JCTC Journal of Chemical Theory and Computation

Article

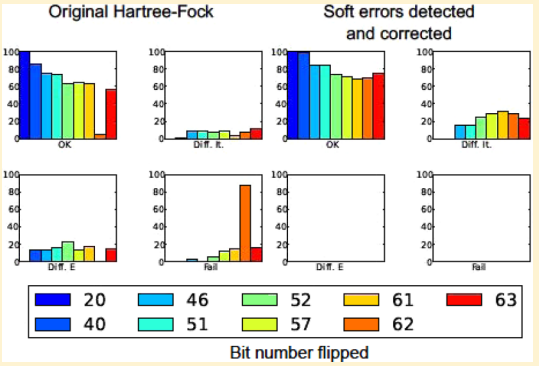
pubs.acs.org/JCTC

# A Case for Soft Error Detection and Correction in Computational Chemistry

Hubertus J. J. van Dam,* Abhinav Vishnu,* and Wibe A. de Jong*,†

Pacific Northwest National Laboratory, 902 Battelle Boulevard, Richland, Washington 99354-1793, United States

**ABSTRACT:** High performance computing platforms are expected to deliver $10^{18}$ floating operations per second by the year 2022 through the deployment of millions of cores. Even if every core is highly reliable the sheer number of them will mean that the mean time between failures will become so short that most application runs will suffer at least one fault. In particular soft errors caused by intermittent incorrect behavior of the hardware are a concern as they lead to silent data corruption. In this paper we investigate the impact of soft errors on optimization algorithms using Hartree−Fock as a particular example. Optimization algorithms iteratively reduce the error in the initial guess to reach the intended solution. Therefore they may intuitively appear to be resilient to soft errors. Our results show that this is true for soft errors of small magnitudes but not for large errors. We suggest error detection and correction mechanisms for different classes of data structures. The results obtained with these mechanisms indicate that we can correct more than 95% of the soft errors at moderate increases in the computational cost.




## 1. INTRODUCTION

The demand for large scale computing has been driven by science domains, such as computational chemistry, which dominate the use of high-end systems existing today.[1] This development is supported by a continuing growth in capability of the high-end computers. As a result all the top 10 machines in the TOP500 list can currently deliver over a petaflop per second (i.e., $10^{15}$ FLoating point OPerations per Second (FLOPS)).[2] To meet the growing computational requirements and solve bigger problems, even larger machines are being planned in the next decade.[3] To deliver the 100× performance increase relative to the largest computers of today the planned systems will need to combine millions of cores. Even though every core is highly reliable their sheer number will make faults commonplace. As a result fault resilience must be considered a central issue in application development rather than an after thought.

In practice a wide variety of faults may occur. Some faults may express themselves as hard faults in which processes terminate prematurely. In an earlier paper we descibed an approach to handling those in compute intensive applications.[4] In other cases the errors may express themselves as soft errors in the form of intermittently incorrect behavior of the compute system. These errors are often related to bit-flips in the compute environment.[5] Soft errors are a major source of Silent Data Corruption (SDC), potentially causing simulation failures or invalid results. In particular the risk of the latter outcome is serious as it can put the scientific value of a project into question. An obvious way to address such problems is to use redundancy. One may, for example, run the same calculation multiple times, or at a more fine grained level duplicate executable statements and check whether the results agree.[6] However, this reduces the efficiency to at most 50%. Therefore, even when redundancy might be useful it should be used only very selectively to minimize the overhead on the whole calculation.

Another important aspect to consider is the inherent soft error resiliency of an algorithm. Optimization algorithms, which iteratively approach a solution by reducing the error in the current guess of the answer until a convergence criterium is satisfied, intuitively seem highly resilient to intermittent perturbations. Of course it is expected that a perturbation might cause the calculation to require more iterations to converge dependent on the perturbation magnitude. Nevertheless convergence would be considered likely. To test to which extent this expectation is actually true we consider the Hartree−Fock method as a well understood example of optimization algorithms. In addition the method is highly relevant to computational chemistry. In general terms the approach used is based on randomly inserted errors by flipping bits and examining the impact on the results. Furthermore, we consider a number of ways to detect and correct soft errors while aiming to minimize the overhead introduced and report on their effectiveness. We expect that the results will help in designing resilient versions of optimization algorithms in general and in chemistry in particular. To the best of our knowledge this is the first study of the impact of soft errors in computational chemistry.

In linear algebra studies based on similar assumptions were published by Bronevetsky and de Supinski[7] and Du et al.,[8] but they focused on the simpler of case of iterative linear system

**Received:** June 10, 2013
**Published:** July 19, 2013

solvers. In addition their error detection mechanisms were based on complementing vectors and matrices with checksums. This is useful when the full matrix is available in advance but problematic when, for example, the matrix is computed on the fly as in the latter case one cannot guarantee that the checksums are computed correctly. Both approaches used in-memory checkpointing for the recovery of read-only data. Finally, fault tolerant runtimes are essential for recovery as proposed by Vishnu et al.[9−12]

The remainder of this article is organized as follows: In section 2 the details of the Hartree−Fock algorithm used will be outlined. The focus is in particular on characteristics of the data structures involved with a view to potential consequences in the presence of faults. In section 3.3 we describe the experiments we have performed injecting errors in the application data and how that impacts the calculation results. Some essential back ground on binary number representations will be provided also to clarify the impact of individual bit-flips on the number values. In addition we will discuss the differences in outcomes obtained when different data structures are perturbed. In section 4 we consider which characteristics of the Hartree−Fock method can be used to formulate error detection mechanisms. Furthermore, we discuss ways in which detected errors can be corrected and what the residual impact is of soft errors on the calculation results. We also discuss the overheads introduced by the error detection and correction mechanisms.

## 2. QUASI-NEWTON−RAPHSON HARTREE−FOCK METHOD

The Hartree−Fock method is a commonly used approach to calculate the lowest energy that can be achieved by a single Slater determinant wave function. It is of particular importance in ab initio quantum chemistry as calculations of this kind or the algorithmitically closely related and widely used Density Functional Theory (DFT) are the starting point of any computational study. Another reason for selecting the Hartree−Fock method is that it is a well understood algorithm hence allowing us to focus on the impact of soft-errors. It is assumed that other optimization algorithms will be affected in ways similar to the Hartree−Fock method. In particular we consider the quasi-Newton−Raphson algorithm.[13] In short the algorithm can be summarized as in Figure 1. Below the components of the algorithm that we have studied are outlined.

A key data structure in the Hartree−Fock method is the collection of orbital coefficients $C$. This data structure is provided as the initial guess, and it is also the data structure the convergence of which determines the termination of the algorithm. The orbital coefficients $C$ are stored in a distributed fashion. They specify the current wave function and initially are used to construct the density matrix

$$D_{pq} = 2\sum_{i=1}^{N/2} C_{pi}C_{qi}^{*} \tag{1}$$

where $N$ is the total number of electrons for a closed shell system. The density matrix is an intermediate quantity that is calculated afresh from the orbital coefficients every iteration and is stored in a distributed way.

