EEL6763 PARALLEL COMPUTER ARCHITECTURE
SPRING 2019

PROJECT PROPOSAL

PARALLELIZATION OF DYNAMIC WORKLOAD PREDICTION TOOL FOR TRACE DRIVEN SIMULATION

Contributors:

Devansh Gulati (9653 6016)
Kshitij Raj (1358 4965)
Kaustubh Shukla (9171 3377)

1. Subject and Purpose:

Dynamic workload prediction tool is used with trace-driven simulations. Trace driven simulation is important because it helps us analyze the workload based on the particle count. Workload per processor depends on the number of particles and if the particles per processor is dynamic the particle flow information varies among processor and varies at each time step. Therefore, it is important for us to extract trace of the particles.

The goal is to predict particle workload across processors for different machine and application configurations. A trace data for a specific problem can be given as input along with applications and machine processor count parameters. The basic assumptions are that on increasing the problem size, particle movement pattern doesn't change, single trace of a specific problem is sufficient to predict particle movement and generates a synthetic trace for larger problem sizes is based on original trace for smaller sizes.

2. Definition of the problem:

Given the serial implementation of the problem, this project aims to implement the tool using parallel heuristics. Using parallel programming models like MPI, OpenMP and CUDA, the Particle-Workload Predictor will see dramatic speedup in simulating the problem, using different profiling tools to measure the performance. The particle movement is machine architecture independent and relies on the parameters of the problem and hence there should not be any problem in correctness & implementing parallelization. Using trace mapping and element to rank input files, we are able to track the particle movement across the processors at a particular timestamp. Along with tracking the particle, we also track the communication workload by tracking the particle movement across processors at a given timestamp. This is achieved by analyzing particle-element mapping.
The first input file is a particle trace file which is of the following format:

<gPID1> <gPID2> <gPID3> <gEID> <Px> <Py> <Pz> <Ts>

Where:

- gPID1, gPID2 and gPID3 -> Global particle IDs which are unique to every particle.
- gEID -> Global Element ID
- Px,Py,Pz -> Location of the particle
- Ts -> Timestep

The second input is an Element to Rank Mapping which is of the following format:

<Rank> {gEID1, gEID2, gEID3, ..., gEIDn}

The first output file is of the following format:

<Rank> <# of Particles> <Ts>

- This file will give the number of particles in each rank per timestep.

The second output file will give the analysis of a particle's movement across different ranks. It has the following format:

<Source Rank> <Destination Rank> <Ts> <# of particles>

Where:

- Source Rank -> Rank of Particle before movement.
- Destination Rank -> Rank of Particle after movement.

3. Need for Solution:

The need for a solution arises from the fact that since the algorithm is embarrassingly parallel, we can achieve significant speedup by parallelizing the tool. Trace-driven simulation is widely used in examining different computer architectures. The need for trace-based simulations is increasing rapidly as the complexity and performance of computer systems increase. Parallelizing the Dynamic Particle-Workload Prediction Tool will dramatically decrease the time it takes a serial code to generate the same set of results and help developers design efficient algorithms. The scope of parallelization also increases as there is almost close to no communication across processors. The tool also provides a way to find the most optimum configuration for a set of problem data, hence streamlining the computation.

4. Benefits:

Parallelizing the Dynamic Particle-Workload Prediction Tool numerous benefits and some of them are listed below:
1] Dramatic reduction in computation time and an increase in data concentration.
2] The tool will enable us to examine key metrics such as computation costs, communication cost, resource sharing & utilization. Examining these key metrics will enable us to make the tool more efficient.
3] Since the particle movement in independent of the system architecture, a single trace is sufficient to predict the workload of the system.
3] Since we are running the simulation and not on the actual system, we can analyze load imbalance and communication overheads, implement various optimization strategies and make algorithmic changes to our design.
4] Using Synthetic Trace allows us to predict workloads under different scenarios by using the same trace to generate new synthetic trace which saves a lot of time since collecting a trace is a time-consuming process.

5. Background on the Problem:

The background on this problem can be understood by evaluating a set of subproblems explained in the following sections namely: Mapping Problem, Movement detection and Synthetic Trace. The flow of the process is shown in Fig 1.

