Report: **Project 2- Hydrodynamic interactions** (CSE 6230)

-- by Harsh Shrivastava (gtID: 903241811)

Report is divided into 4 parts:

Part A: Correctness

All the simulations done gave squared error = 0.000000

(Results shown of the input file 'lac1_novl2.xyz' and parameters xi= 1 . 5 π/ L, nr=2 and nk=3 and are obtained from running on **'mic2'** coprocessor)

Part B: Overall speed

Vectorization : SIMD (off =0, on=1)

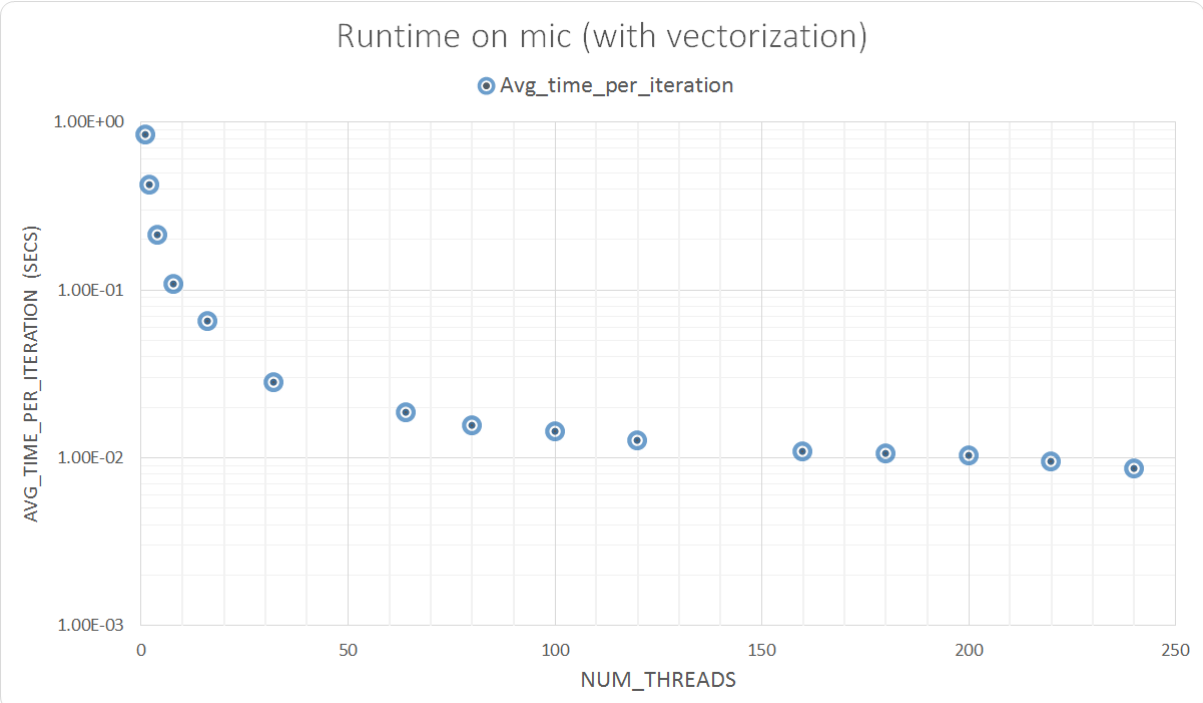
Multithreading: OPENMP (off=0, on=1)

Comparison of BEST TIMINGS (NTHREADS =240 wherever applicable)

