University of Puerto Rico at Mayagüez
Electrical and Computer Engineering Department
High Performance Computing
Dr. Wilson Rivera Gallego
Fall 2017

Project Report:
Parallel Physic-based simulations of theoretical soft matter

Luis O. Vega Maisonet	840-13-9529	luis.vega5@upr.edu
Alejandra Ortiz Capetta	802-13-5520	alejandra.ortiz2@upr.edu
Eric Santillana Santiago	802-13-8045	eric.santillana@upr.edu

I. Project Motivation

The purpose of choosing this project was to collaborate with another department of the University of Puerto Rico at Mayagüez. This partnership was done to expand our knowledge, our social status as professional individuals and to learn new topics that could be used in our future. This project was chosen after speaking with different research groups of Dr. Ubaldo Córdova of the chemical engineering department. After evaluating the different projects inside their research group,it was determined that the project that needed the most work was the Brownian Dynamic Project. An interesting thing that caught our eyes was to learn a new programming language called Fortran which non of the HPC research group members had worked on. Since we had previously used OpenMP libraries on C language scripts for the HPC course assignments and OpenMP libraries are also available for Fortran, this project seemed like the perfect choice us.

II. Project Description

The Brownian Dynamic code provided by the chemical engineering research group was written in Fortran 90. This code makes a particles simulation in three dimensions (x,y,z) inside a box, here the particles interact with each other and the code makes physics calculations based on those interactions. The code normally runs with 500 particles and takes approximately a week to execute and produce the desired results. The code was written in a sequential form which means the parallelization of such code is tedious because it needs to be rearranged almost completely to be effectively parallelized. The code runs exclusively on ifort, a fortran compiler provided by Intel Corporation. The chemical engineering research group also provided us with a MakeFile to compile and execute the program, which had to be slightly edited to utilize OpenMP by adding the appropriate command line flag.

III. Technical Approach

For the installation of ifort, we needed to download the student version of XE Parallel Studio through which we discovered Intel Advisor. This tool helped us pinpoint the modules in the program that were taking the most time to execute. Intel Advisor uses the executable file of the Brownian Dynamic code to scan and make the analysis. Figure 1 shows a summary of Intel Advisor performance results of the Brownian Dynamics code. Figure 2 shows a sample of the module list that Intel Advisor marks as potential candidates for parallelization.

