Optimization of special functions for quantum chemistry

Vladimir Mironov, Alexander Moskovsky

Department of Chemistry, Lomonosov Moscow State University,
Leninskie Gory 1/3, Moscow, 119991, Russian Federation

Summary

Many steps in quantum chemical (QC) calculations use special functions, which are not included in the standard libraries. For example, Boys functions, abscissas, and weights of Rys polynomials are used in electron repulsion integral calculations. Computation of these special functions can be very expensive. Moreover, in many important cases of quantum chemistry calculations they are a major bottleneck. It happens because many QC programs utilize obsolete code for these functions because the code was written decades ago. In this study, we rewrote algorithms for calculation of the Rys polynomial abscissas and weights for modern hardware. We have found that using long polynomial expansions can improve the speed of the calculations ~10x on Intel Xeon Phi x200 and 2-3 times on Intel Xeon v4.

Introduction

Basics of quantum chemistry (QC)

- Adiabatic approximation – separation of motion of electrons and nuclei
- Atomic nuclei – classical charged particles, electrons – quantum particles
- Electronic Schrodinger equation:

$$H_e\Psi_e = E_e\Psi_e$$

$$H_e = \sum_{A,B}\frac{Q_AQ_B}{|R_A-R_B|} - \sum_i\frac{h^2}{2m_i}\nabla_i^2 - \sum_{A,i}\frac{Q_A}{|R_A-r_i|} + \sum_{i,j}\frac{1}{|r_i-r_j|}$$

Internuclear repulsion | Electronic kinetic energy | Electron-nuclear attraction | Interelectronic repulstion

Time complexity: $O(N^2)$ $O(N^4)$

Electrostatic interaction in QC

Gaussian-type orbitals

Gaussian-type orbitals (GTO) are widely used in quantum chemistry as a basis functions. Primitive GTO are functions of the type:

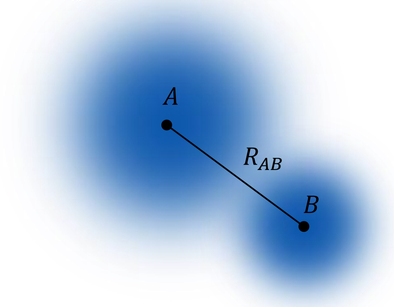
$$\chi(r) = (x-A_x)^{a_x}\left(y-A_y\right)^{a_y}(z-A_z)^{a_z}\,e^{-\alpha(r-A)^2} \quad (1)$$

Typically, linear combinations of primitive GTO sharing the same center and angular momentum (contracted GTO) are sued as a basis functions:

$$(i,j|k,l) = \sum_a^M\sum_b^N\sum_c^O\sum_d^P C_{ai}C_{bj}C_{ck}C_{dl}(ab|cd) \quad (2)$$

Electrostatic interaction of charged Gaussian clouds

$\left(\frac{p}{\pi}\right)^{3/2}\exp(-p(r-A)^2)$



$\left(\frac{q}{\pi}\right)^{3/2}\exp(-q(r-B)^2)$

- Simplest case – spherical Gaussian clouds:

$$E_{AB} = \iint_{-\infty}^{+\infty}\frac{\Phi_1(A,p,r_1)\Phi_2(B,q,r_2)}{|r_1-r_2|}dr_1dr_2,$$

$$\Phi(A,p,r) = \exp(-p(r-A)^2)$$

$$E_{AB} = \sqrt{\frac{4\alpha}{\pi}}\cdot\int_0^1\exp(-\alpha R_{AB}^2t^2)\,dt$$

$$\alpha = \frac{pq}{p+q}$$

- More complex cases – various kinds of electron repulsion integrals (ERIs) over Cartesian Gaussians:

$$E_{ABCD} = \iint_{-\infty}^{+\infty}\frac{\Phi_1(A,a,r_1)\Phi_2(B,b,r_1)\Phi_3(C,c,r_2)\Phi_4(B,d,r_2)}{|r_1-r_2|}dr_1dr_2$$

$$\Phi(A,p,r) = x^{l_x}y^{l_y}z^{l_z}\exp(-p(r-A)^2)$$

$$l_x+l_y+l_z = l - \text{ angular momentum}$$

ERI computation methods and
Type of special functions required

Gauss-like quadrature

Roots and weights of Rys polynomials:

Orthogonal polynomials class on $(0,1)$ with respect to weight function $\frac{\exp(-Tx)}{2\sqrt{x}}$

$T \in (0,\infty)$

Auxiliary function expansion

Boys functions:

$F_m(T) = \int_0^1\exp(-Tx^2)\,x^{2m}dx,$

$m = 0,1,2\ldots$ depends on angular momentum

$T \in (0,\infty)$

Characteristics of special functions in ERI calculation:

- Functions of single argument
- Required implementations for broad angular momentum cases .
- Required precision is high to match ERI precision at least 10^{-10}.
- High performance is needed for low angular momentum cases: in recent codes they take up to 80% of whole computation time

How special functions are calculated in QM software:

- Analytical expansion/numerical integration/recurrence use
  Most precise, computationally very expensive
- Piecewise interpolation
  Relatively fast, not feasible for whole range of arguments
- Asymptotical approximation at $T \to \infty$ or $T \to 0$
  Fast, not applicable for all values of argument

Results

Rys polynomial roots and weights calculation

- Needed implementations for Rys polynomial orders up to 13
- Numerical calculation: discretized Stieltjes procedure
- Asymptotically ($T \to \infty$) trends to roots and weights of Hermite polynomials

$$R_i(T) \xrightarrow{T\to\infty} C_i^R T^{-1},\ W_i(T) \xrightarrow{T\to\infty} C_i^W T^{-1/2}$$

- Piecewise approximation in terms of T, T^{-1} and $\exp(-T)$

Insigths for low angular momentum cases:

- Stieltjes procedure is too slow for $N_{roots} < 6$ ($\sum_{i=a,b,c,d} l_i < 10$): typically a combination of piecewise interpolation with asymptotical approximation is used
- Precision requirements are less strict: only few further multiplications/additions steps – accuracy of ~10^{-12} is enough.
- Asymptotic condition holds for roughly 50% of all practical cases.