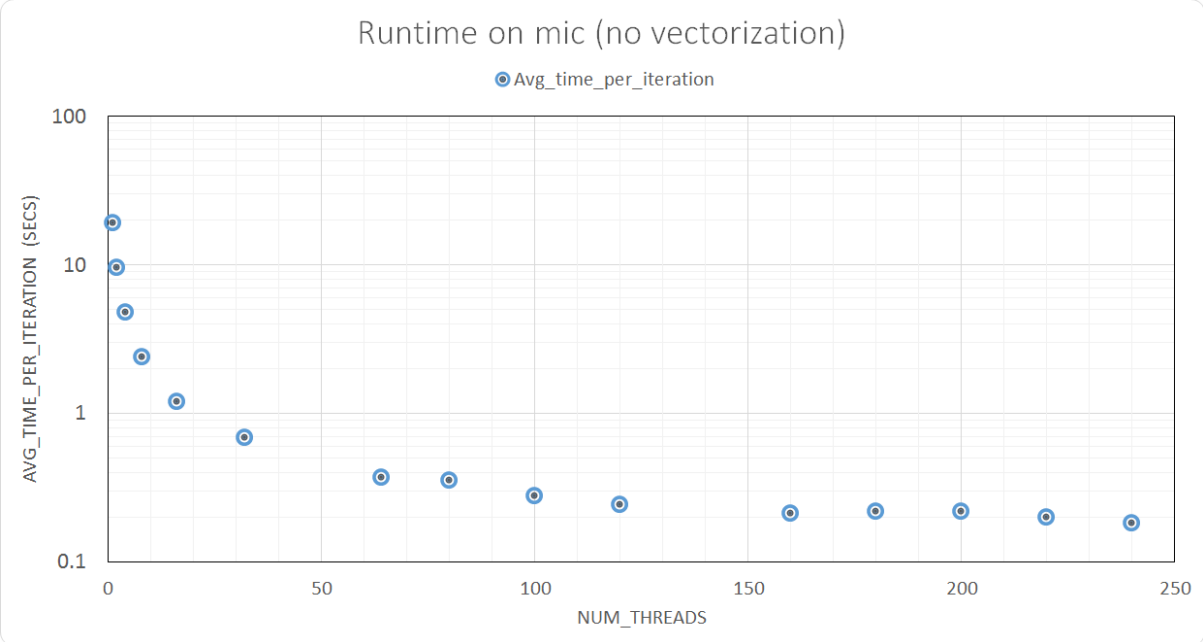
make [option]	SIMD flag	OPENMP flag	Runtime (in secs)	Speedup
check_mic_novec	0	0	19.317257	1.0
check_only_openmp	0	1	0.181829	106.238421
check_only_vec	1	0	0.832667	23.199258
check_mic	1	1	**0.008654**	**2232.248849**

B1: Graph of Runtime vs Number of threads (with vectorization) (LOG plot)

(Note: Change NTHREADS on line 8 of rpy_ewald_polyd.c file and run it using 'make check_mic')



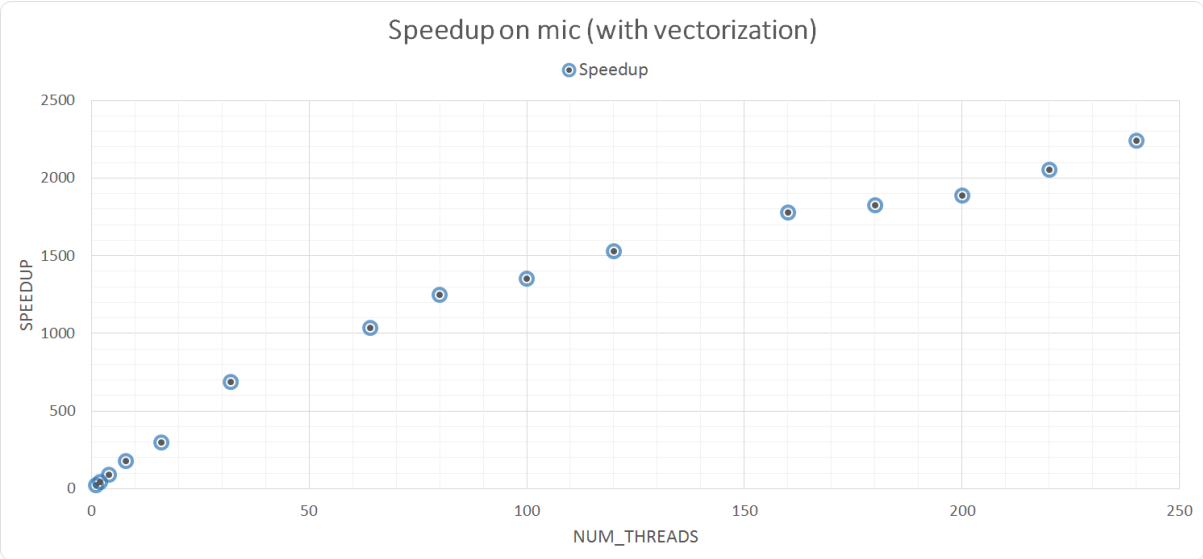
B2: Graph of Runtime vs Number of threads (with no vectorization) (LOG plot) (Note: Change NTHREADS on line 8 of rpy_ewald_polyd.c file and run it using 'make check_only_openmp')



