Optimizing Two-Electron Repulsion Integral Calculation on Knights Landing Architecture

Yingqi Tian, Bingbing Suo, Yingjin Ma and Zhong Jin
Computer Network Information Center of the Chinese Academy of Sciences
IXPUG Workshop at HPC Asia 2018, January 31st, 2018, Tokyo, Japan.

中国科学院
计算机网络信息中心
Computer Network Information Center, Chinese Academy of Sciences

Outline

- Background & Mathematics
- Algorithm
- Benchmark Result

Background: Quantum Chemistry

- Quantum chemistry programs are the most used programs on our supercomputer.
- One hotspot for quantum chemistry software is the two-Electron Repulsion Integral (ERI).

Rank	Program
1	VASP
2	GAUSSIAN
3	Material Studio
4	MOLPRO
5	NWChem

Top 5 most used programs on CAS supercomputer.
Consumes 20% of total CPU hours.

ERI Mathematical Formula

- The mathematical formula of primitive ERI is:

$$[is|jt] = \iint_{-\infty}^{\infty} \varphi_{\mathrm{i}}^{*}(r_1)\varphi_s^{*}(r_2)\frac{1}{r_1 - r_2}\varphi_j(r_1)\varphi_t(r_2)dr_1dr_2$$

  - $\varphi_{\mathrm{i}}^{*}(r_1)$ is the complex conjugate of $\varphi_i(r_1)$. Here all functions are real, so $\varphi_{\mathrm{i}}^{*}(r_1) = \varphi_i(r_1)$.

- $\varphi_i(r_1)$ is gaussian type function with different angular momentum L:

$$\varphi_i(r_1) = (x_1 - x_i)^n(y_1 - y_i)^l(z_1 - z_i)^m \exp(-\alpha_i(r_1 - r_i)^2), \qquad n + l + m = L$$

- The main purpose of ERI is to calculate all the primitive integrals for a given set of $\varphi(r)$ functions.
- So if you have N number of functions, the total number of ERI calculation is N^4. Normally N is around 100s~1000s.
- Integral calculations are independent.

Gaussian Product Theorem

- The Gaussian product theorem is :

$$\exp(-\alpha_i(r_1-r_i)^2)\exp\left(-\alpha_j(r_1-r_j)^2\right) = G_p\exp\left(-\alpha_p(r_1-r_p)^2\right)$$

- Where :

$$\alpha_p = \alpha_i + \alpha_j, \qquad r_p = \frac{\alpha_i r_i + \alpha_j r_j}{\alpha_p}, \qquad G_p = \exp(-\frac{\alpha_i\alpha_j(r_i - r_j)^2}{\alpha_p})$$

- We can use this theorem to transform our integral:

$$[p|q] = \int \frac{1}{r_1 - r_2}$$

$$\times {x_{1i}}^{n_i}{y_{1i}}^{l_i}{z_{1i}}^{m_i}{x_{1j}}^{n_j}{y_{1j}}^{l_j}{z_{1j}}^{m_j}\,G_p\exp\left(-\alpha_p(r_1-r_p)^2\right)$$

$$\times {x_{2s}}^{n_s}{y_{2s}}^{l_s}{z_{2s}}^{m_s}{x_{2t}}^{n_t}{y_{2t}}^{l_t}{z_{2t}}^{m_t}G_q\exp\left(-\alpha_q(r_2-r_q)^2\right)dr_1dr_2$$

- Here we get p from i and j, and q from s and t. So we get a two dimensional grid. This 2-D grid is suitable for parallelization and vectorization.

Recursion Relation Method

- There are three recursion schemes to calculate the integral:
  - The Dupuis-Rys-King (DRK) scheme, the Obara-Saika (OS) scheme and the McMurchie-Davidson (MD) scheme.
  - These 3 schemes have different recursion relations, but are mathematically equivalent.
- We use MD scheme, so the integral become the linear combination of several auxiliary functions.

$$[p|q] = G_p G_q \sum_{n_p}^{n_i+n_j} \sum_{l_p}^{l_i+l_j} \sum_{m_p}^{m_i+m_j} \sum_{n_q}^{n_s+n_t} \sum_{l_q}^{l_s+l_t} \sum_{m_q}^{m_s+m_t} d_{ij}^{x}(n_p) d_{ij}^{y}(l_p) d_{ij}^{z}(m_p)$$
$$\times d_{st}^{x}(n_q) d_{st}^{y}(l_q) d_{st}^{z}(m_q)$$
$$\times \gamma (-1)^{n_q+l_q+m_q} R(n_p+n_q, l_p+l_q, m_q+m_p, 0)$$

