.4)).

Finally, the backward FFT (Eq.(2.14)) is executed in close analogy to the forward transform, with the reversed order of its operations.

(iii) The last operation in the far-field multiplication is the conversion of the field data from its Cartesian to MoM representations (Eq.(2.15)). It is implemented in parallel as

$$y_\alpha^{\mathrm{Far}\,(p+\tau)} = \sum_{\mathbf{u}\in\mathcal{C}^{(p)}} V^{(p+\tau,p)}_{\alpha\mathbf{u}}\, Y^{(p)}_{\mathbf{u}} \qquad \text{for } \tau = -\sigma, \ldots, \sigma \text{ and for all } \alpha \in \mathcal{G}^{(p)} \ . \tag{3.13}$$

As in Eq.(3.11), the expansion coefficients $V^{(p+\tau,p)}$ are locally stored on the processor p in their sparse representation. Also, as before, the procedure involves communication between the nearby processors: The p-th processor evaluates parts of the vector blocks $y_\alpha^{\mathrm{Far}\,(p+\tau)}$, $\tau \neq 0$, and sends their elements to the processors $(p+\tau)$, where they are added to the locally computed elements. At the same time, the processor p receives additional contributions to its far-field vector block from the processors $(p+\tau)$, $\tau \neq 0$.

As we mentioned in Section 3.1.1, we the far-field matrix-vector product, Eq.(3.2), as a sequential sum of contributions of the four components of the Cartesian equivalent sources (the pressure and the three components of its gradient). Therefore, we only need storage for a single-component vector X, for the scalar Green function $\widetilde{G}$, and for an additional work

buffer used in the (not in-place) communication in the FFTs, as discussed in Appendix A [... not really discussed now, add this]; the size of the buffer is one-half of the size of X or $\widetilde{G}$.

Computing contributions to the far-field matrix-vector product sequentially amounts to transmitting, in the parallel FFT, four messages each of the quarter size of the full message. It might seem that such a procedure would be slower than transmitting a single message of the same total size. However, in our computations we rather observed a certain speed-up; this behavior appears to be due to the low latency of the InfiniBand interconnect network and, perhaps, to an overall positive effect of a significant reduction in the sizes of processes.

4 Computational complexity estimates

As we discussed in Ref. [2], the complexity of the serial algorithm algorithm implementation, apart of matrix fill, is dominated by the FFTs in the fast matrix-vector multiplication in the iterative solution stage. As follows from the description in Section 3.2.2, matrix construction requires a minimal amount of inter-processor communication and is expected to scale with the number of processors practically perfectly – a feature confirmed by results of many computations.

The near-field part of the matrix-vector multiplication in the iterative solution is, for large problems, practically negligible in comparison the far-field part of the algorithm. The latter, MoM data exchanges between nearby slices contribute little to the computational cost. The main part of the communication overhead is associated, as we discussed in Section 3.2.3, with collective (all-too-all) communication in the data transposition stage. However, we found that on the presently available parallel cluster systems utilizing fast interconnect networks, the parallel FFT (i.e., in fact, the collective MPI `MPI_Alltoallv` operation) scales quite well: although the speed-up factor is not proportional to P, it scales, typically, as $P^{2/3}$ to $P^{3/4}$. As a result, the computational cost of matrix-vector multiplication is quite low. For instance, in one of our examples (Section 6) it amounts to 6.6 s for a problem with $N \simeq 4,700,000$ unknowns, solved on 128 processors.

5 Storage estimates

In view of the effectiveness of parallelization in the fast matrix-vector multiplication, our main concern is storage per processor, which, on most present systems, is limited to $m_{\max} = 1.75\,\mathrm{GB}$. Therefore, we give a rather detailed discussion of the storage requirements arising in our approach; these results will indicate how large problems can be solved on how many processors, given the existing memory restrictions.

We first consider a rather general volumetric problem, and express the required storage in terms of a few parameters which depend on the object geometry and its discretization, and have to be estimated for specific cases. We then assume a uniform and perfectly balanced distribution of the data and obtain simple estimates in terms of the geometry size, its discretization, and compression parameters. Finally, we discuss limitations on the problem size imposed by the approach based on geometry partition into slices and, possibly, on the use of the interleaved data layout.

In the following we always give expressions for storage on a single processor, say p, out of $P \gg 1$ processors.

A general data distribution. We assume the scatterer's bounding box is oriented according to the conditions (3.1), and that the box sizes, B_i, $i = x, y, z$, are large relative to the Cartesian grid spacing,

$$h \ll B_x \le B_y \le B_z \ . \tag{5.1}$$

We also consider discretization with fairly uniformly sized and regular tetrahedra of average edge length a, hence the average volume

$$v_{\mathrm{t}}(a) = \frac{a^3}{6\sqrt{2}} \simeq \frac{a^3}{8.5} \ . \tag{5.2}$$

Further, we denote by η the ratio of the grid spacing to the tetrahedron size, i.e., assume

$$h = \eta\, a \ . \tag{5.3}$$

We notice here that the tetrahedron volume (5.2) is equal the Cartesian cell volume h^3 when

$$\eta = \eta_0 \equiv \left(6\sqrt{2}\right)^{-1/3} \simeq 0.490 \ ; \tag{5.4}$$

incidentally, this value of η appears to be nearly optimal from the point of view of minimizing the overall computational cost of matrix-vector multiplication [2].

Under the assumptions made above, the numbers of Cartesian grid points assigned to a single processor p are

$$K_{\,i} = \frac{B_i}{h} = \frac{B_i}{\eta\, a} \ , \quad i = x, y, z, \qquad K_z^{(p)} = \frac{K_{\,z}}{P} = \frac{B_z}{\eta\, a\, P} \ . \tag{5.5}$$

It follows that the storage, in bytes, for a single-component Cartesian vector, such as $\widetilde{G}$ or X, is

$$\mathcal{M}\big[\widetilde{G}\big] = \mathcal{M}\big[X\big] = \frac{B_x\, B_y\, B_z}{\eta^3\, a^3\, P} \cdot 64\,\mathrm{B} \ , \tag{5.6}$$

where the coefficient $8 \cdot 8 = 64$ is due to the factor $8 = 2^3$ for zero-padding and 8 bytes for a single-precision complex number. As we discussed in Section 3.2.3, we only need storage for the Green function $\widetilde{G}$, a single vector X, and a communication buffer B. As follows from the discussion in Appendix A, the size of the buffer is one-half of the size of $\widetilde{G}$ or X. Hence, the total storage for the Cartesian data is

$$\mathcal{M}\big[\widetilde{G}, X, B\big] = (1 + 1 + \tfrac{1}{2})\, \mathcal{M}\big[X\big] = \frac{B_x\, B_y\, B_z}{\eta^3\, a^3\, P} \cdot 160\,\mathrm{B} \ . \tag{5.7}$$

Storage for the near-field matrix A^{Near} is given by

$$\mathcal{M}\big[A^{\mathrm{Near}}\big] = N^{(p)}\, N_{\mathrm{N}}^{(p)} \cdot 12\,\mathrm{B} \ , \tag{5.8}$$

where $N^{(p)}$ is the number of unknowns assigned to the processor, $N_{\mathrm{N}}^{(p)}$ is the average number of nonzero elements per row of the matrix, and the storage of 12 B per matrix element arises from the sparse-row matrix representation (a complex value and an integer offset).

Clearly, for general geometry, the number of unknowns $N^{(p)}$ may vary in a wide interval. The number $N_{\mathrm{N}}^{(p)}$ of nonzero elements grows proportionally to the cube of the near-field range; its typical values may be of the order 100 to 200.

