

OPTIMIZING COMPUTATIONAL KERNELS IN QUANTUM CHEMISTRY

A Dissertation
Presented to
The Academic Faculty

By

Matthew C. Schieber

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in the
School of Computational Science and Engineering

Georgia Institute of Technology

February 2018

Copyright © Matthew C. Schieber 2018

OPTIMIZING COMPUTATIONAL KERNELS IN QUANTUM CHEMISTRY

Approved by:

Dr. Burdell, Advisor
School of Myths
Georgia Institute of Technology

Dr. Two
School of Mechanical Engineering
Georgia Institute of Technology

Dr. Three
School of Electrical Engineering
Georgia Institute of Technology

Dr. Four
School of Computer Science
Georgia Institute of Technology

Dr. Five
School of Public Policy
Georgia Institute of Technology

Dr. Six
School of Nuclear Engineering
Georgia Institute of Technology

Date Approved: January 11, 2000

A great quote to start the thesis

George P. Burdell

To my mom and dad.

ACKNOWLEDGEMENTS

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

TABLE OF CONTENTS

Acknowledgments	v
List of Tables	viii
List of Figures	ix
Chapter 1: Introduction and Background	1
Chapter 2: Utilizing Spatial Sparsity	5
2.1 Sparsity Masks	5
2.2 Integral Construction	7
2.3 Integral Transformation	8
2.4 Proof of Implementation	9
Chapter 3: Optimizing Integral Transformations	13
3.1 A note on disk-bound blocking	13
3.2 Memory layout for sparsity-utilized transformations	15
3.3 Context Dependent Workflows	17
3.4 Intermediate Recycling	19
Chapter 4: Evaluating Coulomb and Exchange Matrices	21

Chapter 5: Results	24
5.1 Sparsity Screening	24
5.2 Parallel Scaling of Transformations	27
5.3 Context Dependent Workflows	30
5.3.1 Performace Crossover Through Number of Iterations	30
5.3.2 Superior Workflows in Practice	32
5.4 Coloumb and Exchange Builds	33
Chapter 6: Conclusions	34
Appendix A: Experimental Equipment	36
Appendix B: Data Processing	37
References	37
Vita	38

LIST OF TABLES

2.1	Characteristics of benzene stack systems. N and N_{aux} refer to the number of primary and auxiliary basis functions, respectively. Mask sparsity refers to the percentage of significant AO function pairs in the sparsity mask. Mask sparsity increases with additional benzenes added to the stack.	10
2.2	Speedups obtained from sparsity screening at ten benzenes from data in Figure 2.	11
3.1	Speedups obtainable via pre-transforming the 3-center integrals prior to metric contraction for common occupied-virtual transformations.	18
5.1	Characteristics of benzene stack systems. N and N_{aux} refer to the number of primary and auxiliary basis functions, respectively. Mask sparsity refers to the percentage of significant AO function pairs in the sparsity mask. Mask sparsity increases with additional benzenes added to the stack.	25
5.2	Speedups obtained from sparsity screening at ten benzenes from data in Figure 2.	26
5.3	Characteristics of organoboron catalyst. N and N_{aux} refer to the number of primary and auxiliary basis functions, respectively. Mask sparsity refers to the percentage of significant AO function pairs in the sparsity mask.	27
5.4	Characteristics of organoboron catalyst systems across the cc-pVDZ, cc-pVTZ, and cc-pVQZ basis sets.	31

LIST OF FIGURES

2.1	Schwarz sparsity masks used for benzene stacks containing 1, 5, and 10 benzenes spaced 3 apart. Sparsity masks were obtained by evaluating $(\mu\nu \mu\nu) < \frac{\tau^2}{(\mu\nu \mu\nu)_{max}}$	6
2.2	Comparison of execution times using sparsity screening (blue) against no sparsity screening (orange). Execution time is plotted against number of benzenes in a benzene stack from one to ten benzenes. Transformations involved computing $(ib Q)$, where i and b denote occupied and virtual indices, respectively. Cutoff refers to the Schwarz screening threshold. (a) Computing the integrals. (b) Contracting AO integrals with the fitting metric. (c) first transformation times only. (d) total transformation times.	11
5.1	Comparison of execution times using sparsity screening (blue) against no sparsity screening (orange). Execution time is plotted against number of benzenes in a benzene stack from one to ten benzenes. Transformations involved computing $(ib Q)$, where i and b denote occupied and virtual indices, respectively. Cutoff refers to the Schwarz screening threshold. (a) Computing the integrals. (b) Contracting AO integrals with the fitting metric. (c) first transformation times only. (d) total transformation times.	26
5.2	Transition state for organoboron addition to trifluoroacetone. Taken from Ref. [Lec:2016:768]	28
5.3	Speedup and execution time plots obtained using our optimized memory layout from Section 3 for sparsity screened 3-center integrals. Execution times involve computing three common transformations: $(ij Q)$, $(ib Q)$, and $(ab Q)$, where i, j and a, b denote occupied and virtual indices, respectively. Graphs (a), (c), and (e) include speedups for constructing the integrals (orange), transforming (blue), total time (red), and ideal (black). Graphs (b), (d), and (f) plot total execution times. Problem sizes were increased by increasing basis set size using cc-pVXZ, X = D,T,Q.	29

- 5.4 Comparison of execution times for the Store and Direct algorithms to complete $(ij|Q)$, $(ib|Q)$, and $(ab|Q)$ transformations across the cc-pVDZ, cc-pVTZ, and cc-pVQZ basis sets. A scan from one to ten transformations was performed. In each case, a crossover occurs as the Direct algorithm becomes more expensive. The crossover occurs in fewer iterations for transformations involving larger MO spaces. With increasing basis set size, the crossover point is shifted to the right for the $(ij|Q)$ and $(ib|Q)$ transformations and it is shifted slightly to the left for the $(ab|Q)$ transformation. . . . 31
- 5.5 Systems used for context dependent investigation of the Store and Direct workflows. (a) Hexatriene. (b) Ethylene in 50 methanol solvent molecules. 33

SUMMARY

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

CHAPTER 1

INTRODUCTION AND BACKGROUND

Electron repulsion integrals (ERIs) and operations involving them pose a fundamental computational problem for quantum chemistry. These 2-electron, 4-center integrals take the form:

$$(\mu\nu|\lambda\sigma) = \int \int \mu(\mathbf{r}_1)\nu(\mathbf{r}_1)r_{12}^{-1}\lambda(\mathbf{r}_2)\sigma(\mathbf{r}_2)d\mathbf{r}_1d\mathbf{r}_2 \quad (1.1)$$

Where $\mu, \nu, \lambda, \sigma$ are atomic orbital (AO) basis functions and \mathbf{r}_1 and \mathbf{r}_2 are electron coordinates. For simplicity, we have assumed the orbitals are real functions, as is usually the case in practice. Immediately, one should note the $\mathcal{O}(N_{AO}^4)$ storage costs (where N_{AO} is the number of AO basis functions), which quickly limits the size of in-core investigations. Moreover, the two fundamental operations involving ERIs are also costly. The first operation involves their evaluation to build the Coulomb and exchange matrices:

$$J[D_{\lambda\sigma}]_{\mu\nu} = (\mu\nu|\lambda\sigma)D_{\lambda\sigma} \quad (1.2)$$

$$K[D_{\lambda\sigma}]_{\mu\nu} = (\mu\lambda|\nu\sigma)D_{\lambda\sigma} \quad (1.3)$$

Where $J[D_{\lambda\sigma}]_{\mu\nu}$, $K[D_{\lambda\sigma}]_{\mu\nu}$, and $D_{\lambda\sigma}$ are the Coulomb, exchange, and density matrices, respectively. Evaluating the 4-center ERIs according to (1.2) and (1.3) requires $\mathcal{O}(N_{AO}^4)$ operations, which is a heavy computational burden that plagues even the most basic quantum chemistry procedures. The second fundamental operation when using ERIs involves their transformation from the AO basis to the molecular orbital (MO) basis. This operation

is written as:

$$(pq|rs) = C_{p\mu}C_{q\nu}(\mu\nu|\lambda\sigma)C_{r\lambda}C_{s\sigma}, \quad (1.4)$$

where p, q, r, s denote MO indices and orbital matrices, C , were introduced. These integral transformations scale as $\mathcal{O}(N_{AO}^4 N_p)$ (where N_p is the size of the denoted MO basis) and serve as a major bottleneck for perturbative correlation methods like second-order Møller-Plesset perturbation theory (MP2).

