Proc. IMACS 1st Int. Conf. on Comp. Phys., Boulder, Colorado, June 1990

Multilevel Computations of Integral Transforms and Particle Interactions with Oscillatory Kernels[1]

Achi Brandt

Department of Applied Mathematics & Computer Science
The Weizmann Institute of Science
Rehovot 76100, Israel

Content

[1] Research supported in part by the Air Force Office of Scientific Research, United States Air Force, under grants AFOSR-86-0126 and AFOSR-86-0127, and by the United States National Science Foundation under grant NSF DMS-8704169.

1. Introduction

1.1 Review: types of multiscale computation

Multiscale (multilevel) computations have evolved into an independent discipline by itself; interacting with other computational methodologies, it has its own internal development, gradually increasing the understanding of the many types of multi-scale interactions, their modes of operation and domains of applications.

Beyond the traditional multigrid fast solvers for linear elliptic equations, extremely efficient and highly parallelizable multilevel techniques have recently been developed for the following types of mathematical tasks:

1. Solution of general nonlinear steady-state partial differential systems in general domains, discretized on locally-adaptable grids. (The multigrid efficiency traditionally achieved for simple elliptic problems has recently been extended to general non-elliptic systems, including high-Reynolds flows.)
2. Solution of inverse problems (at cost comparable with that achieved for direct problems).
3. Solution of time dependent partial differential equations (with rare local activation of the fine scales, and with full parallel processing possible not only at each time step, but across the entire space-time domain).
4. Solution of integral equations (in $O(n)$ operations, where n is the number of gridpoints).
5. $O(n)$ (instead of $O(n^2)$) calculation of the interactions of n bodies, and any associated steady states.
6. $O(n \log n)$ calculation, and $O(\log n)$ updating, of the main diagonal terms of the inverse, and the value of the determinant, of $n \times n$ systems of grid equations.
7. Global optimization of systems with a multitude of local optima, having multi-scale attraction basins (in which case solution by simulated annealing would require exponential solution time), including in particular discrete optimization.
8. Image reconstruction.
9. Constrained optimization and linear programming, at least for geometrically oriented problems.
10. Fast Fourier transforms of functions on non-uniform grids.
11. Computing behavior of statistical fields (especially where usual Monte-Carlo techniques suffer from critical slowing); calculating thermodynamic limits to accuracy ε in $O(\varepsilon^{-2})$ operations.

12. Derivation of macroscopic equations from microscopic (e.g., atomic) physics (especially where traditional closed-form derivation, as in continuum mechanics, is impossible, and where group renormalization methods are too complicated).

In all these cases the expected *parallel-processing* efficiency of appropriate multi-level computation is poly-log; i.e., the number of unparallelizable steps is theoretically only $O((\log n)^q)$, where n is the number of variables in the system and $1 \leq q \leq 3$. In many cases the *re*-solution of a problem (upon some changes in the data) is much faster yet. The multilevel structure also helps in formulating more efficient discretizations, thus lowering n.

The multi-level methodology starts to impact in a very fundamental way several sciences, including statistical physics (elementary particles in particular), quantum mechanics, general relativity, molecular dynamics (including understanding of proteins), solid state and material sciences, electromagnetics, semiconductors, image processing, fluid dynamics and elastohydrodynamics.

The main ideas of multilevel computations and review of recent developments can be found in [5], [6], [7], [8] and [3].

1.2 The present paper

A new type of multilevel algorithm will be developed here. It is an extension of algorithms developed earlier for tasks such as 4,5 and 10 in the list above. These earlier algorithms are presented again in Secs. 3 and 4 below. Their main idea was first stated in Sec. 8.6 of [4]; their brief description first appeared in Appendix A of [5], and their detailed description, including results of numerical tests, in [9]. Other, more specialized (dealing only with potential-type kernels) multilevel algorithms for performing tasks like 4 and 5 have appeared in [1], [15], [2], [13] and [11].

The innovation of the *new* algorithm, presented in Sec. 5 below, is that it deals with *oscillatory* kernels, important in many wave theories.

Another, well-known approach for performing the same task, in the special case that it can be presented as periodic convolution on a uniform, decomposable grid, is based on the Fast Fourier Transform (see, e.g., [14]). The FFT can in fact be regarded as a special case of multilevel (or multigrid) algorithms (see [10] and [16]). Relations between, and combinations of the algorithms presented in this article and the FFT evaluation of convolutions are discussed in Sec. 6.

The Fourier transform itself can be performed by the algorithms presented below, with the main gain compared to conventional FFT being obtained for the case of irregular set of points and also for cases of narrow bandwidth: see Sec. 7.

2. The Task: Fast Multiplication by a Dense Matrix

We will generally be concerned with the fast calculation of the $\overline{n}$-vector $v = Gu$, given the $\overline{n} \times n$ matrix G and the n-vector u; often $\overline{n} = n$. If the matrix G has arbitrary entries, each of them must enter the calculations, hence $\overline{n}n$ operations must be used, and no way can exist which is significantly faster than the straightforward multiplication. In most applications of interest, however, G has certain special properties that can be used. In particular, in most physical problems $G_{ij} = G(x_i, y_j)$, where $x_i \in \mathbb{R}^{\overline{d}}$ and $y_j \in \mathbb{R}^d$ (i.e., $x_i = (x_i^1, x_i^2, \ldots, x_i^{\overline{d}})$, $y_j = (y_i^1, y_i^2, \ldots, y_j^d)$ where x_i^α and y_j^β are real numbers) and the "kernel" $G(x, y)$ has some smoothness properties; usually $\overline{d} = d$ and $y_j = x_j$. Such kernels arise in many-body interactions, where y_j is the position of a particle with "charge" $u_j = u(y_j)$ (electrostatic charge, mass, etc.), and $v_i = v(x_i)$ is the total effect (potential, force, etc.) of all particles at point x_i. Such kernels also arise when integral transforms, or integral equations, or integrodifferential equations, are discretized on grids. In such a case, unlike that of particles, the points x_i are usually the points of a uniform grid, and so are the points y_j.

In some particular cases, $\{x_i\}$ and $\{y_j\}$ are not scattered in some open region of $\mathbb{R}^{\overline{d}}$ and $\mathbb{R}^d$, but are actually concentrated in some special manifold, such as the *boundary* of a region in $\mathbb{R}^d$. This is typically the case in boundary-element discretizations.

The given task, in any case, is to *evaluate*

$$v(x_i) = \sum_{j=1}^{\overline{n}} G(x_i, y_j) u(y_j), \qquad (i = 1, \ldots, n), \tag{2.1}$$

to a certain accuracy ϵ given $G(x_i, y_j)$ and $u(y_j)$, where $u(y_j)$ are arbitrary but $G(x, y)$ has certain smoothness properties. For convenience, "accuracy ε" will mean that an error upto ε is allowed in each of the summed terms. Generally, each $u(y_j)$ can be a q-vector, each $v(x_i)$ can be a $\overline{q}$-vector, and each $G(x_i, y_j)$ a $\overline{q} \times q$ matrix. For example, $u(y_j)$ can be the 3-component dipole moment at y_j and $v(x_i)$ the 3-component electric field at x_i (thus $\overline{q} = q = 3$). For convenience, the algorithms are described in this paper in terms of the case $\overline{q} = q = 1$, but in the general case they are essentially the same.

The fast algorithms described below are almost the same for particles and for gridpoints, for an open region in $\mathbb{R}^d$ and for other manifolds. We will distinguish, however, between three different situations regarding $G(x, y)$, with increasing complications: smooth kernels, asymptotically smooth kernels, and oscillatory kernels. Most kernels in physics belong to the latter two categories; the case of smooth kernels serves however as the easiest introduction to our technique. It also has important applications, one of which is discussed in Sec. 7 below.

3. Smooth kernels

The algorithm will be based on a sequence of increasingly coarser uniform grids, employed recursively. By "uniform" we mean a rectangular grid, with constant meshsize in each grid direction.

As a first step it will be shown that if $G(x,y)$ is suitably smooth as a function of y, then, in the calculation of (2.1), the set of locations $\{y_j\}_{j=1}^n$ can be replaced by another set $\{Y_J\}_{J=1}^N$ which constitutes a coarser and/or uniform grid.

That is, in case $\{y_j\}_{j=1}^n$ is itself already a set of points of a uniform grid (the *fine* grid), the set $\{Y_J\}_{J=1}^N$ will be chosen as a natural coarsening of that grid, sufficiently extended (as explained below). For example, this coarser grid can be obtained as a *one dimensional* coarsening of the fine grid in one of the grid directions; that is, it is obtained by omitting from the fine grid every other of its hyperplanes (e.g., every other gridline) perpendicular to that direction. Higher dimensional coarsening can also be used, although, as noted by Ton Lubrecht, a simpler and more efficient algorithm is obtained by one dimensional coarsening at a time.