Finally, we give an estimate for the far-field part of the compressed stiffness matrix, i.e., for the MoM-to-Cartesian mapping coefficients V. According to Eqs. (2.7) or (3.2), for each relevant tetrahedron we need to store $(1 + 3 + 3) = 7$ components of the coefficients $V^{(\mathrm{M})}$, $\mathbf{V}^{(\mathrm{D})}$, and $\mathbf{V}^{(\Lambda)}$. As we discussed in Section 3.1.5, those coefficients have to be stored for all tetrahedra in the home slice of the processor, and for some nearby tetrahedra in the adjacent slices. On the other hand, for tetrahedra near the slice boundary, we store the coefficients not for all nodes of the expansion boxes, but only for those in the home Cartesian slice. In fact, as we find below (under the assumption of uniform discretization) these effects tend to approximately cancel, with the result

$$\mathcal{M}\big[V\big] \simeq N^{(p)} \cdot 3024\,\mathrm{B}\ , \tag{5.9}$$

where the coefficient $7 \cdot (2 + 1)^3 \cdot 16\,\mathrm{B} = 3024\,\mathrm{B}$ is obtained for $M = 2$ and for complex double-precision values of Vs.

A uniform data distribution. We consider now a volumetric rectangularly shaped object completely filling its bounding box of sizes (5.1), and uniformly discretized with tetrahedra of edge length a.

Under these simplifying assumptions, the number of tetrahedra (unknowns) per processor is given by the ratio of the slice volume to the tetrahedron volume (5.2), i.e.,

$$N^{(p)} = \frac{B_x\, B_y\, B_z}{\eta_0^3\, a^3\, P} \simeq 8.5\, \frac{B_x\, B_y\, B_z}{a^3\, P}\ . \tag{5.10}$$

Similarly, the number of nonzero elements per row of the near-field matrix is given by the ratio of the volume of a ball of radius R_{N} and the tetrahedron volume,

$$N_{\mathrm{N}}^{(p)} = \frac{4\pi\, R_{\mathrm{N}}^3}{3\, \eta_0^3\, a^3} \simeq 35.5\, \frac{R_{\mathrm{N}}^3}{a^3}\ . \tag{5.11}$$

We note that the typical compression parameters ($\eta = \eta_0$ and $R_{\mathrm{N}} = 3h$) result in $N_{\mathrm{N}}^{(p)} = 36\,\pi \simeq 113$.

Eqs. (5.10) and (5.10) yield now an estimate of the near-field matrix storage (5.8) as

$$\mathcal{M}\big[A^{\mathrm{Near}}\big] = 32\,\pi\, \frac{B_x\, B_y\, B_z\, R_{\mathrm{N}}^3}{a^6\, P} \cdot 12\,\mathrm{B} \simeq \frac{B_x\, B_y\, B_z\, R_{\mathrm{N}}^3}{a^6\, P} \cdot 1206\,\mathrm{B}\ . \tag{5.12}$$

For the considered uniform geometry discretization we can also obtain a more precise estimate of the storage for the mapping coefficients V, i.e., for the number of grid nodes, associated with the relevant tetrahedra, for which the mapping coefficients have to be stored

(as discussed in Section 3.1.5). To facilitate the reasoning, we first assume that the centroids of the tetrahedra coincide with the grid nodes (or the grid cell centroids), and then multiply the result by the factor $(\eta/\eta_0)^3$, which accounts for the difference in the distribution density of the tetrahedra and grid nodes.

We assume in the following $M \geq 1$, and first consider "thick slices", by which we mean $K_z^{(p)} \geq M+1$. In this case there exist expansion boxes entirely contained in the Cartesian home grid slice $\mathcal{C}^{(p)}$. By counting the numbers of grid nodes $\mathbf{u}$ located in the home grid slice for expansion boxes in the home slice and intersecting its boundaries, and multiplying the result by the volume factor, we obtain the

$$\#\mathbf{u} = \left(\frac{\eta}{\eta_0}\right)^3 K_x\, K_y \left[K_z^{(p)}\,(M+1) - M + 2\right](M+1)^2 \quad \text{for } K_z^{(p)} \geq M+1 \geq 2\ . \tag{5.13}$$

For "thin slices", $K_z^{(p)} \leq M$, a similar reasoning yields simply

$$\#\mathbf{u} = \left(\frac{\eta}{\eta_0}\right)^3 K_x\, K_y\, K_z^{(p)}\,(M+1)^3 \quad \text{for } K_z^{(p)} \leq M\ . \tag{5.14}$$

In the usual case $M = 2$, both of the above expression reduce, in view of Eqs. (5.5) and (5.10), to the simple expectation

$$\#\mathbf{u} = N^{(p)}\,(M+1)^3 \qquad \text{for } M = 2\ , \tag{5.15}$$

which immediately implies Eq.(5.9).

The above storage estimates for the mapping coefficients V apply to the "sparse" V representation, in which, for every tetrahedron, the coefficients are stored only for the relevant Cartesian nodes (those in the home grid slice). Were we to use the "full" representation, i.e., store, for every contributing tetrahedron α, the mapping coefficients for all the $(M+1)^3$ nodes of the expansion box $\mathcal{B}(\alpha)$, the storage would have been, instead of Eq.(5.9),

$$\mathcal{M}\big[V_{\text{full}}\big] \simeq N^{(p)} \left(1 + \frac{2M}{K_z^{(p)}}\right) \cdot 3024\,\text{B}\ . \tag{5.16}$$

The savings due to the sparse storage are thus significant, especially for thin slices: if $K_z^{(p)} \leq M$, then the ratio of the full to sparse storage size is at least 3,

$$1 + \frac{2M}{K_z^{(p)}} \geq 3\ , \tag{5.17}$$

and may reach 5 in the extreme case of $K_z^{(p)} = 1$, and for $M = 2$.

To summarize, under the simplifying assumptions made above, and with the resulting expression (5.10) for the number of unknowns, the total storage can be represented as

$$\mathcal{M}_{\text{tot}} \simeq \frac{N}{P}\, m_0\!\left(\frac{\eta}{\eta_0}, \frac{R_{\text{N}}}{h}\right)\ . \tag{5.18}$$

where N is the total number of unknowns. As follows from Eqs. (5.12), (5.9), and (5.7), the amount of storage per unknown is independent of the problem size and given by

$$m_0\!\left(\frac{\eta}{\eta_0}, \frac{R_{\text{N}}}{h}\right) = \left(\frac{\eta}{\eta_0}\right)^3 \left(\frac{R_{\text{N}}}{h}\right)^3 \cdot 73\,\text{B} + 3024\,\text{B} + \left(\frac{\eta_0}{\eta}\right)^3 \cdot 160\,\text{B}\ , \tag{5.19}$$

where the three terms are due, respectively, to the near-field, the mapping coefficients V, and the Cartesian vector storage. The reason we expressed the estimate (5.19) in terms of $R_{\rm N}/h$ rather than $R_{\rm N}/a$, is that the accuracy of the matrix compression (2.5) depends, at least in the typical range of parameters, mostly on the first ratio [2]. Therefore, storage can be minimized by varying (although in a limited range) the parameter η while keeping $R_{\rm N}/h$, and hence the accuracy, fixed. Without entering into the details, we can state that, depending on the material parameters and thus the magnitude of various terms in the integral equation kernel, the required value of $R_{\rm N}/h$ may vary from about 3 to about 6, while η is limited to $\eta \gtrsim 0.6\,\eta_0$. Correspondingly, the storage m_0 may vary in a wide range, from, say, 5 kB to 20 kB.

Finally, we comment here on the potential disadvantages of our "interleaved" Cartesian storage scheme, mentioned in the Introduction. We note that, for the range of parameters considered above, the entire Cartesian-vector storage (the last term in Eq.(5.19)) takes at most about one-quarter of the V storage, and even less of the total storage, especially for larger near-field ranges (about 10 % for $R_{\rm N}/h = 6$). Out of this fraction, 2/5 constitutes additional storage for padding (the drawback (a)) and 1/5 the storage for the communication buffer (the drawback (b)). Thus, we can conclude that the additional storage cost in the interleaved data layout is minor and should be easily compensated by the advantages of the scheme.