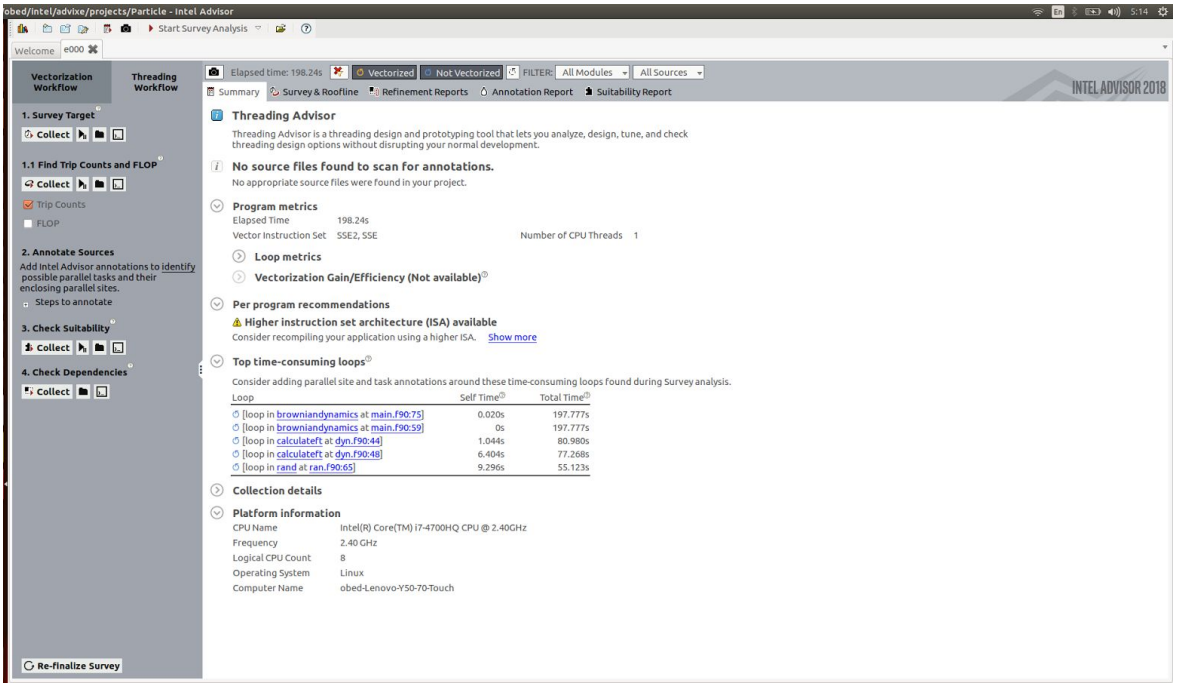


Figure 1: Screenshot of Intel Adviser GUI Summary

Since Intel Advisor is a GUI, it cannot be transferred to an instance to analyze the Brownian Dynamic code. Note that the total elapsed time that Intel Advisor produces should not be taken into consideration because this analysis of the code was done in a local computer but it serves its purpose to measure the execution time of modules and loops.

Function Call Sites and Loops	Performance Issues	Self Time	Total Time	Type
_start		0.000s	203.910s	Function
main		0.000s	203.910s	Function
browniandynamics		0.000s	203.910s	Function
[loop in browniandynamics at main.f90:75]		0.020s	203.818s	Scalar
[loop in browniandynamics at main.f90:59]		0.000s	203.818s	Scalar
updateparticles		0.220s	111.516s	Function
calculateft		0.768s	91.481s	Function
[loop in calculateft at dyn.f90:44]	1 Potential unde...	1.116s	84.641s	Scalar Versions
[loop in calculateft at dyn.f90:48]		6.472s	80.825s	Scalar
randomvectors		1.044s	60.528s	Function
rand		0.088s	57.212s	Function
[loop in rand at ran.f90:65]	1 Data type conv...	9.755s	57.124s	Scalar
updatequat		0.104s	51.044s	Function
[loop in rand at ran.f90:69]	1 Data type conv...	23.344s	47.369s	Scalar
updateorientationquat		0.088s	22.040s	Function
[loop in updateorientationquat at quat.f90:105]		3.612s	21.952s	Scalar
__libm_fmod_e7		21.744s	21.744s	Function
[loop in updatequat at quat.f90:144]		5.376s	21.248s	Scalar
distdip		2.636s	16.972s	Function
magnetictorque		5.000s	16.898s	Function
vdot		7.308s	14.218s	Function
magneticforce		3.554s	11.118s	Function
transform		2.248s	10.206s	Function
[loop in transform at vfun.f90:237]	1 Potential unde...	7.958s	7.958s	Scalar Versions
matrices		4.412s	7.804s	Function
[loop in updateparticles at dyn.f90:147]		0.924s	7.376s	Scalar
[loop in vdot at vfun.f90:64]	1 Potential unde...	6.910s	6.910s	Scalar
minimage		0.888s	6.060s	Function
[loop in minimage at vfun.f90:77]		2.788s	5.172s	Scalar
containparticle		0.720s	4.352s	Function