- The coefficient $d_{ij}^{x}(n_p)$ is generated by recursion relations:

$$d_{ij}^{x}(n_p, n_i+1, n_j) = \frac{1}{2\alpha_p} d_{ij}^{x}(n_p-1, n_i, n_j) + (x_p - x_i) d_{ij}^{x}(n_p, n_i, n_j) + (n_p+1) d_{ij}^{x}(n_p+1, n_i, n_j)$$

$$d_{ij}^{x}(0,0,0) = 1$$

Recursion Relation Method

- Also the auxiliary function $R(n, l, m, j)$ can be calculated by the recursion relations:

$$R(n+1, l, m, j) = x_T R(n, l, m, j+1) + nR(n-1, l, m, j)$$
$$R(n, l+1, m, j) = y_T R(n, l, m, j+1) + lR(n, l-1, m, j)$$
$$R(n, l, m+1, j) = z_T R(n, l, m, j+1) + mR(n, l, m-1, j)$$

- Where:

$$R(0,0,0,j) = (-2\alpha_T)^j F_j(T)$$
$$T = \alpha_T(x_T^2 + y_T^2 + z_T^2),$$
$$x_T = x_p - x_q, y_T = y_p - y_q, z_T = z_p - z_q,$$
$$\alpha_T = \alpha_p \alpha_q / (\alpha_p + \alpha_q)$$

- Finally $F_j(T)$ is a lower incomplete gamma function:

$$F_j(T) = \int_0^1 u^{2j} \exp(-Tu^2) du$$

Lower Incomplete Gamma Function

- Lower incomplete gamma function:

$$F_j(T) = \int_0^1 u^{2j} \exp(-Tu^2) du$$

- For 0≤T≤12:
  - Interpolation method (Taylor expansion):

$$F_j(T) = \sum_{k=0}^{6} F_{j+k}(T^*)(T^* - T)^k / k!$$

  - Series method:

$$F_j(T) = \frac{1}{2} \sum_{k=0}^{15+4T} \frac{(-1)^k T^k}{(j + k + 0.5)k!}$$

- For T>12:
  - Downward recursion relation:

$$F_j(T) = [2TF_{j+1}(T) + \exp(-T)]/(2j + 1)$$

$$F_0(T) = \frac{\pi^{1/2}}{2T^{1/2}} - \int_1^\infty \exp(-Tu^2) du$$

Mathematics Summary

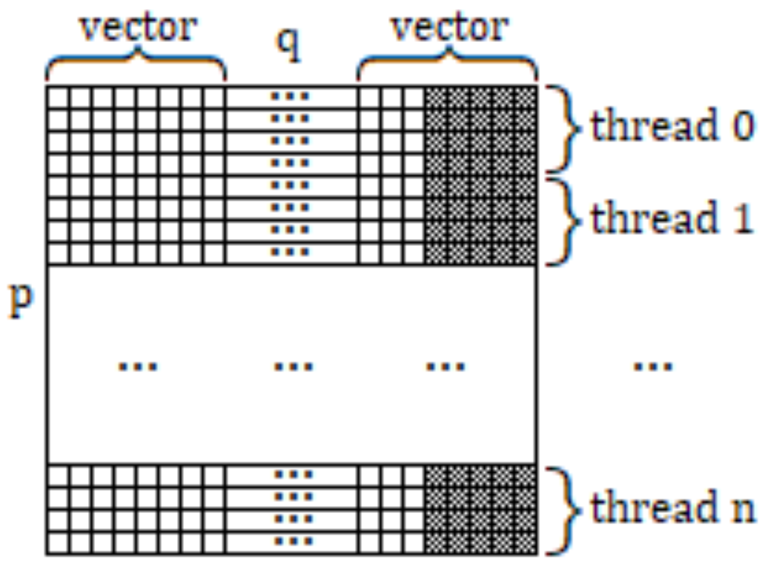
- Given serial of N $\varphi(r)$ functions, the total number of ERI calculation is N^4.
- All primitive integral calculations are independent.
- Gaussian product theorem can help get a 2-D grid.
- For MD scheme, we have 3 recursion relations.
- For incomplete gamma function, interpolation method vs. series method.