To overcome the drastic cost of evaluation, transforming, and processing the 4-center ERIs, various approximations and screening techniques have been developed. Density fitting is one method for alleviating these costs which has been gaining popularity in electronic structure theory. By utilizing the resolution of the identity technique, density fitting reduces the rank of the 4-center ERIs and approximates them using a 3-center form:

$$(\mu\nu|P) = \int \int \mu(\mathbf{r}_1)\nu(\mathbf{r}_1)r_{12}^{-1}P(\mathbf{r}_2)d\mathbf{r}_1d\mathbf{r}_2 \quad (1.5)$$

$$[J]_{PQ} = \int \int P(\mathbf{r}_1)r_{12}^{-1}Q(\mathbf{r}_2)d\mathbf{r}_1d\mathbf{r}_2 \quad (1.6)$$

$$(\mu\nu|\lambda\sigma) \approx (\mu\nu|P)[J]_{PQ}^{-1}(Q|\lambda\sigma) \quad (1.7)$$

Here we have introduced an auxiliary basis P, Q and the Coloumb fitting metric $[J]_{PQ}$. In practice, permuational symmetry is almost always used so that only one set of these 3-center integrals need be built:

$$B_{\mu\nu}^P = [J]_{PQ}^{-\frac{1}{2}}(\mu\nu|Q) \quad (1.8)$$

$$(\lambda\sigma|\mu\nu) \approx B_{\mu\nu}^P B_{\lambda\sigma}^P \quad (1.9)$$

The B tensors indicate integrals that have been contracted with the fitting metric. The rank-reduced 3-center ERIs yield a space complexity of $\mathcal{O}(N_{aux}N_{AO}^2)$, which allows many

more investigations to be run in-core. Using the 3-center ERI's, the Coulomb and exchange matrices are evaluated as:

$$J[D_{\lambda\sigma}]_{\mu\nu} = B_{\mu\nu}^P B_{\lambda\sigma}^P D_{\lambda\sigma} \quad (1.10)$$

$$K[D_{\lambda\sigma}]_{\mu\nu} = B_{\mu\lambda}^P B_{\nu\sigma}^P D_{\lambda\sigma} \quad (1.11)$$

Using (1.8), the Coulomb matrix is evaluated in $\mathcal{O}(N_{aux}N_{AO}^2)$ operations, which is a great success for reducing computational complexity. On the other hand, (1.9) still requires $\mathcal{O}(N_{aux}N^3)$ operations. Since N_{aux} is always larger than N_{AO} , evaluating (1.13) is actually more expensive than evaluating (1.10). This leaves exchange matrix evaluation as the major bottleneck for most density-fitted procedures. One method for alleviating this bottleneck involves employing knowledge of the form of the density matrix:

$$D_{\lambda\sigma} = C_{p\lambda}C_{p\sigma} \quad (1.12)$$

Since MO spaces are often much smaller than AO spaces, it is possible to lower the complexity prefactor when building the exchange matrix. Decomposing the density matrix and rewriting the Coulomb and exchange matrix evaluations, we get:

$$J[D_{\lambda\sigma}]_{\mu\nu} = B_{\mu\nu}^P B_{\lambda\sigma} C_{p\lambda} C_{p\sigma} \quad (1.13)$$

$$K[D_{\lambda\sigma}]_{\mu\nu} = B_{\mu\lambda}^P B_{\nu\sigma} C_{p\lambda} C_{p\sigma} \quad (1.14)$$

Evaluating both (1.13) and (1.14) requires $\mathcal{O}(N_{aux}N_{AO}^2N_p)$ operations. This technique is not beneficial for Coulomb matrix evaluations, since the complexity of (1.13) is actually higher than that of (1.10). However, (1.14) reduces the complexity of exchange matrix evaluation by $\frac{N_{aux}}{p}$, which provides moderate speedups in practice.

Moreover, density fitting also improves the cost of integral transformations. Applying

(1.8) to (1.4), we have:

$$B_{pq}^Q = B_{\mu\nu}^Q C_{p\mu} C_{q\nu} \quad (1.15)$$

$$(pq|rs) \approx B_{pq}^Q B_{rs}^Q \quad (1.16)$$

The formal computational scaling of this operations is $\mathcal{O}(N_{aux}N_pN_qN_rN_s)$. However, in practice, algorithms based on density fitting often involve contracting the transformed 3-index integrals, B_{pq}^Q , with other terms, rather than explicit formulation of $(pq|rs)$ as actual intermediates. Therefore, optimizing these procedures will primarily focus on optimizing the computation of B_{pq}^Q .

Although density fitting makes considerable progress towards reducing the space and computation complexity when using the ERIs, there is still work to be done. Importantly, (1.14) and (1.15) still serve as major computational bottlenecks for iterative and perturbative methods, respectively. In my research, I have sought out and implemented various optimizations to further improve the density fitting regime. The essence of my work involves the marriage of sparsity screening, parallel scaling, minimizing disk IO, and optimal contraction paths. This thesis details the progress made via my research. In Chapter 2, I discuss the spatial sparsity of the 3-index integrals and the utilization of such sparsity for integral construction, metric contractions, and integral transformations. In Chapter 3, I focus on (1.15) by discussing its implementation, blocking procedures, disk implications, and detail how the context in which transformations occur dictate which of two unique algorithms is superior. Finally, Chapter 4 focuses on (1.14) and displays the impact of my work for improving the state of the art.

CHAPTER 2

UTILIZING SPATIAL SPARSITY

For all but the smallest molecules, spatial sparsity is very important for achieving computational savings in quantum chemistry. There exists two primary forms of spatial sparsity in (1.3). The first form results from the Gaussian product $\mu(\mathbf{r}_1)\nu(\mathbf{r}_1)$ diminishing with the overlap of the two AO Gaussians. This product diminishes rapidly with distance between the Gaussian function centers and is also highly sensitive to their degree of locality. If the overlap between two AO Gaussians is insignificant, then it becomes unnecessary to compute any shell triplet featuring this AO Gaussian pair. Utilizing this pairwise sparsity requires only an estimation of the significant AO Gaussian pairs. Once this spatial sparsity is applied and insignificant AO function pairs are screened, the complexity of computing the 3-center integrals becomes $\mathcal{O}(N_{aux}N_{AO}^{1-2})$, where the lower bound is achieved for sufficiently sparse systems. Another type of spatial sparsity in (1.3) involves the auxiliary function, which is more complicated to take advantage of. see Ref. [Werner:2003:8149]. In this thesis, I focus on the former, pairwise sparsity which is featured in the product $\mu(\mathbf{r}_1)\nu(\mathbf{r}_1)$. Utilizing this sparsity not only speeds up the computation of the three-index integrals, but stored in a sparse format, then this sparsity can be utilized in later operations such as the fitting metric contraction and basis transformations.

2.1 Sparsity Masks

The pairwise sparsity available to the 3-center integrals can be determined via an estimation of significant AO function pairs. An implementation will require a two dimensional sparsity mapping structure which keeps track of the significant pairs. We refer to these structures as sparsity masks, which are N_{AO} by N_{AO} boolean matrices denoted as $S_{\mu\nu}^b$. The superscript b reminds the reader that the structure is of boolean type. The element $S_{\mu\nu}^b$ will be true if

the product $\mu(r)\nu(r)$ is estimated to be significant and false otherwise.

For efficiency, it is common to compute batches of integrals in *shell triples*:

$$(MN|Q) = \{(\mu\nu|P), \mu \in M, \nu \in N, Q \in P\} \quad (2.1)$$

An individual integral $(\mu\nu|P)$ is bounded by the Schwarz inequality $(\mu\nu|P) \leq \sqrt{(\mu\nu|\mu\nu)(P|P)}$.

We can eliminate entire shells of integrals, $(MN|P)$, where M and N denote AO shells, by computing and storing the maximum $(\mu\nu|\mu\nu)$ that occurs for a given shell $\mu \in M, \nu \in N$.

We use a relative cutoff tolerance τ and neglect shells for which

$$MN_{max} < \frac{\tau^2}{(\mu\nu|\mu\nu)_{max}}, \quad (2.2)$$

where MN_{max} is the maximum $(\mu\nu|\mu\nu)$ within a shell pair (M, N) and $(\mu\nu|\mu\nu)_{max}$ is the global maximum $(\mu\nu|\mu\nu)$. In addition, even for shell pairs that are not screened out, we can apply function-level screening and avoid computation of $(\mu\nu|P)$, where

$$(\mu\nu|\mu\nu) < \frac{\tau^2}{(\mu\nu|\mu\nu)_{max}}. \quad (2.3)$$

Using this strategy, a sparsity mask can be easily constructed. Figure 1 includes illustrations of sparsity masks for benzene stacks with 1, 5, and 10 benzenes spaced 3 Å apart. As sparsity increases, the number of significant AO function pairs contained in these masks becomes $\mathcal{O}(N_{AO}^{1-2})$, where the lower bound is obtained for sufficiently sparse systems.

