



On ERI sorting for SIMD execution of large-scale Hartree–Fock SCF

Tirath Ramdas^{a,*}, Gregory K. Egan^a, David Abramson^b, Kim K. Baldridge^c

^a Centre for Telecommunications and Information Engineering, Monash University, Melbourne, Australia

^b Centre for Distributed Systems and Software Engineering, Monash University, Melbourne, Australia

^c Organic Chemistry Institute, University of Zurich, Switzerland

Received 23 August 2007; received in revised form 18 January 2008; accepted 22 January 2008

Available online 1 February 2008

Abstract

Given the resurgent attractiveness of single-instruction-multiple-data (SIMD) processing, it is important for high-performance computing applications to be SIMD-capable. The Hartree–Fock SCF (HF-SCF) application, in its canonical form, cannot fully exploit SIMD processing. Prior attempts to implement Electron Repulsion Integral (ERI) sorting functionality to essentially “SIMD-ify” the HF-SCF application have met frustration because of the low throughput of the sorting functionality. With greater awareness of computer architecture, we discuss how the sorting functionality may be practically implemented to provide high-performance. Overall system performance analysis, including memory locality analysis, is also conducted, and further emphasises that a system with ERI sorting is capable of very high throughput. We discuss two alternative implementation options, with one immediately accessible software-based option discussed in detail. The impact of workload characteristics on expected performance is also discussed, and it is found that in general as basis set size increases the potential performance of the system also increases. Consideration is given to conventional CPUs, GPUs, FPGAs, and the Cell Broadband Engine architecture.

© 2008 Elsevier B.V. All rights reserved.

PACS: 02.60.Pn; 07.05.Bx; 31.15.Ar; 89.20.Ff

Keywords: Hartree–Fock self consistent field; Electron repulsion integrals; Single-instruction-multiple-data processing

1. Introduction

The widespread consensus that microprocessor clock-speed improvements will rapidly decay in the foreseeable future encourages embrace of the “manycore” paradigm in computer architecture. Henceforth it is expected that applications would exploit the multiple and heterogeneous resources available in single processor dies and packages in order to continue to enjoy performance growth. This is a significant disruption: until recently, advancement in computer processor technology could be relied upon to produce increments in performance for existing software. However, going forward, significant increments in performance can only be achieved through modification of im-

plementations and/or algorithms and data structures to match the configuration of new processor architectures.

Computational quantum chemistry is a very large consumer of computing resources.¹ Within the broad umbrella of computational quantum chemistry methods is a group of *ab initio* methods which require the evaluation of a very large number of Electron Repulsion Integrals (ERIs). The most basic of these methods is the Hartree–Fock Self Consistent Field (HF-SCF) method, which suffers $O(N^2)$ to $O(N^4)$ computation time in general. Higher-order post-HF methods that account for electron correlation require much greater computational expense, e.g., $O(N^7)$. The evaluation of ERIs within these methods is the focus of this paper. To provide a tangible context for ERI evaluation as a component of an overall computational method,

* Corresponding author.

E-mail addresses: tirath.ramdas@eng.monash.edu.au (T. Ramdas), greg.egan@eng.monash.edu.au (G.K. Egan), david.abramson@infotech.monash.edu.au (D. Abramson), kimb@oci.unizh.ch (K.K. Baldridge).

¹ It has been reported [1] that electronic structure calculations dominate 85% of total computer system usage at *Pacific Northwest National Laboratories*, 30% at the *National Energy Research Scientific Computing Center*, and 50% of one system in the *San Diego Supercomputer Center*.

most of the paper focuses on the HF-SCF method in particular, however there is a discussion extending the findings to post-HF methods as well.

Compared to home/business PC commodity codes, scientific codes are far better positioned to exploit manycore architectures because scientific code-bases have been designed with parallelism in mind for a long time. However, a problem with manycore architectures is that the available parallelism is not necessarily homogeneous; to get the most out of emerging manycore solutions, one should not simply treat a manycore architecture as a simple computational cluster on a chip. A common feature in the current generation of multicore architectures (such as the Cell Broadband Engine [2]) and next generation manycore research prototypes (such as Intel's Larrabee project [3]) is an emphasis on *single-instruction-multiple-data* (SIMD) processing. SIMD processing encompasses several specific architectures, the most famous of which is vector processing, as well as systolic arrays and associative processing. SIMD processing is capable of very high throughput for data intensive computation.

Unfortunately, the HF-SCF application is, in its canonical form, unable to fully exploit SIMD processing for the evaluation of ERIs. This “incompatibility” is due to the variation in ERI classes, and variation in ERI computation time for different classes.

In the heyday of vector processor supremacy in the super-computing space, several attempts were made to overcome this problem. Conceptually the idea is simple: although the universal set of all ERI computations for a given workload consists of varied ERI classes, it would be possible to create subsets of ERI computations out of the universal set which would only contain ERI computations of the same class. For significant HF-SCF workloads, these subsets would be sufficiently large to provide the appearance of a homogeneous computation with all ERI computations being of the same class, thereby fully utilising/saturating a SIMD processor. This basic idea was first proposed (as far as we are aware) over 2 decades ago by Guest and Wilson [4]. Unfortunately, implementing the idea has proven to be frustrating because it has been difficult to obtain adequate performance for the ERI sorting/subsetting functionality.

Envisioning increased emphasis on SIMD processing in the future, we believe it is important to revisit the issue of ERI sorting. We had previously conducted an analysis of the target workload with the goal of identifying appropriate architectural candidates [5], wherein we failed to recognise the potential of SIMD processing. We later corrected this by proposing the use of a special-purpose ERI sorting structure, with a first-order estimate of SIMD utilisation [6], a key metric which will be elaborated later.

In this paper we discuss ERI sorting in terms of viability of implementation and expected performance. We discuss the shortcomings of prior efforts and propose a few alternative approaches with greater awareness of computer architecture, with particular detail for one approach. Assuming successful implementation of ERI sorting functionality, we estimate the memory access locality of the scheme, something which has not been addressed by any proponent of ERI sorting that we are aware of. We also examine the correlation between basis set char-

acteristics and expected performance. We provide a detailed description of a software-only implementation of ERI sorting for conventional processor cores; the proposed implementation yields good performance for very large workloads. Although much better performance may be obtained with a specialised-hardware approach, the software-only approach has the advantage of being immediately implementable.

This paper is organised as follows: Section 2 summarises the most important features of the application. Section 3 reviews the relevant computer architecture literature and concepts, including an overview of the proposed ERI sorting solution. To support the practicality of the proposed hardware-based solution, Section 4 discusses some details of two specific sub-components of the system. Section 5 presents performance simulation results of SIMD processor utilisation and memory access locality. Section 6 discusses the expected impact of workload characteristics on performance. Section 7 discusses the incorporation of the integral block scheme (which re-uses certain bootstrap values for a group of integrals) with the proposed ERI sorting scheme. Section 8 discusses implementation options, with particular detail for a viable software-only solution. Finally, concluding remarks and further work proposals are presented.

2. Application characteristics

The Hartree–Fock method is a numerical procedure used to solve the Schrödinger equation for multi-electron atoms or molecules described in the fixed-nuclei approximation (i.e. the Born–Oppenheimer approximation) by the electronic molecular Hamiltonian. Because of the complexity of the differential equations for any but the smallest of molecules, the problem is usually impossible to solve analytically, and thus, the numerical technique of iteration is used. The method, referred to as the *self-consistent field* (SCF) method, is iterated until a set of convergence criteria is met.

The representation of the molecular system in the SCF numerical procedure is a set of approximate one-electron orbital functions. For atoms, these are atomic-like orbitals that reflect the electron configuration of the atom, while for a molecule, the initial approximation is a linear combination of atomic orbitals (LCAO) specially formulated in terms of a Slater determinant to satisfy the anti-symmetric nature of a molecular electron system. The basis set is the set of mathematical functions, which are finite in number and orthogonal in construction. The basis functions, in simple terms, represent probability clouds describing the distribution of the electrons. Various basis sets are used in practice, most of which involve the use of Gaussian functions. Whenever discussing the performance scaling of quantum chemistry computations, N almost always refers to the number of basis functions describing the molecule.

Canonically, the SCF method suffers $O(N^4)$ scaling—even though techniques exploiting symmetries and cut-off distances [7] may reduce this to roughly $O(N^2)$, or even $O(N)$ for certain cases [8], there is still a very large body of work that requires much larger computing capability. A dominant computational “hotspot” in the SCF procedure is the evaluation of Electron

Table 1
Basis set size and thread count for molecules in 6-31G** basis

Molecule	Basis functions	ERI threads
Water, H ₂ O	25	48×10^3
Nitrous oxide, N ₂ O	45	512×10^3
Zinc chloride, ZnCl ₂	55	1.14×10^6
Butane, C ₄ H ₁₀	110	18.3×10^6
Nicotine, C ₁₀ H ₁₄ N ₂	250	488×10^6
Caffeine, C ₈ H ₁₀ N ₄ O ₂	260	571×10^6
Carotene, C ₅₀ H ₅₅	880	75×10^9
Catenane*	2940	9.3×10^{12}

* Catenane is a “chain-like” molecular structure, which finds use in applications such as nanomachines [9].

Repulsion Integrals (ERIs) which are numerous and individually complex to compute.

Table 1 lists some example molecules and the number of basis functions that may be used to describe the molecule. In addition, the number of ERI operations required within each SCF iteration based on the canonical SCF procedure is also listed. Each of these ERI operations are concurrent and may be evaluated in parallel, and one may think of each ERI operation effectively as an independent thread. Therefore, there is massive *thread-level parallelism* (TLP) inherent in each SCF iteration for significant workloads.

The evaluation of each ERI depends on the input of four basis functions. The basis functions have a Gaussian form:

$$\eta = C(x - R_x)^{a_x}(y - R_y)^{a_y}(z - R_z)^{a_z}e^{-\zeta(r-R)^2}. \quad (1)$$

This is called a *Gaussian-type orbital* (GTO), also often referred to as a *primitive function* or *Gaussian primitive*. The characterising elements for each function η are:

- Center coordinate— $r : \{x, y, z\}$ [Real $\times 3$].
- Exponent— ζ [Real].
- Angular momenta— a_x, a_y, a_z [Int $\times 3$].
- Coefficient— C [Real].

The physical significance of the angular momenta is related to the electron configuration over “spdf. . .” shells.

Typically, a set of GTOs are combined linearly, or *contracted*, to form a single *contracted-GTO* (CGTO). The level of contraction K specifies the number of primitive functions collected into the single contracted function—higher K generally leads to higher accuracy. The CGTOs χ are the basis functions. Each CGTO consists of a collection of GTOs, however all the GTOs share a common center coordinate and common angular momenta. A CGTO may be thought of as a container with the following elements:

- Center coordinate— $r : \{x, y, z\}$.
- Angular momenta— n_x, n_y, n_z .
- K *partial* GTOs (pGTO), which only consist of a coefficient and an exponent.

The ERI computation involves a quartet (4-tuple) of CGTO:

$$\langle ij|kl \rangle = \int dx_1 dx_2 \chi_i^*(x_1) \chi_j^*(x_2) \frac{1}{r_{12}} \chi_k(x_1) \chi_l(x_2). \quad (2)$$

Of all the various data elements characterising a CGTO, the most important in the context of the present work are the level of contraction and the angular momenta. This is because the level of contraction K and angular momenta $[a_x, a_y, a_z]$ of each of the four basis functions determine the execution procedure (formally, the control-flow) of the ERI routine. The angular momenta terms may dictate the choice of ERI routine (e.g., Rys quadrature [10–12] for high angular momenta basis functions vs. the Pople–Hehre “axis-switching” method (PH) [13] for low angular momenta basis functions). However, even if the same algorithm is used for all integrals (e.g., the PRISM algorithm [14]), the sequence of operations performed will vary based on angular momenta and level of contraction of the four basis functions constituent in the ERI quartet.

