Solvers for $O(N)$ Electronic Structure in the Strong Scaling Limit

Nicolas Boc$^{[c]}$ and Matt Challacombe

Theoretical Division,
Los Alamos National Laboratory,
Los Alamos, NM 87544

Laxmi Kant V. Kalé
Parallel Programming Laboratory,
Department of Computer Science,
University of Illinois at Urbana-Champaign

(Dated: May 18, 2014)

We present a hybrid OpenMP/Charm++ framework for solving the $O(N)$ Self-Consistent-Field eigenvalue problem with parallelism in the strong scaling regime, $P \gg N$. This result is achieved with a nested approach to Spectral Projection and the Sparse Approximate Matrix Multiply [Bock and Challacombe, SIAM J. Sci. Comput. 35 C72, 2013], which involves an $N$-Body approach to occlusion and culling of negligible products in the case of matrices with decay. Employing classic technologies associated with the $N$-Body programming model, including over-decomposition, recursive task parallelism, orderings that preserve locality and persistence-based load balancing, we obtain scaling better than $P \sim 500N$ for small water clusters ($([H_2]_N, N = 30, 90, 150)$ and find support for an increasingly strong scalability with increasing system size, $N$.

Keywords: Sparse Approximate Matrix Multiply; Sparse Linear Algebra; SpAMM; Reduced Complexity Algorithm; Linear Scaling; Quantum Chemistry; Spectral Projection; $N$-Body; Charm++; Matrices with Decay; Parallel Irregular; Space Filling Curve; Persistence Load Balancing; Over-decomposition

I. INTRODUCTION

Ab initio electronic structure methods for the Self-Consistent-Field (SCF) problem, involving pure density functional theory (DFT) or hybrid functionals that also include the Fock exchange offer predictive power at low cost and consequently find broad utility in chemistry, biology, materials science and drug design. With conventional methods, solving the SCF eigenvalue problem is rate limiting due its steep $O(N^3)$ scaling which in practice restricts problems to systems with $\sim 1,000$ atoms even on large computers. Recently, alternative methods which are $O(N)$ (linear scaling) have been developed that exploit the local, quantum nature of non-metallic electronic interactions. Early approaches to linear scaling solution of the SCF eigenproblem sought to exploit this quantum locality by avoiding the pair-wise support of local basis functions beyond a cutoff radius, leading to matrix sparsity and an $O(N)$ computational effort through iterative algorithms based on the sparse matrix-matrix multiply (SpMM). Later, more flexible methods based on “filtering” and “sparsification” were developed, relying on iterative truncation of the vector space by element or block magnitudes. Currently, linear scaling methods can access systems involving $\sim 10,000$ atoms for an excellent review and current state of the art see Bowler et. al

The first parallel SpMM was reported by one of us recently, two-dimensional decompositions have lead to improved efficiencies, with Bowler et. al reporting a scalable SpMM up to 196,000 cores and parallel calculations mastering systems sizes up to $\sim 1,000,000$ atoms. Also, generic methods for the SpMM have been developed by Buluç et. al where matrix row and columns are randomly permuted to achieve an even load distribution, yielding high efficiencies. This approach has been adopted for quantum chemistry with a slightly modified Cannon algorithm, radial cutoffs, and static load balancing based on a fixed graph. These parallel approaches to the $O(N)$ SCF eigenvalue problem, based on one- or two-dimensional strategies for matrix decomposition, have established scalability in the weak regime, $P/N \approx$ constant (see for example Fig. 1 of Ref.). However, the full potential of $O(N)$ methods may lie in their ability to access the strong scaling limit, $P \gg N$, particularly for the increasingly large, asynchronous, and heterogeneous next generation of high performance computing systems with $P > 1,000,000$ cores. In this work, we consider access to the strong scaling regime with three-dimensional over-decomposition strategies and heuristic schemes that parlay quantum locality into spatial and temporal data locality. This approach is orthogonal to randomization strategies, which forgo any underlying locality and are in any case throttled to $O(\sqrt{P})$ in the strong scaling limit due to limitations of the Cannon or SUMMA algorithm.

Beyond the SCF eigenproblem, there are typically four additional “fast” solvers that must interoperate with each other, representing a tightly coupled collective of advanced numerical methods. Typically, these solvers are
developed and optimized independently, involving differing data structures and programming models (e.g. 3-D FFT, CSR based SpMM, etc.). In the strong scaling regime, such a piecemeal collective may: (a) disrupt data locality with redistributions and transformations, (b) significantly raise the barrier to entry and innovation, (c) exceed the ability of advanced runtime systems to load balance multiple programming models, (d) lead to divergent rates of error accumulation, and (e) impede deployment for trends such as fine grained check-pointing fault-tolerance, energy aware load balancing and job malleability.