The Fock matrix for the closed shell formalism is given as

$$F_{pq} = 2h_{pq} + \sum_{st}\{2(pq|st) - (pt|sq)\}D_{st} \tag{2}$$

In this expression $h$ represents the one-electron Hamiltonian, consisting of kinetic energy and nuclear attraction terms. The one-electron Hamiltonian is calculated once and stored in distributed memory. It remains constant throughout the Hartree−Fock procedure. The one-electron Hamiltonian is given by

$$h_{pq} = -\frac{1}{2}\int \chi_p(R_A;\, r_1)\nabla_1^2\chi_q^{*}(R_B;\, r_1)\,\mathrm{d}r_1 - \sum_C \int \frac{Z_C\chi_p(R_A;\, r_1)\chi_q^{*}(R_B;\, r_1)}{R_C}\,\mathrm{d}r_1 \tag{3}$$

The $\chi$ are normalized Gaussian type atomic basis functions. They depend on the atom positions as specified in the molecular geometry and the exponents, contraction coefficients, and angular momenta as specified in the basis set. The atom positions are represented as $R_X$ for atom $X$ and the electron positions are represented as $r_n$ for electron $n$. The nuclear charge of an atom is given by $Z_X$. In NWChem both the geometry and the basis set are stored in corresponding data structures. Both the data structures are replicated which means that every process has a local copy of the data structure. The data within these datastructures is the same for all processes. Also during the Hartree−Fock calculation the contents of the geometry and basis set are to remain unchanged.

Another key component in the Fock matrix is the set of two-electron integrals, given by

$$(pq|st) = \int\int \chi_p(R_A;\, r_1)\chi_q^{*}(R_B;\, r_1)r_{12}^{-1}\chi_s(R_C;\, r_2)\,\chi_t^{*}(R_D;\, r_2)\,\mathrm{d}r_1\,\mathrm{d}r_2 \tag{4}$$

With respect to the two-electron integrals it is important to note that their number is typically very large. Hence the complete set of integrals is partitioned into subsets and distributed across all processes. In the so-called conventional approach the integrals are calculated once and stored, whereas in the direct approach the integrals are recalculated whenever needed. In practice the choice between the two approaches depends on the volume of storage available and the speed with which that storage can be accessed.

The gradient with respect to the orbital coefficients of the Hartree−Fock total energy is given by

$$G_{ia} = 2\sum_{pq} C_{ip}^{T}F_{pq}C_{qa} \tag{5}$$

where $i$ and $a$ are occupied and virtual molecular orbitals, respectively. The occupied−occupied and virtual−virtual blocks of the gradient are zero as the Hartree−Fock energy is invariant

Initial MOs in $C^0$
Repeat
  Construct the density matrix $D$ from the MOs
  Construct the Fock matrix $F$ from the integrals and density matrix
  Establish the norm of the gradient $|g|$ wrt. the orbital coefficients
  Precondition the gradient and construct a skew symmetric matrix $k = P^{-1}g$
  Calculate the unitary orbital transformation $U = \exp(k)$
  Transform the orbitals $C^1 = C^0U$
  Orthonormalize the orbitals
  Set $C^0 = C^1$
Until $|g| <$ tolerance

**Figure 1.** Quasi-Newton−Raphson Hartree−Fock algorithm.

under transformations among the occupied orbitals or the virtual orbitals.

The search direction is generated by preconditioning the gradient as

$$S_{ia} = -\frac{G_{ia}}{\varepsilon_a - \varepsilon_i}, \quad S_{ai} = 0 \tag{6}$$

where $\varepsilon_i$ and $\varepsilon_a$ are occupied and virtual orbital energies. The orbital rotation $K$ can then be generated as

$$K = S - S^T \tag{7}$$

The matrix $K$ is a skew symmetric matrix from which a unitary orbital transformation can be calculated as an exponential

$$U = \mathrm{e}^K \tag{8}$$

The evaluation of the exponential is expressed in terms of matrix−matrix multiplications. The algorithm iterates until $U$ is close to unitary. All the matrices $S$, $K$, and $U$ are distributed and calculated from scratch every time. An additional matrix−matrix multiplication applies the unitary transformation to the orbitals to obtain the new orbital coefficients as indicated in Figure 1.

To ensure that orthonormality is maintained the orbitals are orthonormalized using a Gramm−Schmidt method every fifth iteration. The overlap matrix that appears in the dot-product $s_{ij} = \sum_{pq} C_{pi} S_{pq} C_{qj}$ is recalculated for every orthonormalization phase.

## 3. IMPACT OF UNCORRECTED SOFT ERRORS

As stated in the introduction soft-errors are caused by intermittent incorrect behavior of computing equipment. This behavior might occur in memory access operations or during the execution of instructions. Either way the result is that some data elements may come to contain information that is different from what it should be. The consequences of this will typically depend on the size of the difference as well as the characteristics of the algorithm in which this data is used. In this section we investigate how soft-errors impact the Hartree−Fock algorithm described above. Soft-errors are modeled as a single bit-flip in a floating point value. The scale of the resulting change depends on the location of the bit-flip in the binary representation of the floating point value. The goal is to establish how the size of the error and the affected data structure impact the outcomes of the Hartree−Fock method.

**3.1. Double Precision Floating Point Number Representations.** In order to understand the impact of a single bit-flip on a floating point value some knowledge of binary number representations is required. The binary representation of double precision floating point values is described in IEEE Standard 754-2008.[14] In this standard a floating point number is specified by a triple of the sign, exponent, and significand. With a double precision number being represented by 64 bits a number is stored as indicated in Figure 2.

In the calculations outlined in the next subsection we introduce bit-flips in particular bits. In Table 1 we list how the value of a floating point number is changed by particular bit-flips. The data clearly shows that bit-flips in the low bit numbers have little effect on the value of the data. Even bit-flips in bit 40 introduce relative errors of only 1.0e−4. This picture changes dramatically when bit-flips occur in the exponent. Those kinds of bit-flips may change the value by more than 100 orders of magnitude. Worse, the direction of this change depends on the original value. A bit-flip in bit 61 of the value 1.0 decreases the data by 155 orders of magnitude, but if the original value is 10.0 the same bit-flip increases the value by 154 orders of magnitude. These kinds of effects should be expected to have a nontrivial impact on a calculation.

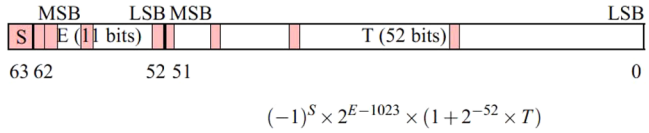



**Figure 2.** Binary presentation of a double precision floating point number where $S$ is the sign, $E$ the biased exponent, and $T$ the trailing significand field. $S$, $E$, and $T$ are interpreted as unsigned integers. MSB and LSB are the Most Significant Bit and Least Significant Bit, respectively. The red fields indicate the bits that are flipped in the tests.

**3.2. Inserting Errors.** In order to assess the impact of soft errors on the outcomes of optimization calculations we modified the NWChem Hartree−Fock code to enable bit-flips to be inserted. The aim of the modified code is to obtain some statistics on how bit flips affect the results by repeatedly running the same test case but inserting different perturbations.

For this purpose a routine was created that uses a pseudo random number generator. The routine decides whether a perturbation should be introduced in the calculation of a quantity, and if so, which data element should be perturbed. To ensure that each run uses a different pseudo random sequence the seed is computed from the date and time as obtained from the operating system. In addition the processor rank feeds into the seed equation to ensure that different processors are subject to different faults. The details of the seed computation are given in Appendix A. The random number generator is seeded at the beginning of the calculation. The perturbation routine also can be enabled selectively in different parts of the code. This way the effects of perturbations in specific terms on the overall calculation can be studied. When perturbations are inserted information on the perturbation is logged in a text file. At the end of a run this information was appended to the regular output to construct a complete record of that calculation.