In case the original points $\{y_j\}_{j=1}^n$ are at arbitrary locations (representing, e.g., particles), the grid $\{Y_J\}_{J=1}^N$ will be constructed, whenever possible, as a uniform grid with a comparable "density", i.e., comparable number of points per unit volume. (A modification for the case of a very non-uniform particle density is discussed in Sec. 4.1 below). It is not even necessary in this case that $N < n$; the main gain here is not in reducing the number of points, but in the transition to a uniform grid; transitions to coarser grids will then be performed at later steps of the algorithm. In various cases, however, such as those where $\{y_j\}$ is given on some complicated manifold, it is impossible to cover it by a uniform grid, and then $\{Y_J\}$ will be a coarser grid, not quite uniform, but regular enough to allow convenient interpolations (as discussed below).

A function $f(y)$ will be called *suitably smooth on the scale of the* (coarser) *grid* $\{Y_J\}$ if,

$$(\gamma h)^p |\partial^p f(y)| \le O(\epsilon) \quad \text{for some} \quad p = O(\log \frac{1}{\epsilon}), \tag{3.1}$$

for any y and any p-order derivative ∂^p, where γ is a constant in the range $\frac{1}{2} \le \gamma \le 1$ (see below) and h is the meshsize of $\{Y_J\}$ (assumed for simplicity to be the same, or at least comparable, for all directions; otherwise $(\gamma h)^p \partial^p$ should be replaced in (3.1) by $\gamma^p (h_1^{p_1} \partial_1^{p_1} \cdots h_d^{p_d} \partial_d^{p_d})$, where h_β is the meshsize in direction y^β, $\partial_\beta = \partial / \partial y^\beta$ and $p_1 + \cdots + p_d = p$). This means that f can be interpolated from that grid with $O(\epsilon)$ error by using p-order multi-polynomial interpolation; i.e., interpolation which is done one direction at a time, using at each direction p points (to interpolate from) and polynomial of degree $p-1$. The well-known error estimates (see, e.g., page 276 in [12]) imply that if $\gamma = 1$ in (3.1), $O(\epsilon)$ error is obtained whenever the interpolation target point is in the convex hull of the set of points from which its value is interpolated. If *central* interpolation is used, i.e., if the interpolation order is even and the interpolation points are always chosen as symmetrically as possible around the target point, then $O(\epsilon)$ error is obtained even if $\gamma = 1/2$ in (3.1). Thus, in any of these cases, one can write

$$f(y_j) = \sum_{J \in \sigma_j} \omega_{jJ} f(Y_J) + O(\epsilon), \tag{3.2}$$

where ω_{jJ} are the interpolation coefficients, and $\{Y_J\}_{J\in\sigma_j}$ is a set of gridpoints in the neighborhood of y_j, containing y_j in its convex hull and large enough to perform p-order interpolation. Namely, in case of one-dimensional coarsening, this set contains only p points (which is exactly why such one dimensional coarsening is advantageous). For full d-dimensional coarsening (required in case of particles), the set will include p^d points.

Note that the (coarser) grid $\{Y_J\}$ should be extensive enough to allow performing (3.2) at all given points y_j. Note also that the requirement that $\{Y_J\}$ constitutes a uniform grid is imposed, wherever possible, mainly for the purpose of having simple interpolation formulae. In fact, as a result, for each and any y_j, the set of coefficients ω_{jJ} can usually be calculated in $O(p^d)$ operations. In case $\{y_j\}$ is also a uniform grid, these coefficients repeat themselves: the same set of coefficients applies for any two points y_j and y_k which occupy the same position in their respective coarse cells; hence ω_{jJ} can be read from a small pre-calculated table.

Assuming $G(x, y)$ to be a suitably smooth function of y, so that (3.2) can be applied to it, it follows from (2.1) that

$$v(x_i) = \sum_j \sum_{J \in \sigma_j} \omega_{jJ} G(x_i, Y_J) u(y_j) + O(\epsilon),$$

or

$$v(x_i) = \sum_J G(x_i, Y_J) U(Y_J) + O(\epsilon) \tag{3.3}$$

where

$$U(Y_J) = \sum_{\substack{j \ s.t. \\ J \in \sigma_j}} \omega_{jJ} u(y_j). \tag{3.4}$$

Thus, the coarse-grid function U, defined by (3.4), is obtained by multiplying u with the matrix ω^T, the transposed—or adjoint—of the interpolation matrix ω. We therefore call this operation *adjoint interpolation*, or *anterpolation*. Note that the summation in (3.4) is over all j such that $J \in \sigma_j$. The most efficient programming of (3.4) is to order it primarily by j, not by J. Namely, starting with all $U(Y_J) = 0$, for each j in its turn add $\omega_{jJ} u(y_j)$ to $U(Y_J)$ for all $J \in \sigma_j$. The number of sequential operations to perform anterpolation in this way is $O(np^d)$ for d-dimensional coarsening. (Parallel processing would require some obvious modifications).

Thus, with this number of operations, the original task (2.1) has been replaced by the task (3.3). If $G(x, y)$ is also a suitably smooth function of x, we can also replace the set $\{x_i\}_{i=1}^{\overline{n}}$ by a coarser and/or uniform set $\{X_I\}_{I=1}^{\overline{N}}$. Denoting its meshsize by $\overline{h}$, for any function $\overline{f}(x)$ which is suitably smooth (satisfying (3.1) with $\overline{h}$ instead of h, and perhaps some $\overline{p}$ instead of p) we have

$$\overline{f}(x_i) = \sum_{I\in\overline{\sigma}_i} \overline{\omega}_{iI}\overline{f}(X_I) + O(\epsilon). \tag{3.5}$$

(In most applications the given sets $\{x_i\}$ and $\{y_j\}$ are identical, and $G(x, y)$ has the same smoothness with respect to x and y, in which case the set $\{X_I\}$ can be taken the same as $\{Y_J\}$, and then $\overline{h} = h$, $\overline{\omega}_{iI} = \omega_{iI}$ and $\overline{\sigma}_i = \sigma_i$.) Applying (3.5) to $G(x_i, Y_J)$ in (3.3) we obtain

$$v(x_i) = \sum_{I\in\overline{\sigma}_i} \overline{\omega}_{iI} V(X_I) + O(\epsilon) \tag{3.6}$$

where

$$V(X_I) = \sum_J G(X_I, Y_J)U(Y_J). \tag{3.7}$$

Observe that (3.7) is a coarse (and/or uniform) grid version of (2.1)

The entire algorithm is first to calculate U by the anterpolation (3.4), then to calculate V defined by (3.7), and then perform the interpolation (3.6). The calculation of (3.7) is done by recursion: similarly to (2.1) it is calculated through the use of still coarser grids, as long as G is suitably smooth on the scale of those grids.

For sufficiently smooth kernels the recursion can proceed until such a coarse grid is reached for which the calculation of (3.7) costs little compared with the rest of the algorithm. The cost of the entire algorithm is then essentially the execution of (3.4) and (3.6) on all the grids of the recursion. Since the work decreases geometrically on increasingly coarser (uniform) levels, the total cost is comparable to that of executing (3.4) and (3.6) on the finest level. Thus, *the total cost is $O(np^d + \overline{n}\overline{p}^d)$ in case of particles, and $O(np + \overline{n}\overline{p})$ in case of uniform grids* (employing one dimensional coarsening at each level, alternating of course the coarsening directions).

It should be noted that, in the present case of "sufficiently smooth" kernels, the intermediate levels described above are not really needed: one could interpolate directly from the coarsest grid to the finest (or to the particle locations). The intermediate levels are essential, however, for extending the algorithms to the cases discussed below.

4. Asymptotically Smooth Kernels

Most kernels $G(x,y)$ appearing in physical applications are not smooth throughout; they usually have unbounded derivatives as $y \to x$. On the other hand, in most cases (except for those discussed in Sec. 5 below) the smoothness of G indefinitely increases with $|y-x|$, the distance from y to x. Generally, we will call $G(x,y)$ *asymptotically smooth as a function of* y if, for any positive integer p and any p-order derivative ∂_y^p with respect to $y = (y_1, \ldots, y_d)$,

$$|\partial_y^p G(x,y)| \le C_p r^{g-p}, \tag{4.1}$$

where g is independent of p, C_p depends only on p, and $r = r(x,y)$ is the distance of y from a finite set of points, which may depend on x (but whose number is bounded independently of x); in most cases $r(x,y) = |x-y|$.

