Efficient $O(N)$ Integration For All-Electron Electronic Structure Calculation Using Numeric Basis Functions

V. Havu$^a,b,*$ V. Blum$^b$ P. Havu$^b$ M. Scheffler$^b$

$^a$Institute of Mathematics, Helsinki University of Technology - TKK, Finland
$^b$Fritz Haber Institute of the Max Planck Society, Berlin, Germany

Abstract

We consider the problem of developing $O(N)$ scaling grid based operations needed in many central operations when performing electronic structure calculations with numeric atom-centered orbitals as basis functions. We outline the overall formulation of localized algorithms, and specifically the creation of localized grid batches. The choice of the grid partitioning scheme plays an important role in the performance and memory consumption of the grid based operations. Three different top-down partitioning methods are investigated, and compared with formally more rigorous yet much more expensive bottom-up algorithms. We show that a conceptually simple top-down grid partitioning scheme achieves essentially the same efficiency as the more rigorous bottom-up approaches.

Key words: electronic structure theory, density functional theory, atom-centered basis functions, numerical integration grid, spatial partitioning

PACS: 02.60.Jh, 71.15.Ap, 71.15.Dx

1 Introduction

Computational electronic structure theory (EST) (e.g., density functional theory [1], Hartree-Fock, or many-body perturbation theories such as MP2 and GW) is playing an increasingly prominent role in science and technology. Traditionally, a large variety of discretization methods via basis sets has been available for the Kohn-Sham equations [2], on which many practical implementations are based. In

* Corresponding author: Address: Helsinki University of Technology, Institute of Mathematics, P.O. Box 1100, FI-02015 TKK, Finland
Email address: Ville.Havu@tkk.fi (V. Havu).
particular, a successful basis choice employed in a variety of all-electron implementa-
tions [3–8] are numeric atom-centered orbitals (NAOs). These offer an efficient
prescription that can be used for accurate full-potential, all-electron calculations of
periodic and non-periodic systems on equal footing.

The present paper examines the real-space, three-dimensional integration grid in-
frastucture needed to optimally handle NAO-based all-electron EST. While all al-
gorithms are described and tested with NAOs in mind, we note that the same algo-
rithms and basic observations will also apply to many other all-electron basis set
prescriptions for EST.

The use of NAOs entails inevitably a real-space grid that is required to perform
the basic operations of EST: integration of the Hamiltonian matrix, update of the
electron density, and solution of the electrostatic potential (we do not specifically
address the latter in this paper). In practice, these grid-based operations are the
main cost associated with NAOs for all but the largest systems (≳ 1000 atoms).
The reason is that the number of NAOs needed to achieve an accurate solution is
relatively low (≲ 50 basis functions / atom for meV / atom total-energy accuracy
[5,8]) compared to the number of basis functions required in other methods. Conse-
quently, even an $O(N^3)$ solution of the discretized Kohn-Sham eigenvalue problem
does not dominate in all but the largest systems.

By using explicit confining potentials in their construction [6,7,9–13], NAOs give
a natural platform to make the grid based operations scale $O(N)$ by properly local-
izing all basis functions. Specifically, the good and eventually nearly linear-scaling
performance of NAOs in the integrations and in the update of the electron den-
sity results from spatial localization of the basis functions combined with a careful
division of the grid into spatially localized regions for the grid based operations
[14–16]. In this paper we (1) reiterate that there is a scheme that via basis and
grid localization results in nearly linear-scaling grid operations for NAOs, (2) study
the problem of partitioning the grid into localized batches in detail using three top-
down grid partitioning methods, and (3) show that a formally more rigorous class of
bottom-up grid partitioning methods is too expensive for our purposes. Finally, we
compare the computational efficiency of these methods for different prototypical
applications (polyalanine chains, Cu-clusters and Au-surfaces).