We have recast all five solvers at the hybrid HF/DFT level of SCF theory within the N-Body programming model, including (1) Fock Exchange, (2) Spectral Projection (this work), (3) inverse factorization, (4) Coulomb summation, and (5) the Exchange Correlation problem. These developments offer a potential solution to challenges (a)-(e), through a unified programming model with a proven record of performance. In this contribution, we develop strategies for recursive over-decomposition and persistence-based load balancing of the SpMM solved as employed by Spectral Projection, an $O(N)$ alternative to the SCF eigenvalue problem for matrices with decay. This paper is organized as follows: In Sec. II we describe in detail the SpAMM algorithm and in Sec. III its parallel implementations within OpenMP and the Charm++ runtime. In Sec. IV we detail our methodology and show parallel scaling results for quantum mechanical matrices with decay and demonstrate scalable high-performance in the strong scaling limit. Finally, we discuss our results in Sec. V.

II. THE SPARSE APPROXIMATE MATRIX MULTIPLY

A wide class of problems exist that involve matrices with decay, often corresponding to matrix functions notably the matrix inverse or the matrix exponential and in the case of electronic structure theory, the Heaviside step function (spectral projector). A matrix $A$ is said to decay when its matrix elements decrease exponentially, as $|a_{ij}| < c \lambda^{|i-j|}$, or algebraically as $|a_{ij}| < c/(|i-j|^{\lambda}+1)$ with separation $|i-j|$. In simple cases, the separation $|i-j|$ may correspond to an underlying physical distance $|\vec{r}_i - \vec{r}_j|$, e.g. of basis functions, finite elements, etc., leading often to a strong diagonal dominance when ordered carefully. For simple decay, truncation in the two-dimensional vector space, e.g. via radial cutoff $a_{ij} = 0$ if $|\vec{r}_i - \vec{r}_j| > r_{cut}$, together with the use of a conventional SpMM algorithm yields a reduced complexity kernel for the iterative construction of matrix functions.

However, truncation may not be the most efficient or accurate approach to exploiting decay, which can be oscillatory, involve quantum beats, or even long range charge transfer as in the case of excited states, see Ref. and references therein. These effects are shown in Fig. which is a density matrix for a large water cluster with the underlying basis reordered to preserve locality; note the large cross-diagonal beats, as well as the strong clustering and segregation of elements with like magnitude. For this type of structured matrix with non-trivial decay, the quadrants pioneered in linear algebra by Wise et. al provides a powerful framework for recursive data base operations such as the metric-query involving the lookup of sub-blocks by magnitude, $\|A_{ij}\|$. Based on this framework, the SpAMM algorithm exploits decay recursively in the three-dimensional convolution space, with adaptive culling and occlusion of insignificant products at each tier $t$, determined by metric-query of the sub-multiplicative Frobenius norm, $\|\cdot\|$, and a numerical threshold $\tau$ controlling precision.

While the discussion has so far involved dense matrices, note that SpAMM is applicable to sparse matrices as well; even with dense matrices, large values of $\tau$ correspond to an implicit truncation and potentially a sparse product. With extreme sparsity, either implicit or explicit, the corresponding irregularity and small work/data ratios challenge scalability. Relative to conventional row-column approaches to the SpMM, the SpAMM algorithm applied to structured matrices with decay may achieve: (i) additional flexibility in the three-dimensional space-time for domain decomposition and load balancing (this work), (ii) the recursive accumulation of terms with like magnitude and an $O(N \log N)$ error accumulation, (iii) occlusions that occur early in recursion, and (iv) a more efficient use of high level memory chunking for message passing and low level blocking strategies for acceleration, and (v) additional flexibility for achieving error control within a culled volume.

III. TASK OVER-DECOMPOSITION

One of the strengths of the N-Body programming model is that there are many ways to realize over-decomposition on a range of hardware, e.g. from long pipe GRAPE (SIMD) accelerators to conventional SMP (MIMD) architecture. Ideally, an architecture independent runtime system would seamlessly enable the recursive generation of lightweight tasks, as OpenMP 3.0 does for SMP. However, while this feature is a target of the Dynamic Parallelism framework of NVIDIA’s CUDA and other runtimes, full support for recursive task parallelism is as yet unrealized.
for distributed memory systems; in this work, we consider simple methods for achieving recursive task parallelism with SpAMM for the ubiquitous “cluster of SMP nodes” architecture\cite{29,20} using the mixed OpenMP/Charm++ runtimes.