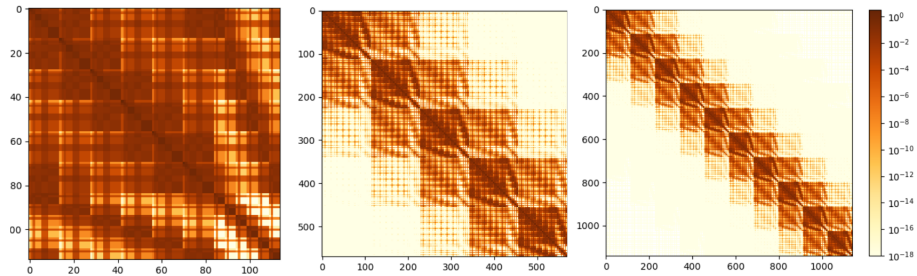


Figure 2.1: Schwarz sparsity masks used for benzene stacks containing 1, 5, and 10 benzenes spaced 3 Å apart. Sparsity masks were obtained by evaluating $(\mu\nu|\mu\nu) < \frac{\tau^2}{(\mu\nu|\mu\nu)_{max}}$.

2.2 Integral Construction

Now, we are equipped to discuss the resulting algorithms when applying the integral sparsity to important operations such as integral construction and transformation. To write more readable algorithms, we employ some tensor notation and write $A_{\mu\nu}^P$ to represent the pre-contracted 3-center integrals $(\mu\nu|P)$. To prune these integrals using sparsity, the following algorithm can be employed:

Algorithm 1 Prune $A_{\mu\nu}^P$ using sparsity

Require: AO integrals: $A_{\mu\nu}^P$, screening mask: $S_{\mu\nu}^b$
for $\mu = 0$ to $\mu = N_{AO} - 1$ **do**
 $A_{\mu\nu}^P S_{\mu\nu}^b \rightarrow A_{\mu\nu\mu}^P$
end for
return $A_{\mu\nu\mu}^P$

Here we have introduced the symbol ν^μ , which indicates that ν is restricted to AO functions which are spatially close enough to μ to survive the Schwarz screening process. The superscript in ν^μ indicates a dependence of ν according to μ . Algorithm 1 is purely pedagogical, as one would never build the full $A_{\mu\nu}^P$ integrals and then prune them for sparsity. Rather, sparsity screening would be applied as the integrals are constructed and insignificant function triplets are never computed. The computation and storage of $A_{\mu\nu\mu}^P$ scales as $\mathcal{O}(N_{aux}N_{AO}^{1-2})$. Moreover, the costly metric contraction becomes

$$B_{\mu\nu\mu}^Q = A_{\mu\nu\mu}^P [J]_{PQ}^{-1} \quad (2.4)$$

and scales as $\mathcal{O}(N_{aux}^2 N_{AO}^{1-2})$, where the lower bound is achieved for sufficiently sparse systems.

2.3 Integral Transformation

The transformation of $A_{\mu\nu}^P$ (or $B_{\mu\nu}^Q$) from the AO basis to the MO basis is an essential operation for many quantum chemistry procedures. The operation requires two MO matrices: $C_{\mu p}, C_{\nu q}$, where p, q are general MO space indices. A first contraction $A_{p\nu}^P = A_{\mu\nu}^P C_{\mu p}$ will half-transform the integrals and costs $\mathcal{O}(N_{aux} N_{AO}^2 N_p)$. The second contraction $A_{pq}^P = A_{p\nu}^P C_{\nu q}$ will cost $\mathcal{O}(N_{aux} N_{AO} N_p N_q)$. Note the first contraction should involve the smaller of N_p, N_q in order to reduce complexity. Also, the first contraction is comparably more expensive than the latter since the size of the AO space is larger than most MO spaces. Therefore, reducing the cost of the first contraction would alleviate a bottleneck overall. Thankfully, we can exploit the sparsity of $A_{\mu\nu}^P$. To do so, we carry out a looping DGEMM through the μ index and apply the sparsity mask to the orbital matrix $C_{\mu p}$. Algorithm 2 illustrates the resulting process:

Algorithm 2 Transform sparse integrals $A_{\mu\nu}^P$ to MO spaces.

Require: Sparse AO integrals: $A_{\mu\nu}^P$, orbital matrices: $C_{\mu p}, C_{\nu q}$, screening mask: $S_{\mu\nu}^b$

for $\mu = 0$ to $\mu = N_{AO} - 1$ **do**

$$C_{\nu p} S_{\mu\nu}^b \rightarrow C_{\nu^\mu p}$$

$$A_{\mu\nu}^P C_{\nu^\mu p} \rightarrow A_{\mu p}^P$$

end for

return $A_{\mu q}^P$

$$A_{\mu q}^P C_{\mu p} \rightarrow A_{pq}$$

return A_{pq}^P

The cost of the first contraction becomes $\mathcal{O}(N_{aux} N_{AO}^{1-2} N_p)$. The remaining sparsity in the half-transformed $A_{\nu q}^P$ is unrelated to the original sparsity mask.

2.4 Proof of Implementation

All methods were implemented in the PSI4 electronic structure software package. The parallelism in PSI4 relies on the shared memory programming model using OpenMP and carries out matrix multiplications using Intel’s Math Kernel Library.

We first demonstrate the performance of our Schwarz screening implementation by measuring the performance when computing the 3-center integrals, contracting the fitting metric, and transforming the integrals into an MO basis. The experiment employed an ideal sparse system: stacked benzenes. Execution times were recorded for each successive benzene added to the stack, from one to ten benzenes, with each benzene spaced 5Å apart. Transformation time involved the wall time required to carry out the common occupied-virtual transformation, as would be required by density-fitted MP2:

$$(ib|Q) = (\lambda\sigma|Q)C_{\sigma i}C_{\lambda b} \quad (2.5)$$

Where i, j and a, b denote occupied and virtual spaces, respectively. Algorithm 4 was implemented and used to carry out these transformations. The cc-pVTZ and cc-pVTZ-jkfit basis sets were used for primary and auxiliary basis sets, respectively. The characteristics of each system are listed in Table 2. The mask sparsity listed in Table 2 refers to the percentage of non-significant AO function pairs appearing in the sparsity mask. The experiment was carried out using one node consisting of an Intel Core i7-5930K processor (6 cores at 3.50GHz) and 50GB DRAM. The results are plotted in Figure 2. Figure 2 (a), (b), (c), and (d) involve time to compute the 3-index integrals, contract the fitting metric, perform the first transformation step: $(\mu\nu|Q)C_{i\mu} \rightarrow (i\nu|Q)$, and total transform time, respectively. Note that our novel contribution involves the application of the sparsity screening in the transformation step.

Table 2.1: Characteristics of benzene stack systems. N and N_{aux} refer to the number of primary and auxiliary basis functions, respectively. Mask sparsity refers to the percentage of significant AO function pairs in the sparsity mask. Mask sparsity increases with additional benzenes added to the stack.

Benzenes	N	N_{aux}	Mask Sparsity (%)
1	264	654	2.6
2	528	1308	24.7
3	792	1962	43.6
4	1056	2616	55.3
5	1320	3270	63.1
6	1584	3924	68.6
7	1848	4578	72.7
8	2112	5232	75.9
9	2376	5886	78.4
10	2640	6540	80.4

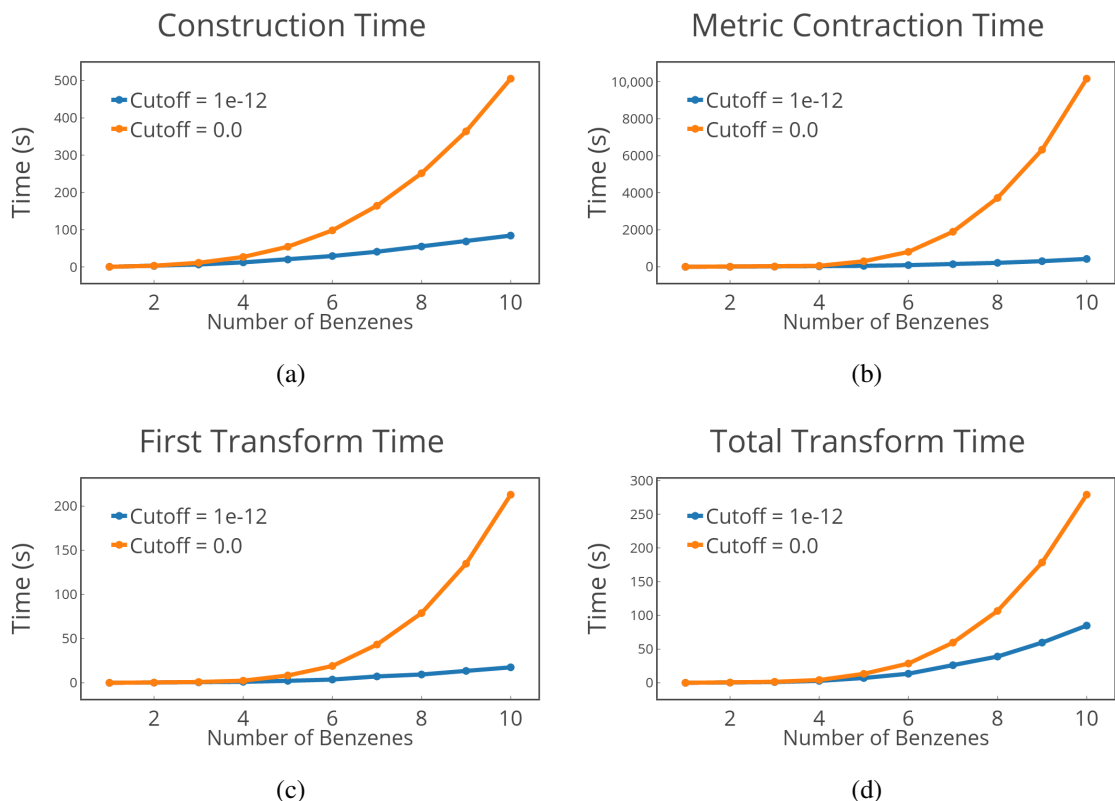


Figure 2.2: Comparison of execution times using sparsity screening (blue) against no sparsity screening (orange). Execution time is plotted against number of benzenes in a benzene stack from one to ten benzenes. Transformations involved computing $(ib|Q)$, where i and b denote occupied and virtual indices, respectively. Cutoff refers to the Schwarz screening threshold. (a) Computing the integrals. (b) Contracting AO integrals with the fitting metric. (c) first transformation times only. (d) total transformation times.