Figure 2: Screenshot of Intel Adviser Gui Module and Loop Analysis

IV. Experimental Setting

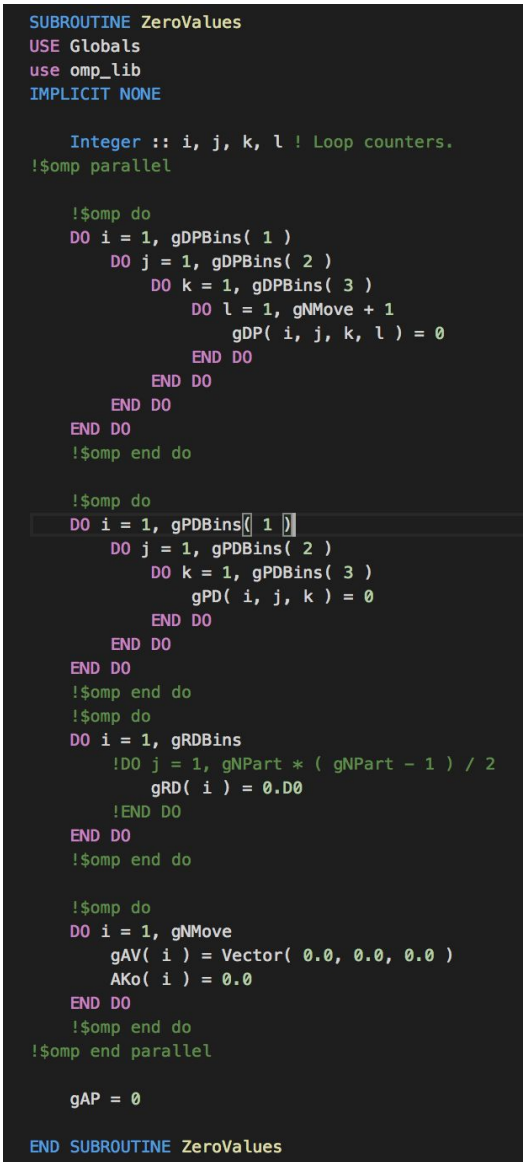
To ensure that we were utilizing the same environment and hardware for each test, we set up an instance of an image through Chameleon Cloud which we utilized for all of the tests done. The selected instance was Ubuntu_16.04_gcc7_openmp4 which already contained OpenMP and gcc7 libraries. By using youtube videos as resources we were able to install XE Parallel Studio. As previously stated, since the code provided would take approximately a week to run the simulation of 500 particles, we decided to lower this amount to something which would run in a more reasonable time frame suitable for testing and experimenting. After some experimentation with the configurations, we found that 5 particles and 10 particles worked best for us since they ran at approximately 3 minutes and 15 minutes respectively. After studying OpenMp Fortran syntax, we attempted to parallelize the modules that took the most execution time one by one. This was a tedious process since we didn't want to change any of the calculations done by the research students, since that could greatly impact the results and was not our area of expertise. Some of the Brownian Dynamics Fortran source codes that we attempted to parallelize with OpenMP were ran.f90, dyn.f90, vfun.f90 and init.f90.

```
integer :: NLength
real, dimension (12) :: sumArray
    NLength = 12*n
    sum = 0.d0
    count = 1

    do j = 1, NLength                       ! why the last value of j is 12 and is not other value?

        seed = dmod( 16807.0 * seed, d2p31m )  ! dmod stand for double-precision module - By Luis

        sum = sum + seed * d2p31m_inv

        if(MODULO(j, 12) .eq. 0) then
            sumArray(count) = sum
            sum = 0
            count = count+1
        end if
    end do
    !$omp parallel shared(n) private(i)
    !$omp do
    do i = 1, n

        rr = 0.25d0 * ( sumArray(i) - 6.d0 )
        rrsq = rr * rr

        term1 = 0.029899776d0 * rrsq + 0.008355968d0
        term2 = term1 * rrsq + 0.076542912d0
        term3 = term2 * rrsq + 0.252408784d0
        term4 = term3 * rrsq + 3.949846138d0 !term4=function(sum)

        u(i) = term4 * rr                   !1.414213562d0 * term4 * rr, u(i)=function(sum)

    end do
    !$omp end do
    !$omp end parallel
```

Figure 3: Screenshot of a parallelization attempt of ran.f90 with a completely separated nested loop and OpenMP directives.