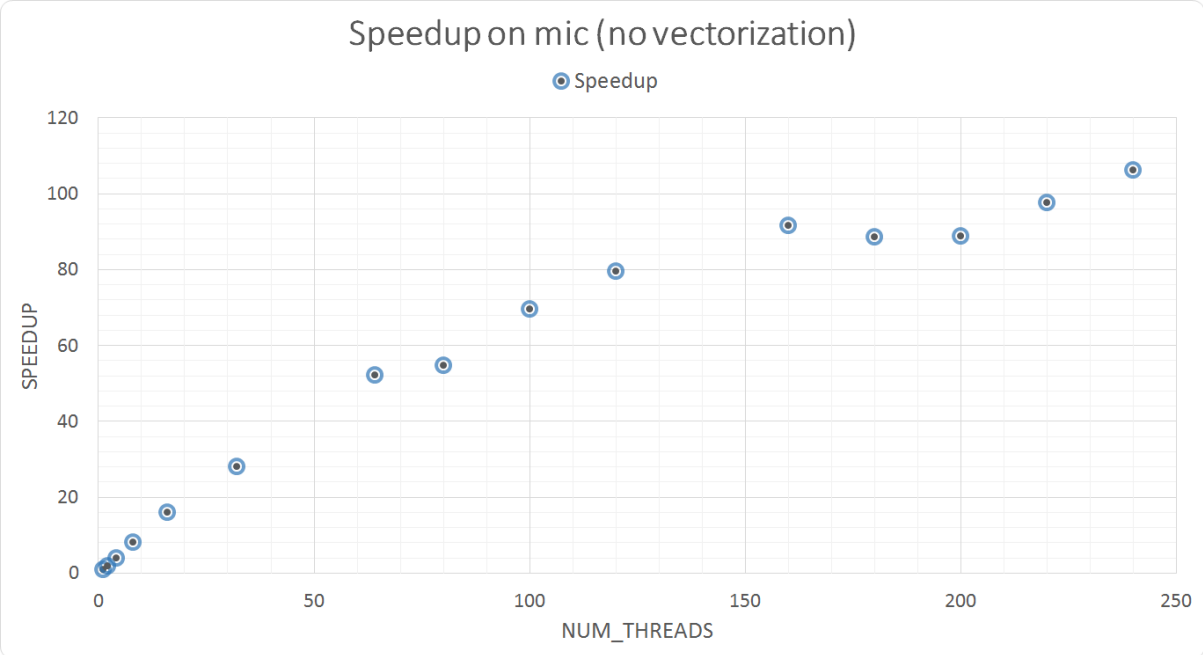
Part C: Speedup graphs

Speedup = (Runtime_for_novec_noopenmp)/(Current_runtime)

C1: Graph of Speedup (with vectorization)



C2: Graph of Speedup (with no vectorization)



So vectorization accounts for approximately ~24 times speedup in my code.

Part D: Discussions:

D1: Parameters tried

1. KMP_AFFINITY=scatter/balanced with granularity=fine were the best choices. KMP_AFFINITY=compact didn't give good results as expected
2. Tried various OMP_SCHEDULE = dynamic, static, guided with various choices, but the best results were obtained by static scheduling. I guess, since the number of particles are less the for loops are not that large for other scheduling techniques to exploit their optimum functioning
3. Various NUMA options tried for mic

D2: Changes made to code:

1. In Real space sum calculation loop (for j=i changed to j=i+1) and the [if(i==j) & if(r2==0)] conditions were removed
2. The if (i==j) condition in reciprocal calculations were also removed and was taken care in the "self-part" section while updating array 'a'
3. The 3 for loops going from -nk/nr to nk/nr were replaced by indices (a single for loop) using which x,y,z were recalculated
4. Functions were made inline and restricts were used
5. The [goto out] condition was also removed by iterating to half the size of original loop.
6. Also, the remaining 2 for loops were flattened to obtain improved performance while running openmp pragmas (Some calculations were tricky to flatten the for loops, which can be observed from the code itself)
7. Even tried flattening the complete for loops in real part calculations by adjusting the variables and removing the 'temp' array and directly updating 'a' instead, but the overhead of accessing 'a' was more and the runtime decreased

8. The temp and other arrays, array size was adjusted to avoid cache sharing
9. Further improvements can be done to avoid possible false sharing for accessing the array ‘a’ by the threads