In collecting the statistics the perturbation logs were used to detect whether a calculation experienced a perturbation. The same calculation was repeated until the outputs of 100 perturbed calculations had been collected. Each of the perturbed calculations was affected by at least one inserted error, but calculations may have suffered multiple errors. This procedure was repeated for inserting bit-flips in different bits of varying significance. The bits selected were bit numbers 20, 40, 46, 51, 52, 57, 61, 62, and 63. This selection includes bit-flips of different significance in the sign, exponent, and significant of floating point values.

The process was repeated again for different quantities in the calculation. The quantities perturbed were the basis set, the molecular geometry, the electron density matrix, the two-electron integrals, the Fock matrix, the orbital transformation matrix, the transformed orbitals, and the orthogonalized orbitals. In a last pass, perturbations in all terms were enabled. The results obtained this way are discussed in the next section.

In light of the large number of calculations that need to be run and considering the degradation of the performance when significant perturbations are imposed, a small molecular system was chosen. The calculations were performed on $H_2O$ at the equilibrium geometry in the 6-31G** basis set.[15,16] Although the calculation involves only three atoms and 25 basis

Table 1. Effect of Particular Bit-Flips on the Numbers 0.1, 1.0, and 10.0 Represented in the IEEE 754-2008 Double Precision Binary Format

| bit | result | | |
|---|---|---|---|
| none | 0.1e+000 | 0.1e+001 | 0.1e+002 |
| 20 | 0.999999999854481E−001 | 0.1000000000232831E+001 | 0.1000000000186265E+002 |
| 40 | 0.999847412109375E−001 | 0.1000244140625000E+001 | 0.1000195312500000E+002 |
| 46 | 0.100976562500000E+000 | 0.1015625000000000E+001 | 0.1012500000000000E+002 |
| 51 | 0.687500000000000E−001 | 0.1500000000000000E+001 | 0.1400000000000000E+002 |
| 52 | 0.500000000000000E−001 | 0.5000000000000000E+000 | 0.2000000000000000E+002 |
| 57 | 0.232830643653870E−010 | 0.2328306436538696E−009 | 0.4294967296000000E+011 |
| 61 | 0.745834073120020E−155 | 0.7458340731200207E−154 | 0.1340780792994260E+156 |
| 62 | 0.179769313486231E+308 | +Infinity | 0.5562684646268003E−307 |
| 63 | −0.100000000000000E+000 | −0.1000000000000000E+001 | −0.1000000000000000E+002 |

functions, deploying 12 cores ensures the presence of data on all cores.

**3.3. Performance Evaluation with Uncorrected Soft Errors.** The data presented here was calculated using the conventional Hartree−Fock implementation where the two-electron integrals are calculated once and stored. The overall number of calculations performed using the methodology presented above requires the results to be classified into broader categories. We classify the results in success and failure categories.

Calculations are classified in the success category if either correct results were obtained at the same effort or correct results were obtained at an increased effort. The results are considered correct if the total energy at convergence deviates less than 1.0e−6 hartree from the one obtained from an unperturbed calculation. The cost of a calculation is measured by the number of Hartree−Fock iterations to convergence. If the number of iterations in a perturbed calculation is the same as for an unperturbed calculation the effort is considered to be the same.

Calculations are classified as failed if either incorrect results were obtained or the calculation never reached a converged solution. The latter category applies, for example, if the calculations exceed the maximum number of iterations but also if a floating point exception causes the program to abort.

The results in Figure 3 show the impact of perturbations. For each data structure, the top left panel and the top right panel show the fraction of the calculations that completed successfully with the same and increased effort relative to the unperturbed case, respectively. The bottom left panel shows the fraction of calculations that failed producing incorrect results. Finally, the bottom right panel shows the fraction of calculations that failed abruptly without a final result. In each panel the different columns correspond to different bits being flipped.

The results in Figure 3 clearly demonstrate that the size of the perturbation has a significant effect on the observed consequences. Bit-flips in bit 20 change the data in the 10th decimal place. Very often these perturbations are insignificant. On the other hand bit flips in bit 62 which is the most significant bit in the exponent almost always result in the calculation failing.

Apart from different bit flips having different consequences the Hartree−Fock algorithm also shows different sensitivities to perturbations in different quantities. In particular errors that affect the read only data structures, such as the basis set and the geometry, tend to affect the results significantly. This should not come as a surprise as the Hartree−Fock method does not update these data structures and hence errors once introduced remain until the calculation terminates.

The algorithm is much less sensitive to perturbations in the density matrix, Fock matrix, matrix exponential, and orbital transformation matrix than to perturbations in the orbital orthogonalization in particular when the perturbations are small. There are two rationalizations for this. First, these quantities are recomputed every iteration. Hence every perturbation directly affects the quantity for one iteration only. Second, some quantities such as the matrix exponential are iterated until a unitary matrix results. This is a particulary straightforward optimization that can be completed successfully despite most perturbations. The resulting orbital transformation matrix will be different of course but it will still amount to a valid rotation. In other cases, for example, when the orbital transformation matrix is perturbed, the transformation step is followed by a reorthogonalization of the orbitals which corrects perturbations to a large extent.

Perturbations in the orbital orthonormalization have a particularly detrimental effect. The reason for this is that the orthonormality of the orbitals is an essential constraint that guarantees that the total energy is bounded from below. Violating this constraint allows the calculation to deviate significantly from the physically admissible values.

Finally, the impact of a permanent perturbation can be compared against a truly intermittent one by looking at the impact of perturbations in the integrals. In Figure 3 the data for the integrals was generated using the conventional Fock builders. In the conventional approach the integrals are computed once and stored. In the direct approach, by contrast, the integrals are computed whenever needed. This difference implies that if an integral is perturbed in the conventional approach the perturbation remains throughout the rest of the calculation. In the direct approach only one Fock matrix is impacted by a perturbation. In Figure 4 the results obtained with both methods are compared. The results clearly show that a persistent perturbation almost certainly invalidates the results, whereas the direct calculation usually recovers from an intermittent perturbation.