Asymptotic smoothness of $G(x,y)$ as a function of x is similarly defined:

$$|\partial_x^p G(x,y)| \le \overline{C}_p \overline{r}^{\overline{g}-p}, \tag{4.2}$$

where $\overline{r} = \overline{r}(x,y)$ is the distance of x from a finite set of points depending on y. Usually, in fact, $G(x,y) = G(x-y)$ and asymptotic smoothness in both x and y is then expressed by $|\partial^p G(x)| \le C_p |x|^{g-p}$, where $|x|^2 = (x^1)^2 + \cdots + (x^d)^2$, so that $\overline{r}(x,y) = r(x,y) = |x-y|$ and $\overline{g} = g$.

If $G(x,y)$ is asymptotically smooth as a function of y, then it clearly satisfies (3.1), but only for $r(x,y) \ge O(h)$. The transition (3.3) to a coarser grid $\{Y_J\}$ can then still be performed, provided a correction is made to account for the large errors in calculating the contribution to $v(x_i)$ from $u(y_j)$ at those points y_j for which

$$\min_{J \in \sigma_j} r(x_i, Y_J) \le c_p h, \tag{4.3}$$

where $c_p = O(C_p^{1/p})$ and hence for all kernels of interest $c_p \le O(p)$. Thus, for each x_i the number of points y_j satisfying (4.3) is at most $O(p^d)$, hence the total work of these corrections is properly bounded (see below).

If $G(x,y)$ is also asymptotically smooth in x, the further transition to (3.6) can also be made. Here large errors are made, and corrections are needed, for the contribution to $v(x_i)$ from $u(y_j)$ for which

$$\min_{I \in \overline{\sigma}_i} \overline{r}(X_I, y_j) \le \overline{c}_p \overline{h}. \tag{4.4}$$

Thus, *if* $G(x,y)$ *is asymptotically smooth in both* x *and* y*, the entire algorithm for calculating* (2.1) *is recursively defined by the following 4 steps.*

(i) Calculate the coarse grid function $U(Y_J)$ by the anterpolation (3.4).

(ii) Compute $V(X_I)$, as defined by (3.7). Since (3.7) is a coarse version of (2.1), its computation is done by the same algorithm, except when $\{X_I\}$ and $\{Y_J\}$ are so coarse that executing (3.7) directly is cheaper. The latter is true when the grids have only $O(p)$ intervals in each direction.

(iii) Interpolate

$$\tilde{v}(x_i) = \sum_{I \in \overline{\sigma}_i} \overline{\omega}_{iI} V(X_I). \tag{4.5}$$

(iv) For each x_i, correct the contribution to $\tilde{v}(x_i)$ from all $u(y_j)$ at points y_j such that (4.3) and/or (4.4) hold. The efficient calculation of these corrections is discussed below.

We want to keep the computational cost of the corrections at most comparable to the cost of steps (i) and (iii), which is at most $O((n+\overline{n})p^d)$. (The lower operation count obtainable for steps (i) and (iii) by one-dimension-at-a-time coarsening is usually not applicable to step (iv). It is natural to assume here $\overline{p} = O(p)$ and $\overline{d} = d$.) This is obtained by fulfilling the following two requirements.

(A) Per point x_i, the number of points y_j which satisfy (4.3) should on the average be $O(p^d)$. Similarly, per each y_j the same should on the average be the number of x_i for which (4.4) is satisfied.

(B) The calculations should be organized so that the corrections cost $O(1)$ operations per pair $(v(x_i), u(y_j))$.

4.1 Requirement A. Variable Particle Density

The first requirement (A) is automatically satisfied if the density of the grids $\{X_I\}$ and $\{Y_J\}$ is comparable with that of $\{x_i\}$ and $\{y_j\}$, respectively. This is obtained by, and is the very reason for, the algorithm having a *sequence* of levels, with bounded (e.g., 1:2) coarsening ratios.

In case of *particles whose density substantially vary over the domain* (e.g., a galaxy of stars with high increase in density toward its center), a modification in fact is needed. Instead of a sequence of increasingly coarser uniform grids which cover the same domain (the domain of all particles), some of the first (i.e., finest) grids in the sequence cover only part of the domain. That is, the coarsening ratio (e.g., 1:2 coarsening in each dimension) of the uniform grids remain the same, but each grid cover only that part of the domain in the interior of which the particle density (averaged over $O(p^d)$ cells) is at least comparable to (i.e., at least a given fraction of) its own density. By the "interior" of a grid we here mean that region to which good p-order interpolation from the grid can conveniently be constructed. As a result, the subdomain covered by any grid is contained in those covered by any coarser grid. (Also, points may be added to coarser grids at the boundary, so that each grid is so removed from the boundary of the next coarser grid that the same central interpolation formulae can everywhere be used between them).

The algorithm with this sequence of non-coextensive uniform grids remains basically as before, but with differently defined points at each level. The finest level $\{y_j\}$ is still the given collection of particle points. The next level $\{Y_J\}$ includes the gridpoints of the finest uniform grid, together with all the particle points not in the interior of that grid. Similarly, each subsequent level includes the gridpoints of the corresponding uniform grid, together with all particle points not in its interior. Steps (i), (iii) and (iv) of the above algorithm are of course executed only at that region (the interior of the current uniform grid), because outside it these steps mean "do nothing".

4.2 Requirement B. Softened Kernels

The main difficulty in satisfying the second requirement (B) is of course not in calculating the correct contribution of $u(y_j)$ to $v(x_i)$, which is simply $G(x_i, y_j)u(y_j)$, but in calculating its erroneous contribution to $\tilde{v}(x_i)$, which should be subtracted out. This contribution is proportional to $u(y_j)$, so it can be denoted $\tilde{G}(x_i, y_j)u(y_j)$.

In the case that $G(x, y) = G(x - y)$ and $\{x_i\}$ and $\{y_j\}$ are uniform grids, the calculations can easily be defined so that $\tilde{G}(x_i, y_j)$ depends only on $x_i - y_j$ and on the position of x_i and y_j relative to their respective coarse-grid cells. Since the vector $x_i - y_j$ is itself on a given uniform grid, and since there are only few different possible positions of a fine gridpoint relative to the coarse cell (e.g., only 2 such positions in a one-dimensional coarsening with 1:2 coarsening ratio), there are actually only $O(p^d)$ values of $\tilde{G}$ different from each other. The values can thus be read directly from a small ($O(p^d)$) pre-calculated table. In fact, instead of the value of $\tilde{G}$ one can store in the table the corresponding values of $G - \tilde{G}$, immediately giving the needed correction. This is the method that was used in the numerical experiments reported in [9].

The calculation of $\tilde{G}(x_i, y_j)$ in other cases, e.g. particles, is less convenient. A simple way out (suggested by Dinshaw Balsara [17]) is to use a *softened kernel*. Namely, perform steps (i), (ii) and (iii) of the above algorithm with a kernel $G^h(x, y)$ instead of $G(x, y)$, where $G^h(x, y) = G(x, y)$ except in a neighborhood of radius $O(h)$ around the singularity, in which $G^h(x, y)$ is constructed so that G^h is everywhere "suitably smooth on scale h" in both x and y (cf. definition (3.1)). The corrections in step (iv) will then be corrections not of interpolation errors—those have the necessary $O(\epsilon)$ bound—but only of the difference $G - G^h$. This of course is a straightforward calculation.

For example, in the quite usual case that $G(x, y) = g(|x - y|)$, the function

$G^h(x,y) = g^h(|x-y|)$ can be defined by

$$g^h(r) = g(r) \qquad \text{for } r \geq p'h$$
$$g^h(r) = \sum_{\ell=0}^{p-1} g_\ell r^{2\ell} \qquad \text{for } 0 \leq r < p'h \tag{4.6}$$

where the coefficients g_ℓ are determined so that $g^h(r)$ and its first $p-1$ derivatives are continuous at $r = p'h$, and where $p' = O(p)$. Usually it turns out that

$$g^h(r) = \tilde{g}(r/h)h^s \qquad \text{for} \qquad 0 \leq r < p'h. \tag{4.7}$$

Note that steps (i) and (iii) are not really modified, since they are independent of G. Note also the dependence of G^h on h; the softened kernel is *different* on different levels, although it may be similar (cf. (4.7)). In practice, though, a softened kernel will usually be used only at the finest level involving particles; at coarser levels, involving only uniform grids, the pre-calculated tables of $G - \tilde{G}$ can normally be used.