Limitations on the problem size. The maximum size of treatable problems is, obviously, limited by the condition $\mathcal{M}_{\rm tot} \le m_{\max}$, or, in view of Eq.(5.18), by

$$\frac{N}{P} \le \frac{m_{\max}}{m_0} \ , \tag{5.20}$$

a value in the range from about 90,000 to 370,000.

At the same time, however, a scheme based on geometry partitioning into slices does not allow us to use more processors than the total number of Cartesian grid points in the z direction, K_z. The severity of this restriction depends, of course, on the geometry of the object. In the simplest case of a cubic shape, with $B_x = B_y = B_z = B$, we can use Eqs. (5.10), (5.5), and (5.3) to relate K_z to the number of unknowns, and obtain the condition

$$P \le K_z = \frac{B}{h} = \frac{\eta_0}{\eta}\, N^{1/3} \ . \tag{5.21}$$

Eqs. (5.20) and (5.21) provide now lower and upper bounds on the number of processors,

$$\frac{m_0}{m_{\max}}\, N \le P \le \frac{\eta_0}{\eta}\, N^{1/3} \ , \tag{5.22}$$

which impose a limit on the number of unknowns,

$$N \le \left(\frac{\eta_0}{\eta}\, \frac{m_{\max}}{m_0} \right)^{3/2} \ . \tag{5.23}$$

Thus, for $m_0 = 20\,$kB and $\eta = 0.6\,\eta_0$, and, under the idealized conditions we assumed, $N \le 60,000,000$.

6 Examples of large-scale computations

6.1 The geometrical models and material properties

We present below results of example computations involving

1. a realistically-shaped model of a human head;
2. the same head model and a model of helmet, with the in-between space filled by air;
3. the model of a head and a helmet, with the in-between space filled by a low-density material (cork).

We used two tetrahedral discretizations of the models, differing by about a factor 2 in the linear sizes of the tetrahedra; the average edge lengths were about 6.3 mm and 3.3 mm. The models of the head and the helmet, with the finer discretization, are shown in Fig. 2, and the parameters of the geometries are listed in Table 1. The numbers given there indicate a fairly uniform discretization: in the worst case the minimum edge length is about 1/5-th of the average value, and the minimum angle of a facet is 4.3°. Volumetric discretization (tetrahedronization) of surface models was carried out by means of the program `tetgen`, version 1.4.1, available in the public domain.

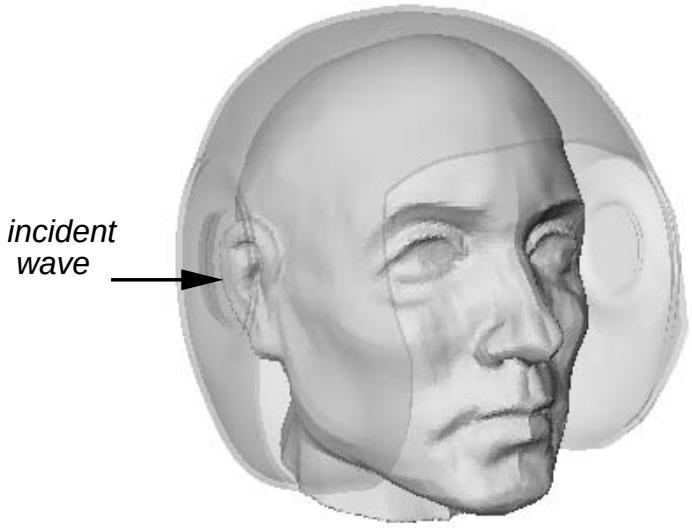


Figure 2: Model of the external head surface and the helmet. The arrow indicates the direction of the incident wave.

Table 1: Parameters of the geometries used in the computations: the number of vertices v, the number of edges e, the number of facets f, the number of tetrahedra t, average edge length $\ell_{\rm av}$, minimum edge length $\ell_{\rm min}$ (both in meters), minimum facet angle $\alpha_{\rm min}$ (in degrees)

geometry	v	e	f	t	$\ell_{\rm av}$	$\ell_{\rm min}$	$\alpha_{\rm min}$
head	65,797	471,186	803,108	397,718	0.00626	0.00233	16.8
head + helmet	82,708	556,731	923,469	449,444	0.00619	0.00221	16.8
head + cork + helmet	112,510	801,487	1,364,352	675,374	0.00627	0.00221	16.8
head	442,065	3,214,434	5,512,045	2,739,675	0.00329	0.00112	13.3
head + helmet	516,465	3,593,613	6,047,750	2,970,600	0.00331	0.00059	4.3
head + cork + helmet	754,796	5,467,095	9,366,177	4,653,875	0.00331	0.00059	4.3

We assumed the head model filled with a homogeneous material,[5] with mechanical properties of bone (which would effectively constitute the the outer layer of the head). The acoustic parameters of the materials are listed in Table 2.

The models were placed in the field of a harmonic acoustic wave of frequency 5 kHz (and the wavelength in air $\lambda = 6.8\,{\rm cm}$), incident laterally on the right ear of the head model, as indicated in Fig. 2.

The computations were carried out on a Linux cluster with the InfiniBand interconnect network, on 32, 108, and 128 processors.

Table 2: Acoustic properties of materials used in the simulations

material	ρ/ρ_0	n^2
bone	1777.0	0.1524
brain	835.4	0.0564
cork	150.0	0.65
steel	6667.0	0.0035

In the following Subsections we discuss some aspects of the computation, such as the accuracy of the far-field matrix compression, physical properties of the solutions, and scaling of the code performance.

[5] We stress, however, that the presence of inhomogeneities would not increase the computational cost of the solution.

6.2 Analysis of the accuracy of the far-field expansion in the problem of the human head helmet with cork filling

The accuracy of the compressed representation of the far-field depends on two compression parameters: the Cartesian grid sampling ratio $\chi := \lambda/h$ and the near-field range (in the solution of the surface problem) measured in units of the Cartesian grid spacing, i.e., the parameter $R_{\rm N}/h$. We present here the comparison of solutions obtained for a fixed value of the sampling ratio, $\chi := \lambda/h$ (i.e., a fixed Cartesian grid spacing h), and several values of the range parameter, $R_{\rm N}/h = 6, 8, 10, 12$. In all cases the near-field range in the volume problem solution is one-half that in the surface problem.

As we mentioned following Eq.(5.19), the far-field accuracy is controlled primarily by the value of the ratio $R_{\rm N}/h$, and depends only weakly on the Cartesian grid spacing h (in a reasonable range of values $h \lesssim a$, where a is the average tetrahedron size). Therefore, by keeping h constant and varying $R_{\rm N}/h$, we are essentially testing the sensitivity of solutions to the far-field compression accuracy.

As we discussed in Sec. 5, the computational cost of the solution (more precisely, the matrix fill) and the required storage strongly depend on the range parameter $R_{\rm N}$. In particular, the size of the near-field component of the stiffness matrix is approximately proportional to $R_{\rm N}^3$ (Eqs. (5.11), (5.12), and (5.19)).

The analysis was carried out for the largest problem we considered, i.e., the model of a human head with a helmet, with the space between the two filled with cork. As indicated in Table 1, the system was discretized with $N \simeq 4,700,000$ tetrahedra. The solutions were computed on $P = 128$ processors.

Fig. 3 shows the distribution of the far-field errors $\mathcal{E}_{\alpha\beta}$ (Eq.(2.9)) as a function of the distance $R \equiv R_{\alpha\beta} \equiv |\mathbf{c}_\alpha - \mathbf{c}_\beta|$ between the centroids of the basis functions ϕ_α and ϕ_β (Eq.(2.8)). More precisely, the range of distances R is divided into nearly 200 bins, and in each bin the average error is computed.