These modules were chosen since we had observed through Intel Advisor that they took up a significant amount of time of the execution. After trying to apply OpenMP parallelization to the modules mentioned above, the only one that

was able to be run faster with OpenMP directives were the nested for loops inside the ZeroValues module in init.f90 source code. The other source codes (ran.f90, dyn.f90 and vfun.f90) contained dependencies and functions that weren't able to run in parallel.



```
SUBROUTINE ZeroValues
USE Globals
use omp_lib
IMPLICIT NONE

    Integer :: i, j, k, l ! Loop counters.
!$omp parallel

    !$omp do
    DO i = 1, gDPBins( 1 )
        DO j = 1, gDPBins( 2 )
            DO k = 1, gDPBins( 3 )
                DO l = 1, gNMove + 1
                    gDP( i, j, k, l ) = 0
                END DO
            END DO
        END DO
    END DO
    !$omp end do

    !$omp do
    DO i = 1, gPDBins( 1 )
        DO j = 1, gPDBins( 2 )
            DO k = 1, gPDBins( 3 )
                gPD( i, j, k ) = 0
            END DO
        END DO
    END DO
    !$omp end do
    !$omp do
    DO i = 1, gRDBins
        !DO j = 1, gNPart * ( gNPart - 1 ) / 2
            gRD( i ) = 0.D0
        !END DO
    END DO
    !$omp end do

    !$omp do
    DO i = 1, gNMove
        gAV( i ) = Vector( 0.0, 0.0, 0.0 )
        AKo( i ) = 0.0
    END DO
    !$omp end do
!$omp end parallel

    gAP = 0

END SUBROUTINE ZeroValues
```

Figure 4: Screenshot of the ZeroValues module of init.f90

V. Results

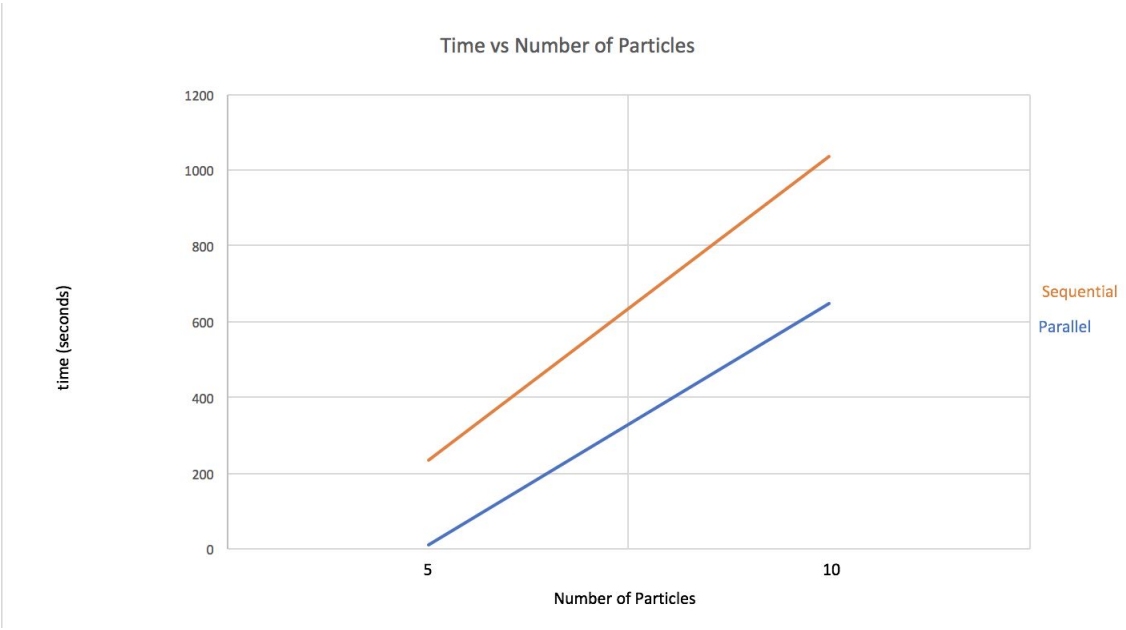


Figure 5: Line plot comparison of the original Brownian Dynamics code vs the parallel code

VI. Conclusions and Future Work

After thorough analysis of the Brownian Dynamics code provided by the Chemical Engineering research students, we determined that most of the code has far too many dependencies to be able to effectively parallelized. Although we did manage to speed it up by applying OpenMP, the code itself could be significantly improved through rewriting the parts that take the most amount of time (ran.f90, dyn.f90, vfun.f90 and init.f90). Rewriting these would take significant modifications to the logic of almost the entire code.