## 4. DETECTING AND CORRECTING SOFT ERRORS

The previous section provided a detailed analysis of the possible failures with bit-flips in multiple data structures. From the performance evaluation, we can conclude that it is insufficient to rely on the algorithmic properties of the Hartree−Fock method to correct all the possible bit-flips and resulting data corruption. It is important to design scalable methods for detecting soft errors. Clearly, it is not possible to

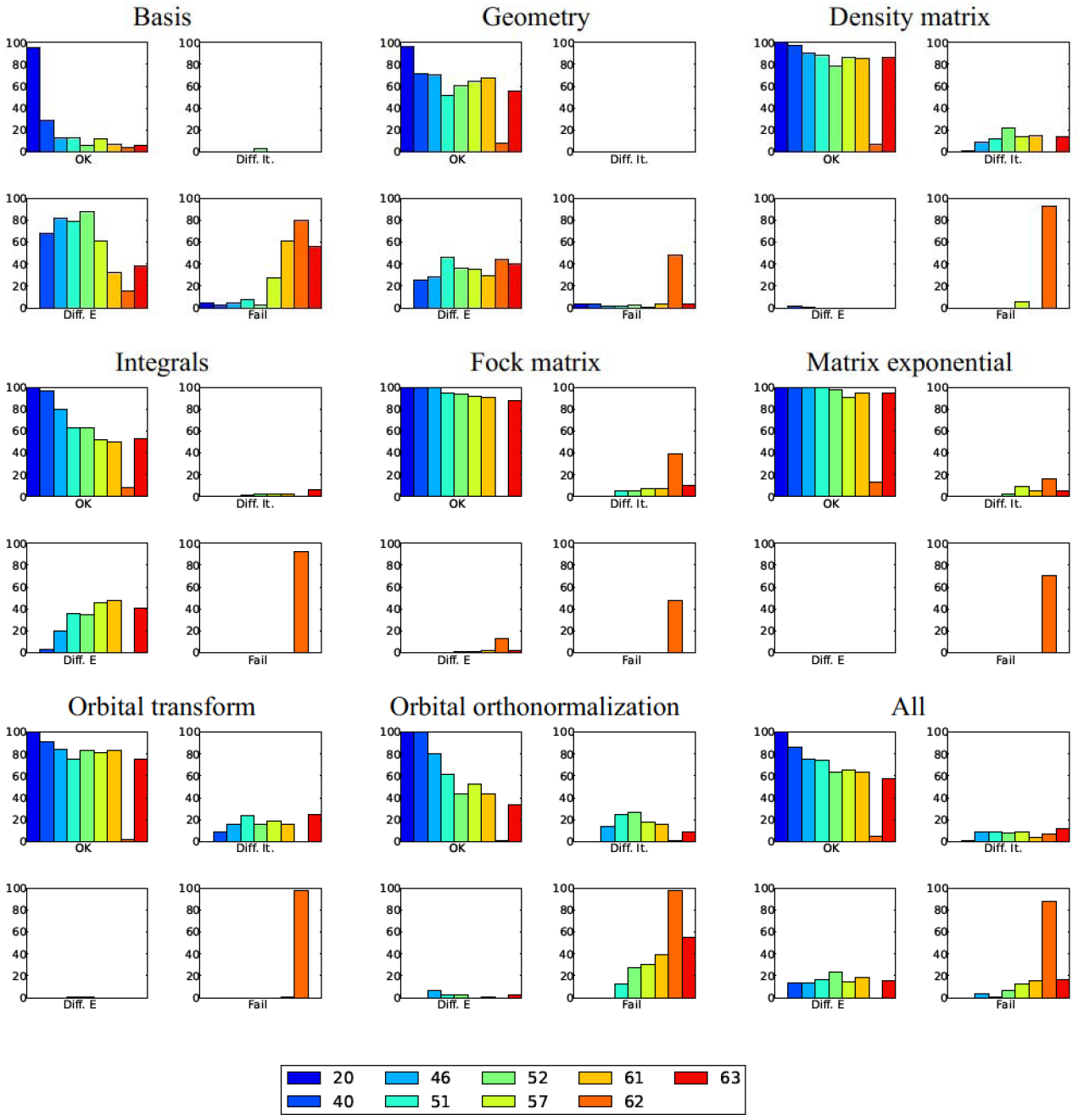



**Figure 3.** Impact of bit flips in different bits for a variety of terms in the Hartree−Fock algorithm using the conventional two-electron integral algorithm. With the histograms the label "OK" refers to calculations where both the total energy as well as the number of iterations to convergence were the same as in an unperturbed calculation, "Diff. It" means that the correct energy was obtained after additional iteration, "Diff. E" means that a different energy was found, and "Fail" means that the program terminated without a converged answer. The vertical axis gives the fraction of calculations as a percentage. Each column refers to a different bit, the number of which is given in the legend.

perform checksums on each bit for detecting soft errors. An important and useful mechanism for detecting soft errors is by designing scalable numerical assertions. These assertions should minimize the overhead in terms of time and space complexity and yet improve the reliability of the calculation.

In the next section, we discuss mechanisms for detecting and correcting soft errors in various data structures. The data structures are classified in replicated (each process owns a complete copy of the data structure) and distributed (the data structure is distributed across the processes). The distinction is important in designing fault detection and correction mechanisms for various data structures.

### 4.1. Addressing Errors in Replicated Data Structures.

Replicated data structures have identical copies of a data structure on each process. A soft error on a replicated copy would result in a divergence from the unaffected copies on other processes. There are many approaches to handle this. A particularly convenient approach is to calculate a checksum of the replicated data structure and communicate this to other processes. Subsequently a quorum can be established to check the correctness of the local data, and if necessary unaffected copies can be used to restore any corrupted data. This approach is beneficial as the checksums can be computed locally and very

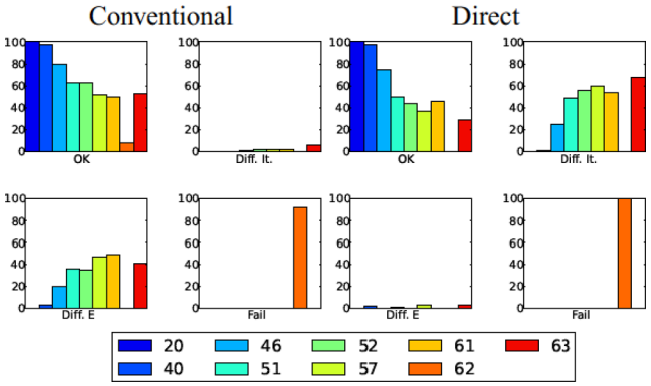



**Figure 4.** Impact of bit-flips when using the conventional integral vs direct intergral approaches in the Hartree−Fock algorithm. The labels on the square have the same semantics as in Figure 3.

little information needs to be communicated to other processes to test the data consistency.

The main replicated data structures in the Hartree−Fock algorithm are the geometry and the basis set. The geometry and basis set objects contain a variety of data elements including names of atoms, atomic coordinates, coefficients, and exponents of primitive basis functions, as well as sizes of various data structures. It is essential that the approach for calculating checksums is very generic. As a result, we have used the MD5 algorithm[17] which is considered to be of acceptable quality for error detection.[18] The MD5 algorithm produces 128-bit digests of the data objects on each processor. This information is shared with other processors to compute the quorum.