Table 2.2: Speedups obtained from sparsity screening at ten benzenes from data in Figure 2.

Operation	Speedup at 10 benzenes
Construction	10.6
Metric Contraction	37.8
First Transformation	25.8
Total Transformation	5.5

Clearly, Figure 2 reveals significant time reductions for all procedures measured. Table 3 reinforces that operations with higher complexity scaling have the largest time reduction,

with the caveat that total transformation time includes portions without sparsity utilization. In the integral computations (Figure 2 (a)), we construct the sparse integrals using function screening; however, we still compute them in shell triplets for efficiency. Therefore the acquired speedup is slightly less dramatic compared to the metric contraction and transformation procedures (Figure 2 (b) and (c), respectively) due to select functions being screened in cases where the entire shell is not screened. Note that the metric contraction is by far the most expensive operation, which should in one part highlight the boon of sparsity utilization and in another part illustrate the pertinence of our workflow investigation later in this paper. Last, our proposal outlined in Algorithm 2 for applying sparsity screening to integral transformations is proven viable by Figure 2 (c). Note that significant reductions are obtained for the first transformation; however, the sparsity thereafter is unrelated to the initial sparsity mask. With sparsity utilization, the first step scales as $\mathcal{O}(N_{aux}N_{AO}^{1-2}p)$, whereas the second transformation step will still scale as $\mathcal{O}(N_{aux}N_pN_q)$. In this work we have not gone on to consider sparsity of the MO indices, which would typically require transformation to local orbitals.

CHAPTER 3

OPTIMIZING INTEGRAL TRANSFORMATIONS

3.1 A note on disk-bound blocking

The size of tensors grow rapidly in quantum chemistry. The 3-center integrals of density fitting are no exception. Often, the size of the AO three-index integrals will exceed 50GB of RAM for systems as small as 40 atoms when a moderate basis set such as aug-cc-pVQZ is used, even with sparsity screening. At that point, it is necessary for any implementation to begin reading and writing these tensors to and from disk-based memory. For any field this can be a major slowdown, but it is especially critical to performance when high dimensional data is involved. To illustrate this issue, I will introduce an adapted tensor notation which better indicates memory layout.

We denote an n -dimensional tensor as $T_{ab...n}$, where the indices from left to right go from the slowest-running to the fastest-running indices. Here, a is the slowest-running index, b is the next slowest-running index, and n is the fastest-running index. The choice of memory layout plays a crucial role when indices are being accessed. Iterating through the slowest-running index, a , would require the largest memory strides whereas the elements of the fastest-running index, n , are contiguous in memory.

Now, we can consider two possible forms for the 3-index integrals: $A_{P\mu\nu}$ or $A_{\mu P\mu}$. If these tensors are too large to fit into memory, we must read and write pieces of them to and from disk-based memory. To accomplish this, we must choose an index to block accross. Primarily, this will involve either the P or μ index. For example, if we choose to block accross the P index, then we will partition the basis of P into discrete blocks. Then, we will read and write only those blocks of P along with all of μ and ν . The latency of these operations is bounded by the movement of a physical read-write head, so it is

critically important to ensure that read and writes are as contiguous as possible. In the case of P blocking, the $A_{P\mu\nu}$ tensor is far superior to $A_{\mu P\mu}$ since the former will yield entirely contiguous operations whereas the latter will require strided operations.

Unfortunately, within the density fitting regime, different operations are optimal under different blocking schemes. For example, consider the construction of the full three-index AO integrals according to (1.8). To accomplish this, the initial integrals, $(\mu\nu|Q)$, are computed and then contracted with the fitting metric. If the AOs are too large to fully fit in-core, then we must choose an index, P or μ , to block across. Consider the following two algorithms that block across either index, respectively:

Algorithm 3 Construct the full AO integrals $B_{\mu\nu}^P$ by blocking across the P index.

Require: Coulomb metric: $[J]_{PQ}^{-\frac{1}{2}}$

for block $P_i \in P$ **do**

 Compute: $(\mu\nu|P_i)$

 Contract: $(\mu\nu|P_i)[J]_{P_iQ}^{-\frac{1}{2}} \rightarrow B_{\mu\nu}^Q$

 Accumulate Write: $B_{\mu\nu}^Q$

end for

return $B_{\mu\nu}^Q$

Algorithm 4 Construct the full AO integrals $B_{\mu\nu}^P$ by blocking across the μ index.

Require: Coulomb metric: $[J]_{PQ}^{-\frac{1}{2}}$

for block $\mu_i \in \mu$ **do**

 Compute: $(\mu_i\nu|P)$

 Contract: $(\mu_i\nu|P)[J]_{PQ}^{-\frac{1}{2}} \rightarrow B_{\mu_i\nu}^Q$

 Write: $B_{\mu_i\nu}^Q$

end for

return $B_{\mu\nu}^Q$

Of the two algorithms, only Algorithm 4 would respect the memory constraints of a blocking procedure. Note that after the contraction $(\mu\nu|P_i)[J]_{P_iQ}^{-\frac{1}{2}} \rightarrow B_{\mu\nu}^Q$ in Algorithm 3, the full 3-dimensional quantity $B_{\mu\nu}^Q$ is returned, which would immediately violate memory constraints. For this operation, only one blocking method is *possible*. If we are constrained to blocking across the μ index, then the tensor form $A_{\mu P \mu}$ will yield superior disk performance. The purpose of this illustration is to remind the reader that for large enough systems, disk performance is crucially important, therefore memory layout should be considered carefully.

3.2 Memory layout for sparsity-utilized transformations

Algorithm 2 revealed a technique that can be used to utilize sparsity when carrying out three-index integral transformations. Although utilizing sparsity is imperative for cost reduction, an optimal implementation must be tailored to fully exploit modern computing hardware. The trending paradigm of multicore processors must be respected. Multicore processors consisting of upwards of ten cores are found commonly both at the desk of computational chemists and in commodity computing clusters. Moreover, the birth of Intel’s manycore architecture further necessitates that scientific computing exploit every means of parallelism. Although the density fitting technique has been shown to be challenging to exploit in massively parallel algorithms (cite K computer paper), it remains an essential technique for accelerating computations of small to intermediate size chemical systems. The communication overhead that hinders large scale parallelism for density fitting is nominal for single node investigations, but even a single multicore processor contains viable parallelism that can be challenging to fully exploit. Coupling the cost reduction of sparsity approximations with a finely tuned parallel code is crucial to performance.

Maximizing parallelism in Algorithm 2 will require careful implementation design. The choice of memory layout for the sparse 3-dimensional integral tensors will affect both algorithmic complexity and parallel scalability. For the 3-index integrals, we use $A_{P\mu\nu}$

to denote a memory layout with the auxiliary index P as the slowest-running index. The $A_{P\mu\nu}$ layout is intuitive, as it allows looping through P and directly apply a sparsity mask for each submatrix. The resulting sparse form $A_{P\mu\nu^\mu}$ would yield submatrices of identical structure. However, another form, $A_{\mu P\nu}$, must be considered. The sparse form $A_{\mu P\nu^\mu}$ results in submatrices of differing sizes. However, some advantages may be ascertained. We sought to determine which of these two sparse memory layouts, $A_{P\nu\mu^\nu}$ or $A_{\mu P\nu^\mu}$, is optimal for transforming the integrals into an MO basis. Algorithms 5 and 6 illustrate the difference:

Algorithm 5 Transforming sparse integrals using $A_{P\mu\nu^\mu}$ form.