When $G(x,y)$ cannot be written as a function of just $x-y$, a softened kernel may not be simple and efficient to use. A *general way* to satisfy Requirement B is to arrange the points x_i in disjoint *blocks*, each block containing all points x_i falling in a cube of linear dimension $O(ph)$, i.e., $O(p^d)$ points per block. For each such block B, a block B' of points y_j is assigned so that, for any $x_i \in B$, $\tilde{G}(x_i, y_j)$ need be calculated only for $y_j \in B'$. The block B', to contain all points y_j satisfying (4.3) and/or (4.4) for any $x_i \in B$, should be considerably larger than B, but it still has $O(ph)$ linear dimension and contains $O(p^d)$ points. The calculation of $\tilde{G}(x_i, y_j)$ proceeds by essentially repeating Steps (i), (ii) and (iii) of the algorithm, but only for points $y_j \in B'$, $x_i \in B$ and the points Y_J and X_I associated with them by anterpolation/interpolation formulas. Step (ii) in this case is of course executed by a direct calculation of (3.7). It is easy to see that each step costs $O(p^{2d})$ operations, hence Requirement B is satisfied.

5. Oscillatory Kernels

5.1 The Task

In various wave theories, in acoustics, in light and other electromagnetic scattering problems, etc., the matrices that arise, either in particle simulations or in discretizations (including boundary-element discretizations), are oscillatory. Generally, the computational task here is the evaluation, to $O(\varepsilon)$ accuracy, of v which, instead of (2.1), has the form

$$v(x_i) = \sum_{j=1}^{n} G(x_i, y_j) e^{ik|x_i - y_j|} u(y_j), \qquad (i = 1, \dots, \overline{n}) \tag{5.1}$$

where $G(x, y)$, as before, is asymptotically smooth in both x and y, and where k is some given positive constant—the wave number. (The i preceding k will always denote $\sqrt{-1}$, unrelated to i appearing elsewhere.) We can assume here $\overline{d} = d$, i.e., $x, y, x_i, y_j \in \mathbb{R}^d$, so that $x = (x^1, \dots, x^d)$ and $|x|^2 = (x^1)^2 + \cdots + (x^d)^2$. The sets $\{x_i\}_{i=1}^{\overline{n}}$, $\{y_j\}_{j=1}^{n}$ are again either gridpoints or arbitrary particle locations, and the same type as before of uniform and/or coarser grids $\{X_I\}_{I=1}^{\overline{N}}$ and $\{Y_J\}_{J=1}^{N}$ are constructed.

Actually, here we will construct the grids $\{X_I\}$ and $\{Y_J\}$ so that (for simplicity) they have the same meshsize h, and so that

$$pkh \le \eta, \tag{5.2}$$

ensuring at least $O(p) = O(\log \frac{1}{\varepsilon})$ meshsizes per wavelength. The constant η will be determined later. Usually the *given* sets $\{x_i\}$ and $\{y_j\}$ would properly resolve the wavelength $2\pi/k$; otherwise (5.1) cannot represent a meaningful discretization of order p. Hence, the uniform grids $\{X_I\}$ and $\{Y_J\}$ constructed to satisfy (5.2) will still have density at most comparable to that of the given sets $\{x_i\}$ and $\{y_j\}$. If on the other hand the given $\{x_i\}$ and $\{y_j\}$ *over*-resolve the wavelength, the uniform grids can be constructed with $pkh \ll \eta$. The algorithm can then start with several coarsening steps of the type described in Sec. 4 above, until the meshsize approaches the bound (5.2), at which stage the algorithm described in this section must be switched on.

Due to (5.2), the "total kernel"

$$G_{\text{TOTAL}}(x, y) = G(x, y) e^{ik|x-y|} \tag{5.3}$$

as a function of y satisfies (3.1) for $r(x, y) \ge O(h)$, and similarly as a function of x, so transition to the coarser grids $\{X_I\}$ and $\{Y_J\}$ could be made in the same way as in Sec. 4 above, with the same corrections (step (iv) there). However, the recursion (step (ii)) to still coarser grids would not be possible: with substantial violation of (5.2) due to coarser meshsizes, the total kernel is no longer suitably smooth. In many cases of convolution, i.e., when $G(x,y) = G(x-y)$, by suitably choosing $\{X_I\}$ and $\{Y_J\}$, the recursion can be replaced by an FFT approach: see Sec. 6. More generally, the recursion can proceed in the modified way presented below.

5.2 One Dimensional Case

To clearly see the main device for dealing with the oscillatory exponential, we first study the one dimensional ($d=1$) case. In this case (5.1) can be written as

$$v(x_i) = e^{-ikx_i} v_+(x_i) + e^{ikx_i} v_-(x_i) \tag{5.4}$$

where

$$v_+(x_i) = \sum_{j \ge i} G(x_i, y_j) e^{iky_j} u(y_j) \tag{5.5a}$$

$$v_-(x_i) = \sum_{j < i} G(x_i, y_j) e^{-iky_j} u(y_j). \tag{5.5b}$$

Introducing the notation

$$u_+(y_j) = e^{iky_j} u(y_j) \tag{5.6a}$$

$$u_-(y_j) = e^{-iky_j} u(y_j) \tag{5.6b}$$

$$G_+(x,y) = \begin{cases} G(x,y) & : y \ge x \\ 0 & : y < x \end{cases}$$

$$G_-(x,y) = \begin{cases} 0 & : y \ge x \\ G(x,y) & : y < x \end{cases}$$

we can rewrite (5.5) in the form

$$v_+(x_i) = \sum_j G_+(x_i, y_j) u_+(y_j) \tag{5.7a}$$

$$v_-(x_i) = \sum_j G_-(x_i, y_j) u_-(y_j) \tag{5.7b}$$

Clearly, if $G(x,y)$ is asymptotically smooth, so are also $G_+(x,y)$ and $G_-(x,y)$. Hence (5.7) can be evaluated by using (twice) the algorithm of Sec. 4 above. *The entire algorithm* for evaluating (5.1) is thus first to calculate (5.6), then evaluate (5.7) by the algorithm of Sec. 4, then compute (5.4).

Thus, in the one dimensional case one needs to add only $O(n + \overline{n})$ operations to the previous algorithm, executed twice. The main device here is to incorporate the oscillating factors into u and v. This is done by separately treating the two directions $x - y > 0$ and $x - y < 0$.

5.3 The Generalized Task

The one-dimensional case is of course *too* simple. At higher dimensions we will need to separate out more directions of $x - y$. The main idea is to increase the number of such directions, by the factor 2^{d-1}, at each coarsening level. A coarser grid of u, and similarly of v, will be defined and calculated for *each* of these directions. Since the number of points in each grid decreases by the factor 2^d at each level of coarsening, the overall work will still be reasonably bounded, as we will see. It is assumed hereinafter that $d \geq 2$.

To make our description fully recursive, we will generalize the task (5.1), giving it the more general form it would acquire at any coarser level of the algorithm.

Let $e_1, e_2, \ldots, e_\lambda$ be the set of directions for our present task. [The original task (5.1) will correspond to the special case $\lambda = 1$]. These directions satisfy $e_\ell \in \sigma^d = \{e \in \mathbb{R}^d : |e| = 1\}$, and they cover the unit sphere σ^d in some *regular* manner, with distances between neighboring directions not larger than some δ. For example, in the two dimensional ($d = 2$) case, $e_\ell = (\cos\theta_\ell, \sin\theta_\ell)$, where $\theta_\ell = 2\pi(\ell - 1)/\lambda$, ($\ell = 1, \ldots, \lambda$), hence $\delta = 2\pi/\lambda$.

We introduce p-order interpolation, of any function φ defined on the grid of directions $\{e_\ell\}_{\ell=1}^{\lambda}$, by writing, for any direction $e \in \sigma^d$,

$$\varphi(e) = \sum_{\ell \in s} w_\ell^s(e)\varphi(e_\ell), \tag{5.8}$$

where $\{e_\ell\}_{\ell \in s}$ is some complete subset of directions close to e. By "complete" we mean that it contains enough points to indeed define a p-order interpolation, and by "close" it is meant that $|e_\ell - e| \leq O(p\delta)$ for all $\ell \in s$. In case $d = 2$, for example, the interpolation is actually a usual (one dimensional polynomial) interpolation from $\{\theta_\ell\}_{\ell \in s}$ to θ, where $e = (\cos\theta, \sin\theta)$, and s denotes a set of p consecutive indices such that θ is in the convex hull of $\{\theta_\ell\}_{\ell \in s}$. Note that the notation explicitly show the dependence of the interpolation weights $w_\ell^s(e)$ on the choice of the subset s: in the final algorithm below a particular $s = s(e)$ will be used, where $\{e_\ell\}_{\ell \in s(e)}$ is the complete subset of directions such that e belongs to its central interval; but in the *derivation* of the algorithm, interpolation from other subsets to the same e will be used as well. Thus, in case $d = 2$, $s(e)$ will denote the subset of p subsequent indices for which e is in the central interval of the subgrid $\{e_\ell\}_{\ell \in s(e)}$. Note also that the interpolation (5.8) is well defined also when λ is small; e.g., in case $d = 2$ it is well defined even if $\lambda < p$, because the set of angles $\{\theta_\ell\}$ is actually periodic. In case $\lambda = 1$, for example, the p-order interpolation would thus collapse to just $w_\ell^s(e) \equiv 1$. [This is the case corresponding to the original task (5.1).]