Before moving on to detailed description and analysis of our approach, we show
what properly implemented [8] basis and grid localization schemes should achieve
in terms of scaling, in the case of NAOs. Figure 1 shows the computational scaling
for a series of real molecules, specifically a series of fully extended conforma-
tions of polyalanine peptide molecules. In Fig. 1(a), a single alanine aminoacid is
shown, identified by its CH$_3$ side group. Figure Fig. 1(b) shows how several ala-
nine units are linked together into a peptide chain. The conformation shown here
is known as “fully extended.” Because it is essentially linear, the distance between
successive basis function centers in this system class grows rapidly with length,
well suited to demonstrate the performance of a localized basis function method in the limit of large systems. Finally, Fig. 1(c) shows the scaling as the polyalanine chain length increases from a single residue of 18 atoms (the terminations are included in this count) up to a molecule of sixty residues with 603 atoms. Figure 2 shows the parallel scalability for our approach in the case of the 603 atom fully extended polyalanine chain. We emphasize that all computational settings reflect tightly converged production settings as detailed in Table 1 below. It can be seen that the grid based operations (update of the electron density using a density matrix and integration of the Hamiltonian matrix) are nearly linear scaling and the overall scaling reaches the value $O(N^{1.3})$ for this range of systems. Since our focus is on those methodological steps that involves localized basis functions in real space, the solution of the Hartree potential and the eigenvalue problem are shown for illustration only, but have not yet been adapted for the same rigorous scaling requirements as the (without localization) dominant basis-dependent steps. All our results are computed using a recently developed NAO-based implementation for electronic structure calculations [8] and the timings are obtained on IBM p575 Power5+ system using 16 CPU cores, except for Figs. 1(c) and 2 that were calculated on a Cray XT/4 system.

2 Localization of the basis functions

The general form of a NAO basis function is given by

$$\varphi_{i,l,m} = \frac{u_i(r)}{r} Y_{lm}(\theta, \phi)$$

where $Y_{lm}$ is a spherical harmonic and $u_i(r)$ is a real-valued function representing the radial shape of the atom-centered basis function. In our actual implementation, we use as angular momentum functions $Y_{lm}$ the real and imaginary parts of the complex spherical harmonics, leading to only real-valued functions $\varphi_{i,l,m}$. The radial function, $u_i(r)$, is usually taken to be a solution to a Schrödinger equation of atomic, ionic or hydrogenic type, and is localized using a smooth cut-off potential so that $u_i(r)$ becomes zero beyond some cutoff radius, $r_{\text{cutoff}}$. To illustrate the effect of this localization, let us consider the two operations we are focusing on: (1) the integration of the Hamiltonian matrix and (2) the update of the electron density.

(1) The Hamiltonian matrix, $h$, has the entries $h_{ij} = \int_{\mathbb{R}^3} \varphi_i \hat{h} \varphi_j dr^3$ where $\varphi_i$ and $\varphi_j$ are the basis functions. If implemented without localization, the integration leads to $O(N^3)$ scaling of the computation, since for each pair of basis functions we need to run over all the grid. However, this can be avoided by enforcing a localization of the basis functions using a cutoff potential. As the spatial extent of the considered system grows, the number of non-zero basis functions that need to be consider for each grid point levels off to a constant value, leaving only the number of grid points as a growth factor for the complexity of the operation.
Fig. 1. Scalability for fully extended polyalanine molecules with respect to system size. (a) Single alanine aminoacid. (b) Segment of a fully extended polyalanine chain (five \CH_3 residues). (c) Timings in seconds for one self-consistency iteration of finite, fully extended polyalanine chains of increasing length. The scaling exponents of the operations are: Total: \(O(N^{1.3})\), electron density update using Eq. (6): \(O(N^{1.2})\) and using Eq. (5): \(O(N^{1.7})\) solution of the Hartree potential with an atom-wise multipole decomposition: \(O(N^{1.4})\) integration of the Hamiltonian matrix: \(O(N^{1.2})\), ScALAPACK based solution of the eigenvalues and eigenvectors \(O(N^{1.6})\).

(2) The electron density is defined by

\[
n(r) = \sum_{k=1}^{n_{\text{states}}} f_k |\psi_k(r)|^2
\]

where \(f_k\)'s are the occupation numbers, and the sum runs over all Kohn-Sham
Fig. 2. Scalability for fully extended polyalanine of sixty residues (603 atoms, shown at the top) with respect to number of CPU cores. Timings are in seconds for one self-consistency iteration.

eigenstates, which are given by

$$\psi_k(r) = \sum_{i=1}^{n_{basis}} c^k_i \varphi_i(r)$$  \hspace{1cm} (3)

so that

$$n(r) = \sum_{k=1}^{n_{occ}} \sum_{i=1}^{n_{basis}} c^k_i \varphi_i(r) \sum_{r} (c^k_{ij})^* \varphi_j(r)$$  \hspace{1cm} (4)

$$= \sum_{k=1}^{n_{occ}} \sum_{i=1}^{n_{basis}} \sum_{j=1}^{n_{basis}} c^k_{ij} \varphi_i(r) (c^k_{ij})^* \varphi_j(r)$$  \hspace{1cm} (5)

$$= \sum_{i=1}^{n_{basis}} \sum_{j=1}^{n_{basis}} D_{ij} \varphi_i(r) \varphi_j(r).$$  \hspace{1cm} (6)