Require: Sparse AO integrals: $A_{P\mu\nu^\mu}$, orbital matrices: $C_{\mu p}, C_{\nu q}$, screening mask: $S_{\mu\nu}^b$

for $P = 0$ to $P = N_{aux} - 1$ **do**

Trim from dense to sparse: $C_{\nu q} S_{\mu\nu}^b \rightarrow C_{\nu^\mu q}$

$A_{P\mu\nu^\mu} C_{\nu^\mu p} \rightarrow A_{P\mu p}$

end for

return $A_{P\mu p}$

Final transform: $A_{P\mu q} C_{\mu p} \rightarrow A_{Ppq}$

return A_{Ppq}

Algorithm 6 Transforming sparse integrals using $A_{\mu P\nu^\mu}$ form.

Require: Sparse AO integrals: $A_{\mu P\nu^\mu}$, orbital matrices: $C_{\mu p}, C_{\nu q}$, screening mask: $S_{\mu\nu}^b$

for $\mu = 0$ to $\mu = N_{AO} - 1$ **do**

Trim from dense to sparse: $C_{\nu q} S_{\mu\nu}^b \rightarrow C_{\nu^\mu q}$

$A_{\mu P\nu^\mu} C_{\nu^\mu q} \rightarrow A_{\mu Pq}$

end for

return $A_{\mu Pq}$

Final transform: $A_{\mu Pq} C_{\mu p} \rightarrow A_{pPq}$

return A_{pPq}

To carry out the first step of the transformation, both algorithms must loop through the slowest-running index of the integrals. Operations within this loop should be parallelized. The number of iterations for this step are greater in Algorithm 3 than in Algorithm 4 since the auxiliary index will always be larger than the AO index. Conversely, the matrix-matrix multiplications occurring in Algorithm 4 are larger. As a result, Algorithm 4 will benefit from delegating larger problem sizes to highly optimized level 3 BLAS routines.

However, the crucial difference is ascertained when one considers which index, μ or P , would be most appropriate to block across for a disk-bound implementation. If we choose to block across μ , the result of the final transformation, A_{pq}^P , would be incomplete, and a full cumulative disk write of $\mathcal{O}(N_{aux}N_pN_p)$ size would be necessary for each block of μ . Moreover, there is a chance this operation could be altogether impossible if memory constraints would be violated by having a full A_{pq}^P tensor. On the other hand, blocking across the P index would not pose this problem, as no contractions are occurring across the P index. Therefore, blocking across P is the only scalable solution.

Now, if it is necessary to block across P , consider the implications for Algorithms 5 and 6. For Algorithm 5, this means that if the blocks over P are very small, e.g. < 10 , then the parallelized loops will have very few iterations. Typically, the fewer the iterations the worse the parallel scalability as the workloads are much more susceptible to being unbalanced. To the contrary, Algorithm 6 will not suffer this drawback and is therefore the better option. Since Algorithm 4 contains higher concurrency and utilizes larger matrix-matrix multiplications, we propose that it will yield enhanced parallel scaling.

3.3 Context Dependent Workflows

Equation (1.8) demonstrates the necessity of the fitting metric when using the 3-center density-fitted integrals. Unfortunately, the metric contraction in equation (1.8) comes at the heavy price of $\mathcal{O}(N_{aux}^2N^2)$ operations. Considering that N_{aux} can be 2-3x larger than N_{AO} , this can be an extremely costly operation. One remedy for cost reduction is to transform

the 3-center integrals prior to contracting them with the metric:

$$(pq|Q) = (\mu\nu|Q)C_{\mu p}C_{\nu q} \quad (3.1)$$

$$B_{pq}^Q = [J]_{PQ}^{-\frac{1}{2}}(pq|Q) \quad (3.2)$$

Since N_p and N_q are often much smaller than N_{AO} , the speedup of $\mathcal{O}(\frac{N_{AO}^2}{N_p N_q})$ can be substantial. Table 1 illustrates the potential benefit for some commonly used transformed integrals using occupied and virtual spaces. Orbital indices i, j denote occupied spaces and a, b denote virtual spaces.

Table 3.1: Speedups obtainable via pre-transforming the 3-center integrals prior to metric contraction for common occupied-virtual transformations.

Transformation	$\frac{N_{AO}}{N_i} = 2$	$\frac{N_{AO}}{N_i} = 5$	$\frac{N_{AO}}{N_i} = 5$
$(Q ij)$	4	25	100
$(Q ia)$	4	6.25	11.1
$(Q ab)$	4	1.56	1.23

The speedups in Table 1 are undoubtedly beneficial. However, we propose that this technique will be a disadvantageous in certain contexts. Namely, applying this workflow to methods using many transformations will increase cost unnecessarily. Using this method, a metric contraction is necessary for each transformation, whereas only one contraction is required if this technique is not used. If many transformations occur, the cost of contracting the metric for each transformation will eventually outweigh the speedups attainable in Table 1. Therefore, both workflows must be considered when carrying out 3-center integral transformations. Algorithm 7 and 8 illustrate the corresponding workflows:

Algorithm 7 "Store" algorithm - contract metric then transform.

Require: AO integrals: $A_{\mu\nu}^P$, fitting metric: $[J]_{PQ}^{-\frac{1}{2}}$, orbital matrices: $C_{\sigma p}, C_{\nu q}$

Contract metric: $A_{\mu\nu}^P [J]_{PQ}^{-\frac{1}{2}} \rightarrow B_{\sigma\nu}^Q$

Transform: $B_{\mu\nu}^Q C_{\sigma p} C_{\nu q} \rightarrow B_{pq}^Q$

return B_{pq}^Q

Algorithm 8 "Direct" algorithm - transform then contract metric.

Require: AO integrals: $A_{\mu\nu}^P$, fitting metric: $[J]_{PQ}^{-\frac{1}{2}}$, orbital matrices: $C_{\sigma p}, C_{\nu q}$

Transform: $A_{\mu\nu}^P C_{\sigma p} C_{\nu q} \rightarrow A_{pq}^P$

Contract metric: $A_{pq}^P [J]_{PQ}^{-\frac{1}{2}} \rightarrow B_{pq}^Q$

return B_{pq}^Q

Hereafter we refer to the Store algorithm being the workflow which contracts the metric and then transforms the integrals. Conversely, we refer to the Direct algorithm as the workflow which transforms the integrals then contracts the metric.

Depending on the context, either the Store or Direct algorithm may be superior. We propose the Store algorithm will be superior for procedures requiring many transformations. This includes contexts which iteratively recompute transformations (MCSCF) and contexts having a large number of spaces (USAPT). Conversely, the Direct algorithm will be superior in contexts requiring a small number of transformations. Most notably, this includes DFMP2.

3.4 Intermediate Recycling

Most often, quantum chemistry procedures will require numerous integral transformations. For example, a USAPT procedure will employ upwards of 24 unique integral transformations. Since all transformations are carried out at the same time, one should build an implementation that queues the transformations, gathers information, and deploys strategic contraction paths. As mentioned previously, the first contraction of these integral transfor-

mations $A_{p\nu}^P = A_{\mu\nu}^P C_{\mu p}$ should always be carried out on the smallest MO index p possible. Thereafter, transformations using the same intermediate, $A_{p\nu}^P$, should all occur at the same time to avoid recomputing $A_{p\nu}^P$. For example, if two sets of transformed integrals are required A_{uv}^P, A_{up}^P where $u, v \ll N$ are active indices and p is a general MO index where $N_p \approx N_{AO}$. For both transformations, the first step will be:

$$A_{\mu\nu}^P C_{\mu u} \rightarrow A_{uv}^P \quad (3.3)$$

If this is recognized beforehand, then this operation need only be carried out once and the intermediate A_{uv}^P can be recycled for both transformations. The speedup of doing so is $\mathcal{O}(\frac{2N_{AO}+N_v+N_p}{N_{AO}+N_v+N_p})$, which as $N_p \rightarrow N_{AO}$ approaches 50%. Although the benefit is modest, it must be considered for optimized procedures.

CHAPTER 4

EVALUATING COULOMB AND EXCHANGE MATRICES

From Chapter 1, we wrote the evaluations of the three-index ERIs to build the Coulomb and Exchange matrices as:

$$J[D_{\lambda\sigma}]_{\mu\nu} = B_{\mu\nu}^P B_{\lambda\sigma}^P D_{\lambda\sigma} \quad (4.1)$$

$$K[D_{\lambda\sigma}]_{\mu\nu} = B_{\mu\lambda}^P B_{\nu\sigma} C_{p\lambda} C_{p\sigma}, \quad (4.2)$$

where (4.1) and (4.2) are evaluated in $\mathcal{O}(N_{AO}^2 N_{aux})$ and $\mathcal{O}(N_{AO}^2 N_{aux} N_p)$ operations, respectively. The exchange matrix evaluation is a major bottleneck within the density fitting regime. As such, it is imperative to take advantage of sparsity and optimize parallel scaling. Thankfully, it turns out that Algorithm 6 is easily extendible to (4.1). In fact, if one uses orbital matrices to build the exchange matrix as in (4.2), the half-transformed intermediate of integral transformations, $A_{\mu Pq}$, is actually identical to the intermediate when building the exchange matrix. Moreover, the memory layout of $A_{\mu Pq}$ is ideal for the subsequent contraction to form K . The following algorithm results:

Algorithm 9 Building the K matrix.