For any pair (x_i, y_j) we will introduce the notation $e_{ij} = (y_j - x_i)/|y_j - x_i|$, and $s_{ij} = s(e_{ij})$. *The generalized task* can now be defined as the task of evaluating

$$v_\ell(x_i) = \sum_{j=1}^{\overline{n}} G(x_i, y_j) e^{ik|x_i - y_j|} w_\ell^{s_{ij}}(e_{ij}) \sum_{m \in s_{ij}} w_m^{s_{ij}}(e_{ij}) u_m(y_j) \tag{5.9}$$

for all $i = 1, \ldots, \overline{n}$ and all $\ell = 1, \ldots, \lambda$. Clearly, the original task (5.1) is the special case $\lambda = 1$.

Relation (5.2) will also be generalized, taking now the form

$$p \delta k h \le \eta_1, \tag{5.10}$$

where η_1 is proportional to η; e.g., $\eta_1 = 2\pi\eta$ in case $d = 2$. This of course assumes that δ, the size of the interval in the set of selected directions, is reduced proportionately to the increase in h at each coarsening level, i.e., by the factor 1:2, implying an increase of λ by the factor 2^{d-1}.

5.4 Derivation of coarsening

The algorithm below will show how to evaluate (5.9) by using a similar evaluation on the next coarser level. The integer λ, functions such as $s(e)$ and $w_\ell^s(e)$, variables such as x_i,y_j,e_ℓ, v_ℓ and u_m, and notation such as $e_{ij} = (y_j - x_i)/|y_j - x_i|$ and $s_{ij} = s(e_{ij})$ will all have coarse-level counterparts, denoted by the corresponding capital letters: $\Lambda = 2^{d-1}\lambda$, $S(E)$, $W_L^S(E)$, X_I, Y_J, E_L, V_L, U_M, $E_{IJ} = (Y_J - X_I)/|Y_J - X_I|$ and $S_{IJ} = S(E_{IJ})$. In particular, $\{E_L\}$ will be a set of $\Lambda = 2^{d-1}\lambda$ directions, at intervals $\delta/2$ on σ^d, and an interpolation from the subset $\{E_L\}_{L \in S}$ to any other direction E will have the weights $W_L^S(E)$. For the purpose of deriving the coarsening relations, we introduce the auxiliary kernels

$$G_{ij\ell m}(x, y) = G(x, y) e^{ik(|x-y| + e_\ell x - e_m y)} w_\ell^{s_{ij}}(x, y) w_m^{s_{ij}}(x, y), \tag{5.11}$$

where $w_\ell^s(x, y) = w_\ell^s((y-x)/|y-x|)$ and $e_\ell x = e_\ell^1 x^1 + \cdots + e_\ell^d x^d$; for example in two dimensions $e_\ell \cdot x = (\cos\theta_\ell)x^1 + (\sin\theta_\ell)x^2$. From (5.9),

$$v_\ell(x_i) = \sum_j \sum_{m \in s_{ij}} G_{ij\ell m}(x_i, y_j) e^{ik(e_m y_j - e_\ell x_i)} u_m(y_j). \tag{5.12}$$

It will be shown below (Sec. 5.6.A) that for *fixed* indices (i, j, ℓ, m), and in the subdomain where

$$|x - y| \ge h/(\delta\eta_2), \quad (\eta_2 \text{ to be determined}) \tag{5.13}$$

the kernel $G_{ij\ell m}(x,y)$ is suitably smooth (cf. (3.1)) in terms of both x and y. Hence it can be interpolated (cf. (3.2) and (3.5)) from the (coarser) uniform grids $\{X_I\}$ and $\{Y_J\}$, so that (5.12) can be approximated, to $O(\varepsilon)$ accuracy, by

$$\tilde{v}_\ell(x_i) = \sum_j \sum_{m\in s_{ij}} \sum_{I\in\bar{\sigma}_i} \sum_{J\in\sigma_j} \bar{\omega}_{iI}\omega_{jJ} G_{ij\ell m}(X_I,Y_J) e^{ik(e_m y_j - e_\ell x_i)} u_m(y_j).$$

Hence from (5.11),

$$\begin{aligned}\tilde{v}_\ell(x_i) = \sum_I \bar{\omega}_{iI} e^{ike_\ell(X_I-x_i)} \sum_{j,J} \omega_{jJ} G(X_I,Y_J) e^{ik|X_I-Y_J|} \\ w_\ell^{s_{ij}}(X_I,Y_J) \sum_{m\in s_{ij}} w_m^{s_{ij}}(X_I,Y_J) e^{ike_m(y_j-Y_J)} u_m(y_j).\end{aligned} \tag{5.14}$$

It will be shown below (in Sec. 5.6.B) that $u_m(y_j)$ and $e^{ike_m(y_j-Y_J)}$ are sufficiently smooth on the grid of *directions* (i.e., as functions of m), and that hence their interpolation from $\{e_m\}_{m\in s_{ij}}$ can be replaced, to $O(\varepsilon)$ accuracy, by an interpolation from any other close subset of $\{e_m\}$, so that in (5.14) the string $\sum_{m\in s_{ij}} w_m^{s_{ij}}(X_I,Y_J)$ could for example be replaced by $\sum_{m\in s(E_{IJ})} w_m^{s(E_{IJ})}(E_{IJ})$. Alternatively, that interpolation could as well be replaced by an interpolation from the set $\{E_M\}_{M\in S_{IJ}}$, provided $u_m(y_j)$ were defined on $\{E_M\}$ instead of on $\{e_m\}$. To achieve just that we use the fact that $u_m(y_j)$ is suitably smooth on $\{e_m\}$ and interpolate it to a function $\tilde{u}_M(y_j)$ defined on $\{e_M\}$,

$$\tilde{u}_M(y_j) = \sum_{m\in s(E_M)} w_m^{s(E_M)}(E_M) u_m(y_j), \tag{5.15}$$

whereupon (5.14) can thus be replaced by

$$\begin{aligned}\tilde{v}_\ell(x_i) = \sum_I \bar{\omega}_{iI} e^{ike_\ell(X_I-x_i)} \sum_{j,J} \omega_{jJ} G(X_I,Y_J) e^{ik|X_I-Y_J|} \\ w_\ell^{s_{ij}}(E_{IJ}) \sum_{M\in S_{IJ}} W_M^{S_{IJ}}(E_{IJ}) e^{ikE_M(y_j-Y_J)} \tilde{u}_M(y_j).\end{aligned} \tag{5.16}$$

We will also show below (in Sec. 5.6.C) that the final result of the algorithm, upto the permissible $O(\varepsilon)$ error, is not changed when the interpolation coefficients $w_\ell^{s_{ij}}(E_{IJ})$ in (5.16) are replaced by any p-order interpolation coefficients to the same E_{IJ} from any (other) close set. In particular they can be replaced by coefficients of p-order interpolation via the set $\{E_L\}$; i.e. a p-order interpolation from $\{e_\ell\}$ to $\{E_L\}$ followed by a p-order interpolation from $\{E_L\}$ to E_{IJ}. The term $w_\ell^{s_{ij}}(E_{IJ})$ in (5.16) can therefore be replaced by $\sum_L w_\ell^{s(E_L)}(E_L) W_L^{S(E_{IJ})}(E_{IJ})$, where the summation is over all L such that $\ell \in s(E_L)$. Introducing this replacement we obtain

$$\tilde{v}_\ell(x_i) = \sum_{I \in \overline{\sigma}_i} \overline{\omega}_{iI} e^{ike_\ell(X_I - x_i)} \tilde{V}_\ell(X_I), \tag{5.17}$$

where

$$\tilde{V}_\ell(X_I) = \sum_{\substack{L \ s.t. \\ \ell \in s(E_L)}} w_\ell^{s(E_L)}(E_L) V_L(X_I) \tag{5.18}$$

$$V_L(X_I) = \sum_{J=1}^{N} G(X_I, Y_J) e^{ik|X_I - Y_J|} W_L^{S_{IJ}}(E_{IJ}) \sum_{M \in S_{IJ}} W_M^{S_{IJ}}(E_{IJ}) U_M(Y_J) \tag{5.19}$$

$$U_M(Y_J) = \sum_{\substack{j \ s.t. \\ J \in \sigma_j}} \omega_{jJ} e^{ikE_M(y_j - Y_J)} \tilde{u}_M(y_j). \tag{5.20}$$

Since (5.19) is exactly a coarse-grid version of (5.9), we can now define the recursive algorithm.