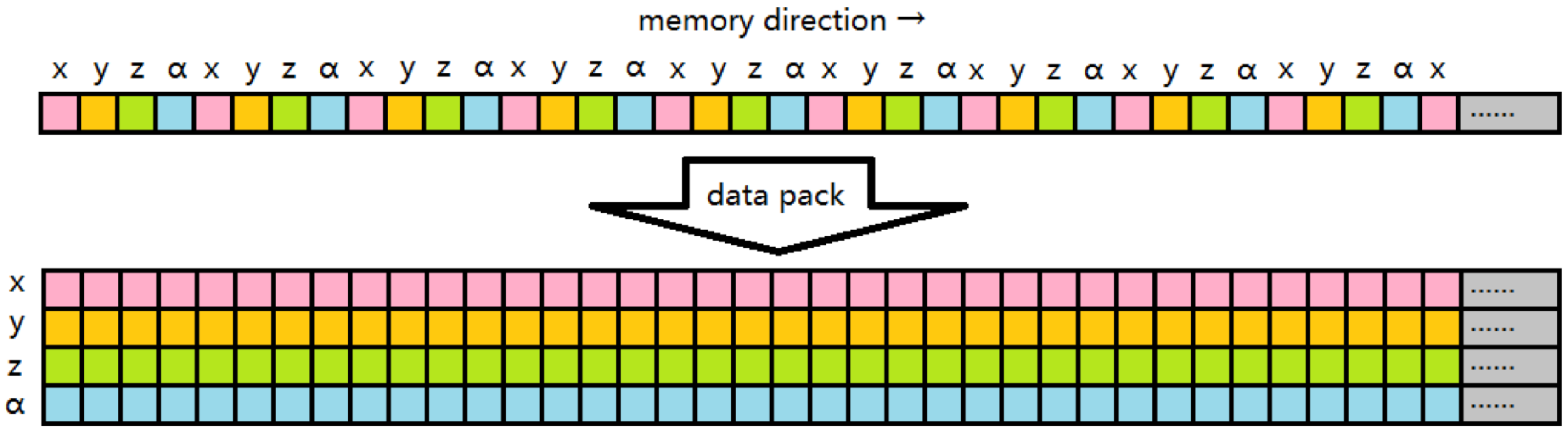
Outline

- Background & Mathematics
- Algorithm
- Benchmark Result

Parallelization and Vectorization Schedule

- This picture shows the 2-D grid:
  - Small block represents a integral task;
  - Date repacked and aligned to 64bit;
  - Vectorization along the q direction;
  - Thread parallelization along the p direction.

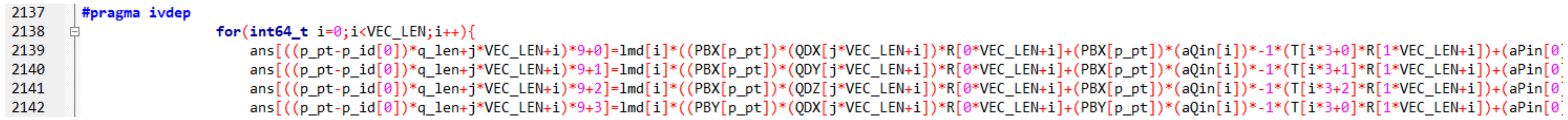




memory direction →
x y z α x y z α x y z α x y z α x y z α x y z α x y z α x y z α x y z α x
......
data pack
x
y
z
α

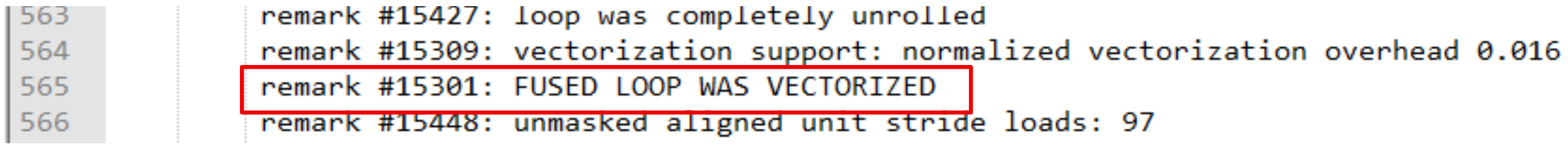
Vectorization for Recursion Relations

- A loops cannot be vectorized if a function is called in the loop.
- Inline method cannot be used for recursive functions.
- How to vectorize the recursion relations? Transform into loop?
- As the recursion relations are too complex, we decided to expand the calculation formula directly.
- We developed a program to automatically expand the formula and generate the codes:



```
#pragma ivdep
                for(int64_t i=0;i<VEC_LEN;i++){
                    ans[((p_pt-p_id[0])*q_len+j*VEC_LEN+i)*9+0]=lmd[i]*((PBX[p_pt])*(QDX[j*VEC_LEN+i])*R[0*VEC_LEN+i]+(PBX[p_pt])*(aQin[i])*-1*(T[i*3+0]*R[1*VEC_LEN+i])+(aPin[0
                    ans[((p_pt-p_id[0])*q_len+j*VEC_LEN+i)*9+1]=lmd[i]*((PBX[p_pt])*(QDY[j*VEC_LEN+i])*R[0*VEC_LEN+i]+(PBX[p_pt])*(aQin[i])*-1*(T[i*3+1]*R[1*VEC_LEN+i])+(aPin[0
                    ans[((p_pt-p_id[0])*q_len+j*VEC_LEN+i)*9+2]=lmd[i]*((PBX[p_pt])*(QDZ[j*VEC_LEN+i])*R[0*VEC_LEN+i]+(PBX[p_pt])*(aQin[i])*-1*(T[i*3+2]*R[1*VEC_LEN+i])+(aPin[0
                    ans[((p_pt-p_id[0])*q_len+j*VEC_LEN+i)*9+3]=lmd[i]*((PBY[p_pt])*(QDX[j*VEC_LEN+i])*R[0*VEC_LEN+i]+(PBY[p_pt])*(aQin[i])*-1*(T[i*3+0]*R[1*VEC_LEN+i])+(aPin[0
```