A systolic loop based approach is used to communicate the checksums as shown in Figures 5, 6 and 7. In the figures the vertical axis refers to the processor ranks, the horizontal axes are the processor ranks that a processor holds checksum data about. The boxes represent arrays of checksum data held by a processor. Initially the processors compute checksums of the data they hold locally. In the figures processor 2 is assumed to hold corrupt data as indicated by the red box holding the checksum. The checksum of a given processor is passed *nprange* times in the direction of higher processor ranks (see Figure 5) and then the same number of times in the direction of lower processor ranks (see Figure 6). The value of *nprange* must be at least 2. In addition at the lowest and highest processor rank the processor numbers are assumed to wrap around. After distributing the checksums every processor has 2 * *nprange* + 1 checksums. Subsequently, each processor finds the highest frequency checksum. The quorum of this checksum has to be at least *nprange* + 2 to guarantee complete recovery is possible. If

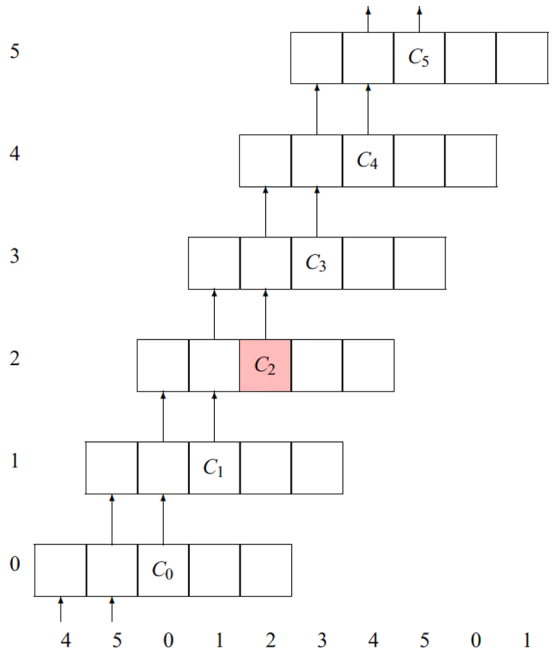



**Figure 5.** Distributing the checksums using a systolic loop migrating in the upward direction. Rank 2 has corrupt data and *nprange* Equal 2.

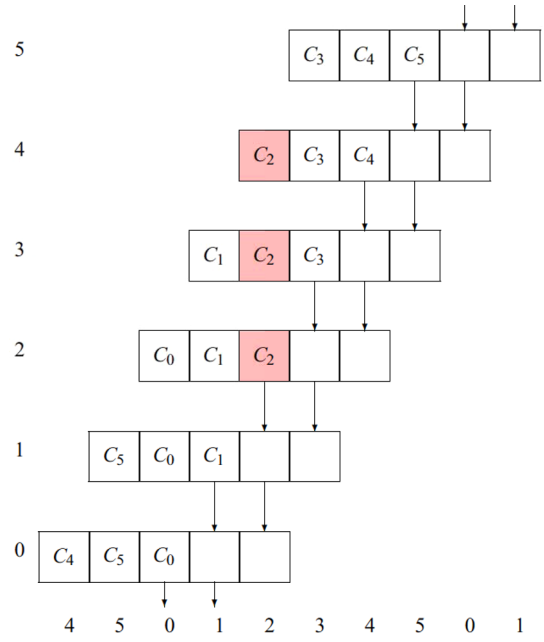



**Figure 6.** Data after the upward pass and migrating the checksums in the downward direction.

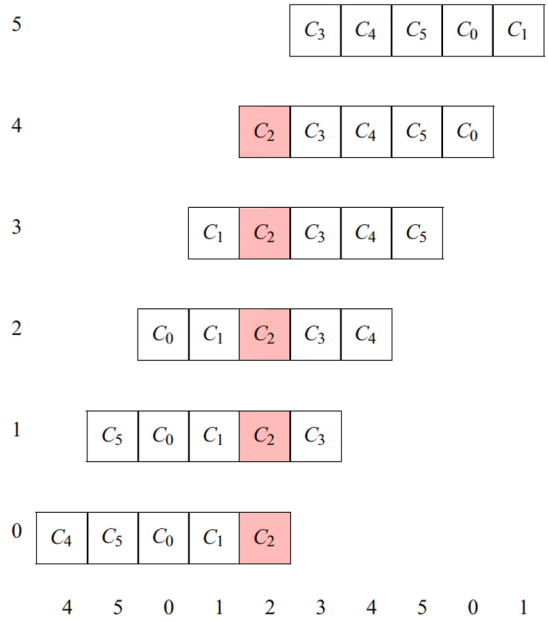



**Figure 7.** Checksums that each processor holds after the systolic loops in both directions.


dx.doi.org/10.1021/ct400489c | J. Chem. Theory Comput. 2013, 9, 3995−4005

a processor finds its own checksum does not match the highest frequency checksum it concludes its data is corrupt. For example, in Figure 7 processor 2 will find its data is corrupt. The processor then searches in the direction of lower processor ranks for the first processor with a correct copy and waits for that processor to send that data. In the example this leads to processor 2 expecting to receive a correct copy from processor 1. If a processor concludes its data is correct it sends a copy to every processor of higher rank holding incorrect data until a processor with a correct copy is found. In the example processor 1 will find that 2 holds incorrect data and send it a correct copy, and subsequently it finds that processor 3 holds correct data and concludes that the recovery is complete. All other processors holding correct data find that their neighbor of higher rank already has a correct copy and nothing needs to be done. The minimum quorum is required to guarantee that there always is a processor with a correct copy to terminate the recovery phase. That is, in the example processor 1 has to find at least one processor of higher rank that holds the correct data to safely conclude that all errors have been addressed. This systolic loop approach has the advantage that the elapse time of the recovery phase is independent of the total number of processors.

Obviously this recovery mechanism can be readily applied to all replicated data structures. However, some care is required choosing when it will be used because this mechanism requires the synchronization of all processors. In principle this limitation can be lifted by storing all replicated data structures in globally addressible memory. Then when a processor notices that the checksum of its copy no longer matches the other checksums it can unilaterally decide to fetch another process's copy. In practice, however, it requires a significant programming effort to change the way the replicated data objects are stored so that they are globally accessible.

Within the approach described above the consideration of when to invoke this procedure is based on isolating errors as much as possible. Therefore, when a quantity has been calculated that directly depends on replicated data structures those data structures are checked for errors. If any of the replicated data structures has been affected by an error it is restored. Also the quantity just calculated is flagged as potentially incorrect because the corrupt data might have been used to calculate it. In many cases it is best to recalculate the quantity to ensure it is calculated from correct data. This is feasible if the time needed to calculate the quantity is significantly shorter than the Mean Time Between Failures (MTBF). Hence we use this approach for most quantities that depend directly on the geometry and basis set. However, the evaluation of the two-electron integrals is far too time-consuming and cannot be repeated as part of the recovery strategy. For these quantities we rely on keeping the size of the errors within limits using techniques described below. The remainder of the error has to be polished away using the iterative nature of the Hartree−Fock algorithm.

**4.2. Detecting and Correcting Errors in Distributed Data Structures.** The distributed data structures used in our calculation are partitioned across all processes in the calculation. As a result, there is only one copy of the data structure in the entire system and therefore a quorum based approach cannot be used. A useful methodology for detection of soft errors in distributed data structures is by using numerical assertions. These assertions may take the form of conditions or bounds on the valid values in these data structures. Obviously when exact conditions are known they may be used to remedy errors with high levels of confidence in the outcome. Instead when only bounds on the correct values are known the best that can be achieved is to detect outliers and replace those values with "reasonable" ones. An average of the class of values considered is a good example of what could be considered "reasonable". Such a strategy of course leaves the calculation subject to residual errors, and it relies on the self-correcting nature of optimization algorithms to eliminate those errors in the end. What assertions can be used depends highly on the algorithm under consideration. Also there may be multiple assertions that can be used with different trade-offs between the time complexity and reliability of the overall algorithm that can be achieved. Below the assertions we have implemented as well as the associated recovery mechanisms are discussed. The discussion starts with quantities for which exact conditions are known, i.e., the orthonormality of the orbitals, the normalization of the density matrix, and the unitarity of the orbital transformation matrix.