Fig. 3 indicates that, in order to achieve a given accuracy, the range R must be significantly (about a factor of 2) larger in the surface problem than in the volume problem. This behavior is expected on theoretical grounds, since the matrix elements in the volume problem are dominated by monopole-monopole interactions, while those in the surface problem involve only monopole-dipole couplings. Therefore, in the latter case, the dipole field has to be approximated by a set of monopole sources located in the "expansion box" of the considered basis function, and such an approximation cannot be valid in the immediate vicinity of the box.

As we stated above, we compare results of computations for near-field range parameters $R_{\rm N}/h = 6, 8, 10, 12$ (referring to the surface problem; in the volume problem we assume ranges equal one-half of these values). For those choices, the error values at the maximum distance drop from about 3 % to below 1 %, in both surface and volume problems.

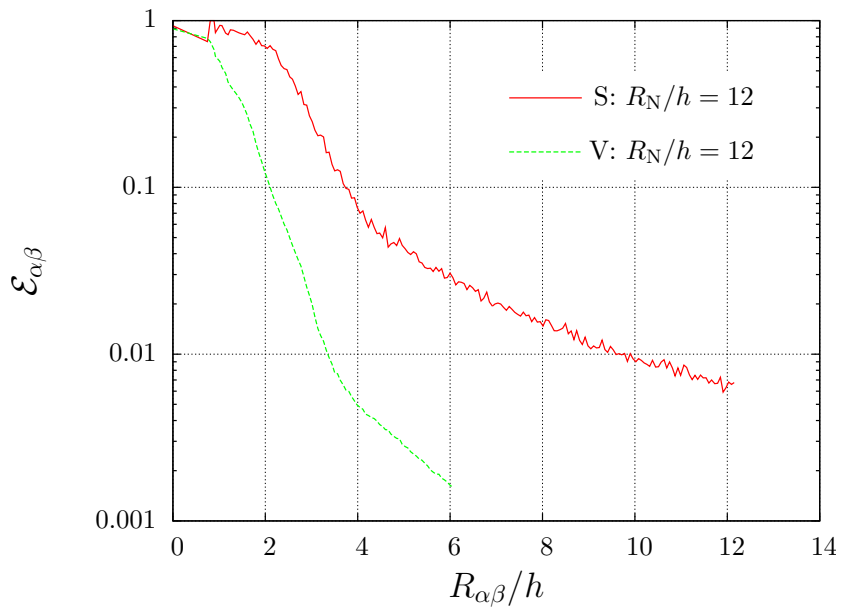


Figure 3: Averaged (over distance bins) relative far-field errors in the surface (S) and volume (V) problems for the range parameter $R_{\rm N}/h = 12$, plotted as functions of the distance between basis function supports, measured in units of the Cartesian grid spacing h. The plots represent output of the processor $p = 59$ with the assigned $N^{(p)} \simeq 60,000$ unknowns.

As the first check on the effects of the near-field range parameter we show in Fig. 4 plots of convergence of the iterative solutions. It is evident that convergence in the surface problem is practically independent of $R_{\rm N}$. In the volume problem the solution for $R_{\rm N}/h = 6$ converges differently than in the remaining cases, but for $R_{\rm N}/h \geq 8$ the curves are nearly indistinguishable.

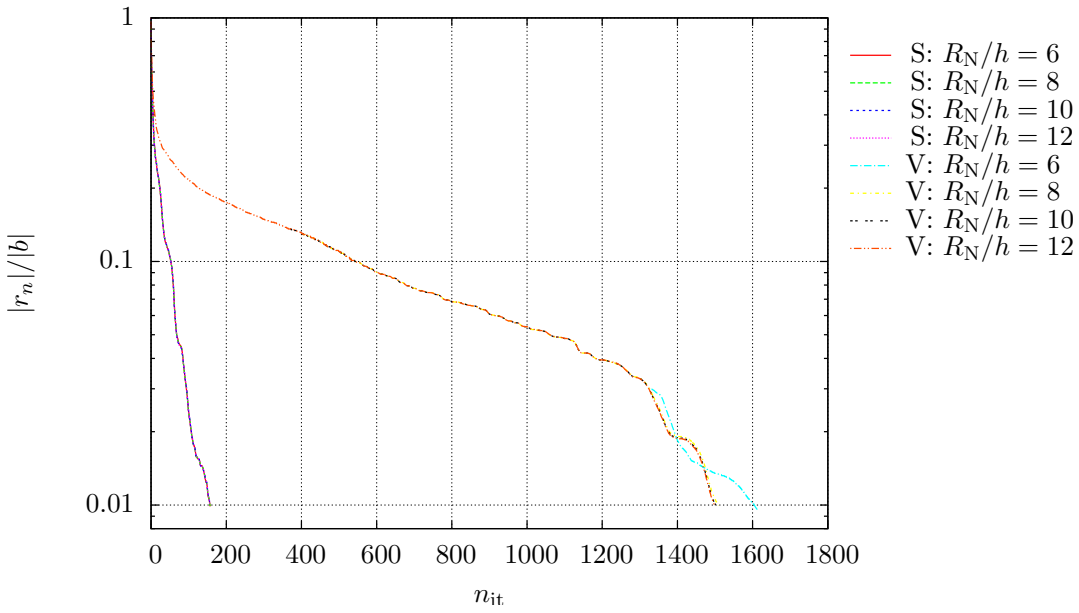
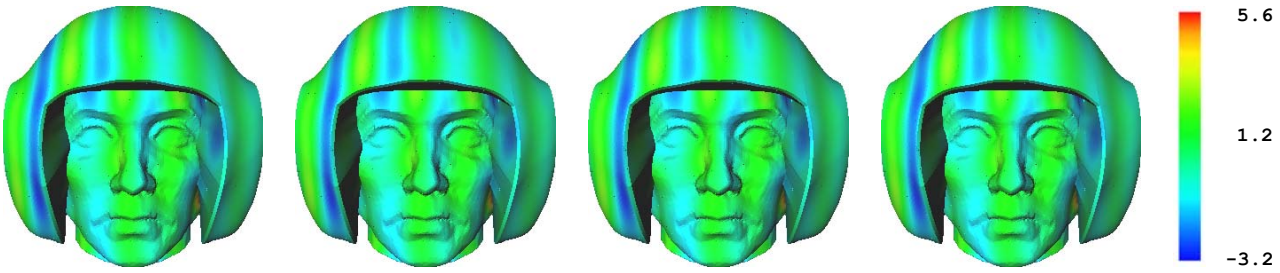


Figure 4: Convergence (the relative residual norm) of iterative solutions in the surface (S) and volume (V) problems for the indicated values of the near-field range parameter R_{N}/h.

The iterative convergence results already suggest that even the smallest near-field range and the resulting far-field error on the level of $\sim 3\,\%$ are sufficient for obtaining accurate solutions. This assertion is verified by the following plots of pressure distributions.

Fig. 5 shows pressure distributions on the exterior surface of the model. These results, obtained by solving only the surface problem, are visually practically indistinguishable.