Here $D_{ij} = \sum_{k=1}^{n_{occ}} f_k c^k_{ij} (c^k_{ij})^*$ is the density matrix. Again, if implemented naively with no regard to localization, Eq. (4) leads to a $O(N^3)$ operation count. Taking into account localization the number of non-zero basis functions at each grid point will eventually level off as for the Hamiltonian as the system size grows and the complexity will be reduced to an $O(N^2)$ algorithm Eq. (5). After summing up the density matrix, the actual grid-based density update Eq. (6) will be a linear scaling
operation. Note that the density matrix based density update Eq. (6) will be superior to the orbital based Eq. (5) only when the number of occupied orbitals becomes larger than the number of basis functions that are locally non-zero. Due to localization there will be such a turning point as the system size grows. This is also visible in Fig. 1, which shows the cross-over point (dashed and solid blue lines) around 300 atoms.

3 Grid operations

To illustrate the role that is played by the grids in NAO based calculations, we return to the example of computing the Hamilton matrix. In practice, the integration grid used for evaluating an approximation to \( h_{st} \) is composed of overlapping atom-centered grids, where each radial shell of points is a Lebedev grid [17–19]. The radial positions of the shells on the axis ranging from zero to infinity are taken to be

\[
s = -\log \left( 1 - \left( \frac{s}{n_{\text{radial}}+1} \right)^2 \right), \quad s = 1, \ldots, n_{\text{radial}} [20].
\]

For other options and extensive tests on the grids in conjunction of Gaussian basis functions we refer to Ref. [21]. The overlapping grids are then partitioned using a \textit{partitioning of unity} method [5,22], i.e. each atom-centered grid (centered at a site \( \alpha \)) is associated with a weight function \( p_\alpha(r) \), \( \alpha = 1, \ldots, N_{\text{grids}} \) such that \( \sum_\alpha p_\alpha(r) = 1 \) for every point \( r \in \mathbb{R}^3 \). On this grid the exact value of \( h_{ij} \) is approximated using a quadrature over the discrete integration points \( \{r\} \), i.e. we set \( h_{ij} = \sum_r w(r)f(r) \) where \( w(r) \) is the combined weight of radial, angular and partition weights at \( r \), \( f(r) = (\hat{\phi}_i \hat{\phi}_j)(r) \) is the integrand and the summation runs over all grid points.

In practice the quadrature is not done one grid point at a time but larger batches of points, \( B_\nu \), are used [23]. This allows the use of matrix-matrix products for computing the matrix elements \( h_{ij} \). The full algorithm (closely related to the ones in [15,16,24,23]) is presented in detail in Algorithm 1. Three observations are immediate.

\begin{algorithm}
\caption{Integration of the Hamilton matrix with given grid batches \( B_\nu \)}
\begin{enumerate}
    \item For each batch of points \( B_\nu \):
        \begin{enumerate}
            \item Find the non-zero basis functions in the batch. Denote the number of these by \( \text{nnz}(B_\nu) \).
            \item For each \( r \in B_\nu \) and each non-zero basis function \( \phi_i \), \( i = 1, \ldots, \text{nnz}(B_\nu) \) evaluate the matrices \( K_{i,r} = \phi_i(r) \) and \( L_{j,r} = (\hat{\phi}_j)(r) \).
            \item Compute the part of the submatrix of the Hamiltonian matrix that corresponds to \( B_\nu \), \( h_{ij}[B_\nu] \), with a matrix-matrix product over the points in \( B_\nu \), i.e. \( h_{ij}[B_\nu] = \sum_r K_{i,r} L_{j,r} \).
        \end{enumerate}
    \item After the loop over \( B_\nu \)'s is finished sum up the results: \( h_{ij} = \sum_\nu h_{ij}[B_\nu] \).
\end{enumerate}
\end{algorithm}

In this point. First, for Step 1a to yield as small number of non-zero basis functions as possible it is important to have grid batches that are as localized as possible. Second, Step 1a scales technically \( O(N^2) \) but has an extremely small prefactor. In
fact, the indexes of the non-zero basis functions for each batch can be computed at
the initialization stage and stored for the rest of the computation if desired, remov-
ing the $O(N^2)$ step. Third, the loop over the batches is an embarrassingly parallel
operation. In fact, the only communication operation is a global reduction that takes
place when summing up the results in Step 2.