Although level of contraction (in addition to angular momenta) dictates ERI procedure control-flow, there is a simple approach to compute integrals which essentially removes the control-flow dependence on level of contraction. One simply “unrolls” the contractions by computing integrals for quartets of primitive basis functions and subsequently contracting each “primitive integral”. This naive approach is computationally expensive when compared to algorithms which are optimised for highly contracted workloads by effectively performing the contractions “early”, negating the need to compute a large number of primitive integrals. However, as will be shown in Section 6, the naive approach has a special significance in the context of SIMD processing. With that in mind, we will initially ignore level of contraction in the evaluation of the proposed ERI sorting scheme.² Specific treatment of contractions will be deferred to Section 6.

An important optimisation is to exploit the permutational symmetry inherent in the computation of ERIs. Each ERI may be characterised as a quartet of input functions, $\langle ij|kl \rangle$. Due to the mathematical foundation underlying the SCF algorithm, the following equalities hold true:

$$\begin{aligned} \langle ij|kl \rangle &= \langle ji|kl \rangle \\ &= \langle ij|lk \rangle = \langle ji|lk \rangle \\ &= \langle kl|ij \rangle = \langle lk|ij \rangle \\ &= \langle kl|ji \rangle = \langle lk|ji \rangle. \end{aligned} \quad (3)$$

By reordering the input functions for each ERI, subject to the constraints described in Eq. (3), there would exist a larger number of threads with matching control-flow, thus significantly improving the thread-match hit rate and consequently SIMD processor utilisation. This idea is illustrated in Fig. 1.

The usefulness of our solution would be limited if electron correlation computations were not also accelerated. We have focused primarily on the Hartree–Fock computation because it is a necessary first step, however it would not be difficult to expand the scope of the solution to also serve post-HF computations. This is because even with post-HF computations, ERI

² The implicit assumption here is that all basis functions have the same level of contraction, which although unconventional, is not an entirely contrived assumption.

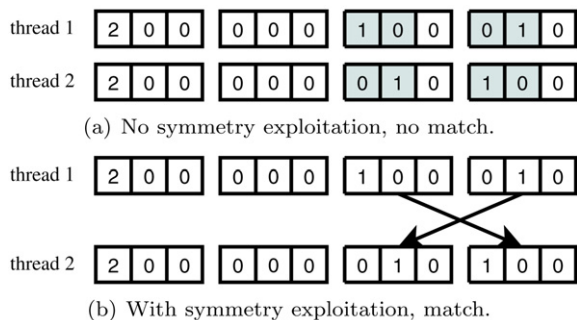


Fig. 1. Thread matching with symmetry.

are a significant hot-spot, and in general there are computational similarities between HF and post-HF methods.

Let us consider the four-index transformation, an important component of electron correlation computations, which scales $O(N^5)$ with basis size [15]. The four-index transform operation requires four terms with similar form to the following:

$$[iq|rs] = \sum_{p=1}^N T_{ip}[pq|rs], \quad i, q, r, s = 1, \dots, N. \quad (4)$$

Once again the heart of the problem is generating, evaluating, and contracting integrals. The requirements of this stage are therefore similar to the requirements of HF.

We will conduct more rigorous analysis of different post-HF methods in a later communication, but an important point has already been alluded to here: computation of ERIs are a significant problem, but “collecting” ERIs into data structures such as a Fock-matrix is also a significant problem. The latter operation is a form of tensor contraction, which are inherently straight-forward to implement on SIMD architectures.

More generally, beyond ERI, dense linear algebra operations are also a dominant computational hot-spot, particularly in post-HF methods. Dense linear algebra is another application class which is known to be inherently straight-forward to implement on SIMD architectures. This clearly provides even more incentive to engage SIMD processing for computation of ERIs as well, thus “SIMD-ifying” a great proportion of the entire application.

3. Computer architecture overview

SIMD processing is highly efficient because it maximises the arithmetic intensity of the underlying hardware. Any hardware system is constrained in scale by logic area and power/cooling limitations. Generally speaking, in order to achieve high computational throughput one aims to instantiate as many utilisable arithmetic units as possible within given resource limits, i.e. to maximise the floating point operations per second (FLOPS) metric for given resource constraints. With conventional *single instruction single data* (SISD) architectures such as commodity general-purpose RISC processors, significant hardware resources are dedicated to control functions. Generally speaking, SIMD processors are geared towards greater arithmetic throughput by reducing the emphasis on control-oriented resources. Conceptually, SISD requires that for V data processing

capacity one pays V amount of control “cost”, whereas SIMD requires that for V data processing capacity one pays only a single unit of control “cost”. However, only applications exhibiting *data-level parallelism* (DLP) may fully exploit SIMD processing (i.e. utilise all the instantiated arithmetic units). Examples of SIMD processors include vector processors, general-purpose graphics processors (GPUs) [16,17], streaming processors [18], and application specific processors such as for homology search in bioinformatics [19] and molecular dynamics [20,21].

The simplest possible manifestation of the emerging many-core paradigm in computer architecture [22] is that of a computational cluster on a chip—i.e. a significant number of homogeneous conventional processor cores interconnected with a network of some form. One may be tempted to simply await the emergence of such solutions to negate the need for less generally applicable approaches such as SIMD, especially for applications that cannot immediately exploit SIMD, such as HF-SCF. Unfortunately, the many-core design road-map is not without difficulty; issues such as inter-core communications requirements, external memory bandwidth contention and overall system reliability [22,23] will make linear speedup very hard to achieve for any application. With these emerging designs the chief resource constraint is electrical power and heat dissipation [22,24]. SIMD processing, especially when implemented as vector processing, is known to be highly power efficient [25]. It is for this reason that one current generation high-performance many-core processor, the Cell Broadband Engine [2], relies heavily on SIMD processing nodes to achieve high computational throughput (although we view the Cell BE’s SIMD pedigree to be somewhat diminished due to the fact that the data stream width is limited to 4 words—in contrast the Cray-1 had 64 word vectors). It is also for this reason that GPUs have traditionally had relaxed cooling constraints when compared to CPUs of roughly equal transistor count and lesser processing capability.

The excellent performance of GPUs (in terms of FLOPS/Watt and FLOPS/\$) has encouraged their use in non-graphical applications, such as dense linear algebra [26] and recently even in quantum Monte Carlo simulations [27]. While applying the GPU execution model (i.e. stream processing, a form of SIMD) to the quantum Monte Carlo application was by no means trivial, it was made immediately possible by the fact that the application bottleneck inherently exhibits very large data-level parallelism (DLP). This is not the case with the Hartree–Fock Self Consistent Field application.

3.1. Previous vectorisation efforts

A broad review of high-performance computer architectures for the target computation would certainly include networks of computers operating in parallel (such as the comprehensive review provided by Harrison [28]), however the focus of this paper is restricted to the architecture of a single processor, and specifically we consider only SIMD processing.

Vectorisation (and therefore by implication “SIMD-ification”) of the workload has been attempted by several groups. Saunders and Guest [29] considered vectorisation of the proce-

ture on the classic vector architecture, the Cray-1. Worthwhile gains were found in the vectorisation of matrix operations. Specifically for the vectorisation of ERI evaluation, the authors propose that the entire set of integrals be partitioned into subsets of integrals of identical type, e.g., one set of [PP/PP], another set of [PP/SP], etc. The complete set of ERIs required to construct the Fock matrix are sorted into sets of ERIs with identical computational procedure. Unfortunately, details of the implementation mechanics of the ERI sorting are omitted, and consequently so is a comprehensive assessment of the performance of the ERI sorting operation, although the authors allude to the significant memory requirements of such a scheme.

In fact, the idea of sorting ERIs in order to achieve high vector processor utilisation had been proposed at least as far back as 1981 by Guest and Wilson [4]. The specific proposal is to iterate over all possible thread classes at the outer loop of the thread generation code (i.e. the 4 nested loops that generate ERI quartets). This approach is highly inefficient, as there is a lot of sparsity in the distribution of thread classes dependant on workload characteristics, however it serves as a starting point for the idea.

The basic idea was improved upon in 1989 by Ernenwein et al. [30], and this is, as far as we are aware, the most recent attempt on this idea to date. The cornerstone of the improvement is liberal use of buffering, dynamic storage allocation, and pointer-chasing, coupled with the spawning of tasks (concurrent processes) that are assigned specific ERI classes to manage. The idea is to allow ERI quartets to be generated and stored in main memory, and to be managed/scheduled for execution by these concurrent processes. The use of dynamic storage allocation and pointers allow the efficient use of memory in light of unpredictable ERI class population distributions. The use of concurrent processes provides a convenient means for the computation of different ERI classes to proceed asynchronously, also compensating for the lack of predictability in ERI class generation. Code profiling had indicated that implementation of this scheme on a Cray-2 results in the bulk of computation time being consumed by buffer management, and note that since the late 80s the disparity between microprocessor speed and memory latency has in fact increased dramatically, which would worsen the performance of the scheme.

3.2. ERIC special-purpose processor

As far as we are aware, there has only been one prior attempt at a special-purpose processor specifically targeting ERIs and the HF-SCF algorithm. The strategy underlying the ERI Computer (ERIC) [31–33] was to exploit fine-grained instruction-level parallelism (ILP) inherent in the computation. To achieve this ends, a VLIW core was proposed, with some novel components such as specialised arithmetic units. The system was implemented using VLSI technology, and the performance of the completed system was found to be disappointing [33]. A strategy based on the exploitation of ILP would be limited in the amount of parallelism available, as the amount of parallelism that may be observed and leveraged is limited to that within a single thread. Our approach is fundamentally different because

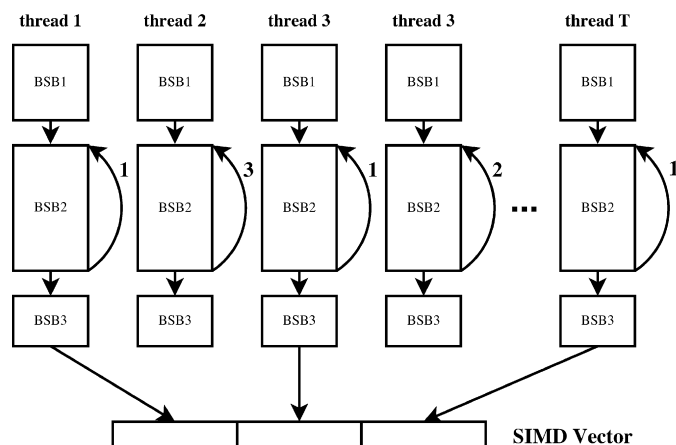


Fig. 2. Converting TLP to DLP. Each basic scheduling block (BSB) corresponds to a contiguous chunk of instructions. The BSB2 block has a cyclic arc with an associated variable integer—the arc corresponds to a loop identifier and the associated integer specifies the loop count; e.g., in thread 2 BSB2 is executed 3 times, whereas in thread 3 BSB2 is executed once. The difference in loop count is effectively a difference in control-flow. In this example the loop count of BSB2 is the only source of control-flow variation between different threads, therefore to identify threads with identical control-flow one merely has to check the loop count specified for BSB2. Matching threads can then be processed with exactly the same sequence of instructions, and may therefore be issued for SIMD processing.

it is based on the exploitation of TLP (albeit the conversion of TLP to DLP), which yields massive amounts of parallelism, and furthermore the amount of parallelism available increases as problem size increases, a phenomenon which is not characteristic of ILP in general and for this application in particular. A more comprehensive review of the ERIC, as well as a review of other computer architectures applied to the target computation, was previously conducted [5].

3.3. Functional requirements of ERI sorting

The basic idea behind our proposal is to match ERI tasks—effectively independent threads of control—which have identical control-flow, and subsequently re-order and issue these threads to a SIMD processor as a data-parallel stream, essentially converting TLP to DLP. This idea is illustrated in Fig. 2.

The functionality required for ERI thread sorting is memory intensive. Subsets of threads with identical control-flow have to be formed from a universal set of threads with varied control-flow. Creating these subsets requires either

- (i) Thread-generation-time pre-sorting of integrals (early/eager subset creation).
- (ii) Very rapid SIMD vector issue-time thread matching (late/late subset creation).