Require: Sparse AO integrals: $A_{\mu P \nu \mu}$, orbital matrices: $C_{\mu p}, C_{\nu q}$, screening mask: $S_{\mu \nu}^b$

```
for  $\mu = 0$  to  $\mu = N_{AO} - 1$  do  
    Trim from dense to sparse:  $C_{\nu q} S_{\mu \nu}^b \rightarrow C_{\nu \mu q}$   
     $A_{\mu P \nu \mu} C_{\nu \mu q} \rightarrow A_{\mu P q}$   
end for  
return  $A_{\mu P q}$   
 $A_{\mu P q} A_{\mu P q} \rightarrow K$   
return  $K$ 
```

Algorithm 9 is an ideal candidate for building the K matrix. However, we must still consider how Algorithm 9 behaves when the in-core memory is constrained such that it becomes necessary to perform blocking operations across the P index. Then, the algorithm becomes:

Algorithm 10 Building the K matrix using $A_{\mu P \nu \mu}$, blocking across P

Require: Sparse AO integrals: $A_{\mu P \nu \mu}$, orbital matrices: $C_{\mu p}, C_{\nu q}$, screening mask: $S_{\mu \nu}^b$

```
for  $P^i \in P$  do  
    Read from disk:  $A_{\mu P^i \nu \mu}$   
    for  $\mu = 0$  to  $\mu = N_{AO} - 1$  do  
        Trim from dense to sparse:  $C_{\nu q} S_{\mu \nu}^b \rightarrow C_{\nu \mu q}$   
         $A_{\mu P^i \nu \mu} C_{\nu \mu q} \rightarrow A_{\mu P^i q}$   
    end for  
     $K = K + A_{\mu P^i q} A_{\mu P^i q}$   
end for  
return  $K$ 
```

Unfortunately, this algorithm will require strided disk reads when reading the $A_{\mu Q^i \nu \mu}$ tensor into memory. Since poor disk IO can drastically decrease performance, it is often better

to force disk operations to be contiguous and any tensors are transposed as necessary in memory. Since it is best to block accross the P index, a better memory layout for the sparse integrals is the $A_{P\mu\nu\mu}$ form. Another possible algorithm can then be formulated:

Algorithm 11 Building the K matrix using $A_{P\mu\nu\mu}$, blocking accross P

Require: Sparse AO integrals: $A_{P\mu\nu\mu}$, orbital matrices: $C_{\mu p}, C_{\nu q}$, screening mask: $S_{\mu\nu}^b$

```

for  $P^i \in P$  do
  Read from disk:  $A_{P^i\mu\nu\mu}$ 
  for  $\mu = 0$  to  $\mu = N_{AO} - 1$  do
    Trim from dense to sparse:  $C_{\nu q} S_{\mu\nu}^b \rightarrow C_{\nu\mu q}$ 
    Copy:  $A_{P^i\mu\nu\mu} \rightarrow B_{P^i\nu\mu}$ 
     $B_{P^i\nu\mu} C_{\nu\mu q} \rightarrow A_{\mu P^i\nu\mu}$ 
  end for
   $K = K + A_{\mu P^i q} A_{\mu P^i q}$ 
end for
return  $K$ 

```

Here, we used a buffer, $B_{P^i\nu\mu}$ to transpose pieces of $A_{P^i\mu\nu\mu}$ into while looping over μ . While this algorithm may involve a strided copy and more additional memory usage, the disk reads for the $A_{P^i\mu\nu\mu}$ tensor are completely contiguous.

When the investigation is completely in-core, then Algorithm 10 should yield superior performance. However, for larger systems, the strided disk reads in Algorithm 10 may cause performance degradation to the point that Algorithm 11 may become superior.

CHAPTER 5

RESULTS

All methods were implemented in the PSI4 electronic structure software package. The parallelism in PSI4 relies on the shared memory programming model using OpenMP and carries out matrix multiplications using Intel’s Math Kernel Library.

5.1 Sparsity Screening

We first demonstrate the performance of our Schwarz screening implementation by measuring the performance when computing the 3-center integrals, contracting the fitting metric, and transforming the integrals into an MO basis. The experiment employed an ideal sparse system: stacked benzenes. Execution times were recorded for each successive benzene added to the stack, from one to ten benzenes, with each benzene spaced 5Å apart. Transformation time involved the wall time required to carry out the common occupied-virtual transformation, as would be required by density-fitted MP2:

$$(ib|Q) = (\lambda\sigma|Q)C_{\sigma i}C_{\lambda b} \quad (5.1)$$

Where i, j and a, b denote occupied and virtual spaces, respectively. Algorithm 4 was implemented and used to carry out these transformations. The cc-pVTZ and cc-pVTZ-jkfit basis sets were used for primary and auxiliary basis sets, respectively. The characteristics of each system are listed in Table 2. The mask sparsity listed in Table 2 refers to the percentage of non-significant AO function pairs appearing in the sparsity mask. The experiment was carried out using one node consisting of an Intel Core i7-5930K processor (6 cores at 3.50GHz) and 50GB DRAM. The results are plotted in Figure 2. Figure 2 (a), (b), (c), and (d) involve time to compute the 3-index integrals, contract the fitting metric, perform

the first transformation step: $(\mu\nu|Q)C_{i\mu} \rightarrow (i\nu|Q)$, and total transform time, respectively.

Note that our novel contribution involves the application of the sparsity screening in the transformation step.

Table 5.1: Characteristics of benzene stack systems. N and N_{aux} refer to the number of primary and auxiliary basis functions, respectively. Mask sparsity refers to the percentage of significant AO function pairs in the sparsity mask. Mask sparsity increases with additional benzenes added to the stack.

Benzenes	N	N_{aux}	Mask Sparsity (%)
1	264	654	2.6
2	528	1308	24.7
3	792	1962	43.6
4	1056	2616	55.3
5	1320	3270	63.1
6	1584	3924	68.6
7	1848	4578	72.7
8	2112	5232	75.9
9	2376	5886	78.4
10	2640	6540	80.4

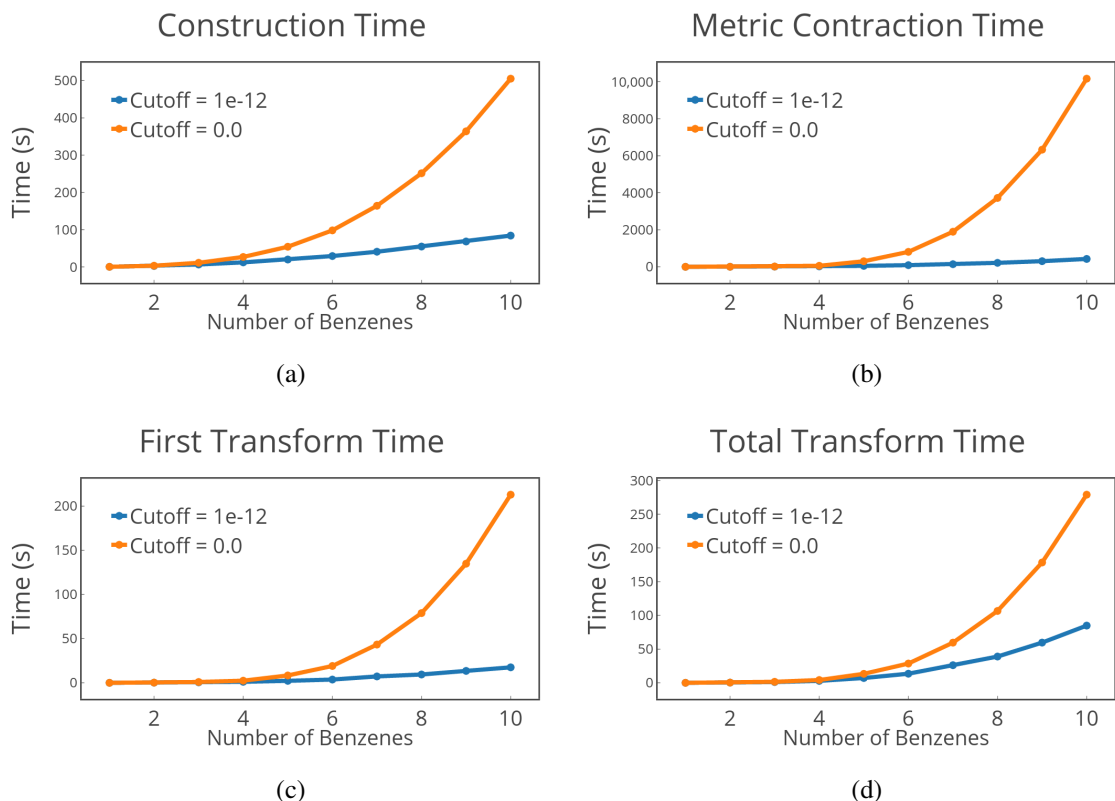


Figure 5.1: Comparison of execution times using sparsity screening (blue) against no sparsity screening (orange). Execution time is plotted against number of benzenes in a benzene stack from one to ten benzenes. Transformations involved computing $(ib|Q)$, where i and b denote occupied and virtual indices, respectively. Cutoff refers to the Schwarz screening threshold. (a) Computing the integrals. (b) Contracting AO integrals with the fitting metric. (c) first transformation times only. (d) total transformation times.