4 Grid partitions

4.1 General form of the problem

Let us start with the formal definition of the general problem of finding the grid
batches that are as localized as possible for the grid based operations as given in
Problem 1:

**Problem 1** Given a (finite and non-empty) set of points $P \subset \mathbb{R}^3$ find the batches
\( \{B_\nu\}, \nu = 1, \ldots, N_{\text{batches}} \) such that

1. $\bigcup_\nu B_\nu = P$
2. $B_\nu \cap B_\mu = \emptyset$ for $\nu \neq \mu$
3. $c_0 \leq \#B_\nu \leq c_1$ for some $c_\xi$, $\#P \geq c_\xi \geq 1$. Here $\#S$ denotes the number of points
in the set $S$.
4. The quantity
   (a) $\text{avg}_\nu \{\#B_\nu\}$ or
   (b) $\text{max}_\nu \{\#B_\nu\}$
   is minimized.

The target 4a aims for optimal performance whereas the target 4b is geared towards
minimal workspace memory consumption.

It should be noted that the general form of Problem 1 is not solvable in practice
due to high complexity, and therefore it is necessary to employ heuristic methods
instead of solving the optimization problem directly. However, in the special case
when $c_0 = 1$, the optimal solution can be found and it is given by batches containing
only single points. We use this case as a reference for our algorithms and denote
$B_{\nu_{\text{ref}}} = \{r | r = r(\nu)\}$, the set containing the $\nu$th integration point.

We note that the problem setting is closely related to partitioning methods needed
in adaptive multilevel methods for solving parallel differential equations (see e.g.
Ref. [25] and references therein) except that in our case the grid is not evolving
and the global localization of the partitions is not an issue. Another relation can
be found to the integration methods discussed in Ref. [26]. However, our aim is
not to partition the space into regions where different cubatures can be applied but
focus on the localization of the integration batches for a given grid and cubature. Finally, the problem of finding good grid partitions is a generic one for many codes similar to ours. Nonetheless, while the use of grid partitions is mentioned in several works (e.g., Refs. [14–16,27], in the slightly different context of load balancing in Ref. [28]), to our knowledge only one work (Ref. [15]) discusses the choice of their shape to minimize the computational work in a comparative way (in that case, a hybrid of the radial-shell and octree methods below).

4.2 Three top-down methods to partition the grid

In this section we present three top-down methods to partition the atom centered real-space grids. The methods are top-down in the sense that they generate the batches without using information on the structure of the grid within the generated batch. The resulting algorithms are very efficient and the batching can be done with minimal cost. On the other hand, the local distribution of the points is not fully accounted for and, e.g., local variations in the density of points are not considered. We will return to this question in Sect. 5.3 for our discussion about alternative bottom-up methods for generating the grid batches.

4.2.1 Radial shells and their partitions

The first method is based on the geometric properties of the overlapping grids and is the most straightforward of the methods we consider. In this case, the batches are simply taken to be the Lebedev grids making up the radial shells. Furthermore, they can be refined by considering halves, quadrants and octants of the shells, leading to

\[ B_\nu = \{ \text{angular grid / subset of the grid for one radial shell} \} \]

for \( \nu = 1, \ldots, N_{\text{batches}} \).

4.2.2 Octree partitions of the grid

The second method does away with the relation between the grids and the atoms and considers the grid only as a set of points in \( \mathbb{R}^3 \). Then the grid is recursively partitioned in eight sub-grids by splitting the set of points with planes parallel to the coordinate axes. The details are given in Algorithm 2 that uses a depth-first method to build the octree defining the grid batches \( B_\nu \). In the step 4 several conditions can of course be employed to accept \( S_\mu \) as a new batch. We have used two conditions that must be satisfied simultaneously: (1) the value representing a weighted size of the batch,

\[ \#S_\mu \times \frac{\max_{r \in S_\mu} \{ \text{nnz}(\{r\}) \}}{\max_{r \in P} \{ \text{nnz}(\{r\}) \}} \]
Algorithm 2 Octree method for grid partitioning

(1) Initialize the active set $S$ to contain all the points $S = P$
(2) Compute the center of mass for $S$
(3) Split $S$ into eight subsets $S_1, \ldots, S_8$ using cut-planes parallel to coordinate axes and going through the center of mass.
(4) For each subset $S_j$ check if $S_j$ is acceptable:
   (a) If yes, make $S_\mu$ into a batch $B_\nu = S_\mu$.
   (b) If no, go to 2 with $S_\mu$ as the new active set $S$.

must be less than a given bound, $C_S$, and (2) the absolute size of the batch, $\#S_\mu$, must be less than a given (different) bound $C_H$. We note that the octree method has a close relation to spatial division methods used for a long time in computational geometry, e.g., for constructing a mesh for finite-element calculations [29,30].