ERI sorting through thread pre-sorting requires the instantiation of a number of buffers, each buffer corresponding to a subset of ERI threads with identical control-flow. Each buffer is indexed with an identifier corresponding to a unique thread class, which is a tuple containing the levels of contraction and angular momenta terms for the four constituent basis func-

tions in the ERI operation. Threads are added to the appropriate buffers until the population of a buffer exceeds a certain threshold and it is streamed to the SIMD processor as a vector of data which are to be processed with the exact same procedure, i.e. with identical control-flow. This process is repeated until there are no new threads being generated. The challenges associated with this approach are (1) the *management* of individual buffers, and (2) the *indexing* of individual buffers.

The first problem—the management of individual buffers—may be addressed through dynamic memory management: when a new thread is to be added to a full buffer, the buffer is expanded through a memory allocation operation to accommodate the new addition. Memory allocation is an expensive operation, so this approach is unlikely to be capable of high throughput, as was discovered by Ernenwein et al. [30]. An alternative approach is to deliberately over-provision buffer space a priori so that no dynamic memory management is required. This approach is space expensive, but given long-standing trends in logic and memory device density growth, which corresponds to memory capacity growth, and given the consensus that these trends are unlikely to abate in the near future [22,23], memory storage capacity may be considered a relatively inexpensive resource. Furthermore, since the buffer is to be streamed out as a large block of contiguous data, the issue of memory latency is less critical since the data access pattern of issuing an entire buffer of ERI threads is perfectly sequential, exhibiting large spatial locality, which is amenable to pre-fetching, memory banking, etc.

The second problem—the random indexing/selecting of individual buffers—is more difficult to resolve. One extreme possibility is to utilise a perfect hash function, generated with all possible ERI thread class identifiers. Theoretically this would yield a solution capable of identifying the correct buffer for a given ERI thread in $O(1)$ time, however the size of the required hashing table would be massive, given the very large (albeit bounded) range of ERI thread classes, in the order of 10^8 for $0 < K < 7$ and angular momenta terms $a_x + a_y + a_z < 3$; once again, the large storage capacity required is not of immediate concern, however in this case one does not have the advantage of predictable memory access patterns,³ and therefore memory latency is a concern. Alternatively one may instantiate a smaller look-up table which maps ERI thread class identifiers to thread buffers—while this approach may result in a *identifier-to-buffer* mapping data structure that fits entirely within a small high-speed data cache, efficient indexing operations may be elusive, the worst-case requiring an exhaustive search.

In general, ERI thread sorting through thread pre-sorting seems attractive because it appears amenable to implementation on existing systems with conventional/entrenched processing and memory structures, and would therefore be widely applicable. However, it is challenging because of the limitations of

conventional systems vis-à-vis the implementation of search operations, including the constraints of conventional memory subsystems.

An alternative approach is to consider vector issue-time matching, which we call the *associative-search* approach. This approach requires only a single buffer containing a large population of threads, or a thread-window W_{threads} . Unlike with the pre-sorting approach, with this approach there is no redundant memory allocation or expensive dynamic memory management. An ERI thread class identifier is selected according to some criteria and the population of threads are searched for all threads that match the given identifier. The matching threads are then issued for SIMD processing. Unlike with the pre-sorting approach, with this approach there is no requirement of buffer indexing, since there are no individual distinct buffers corresponding to distinct ERI thread classes. However, this approach appears extremely expensive since each vector issue requires $O(W_{\text{threads}})$ search operations, and the performance of such a scheme on a conventional system would be completely inadequate. However, the approach would be viable with non-conventional hardware. As each search operation is independent of the others, it is possible to perform W_{threads} search operations in $O(1)$ time. This may be achieved with a special memory structure called a *content-addressable memory*, details of which are presented in Section 3.4.

A high performance implementation based solely on software (for which the most suitable strategy would be early/eager pre-sorting) would be the most beneficial as it would be easily adopted. We attempt implementation of such a solution, and find that good performance may be achieved for very large workloads—the implementation is discussed in detail in Section 8.1. However, much greater gains may be obtained through the use of specialised ERI sorting hardware, and so a greater proportion of this paper will be spent discussing a hardware-based associative-search (issue-time matching) approach. It will be shown that the strategy is viable, and that SIMD processing of ERI threads is a worthwhile field of further work, likely to yield breakthrough performance improvements and greater Hartree–Fock SCF computing capability.

3.4. Proposed hardware architecture

The proposed architecture, illustrated in Fig. 3, consists of three major conceptual blocks:

- (i) the **ERI Generator**.
- (ii) the **ERI Sorter**.
- (iii) the **ERI Cruncher** (i.e. the SIMD processor).

This paper is only concerned with the details of 1 and 2: details of 3 will be left for a future communication. The role of the ERI Generator is to generate ERI “threads” by constructing independent quartets of basis functions, and functionality such as molecular symmetry exploitation and integral pre-screening may be built into the ERI Generator which would result in the elimination of insignificant/redundant integrals—consequently the ERI Generator would generate fewer ERI threads. These threads

³ It remains to be seen if practical workloads exhibit exploitable patterns in ERI class identifier indexing—if such patterns do exist it may be possible to predict ERI class identifier indexing patterns and thereby reduce the severity of the memory wall problem for this particular functionality through pre-fetching, cache-line biasing, and other techniques.

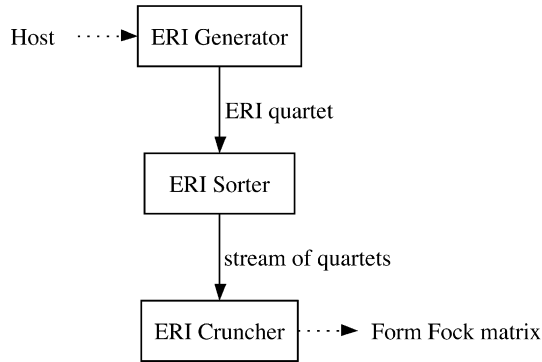


Fig. 3. Architecture overview.

are then sent to the ERI Sorter, wherein they are matched for identical control-flow. Once a number of matched threads have been identified, they are issued to the ERI Cruncher for efficient SIMD processing.

The ERI Sorter is arguably the most interesting component in our proposal. The role of this unit is to examine a population of threads and identify a sufficiently large number of threads that have identical control-flow to be efficiently streamed to a SIMD processor. A distinct challenge is the task of searching a significant population of threads rapidly in order to allow high computation throughput. As was previously mentioned, in this paper we discuss two approaches to this problem: a widely applicable “software-only” approach, and a significantly higher performance hardware implementation. Details of the software-only approach will be left for Section 8.1; unless stated otherwise, the remainder of this paper assumes the hardware implementation which is about to be discussed, although in most cases the arguments presented may apply to any ERI Sorter implementation.

The key concept underlying the proposed hardware solution is associative searching, or content addressing. This functionality is present in the ERI Sorter, in the form of a *content-addressable memory*. Our proposal to construct streams of class-matched ERI quartets with a content-addressable memory directly addresses the problem of efficiently and rapidly sorting ERIs into subsets, providing a high-performance highly parallel dedicated hardware alternative to dynamic memory management and pointer-chasing.

A content-addressable memory (CAM)—illustrated logically in Fig. 4—is a special memory structure capable of high speed search operations, often used in packet-switched Internet equipment for operations like routing table look-up [34]. When attempting to extract information from a standard memory structure, one supplies the memory with an address and the memory returns the information at the corresponding address. In contrast, when using a CAM one supplies the CAM with a (typically short) data word, called the *searchword*. Each individual memory location within the CAM is called a *dataline*. The CAM then searches a portion of all datalines (this portion is called the *matchword*) to locate all locations with the corresponding searchword, and consequently return a piece of data (called the *returnword*) associated with the corresponding location. Neglecting technology capacity limitations, logically a

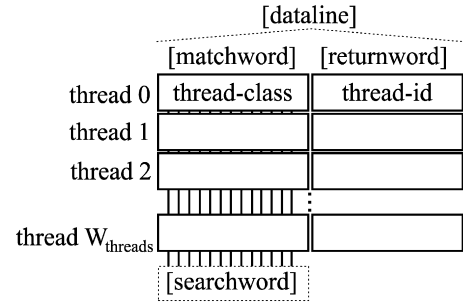


Fig. 4. Logical view of content-addressable memory structure.

CAM can conduct a search over its entire memory space in a single clock-cycle. In practice, the size of CAM structures is limited by technology capabilities—in order to support single-clock-cycle search capability very large datapaths have to be allocated to carry the searchword across all memory locations in the CAM, and driving such datapaths requires high electrical power. Having said that, there are CAMs available with significant capacity—18 Mb CAMs have been commercially available for some time [34].

The CAM structure is the foundation of the ERI Sorter. We propose to utilise a CAM by populating each dataline with the information characterising each ERI thread. The matchword of each dataline is populated with a token corresponding to the control-flow of the ERI, which we will call the *thread-class*. The thread-class may be a simple concatenation of all the components of the basis function inputs that influence ERI control-flow (i.e. a control tuple), or, as we shall see in Section 4.2, more storage-efficient approaches by means of a look-up table. The returnword of each dataline is populated with the four basis functions making up the quartet to the ERI operation and any other information that may be deemed appropriate—we will call this information the *thread-id*. One may then search for a particular thread-class by loading the desired thread-class as the searchword, which is then compared against all matchwords in the CAM, and the CAM structure returns *all* ERI thread-ids with the same thread-class. This stream of thread-ids is then fed into the ERI Cruncher.

4. System-level details

Fig. 5 depicts the block diagram of the proposed system. All the main components are illustrated. Provisional bandwidths have been allocated to sustain a throughput of 1 thread per cycle through the ERI Sorter. Only major data buses have been indicated—due to space constraints control buses have been omitted.

Of all the components, the two least intuitive are probably the tuple sorting in the ERI Generator and the Searchword LUT in the ERI Sorter. Both these architectural improvements are designed to ease the burden placed upon the CAM structure, specifically the searchword bus [34], and dramatically improve the practicality of the design. We now elaborate on these two aspects specifically.

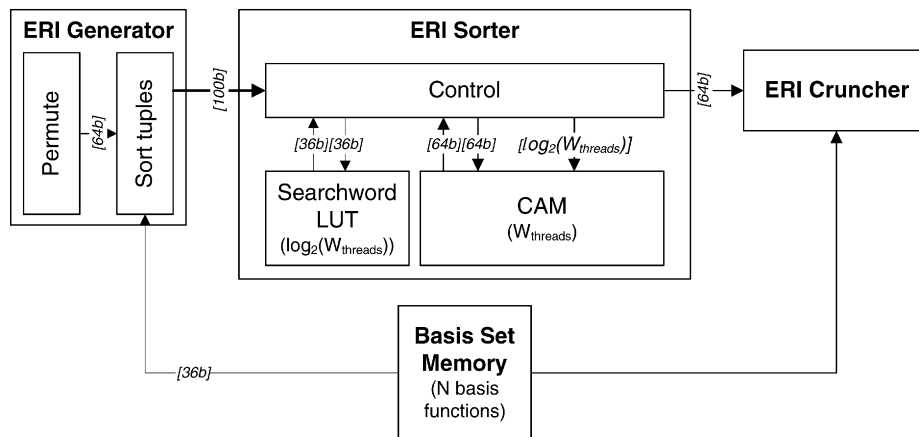


Fig. 5. System block diagram. Values in “(…)” parentheses within blocks indicate storage capacity. Values in “[...]” parentheses on arrows indicate provisional bit-width allocation. Only major data busses are indicated; control busses have been omitted for clarity.

4.1. Tuple sorting

The process of generating each thread is simply a 4-deep nested loop, according to the canonical ERI computation sequence [5]. However, an added complication to ERI thread generation is the exploitation of ERI symmetry, as described with Eq. (3). This may seem out of place, since symmetry is related to increasing the thread match-rate, and therefore it would apparently be a task for the ERI Sorter, however we propose to handle symmetry implicitly before issuing threads to the ERI Sorter by sorting the basis functions within each thread, subject to the symmetry rules stipulated by Eq. (3), in a manner that moves the larger angular momenta terms to the left of the quartet. In this manner, the threads are essentially “symmetrised” before being issued to the ERI Sorter. This greatly reduces the expense of the ERI Sorter by eliminating the need for large and complicated searchline layout [6,34]. We have previously demonstrated the value of symmetry exploitation for the purpose of increasing the match-rate for any given workload [6]. In this section we describe a practical “tuple pre-sorting” mechanism to implement this symmetry exploitation functionality.