- And the compiler report shows the loop was vectorized:



```
remark #15427: loop was completely unrolled
remark #15309: vectorization support: normalized vectorization overhead 0.016
remark #15301: FUSED LOOP WAS VECTORIZED
remark #15448: unmasked aligned unit stride loads: 97
```

Interpolation Method vs. Series Method

- As I mentioned before, lower incomplete gamma function can be calculated by two method:
  - Interpolation method (Taylor expansion):

$$F_j(T) = \sum_{k=0}^{6} F_{j+k}(T^*)(T^* - T)^k / k!$$

  - Series method:

$$F_j(T) = \frac{1}{2} \sum_{k=0}^{15+4T} \frac{(-1)^k T^k}{(j + k + 0.5)k!}$$

- For interpolation method, $F_{j+k}(T^*)$ is previously calculated and stored in memory for $T^* = 0, 0.1, 0.2 \ \ldots 12$ and j from 0 to 16. So it has 7 memory load instructions for each $F_j(T)$.
- For series method, it`s just calculations.

Code structure

```
Loop i:
    Loop j:
        Get p in aligned memory;
    End loop j;
End loop i;

Parallel loop p:
    Loop q/(vector length):
        Vector loop:
            Get R(0,0,0,j) for j=0~L from $F_j(T)$;
            Integral calculation [p|q] with expanded codes;
        end vector loop;
    End loop q;
End loop p;
```