4.2.3 Grid adapted cut-plane method

The obvious drawback (from the algorithmic point of view) of the octree method is that the coordinate axes are given a special role as the planes determining the three planes to cut the set $S$. Also tying the local origin to the center of mass of $S$ does not necessarily result in even-sized subsets $S_\mu$. Both shortcomings are relatively easy to overcome by (1) using only a single plane to cut $S$ but adapting the orientation of the plane to $S$ and (2) adjusting the location of the plane so that the resulting partitions are even-sized. The details are given in Algorithm 3. In Step 6 we use the same criteria as in the case of the octree method. We note that the adapted cut-plane method is a variation of a method presented in the lecture notes by W. Kahan [31].
Table 1

Basis and grid settings for all elements as used in the calculations. The basis sizes given are for ionic and hydrogen-like functions in addition to the minimal basis of occupied atomic orbitals.

<table>
<thead>
<tr>
<th></th>
<th>H</th>
<th>C</th>
<th>N</th>
<th>O</th>
<th>Cu</th>
<th>Au</th>
</tr>
</thead>
<tbody>
<tr>
<td>basis min+</td>
<td>3s2p</td>
<td>2s2p2df</td>
<td>2s2p2df</td>
<td>2s2p2df</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$r_{cutoff}$ (Å)</td>
<td>5.5</td>
<td>5.5</td>
<td>5.5</td>
<td>5.5</td>
<td>6.0</td>
<td>6.0</td>
</tr>
<tr>
<td>min angular grid</td>
<td>110</td>
<td>110</td>
<td>110</td>
<td>110</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>max angular grid</td>
<td>590</td>
<td>590</td>
<td>590</td>
<td>590</td>
<td>590</td>
<td>590</td>
</tr>
</tbody>
</table>

| $n_{radial}$ | 49 | 69 | 71 | 73 | 107 | 147 |

5 Results

5.1 Top-down methods: timings and batch size

To test the different grid partitioning methods, we consider three different physical systems: fully extended polyalanine chains, cluster cutouts of fcc-Cu, and Au(100)-hex surfaces. The basis sizes and grid parameters used in our calculations are shown in Table 1 and they correspond to the values given in Ref. [8].

We show the changes in timings for one self-consistency cycle, the weighted average number of non-zero basis functions over batches, $\langle \text{nnz} \rangle$:

$$\langle \text{nnz} \rangle = \frac{\sum_i \#B_v \cdot \text{nnz}(B_i)}{\sum_i \#B_v}$$

as well as the maximal number of non-zero basis functions over all the batches: $\text{nnz}_{max} = \max \{ \text{nnz}(B_v) \}$. In addition, we show the average size of the batches, $B_v$ and their standard deviation, $\sigma$. With octree and adapted methods we have used the values $C_S = 200$ for the desired weighted size of a batch and $C_H = 400$ for the maximal size of a batch as the parameters controlling the termination of the algorithm.

The results for the polyalanine chains are displayed in Figs. 3, 4, and 5. It is clearly visible that the adapted cut-plane method yields a partitioning that (1) gives smallest average and maximal number of non-zero basis functions, (2) allows for largest average batch size and consequently (3) gives the best performance. The actual savings for a single self-consistency cycle for the largest molecule are 14% and 24% when compared to the radial shells and octree methods, respectively.

The results for the octree method are notably bad for the longest polyalanine chains. This is a result of the “long and thin” nature of the molecules. Since the octree method recursively splits each batch into eight sub-batches the shape of the origi-
Fig. 3. Timings in seconds for one self-consistency cycle of different grid partitioning methods for the fully extended polyalanine structures, full lines = entire cycle, dashed lines = integration of the Hamiltonian matrix.

For cluster cutouts of fcc Cu, we obtain results with a similar trend except that in this case the radial shells method is clearly the worst choice whereas the octree method is closer to the adapted method as shown in Fig. 7. Actual savings are 33% and 11% for the largest cluster. The difference in favor of the adapted method over the octree method is explained in Figs. 8 and 9. In Fig. 8 it can be seen that the maximal and average number of non-zero basis functions is almost equal for both methods but the adapted method obtains this result with considerably larger batches as shown in Fig. 9. It follows that the adapted method is able to perform the matrix multiplication in Step 1c of Algorithm 1 with a larger matrix size, giving better overall performance. In addition, when working with packed matrix storage for the global Hamiltonian $h_{ij}$, fewer batches mean that less time is consumed for sorting the locally non-zero matrices $h_{ij}[B]_n$ into the global structure (which must be done once per batch). It should also be noted from Fig. 8 that the number of
Fig. 4. Maximal and average number of non-zero basis functions per batch, nnz, for the fully extended polyalanine structures.