Operations required to perform this “tuple pre-sorting” include bus concatenation and integer comparison—provisioning highly pipelined high throughput hardware capable of generating 1 thread per cycle is therefore practical. An alternative would be to do so in software with a simple processor core, which would consume many more cycles, however we may provision multiple ERI Generator cores, with a shared bus to the ERI Sorter, with the goal of issuing 1 thread per cycle to the ERI Sorter. These are the two extremes in the design-space.

4.2. Searchword LUT

We previously adopted a constraint on the maximum possible angular momenta term to be 7 [5,6], which means 3 bits are required for each angular momenta term, and as there are 3 angular momenta terms in each basis function and there are 4 basis functions in an ERI quartet, $3 \times 3 \times 4 = 36$ bits are required to completely characterise the control-flow of any ERI computa-

tion.⁴ Thus, we previously specified a 36 bit searchword and matchwords in the CAM [6] which, though practical, is expensive. By observing some characteristic workloads at run-time, we observed significant sparsity in the range of actual angular momenta combinations present in the CAM at any given cycle, especially when permutational symmetry is exploited.

Therefore, by employing a reasonably sized look-up table, we can significantly reduce the required size of the CAM searchword and matchwords to $\ll 36$ bits—a word of size $\log_2(W_{\text{threads}})$ would be sufficient, where W_{threads} is the thread window, i.e. the CAM’s capacity in terms of number of threads. For practical workloads an even smaller word would suffice because there would typically be multiple threads in the CAM with identical thread-class. Assuming the thread window $W_{\text{threads}} = 16$ kilothreads, the required searchword size for some actual workloads (Zinc Chloride, Butane, and Nicotine, with 6-31G** basis) is just 11 bits; less than $\log_2(16k) = 14$ bits, and much less than 36 bits.

5. Performance

In this section we assess some important aspects of the system performance of our proposal. Although the data presented was obtained thru simulation of the hardware-based solution, the arguments and conclusions drawn here are also generally applicable to the software-based solution. We specifically focus on two key aspects:

- (i) SIMD unit utilisation—Section 5.1.
- (ii) Memory access locality—Section 5.2.

SIMD unit utilisation is a crucial metric because it reflects how amenable the computation can be made to SIMD processing. A low utilisation score indicates that there is little to be gained through SIMD processing, and that significant resources

⁴ If level of contraction K was also to be included, then allowing a range of 3 bits for K would increase the size of the control tuple to $4 \times 3 + 36 = 48$ bits; however, for the rest of the discussion in this section, we will continue to neglect K .

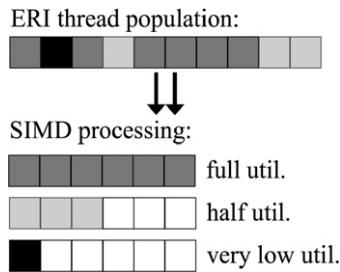


Fig. 6. ERI threads and SIMD utilisation. The ERI thread population contains 10 threads of three classes; 6 class A threads, 3 class B threads, and 1 class C thread. ERI threads of the same class may be computed in SIMD fashion. With a SIMD width of 6, a maximum of 6 ERIs may be computed simultaneously. If fewer than 6 ERIs are computed simultaneously, some processing resources are left idle/unutilised, and are basically wasted. In the example above, an aggregate efficiency of 55.56% is achieved.

are idling, whereas a high utilisation score indicates that individual processing elements in the SIMD processor are being well utilised. Fig. 6 illustrates the relationship between ERI thread population characteristics and SIMD utilisation.

An important phenomena in modern computer architectures is the “memory wall”. Over the past two decades it has been observed that processor speeds have advanced at a much faster pace than memory speeds—the processor has “left the memory behind”. Consequently, for many workloads, the processor is stalled on memory, i.e. the processor is waiting for some data to be fetched from memory which is needed to perform some operation (e.g., adding two values currently stored in memory). During this time, the processor is idle.

There are several approaches to mitigate the memory wall problem. The most well-known approach is the use of fast (but relatively small) high-speed memories in a hierarchical fashion as intermediaries between the processor and main memory. These memories are called caches. The first time a piece of data is accessed from main memory a large performance penalty is incurred (in the order of 100s of CPU cycles), but after the first access the data resides in cache, and subsequent accesses to the same data incurs a much smaller performance penalty (in the order of 10s of idle CPU cycles). Clearly, caches are only effective in cases where repeated accesses are made to the same data — a characteristic termed *temporal locality*. For example, consider the following sequence representing the sequence of memory addresses indexed: [43, 6, 11, 6, 5, 90, 11, 6, 11, 6]. Location 6 is accessed repeatedly, however of the four accesses to location 6 only the first access incurs the full expense of fetching data from main memory—the remaining 3 accesses are cached. Similarly, the second and third accesses to location 11 are also cached. In fact, for this particular sequence, of the 10 memory accesses, 5 are cached, so there is significant temporal locality exhibited by the program, and some caching would be beneficial.

A complementary and common enhancement is memory pre-fetching, which basically works by predicting future accesses based on the access history. A simple yet extremely effective (and common) strategy is to assume that when a piece of data is accessed, adjacent pieces of data will soon be required as well. Therefore, when an access is made to a particular memory

location, adjacent memory locations are moved into the cache in anticipation of being accessed in the near future. This approach is effective for workloads that require contiguous access to memory locations that are adjacent to (or at the very least in the vicinity of) each other—a characteristic termed *spatial locality*. For example, consider the following sequence representing the sequence of memory addresses indexed: [92, 0, 1, 2, 3, 78, 45, 18, 15, 16, 17, 18]. The two access sequences {0, 1, 2, 3} and {18, 15, 16, 17, 18} exhibit spatial locality—in the latter case although the initial part of the sequence 18–15 has a distance of 3 (and is therefore not adjacent) this is still considered sufficiently close by most standards of spatial locality. A pre-fetching scheme that fetches the 4 nearest neighbours of each access would substantially improve the performance of the system—in fact in this example of the 12 memory accesses made, only five accesses would incur the full expense of fetching data from main memory—the remaining accesses are *pre-cached*.

Memory access locality is a particularly important consideration because our scheme relies on reordering ERI operations to maximise SIMD unit utilisation, which may result in the “randomisation” of memory access patterns. Random accesses are problematic in general because randomisation removes locality, thereby rendering the solution vulnerable to the memory wall phenomenon. Therefore, we empirically examine the locality of memory access patterns with the proposed ERI reordering scheme.

5.1. SIMD unit utilisation

The methodology employed to obtain the performance plots in this section is as follows. The ERI Sorter streams a vector of threads to the ERI Cruncher, where the maximum stream width is nominally set to $V = 64$. If a full stream is issued, then a utilisation score of 1 is obtained. However, if the CAM is full and there are fewer than V threads with the same thread-class for all thread-classes present in the CAM, then a partial stream has to be issued to make room for new threads. In this case, a utilisation penalty is incurred. The utilisation scores are then summed up and divided by the total number of SIMD vector packing operations performed, to obtain the final utilisation score.

The results obtained with the above methodology are as follows. Fig. 7 depicts the SIMD unit utilisation for Zinc Chloride, Butane, and Nicotine workloads, for different CAM sizes. As expected, thread population size W_{threads} has a significant impact on SIMD unit utilisation. This is because a larger thread window allows a greater number of threads to be present for matching, and consequently a higher probability of finding a number of threads that meet the vector width V .

Fig. 8 depicts the SIMD unit utilisation rate for several workloads, with vector width $V = 64$ and thread window $W_{\text{threads}} = 16$ kilothreads. Based on the results in Fig. 8, it is possible to issue a rough conjecture that in general for a given W_{threads} and V , SIMD unit utilisation increases with the size of the workload basis set. At this time we cannot issue a more precise conjecture because the matching rate (and consequently vector utilisation)

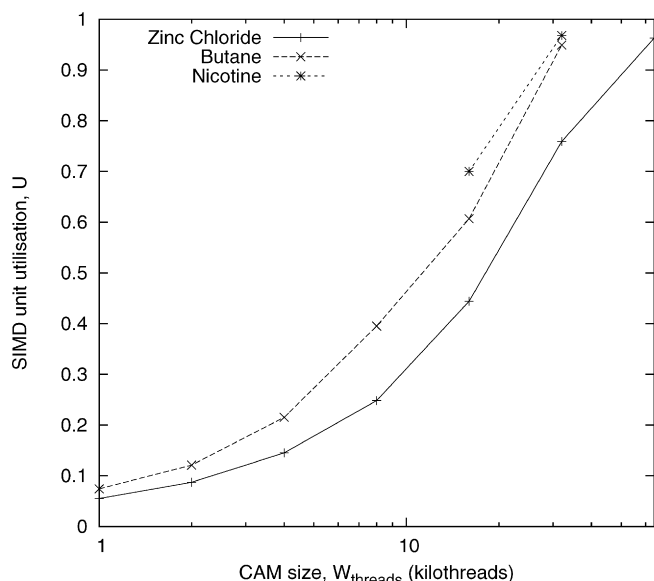


Fig. 7. Impact of W_{threads} on SIMD unit utilisation. All workloads in 6-31G** basis.

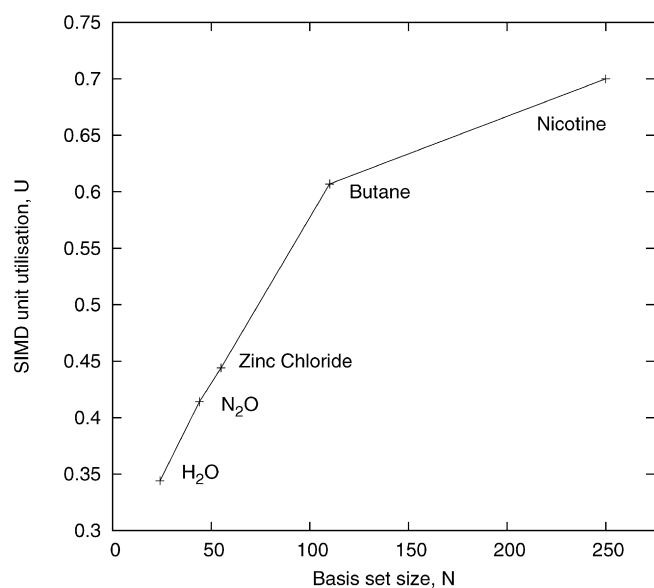


Fig. 8. SIMD unit utilisation for candidate workloads. All workloads in 6-31G** basis. $V = 64$ and $W_{\text{threads}} = 16k$.

also depends on the *nature* of the workload,⁵ a subject that we have not thoroughly studied yet.

We have previously extolled the virtues of symmetry exploitation [6]. To reiterate, symmetry exploitation does significantly improve the vector unit utilisation of the system: for the Zinc Chloride candidate workload, with $W_{\text{threads}} = 4$ kilothreads, vector unit utilisation without symmetry exploitation is 4.7%. With symmetry exploitation, the utilisation improves dramatically to 14.5%. An additional advantage is observed in

the present work: tangible improvements are also observed for required searchword bit width (as discussed in Section 4.2): without symmetry 12 bits are required, with symmetry 11 bits are required. This is because symmetry exploitation reduces the diversity of threads within a workload, therefore increasing the likelihood of finding matching threads (thus yielding better vector utilisation), while also reducing the number of unique thread-classes (thus yielding searchword width savings, resulting in cheaper and more practical implementation).

5.2. Basis set access locality

Rescheduling each ERI computation may have the effect of diminishing the locality of basis function accesses. The precise impact ERI sorting on memory access locality may be ascertained through memory address tracing. The methodology employed to obtain the performance plots in this section is as follows. Basis function access traces are fed into a locality measurement kernel, loosely based on metrics proposed by Weinberg et al. [35]. A more sophisticated and realistic cache simulation was ruled out because the results of such an analysis would be influenced by architectural nuances of the memory system (e.g., set associativity, bank configuration, etc.), and the results would not be a pure reflection of access locality.