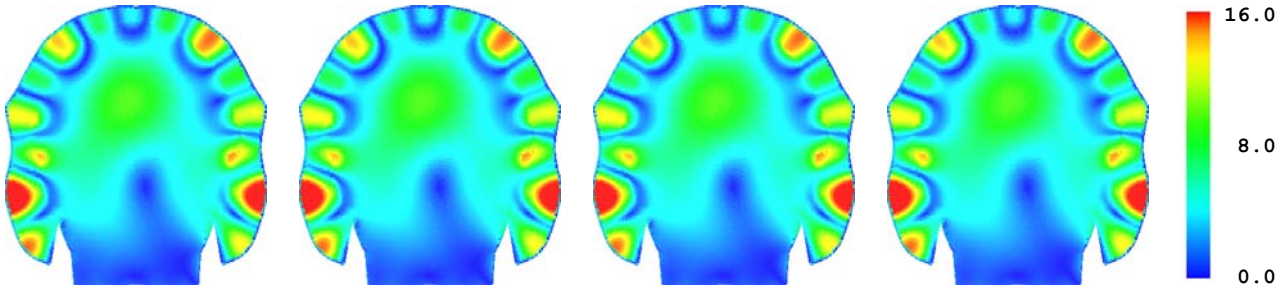


(a) (b) (c) (d)

Figure 5: Distributions of the real part of pressure on the external surface of the model of the human head, the helmet, and the cork filling, with $N \simeq 4,700,000$ unknowns. The distributions, plotted in linear scale, were obtained for the same values of the near-field range as in Fig. 4, i.e., (a) $R_{\mathrm{N}}/h = 6$, (b) $R_{\mathrm{N}}/h = 8$, (c) $R_{\mathrm{N}}/h = 10$, (d) $R_{\mathrm{N}}/h = 12$.

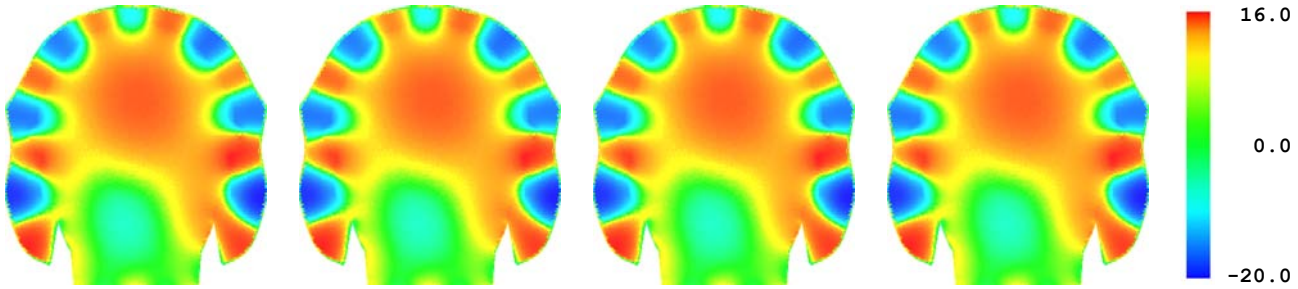
The following Figures, 6 and 7, show pressure distributions inside the model. These solutions are obtained from the volumetric integral equations in the second step of the solution procedure, which uses the surface problem solution as input.

In order to exhibit the oscillatory nature of the solution we plotted in Fig. 7 the real part of the pressure. For additional emphasis, that distribution is displayed in the logarithmic scale (modified near zero values). The pattern of oscillation on the object surface matches that shown in Fig. 5.



(a) (b) (c) (d)

Figure 6: Distributions of the absolute value of pressure in the coronal section of the model of the human head, the helmet, and the cork filling, with $N \simeq 4,700,000$ unknowns. The distributions, plotted in linear scale, were computed for the same values of the near-field range as in the previous Figures, i.e., (a) $R_{\rm N}/h = 6$, (b) $R_{\rm N}/h = 8$, (c) $R_{\rm N}/h = 10$, (d) $R_{\rm N}/h = 12$.



(a) (b) (c) (d)

Figure 7: Distributions of the real part of pressure in the coronal section of the model of the human head, the helmet, and the cork filling, with $N \simeq 4,700,000$ unknowns. The distributions, plotted in logarithmic scale, were obtained for for the same values of the near-field range before, i.e., (a) $R_{\rm N}/h = 6$, (b) $R_{\rm N}/h = 8$, (c) $R_{\rm N}/h = 10$, (d) $R_{\rm N}/h = 12$.

To summarize, Figs. 3 to 7 indicate that the near-field range $R_{\rm N}/h = 8$ (or even $R_{\rm N}/h = 6$), resulting in far-field expansion accuracy on the level of 2 % (or 3 %), yields results practically indistinguishable from those for a much large range, $R_{\rm N}/h = 12$.

6.3 Pressure distributions in a human head model: dependence on the discretization

We compare now pressure distributions in the head (without the helmet), computed for two discretizations. In the more coarsely discretized model (a) the number of tetrahedra (and unknowns) is $N \simeq 400,000$; the finer discretization (b) yields $N \simeq 2,700,000$ (Table 1).

The computations have been carried out with the Cartesian grid sampling rate $\chi = 80$ and with the near-field range $R_{\rm N}/h = 6$. As we have discussed in Subsection 6.2, these compression parameters are entirely sufficient for obtaining accurate solutions. The total solution times were (a) about 30 minutes, with 2.5 s per iteration, on $P = 32$ processors, and (b) about 50 minutes with 4.6 s per iteration, on $P = 108$ processors.

The results, shown in Fig. 8, demonstrate a good agreement between the pressure distributions obtained for the two discretizations.

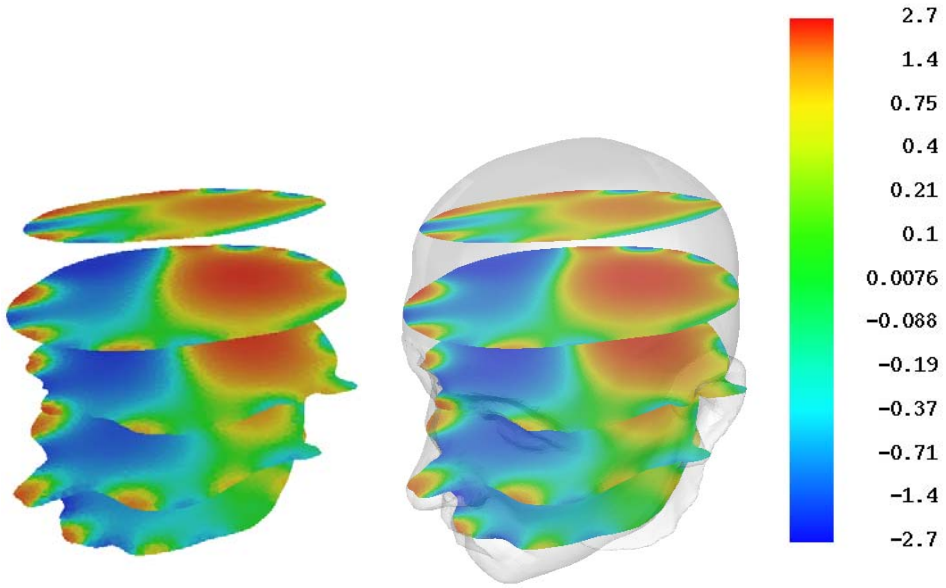


Figure 8: Distributions of the real part of the pressure, plotted in the logarithmic scale, on several sections in the axial plane of the human head model with (a) a coarser discretization ($N \simeq 400,000$ tetrahedra), and (b) a finer discretization ($N \simeq 2,700,000$ tetrahedra). The models are subject to an acoustic wave of unit pressure amplitude and frequency 5 kHz, incident horizontally on the right ear. In (b) the outline of the head outer surface is superimposed on the pressure distributions.

6.4 Pressure distributions in a human head model in the presence of an unattached helmet

Next, we show results for the human head model with the finer discretization, suspended inside a helmet. The problem involves $N \simeq 3,000,000$ (Table 1) and has been solved on

$P = 108$ processors, with the same matrix compression parameters as before ($\chi = 80$, $R_{\rm N}/h = 6$).

The results (Fig. 9) show the pressure distribution inside the head roughly similar, and of similar magnitude, as for the isolated head (Fig. 8(b)), although the details are different. One can thus infer that the presence of a helmet – not attached in any way to the head – has only a moderate effect on the pressure values inside the head model. However, a configuration with a helmet "floating" in the air can hardly be considered a physically realistic model.