*Exact Condition for Detecting Orbital Orthonormality Violations and Recovery.* In the case of the Hartree−Fock algorithm exact conditions are available for the orthonormalization, the density matrix, and the matrix exponential. For the orthonormality the following condition has to be satisfied

$$\sum_{\mu\nu} C_{\mu i} S_{\mu\nu} C_{\nu j} - \delta_{ij} = 0 \tag{9}$$

This test can be implemented using two matrix−matrix multiplications, an addition to the diagonal elements, and a Frobenius norm evaluation. The overall cost of this test is $2O(N^3) + O(N^2)$ where $N$ is the dimension of the matrices. If the result does not match the required accuracy the vectors stored in the columns of $C$ can be reorthonormalized using a repeated Gramm−Schmidt algorithm at a cost of $O(N^3)$. The cost of the recovery is relatively high if a straightforward modified Gramm−Schmidt algorithm is used. The cost can be reduced based on the assumption that perturbations occur in a specific place. If vector $j$ is perturbed then the row and column corresponding to $j$ in eq 9 will be nonzero. In that case only that vector needs to be reorthonormalized thus lowering the recovery cost to $O(N^2)$. This leaves the error detection being the most expensive part of managing soft-errors. However, the impact of even small errors in the orthonormality on the outcomes of the calculation, as was found in Section 3.3, leaves no room for less accurate but lower cost detection algorithms.

*Detecting Failures in the Orbital Transformation and Recalculation for Recovery.* The same condition as in the previous section can be used to detect failures of the orbital transformation. If after the orbital transformation the orbitals are found not to be orthonormal the program can automatically trigger a reorthonormalization. This step is part of the operation of the original code as a means to control the impact of accumulating numerical errors. This operation costs $O(k{\cdot}N^3)$ where $k$ is a scalar. Instead, the matrix−matrix multiplication needed to perform the orbital transformation costs only $O(N^3)$. Hence it is more efficient to recalculate the transformed orbitals than to recover from this violation by reorthonormalization.

*Exact Condition for Density Matrix Corruption and Conditional Recalculation.* The density matrix is a representation of the electron distribution in a chemical system. This implies that the number of electrons can be obtained from this

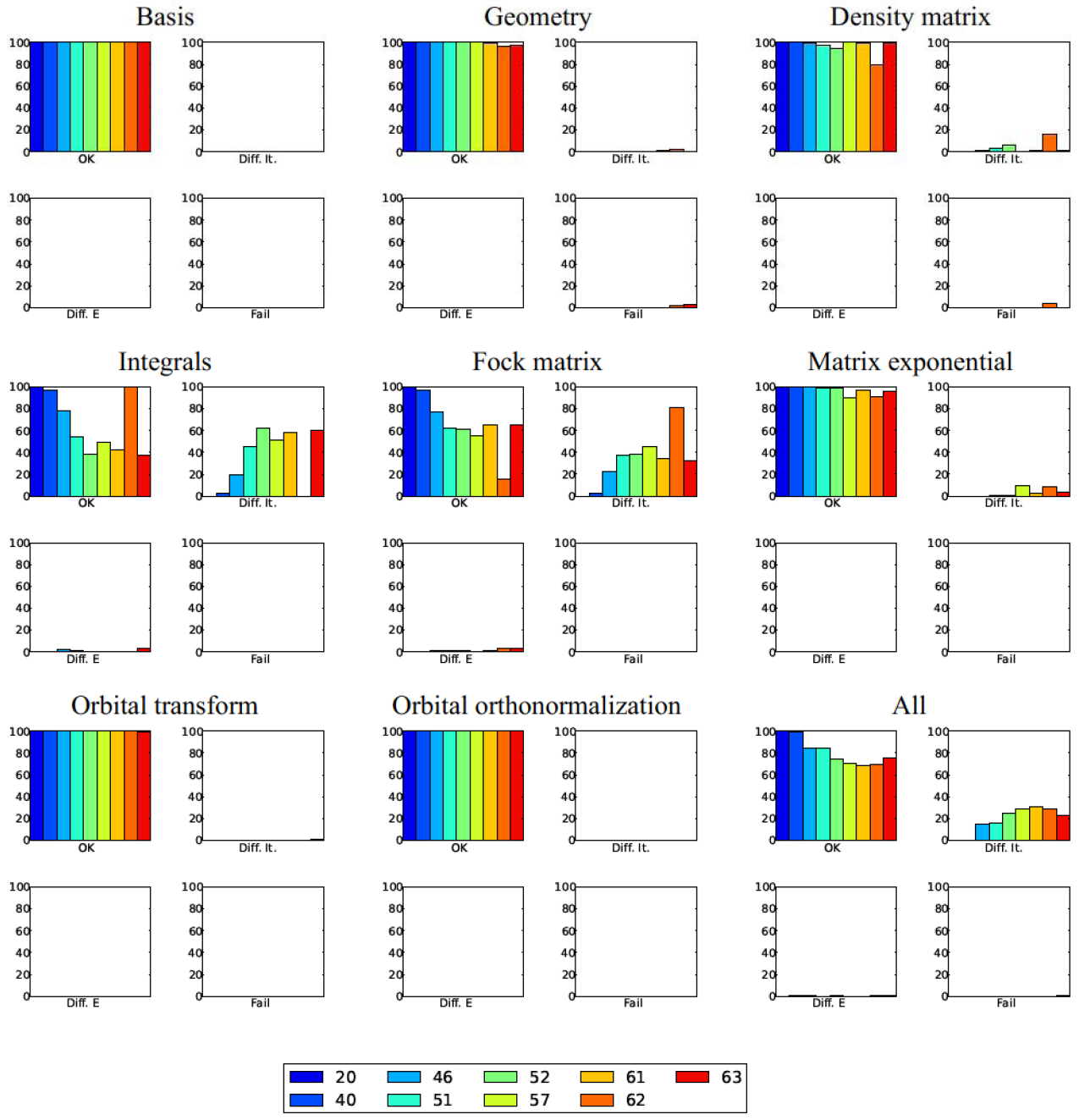



**Figure 8.** Residual impact of bit-flips after error correction in different bits for a variety of terms in the Hartree−Fock method using the direct two-electron integral algorithm. The labels on the squares have the same semantics as in Figure 3.

quantity, or equivalently that the density matrix is normalized such that the number of electrons can be calculated as

$$n = \sum_{\mu\nu} S_{\mu\nu} D_{\nu\mu} \tag{10}$$

at a cost of $O(N^2)$. An alternative approach would be to check the density matrix elements against their bounds. However, this approach would be equally expensive to perform and be less accurate than using eq 10. If the result is inaccurate the most straightforward approach is to recalculate the density matrix using a matrix−matrix multiplication at a $O(N^3)$ cost.

*Combined Bounds and Exact Conditions To Obtain a Unitary Orbital Transformation.* The aim of the matrix exponential is to compute a unitary transformation. The characteristic of such a matrix is that

$$U^T U = U U^T = I \tag{11}$$

In NWChem the matrix exponential is approximated by a third order Taylor series. Subsequently the approximately unitary matrix is optimized with an iterative approach using

$$U_{i+1} = U_i^*(1.5I - 0.5U_i^T U_i) \tag{12}$$

Testing whether $U$ is unitary is an inherent part of this algorithm and therefore is essentially free. However, because $U$ is initially not strictly unitary this test does not allow the presence of soft errors to be detected. For that purpose one should realize that the values of the matrix elements of $U$ are

bounded by $-1 \le u_{kl} \le 1$. Values outside this range must relate to soft errors. Checking for this requires $O(N^2)$ operations. The recovery strategy depends on which matrix element is affected. If the matrix element is on the diagonal then the faulty value is replaced by 1, and otherwise it is replaced by 0. The cost of this operation is $O(1)$ but subsequently additional iterations are needed to achieve unitarity. Hence, the true cost of recovery is expected to be $O(N^3)$ for the additional matrix−matrix multiplications required.