The approach used to quantify spatial locality is particularly similar to the approach proposed by Weinberg et al. [35]; for each access, one looks for the minimum stride distance between the current access and $W_{\text{look-back}}$ previous accesses, i.e. a “look-back window”—Weinberg et al. set $W_{\text{look-back}} = 32$ for pragmatic data collection reasons [35]; we cannot think of a clearly better value, and so shall select $W_{\text{look-back}} = 32$ as well. Subsequently, the stride population is summed up to produce a normalised distribution of strides—a plurality of smaller strides (excluding stride-0, which as described by Weinberg et al. [35] reflects temporal locality) indicate a high degree of spatial locality.

One may simply assume that strides of length 0 indicate temporal locality. Temporal locality is difficult to quantify without assuming a reuse distance, in essence the “size” of the “cache”; we effectively set reuse distance to be the same as our “look-back window”, $W_{\text{look-back}}$.

With the above methodology, we quantitatively compare the access locality of the canonical ERI computation sequence [5] with the locality exhibited with reordered threads. Locality simulation for our largest candidate workload, Nicotine, has been omitted due to the large computational expense of obtaining the results. Locality results for this workload will be published at a later date.

Figs. 9 and 10 illustrate the spatial locality of different workloads, with canonical computation sequence and re-ordered computation. Strides exceeding 32 have been lumped together with stride-32 for ease of illustration. Note that in Fig. 10 the canonical scheme exhibits a large number of accesses with stride distance exceeding 32, while the thread re-ordering scheme has far fewer accesses exceeding 32. A larger right-skew reflects better spatial locality. In both test cases, the results indicate the thread-reordering increases the population of ac-

⁵ For example, organic chemistry workloads exhibit different basis set characteristics than chemistry involving transition metals, especially differences in distribution of angular momenta, which would have a direct impact on the performance of any ERI sorting scheme.

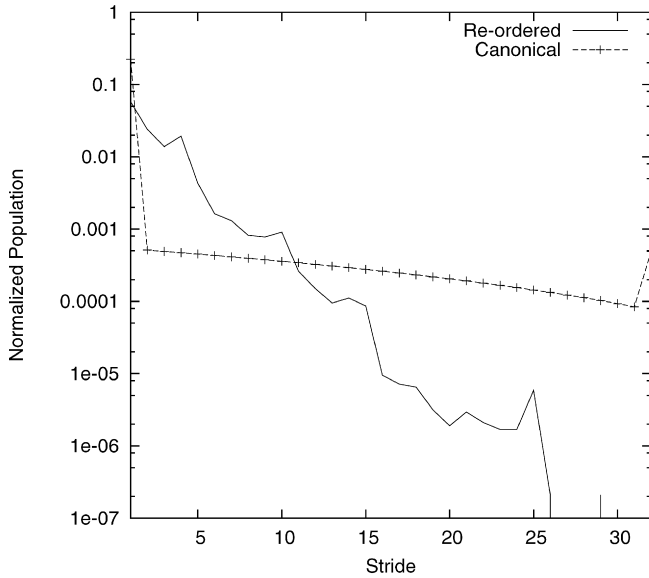


Fig. 9. Spatial locality for Zinc Chloride (6-31G** basis) workload.

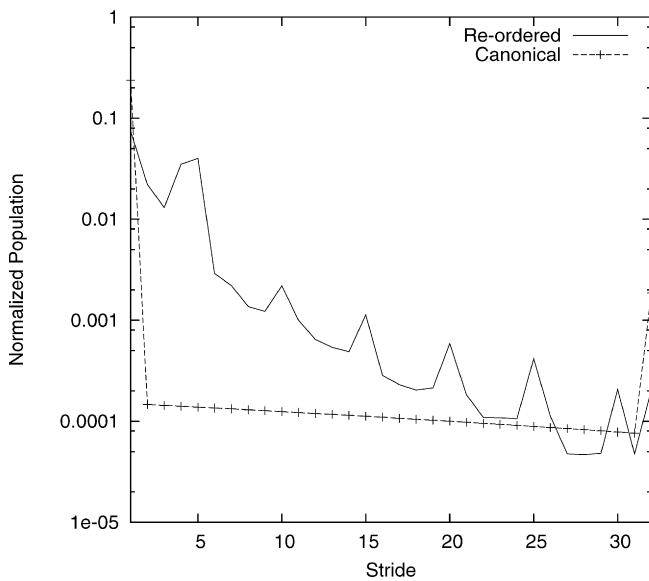


Fig. 10. Spatial locality for Butane (6-31G** basis) workload.

cesses that have a small stride distance, though the reduction in stride-1 references causes some concern.

Fig. 11 depicts the temporal locality exhibited by the two candidate workloads. The results here conclusively favour thread-reordering. Significant improvements in stride-0 references (implying the existence of the required data within a cache of $W_{\text{look-back}} = 32$ previous references) were noted for both workloads. The results are more impressive for the smaller workload—this merely illustrates the fact that a greater portion of the entire basis set could fit within the implied cache of size $W_{\text{look-back}} = 32$ words. The ERI sorting/re-ordering scheme does not deliberately attempt to improve temporal locality. Based on observation of just two simulations, it is not possible to declare that thread re-ordering does implicitly perform cache-friendly loop transformations such as strip-mining,

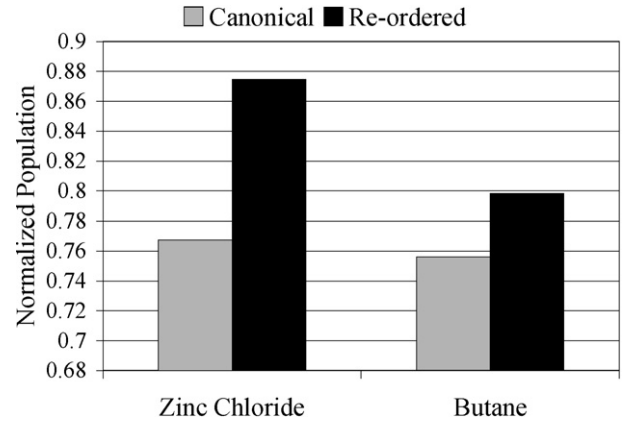


Fig. 11. Temporal locality for candidate workloads. All workloads in 6-31G** basis.

however these results certainly allude to the possibility of such a finding, and serve to motivate further investigation.

These results indicate that the architecture as it is exhibits reasonable data access patterns for the HF-SCF application, and that provisioning a memory interface capable of sustaining high vector processor throughput would be possible. From the temporal locality plot (Fig. 11), it is apparent that thread-matching is sequencing ERI computation in a manner that is cache-friendly. Unfortunately, a correspondingly clear improvement in spatial locality is harder to declare. The results indicate that more thoughtful vector issue policies may prove beneficial. We had previously envisioned the need for a smart vector issue scheme to improve spatial locality at run-time [6], i.e. a system that would issue a stream of threads not only based on a criteria for number of matching threads, but also a criteria for number of threads with constituent basis functions that are “near each other”. The analysis presented here has indicated that such a solution may be worthwhile, subject to the results of locality analysis with larger workloads.

6. Impact of workload characteristics

To reiterate, the control-flow of each ERI thread is determined by:

- (i) The level of contraction K of each of the four constituent basis functions $\langle i, j | k, l \rangle$.
- (ii) The angular momenta (a_x, a_y, a_z) of each of the four constituent basis functions $\langle i, j | k, l \rangle$.

Thus far when matching ERI threads for identical control-flow we have been ignoring the level of contraction of the four constituent basis functions because there remains some uncertainty about the best way to handle this issue. This uncertainty arises from the specific manner with which contractions are handled by ERI algorithms.

Take, for example, the Rys quadrature algorithm [10–12]. This algorithm is known to perform well for workloads which exhibit high angular momenta but lowly contracted basis functions [14]. Rys quadrature is regarded as being inefficient for highly contracted basis sets because it handles contractions sim-

ply by unrolling the contractions, thereby requiring a relatively large number of primitive ERI computations, the results of which are contracted at the end—in other words such an algorithm performs contractions “late”. This is in contrast to some algorithms which perform contractions “early”, thus reducing the number of primitive ERI computations, and which are therefore more efficient for highly contracted basis functions (e.g., the PH method [13]). Although most workloads tend to exhibit highly contracted basis sets—and therefore it seems as though there would be a preference for algorithms which are efficient for highly contracted basis functions—these are often very inefficient or incapable of handling high angular momenta basis functions. Furthermore, as control-flow simplicity and regularity is a desirable feature for specialised hardware solutions, Rys quadrature is an attractive candidate [36].

The relative inefficiency of Rys quadrature is due to the many primitive ERI operations which need to be performed for a quartet of highly contracted basis functions. However, with SIMD processing, this flaw is less critical. Since each primitive ERI operation will be of identical control-flow (the angular momenta terms for all the primitive ERI operations would be the same), these primitive ERI computations could saturate a SIMD processor. Thus one may unroll the contraction loops and computing each ERI primitive operation in parallel. One would expect this to saturate a SIMD processor. The extent to which this is true is discussed in Section 6.1.

Although SIMD unit utilisation is a useful measure of success, saturation of a SIMD processor would be a meaningless victory if superfluous work was being done. Therefore, using Rys quadrature to compute highly contracted ERI quartets may be inappropriate even with a SIMD processor, especially since as we have shown it is possible to extract DLP out of multiple independent ERI threads and subsequently employ SIMD processing over multiple ERI threads, thus negating the need to extract DLP from individual ERI threads. With that in mind it may be more appropriate to utilise the best available ERI algorithm for any given workload based on its contraction and angular momenta characteristics. However, such a solution would require one to account for level of contraction in addition to angular momenta when matching ERI threads with identical control-flow, and intuitively this additional matching constraint may reduce the observed SIMD unit utilisation rates for any given realistic workload. The impact of accounting for contractions when matching threads is discussed in Section 6.2.

6.1. Unrolling contractions

Unrolling contractions exposes significant DLP within each thread (effectively “intra-thread” DLP). Combined with the DLP exposed through the proposed thread-matching scheme (effectively “inter-thread DLP”), one would expect very high SIMD utilisation. This is the case for our test workloads, as shown in Fig. 12.

However, one unexpected observation may be made based on Fig. 12 which contradicts the conjecture about SIMD unit utilisation increasing as workload size increases (which was

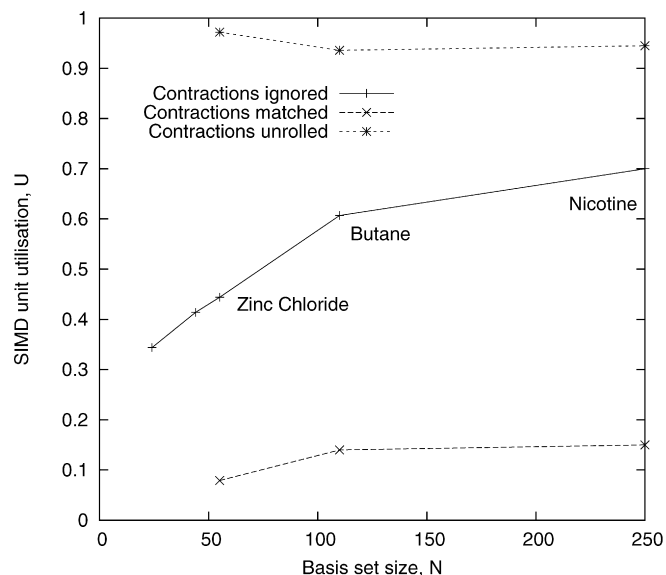


Fig. 12. SIMD unit utilisation for candidate workloads with various treatments of contraction. All workloads in 6-31G** basis. $W_{\text{threads}} = 16k$.

discussed in Section 5.1). It is apparent that the Butane workload and the Nicotine workload (run with contraction unrolling) have lower SIMD utilisation than the smaller Zinc Chloride workload (run with contraction unrolling). This is in fact due to the character of each basis set (rather than just the number of basis functions N within each basis set), and will be discussed in Section 6.3.