5.5 The algorithm

Having replaced the task (5.1) by the more general task (5.9), the algorithm for performing it is defined recursively by the following 6 steps. We assume here full d-dimensional coarsening; for one-dimensional coarsening at a time—see Sec. 5.7.

(i) *Angular density interpolation*: using (5.15), calculate $\{\tilde{u}_M(y_j)\}_{j=1}^{n}$ for all $M = 1, \ldots, \Lambda$.

(ii) *Density oscillatory anterpolation*: using (5.20) calculate $\{U_M(Y_J)\}_{J=1}^{N}$ for $M = 1, \ldots, \Lambda$. (For the efficient way of performing anterpolation—see Sec. 3.)

(iii) *Recursion*: compute $V_L(X_I)$, as defined by (5.19), for all $I = 1, \ldots, \overline{N}$ and $L = 1, \ldots, \Lambda$. Since (5.19) is a coarse version of (5.9), its calculation is done by the same algorithm, except when $\{X_I\}$ and $\{Y_J\}$ are already the coarsest grid (see below).

(iv) *Angular field anterpolation*: using (5.18), calculate $\{\tilde{V}_\ell(X_I)\}_{I=1}^{\overline{N}}$ for $\ell = 1, \ldots, \lambda$ (programmed also in the manner explained in Sec. 3).

(v) *Field oscillatory interpolation*: using (5.17), calculate $\{\tilde{v}_\ell(x_i)\}_{i=1}^{\overline{n}}$ for $\ell = 1, \ldots, \lambda$.

(vi) *Field corrections*: For each x_i correct the contribution to $\tilde{v}_\ell(x_i)$ from $u_m(y_j)$ for all y_j satisfying both

$$|x_i - y_j| < h/(\delta\eta_2) \tag{5.21a}$$

and

$$\left| \frac{y_j - x_i}{|y_j - x_i|} - e_\ell \right| \le \eta_3 p\delta, \tag{5.21b}$$

where the constant η_3 is defined in (5.22). Corrections are of course due only for the relevant values of m, i.e., those satisfying

$$\left| e_m - \frac{y_j - x_i}{|y_j - x_i|} \right| \le \eta_3 p\delta. \tag{5.21c}$$

The ways these corrections can be calculated efficiently is described in Sec. 4.2 above. (Softened kernels are usually used on the finest level only, if at all, hence softening can be done to G, not to G_{TOTAL}, since $e^{ik|x-y|}$ still has on that level the required smoothness.)

Requirement (5.21a) results of course from (5.13). The restriction of y_j to the range (5.21b) is possible since outside that range $w_\ell^{s_{ij}}(e_{ij}) = w_\ell^{s_{ij}}(X_I, Y_J) = 0$, so the contributions of $u_m(y_j)$ to both $v_\ell(x_i)$ and $\tilde{v}_\ell(x_i)$ vanish (cf. (5.9) and (5.14)). Within that range of y_j the values of m for which the contribution of $u_m(y_j)$ does not vanish is given by (5.21c), because for other values $w_m^{s_{ij}}(e_{ij}) = w_m^{s_{ij}}(X_I, Y_J) = 0$ (cf. again (5.9) and (5.14)).

The coarsest grid is reached when subdomain (5.21a) is comparable to the entire problem domain. In that case (5.19) is calculated directly.

5.6 Proofs

A. We first need to show (cf. Sec. 5.4) that, in the subdomain (5.13), the kernel $G_{ij\ell m}(x, y)$, defined by (5.11), is a suitably smooth function of y on the scale of the grid $\{Y_j\}$, i.e., that it satisfies a relation like (3.1). Smoothness in x can be proved similarly. The proof is needed of course only for that subdomain over which $G_{ij\ell m}(x, y)$ is actually interpolated, i.e., for points x at distance upto $C_1 ph$ from x_i, and y at similar distance from y_j, and for m (and similarly ℓ) such that $w_m^{s_{ij}}(x_i, y_j)$ does not vanish, i.e., $|e_m - e_{ij}| \le C_2 p\delta$. This implies, under (5.13), that

$$\left| \frac{y - x}{|y - x|} - e_m \right| \le \eta_3 p\delta, \quad (\eta_3 = C_2 + 2C_1\eta_2). \tag{5.22}$$

The constant C_1 and C_2 are easy to compute; in case $d = 2$, for example, $C_1 = 2^{-1/2}$, $C_2 = .5$.

Since $G(x,y)$ is assumed to be suitably smooth, it is enough to show such smoothness for each of the other factors of $G_{ij\ell m}(x,y)$, i.e., to demonstrate that $\varphi_m(x,y) = e^{ik(|x-y|-e_m y)}$ and $w_m^{s_{ij}}(x,y)$ are suitably smooth functions of y. The smoothness of $w_\ell^{s_{ij}}$ is proved similarly.

To check the smoothness of φ_m, observe that, by (5.22),

$$\begin{aligned} \left|\frac{\partial \varphi_m(x,y)}{\partial y^\beta}\right| &= \left| ik\left(\frac{y^\beta - x^\beta}{|y-x|} - e_m^\beta\right)\varphi_m\right| \\ &\le \eta_3 p\delta k. \end{aligned} \tag{5.23}$$

Each additional differentiation produces, at worst, a similar factor, hence $|h^p\partial^p\varphi(x,y)| \le (\eta_3 hpk\delta)^p \le (\eta_1\eta_3)^p$, due to (5.10). This implies the needed relation analogous to (3.1), provided η (hence η_1) and η_2 (hence η_3) are chosen so that $\eta_1\eta_3 < 1$. Reasonable choices in case $d = 2$, for example, are $\eta_2 = .12$ and $\eta = .04$, corresponding to $\eta_1\eta_3 = .21$. Since these are worst-case estimates, the practical values of η and η_2 can be substantially larger.

To see the smoothness of $w_m^{s_{ij}}(x,y)$, take again the case $d = 2$. Let $(y-x)/|y-x| = (\cos\theta, \sin\theta)$ and observe that

$$\begin{aligned} \frac{\partial}{\partial y^\beta} w_m^{s_{ij}}(x,y) &= \frac{\partial}{\partial\theta} w_m^{s_{ij}}(\cos\theta,\sin\theta)\frac{\partial\theta}{\partial y^\beta} \\ &\le \eta_2 O(h^{-1}), \end{aligned} \tag{5.24}$$

since the first factor in (5.24) is $O(\delta^{-1})$, while the second is $O(|y-x|^{-1}) \le \eta_2 O(h^{-1}\delta)$, due to (5.13). Each additional differentiation with respect to y yields essentially another such $\eta_2 O(h^{-1})$ factor, since further differentiation of the $\partial\theta/\partial y^\beta$ term in (5.24) produces a much smaller contribution. It can thus be shown that, with η_2 chosen small enough, the requirement analogous to (3.1) is satisfied. ∎

B. Next it should be proved that $\{u_m(y_j)\}_{m=1}^{\lambda}$ and $\{e^{ike_m(y_j-Y_J)}\}_{m=1}^{\lambda}$, for fixed y_j and Y_J such that $|y_j - Y_J| \le O(ph)$, are suitably smooth on the grid of integers $\{m\}$ (mod λ). For the exponential function, this follows immediately from (5.10). For $u_m(y_j)$ we prove this by induction on the levels. On the finest ($\lambda = 1$) level this is trivial. Each higher level is formed from the next lower one, so to prove our assertion by induction we assume $\{u_m(y_j)\}_1^\lambda$ to be suitably smooth on $\{m\}$, and we need to show that $\{U_m(Y_J)\}_1^\Lambda$ is suitably smooth on $\{M\}$.