There are two main considerations in our scheme that involve memory and task management: First, the native use of task parallelism at the SMP level has the potential to involve non-contiguous memory and high packing/unpacking overheads when redistributing memory between nodes. Thus, it is convenient to chunk contiguous memory used at the SMP level. In this work, memory is allocated to hold a full sub-octree of depth 5, together with a 4 × 4 blocking at the lowest level as outlined in Ref\cite{20}, corresponding to a 128 × 128 sub-block. Second, an explicitly allocated, unrolled octree is a necessary structure that enables Charm++ to manage tasks involving occlusion and culling as well as node-level SMP work. In the following, we develop the SpAMM kernel in the context of Spectral Projection, using a combined OpenMP/Charm++ framework and a unified N-Body outlook.

### A. OpenMP

Shown in Alg. 1 is the SMP parallel implementation within the OpenMP programming model; SpAMM_omp recursively walks a transient octree generated dynamically on the stack through the OpenMP 3.0 tasking feature\cite{50}. Guided by the binary convolution of matrix quadtrees A^t and B^t at each tier t (3), the implicit octree traversal may be sparse and irregular due to culling and occlusion (4). The parallel tree traversal is extended through untied OpenMP tasks (5) and recursive calls to SpAMM_omp (6). Per node synchronization (to ensure appropriate variable lifetimes) is achieved through the OpenMP taskwait statement (9). Finally, at the leaf tier, dense matrix products are performed and the result reduced into the C quadtree (12), with a data race on C prevented through explicit use of OpenMP locks (11, 13).

In this work we have made little effort to optimize the SpAMM_omp implementation, or even the dense contraction on line 12. In Ref\cite{50} we showed that accuracies better than the native GEMM are possible also with N-scaling, but that difficult, platform specific optimizations were necessary; the corresponding OpenMP algorithm is likely to be equally complex and involve platform dependent vector (SSE) optimizations. It is perhaps wise to avoid these efforts until the OpenMP 4.0 SIMD constructs\cite{50} are realized.

**Algorithm 1** The OpenMP SpAMM algorithm, recursively multiplying matrices C ← A × B under a SpAMM tolerance τ. The function matrix arguments are pointers to tree nodes.

```
1: function SpAMM_omp(τ, t, A^t, B^t, C^t)
2: if t < depth then
3:   for all {i,j,k | C^t_{ij} ← A^t_{ik}B^t_{kj}} do
4:     if ||A^t_{ik}|| ||B^t_{kj}|| > τ then ▶ Culling
5:       OpenMP task untied
6:       SpAMM_omp(τ, t+1, A^{t+1}_{ik}, B^{t+1}_{kj}, C^{t+1}_{ij})
7:   end if
8: end for
9: OpenMP taskwait
10: else
11:   OMP_SET_LOCK ▶ Acquire OpenMP lock on C
12:   C ← C + A × B ▶ Dense product
13:   OMP_UNSET_LOCK ▶ Release OpenMP lock on C
14: end if
15: end function
```

### B. Charm++

Charm++\cite{20} is a mature runtime environment on distributed memory platforms available for all major supercomputer systems, allowing for efficient scalable high performance implementations\cite{20,20,20}. In the message-driven execution model of Charm++, code and data are encapsulated in C++ objects called “chares” which are initially placed by static load balancing algorithms. Dynamic persistence-based load balancing strategies migrate chares transparently during solver execution based on load and communication measurements from previous solver iterations and efficiently optimize load distri-

---

FIG. 1: The decay of matrix element magnitudes of a converged spectral projector (density matrix) for a (H\textsubscript{2}O\textsubscript{300} water cluster at the RHF/6-31G** level of theory (n = 7500), where the molecular geometry has been reordered with a space filling Hilbert curve. The different colors indicate different matrix element magnitudes; red: [0, 10^{-8}); green: [10^{-8}, 10^{-6}); blue: [10^{-6}, 10^{-2}); violet: [10^{-2}, 1], corresponding to approximate exponential decay.
bution and communication cost. The Charm++ runtime transparently manages chare placement and migration and proxy objects are used to send messages to particular chare instances or groups thereof without explicit specification of their location. Chares can be grouped in multi-dimensional sparse arrays or used as “singleton” objects.