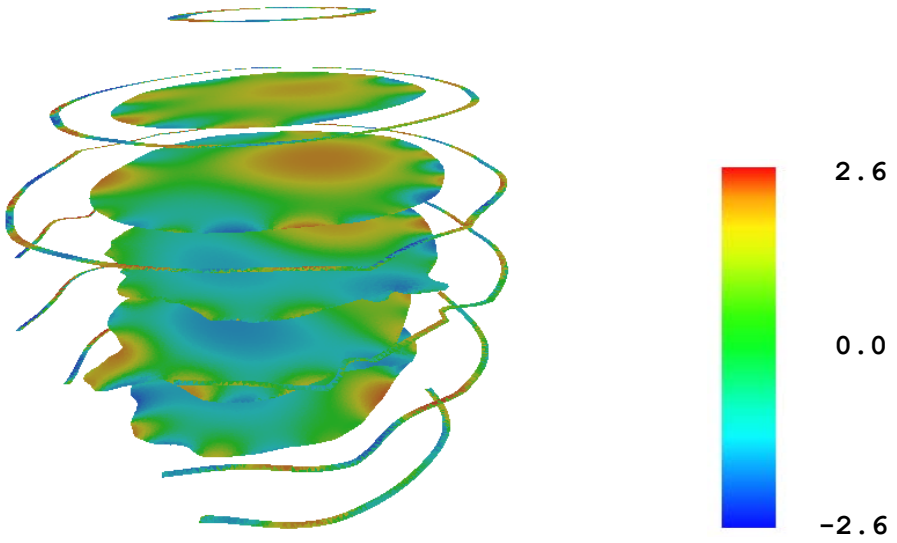


Figure 9: Distribution of the real part of the pressure, plotted in the logarithmic scale, on several sections in the axial plane of the human head model with an unattached helmet, and the total number of unknowns $N \simeq 3,000,000$. The frequency and the wave incidence direction are as in previous Figures.

6.5 Pressure distributions in a human head model in the presence and absence of a helmet with a cork lining

As the final example, we compare the pressure distribution in the head model obtained in the previous calculation (for the finer discretization) with the distribution for the same model with an addition of a steel helmet and the in-between space filled with cork. The latter geometry is discretized with $N \simeq 4,700,000$ tetrahedra, also with the tetrahedron size of about 3 mm (Table 1). The solution for the head and helmet system was carried out on 128 processors in the total time of about 2 hours, and 6.6 s per iteration. The overall time increase in that case is mostly due to the increased number of iterations.

We have already described a set of computations for the head, cork, and helmet system in Subsection 6.2. Here we comment on the physical features of the problem and its solution, and their relation to the problem of the isolated head.

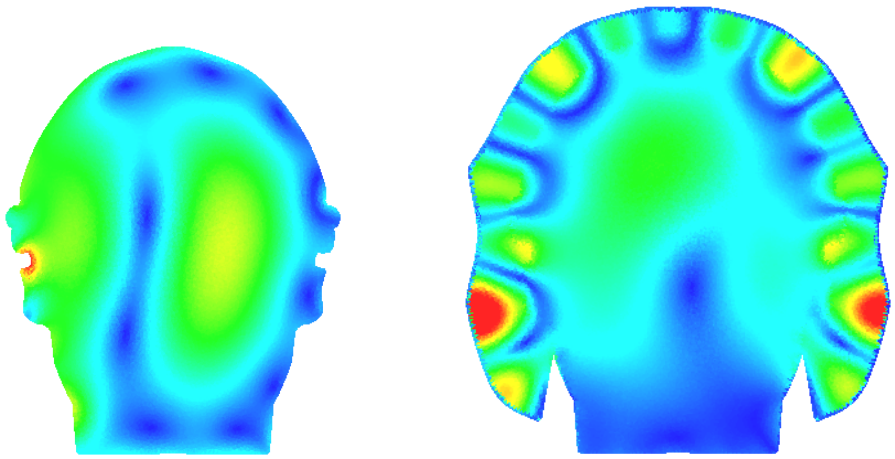
The results of the computations are visualized, as distributions of the absolute value of the pressure in the coronal plane of the models, in Fig. 10. Fig. 10(ii) is identical to the previous Fig. 6(b). The real part of the pressure has been shown before in Fig. 7(b).

The solutions show a nontrivial behavior and exhibit physical phenomena which may be relevant in the design of protective devices:

In the case of the head, Fig. 10(i), the pressure is maximal at the entrance to the ear canal, and it is smoothly distributed inside the head: while the wavelength in the air is about 6.8 cm, it is $6.8\,\mathrm{cm}/n = 6.8\,\mathrm{cm}/\sqrt{0.1524} = 17.4\,\mathrm{cm}$ in the head material (it would have been even longer in soft tissues, e.g., in the brain). In fact, the solution is suggestive of a resonance-type (P-wave) behavior: the pressure changes sign along the approximately vertical line seen in the Figure and, more distinctly, in the spatial distribution shown in Figs. 8.

The solution for the head and helmet system, Fig. 10(ii), is quite different. It exhibits a distinct oscillatory behavior along the surface of the helmet and in the region filled by cork. This region appears to have properties of a "waveguide": because of the cork density being significantly lower than that of the surrounding materials (the helmet and the head), and the resulting impedance mismatch at the boundaries, the wave tends to be trapped in that region. Since the refractive index of cork is not much different from that of air, wave oscillations are relatively rapid. We stress, however, that the physical picture suggested by Fig. 10(ii) would change if we considered a dissipative (attenuating) filling material, e.g., a strongly damping porous material characterized by a complex refractive index.

We note, finally, that, for the particular frequency considered here, the presence of the helmet with a cork lining completely changes the pressure distribution inside the head, but does not reduce its maximum value (we note that the data Figs. 10(i) and 10(ii) are rendered in different scales). This feature, however, is likely to depend on properties of the lining (filling) material.



(i) (ii)

Figure 10: Pressure distributions in the coronal plane for (i) the human head model and (ii) the system consisting of the human head and a steel helmet models, with the in-between space filled by cork. The numbers of unknowns in these two problems are $N \simeq 2,700,00$ and $N \simeq 4,700,00$. The maximum pressure values are about 4 in (i) and 15 in (ii).

6.6 Scaling in parallel computations

To assess scaling of our solver (with the problem size and the number of processors) we summarize the results of a set of computations for the models described at the beginning of this Section. We list in Table 3 the time of matrix-vector multiplication[6] as a function of (a) the number of unknowns N; (b) the Cartesian grid sizes K_x, K_y, and K_z (always without padding); (c) the number of processors P; (d) the number of Cartesian grid nodes per processor, $K_{\mathcal{C}}/P := K_x K_y K_z/P$; and (e) the near-field range in units of the Cartesian grid spacing, $R_{\rm N}/h$ (which affects the size of the near-field part of the matrix).

[6]More precisely, we list the time per iteration, which includes additional computations of vector dot-products and of linear combinations of vectors.

Table 3: Matrix compression parameters used in the computations

N	$K_x \times K_y \times K_z$	P	$K_{\mathcal{C}}/P$	R_{N}/h	t (s)
397,718	$160 \times 192 \times 180$	32	172,800	3.2	2.1
397,718	$180 \times 216 \times 180$	32	218,700	6.0	2.5
449,444	$180 \times 200 \times 192$	32	216,000	3.2	2.6
449,444	$192 \times 216 \times 192$	32	248,832	6.0	3.5
675,374	$180 \times 200 \times 192$	32	216,000	4.8	2.4
2,739,675	$300 \times 384 \times 324$	108	345,600	6.0	4.6
2,970,600	$375 \times 405 \times 384$	128	455,625	6.0	7.0
4,653,875	$375 \times 405 \times 384$	128	455,625	6.0	6.9
4,653,875	$375 \times 405 \times 384$	128	455,625	12.0	7.1

Even though the number of unknowns and the number of processors vary by the factors of 12 and 4, the Table shows that the matrix-vector multiplication time depends mainly on a single parameter: the number of Cartesian grid nodes per processor. This behavior is expected when matrix-vector multiplication complexity is dominated by the Fourier transform, which also approximately scales with the number of processors. These assumptions imply

$$t(K_{\mathcal{C}}, P) \simeq \gamma \, \frac{1}{P} \, K_{\mathcal{C}} \, \log_2(K_{\mathcal{C}}) \equiv \gamma \, \frac{K_{\mathcal{C}}}{P} \left(\log_2 \frac{K_{\mathcal{C}}}{P} + \log_2 P \right) \tag{6.1}$$

with some fixed coefficient γ (depending also on the hardware, particularly on the communication network).

