Evaluation of two-electron repulsion integrals over Gaussian basis functions on SRC-6 reconfigurable computer

Volodymyr Kindratenko[1], Ivan Ufimtsev[2], and Todd Martínez[2]
1) National Center for Supercomputing Applications, 2) Department of Chemistry
University of Illinois at Urbana-Champaign, Urbana, IL 61801
1) kindr@ncsa.uiuc.edu, 2) iufimts2@uiuc.edu, tjm@spawn.scs.uiuc.edu

Abstract

We demonstrate an implementation of the two-electron repulsion integrals code for the direct self-consistent field calculations on a reconfigurable computer. We analyze different strategies and optimization techniques necessary to port the code to SRC-6 reconfigurable computer and provide performance results for models using relatively uncontracted and highly contracted basis sets. Our implementation achieves an order of magnitude performance improvement over the conventional microprocessor system when using highly contracted basis sets and only a factor of 2 performance improvement for models that use relatively uncontracted basis sets. We also point out limitations of the SRC-6 reconfigurable computer that prevent using it for running ab initio *quantum chemistry applications.*

1. Introduction

Two-electron repulsion integrals remain the bottleneck in many of the *ab initio* molecular orbital (MO) or density functional theory (DFT) electronic structure codes. For example, in direct self-consistent field (SCF) methods many millions of electron repulsion integrals are recomputed every SCF iteration and count for the vast majority of the execution time. Accelerating the calculation of two-electron repulsion integrals has been a subject of several recent studies with most notable results obtained on the graphical processing units (GPUs) [1]. In this study, we evaluate the suitability of reconfigurable computing (RC) based on field programmable gate arrays (FPGAs) for evaluation of (*ss*|*ss*) integrals over contracted *s*-orbitals. We start with a reference microprocessor implementation of the (ss|ss) integrals and implement it to run on SRC-6 MAP Series E processor using SRC Carte development tools. We compare performance of our SCR-6 implementation with the performance of the reference C implementation and point out limitations of the FPGA-based systems for computing two-electron repulsion integrals.

2. Two-electron repulsion integrals

Our goal is to calculate all two-electron repulsion integrals for a given set of basis shells. The two-electron repulsion integrals over contracted basis functions can be computed as

$$(\mu\nu|\lambda\sigma) = \sum_{p=1}^{N_\mu}\sum_{q=1}^{N_\nu}\sum_{r=1}^{N_\lambda}\sum_{s=1}^{N_\sigma} d_{\mu p}\, d_{\nu q}\, d_{\lambda r}\, d_{\sigma s}\, [pq|rs]$$

where N_* is the contraction length, d_{**} are contraction coefficients, and $[pq|rs]$ are integrals evaluated over primitive basis functions. Rys quadrature scheme for a two-electron Coulomb repulsion integral is used to evaluate primitive integrals $[pq|rs]$ for Gaussian-type orbitals (GTO) basis sets [2]. In this initial implementation we only consider (*ss*|*ss*) integrals over contracted *s*-orbitals. The general formula for primitive [*ss*|*ss*] integrals is as follows [3]:

$$[s_1 s_2|s_3 s_4] = \frac{\pi^3}{AB\sqrt{A+B}} K_{12}(\vec{\mathbf{R}}_{12}) K_{34}(\vec{\mathbf{R}}_{34}) F_0\left(\frac{AB}{A+B}[\vec{\mathbf{R}}_{\mathbf{P}} - \vec{\mathbf{R}}_{\mathbf{Q}}]^2\right)$$

where α_k is the exponent and $\vec{\mathbf{R}}_{\mathbf{k}}$ is the atomic center of the k^{th} primitive basis function in the integral and

$$A = \alpha_1 + \alpha_2,\ B = \alpha_3 + \alpha_4,\ F_0(t) = \frac{erf(\sqrt{t})}{\sqrt{t}},$$
$$\vec{\mathbf{R}}_{\mathbf{kl}} = \vec{\mathbf{R}}_{\mathbf{k}} - \vec{\mathbf{R}}_{\mathbf{l}},\ \vec{\mathbf{R}}_{\mathbf{P}} = \frac{\alpha_1\vec{\mathbf{R}}_{\mathbf{1}} + \alpha_2\vec{\mathbf{R}}_{\mathbf{2}}}{A},\ \vec{\mathbf{R}}_{\mathbf{Q}} = \frac{\alpha_3\vec{\mathbf{R}}_{\mathbf{3}} + \alpha_4\vec{\mathbf{R}}_{\mathbf{4}}}{B},$$
$$K_{ij}\left(\vec{\mathbf{R}}_{ij}\right) = exp\left(-\frac{\alpha_i\alpha_j}{\alpha_i+\alpha_j}\left[\vec{\mathbf{R}}_i - \vec{\mathbf{R}}_j\right]^2\right)$$