Persistence-based load balancing exploits temporal localities in iterative solvers through decomposition of the load and communication graph. Since the dynamic load balancing strategies of Charm++ only consider chares organized in arrays that are still instantiated at the time of the load balancing step, the multiplication octree has to be explicitly stored in memory (as opposed to the transient stack based “storage” used in the SMP implementation). The matrix quadtree starting from the root down to the 128 × 128 chunk is stored in two-dimensional chare arrays which cover the full matrix on a tier. The corresponding unrolled octree is stored in three-dimensional chare arrays with occlusion and culling carried out iteratively, tier-by-tier, until the chunk level at which SpAMM_omp is invoked.

Data and work locality is exploited through communication aware load balancing strategies in Charm++. However, a bug in the Charm++ runtime currently prevents the use of sparse load balanced chare arrays introducing a \(O(N^3)\) communication component with a small prefactor that is found to be negligible.

**Algorithm 2** The SpAMM algorithm in the Charm++ programming model. Tree occlusion is done by iterating over the three-dimensional multiplication chare arrays, `CONVOLUTION[d]`. In Charm++ a call such as `CONVOLUTION[t] . occlude` translates into a broadcast to all array elements of `CONVOLUTION[t]`.

1: function SpAMM_charm(\(\tau, A, B, C\))
2: for \(t \geq 0 \wedge t < d\) do
3: `CONVOLUTION[d] . occlude(\(\tau\))`  \(\triangleright\) See Alg. 3
4: end for
5: `CONVOLUTION[d] . multiply`
6: `CONVOLUTION[d] . store`
7: end function

The Charm++ algorithm is outlined in Alg. 2 and proceeds in three phases. In the first phase, the multiplication octree is constructed iteratively over the tiers of the three-dimensional chare arrays, shown in lines 2 and 3 of Alg. 2. This phase is equivalent to the recursion of Alg. 1 over tree nodes and is the complexity reducing mechanism of SpAMM. In each iteration of this phase, a broadcast message is sent to all multiplication chares of tier \(t\) (line 3) executing the `occlude` method on the enabled array elements, shown in Alg. 3. The eight matrix norms of the \(A\) and \(B\) nodes of the next tier are multiplied, lines 2-10 of Alg. 3, and Eq. 2 is used to decided whether to enable or disable the corresponding multiplication chares. Disabled multiply chares are skipped during the next iteration of the pruning phase, shown in line 2-3.

<table>
<thead>
<tr>
<th>Algorithm 3</th>
<th>Tree occlusion in the Charm++ programming model of the multiplication chare element on tier (t) with index ((i, j, k)). In Charm++, the call <code>CONVOLUTION[t + 1](i, j, k) . enable</code> translates into a direct message to the <code>enable</code> method of the multiplication chare element with index ((i, j, k)) on tier (t + 1).</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: function occlude((\tau)) (\triangleright) On tier (t), index ((i, j, k)) (\in {1, 2^t})</td>
<td></td>
</tr>
<tr>
<td>2: if isDisabled then return end if</td>
<td></td>
</tr>
<tr>
<td>3: 5: for all (i', j', k' \mid C_{i', j', t} \leftarrow A_{i', j', t} B_{i', j', t} ) do</td>
<td></td>
</tr>
<tr>
<td>4: 6: if (</td>
<td></td>
</tr>
<tr>
<td>7: <code>CONVOLUTION[t + 1](i', j', k').enable</code></td>
<td></td>
</tr>
<tr>
<td>8: else</td>
<td></td>
</tr>
<tr>
<td>9: <code>CONVOLUTION[t + 1](i', j', k').disable</code></td>
<td></td>
</tr>
<tr>
<td>10: end if</td>
<td></td>
</tr>
<tr>
<td>11: end for</td>
<td></td>
</tr>
<tr>
<td>12: end function</td>
<td></td>
</tr>
</tbody>
</table>

During the second phase, line 5 of Alg. 2 the SMP SpAMM code is called to compute the 128 × 128 submatrix products in each remaining, enabled multiplication chare, and the results are stored in a temporary variable local to the chare. In the final phase, line 6 of Alg. 2 all temporary matrix products are gathered in the `store` method, summed, and added to the corresponding chares of \(C\). Since the Charm++ runtime guarantees exclusive execution of chare instances, and explicit locking or other means of synchronization as in the OpenMP implementation are not necessary.