The function $\{\tilde u_M(y_j)\}_1^\Lambda$, defined by (5.15), is certainly suitably smooth on grid $\{M\}$, because it is an interpolation, of exactly the desired order, of the function $\{u_m(y_j)\}_1^\lambda$, which is smooth even on the scale of a grid twice coarser. The function $\{e^{ikE_M(y_j-Y_J)}\}_1^\Lambda$, too, is suitably smooth, due to (5.10). Hence, by (5.20), so also is $\{U_M(Y_J)\}_1^\Lambda$. ∎

C. Finally it should be shown that the results of the algorithm, to $O(\varepsilon)$ accuracy, are not affected by a set of changes $\{\delta v_\ell\}_{\ell=1}^{\lambda}$ introduced to $\{v_\ell(x_i)\}_{\ell=1}^{\lambda}$

at any fixed point x_i, if these changes are such that $\sum_\ell \delta v_\ell \psi_\ell = O(\varepsilon)$ for any function $\{\psi_\ell\}_{\ell=1}^{\lambda}$ which is suitably smooth on the grid of integers $\{\ell\}$ (mod λ). This is trivially true for the original level ($\lambda = 1$). To prove it by induction on the level number, we assume it to be true for fine level changes $\{\delta v_\ell\}$, and we need to prove its validity for changes $\{\delta V_L\}_{L=1}^{\Lambda}$ introduced to $\{V_L(X_I)\}_{L=1}^{\Lambda}$ at any fixed X_I; i.e., given that

$$\sum_{L=1}^{\Lambda} \delta V_L \; \Psi_L = O(\varepsilon) \quad \text{for any suitably smooth} \quad \{\Psi_L\}_1^{\Lambda}, \tag{5.25}$$

we need to show that the effect of these changes on the final results is $O(\varepsilon)$.

Indeed, by (5.17)–(5.18), the changes $\{\delta V_L\}$ cause fine level changes

$$\delta v_\ell(x_i) = \overline{\omega}_{iI} e^{ike_\ell(X_I - x_i)} \sum_{\substack{L \ s.t. \\ \ell \in s(E_L)}} w_\ell^{s(E_L)}(E_L)\delta V_L, \quad (\ell = 1, \ldots, \lambda)$$

at each point x_i such that $I \in \overline{\sigma}_i$. Hence, for any x_i and any function $\{\psi_\ell(x_i)\}_{\ell=1}^{\lambda}$,

$$\sum_{\ell=1}^{\lambda} \delta v_\ell(x_i)\psi_\ell(x_i) = \overline{\omega}_{iI} \sum_L \delta V_L \Psi_L \tag{5.26}$$

where

$$\Psi_L = \sum_{\ell \in s(E_L)} w_\ell^{s(E_L)}(E_L)\tilde{\psi}_\ell$$

and

$$\tilde{\psi}_\ell = \psi_\ell(x_i) e^{ike_\ell(X_I - x_i)}.$$

If $\{\psi_\ell(x_i)\}$ is suitably smooth on $\{\ell\}$, so is also $\tilde{\psi}_\ell$ (by (5.10), since $|X_I - x_i| \le O(ph)$). Hence $\{\Psi_L\}$, which is just a p-order interpolation of $\{\tilde{\psi}_\ell\}$ to a grid twice finer, is suitably smooth on $\{L\}$. Hence, by (5.25) and (5.26),

$$\sum_{\ell=1}^{\lambda} \delta v_\ell(x_i)\psi_\ell(x_i) = \overline{\omega}_{iI} O(\varepsilon). \tag{5.27}$$

We have thus shown that for every $\{\psi_\ell(x_i)\}$ suitably smooth, (5.27) holds. By the induction hypothesis we can therefore conclude that the changes $\delta v_\ell(x_i)$, caused by δV_L, will have only $O(\varepsilon)$ effect on the final results. ∎

5.7 Total work

Steps (i) and (iv) of the algorithm require $O(\lambda np)$ operations each. Steps (ii) and (v) cost $O(\lambda np^d)$ in case of particles and $O(\lambda np)$ in case of grids with one-dimension-at-a-time anterpolation/interpolation. (Steps (i), (iv) and (v) are of course performed only once per d such one-dimensional steps (ii) and (iv).)

But the main cost is that of step (vi). By (5.21a–b), for each i and ℓ, corrections are made from $O(\delta^{-d})O((p\delta)^{d-1}) = O(p^{d-1}\delta^{-1})$ points y_j, and, by (5.21c), for each such y_j this is made for $O(p)$ values of m. Since there are n values of i and $\lambda = O(\delta^{1-d})$ values of ℓ, the overall work of step (vi) is $O(\delta^{-d}np^d)$. Now, this step is performed at each coarsening level. At each such level δ is reduced by the factor $1/2$ while n decreases by the factor 2^d. Hence, unlike the non-oscillatory case, **the work remains the same at all levels**. The same is also the cost of the direct evaluation of (5.19) on the coarsest level. Hence, the total cost of the algorithm is mainly the number of levels times the cost of step (vi) on, e.g., the finest (original) level, where $\delta = O(1)$ and n is the number of points in the original task (5.1). Conclusion: *The calculation of* (5.1) *to* $O(\varepsilon)$ *accuracy costs* $O(np^d \log n)$ *computer operations.*

This count is based on the assumption that uniform grids with density comparable to that of the given sets $\{x_i\}$ and $\{y_j\}$ have meshsize h_0 comparable with the meshsize h of $\{X_I\}$ and $\{Y_J\}$. More generally, n should be replaced by $N = O((h_0/h)^d n)$, where $h = O((pk)^{-1})$ (see (5.2) and the discussion following it), and the operation count becomes

$$O((kh_0)^d p^{2d} n \log n) = O((kh_0)^d (\log \frac{1}{\epsilon})^{2d} n \log n). \qquad (5.28)$$

In one dimension ($d = 1$), the cost is only $O(n \log \frac{1}{\varepsilon})$ operations (see Sec. 5.2).

6. FFT Evaluation of Convolutions

A special but important class of kernels are those of the form $G(x, y) = G(x - y)$, in which case the task (2.1) is called *convolution*. The task (5.1) is then of coarse also a convolution. Most such convolutions can be performed by an approach which combines the one described above with another based on FFT. For some cases, especially in performing (5.1), better efficiency can thus be gained.

The convolution will be called *periodic*, with period $b = (b^1, \ldots, b^d)$, if, for any integers $\nu^1, \ldots, \nu^d$, $G(z + (\nu^1 b^1, \ldots, \nu^d b^d)) = G(z)$ for all z and if all points x_i and y_j fall in one period, i.e., in a rectangular parallelopiped of dimensions $b^1 \times \cdots \times b^d$. A grid will be called *decomposable* (with respect to the period b) if it consists of all points of the form $a + (i_1 h_1, \ldots, i_d h_d)$, where $a \in \mathbb{R}^d$ and $(h_1, \ldots, h_d) \in \mathbb{R}^d$ are fixed and each i_β is an integer in the range $1 \le i_\beta \le N_\beta = b^\beta / h_\beta$, and if each N_β is a power of 2 (or some other highly composite number). The task (2.1), and hence also (5.1), will be called *decomposable periodic convolution* if $G(x,y) = G(x-y)$ is periodic and both $\{x_i\}$ and $\{y_j\}$ are decomposable grids with respect to the period b.

A decomposable periodic convolution (2.1) can be executed by the following *Fourier method* (see, e.g., [14]): Calculate the Fourier transforms $\hat{G}(\xi)$ and $\hat{u}(\xi)$ of G and u respectively, then the inverse transform of $\hat{v}(\xi) = \hat{G}(\xi)\hat{u}(\xi)$ will give the desired function v (times a constant).

Each of the three transforms in this algorithm is a d-dimensional one, and can be calculated by a sequence of d one-dimensional transforms, each of which uses an FFT procedure for every row in the grid. The total operation count of such an algorithm is about $3C_* dn \log n$, where $C_* N_\beta \log N_\beta$ is the operation count of the FFT on N_β points. Note that this algorithm does not require any smoothness property of the kernel.

A *non-periodic* convolution can often be completed into a periodic one by adding points. In particular, if the given $\{x_i\}$ and $\{y_j\}$ are scattered in some open domain of $\mathbb{R}^d$, one can contain that domain in a rectangular parallelopiped of dimensions $a^1 \times \cdots \times a^d$, and then extend it to a rectangular parallelopiped P of dimensions $b^1 \times \cdots \times b^d$, where $b^\beta = 2a^\beta$, $(\beta = 1, \ldots, d)$. It is straightforward to extend $G(x,y) = G(x-y)$ to all points $x, y \in P$ so that the extended kernel is periodic with the period b. There are of course cases where such a transition to a periodic convolution is *not* possible (without increasing the volume by orders of magnitude); e.g., when $\{x_i\}$ and $\{y_j\}$ are defined on more complicated manifolds, such as boundaries.