3. Reference C implementation

Reference C implementation of the two-electron repulsion integrals evaluation code is straightforward: The four outer loops sequence through all unique combinations of electron shells:

```
for (s1 = 0; s1 < totNumShells; s1++)
   for (s2 = s1; s2 < totNumShells; s2++)
      for (s3 = s1; s3 < totNumShells; s3++)
         for (s4 = s3; s4 < totNumShells; s4++)
```

For each such combination, the four inner loops sequence through all shell primitives:

```
for (p1 = 0; p1 < numPrimitives[s1]; p1++)
   for (p2 = 0; p2 < numPrimitives[s2]; p2++)
      for (p3 = 0; p3 < numPrimitives[s3]; p3++)
         for (p4 = 0; p4 < numPrimitives[s4]; p4++)
```

Inside these four nested loops, primitive [*ss*|*ss*] integrals are computed via the above equations and are summed as follows:

```
H_ReductionSum[s1,s2,s3,s4] += sqrt(F_PI * rho) *
      I1*I2*I3*weight * Coeff1*Coeff2*Coeff3*Coeff4;
```

In this reference implementation, contracted integrals are stored in memory for the follow-up use for constructing the Fock matrix necessary to solve the electronic time-independent Schrodinger equation. We stop short of solving this equation as our main goal is to speed up the calculation of the two-electron repulsion integrals.

4. SRC-6 reconfigurable computer

The SRC-6 MAPstation used in this work consists of a commodity dual-CPU Intel Xeon board and one MAP Series E processor interconnected with a 1.4 GB/s low-latency Hi-Bar™ switch [4]. The SNAP™ Series B interface board is used to connect the CPU board to the Hi-Bar switch.

The MAP Series E processor contains two user FPGAs, one control FPGA, and memory. There are six banks of on-board memory (OBM); each bank is 64 bits wide and 4 MB deep for a total of 24 MB. There is an additional 4 MB of dual-ported memory used for data transfer between the two FPGAs. The two user FPGAs in the MAP Series E are Xilinx Virtex-II Pro XC2VP100. The FPGA clock rate of 100 MHz is set from within the SRC programming environment.

Code for SRC MAPstation is written in the MAP C programming language using the Carte programming environment [5]. The Intel C (icc) compiler is used to generate the CPU-side of the combined CPU/MAP executable. The SRC MAP C compiler produces the hardware description of the FPGA design for our final, combined CPU/MAP target executable. This intermediate hardware description of the FPGA design is passed to Xilinx ISE place and route tools to produce the FPGA bit file. Finally, the linker is invoked to combine the CPU code and the FPGA hardware bit file(s) into a unified executable.

5. SRC-6 implementation

While the reference C implementation of the two-electron repulsion integrals code is straightforward, porting it to the SRC-6 reconfigurable processor is a challenging task. The challenge comes from the deeply nested structure of the code and the need for a large number of floating-point operations. Note that similarly to [1] we restrict ourselves to using only single-precision floating-point arithmetic.

5.1. Nested loops pipelining

The Carte compiler attempts to pipeline only the innermost loop. Even though all the arithmetic operations in the two-electron repulsion integral code can be enclosed in the innermost loop, simply pipelining it is not sufficient to achieve good performance. Since the innermost loop resides deep inside seven other nested loops, it will be invoked a significant number of times, thus accumulating execution overhead due to the loop pipeline depth. The pipeline depth of the loop body, measured in number of clock cycles, is substantial due to the large number of sequentially executed arithmetic operations. Moreover, for relatively uncontracted basis sets, the innermost loop frequently has only a few iterations, thus providing very poor amortization of the pipeline depth overhead.

A common approach to deal with the nested loop inefficiencies is to combine several such loops into one that can be pipelined by the compiler. Ideally, one would want to combine as many—desirably all—loops as possible. Combining all eight nested loops is not an option in our case for the following reason: In order to contract the primitive integrals in a manner that will allow loop pipelining, we need to use the floating point macro, *fp_accum_32*, available in the Carte floating point library. This macro does not provide the final sum until we exit the pipelined loop. However, we need to store multiple contracted integrals as we compute them, thus requiring access to the summed primitive integrals at will.