### C. The OpenMP/Charm++ Hybrid

In our hybrid approach, we found the best performance with one node per Charm++ Processing-Element (PE). OpenMP commanding all on node threads and 128 × 128 quadtree chunking as discussed above. This approach avoids the problem of packing and unpacking fragmented memory during chare migration, enabling use of a single `memcpy`, which is efficient in standard libraries such as `libc`. Certainly, optimal chunk and block sizes are likely to be application dependent, an issue beyond the scope of the current work. A further complication of the hybrid approach involves the issue of local vs. absolute addressing: by wrapping an address offset with convenience macros, the OpenMP programming model given in Alg. 1 can be used without modification.

### IV. RESULTS

In this work we consider scalability of the SpAMM kernel in the context of Spectral Projection. In Alg. 1, a
replacement of the SCF eigenvalue problem. Spectral Projection involves nested construction of the matrix Heaviside step-function from the effective SCF Hamiltonian (Fockian), in our case computed in a basis of atom-centered functions. In this work, tightly converged, dense matrices for a sequence of water clusters were computed at the B3LYP/6-31G** level of theory, using FreeON, a suite of programs for O(N) quantum chemistry. This sequence of water clusters corresponds to standard temperature and pressure, and has been used in a number of previous studies. A key aspect of this work is ordering of each cluster with the locality preserving Hilbert curve, yielding clustering and segregation of elements by magnitude as in Fig. 1.

In a previous study, we reported linear scaling computational complexities and SpAMM errors as the max norm of the difference between the SpAMM product and a dense reference product. Here, we consider SpAMM errors that accumulate in iterative application of the second order Spectral Projection scheme (SP2). In all cases, the SP2 solver was run to convergence, taking 40 iterations. Values of $\tau = 10^{-6}$, $10^{-8}$, and $10^{-10}$ are considered for scaling experiments, with $\tau = 10^{-6}$ corresponding to extreme truncation (a highly sparse representation).

All OpenMP scaling studies were run on a fully allocated 48 core, 4-socket AMD Opteron 6168 (Magny Cours architecture) systems running at 1.9 GHz using GNU gcc 4.6.3 and Intel Composer XE 2013 SP1. The Charm++ scaling studies were run on the largest open computer cluster at Los Alamos National Laboratory (LANL), “mustang”, which consists of 1,600 dual socket AMD Opteron 6167 (Magny Cours) nodes with 24 cores per node running at 2.3 GHz, and a total of 38,400 cores using GNU gcc 4.7.2.

A. Error Accumulation

The accumulation of error in Spectral Projection due to the SpAMM kernel is computed here as $\|F(P - \tilde{P})\|$, where $F$ is the Fockian, $\tilde{P}$ is the approximate density matrix computed from $F$ with $\tau \neq 0$, and $P$ is a reference computed with $\tau = 0$. These errors are reported in Fig. 2 demonstrating that the error per molecule exhibits no significant system size dependence for the cases studied here, in agreement with our earlier results on the max norm error behavior of SpAMM, Figs. 5.2 and 5.3 of Ref. Roughly, these results suggest that chemical accuracies (2 kcal/mol) may be retained with 105 water molecules and a SpAMM threshold $\tau = 10^{-10}$.

B. OpenMP scaling

SpAMM_omp (Alg. 1) was benchmarked for the last SP2 iteration of the (H2O)90 density matrix with different SpAMM tolerance values, shown in Fig. 3 for a 48 core AMD Opteron. This example is well within the linear scaling regime for SpAMM calculations, yet small enough to probe a lower molecule/core ratio available on modern SMP platforms. Note that in the fit to Amdahl’s law, $T_s^* + T_p^*/P$, the serial component $T_s^*$ is approximately constant, while an increasing $\tau$ leads to a decreasing parallel component, $T_p^*$, consistent with an increase in sparsity and irregularity. We find that Intel’s
OpenMP runtime exhibits near ideal parallel scaling with a very small serial component up to 48 threads. While the OpenMP runtime of gcc exhibits near ideal parallel scaling up to 24 threads, it diverges strongly from Amdahl like behavior beyond 24 threads.