Fig. 11, in which we plot the matrix-vector multiplication times for the problems of Table 3, shows that the actual performance of the solver is close to the behavior predicted by Eq.(6.1).

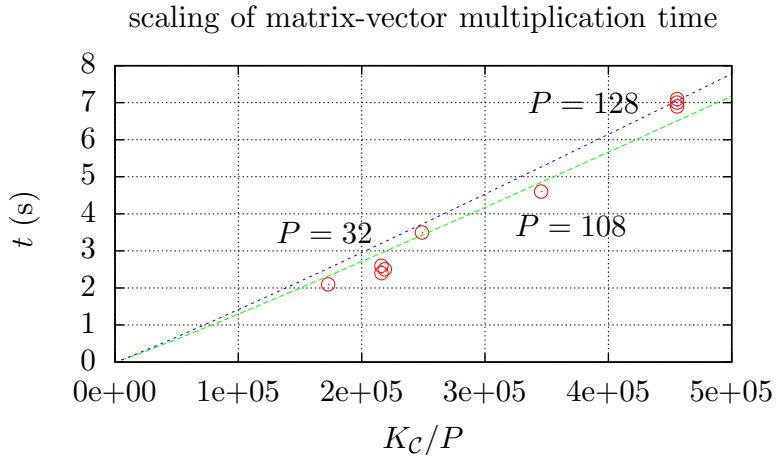


Figure 11: Matrix-vector multiplication time as a function of the number of Cartesian grid nodes per processor, $K_{\mathcal{C}}/P$, for the computations listed in Table 3. The curves are defined by Eq.(6.1) with $\gamma = 0.6 \cdot 10^{-6}$ s, and plotted for $P = 32$ (the lower curve) and $P = 128$ (the upper curve). Numbers of processors, P, are shown next to the corresponding groups of points.

Appendices

A Handling of zero padding and communication in parallel implementation of FFTs

We discuss here some aspects some aspects, essential for the code performance, of parallel implementation of the FFTs.

In Fig. 12 we illustrate the operations carried out on the Cartesian vector X in the case of $P = 3$ processors. In the problem shown in the Figure the Cartesian vectors are specified in terms of their sizes K_i (without padding) and $N_i = 2\,K_i$ (with padding); in the Figure we do not display the x-dimensions, and leave them unspecified.

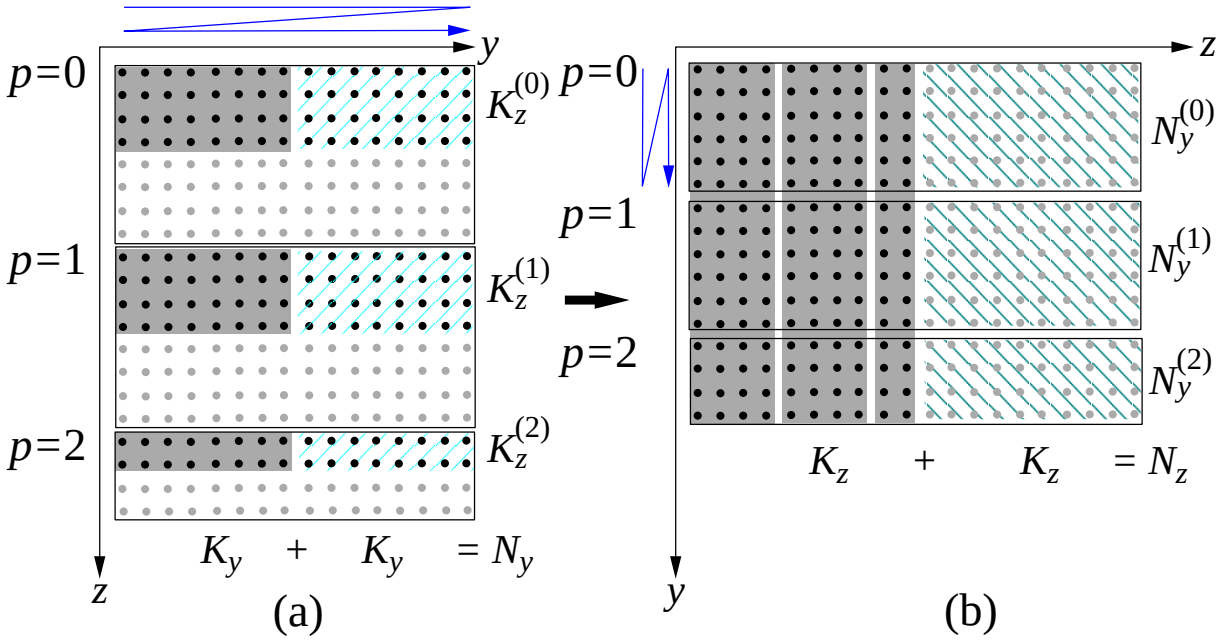


Figure 12: FFT in the (x, y) directions of the distributed data (a), followed by transposition and FFT in the z direction (b). The FFTs extend the range of data from the original storage without padding (grey background) to the storage with padding (grey + cross-hatched areas). Data owned by the individual processors are shown in boxes labeled with the processors numbers p. The zig-zag lines indicate the data storage order. Further details are described in the text.

Our parallel FFT algorithm implementation, while it uses the FFTW routines, is significantly more efficient than the original, totally "encapsulated" three-dimensional FFT provided in the FFTW package. In fact, the "encapsulated" FFT, with the transformation plan generated by the routine `fftw_plan fftw_mpi_plan_dft_3d` is practically not

usable in our solver. The principal reason is a conflict between the way this routine handles padding, and an economical distribution of data in the entire matrix-vector multiplication scheme:

FFT padding in the z direction has to occupy a contiguous range of n_z indices; hence, if z-slices of the Cartesian grid are distributed across the processors, about one-half of them will own exclusively the padding space. This configuration is more-or-less adequate if only FFTs are being computed, although even in this case handling of zero padding is not efficient. However, the other operations in the matrix-vector multiplication (near-field computation and conversion between the MoM and Cartesian representations) require access to the MoM vector x and to the multipole expansion coefficients V, and question arises how to to distribute these data. There are at least two possibilities:

1. Distribute slices of the x and V data only across the processors storing the "physical" segment of the Cartesian grid. In this case near-field and MoM-Cartesian conversion operations can be performed locally, but only one-half of the processors participates.

2. Distribute slices of the x and V data across all processors. In this arrangement all processors are involved in the near-field computation and MoM-Cartesian conversion, but the resulting Cartesian vector data have to be redistributed before and after performing the FFTs.

Evidently, none of them is computationally efficient.

Another potential source of inefficiency of the encapsulated three-dimensional FFTW transform is the redundant (in our case) "back-transposition", which the FFTW routine performs in order to restore the original layout of the data. This operation can be eliminated by creating the forward and backward FFT plans with the flags `FFTW_MPI_TRANSPOSED_OUT` and
`FFTW_MPI_TRANSPOSED_IN`, which suppress the "back-transposition", and cause the Fourier-transformed data to be left in the transposed storage form. Nevertheless, the previously listed difficulties remain.