Outline

- Background & Mathematics
- Algorithm
- Benchmark Result

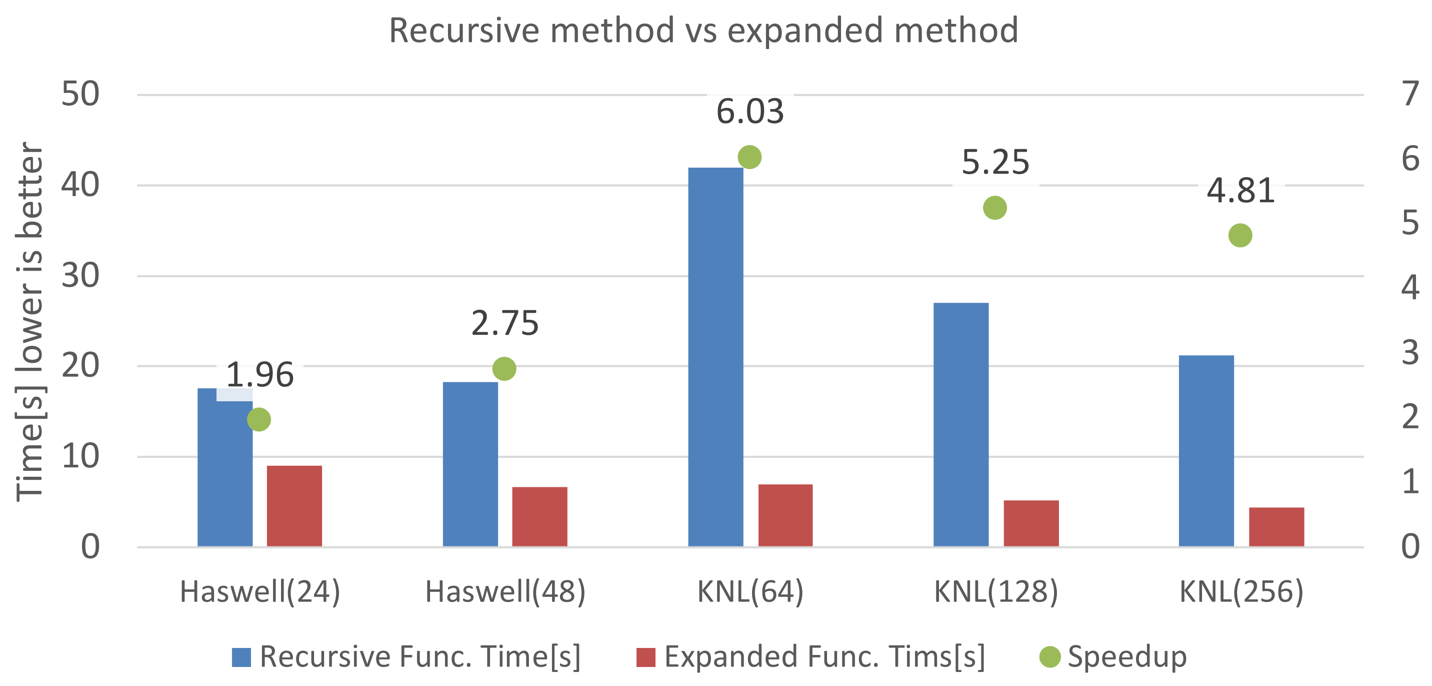
Platform and Benchmark Molecules

- Haswell platform:
  - 2 × Intel Xeon E5-2680v3 12 cores @ 2.50GHz 120W
  - 2 × 256-bit Vector Processing Unit per core
  - 0.96GFLOPS
- KNL platform:
  - Intel Xeon Phi 7210 64 cores @ 1.30GHz 215W
  - 2 × 512-bit Vector Processing Unit per core
  - 2.66GFLOPS
- Benchmark Molecules:

No.	Molecule	No. of atoms	No. of basis functions
1	Caffeine	24	146
2	Cocaine	43	240
3	Protein 1	71	408
4	Taxol	110	647
5	Protein 2	145	815
6	Valinomycin	168	882

Recursive vs. Expanded

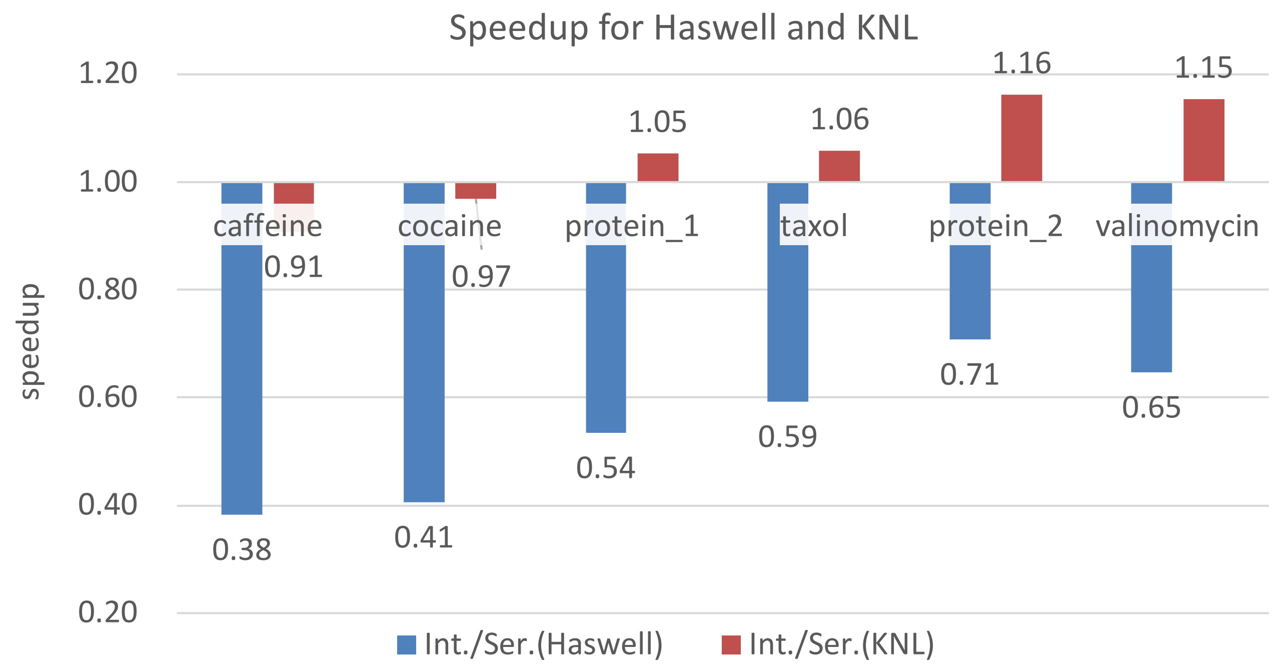
- We choose the $F_j(T)$ downward recursion relation (3rd) to show the different performances between recursive function and expanded function:



- As the expanded function can be vectorized and no function call, it performs better on both platform.