C. Charm++ Scaling

This study involved scaling with the progression \( P = 24 \times 2^m \), up to an available 24,576 cores (1024 nodes) on LANL’s largest, “mustang”. The study consisted of Spectral Projection via the SP2 method until convergence (40 iterations) with \( m = 1, 2, \ldots, 10 \), and \( \tau = 10^{-6}, 10^{-8}, \) and \( 10^{-10} \). The initial data distribution during the first SP2 iteration was given by the Charm++ default static load balancer. After each iteration of the SP2 algorithm, the GreedyCommLB load balancer of Charm++ is called to migrate matrix and multiply chares, rebalancing work and data. To demonstrate the efficiency of the GreedyCommLB load balancer, we show walltime vs. cores for the first iteration (only statically balanced), Figs. 4, 5, and for the final iteration, Figs. 7, 8, 9. Notice that the difference in wall time between the first and last iteration is due to matrix fill-in (the decay slows from Fockian to density matrix). On the first iteration, we observe scaling roughly to \( P = 30 N \), corresponding to the default Charm++ data distribution. After a few iterations however, the communication aware persistence-based GreedyCommLB load balancer dynamically migrates chares to achieve a balance of very high quality. In applications, the persistence-based load balance will remain effective between SCF cycles, and also as atomic-positions gradually evolve, e.g. in a molecular dynamics simulation, geometry optimization, etc., mitigating inefficiencies associated with the first iterations.

In Table I we list parameters for Amdahl’s law, \( T_s + \frac{T_p}{P} \), corresponding to the fitted lines in Figs. 7, 8, 9. Also given in Table I are the corresponding break-even core counts, \( P_{\text{even}} = \frac{T_p}{T_s} \), the ratio between parallel and serial components. The break-even is a conservative estimate of the core-count at which additional scaling becomes ineffective due the left over serial component, which was found to be 1-3 seconds in all cases. Also, as in the pure OpenMP case, Fig. 3 we notice a pronounced decrease in the parallel component with decreasing values of \( \tau \), due to sparse-irregular effects.

V. CONCLUSIONS

Relative to the \( \sim 4 \) heavy atoms/core granularity achieved in the weak limit by advanced parallel methods, the default “static” distribution of work exhibited by our OpenMP/Charm++ implementation achieves roughly \( P = 30 N \), as shown in Figs. 4, 5, and 6. Assuming 1 water molecule \( \approx 2 \) heavy atoms, our default is \( \sim 60 \times \) more scalable. Once persistence
is employed however, our results extend into the strong scaling regime, yielding $P = 400 N$ to $600 N$ as inferred from Table I. For working accuracies and larger systems, e.g. $N \gg 150$ and $\tau \in [10^{-8}, 10^{-12}]$, we expect substantially better results as suggested by Table I. We also expect substantially better results for problems with slower decay, as for example problems involving semi-conductors and metal oxides.

Based on the results given in Fig. 3, the (very modest) serial component seems to be due to the Charm++ runtime. Larger calculations on larger computers will allow the reliable collection of diagnostics, as well as examination of the relationships between data locality and communication.

These satisfactory results follow from over-decomposition of the three-dimensional convolution space, relative to conventional methods that involve one- or two-dimensional decomposition, and from runtime systems that support the irregular task parallelism inherent in the $N$-Body programming model. The ability to recursively generate singleton chares would greatly simplify the implementation of $N$-Body methods, and enhance their efficiency by eliminating the explicit management of tree-traversal as explained in Section III B. This prospect, together with a unified code base for $N$-Body solver collectives (based on established prototypes), may offer a simple and well posed approach to meeting the challenges of increasing hardware complexity.

VI. ACKNOWLEDGEMENTS

NB and MC thank the LDRD program for funding this research under LDRD-ER grant 20110230ER. Both would also like to acknowledge generous support from the Ten-Bar Café providing stimulating and helpful discus-
The Los Alamos National Laboratory is operated by Los Alamos National Security, LLC for the NNSA of the US-DoE under Contract No. DE-AC52-06NA25396.

Electronic address: nicolasbock@freeon.org

M. Benzi and G. H. Golub, BIT
S. K. Abdali and D. S. Wise, in


Charm++, http://charm.cs.uiuc.edu/


J. Lifflander, S. Krishnamoorthy, and L. Kalé, in Proceed-


100 A. Szabo and N. Ostlund, Modern quantum chemistry: introduction to advanced electronic structure theory (Dover Pubns, 1996).


108 Already, the number of cores in the current top 5 supercomputers is close to or even exceeds this number: Tianhe-2 – 3,120,000 cores, Titan – 560,640 cores, Sequoia – 1,572,864 cores, K – 705,024 cores.