Fig. 5. Average size of the batches and their standard deviation for the fully extended polyalanine structures.
Fig. 6. Polyalanine chain of thirty residues (308 atoms) and the worst batch produced by the octree method (red).

Fig. 7. Timings in seconds for one self-consistency iteration of different grid partitioning methods for the fcc Cu cutouts, full lines = entire cycle, dashed lines = integration of the Hamilton matrix.

non-zero basis functions has not yet saturated, contrary to the case of the fully extended polyalanines. Consequently, the method has not yet reached the (near) linear scaling regime for these systems.

Our final example is a periodic system: a reconstructed Au(100)-(5 × m)-hex surface slab with three layers and an increasing number m of lateral rows. In periodic systems, we need to map the grid from the periodic images to the “zeroth” supercell. This effectively breaks the original shell structure and leads to a complete failure of the radial shells method. This is clearly visible in Fig. 10 where the integration
Fig. 8. Maximal and average number of non-zero basis functions per batch, nnz, for the cluster cutouts of fcc Cu. The graphs for the octree and adapted methods practically overlap.

Fig. 9. Average size of the integration batches and their standard deviation for the cluster cutouts of fcc Cu.
alone with the radial shells method takes more time as the entire cycle using the octree or the adapted methods. The adapted method is again the winner here saving 72% and 10% in calculation time for the largest surface when compared to radial shells and octree methods, respectively.

The failure mode of the radial shells method becomes evident when we inspect Figs. 11 and 12. The extremely high number of the non-zero basis functions clearly show that the radial shell based partitioning is not the correct approach for periodic systems. On the other hand, the two other methods show a clear saturation of the number of non-zero basis functions, indicating good localization of the batches already from the start. Since the integration grid is distributed rather uniformly over the supercell, the differences between octree and adapted methods are relatively small. However, again the adapted method is able to produce the same localization effect using considerably larger batches.

The general trend in all cases is thus similar: The adapted cut-plane method performs best in all respects and the octree method and the radial shells with subdivisions are worse. A remarkable failure of the method based on radial shells can be seen in the case of the periodic surface, where the maximal number of non-zero basis functions skyrockets. This is due to the fact that, in periodic systems, the grid operations are all performed on the supercell and thus the ’natural’ ordering of the radial shells is destroyed.

5.2 On the optimality of the grid partitions

Due to the complexity of the general problem of creating the grid batches we cannot determine accurately how close our heuristic methods are to the optimal solution. However, we can get some idea on the quality of our results by comparing to the case where the batches are taken to be only single points, i.e. when $c_0 = c_1 = 1$. In this case we can compare the actual number of the evaluations of the basis functions to the number of evaluations required in the case of single-point batches. To this end, recall that $B_{v}^{ref} = \{ r(\mu) \}$ and define the effectivity of the batches, $e$, as

$$e = \frac{\sum_v \text{nnz}(B_v) \times #B_v}{\sum_v \text{nnz}(B_{v}^{ref})}.$$  \hspace{1cm} (7)

The effectivities for all three different batching schemes for all three test cases are reported in Tables 2 – 4. As expected from the results above, the adapted method provides best overall effectivity and the results for the radial shells method are worst by a large factor. When comparing the values for $e$ it should be taken into account that the average batch size is largest for the adapted method and consequently effectivity close to one is even harder to achieve.

The last two rows of Table 4 show the improvement in $e$ in the case of the recon-
Fig. 10. Timings in seconds for one self-consistency iteration of different grid partitioning methods for the Au(100)-(5 x m)-hex surface (3 layers), full lines = entire cycle, dashed lines = integration of the Hamiltonian matrix. Note the logarithmic scale of the plot.