6.2. Thread-matching with contractions

Because matching threads for identical basis function quartet level of contraction is an additional constraint over matching threads for identical basis function quartet angular momenta, one expects to observe lower SIMD utilisation due to the more stringent matching requirements. This is the case for our test workloads, as shown in Fig. 12 (the Nicotine workload exhibits a very slight improvement in SIMD unit utilisation over the Butane workload).

The results seem to suggest that thread-matching with contraction dramatically diminishes the impact of SIMD processing. While it is true that contraction matching significantly increases the required ERI Sorting effort, it is incorrect to conclude that contraction matching must therefore be eliminated.

As will be shown in Section 8, increasing the thread population (W_{threads} in the discussion thus far) would dramatically increase the SIMD unit utilisation for the Butane and Nicotine workloads. However, keeping in mind that thus far we have presented results from a simulation of a hardware-based solution, running the experiments with larger W_{threads} will require very long computation time. Therefore, we later switched to a software-optimised implementation to carry out larger simulations. These larger simulations indicate that even with contraction matching, over 99% utilisation is possible with $V = 64$ for significant workloads.

6.3. Workload characteristics

Some may be concerned that the workloads tested thus far are relatively small, especially since we are targeting large capability problems. Fortunately, as the ERI matching problem typically diminishes in severity as workload size increases, we expect that an increase in workload size would reduce the required thread population W_{threads} and consequently a reduction in the required size of the CAM, which improves the practicality of our proposed solution. Nevertheless, we will proceed with simulations of larger workloads which, unfortunately, will scale as badly as the problem itself, i.e. $O(N^4)$, and are therefore very time-consuming.

The level of contraction and angular momenta of basis functions in the basis set characterise the computational workload. However, for the sample workloads, only the contraction characteristics have had an interesting and measurable impact on observed performance (revealing the impact of angular momenta characteristics would require the consideration of specific ERI evaluation algorithms, which we sought to avoid in the present work).

In Section 6.1 it was observed that, contrary to our conjecture that SIMD unit utilisation increases as workload size increases, the results with loop unrolling indicate that the best performing workload is the smallest workload. An examination of the basis sets in detail reveals the cause of this behaviour. The basis set used to express the Zinc Chloride molecule contains a greater proportion of highly contracted basis functions when compared to the basis set used to express the Butane molecule. Hence with contraction unrolling the ERI operations with the Zinc Chloride basis set exhibit greater “intra-thread” DLP which compensates for the lack of “inter-thread” DLP.

This raises interesting prospects for “short-vector” SIMD processing in particular. Whereas thus far we have observed a nominal vector width of $V = 64$, it should be noted that many existing systems have a much shorter vector width, particularly $V = 4$ (e.g., the SIMD processing units in the Cell Broadband Engine, and SIMD multimedia “extensions” in conventional processors such as x86 SSE and PowerPC AltiVec). With these systems, contraction unrolling may be especially appropriate, if contraction unrolling by itself were able to extract sufficient DLP such that ERI sorting were not needed. Such a scheme would be highly sensitive to basis set contraction (K) characteristics, rather than basis set size (N) characteristics. Table 2 compares the characteristics of several workloads to vector utilisation achieved with contraction unrolling only, with $V = 4$ (which is the SIMD width of the Cell BE SPEs, as well as SSE and AltiVec SIMD processing units in commodity x86/PowerPC processors). As illustrated in Fig. 13, there is a strong correlation between the mean level of contraction of the basis set ($\bar{K}$) with SIMD unit utilisation.

One expects that the angular momenta would also have an impact on the level of DLP inherent within each thread. However, this depends on the specific ERI evaluation algorithm used (e.g., Rys quadrature [10–12] would yield a different amount of DLP within a primitive integral computation than the PH algorithm [13] or the PRISM algorithm [14]). We have yet to review

Table 2

Vector utilisation with contraction unrolling only

Workload		N	Max. K	$\bar{K}$	$U_{V=4}$
6-31G**	C ₄ H ₁₀	110	6	1.65455	0.787212
	Nicotine	250	6	1.73600	0.818858
cc-pVTZ(-f)	C ₄ H ₁₀	270	8	1.37037	0.428996
	Nicotine	602	8	1.44518	0.434254
cc-pVTZ	C ₄ H ₁₀	470	8	1.21277	0.350279
	Nicotine	1010	8	1.26535	0.356933

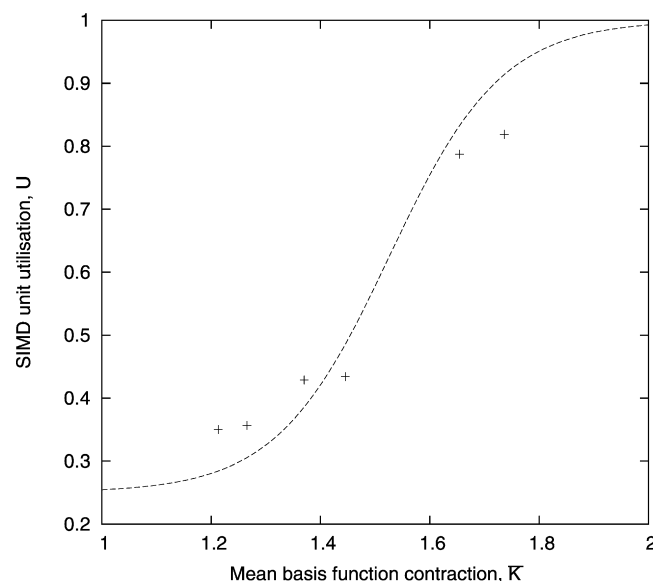


Fig. 13. SIMD unit utilisation with contraction unrolling only. $V = 4$, $W_{\text{threads}} = 1$ (i.e. ERI sorting effectively disabled). A “late-contraction” ERI routine (such as Rys quadrature) is assumed.

the level of DLP inherent within different ERI evaluation algorithms, however as an example we know from analysis of Rys quadrature that the recurrence relations for the evaluation of the “I-factors” (see Section 3 by Rys et al. [11]) are a good source of DLP for workloads with high angular momenta [5].

In summary, different workloads would contain different levels of DLP across three distinct layers:

- (i) Across thread-boundaries: DLP may be extracted from combinations of compatible ERI threads. The level of DLP available at this layer is a factor of workload size N .
- (ii) By unrolling contractions: Higher levels of contraction would (in general) yield a greater number of primitive ERI operations. The level of DLP available at this layer is a factor of workload characteristics, specifically the population of basis function contraction—a plurality of highly contracted basis functions may yield a greater level of DLP due to contraction unrolling, however some ERI evaluation algorithms seek to minimise the number of primitive ERI operations with the aim of reducing overall computation time, therefore thoroughly assessing the level of DLP at this stage requires specific knowledge of ERI algorithms, which is beyond the scope of the present work.

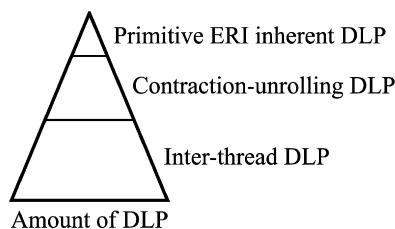


Fig. 14. Sources of DLP.

- (iii) Within evaluation of primitive ERI: Specific ERI evaluation algorithms will have a certain level of inherent DLP, particularly for the computation of recurrence relations which are common to many ERI evaluation algorithms. The level of DLP available at this layer is a factor of workload characteristics, specifically the population of basis function angular momenta—a plurality of high angular momenta basis functions may yield a greater level of DLP inherent in the computation of recurrence relations. Assessing the level of DLP at this stage requires specific knowledge of ERI algorithms, which is beyond the scope of the present work.

We hypothesise (as illustrated in Fig. 14) that for reasonably large workloads, one expects the greatest level of DLP to come from Layer 1 (inter-thread DLP). Considering typical workload characteristics (highly contracted, but with very few sources of high angular momenta such as transition metals) the next greatest level of DLP would come from Layer 2 (contraction unrolling), followed by some DLP from Layer 3 (DLP inherent in the ERI evaluation, e.g., due to recurrence relations).

7. Integral blocks

One of the most important improvements to the SCF procedure, originally introduced in the GAUSSIAN 70 package [37], is the so-called “shell structure”, whereby ERI are generated not by iterating over atoms, but by iterating over shells of basis functions. A shell is a collection of basis functions which have common center coordinates and exponential parameters. When computing a series of ERI with quartets of basis functions that have the same center coordinate and exponential parameters one may reuse many intermediary computer values, thereby obtaining a significant reduction in floating point operation count (and consequently computation time).

Because the shell structure relies on scheduling ERI quartets in a certain way, it is incongruous with ERI sorting, which randomises the scheduling of ERI quartets. One possible solution to this problem is to store these common intermediary values in a look-up-table (LUT) memory structure (which is similar to a CAM structure, although it may also be implemented as a “cheaper” conventional memory structure as there is no massive high-speed search capability required). The SIMD processor then reads these intermediaries as required from the shell structure LUT. This approach would not affect SIMD unit utilisation; it’s only impact is to reduce the number of floating-point operations (FLOPs) required by the SIMD processor to evaluate each ERI.

8. Implementation practicalities

There are two proposed approaches (as described in Section 3.3) to be examined here:

- (i) Implementation of sorting functionality on an existing conventional architecture.
- (ii) Implementation of sorting functionality with a fully custom hardware solution.

Implementation on an existing conventional architecture is immediately possible. In Section 8.1 we present an implementation strategy that obtains good performance, especially for very large workloads.

A fully custom hardware solution may not be immediately accessible by the broader computational chemistry community, however the potential performance gains are substantial. In Section 8.2 we discuss some issues and strategies for the implementation of such a system.

While sorting of ERI to extract DLP is essential, it is equally essential to actually perform SIMD execution of the matched ERI threads. There are several viable options for a high performance implementation of this processor. These are discussed in Section 8.3.

8.1. Software solution

The strategy employed with our software based ERI sorting solution is to perform early/eager subset creation by placing ERI quartets into ERI class subsets as soon as they are issued by the generating nested loop. Each subset is a container with space pre-allocated for V quartets—this often results in space wastage, however it avoids the relatively massive expense of dynamic memory allocation. In addition to each subset being pre-allocated, the number of subsets is also set a priori and pre-allocated, with no runtime dynamic memory allocation being used. Let the number of subsets be S . If one has S subsets and each subset has capacity for V quartets, then the system has the capacity to hold a maximum of $S \times V$ quartets.⁶ In order to mitigate the effects of the memory wall, specifically to increase the degree of spatial locality in the program, all S subset containers are allocated in contiguous chunks of memory (and each subset container is in turn allocated as a contiguous chunk of memory as well).

As was previously described, to have good vector unit utilisation one has to maximise the number of quartets present in

⁶ However, one cannot take $S \times V$ to be exactly the same as W_{threads} as has been defined earlier. Although the system may hold $S \times V$ ERI quartets, it can only hold S ERI classes, regardless of the number of quartets in each class. This upper limit of $S \times V$ quartets may only be reached if all subsets are full (which would lead to correspondingly perfect vector utilisation); in practice there would be partially filled subsets (i.e. individual subset population $< V$) since subset containers may only hold ERI quartets of equivalent class. The worst case performance would be for all subsets to each hold only 1 quartet, in which case the total population of quartets in the system is $S \times 1$. There is a general functional equivalence between W_{threads} and $S \times V$, however the terms may not be compared directly, therefore an early/eager implementation may not be straightforward to compare with a late/lazy implementation.