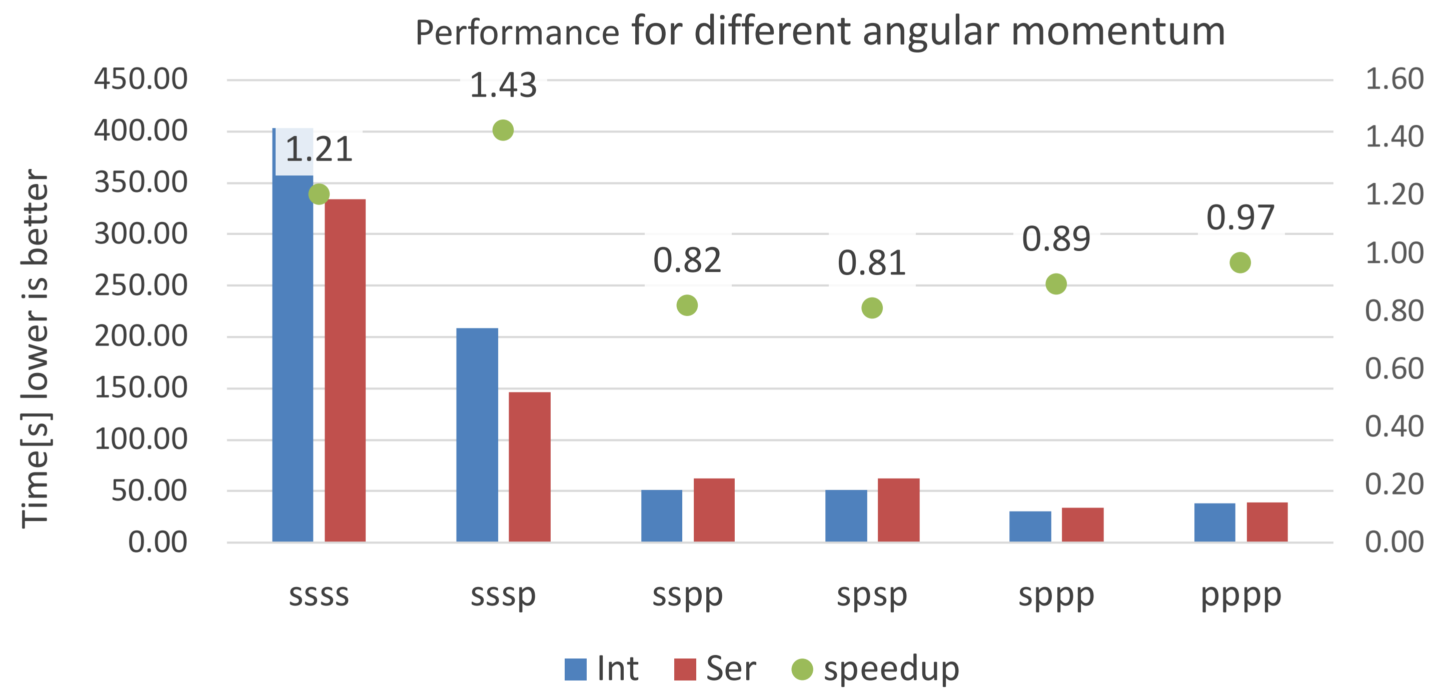
Interpolation vs. Series

- We implemented both interpolation method and series method for $F_j(T)$ calculation, and tested on both platform:
  - Interpolation method performs better on Haswell;
  - Series method performs better on KNL;