*Bounds on the Two-Electron Integrals.* There are no known exact conditions that the two-electron integrals have to satisfy. Instead only bounds on their values are provided by the Gauchy−Schwarz inequality which states

$$(ij|kl) \le \sqrt{(ij|ij)(kl|kl)} \tag{13}$$

In practice this inequality is used already to avoid calculating integrals of insignificant value. Here we extend this use to also test the values of the integrals obtained for soft errors. In the actual calculation the complete set of integrals is broken up into blocks where each block corresponds to a subrange of values for the labels *i*, *j*, *k*, and *l*. Because not all integrals are calculated the cost of checking the integrals lies in the range $O(N^2)$ to $O(N^4)$ depending on the length scales of the molecule. If an error is detected an efficient way to remedy it is to recalculate the block of integrals at a cost of $O(n^4)$. Here $n$ is the size of the label subrange. As $n$ is small relative to $N$ the cost of recalculating a block of integrals is essentially insignificant.

Obviously because the Gauchy−Schwarz inequality only bounds the integral values not all errors can be detected. This is particularly problematic in the conventional Hartree−Fock algorithm where remaining errors will persist throughout the calculation. This problem adds to the case for using integral direct approaches in addition to data storage and movement issues associated with the conventional approach.

Finally, the compressed set of Gauchy−Schwarz integrals $\sqrt{(ij|ij)}$ could suffer from errors. In our code these integrals are currently stored in a replicated fashion and therefore could be protected the same way as the basis set and the geometry. However, we have not considered perturbations of this data in this work.

*Restraints on the Fock Matrix Elements.* The final quantity to consider is the Fock matrix. For this quantity no sharp bounds are known. One could of course consider the largest integral and multiply this with the number of electrons to obtain a firm upper bound given by

$$I = \max_{ijkl}(ij|kl) \tag{14}$$

$$H = \max_{ij} h_{ij} \tag{15}$$

$$D = n_e \tag{16}$$

$$|F_{ij}| \le 2*H + 2*I*D \tag{17}$$

where $n_e$ is the number of electrons. However, this bound is certainly not sharp. As an alternative we can consider that as the calculation progresses the Fock matrix converges to its final value. Obviously there is no guarantee that the values in the Fock matrix converge in a particular direction; they might both increase or decrease. However, they should not undergo dramatic changes from one iteration to the next. Based on this we can construct a heuristic limit

$$|F_{ij}^{k+1}| \le \max(1/2,\ 10|F_{ij}^{k}|) \tag{18}$$

In applying this limit the value 1/2 allows some flexibility even if the corresponding value in the Fock matrix of the previous iteration is 0. The factor 10 ensures that the Fock matrix elements can grow relative to the previous iteration. If Fock matrix elements move outside of these bounds we truncate them to the boundary value. This approach, combined with the knowledge that soft errors will arise in different locations from iteration to iteration, is likely to control the propagation of errors to a reasonable degree. The cost of applying the bounds checking is $O(N^2)$ whereas the correction costs $O(1)$. There is an additional cost however in that we need to keep the current corrected Fock matrix as a reference for the next iteration. This imposes an additional $O(N^2)$ memory cost.

All of the above error correction mechanisms have been implemented in the Hartree−Fock code together with controls that allow each mechanism to be triggered independently. In the next section we consider how well this code fares when subjected to the same error insertions we applied earlier.

## 5. PERFORMANCE EVALUATION WITH SOFT ERRORS AND ERROR CORRECTION

The calculations reported on in this section were run in the same way as the calculations in Section 3.3 except that the error correction mechanisms discussed above are used attempting to recover from errors. In addition the integral direct algorithms were used for the Fock matrix construction. This choice is based on the improved inherent resiliency of the direct algorithms as well as the fact that these methods are likely to be preferred for calculations of the scale for which soft error corrections are important. The results from these calculations are summarized in Figure 8. The choice of the integral evaluation approach affects the response of the code to errors only when the inserted errors perturb the integral values, i.e., the results of calculations perturbing the integrals only and calculations perturbing all terms. The impact of this choice and the associated difference in the persistence of small errors is highlighted in Figure 9.

A first observation is that the results clearly highlight that the correction of the replicated data structures is highly successful. This is not unexpected as many sources for correct copies of these data structures are available, and the error detection is highly reliable. Hence successful error recovery should be the

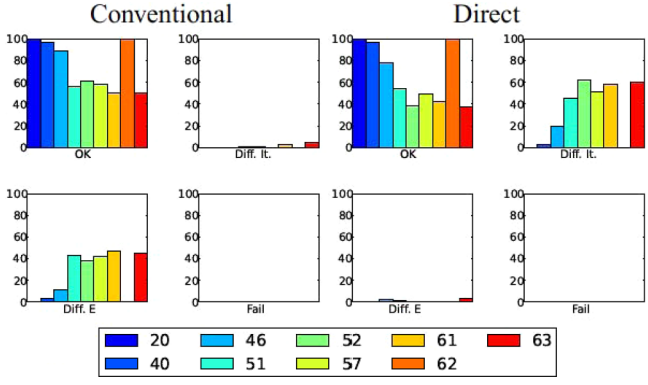



**Figure 9.** Residual impact of bit-flips conventional integral vs direct integral approches in the Hartree−Fock algorithm after error correction. The labels on the squares have the same semantics as in Figure 3.


dx.doi.org/10.1021/ct400489c | J. Chem. Theory Comput. 2013, 9, 3995−4005

norm. Nevertheless, in a few case the data related to perturbations in the geometry show that calculations still failed. These events are related to the fact that the two-electron integrals are not recalculated after an error in the geometry was found because the cost of doing so would be excessive. However, this does leave a chance that the calculation might be significantly perturbed by an unfortunate error. In principle even these errors can be addressed, but that requires a much more frequent testing of the correctness of the geometry data at which point the performance of the program might be unacceptably affected.

The success rates for the error correction in the distributed data structures are not as good as for the replicated data structures. For the quantities for which exact conditions are available such as the orthonormality of the orbitals, the transformed orbitals, the normalization of the density matrix, and the matrix exponential, the errors are essentially perfectly recovered. The only exception is in the matrix exponential where the recovery mechanism guarantees that a unitary matrix is produced, but this matrix may not correspond entirely to the correct rotation. Hence the calculations occasionally take more cycles to converge. Nevertheless, failures of the calculations are avoided.

The error recovery from perturbations in the two-electron integrals is noticeably less successful than the recovery in the orbital transformation. Comparing the results from the conventional and direct calculations as presented in Figure 9 shows some clear differences between the two approaches. Because the Gauchy−Schwarz inequality cannot detect all errors the integrals stored in the conventional approach may contain small errors. As the same integrals are used in every iteration it is essentially unavoidable that these errors are ultimately reflected in the final result. By contrast in the direct approach the large errors are corrected successfully and the remaining small errors only affect the calculation during one iteration. Hence they do not introduce a systematic perturbation in the calculation, and ultimately the correct results are obtained although sometimes at the cost of additional iterations.