### Table 2
Effectivity e of different batching schemes for the fully extended polyalanines.

<table>
<thead>
<tr>
<th>Number of atoms</th>
<th>18</th>
<th>58</th>
<th>108</th>
<th>158</th>
<th>208</th>
<th>308</th>
</tr>
</thead>
<tbody>
<tr>
<td>radial shells</td>
<td>1.17</td>
<td>1.27</td>
<td>1.29</td>
<td>1.29</td>
<td>1.29</td>
<td>1.29</td>
</tr>
<tr>
<td>octree</td>
<td>1.08</td>
<td>1.12</td>
<td>1.16</td>
<td>1.21</td>
<td>1.23</td>
<td>1.26</td>
</tr>
<tr>
<td>adapted</td>
<td>1.08</td>
<td>1.11</td>
<td>1.11</td>
<td>1.11</td>
<td>1.11</td>
<td>1.12</td>
</tr>
</tbody>
</table>

The constructed Au-surface when the parameters controlling the batch sizes are halved and divided by four, i.e., when $C_S = 100$, $C_H = 200$ and $C_S = 50$, $C_H = 100$, respectively. This aids in improving the effectivity e even further, albeit at the price of a larger overall number of batches (increasing the overhead from per-batch operations such as sorting matrix elements into the global $h_{ij}$), and reducing the size of the performed matrix products.

#### 5.3 How about bottom-up methods?

All the methods presented above are so called top-down methods, i.e. they start from a given set of points and recursively divide it into smaller chunks until a desired batch size is reached. The inherent drawback of top-down methods is that
Fig. 11. Maximal and average number of non-zero basis functions per batch, nnz, for the Au(100)-(5 x m)-hex surface. The graphs for the octree and adapted methods practically overlap.

Table 3
Effectivity $e$ of different batching schemes for the fcc Cu-clusters.

<table>
<thead>
<tr>
<th>Number of atoms</th>
<th>13</th>
<th>19</th>
<th>43</th>
<th>55</th>
<th>79</th>
<th>87</th>
</tr>
</thead>
<tbody>
<tr>
<td>radial shells</td>
<td>1.22</td>
<td>1.31</td>
<td>1.47</td>
<td>1.51</td>
<td>1.55</td>
<td>1.56</td>
</tr>
<tr>
<td>octree</td>
<td>1.10</td>
<td>1.13</td>
<td>1.13</td>
<td>1.13</td>
<td>1.12</td>
<td>1.12</td>
</tr>
<tr>
<td>adapted</td>
<td>1.11</td>
<td>1.12</td>
<td>1.16</td>
<td>1.17</td>
<td>1.17</td>
<td>1.17</td>
</tr>
</tbody>
</table>

Table 4
Effectivity $e$ of different batching schemes for the Au(100) - (m x 5) -surfaces.

<table>
<thead>
<tr>
<th>Number of Au rows</th>
<th>1</th>
<th>3</th>
<th>5</th>
<th>7</th>
<th>10</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>radial shells</td>
<td>2.14</td>
<td>2.57</td>
<td>2.67</td>
<td>2.64</td>
<td>2.56</td>
<td>2.52</td>
</tr>
<tr>
<td>octree</td>
<td>1.17</td>
<td>1.11</td>
<td>1.13</td>
<td>1.15</td>
<td>1.17</td>
<td>1.15</td>
</tr>
<tr>
<td>adapted ($C_S = 200$, $C_H = 400$)</td>
<td>1.17</td>
<td>1.16</td>
<td>1.17</td>
<td>1.16</td>
<td>1.17</td>
<td>1.16</td>
</tr>
<tr>
<td>adapted ($C_S = 100$, $C_H = 200$)</td>
<td>1.12</td>
<td>1.12</td>
<td>1.13</td>
<td>1.12</td>
<td>1.13</td>
<td>1.12</td>
</tr>
<tr>
<td>adapted ($C_S = 50$, $C_H = 100$)</td>
<td>1.09</td>
<td>1.09</td>
<td>1.09</td>
<td>1.09</td>
<td>1.09</td>
<td>1.09</td>
</tr>
</tbody>
</table>
the local features of the distribution of the grid points in the three-dimensional space are not accounted for in detail. This observation gives rise to another set of approaches, so called bottom-up methods, where the local environment of each of the grid points is analyzed before creating the grid batches.

We have implemented and tested two bottom-up methods: First, a method where a Delaunay mesh with the grid points as nodes is created and the mesh is partitioned with a multilevel graph partitioning method. This is all realized using external established tools [32,33]. Second, a method where the batches are built by grouping nearby points together and then recursively merging the groups until a desired batch size is reached. We denote the number of merged items per level a grouping factor, \( g_f \).

The bottom-up methods are able to produce a set of batches that is similar to the ones produced the grid adapted method as can be seen from Tables 5 – 6. However, they require more resources. The first method, a combined Delaunay-mesh and graph partitioning approach, uses a large amount of memory to store the mesh. The second method, the grouping algorithm, needs a lot of searches to find the nearest neighbors of the groups at each level. These searches take a lot of computation time to complete. Even for small test systems the grid partitioning using grouping algorithm becomes by far the most time consuming part of the electronic structure calculation rendering the approach useless in practice.
Table 5
Performance of the bottom-up methods for a single polyalanine residue (18 atoms).