A close examination of the code reveals that we can combine just four of the innermost loops since they are responsible for computing a single contracted integral value. The fused loop will have *nPrims[s1] * nPrims[s2] * nPrims[s3] * nPrims[s4]* iterations and will compute a single *H_ReductionSum[s1,s2,s3,s4]* value. Computing *p1-p4* indexes for the four innermost nested loops is done using *cg_count_ceil_32* Carte macro in the usual way [5].

The resulting fused loop will still require four clock cycles per single loop iteration to complete – not an optimal implementation. The four clocks per iteration are needed to access the coordinates of the four atoms needed to compute the primitive integral. Since these coordinates are stored in a single-ported array, the compiler adds delays necessary to access the required data before all the calculations can be done. This brings us to the data storage challenge.

5.2. Data storage

The data storage schema used in the reference C implementation is not appropriate for the MAP Series E implementation due to the OBM and BRAM memory layout and the limited amount of memory accessible from FPGA. Therefore, before we can transfer the data to the MAP Series E processor, we need to re-shape it on the CPU.

First, we copy all atom center coordinates into a 1D array and pad triplets of coordinates of each individual atom to 64 bits–the storage unit size of the OBM banks. We then transfer this array to the MAP Series E processor and stripe it across two OBM banks. These atom coordinates, stored in two on-board memory banks, will need to be accessed four times per loop iteration, thus leading to the loop slowdown by four clock cycles as discussed above.

Similarly, basis shell primitives, which are already aligned to 64 bits, are transferred and stored in a single OBM bank. They too will need to be accessed 4 times per the fused loop iteration.

In order to eliminate the fused loop slowdown due to the OBM access delays, we make four identical copies of the atom coordinates and shell primitives to BRAM memory located directly in the FPGA and access data from BRAM instead of the original OBM banks. Thus, at this point our fused loop becomes fully pipelined with one clock cycle per single loop iteration. An implementation with BRAM allocated for 128 basis shells and 128 atoms uses up only 8% of the BRAM memory available on XC2VP100 FPGA.

In addition to raw atom locations and shell primitives, we need per-'atom shell' descriptors that identify where atom coordinates and basis shell primitives are stored for a given atom shell. These descriptors are assembled on the CPU, padded to 64 bits, transferred and stored in a single OBM bank. This data is accessed once in each of the outer loops. However, since the outer loops are not pipelined, there is no need to duplicate the data for efficient access.

5.3. Code optimization

In the reference C implementation, Gauss error function, *erf*, is computed by expanding the integrand in a Taylor series. In the general case, its evaluation requires up to 12 floating-point multiplication operations, up to 14 floating-point addition operations, floating-point division, exponent, and square root. The exact polynomial used in evaluating this function depends on the value of *t*; in its current implementation sufficient for *s*-orbitals evaluation [6] it consist of eight special cases resulting in eight branches with 53 floating-point multiplication operations alone. The rest of the kernel contains 41 floating-point multiplications, 12 additions, and four division operations as well as three exponent and two square root operations. As such, this code poses a significant challenge for FPGA implementation.

We restructured the Gauss error function implementation to eliminate the calculations in eight individual branches. Instead, we pre-compute all commonly used parameters and for each of the eight cases fill in the corresponding polynomial coefficients and then apply a common equation at the end. As the result, we were able to reduce the 53 floating-point multiplication operations to just 12. Still, this implementation does not fit on a single chip and more aggressive optimizations are needed.

Once such optimization is to use the smaller area floating-point library that is provided with the Carte development environment. Floating-point operations implemented in this library require less logic to implement, but provide lower accuracy. Another technique is to divide the calculations between the two FPGAs available in the MAP Series E processor. In particular, $d^2\,AB/(A+B)$, *erf* function, part of the final equation, contracted integrals summation, and data storage code were moved to the secondary FPGA. Data exchange between the primary and secondary FPGAs is implemented via cross-FPGA bridge streams.

The final dual-FPGA implementation occupies all slices on the primary FPGA, over 60% of slices on the secondary FPGA, and about 30% of the hardware multipliers available on two chips. Pipeline depth of the fused loop on the primary chip is 209 clock cycles and additional 353 clock cycles on the secondary chip.