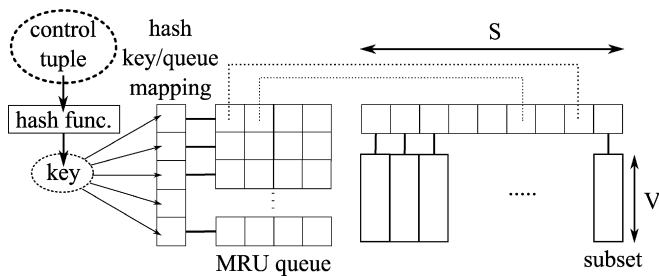


Fig. 15. ERI Sorting with conventional systems. Recall that the control tuple is a token comprising all the data that expresses the control-flow of the thread—in this application the control tuple contains the 3 angular momenta terms (a_x , a_y , a_z) for each of the 4 basis functions constituent in an ERI quartet, and optionally also the level of contraction K of each of the basis functions. A 64 bit control tuple is sufficiently large for this application.

the system. With the early/eager subsetting approach, this requires a sufficiently large S . Unfortunately, with a conventional processor, this imposes a worst-case $O(S)$ search operation when trying to match a new quartet with its corresponding subset (and typically, $S \geq 1k$). This problem can be mitigated through exploitation of temporal locality exhibited by the subsetting process, as well as use of *hashing*.

We have observed that subset mapping tends to exhibit a significant degree of temporal locality when the canonical ERI generation nested loop scheme [5] is employed. Basically, when a subset has been used, it is likely to be used again within a short period of time. To exploit this behaviour the history of subset access is maintained in the form of a queuing structure, such that whenever a new quartet is to be placed in a subset, the subset lookup/matching is performed in *most recently used* (MRU) sequence which would typically result in most lookups being completed in $\ll O(S)$ time. We have found that this optimisation reduces sorting time by 75% for the Zinc Chloride workload with $V = 64$ and $S = 16k$ on an AMD Opteron system. However, there are still a large number of lookups which require a significant number of search operations (up to $O(S)$) before the correct subset is identified, such that the overall performance is still inadequate.

The next optimisation is incorporation of a hashing function, which further reduces the lookup time and is particularly effective at “mopping up” the remaining subset lookups which do not benefit from exploitation of temporal locality. A *hash function*⁷ takes a control tuple (i.e. K , a_x , a_y , and a_z for each of the four basis functions) and generates a key that corresponds to a small number of ERI class subsets, effectively reducing the search-space. This smaller number of subsets may then be checked to identify an exact match. Therefore, hashing dramatically limits the number of subsets that need to be checked exhaustively. We propose that the MRU queue data structure be incorporated with this hashing approach. Our proposed system is illustrated in Fig. 15.

⁷ A hash function may be as rudimentary as a modulo operation performed on the input value, or as complex as the SHA-1 cryptography algorithm. The goal in either case is to reduce some data to a smaller “digest” of the information.

It should be noted that it is possible to increase the hashing width (i.e. the number of MRU queues) to such a large number that the hashing maps a quartet immediately to the correct subset, achieving $O(1)$ lookup time—this special case is called *perfect hashing*. This would correspond to the MRU queues in Fig. 15 having a maximum population of 1. However, it is likely that some queues will never be utilised (and hence their allocation would be a waste). This is because the hash functions we have tested do not generate keys in an optimum manner: there is some sparsity in the mapping which one cannot eliminate without employing look-up-table based methods, which require very large data structures that overwhelm the cache capacity of conventional systems, therefore resulting in poor overall system performance.

Note that if each MRU queue is perfectly utilised and has a minimum and maximum population of 1, i.e. the MRU queue always contains 1 subset and there are no wasted queues, then the hash is said to be *perfect and minimal*. Perfect minimal hashing would be ideal, however discovery of hashing functions that exhibit such characteristics (without the use of very large look-up-tables) is notoriously hard, and highly application specific. Therefore, one expects the hashing scheme employed to suffer some inefficiency and therefore overprovisioning MRU queues is necessary, which leads to some space wastage.

Although space wastage per se is not a critical problem (since memory capacity is relatively inexpensive), as we have already mentioned, one must be careful with *excessive* space wastage because space wastage implies large strides in memory access patterns, which in turn exhibit poor spatial locality and correspondingly poor overall system performance. Therefore one has to tradeoff between:

- (i) The average number of cycles required to identify the correct subset within an MRU queue—more MRU queues implies that each queue will be smaller, and therefore there will be a smaller population of subsets to check/match sequentially.
- (ii) The time taken to index the required MRU queue—more MRU queues implies greater queue structure overhead cost, and also potentially more wasted (unutilised) space, which in turn results in poorer usage of high-speed caches, and poorer performance overall.

We have observed that with AMD Opteron and Pentium 4 systems, a good balance is to have 256 hash key locations—corresponding to 256 MRU queues. System performance with less than 256 queues is limited by a large number of iterations spent trying to match a quartet to a subset. The performance at greater than 256 queues is limited by a large number of cache misses. This observation was made with several different workloads, with $V = 64$, $S = 1k$, and a CRC hashing function. The inflection point may change for other architectures and/or with different hash functions.

The results illustrated in Fig. 16 indicate that as workload size increases, the average per-ERI sorting time drops to 122 ns on a 1.8 GHz AMD Opteron system. Comparatively,

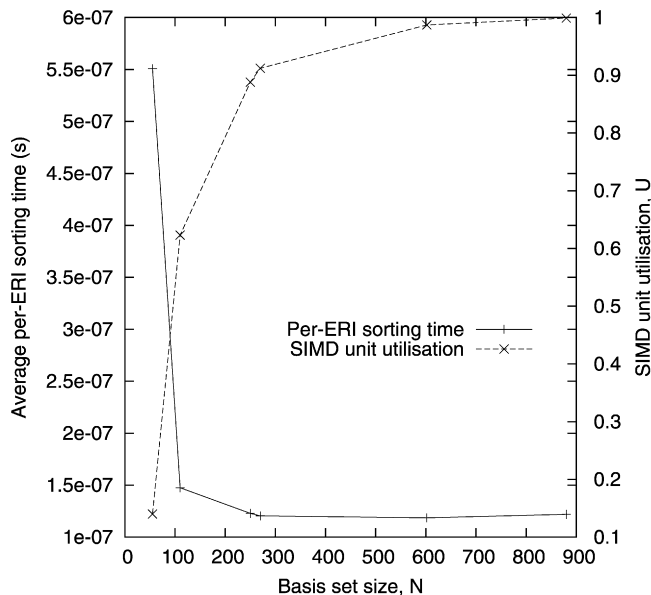


Fig. 16. Software-based ERI Sorting time and SIMD unit utilisation. $V = 64$, $S = 1k$, CRC hash function, 256 hash keys. Contractions are *matched* (as opposed to *unrolled* or *ignored*—see Section 6 for explanation of these different matching schemes).

we estimate the average time taken to compute a single ERI for most common workloads on 64-bit x86 cores is just over 200 ns. One may coarsely estimate the potential speedup of the system, assuming a $V = 64$ wide SIMD cruncher capable of computing V ERIs is 200 ns, the total aggregate performance of the system with the Carotene workload would be $(N_{\text{ERI}} \times 200 \text{ ns}/V) + T_{\text{sorting}} = 9372.26 \text{ s}$, or 125 ns per ERI on average, which already yields a small speedup of 1.6 over a conventional scalar core. Furthermore, the average per-ERI sorting time would continue to drop as workload size increases, albeit at a very low rate.

In addition, the utilisation plot in Fig. 16 indicates that even with contraction matching, very high vector utilisation may be achieved with larger workloads and larger effective W_{threads} —the curve may be considered a rough extension of the results presented in Fig. 12.

For very large workloads, the nature of the ERI sorting computation characteristics change such that the limiting factor is no longer hashing or queue traversal. For each integral one has to perform tuple sorting (described in Section 4.1) and one also has to compute a hash function. Both these operations are not negligible, and furthermore the cost of these operations do not diminish with the size of the workload. Our code performance profiling indicates that these operations are becoming a substantial consumer of processing time as workload size increases, and are likely to eventually be the limiting factor with this implementation. Having said that, for extremely large workloads it may be possible to forego the SIMD utilisation benefits of tuple sorting, and it may even be possible to use a simple modulo hash function. Therefore, this strategy may be viable for very large workloads, $N \gg 1k$.

8.2. Fully custom hardware solution

Although the software approach previously discussed may yield good performance on contemporary cores for very large workloads, even greater performance gains are possible with a CAM-based hardware ERI sorting solution.

The CAM-based approach is capable of consistently high ERI sorting performance—specifically, a batch of ERIs may be issued for SIMD computation on every clock cycle. Assuming a 500 MHz hardware platform, a vector of ERI quartets could be issued every 2 ns, and this performance will be sustainable even for medium-sized workloads such as those considered in this paper. A very rough estimate would suggest that with the Carotene workload, a speedup of almost 40 may be achieved with $V = 64$. Although this hardware-based solution is likely to be more costly and less accessible than the previously discussed software-based approach, the potential for much larger speedup across a broader range of workloads compels us to explore this possibility.

Any application-specific architecture proposal is confronted by the contemporary reality of extremely high once-off *non-recurring engineering* (NRE) expenses for the production of quality integrated circuits (IC)—costs are typically measured in the order of millions of dollars. Compounding this problem is the reality of an application such as HF-SCF having only a very small user-base by commodity standards (therefore it is not possible to distribute the high NRE expenses over a large number of users).

An obvious and attractive alternative to development of custom IC is the use of programmable logic devices, such as *field-programmable gate arrays* (FPGAs). An FPGA may be notionally thought of as a “blank sea of gates”, and the user may configure the “equivalent gates” to form any arbitrary digital circuit. As one would expect, this re-configurability incurs a substantial overhead, however this overhead is reduced by introducing heterogeneity into the FPGA fabric—specifically the introduction of dedicated multiplier logic makes FPGAs well suited to arithmetic applications. In fact, Underwood’s analysis of performance trends of CPUs and FPGAs reveals that the floating-point capabilities of FPGAs are growing at a rate significantly faster than conventional CPUs [38]. The success of FPGAs in floating-point applications goes beyond synthetic benchmarks: high performance has been observed for applications such as lattice quantum chromodynamics using logarithmic arithmetic [39] and molecular dynamics (over $30\times$ speedup vs. PC) [21]. Further encouraging optimism was the announcement by Hewlett-Packard regarding *field-programmable nanowire interconnect* (FPNI) technology, which suggests gains of a factor of eight in FPGA density using current IC fabrication lines [40]. The increase in functionality offered for large FPGA dies is substantial. It is also broadly accepted that the clock rates for FPGAs will continue to increase.

8.3. SIMD ERI processing

Besides ERI Sorting, there is the SIMD computation of ERI evaluation to consider. One possibility is to employ a general-

purpose SIMD processor for a back-end. Presently, the most widely available SIMD processors are graphics processing units (GPUs). As previously mentioned, the use of GPUs for quantum Monte Carlo computations has been proposed [27], citing the excellent performance and efficiency of GPUs, which is a result of the underlying SIMD architecture. The chief argument against the adoption of GPUs for scientific computation has been the issue of floating-point precision; we share the view of Anderson et al. that going forward this problem is likely to be solved [27,41]. Even if GPUs fail to evolve in floating-point precision or if it can be shown that they are unsuitable for scientific computation for some other reason, there are other SIMD processors which are more focused on scientific applications, such as the line of streaming processors from Stanford University, the most recent incarnation of which is the Merri-mac [42]. Furthermore, the only constraint placed upon the ERI Cruncher is the SIMD model—a concept which is unlikely to fade away in the foreseeable future. The proposed scheme is therefore likely to be relevant beyond the lifetime of specific computer implementations.

9. Conclusions and further work

When vector processing declined in favour of computational clusters, SIMD processing appeared to fade from the mainstream. It could be argued that SIMD processing is now undergoing a “rebirth” as it is becoming a centrepiece in modern architectures such as the Cell BE [2], modern general-purpose graphics processors [16] and likely the fruits of the Intel Larrabee project [3] as well. On the other hand, it could also be correctly argued that SIMD processing was “reborn” a long time ago, in the form of traditional graphics processors and multimedia extensions on commodity processors, which have driven the computer gaming and multimedia industry, although the birth of these went slightly unheeded by the scientific community due to the difficulty of using traditional GPUs for non-graphics purposes and also the traditionally limited arithmetic precision offered by GPU devices and most multimedia extension SIMD instruction sets. Finally, since vector processors never completely faded away, another group could correctly argue that SIMD processing never died in the first place—it was merely reduced to niche status. Whatever the history may be, it does appear that SIMD processing will be a more significant source of computational power across broad application strokes in the foreseeable future.