Interpolation vs. Series

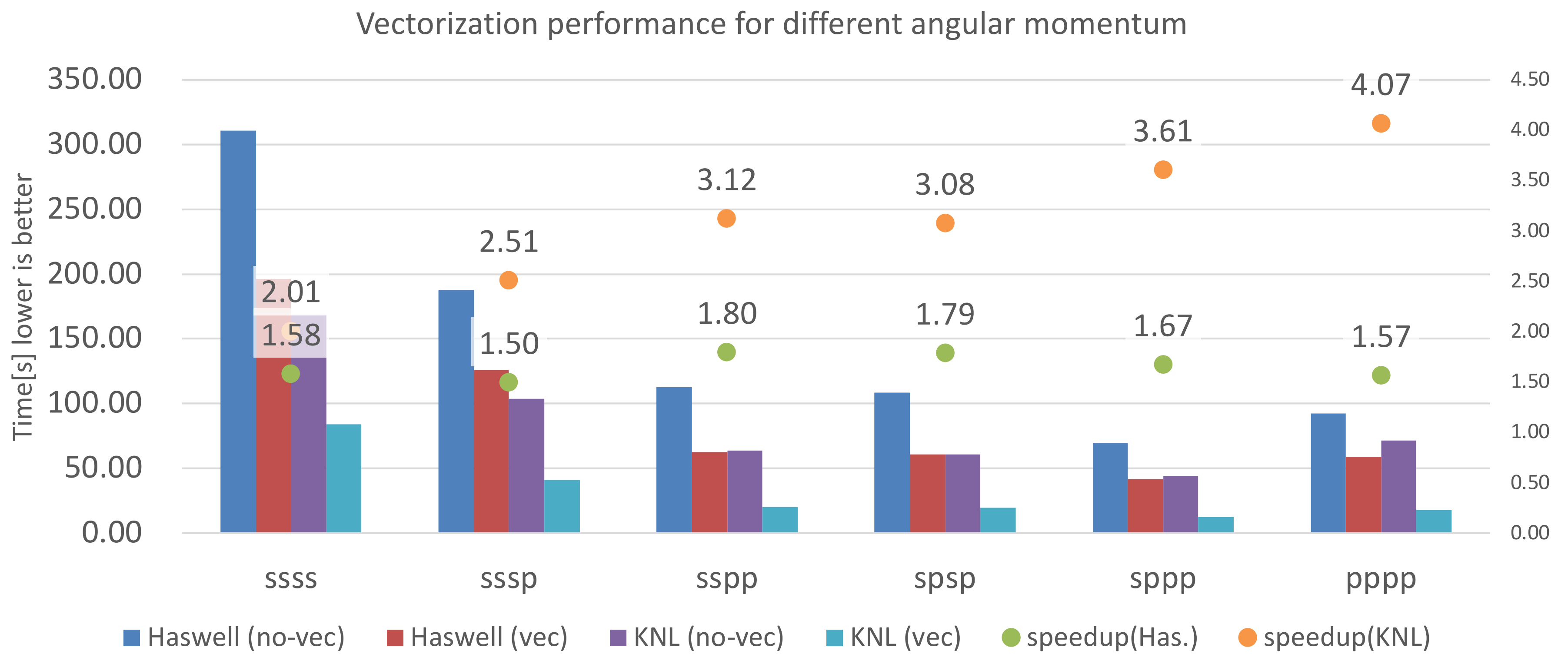
- If we check the series method performance for different angular momentum L on KNL:
  - Series method performs better when angular momentum L is small.



Vectorization

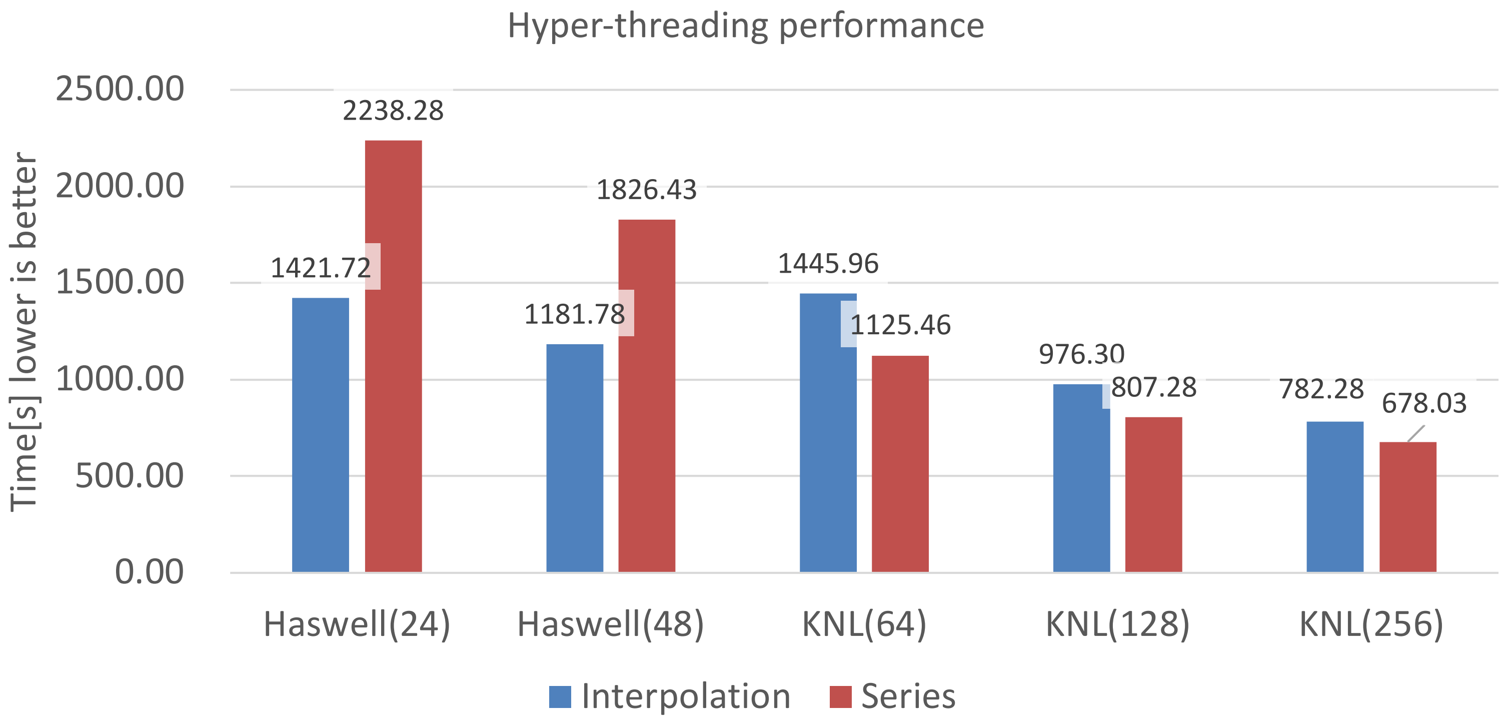
中国科学院 计算机网络信息中心 Computer Network Information Center, Chinese Academy of Sciences