Table 5.2: Speedups obtained from sparsity screening at ten benzenes from data in Figure 2.

Operation	Speedup at 10 benzenes
Construction	10.6
Metric Contraction	37.8
First Transformation	25.8
Total Transformation	5.5

Clearly, Figure 2 reveals significant time reductions for all procedures measured. Table 3 reinforces that operations with higher complexity scaling have the largest time reduction,

with the caveat that total transformation time includes portions without sparsity utilization. In the integral computations (Figure 2 (a)), we construct the sparse integrals using function screening; however, we still compute them in shell triplets for efficiency. Therefore the acquired speedup is slightly less dramatic compared to the metric contraction and transformation procedures (Figure 2 (b) and (c), respectively) due to select functions being screened in cases where the entire shell is not screened. Note that the metric contraction is by far the most expensive operation, which should in one part highlight the boon of sparsity utilization and in another part illustrate the pertinence of our workflow investigation later in this paper. Last, our proposal outlined in Algorithm 2 for applying sparsity screening to integral transformations is proven viable by Figure 2 (c). Note that significant reductions are obtained for the first transformation; however, the sparsity thereafter is unrelated to the initial sparsity mask. With sparsity utilization, the first step scales as $\mathcal{O}(N_{aux}N_{AO}^{1-2}p)$, whereas the second transformation step will still scale as $\mathcal{O}(N_{aux}N_pN_q)$. In this work we have not gone on to consider sparsity of the MO indices, which would typically require transformation to local orbitals.

5.2 Parallel Scaling of Transformations

To measure parallel scaling, we performed integral transformations for the boron catalyst system shown in Figure 3.

Table 5.3: Characteristics of organoboron catalyst. N and N_{aux} refer to the number of primary and auxiliary basis functions, respectively. Mask sparsity refers to the percentage of significant AO function pairs in the sparsity mask.

Basis	N	N_{aux}	Mask Sparsity (%)
cc-pVDZ	671	3277	29.6
cc-pVTZ	1566	3856	41.1
cc-pVQZ	3040	5593	50.2

We varied the problem size by adjusting the ζ level for the Dunning basis sets with $\zeta = \text{D}$,

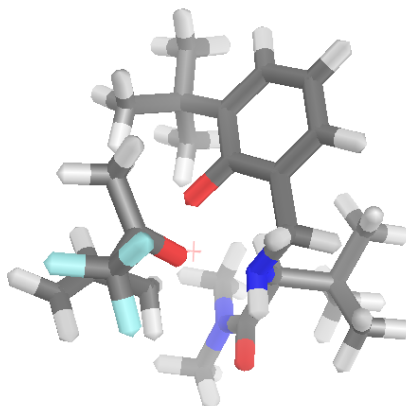


Figure 5.2: Transition state for organoboron addition to trifluoroacetone. Taken from Ref. [Lec:2016:768]

T, Q. The characteristics of these systems are included in Table 4. Note that while parallel scaling typically improves for larger systems (i.e. due to larger workloads), this is not guaranteed in a sparsity regime. Larger systems may contain more sparsity; more sparsity will result in more striding, copying, and irregular sizing, which will hinder parallel scaling. Nonetheless, our method outlined in Algorithm 2 is designed to utilize sparsity while also obtaining maximum parallel efficiency. For each system, we performed the three common transformations:

$$(ij|Q) = (\lambda\sigma|Q)C_{\sigma i}C_{\lambda j} \quad (5.2)$$

$$(ib|Q) = (\lambda\sigma|Q)C_{\sigma i}C_{\lambda b} \quad (5.3)$$

$$(ab|Q) = (\lambda\sigma|Q)C_{\sigma a}C_{\lambda b} \quad (5.4)$$

The experiment was carried out using one node consisting of an Intel Xeon E5-2630 processor (10 cores at 2.20GHz) and 24GB DRAM. Figure 4 includes plots of both speedups and execution times for each system.

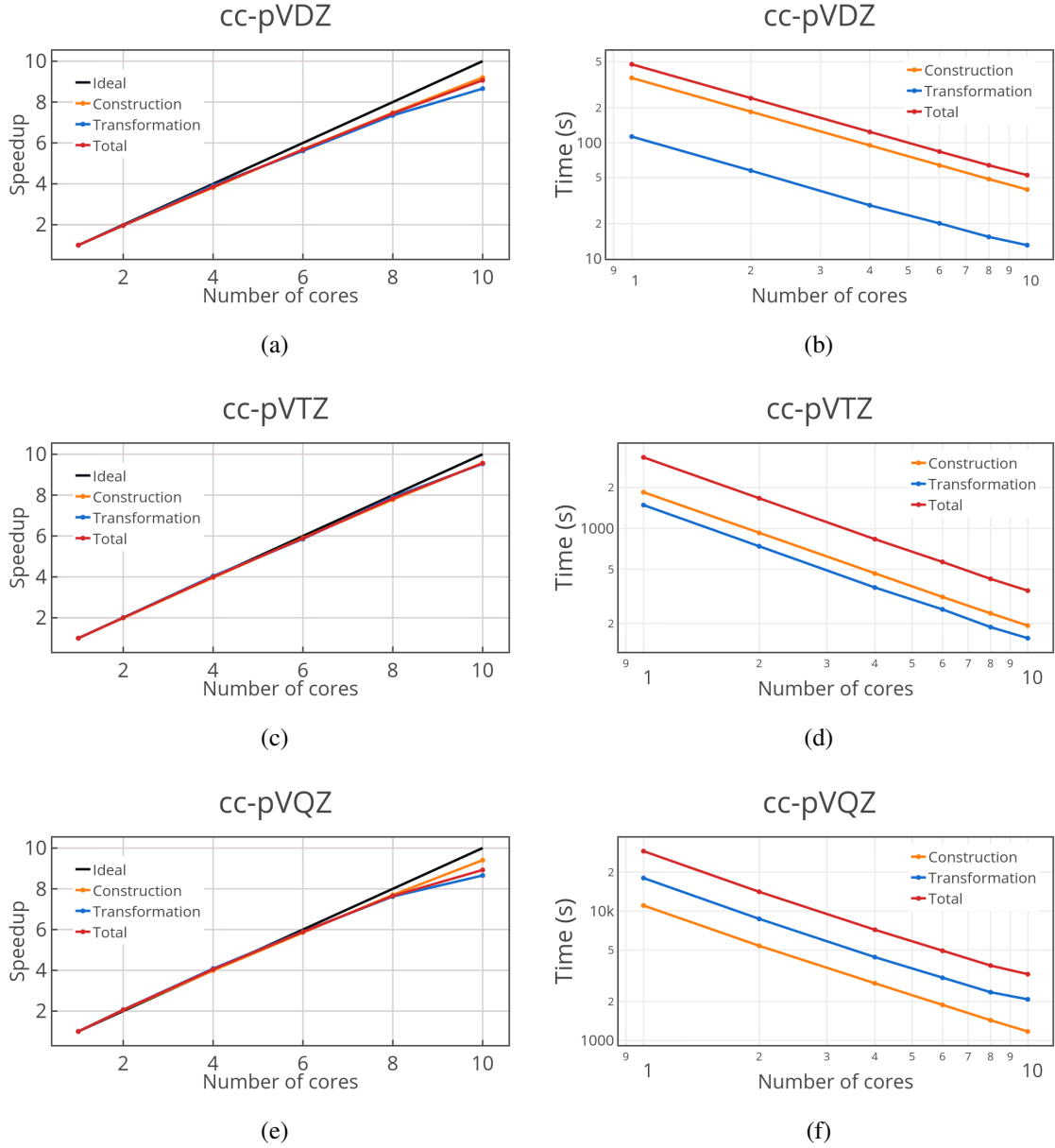


Figure 5.3: Speedup and execution time plots obtained using our optimized memory layout from Section 3 for sparsity screened 3-center integrals. Execution times involve computing three common transformations: $(ij|Q)$, $(ib|Q)$, and $(ab|Q)$, where i, j and a, b denote occupied and virtual indices, respectively. Graphs (a), (c), and (e) include speedups for constructing the integrals (orange), transforming (blue), total time (red), and ideal (black). Graphs (b), (d), and (f) plot total execution times. Problem sizes were increased by increasing basis set size using cc-pVXZ, $X = D, T, Q$.