ERI sorting is an essential step to tap the power of SIMD processing for the HF-SCF application. This sorting functionality may be implemented on existing systems—with hashing and queuing, good performance may be obtained, and substantial speedup is especially possible for very large workloads. However, even greater gains are possible with a specialised hardware system which would be capable of yielding the required high-performance sorting. With programmable logic devices, implementation of such specialised systems is feasible.

Further work should include prototype construction of the hardware-based solution. Further work may also continue on the software-only solution, particularly testing with much larger

workloads and experimentation with other hash functions. Consideration of specific ERI evaluation algorithms should also follow. Finally, the benefits of bootstrap value reuse through the “shell structure” (as described in Section 7) should be considered vs. the cost of implementation, and methods to integrate this FLOP-saving feature with ERI sorting should be studied. Expanding the scope of this work would include exciting prospects. An obvious path is to consider the “SIMD-ification” of DFT methods, which naturally benefit from SIMD architectures due to very heavy use of dense linear algebra routines, which benefit greatly from SIMD architectures [43].

It was shown that the software-only approach (through benchmarking of an actual implementation) is capable of adequate performance for medium workloads ($N \approx 1k$), and it is suggested that this implementation will yield good speedup for much larger workloads ($N \gg 1k$). We have also discussed that the implementation approach centred around the use of a content-addressable memory (CAM) is viable, since the proposed size of the CAM in terms of searchword width ((36 bits) and number of datalines (16k) is reasonable and well within the limits of current-generation technology. The level of temporal locality exhibited by the application is encouraging as it appears likely that caching would be adequate to prevent the SIMD processor from being under-utilised due to memory latency. We have also indicated that the proposed scheme is able to leverage techniques that minimise redundant and insignificant ERIs such as screening and molecular symmetry, therefore a “brute-force” computation approach is not imposed—on the contrary, future efforts in the development of sophisticated techniques to minimise the number of ERI should continue to be exploited.

This study has indicated that the problem of mapping the application to specific computer architectural traits is difficult, because different workload characteristics map to different computational requirements, thus causing the optima to be a “moving target”. For example:

- High mean basis set contraction ($\bar{K}$) would suggest that contraction unrolling is sufficient for high SIMD unit utilisation (for short SIMD vectors) and consequently high performance.
- On the other hand, exploiting basis set size (N) through ERI sorting is less sensitive to particular basis set characteristics, although it is not as “cheap” as contraction unrolling.
- Even specifically with ERI sorting, there are at least two approaches:
 - The late/lazy associative-search approach matches well to a dedicated hardware implementation, which is costly, though it would yield very high performance for a very large subset of workloads.
 - The early/eager pre-sorting approach matches well to implementation on commodity processors, which is less costly than a hardware implementation, though more costly than contraction unrolling, and this approach yields good performance for very large N .
- Additional disruptions may occur due to different ERI routines being applied for different ERI classes.

However, more importantly, this study has also shown that there is substantial DLP available in significant workloads, and the level of DLP available grows as basis set size grows, thus fuelling optimism about ERI sorting. This is important given the growing realisation that the coming “manycore era” is ushering a secondary “SIMD era” as well. This is likely to take the form of processing nodes containing multiple heterogeneous cores. A substantial proportion of these cores are very likely to implement SIMD processing. This could potentially change the run-time performance profile of applications, such that components of the program that exhibit large DLP would enjoy greater speedup. Consequently, it may be necessary to reconsider the design of programs to take into consideration shifting bottlenecks. Tackling these issues adequately requires real-world performance numbers, and would be the subject of future research.

Although much more needs to be known about the specific details of future manycore processors, it is apparent that the pursuit of larger computing capability and capacity for ab initio computational chemistry workloads requires the exploitation of SIMD processing. Specifically for ERI evaluation, the proposed ERI sorting approaches would enable high performance SIMD processing, which will in turn enable highly accurate computations for very large workloads.

References

- [1] G. Baumgartner, et al., A high-level approach to synthesis of high-performance codes for quantum chemistry, in: *Supercomputing '02: Proceedings of the 2002 ACM/IEEE Conference on Supercomputing*, Los Alamitos, CA, USA, 2002, IEEE Computer Society Press, pp. 1–10.
- [2] S. Williams, et al., The potential of the cell processor for scientific computing, in: *CF'06: Proceedings of the 3rd Conference on Computing Frontiers*, New York, NY, USA, 2006, ACM Press, pp. 9–20.
- [3] J. Stokes, Clearing up the confusion over Intel's Larrabee, <http://arstechnica.com/articles/paedia/hardware/clearing-up-the-confusion-over-intels-larrabee.ars/1>, 2007.
- [4] M.F. Guest, S. Wilson, in: *Supercomputers in Chemistry*, American Chemical Society, 1981, pp. 1–37 (Chapter 1).
- [5] T. Ramdas, G.K. Egan, D. Abramson, K. Baldrige, Towards a special-purpose computer for Hartree–Fock computations, in: *Theoretical Chemistry Accounts* [online first], <http://dx.doi.org/10.1007/s00214-007-0306-6>, 2007.
- [6] T. Ramdas, G.K. Egan, D. Abramson, K. Baldrige, Converting massive TLP to DLP: A special-purpose processor for molecular orbital computations, in: *CF'07: Proceedings of the 4th Conference on Computing Frontiers*, New York, NY, USA, 2007, ACM Press.
- [7] J. Cioslowski, Ab Initio Calculations on Large Molecules: Methodology and Applications, in: *Reviews in Computational Chemistry*, vol. 4, VCH Publishers, Inc., 1993, pp. 1–33.
- [8] E. Schwegler, M. Challacombe, *J. Chem. Phys.* 105 (1996) 2726.
- [9] C. Dietrich-Buchecker, M.C. Jimenez-Molero, V. Sartor, J.-P. Sauvage, *Pure Appl. Chem.* 75 (2003) 1383.
- [10] M. Dupuis, J. Rys, H.F. King, *J. Chem. Phys.* 65 (1976) 111.
- [11] J. Rys, M. Dupuis, H.F. King, *J. Comp. Chem.* 4 (1983) 154.
- [12] M. Dupuis, A. Marquez, *J. Chem. Phys.* 114 (2001) 2067.
- [13] J.A. Pople, W.J. Hehre, *J. Comp. Phys.* 27 (1978) 161.
- [14] P.M.W. Gill, J.A. Pople, *Internat. J. Quantum Chem.* 40 (1991) 753.
- [15] S. Wilson, in: *Four-Index Transformations*, in: *Methods in Computational Chemistry*, vol. 1, Plenum Press, 1987, pp. 251–350 (Chapter 3).
- [16] J.D. Owens, et al., A survey of general-purpose computation on graphics hardware, in: *Eurographics 2005, State of the Art Reports*, 2005, pp. 21–51.
- [17] C.J. Thompson, S. Hahn, M. Oskin, Using modern graphics architectures for general-purpose computing: a framework and analysis, in: *MICRO 35: Proceedings of the 35th Annual ACM/IEEE International Symposium on Microarchitecture*, Los Alamitos, CA, USA, 2002, IEEE Computer Society Press, pp. 306–317.
- [18] U.J. Kapasi, et al., *IEEE Comput.* 36 (2003) 54.
- [19] T. Ramdas, G. Egan, A survey of FPGAs for acceleration of high performance computing and their application to computational molecular biology, in: *Proceedings of IEEE TENCON*, 2005.
- [20] R. Susukita, et al., *Comput. Phys. Comm.* 155 (2003) 115.
- [21] Y. Gu, T.V. Court, M.C. Herboldt, Accelerating molecular dynamics simulations with configurable circuits, in: *Proceedings of FPL 2005*, 2005, pp. 475–480.
- [22] K. Asanovic, et al., The landscape of parallel computing research: A view from Berkeley, UCB/EECS-20060183, Technical report, Electrical Engineering and Computer Sciences, University of California at Berkeley, 2006.
- [23] M.J. Irwin, J.P. Shen, Revitalizing computer architecture research, Technical report, Computing Research Association, 2005.
- [24] T. Mudge, *IEEE Comput.* 34 (2001) 52.
- [25] C. Lemuet, J. Sampson, J.-F. Collard, N. Jouppi, The potential energy efficiency of vector acceleration, in: *SC'06: Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, New York, NY, USA, 2006, ACM Press, p. 77.
- [26] N. Galoppo, N.K. Govindaraju, M. Henson, D. Manocha, Lu-gpu: Efficient algorithms for solving dense linear systems on graphics hardware, in: *SC'05: Proceedings of the ACM/IEEE Supercomputing 2005 Conference*, New York, NY, USA, 2005, ACM Press.
- [27] A.G. Anderson, W.A. Goddard III, P. Schröder, *Comput. Phys. Comm.* 117 (2007) 298.
- [28] R.J. Harrison, R. Shepard, *Annu. Rev. Phys. Chem.* 45 (1994) 623.
- [29] V.R. Saunders, M.F. Guest, *Comput. Phys. Comm.* 26 (1982) 389.
- [30] R. Ernenwein, M.-M. Rohmer, M. Bernard, *Comput. Phys. Comm.* 58 (1990) 305.
- [31] K. Hashimoto, et al., MOE: a special-purpose parallel computer for high-speed, large-scale molecular orbital calculation, in: *Supercomputing '99: Proceedings of the 1999 ACM/IEEE Conference on Supercomputing (CD-ROM)*, New York, NY, USA, 1999, ACM Press, p. 58.
- [32] K. Nakamura, et al., Eric: A special-purpose processor for ERI calculations in quantum chemistry applications, in: *HPC-Asia*, 2002.
- [33] K. Nakamura, et al., A high-performance, low-power chip multiprocessor for large scale molecular orbital calculation, in: *Proceedings of the Workshop on Unique Chips and Systems (UCAS)*, 2005, pp. 87–94.
- [34] K. Pagiamtzis, A. Sheikholeslami, *IEEE J. Solid-State Circuits* 41 (2006) 712.
- [35] J. Weinberg, M.O. McCracken, E. Strohmaier, A. Snavely, Quantifying locality in the memory access patterns of HPC applications, in: *SC'05: Proceedings of the 2005 ACM/IEEE Conference on Supercomputing*, Washington, DC, USA, 2005, IEEE Computer Society, p. 50.
- [36] J.D. Auspurg, D.E. Bernholdt, C.E. Dykstra, *J. Comp. Chem.* 11 (1990) 972.
- [37] M.D. Newton, W.J. Lathan, W.A. Hehre, J.A. Pople, *J. Chem. Phys.* 51 (1969) 3927.
- [38] K. Underwood, FPGAs vs. CPUs: trends in peak floating-point performance, in: *FPGA'04: Proceedings of the 2004 ACM/SIGDA 12th International Symposium on Field Programmable Gate Arrays*, New York, NY, USA, 2004, ACM Press, pp. 171–180.
- [39] O. Callanan, A. Nisbet, E. Ozer, J. Sexton, D. Gregg, FPGA implementation of a lattice quantum chromodynamics algorithm using logarithmic arithmetic, in: *IPDPS'05: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05)—Workshop 3*, Washington, DC, USA, 2005, IEEE Computer Society, p. 146.2.
- [40] HP presents alternate strategy for chip improvement, <http://www.hp.com/hpinfo/newsroom/press/2007/070116a.html>, 2007.
- [41] B. Dipert, Instigating a platform tug of war: Graphics vendors hunger for CPU suppliers' turf, <http://www.edn.com/article/CA6262535.html>, 2005.
- [42] W. Dally, P. Hanrahan, Merrimac: Supercomputing with streams, in: *Proceedings of Supercomputing '03*, 2003.
- [43] P.A. Agarwal, et al., ORNL/TM-2004/13: Cray X1 evaluation status report, Technical report, Oak Ridge National Laboratory, 2004.