Finally, the recovery in the Fock matrix construction is the most challenging. In particular as no rigorous bounds on this quantity are available the heuristic restraints applied have to be rather loose which may leave relatively large errors undetected. In most cases restraining the change in the Fock matrix is sufficient to successfully recover from errors albeit at the cost of additional iterations. However, in some cases the residual error in the Fock matrix causes the calculation to converge to an incorrect result.

Overall the results show that the resilience of the Hartree−Fock method can be significantly enhanced using knowledge of the quantities involved to detect and correct large errors. In most cases the optimizing nature of the algorithm is sufficient to eliminate the undetectable residual errors.

## 6. CONCLUSIONS

The resilience of the Hartree−Fock method to perturbations introduced by intermittent incorrect behavior of a computing platform has been investigated. It was shown that the resulting soft errors cause calculations to fail in a significant fraction of the cases even though the Hartree−Fock method by its nature would be expected to be relatively resilient to such errors. Using knowledge about the data structures involved in such calculations conditions, bounds and restraints can be defined that allow large errors to be detected and corrected. The remaining residual errors are small enough in the majority of cases that they are eliminated in the normal execution of the optimization.

## APPENDIX A: RANDOM NUMBER SEED COMPUTATION

To ensure that every test run the calculation is subject to different errors being inserted the pseudo random number generators need to be seeded in a unique way. The seeds need to be unique from one calculation to the next. They also need to be unique for every processor. To ensure these criteria are met the seeds are calculated from the time at which the calculation was run as well as the rank of the processor. The time information was obtained from the library routine date_and_time which produces a string, for example,

```
20130430170113.495
```

where the first four digits refer to the year followed by two digits for the month and the date, respectively. This is followed by the time with two digits for the hour (in a 24 hour clock representation), the minute, and finally the seconds with a millisecond granularity.

Each digit in the string is converted to an integer based on the character number in the ASCII character set, and a polynomial of the values is computed. To guard against overflows the value is truncated using the mod function. That is, the following expression is evaluated

$$s(0) = 1 \tag{19}$$

$$s(i) = \mathrm{mod}(s(i-1)*(1 + ichar(string(i{:}\ i))),\ 2^{54}) + 1 \tag{20}$$

where the expression is given as a recursive relationship. The final value $s(18)$ is combined with the processor rank as

$$seed = \mathrm{mod}((1 + \mathrm{mod}(s(18),\ 2^{32} - 1))*(1 + 2683*p),\ 2^{32} - 1) + 1 \tag{21}$$

where $p$ is the processor rank ranging from 0 to the number of processors minus one. The use of the mod function ensures that the seed value can be represented in a four byte number.

## AUTHOR INFORMATION


### Corresponding Author

*E-mail: hubertus.vandam@pnnl.gov (H.J.J.v.D.); abhinav.vishnu@pnnl.gov (A.V.); wadejong@lbl.gov (W.A.d.J.). Phone: +1 509 372 6441. Fax: +123 (0)123 4445557.

### Present Address

†Lawrence Berkeley National Laboratory, 1 Cyclotron Road, Berkeley, CA 94720-8297, USA (W.A.d.J.).


### Notes

The authors declare no competing financial interest.

## ACKNOWLEDGMENTS

This work was supported by the eXtreme Scale Computing Initiative at Pacific Northwest National Laboratory. Pacific Northwest National Laboratory is operated for the U.S. Department of Energy by Battelle. This work was done in part using EMSL, a national scientific user facility sponsored by the Department of Energy's Office of Biological and Environmental Research and located at Pacific Northwest National

Laboratory, operated for the U.S. Department of Energy by Battelle under contract DE-AC05-76RL01830.

## REFERENCES

(1) Joubert, W.; Su, S.-Q. An analysis of computational workloads for the ORNL Jaguar system. *Proceedings of the 26th ACM international conference on Supercomputing*, Venice, Italy; ACM: New York, NY, USA, 2012; pp 247−256.
(2) TOP 500 Supercomputer Sites. http://www.top500.org (accessed July 16, 2013).
(3) Geist, A.; Dosanjh, S. *Int. J. High Perform. Comput. Appl.* **2009**, *23*, 401−402.
(4) van Dam, H. J. J.; Vishnu, A.; de Jong, W. A. *J. Chem. Theory Comput.* **2011**, *7*, 66−75.
(5) Cappello, F.; Geist, A.; Gropp, B.; Kale, L.; Kramer, B.; Snir, M. *Int. J. High Perform. Comput. Appl.* **2009**, *23*, 374−388.
(6) Rebaudengo, M.; Sonza Reorda, M.; Torchiano, M.; Violante, M. Soft-error detection through software fault-tolerance techniques. In *DFT '99*, Proceedings of the 14th International Symposium on Defect and Fault-Tolerance in VLSI Systems, Albuquerque, NM, Nov 1−3, 1999; IEEE Computer Society: Washington, DC, 2002; pp 210−218.
(7) Bronevetsky, G.; de Supinski, B. Soft error vulnerability of iterative linear algebra methods. *Proceedings of the 22nd annual internatioanal confrence on supercomputing*, Island of Kos, Greece; ACM: New York, NY, USA, 2008; pp 155−164.
(8) Du, P.; Luszczek, P.; Dongarra, J. High performance dense linear system solver with soft error resilience. *CLUSTER* **2011**, 272−280.
(9) Vishnu, A.; Van Dam, H.; De Jong, W.; Balaji, P.; Song, S. Fault-tolerant communication runtime support for data-centric programming models. Presented at *2010 International confrence on High Performance Computing (HiPC)*, Cidade de Goa, Goa, India; IEEE: 2010; pp 1−9.
(10) Vishnu, A.; Mamidala, A. R.; Narravula, S.; Panda, D. K. Automatic Path Migration over InfiniBand: Early Experiences. *Parallel and Distributed Processing Symposium*, 2007; IEEE International; pp 1−8, 26−30 March 2007; doi: 10.1109/IPDPS.2007.370626.
(11) Vishnu, A.; Krishnan, M.; Panda, D. K. An efficient hardware-software approach to network fault tolerance with InfiniBand. *CLUSTER* **2009**, 1−9.
(12) Vishnu, A.; Gupta, P.; Mamidala, A. R.; Panda, D. K. A software based approach for providing network fault tolerance in clusters with uDAPL interface: MPI level design and performance evaluation. *SuperComputing* **2006**, 85−96.
(13) Wong, A. T.; Harrison, R. J. *J. Comput. Chem.* **1995**, *16*, 1291−1300.
(14) Standard for Floating-Point Arithmetic; IEEE Standard 754-2008, 2008.
(15) Hehre, W. J.; Ditchfield, R.; Pople, J. A. *J. Chem. Phys.* **1972**, *56*, 2257−2261.
(16) Hariharan, P.; Pople, J. *Theor. Chim. Acta* **1973**, *28*, 213−222.
(17) Rivest, R. *The MD5Message Digest Algorithm*; Request for Comments RFC 1321, 1992.
(18) Turner, S.; Chen, L. *Updated Security Considerations for the MD5Message-Digest and the HMAC-MD5 Algorithms*; Request for Comments RFC 6151, 2011.