<table>
<thead>
<tr>
<th></th>
<th>graph part.</th>
<th>groups, $gf = 2$</th>
<th>groups, $gf = 4$</th>
<th>groups, $gf = 8$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\langle \text{nnz} \rangle$</td>
<td>257.97</td>
<td>265.37</td>
<td>266.01</td>
<td>273.36</td>
</tr>
<tr>
<td>effectivity</td>
<td>1.08</td>
<td>1.11</td>
<td>1.11</td>
<td>1.14</td>
</tr>
<tr>
<td>average batch size</td>
<td>200.00</td>
<td>255.90</td>
<td>255.56</td>
<td>508.81</td>
</tr>
<tr>
<td>std. dev. of batch size</td>
<td>4.93</td>
<td>3.37</td>
<td>8.65</td>
<td>36.25</td>
</tr>
</tbody>
</table>

Table 6
Performance of the bottom-up methods for a Au(100) - (1x5) surface.

<table>
<thead>
<tr>
<th></th>
<th>graph part.</th>
<th>groups, $gf = 2$</th>
<th>groups, $gf = 4$</th>
<th>groups, $gf = 8$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\langle \text{nnz} \rangle$</td>
<td>796.73</td>
<td>834.79</td>
<td>845.39</td>
<td>902.90</td>
</tr>
<tr>
<td>effectivity</td>
<td>1.17</td>
<td>1.22</td>
<td>1.24</td>
<td>1.32</td>
</tr>
<tr>
<td>average batch size</td>
<td>199.98</td>
<td>255.99</td>
<td>255.89</td>
<td>509.54</td>
</tr>
<tr>
<td>std. dev. of batch size</td>
<td>4.65</td>
<td>0.61</td>
<td>3.91</td>
<td>32.13</td>
</tr>
</tbody>
</table>

On the other hand, the actual graph partitioning method is fast and it can accept also other graphs than Delaunay meshes as input. The bottom-up methods can thus be developed further by building a graph by connecting nearby points and then splitting the graph. In this case, it is important to include the local distribution of the grid points by using a graph whose nodes can have a varying index.

6 Conclusions

The results and theoretical considerations above show how grid partitioning combined with localization of the basis functions leads to linearly scaling grid based operations, i.e., the integration and the electron density update, in EST calculations using NAOs as the basis set. The effect of the grid partitions is most pronounced for periodic systems, but also the performance for non-periodic cases is notably improved when the grid is properly divided into batches.

The fact that localization entails the performance of NAOs is not a surprise. However, the complexity of the actual problem of finding an optimal grid partitions is too high to be tackled in full, and heuristic methods must be employed. It is somewhat more surprising that the methods implemented and tested here exhibit such a big difference in their performance. The best method we have obtained, the adapted cut-plane method, is rather close to the theoretical optimum for our test systems, indicating a good level of heuristic approach. The octree method suffers from the tendency to generate batches with very few points leading to inefficiency and has the drawback of unnecessarily replicating the geometry of the system. Finally, peri-
odic systems present a more complicated environment due to the complex mapping of the periodic images of the gridpoints. This problem manifests itself most strikingly in the failure of the radial shell method.

In this work we have focused mainly on the top-down methods. The other approach, bottom-up methods, suffers from the fact that it is problematic to generate a graph over the grid that accurately describes the local environment of the grid points. In addition, however, the much simpler top-down adaptive grid method used above performs at nearly the same or better effectivity than the formally more rigorous bottom-up methods attempted here. Apparently, this fast, well-tuned top-down approach captures all practically needed aspects of the grid partitioning problem, leaving no incentive to pursue any more complicated schemes.

Acknowledgments

Ville Havu acknowledges the use of the ”Louhi” supercomputer of the Finnish IT center for sciences, CSC.

References


  URL http://www.cs.cornell.edu/home/vavasis/qmg-home.html

[31] W. Kahan, Separating clouds by a plane, lecture notes, CS Division, UC Berkeley.  

    Trans. on Mathematical Software 22 469 – 483.  
    URL http://www.qhull.org

[33] G. Karypis, V. Kumar, METIS A Software Package for Partitioning Unstructured  
    Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse  
    Matrices, Version 4.0.  
    URL http://glaros.dtc.umn.edu/gkhome/views/metis