If the convolution is periodic but its grids are *not decomposable*, one can use (one level of) the algorithm of Sec. 4, choosing the next coarser grids $\{X_I\}$ and $\{Y_J\}$ to be decomposable. Instead of Step (ii) of the algorithm (the recursion), one can then use the Fourier method. This approach can be used also in the oscillatory case (5.1), provided the grids $\{X_I\}$ and $\{Y_J\}$ are constructed so that their meshsize h also satisfies (5.2). The operation count of such an algorithm in dimension $d > 2$ is

$$\begin{aligned} O(Np^d + N \log N) &= O(n(kh_0)^d p^{2d} + n(kh_0 p)^d \log n) \\ &= O(n(kh_0 \log \frac{1}{\varepsilon})^d ((\log \frac{1}{\varepsilon})^d + \log n)), \end{aligned} \tag{6.1}$$

where h_0^{-d} is the number of gridpoints $\{x_i\}$ and $\{y_i\}$ per unit volume (cf. Sec. 5.7). In the non-oscillatory case, and also in the one dimensional ($d = 1$) oscillatory case, the operation count is

$$O(Np^d + N \log N) = O(n(\log \frac{1}{\varepsilon})^d + n \log n). \tag{6.2}$$

For low enough accuracy (relatively large ε), the work (6.2) exceeds the $O(n(\log\frac{1}{\varepsilon})^d)$ work required by the full algorithm of Sec. 4. At high accuracy their work is approximately the same: the most expensive part (the transition to uniform grids) is common to them, except in cases of variable density (cf. Sec. 4.1), or when the uniform grids need to be much larger to obtain periodicity and decomposability, in which cases, again, the algorithm of Sec. 4 will be faster. If, however, the given convolution is already periodic and decomposable, the pure Fourier method can be used, costing only $O(n\log n)$ operations, and will be faster at high enough accuracy (small ε); indeed, its accuracy is only limited by round-off errors.

In the oscillatory case, the work (6.1) would usually be less than (5.28), so the Fourier method should be used whenever possible. The types of problems in which it *cannot* be used, so that the full algorithm of Sec. 5.1 should be employed, are those in which $G(x,y)$ cannot be represented as $G(x-y)$, or where the points $\{x_i\}$ and $\{y_j\}$ are given on complicated manifolds (boundaries).

Even when the Fourier method can be employed and is faster, the full multi-level coarsening may sometimes be preferred because it better fits other tasks one may like to perform along with the evaluation of Gu, such as solving the equations $Gu = v$, or performing multilevel Monte-Carlo or processes governed by a Hamiltonian of the form u^*Gu.

7. Fourier Transform on Arbitrary Sets

Consider the task of calculating the Fourier transform

$$\tilde{v}(x_m) = \tilde{A}\sum_{j=1}^{n} e^{ix_m y_j}\tilde{u}(y_j), \quad m = 1,\ldots,\overline{n}, \tag{7.1}$$

on arbitrary sets $\{x_m\}$ and $\{y_j\}$ of real numbers. Define the bandwidth and the conjugate bandwidth, respectively, by

$$B = \max_{1\le j\le n} y_j - \min_{1\le j\le n} y_j \quad \text{and} \quad \overline{B} = \max_{1\le m\le \overline{n}} x_m - \min_{1\le m\le \overline{n}} x_m .$$

Introducing $x_* = (\max_m x_m + \min_m x_m)/2$, $y_* = (\max_j y_j + \min_j y_j)/2$, $v(x_m) = e^{-ix_m y_*}\tilde{v}(x_m)$, $u(y_j) = e^{-ix_* y_j}\tilde{u}(y_j)$ and $A = e^{-ix_* y_*}\tilde{A}$, the task (7.1) can be rewritten as

$$v(x_m) = A\sum_{j=1}^{n} e^{i(x_m - x_*)(y_j - y_*)}u(y_j), \quad m = 1,\ldots,\overline{n}. \tag{7.2}$$

Using the terminology of Sec. 3, the kernel $G(x,y) = Ae^{i(x-x_*)(y-y_*)}$ is a suitably smooth function of y (respectively x) on a grid with meshsize h (respectively $\overline{h}$), if

$$h \le \eta_* \overline{B}^{-1} \quad \text{and} \quad \overline{h} \le \eta_* B^{-1} \tag{7.3}$$

(see the choice of η_* below). The calculation of (7.2) to accuracy ε can then be replaced by the calculation of (3.7), where $\{X_I\}$ and $\{Y_J\}$ are uniform grids with meshsizes $\overline{h}$ and h respectively, $U(Y_J)$ is formed from u by the anterpolation (3.4), and $v(x_m)$ is obtained from $V(X_I)$ by the interpolation (3.6).

To roughly minimize the work, the order of both the interpolation and anterpolation should be taken as $p \approx \log \frac{1}{\varepsilon}$, and in (7.3) the value $\eta_* = 4/e = 1.47$ should approximately be taken. The overall number of points in $\{X_I\}$, and also in $\{Y_J\}$, is then $N \geq B\overline{B}/\eta_*$. By choosing in this range the smallest N which is a power of 2, (3.7) can be executed by a conventional FFT procedure, costing $O(N \log N)$ computer operations. Thus, *the overall operation count is*

$$O((n+\overline{n})\log\frac{1}{\epsilon} + B\overline{B}\log(B\overline{B})). \tag{7.4}$$

Even when the given sets $\{x_i\}$ and $\{y_j\}$ are both equidistant, this procedure to evaluate (7.1) is preferable to direct FFT whenever the bandwidth product $B\overline{B}$ is small and the desired accuracy is not too high.

Instead of calculating (3.7) by a conventional FFT, one could split the set $\{X_I\}$ into two equal (or about equal) subsets with about half the conjugate bandwidth each, hence allowing the calculation of each of them on a y grid with meshsize $2h$. Each of these two tasks can similarly be split into two tasks that can be calculated on meshsize $4h$, etc. This would give a new fast way of calculating Fourier transforms; but the above approach, which uses a conventional FFT after the first step, is faster.

The extension to a general dimension (arbitrary $x_i, y_j \in \mathbb{R}^d$) is straightforward.

References

[1] A.W. Appel, *SIAM J. Sci. Stat. Comput.* **6**, 85 (1985).

[2] J. Barnes and P. Hut, *Nature* **324**, 446 (1986).

[3] R. Ben-Av, A. Brandt, M. Harmatz, E. Katzenelson, P. Lauwers, S. Solomon and K. Wolowesky, Fermion simulations using parallel transported multi-grid. Submitted to *Phys. Rev. Lett.* Aug. 1990.

[4] A. Brandt, Guide to multigrid development. In: *Multigrid Methods* (Proc. Köln-Porz, 1981), edited by W. Hackbusch and U. Trottenberg, (*Lecture Notes in Math.* **960**, Springer Verlag, 1982), p. 220.

[5] A. Brandt, Multilevel computations: review and recent developments. In: *Multigrid Methods: Theory Application and Super-Computing* (S.F. McCormick, ed.), Marcel-Dekker, 1988, p. 35.

[6] A. Brandt, The Weizmann Institute Research in Multilevel Computation: 1988 Report. In: Proc. *4th Copper Mountain Conf. on Multigrid Methods*, (J. Mandel et al, eds.), *SIAM*, 1989, p. 13.

[7] A. Brandt, The scope of multiresolution iterative computations, *SIAM News* **23**, 8 (1990).

[8] A. Brandt, M. Galun and D. Ron, Optimal multigrid algorithms for calculating critical temperature and thermodynamic quantities. Report, The Weizmann Institute of Science, Rehovot, Israel, August 1990.

[9] A. Brandt and A.A. Lubrecht, Multilevel matrix multiplication and fast solution of integral equations, *J. Comp. Phys.* **90**, 348 (1990).

[10] W.L. Briggs and V.E. Henson, The FFT as a multigrid, *SIAM Rev.* **32**, 252 (1990).

[11] J. Carrier, L. Greengard and V. Rokhlin, *SIAM J. Sci. Stat. Comput.* **9**, 669 (1988).

[12] G. Dahlquist and A. Björck, *Numerical Methods*, Prentice-Hall, 1974.

[13] L. Greengard and V. Rokhlin, *J. Comput. Phys.* **73**, 325 (1987).

[14] P. Henrici, Fast Fourier methods in computational complex analysis, *SIAM Rev.* **21**, 481 (1979).

[15] V. Rokhlin, *J. Comp. Phys.* **60**, 187 (1985).

[16] Y. Tsuria, Bilevel Properties of Fourier Transform, M.Sc. Thesis, The Weizmann Institute of Science, Rehovot, Israel, Nov. 1988.

[17] D. Balsara and A. Brandt, in: Multigrid Methods III (Proc. 3rd European Conf. on Multigrid Methods, Bonn, October, 1990) (W. Hackbusch and U. Trottenberg, eds.), Springer, 1991.