At ten cores, speedups for total computation time were recorded as 9.09, 9.56, and 8.93 for $\zeta = D, T, Q$, respectively. The improvement in scaling between $\zeta = D$ to $\zeta = T$

may be attributed to larger system sizes. The sizes of the sparsity screened AO integrals were 10.38GB and 55.70GB for these systems, respectively. In the latter case, the 24GB of memory at the compute node was fully used and work for each thread was increased to maximal levels. Conversely, when the system size was increased again using $\zeta = Q$, the memory constraint did not allow for further increase in work per thread. The hindrance in scaling from $\zeta = T$ to $\zeta = Q$ is explained by the workload imbalance incurred by the increase in sparsity.

5.3 Context Dependent Workflows

5.3.1 Performace Crossover Through Number of Iterations

In this section, we reveal the contexts in which either the Store or Direct algorithms are superior. First, we analyzed performance when applying either algorithm to carry out each of the three common integral transformations: $(Q|ij)$, $(Q|ib)$, and $(Q|ab)$. Doing so reveals the crossover in computational complexity that occurs between the two algorithms. For few transformations, the Direct algorithm will be superior as it benefits from the speedups given in Table 1. However, if many transformations occur, we propose the Store algorithm will become superior as it avoids the costly metric contraction for each transformation.

We applied both algorithms to each transformation using the same boron catalyst system in Section 6.2. To reveal the crossover in computational work between the two algorithms, the execution times to carry out one to ten transformations were recorded. To reveal additional trends, we varied the system size by adjusting the basis set size using $\zeta = D, T, Q$. The characteristics of these systems are described in Table 5. The experiments were carried out using one node consisting of an Intel Core i7-5930K processor (6 cores at 3.50GHz) and 50GB DRAM. The results are plotted in Figure 5.

Table 5.4: Characteristics of organoboron catalyst systems across the cc-pVDZ, cc-pVTZ, and cc-pVQZ basis sets.

Basis	N	N_{aux}	N_{occ}	N_{virt}
cc-pVDZ	671	3277	129	542
cc-pVTZ	1566	3856	129	1437
cc-pVQZ	3040	5593	129	2911

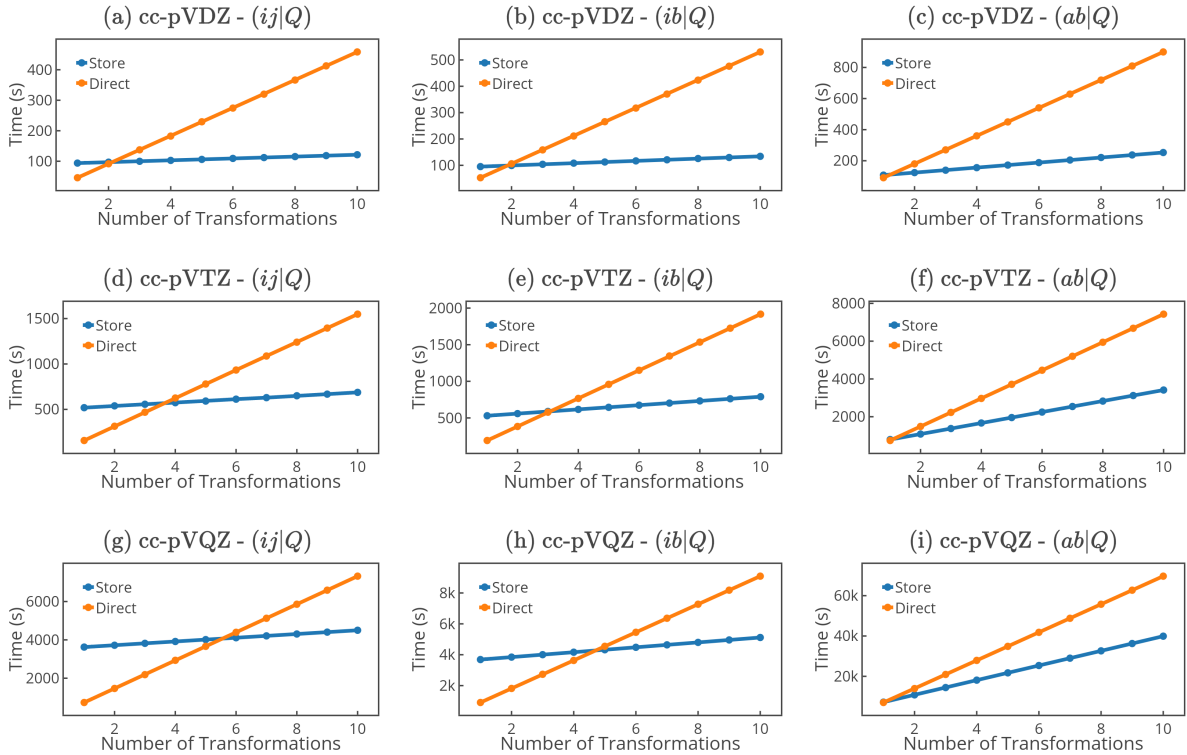


Figure 5.4: Comparison of execution times for the Store and Direct algorithms to complete $(ij|Q)$, $(ib|Q)$, and $(ab|Q)$ transformations across the cc-pVDZ, cc-pVTZ, and cc-pVQZ basis sets. A scan from one to ten transformations was performed. In each case, a crossover occurs as the Direct algorithm becomes more expensive. The crossover occurs in fewer iterations for transformations involving larger MO spaces. With increasing basis set size, the crossover point is shifted to the right for the $(ij|Q)$ and $(ib|Q)$ transformations and it is shifted slightly to the left for the $(ab|Q)$ transformation.

If only one transformation occurs, then the speedups of Table 1 enable the Direct algorithm to be superior. However, the Direct algorithm must carry out expensive metric

contractions for every transformation. As the number of transformations increase, the expense of these contractions overtakes the speedups of pre-transforming the integrals. Note that a crossover between the two algorithms occurs in each system. This finding supports our conjecture that the Store algorithm is advantageous in contexts where many transformations occur. This includes iterative methods where the transformations are carried out in each iteration, such as MCSCF, as well as methods which require many transformations, such as FSAPT and USAPT. Conversely, the Direct algorithm is advantageous for methods requiring few transformations, such as DFMP2.

Additionally, the crossover point shifts to the left as the transformation spaces get larger (ij, ia, ab) , occurring in fewer transformations. This finding is supportive of the proposed speedups in Table 1. Therefore procedures using anything larger than a $(ib|Q)$ transformation will receive nominal benefits from employing the Direct algorithm and may incur slowdowns if many transformations occur. Last, the crossover point shifts to the right for the $(ij|Q)$ and $(ib|Q)$ transformations as larger basis sets are used. This will allow for continued benefits with more transformations. Conversely, the crossover shifts to the left for the $(ab|Q)$ transformation for larger basis sets. Either of these findings are elucidated by the increasing ratio of $\frac{N_{aux}}{N_{AO}}$.

5.3.2 Superior Workflows in Practice

In the previous section, we determined the Direct algorithm will be superior in methods such as DFMP2, while the Store algorithm will be superior in methods such as MCSCF. After determining the contexts in which either algorithm prevail, we sought to reveal their benefits when applied in practice. To do so, we employed either algorithm in the contexts of different procedures and systems. For procedures, we tested MCSCF, DFMP2, and FSAPT. We ran these procedures on the systems included in Figure 6. Figure 6 (a) is a hexatriene molecule. Figure 6 (b) is ethylene molecule surrounded by 50 methanol solvent molecules. Table 6 lists the characteristics of each system, which includes the basis set, the number of

primary and auxiliary basis functions, and the mask sparsity.

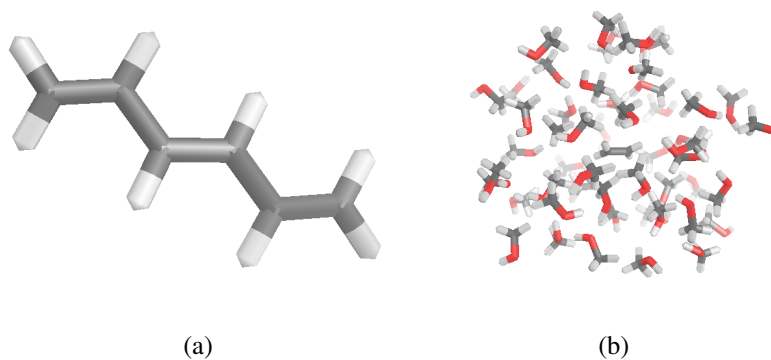


Figure 5.5: Systems used for context dependent investigation of the Store and Direct workflows. (a) Hexatriene. (b) Ethylene in 50 methanol solvent molecules.

5.4 Coloumb and Exchange Builds

CHAPTER 6
CONCLUSIONS

Appendices

APPENDIX A

EXPERIMENTAL EQUIPMENT

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

APPENDIX B

DATA PROCESSING

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

VITA

Vita may be provided by doctoral students only. The length of the vita is preferably one page. It may include the place of birth and should be written in third person. This vita is similar to the author biography found on book jackets.