The procedure we implemented in our code avoids the difficulties entirely: both the FFT and the $\{x, V\}$ data are distributed across all processors, all processors participate approximately equally in the near- and far-field operations, as well as in the FFT computations, and, besides, by the appropriate data arrangement, we reduce communication costs by not transferring the z-padding regions.

In our algorithm we use, instead of the FFTW transpose, the generic MPI all-to-all communication. A potential drawback of this implementation is that the MPI routine is not executed in-place, as the FFTW transpose, hence an additional storage is required. However, the storage required for the additional buffers is minor (see Eq.(5.7)).

If we use the generic MPI collective communication, an additional complication (but leading to an insignificant cost increase) is the necessity of rearranging the FFT data before transposing them. The entire procedure is illustrated in Fig. 13.

As in Fig. 12, we start with the FFT data distributed across the processors in "z-slices" (Fig. 13(a)). The data blocks on the individual processors have sizes $(N_x \times N_y \times N_z^{(p)})$, i.e., they are allocated with the padding space.

Originally, the input data X are stored only in the "physical" parts ($K_x \times K_y \times K_z^{(p)}$) of the blocks, and the remaining padding space is filled with zeros. After the first FFT, in the (x, y) directions, is performed (by each processor independently), the data expand to ($N_x \times N_y \times K_z^{(p)}$), i.e., they fill the padding in the x and y directions, but the z-padding remains unused. This is the situation shown in Fig. 13(a), where the grey areas indicate the actual data, and the white areas the zero padding. The ordering of the data elements X_{n_x,n_y,n_z} (the n_x changing the fastest) is shown by the zig-zag lines in the Figure.

Fig. 13(b) shows a rearrangement of the data of Fig. 13(a), executed prior to transposition: Now, each block of the actual data, of size ($N_x \times N_y \times K_z^{(p)}$), is split into P sub-blocks of sizes ($N_x \times N_y^{(q)} \times K_z^{(p)}$), $q = 0, 1, , \ldots , P-1$. The blocks (marked "00" to "22") are stored in memory of the processor p one after the other, with the indicated element ordering within each block (again, with the fastest changing index n_x). Clearly, the rearrangement is carried out locally, and its cost is proportional to $N_x\, N_y\, K_z^{(p)}$ (no z-padding storage is involved).

Finally, the data layout after the transposition is visualized in Fig. 13(c). The blocks of Fig. 13(b) have been simply moved from processor to processor, with changing only their locations (offsets), but not their internal structure or ordering. Due to the offset changes, the data blocks on each processors are now "concatenated" in the index n_z, leaving contiguous zero padding in z direction. We stress that only the filled (here, shaded in grey) blocks, and not the empty padding space, are transferred between the processors.

The above description applies to the forward Fourier transform of Eq.(2.12). In the backward FFT (Eq.(2.14)) the transposition of data is the reverse of that in Fig. 13. In this case also only the shaded blocks of Fig. 13(c) are transmitted between the processors to obtain the configuration of Fig. 13(c). The data values in the padding areas are irrelevant, and are discarded in the process.

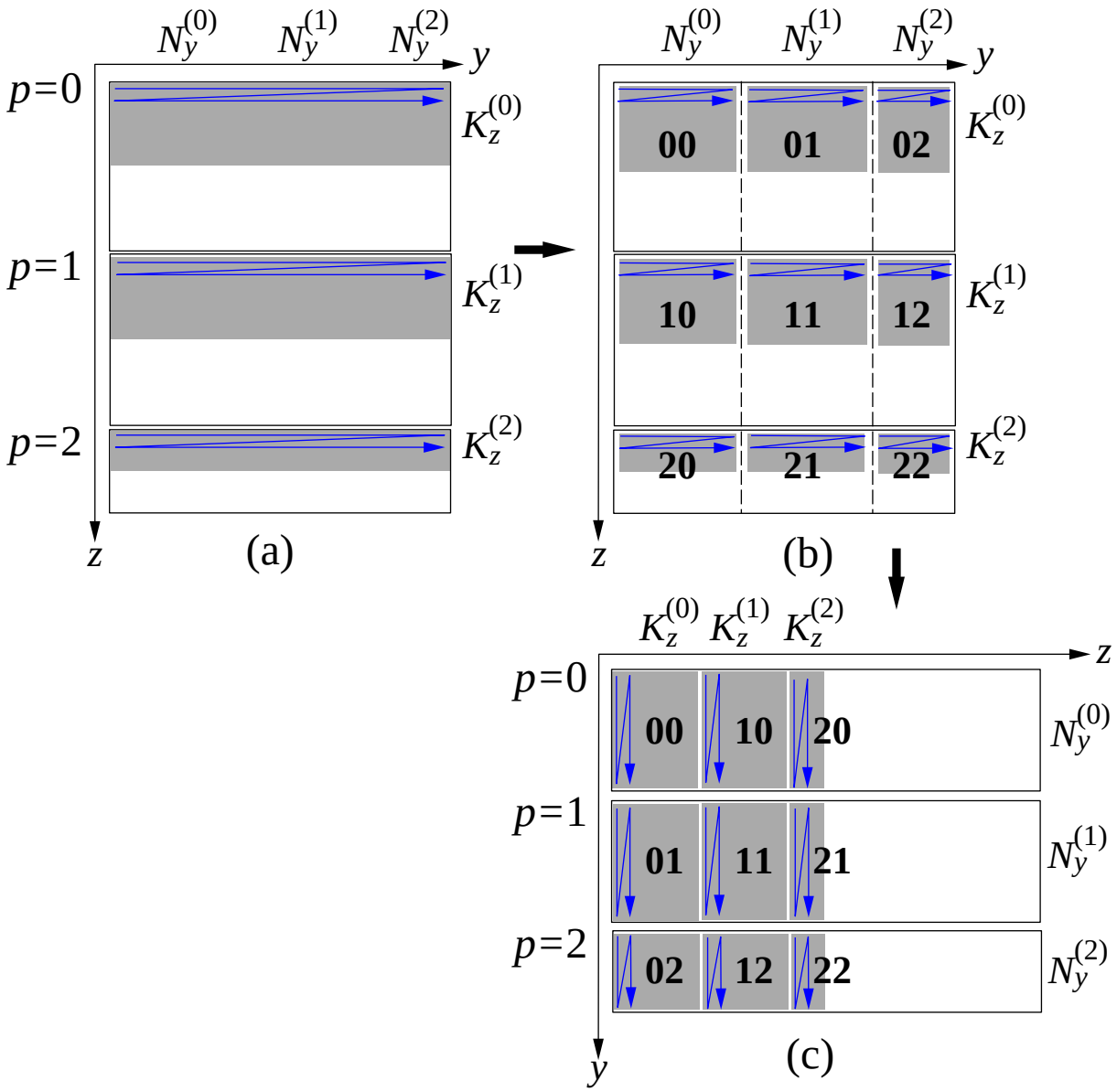


Figure 13: Original FFT data (a), their rearranged storage (b), and the result of their transposition by means of the MPI all-to-all communication routine (c). As before, the zig-zag lines indicate ordering of elements.

References

[1] E. Bleszynski, M. Bleszynski, and T. Jaroszewicz, "AIM: Adaptive Integral Method for solving large-scale electromagnetic scattering and radiation problems," *Radio Science*, vol. 31, pp. 1225–1251, 1996.

[2] ——, "Fast volumetric integral solver for acoustic wave propagation through inhomogeneous media," *J. Acoust. Soc. Am.*, vol. 124, pp. 396–408, 2008.

[3] ——, "Fast volumetric integral-equation solver for high-contrast acoustics," pp. 3684–3693, 2008.

[4] S. Y. and M. H. Schultz, "GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems," *SIAM J. Sci. Stat. Comput.*, vol. 7, pp. 856–869, 1986.

[5] H. C. Elman, "Iterative methods for large sparse nonsymmetric systems of linear equations," Ph.D. dissertation, Computer Science Dept., Yale University, 1982.