6. Results and discussion

We consider two test cases: a molecular system consisting of 10 water molecules (30 atoms in total) using cc-pVDZ basis set with only *s*-type functions taken into account and a molecular system composed of 64 hydrogen atoms arranged in a lattice using the STO-6G basis set. The first model is an example of a relatively uncontracted basis set whereas the second model is an example of a highly contracted basis set.

Table 1 summarizes performance results obtained while executing our reference C implementation and the SRC-6 Map Series E processor implementation. An obvious observation is that the speedup obtained for the model that uses a highly contracted basis set is substantially higher than for the model that uses a relatively uncontracted basis set. The degree at which the basis set is contracted translates into the number of primitive integrals to be computed for each contracted integral. In terms of the code execution profile, this corresponds to the number of iterations of the fused innermost loop. In case of the highly contracted basis set, the innermost loop has a constant number of iterations per each invocation: 1,296. In the case of the relatively uncontracted basis set, the number of iterations of the fused loop varies from 4,096 to just 1, or 168 iterations on average. Since the overall execution time of this code is proportional to $N(n+p)$ where N is the number of reduction elements, n is the number of innermost loop iterations, and p is the loop pipeline depth, the case of the highly contracted basis set, or when $n > p$, makes a much better use of the innermost loop pipeline. In other words, pipeline depth overhead is better amortized for the loops that have more iterations.

	Model 1	Model 2
# of atoms	30	64
Basis set	cc-pVDZ	STO-6G
# of integrals	528,569,315	2,861,464,320
# of reduction elements	3,146,010	2,207,920
SRC-6 host (sec)	70.55	518.90
SRC-6 MAP E (sec)	25.42	42.85
Speedup	2.8x	12.1x

Table 1. Performance results for two datasets.

We also observe that in many cases numerical values of the integrals computed on the MAP Series E processor differ significantly from the results obtained while executing code on the CPU. We observe that values of integrals that fall between 0.0001 and 0.1 are the same or very close to those computed on the CPU. However, integral values smaller than 0.0001 (as computed on the CPU) are almost always "inverted" to appear as $X \cdot 10^{+34}$ or $X \cdot 10^{+36}$. A plausible explanation for this may be the use of the smaller area floating-point arithmetic library, which is less accurate than the standard floating-point library, but the use of which was necessary because of the limited FPGA resources.

7. Conclusions

Reconfigurable computing holds the potential for an order of magnitude speedup for the *ab initio* electronic structure codes. However, current systems, such as SRC-6, are not yet capable of fully supporting such applications due to the limited size of FPGAs and the limitations in floating-point libraries. The limited FPGA size translates into an inability to implement fully featured codes even for the simplest type of integrals. The floating-point library limitations do not allow the implementation of a fully optimized code.

The SRC-7 MAP Series H processor holds promise to overcome the space limitation as its Altera Stratix II EP2S180 FPGAs are substantially larger than those used in MAP Series E processors.

8. Acknowledgement

This work is sponsored by the National Science Foundation (CHE-06-26354). Special thanks to Trish Barker from NCSA's Office of Public Affairs for help in preparing this publication.

9. References

[1] Ufimtsev, I.S. and Martinez, T.J., Quantum Chemistry on Graphical Processing Units. 1. Strategies for Two-Electron Integral Evaluation, J. Chem. Theory Comput., 4, 2, 222 - 231, 2008

[2] Dupuis, M.; Rys, J.; King, H. F. Evaluation of molecular integrals over Gaussian basis functions. J. Chem. Phys. 65, 111, 1976

[3] Boys, S. F. Electronic Wave Functions. I. A General Method of Calculation for the Stationary States of Any Molecular System. Proc. R. Soc. London, Ser. A, 200, 542, 1950

[4] SRC Computers Inc., Colorado Springs, CO, SRC Systems and Servers Datasheet, 2005.

[5] SRC Computers Inc., Colorado Springs, CO, SRC C Programming Environment v 1.9 Guide, 2005

[6] Gordon, M. S.; Schmidt, M. W. Advances in electronic structure theory: GAMESS a decade later. In Theory and Applications of Computational Chemistry: the first forty years; Dykstra, C. E., Frenking, G., Kim, K. S., Scuseria, G. E., Eds.; Elsevier: Amsterdam, p 1167, 2005