- We compiled our codes with –vec and –no-vec compile options to see how well our vectorization method performed:



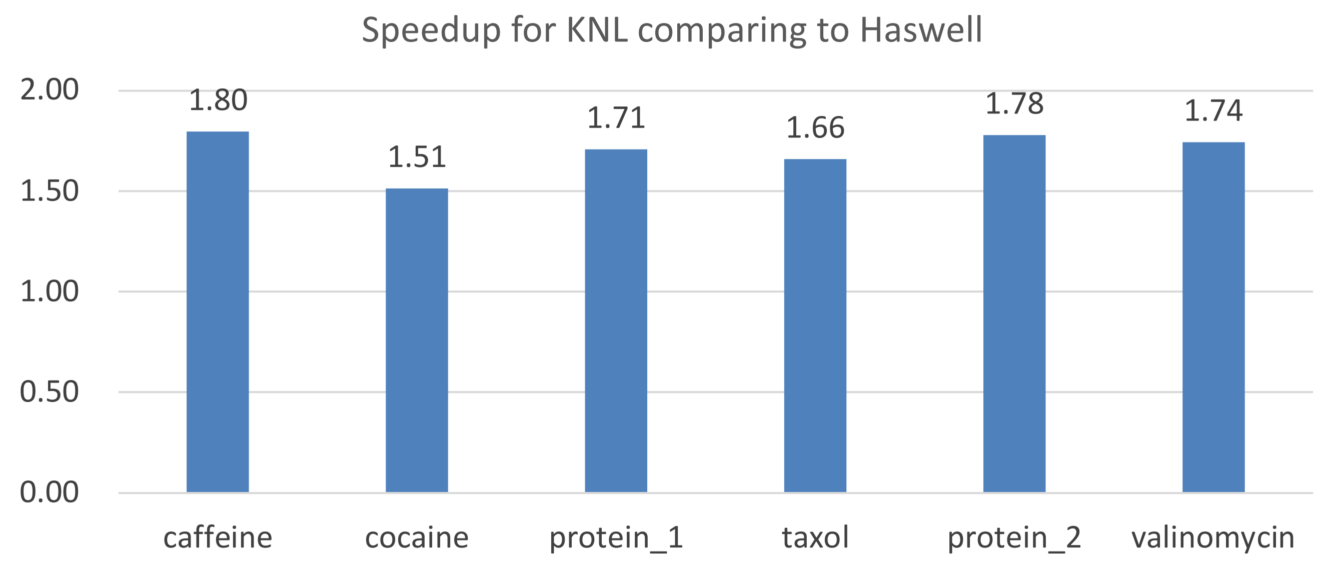
Hyper-threading

- We also tested the hyper-threading performance:
  - Haswell support 2 thread per core
  - KNL support up to 4 thread per core
  - KMP_AFFINITY=scatter



Over All KNL Speedup

- By applying all these optimization methods, we have the final result:
  - We get up to 1.80 performance speed up on KNL comparing to Haswell.
  - Average speedup is 1.70.



Conclusion

- We developed a parallelization and vectorization scheme for ERI.
- We developed a program to automatically expand recursion relation formula and generate vectorizable codes. Expanded codes performed pretty well both on Haswell and KNL.
- We found interpolation method performed better on Haswell but series method performed better on KNL.
- Over all, we get up to 1.80 speedup on KNL comparing to Haswell.

Acknowledgments

- Prof. Zhong Jin (PhD supervisor)
- Prof. Bingbing Suo (Northwest University)
- Dr. Yingjin Ma (CNIC, Chinese Academy of Science)
- $$:
  - Informationization Program of the Chinese Academy of Science: Development of first principle software;
  - National Key R&D Program of China.

Thank you for your attention！

中国科学院
计算机网络信息中心
Computer Network Information Center, Chinese Academy of Sciences