Implementation design:

- Polynomial fitting for non-asymptotic cases
- Fit in Chebyshev basis, calculate in ordinary polynomial basis
- If beneficial – use rational polynomial (Padé) expansion: one division ↔ several FMA, higher parallelism, higher precision for substantially non-polynomial interpolants
- $(0,2^n) \to (0,1)$ range reduction for numerical stability: no need for an expensive division operation

Polynomial fitting:

- Very common approach for fast and precise approximation
  - Used in calculation of multiple transcendental functions
  - Spline fitting is a particular case
  - Pros: ×,+, FMA operations only; good pipeline utilization
- In most cases it is impossible to find single formula for the whole domain of target function – it is usually separated in several smaller intervals
- Problem of choice
  - few large intervals and high-degree polynomials – lot of clock cycles
  - multiple short intervals and low-degree polynomials – code optimization issues

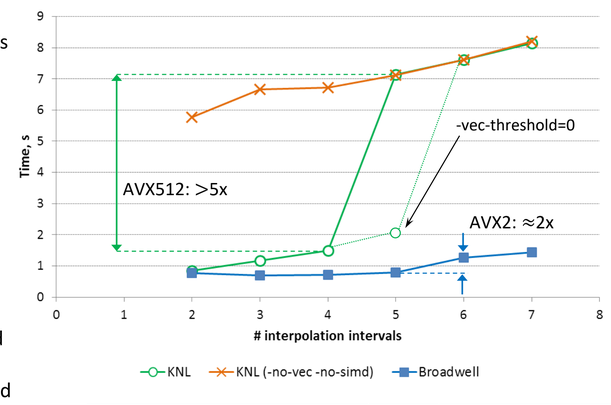
Polynomial evaluation schemes:

- Horner's method:
  - $S_n = (\ldots(a_nx + a_{n-1})x + a_{n-2})x + \cdots)x + a_0$
  - $FMA \to FMA \to FMA \to \cdots \to FMA$
  - Commonly used for generic polynomial evaluation
  - Only external parallelism is possible
- Estrin's method:
  - Binary tree-like evaluation scheme
  - $\frac{n}{2}\times FMA \to \frac{n}{4}\times FMA \to \cdots \to FMA$
  - Efficient utilization of SIMD parallelism
  - Best latency for $P_n(x), n = 2^k-1$

  Method of choice
- *Ad hoc* schemes:
  - Can be tuned for hardware
  - Extremely hard to generate even for small n
  - Performance gain over other schemes may be negligible

Selecting the number of intervals

- Workload:
  - 2x 12-order polynomials over $k-1$ intervals, 1 case is asymptotic expansion
  - random argument, 50% cases are asymptotical
  - 1 thread, N_{tests}=204800000
- Hardware:
  - Broadwell – Intel Xeon E5-2697A v4
  - KNL – Intel Xeon Phi 7250 (quadrant flat)
- Results:
  - (auto-)Vectorization works for up to 5 interpolation intervals on both Intel Xeon and Xeon Phi processors
  - AVX2 – better to use more interpolation intervals and short polynomials
  - AVX512 – better to use few interpolation intervals and long polynomials



Finding best interpolation scheme

- Precision requirements: $max.rel.err. \le 10^{-10}$
- Best evaluation scheme depends on the architecture and the order of Gauss-Rys quadrature
- What is important for performance:
  - Aggressive inlining
  - Interprocedural optimization
  - Vectorization

Implementation	Relative error	Time, s	
		Broadwell	KNL
N_{roots} = 1, original (8 "if" cases)	-	2.48	13.9
N_{roots} = 1, poly12 (7 "if" cases)	7.5E-13	1.13	7.46
N_{roots} = 1, poly32 (2 "if" cases)	1.2E-10	0.98	1.38
N_{roots} = 1 , Padé 11/11 (2 "if" cases)	3.1E-14	1.41	1.54
N_{roots} = 2, original (9 cases)	-	3.20	21.0
N_{roots} = 2, poly32 (3 "if" cases)	7.1E-14	2.81	4.40
N_{roots} = 2 , Padé 11/13 (2 "if" cases)	3.6E-11	1.85	3.83

Conclusion

- Intel Xeon CPU and Intel Xeon Phi x200 are still very different architectures
- AVX512 favors long polynomial expansion and few interpolation intervals
- Traditional piecewise approximations (including splines) should be used with care on Intel Xeon Phi. Intel Xeon CPUs are much less sensitive to the number of interpolation intervals
- Vectorization is crucial for Intel Xeon Phi to compete with Intel Xeon CPU

Acknowledgements

This work is supported by Intel Parallel Compute Center program. We thank Klaus-Dieter Oertel (Intel Corp.) and Yuri Alexeev (Argonne National Laboratory) for help in this research.