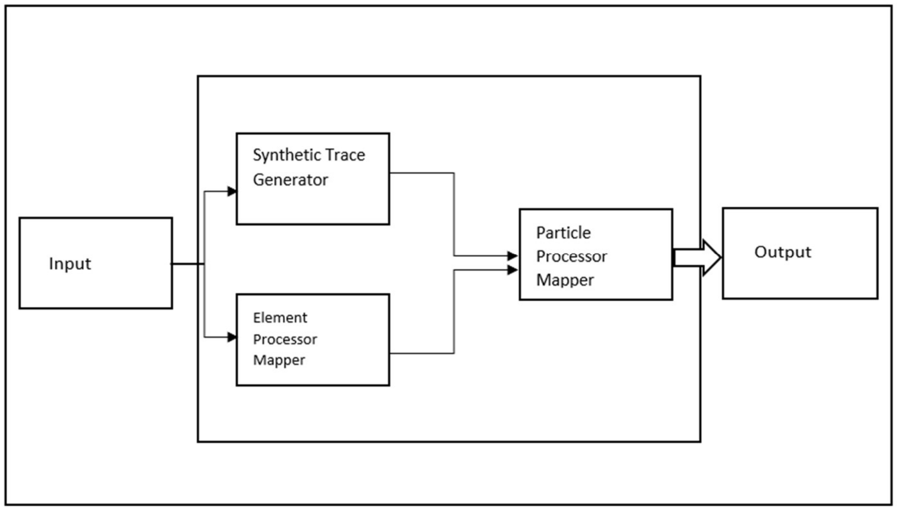


Fig 1:   Workflow of Particle Workload Prediction Tool

Mapping Problem

In order to efficiently implement the Dynamic Workload Prediction Tool, we need to parallelize a set of problems that the serial code/program solves. The first is the mapping problem. The mapping problem considers a mesh or tile like framework of elements as shown in Fig 2. Each of these elements consists of a particle-space and the particles in the element are free to move, even across different elements as we will see eventually.

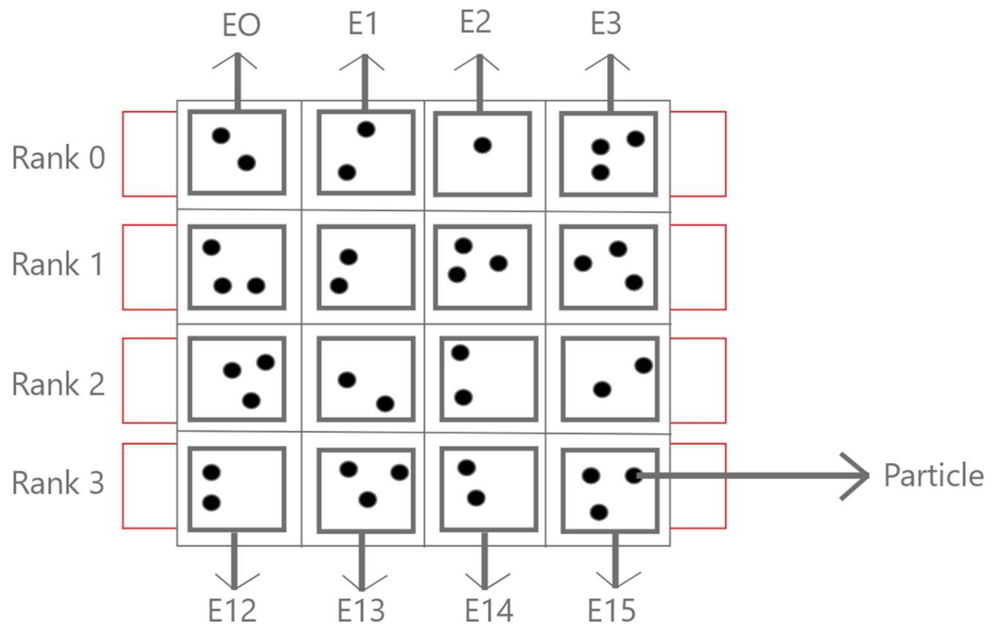


Fig 2 – Element Layout across Ranks

The mapping problem is solved by the tool to evaluate the number of particles per processor/rank. This is done using two input files and a variety of input of parameters namely, {gpid1, gpid2, gpid3} : A unique combination of particle ids that uniquely identify a particle across the spatial and temporal space, {geid} : element, {px, py, pz} : coordinates of a particle across the spatial space and {ts} : time-stamp. Timestamp is the most important aspect of the particle/element space. Since this is a simulated environment, it is important to capture the event across different time-stamps. The first input to the file can be structured as :

IP 1 : <gpid1, gpid2, gpid3> <geid> <px, py, pz> <ts>

The idea is to parallelize this code by following the input data decomposition mechanism and evenly divide the line of input or each time stamp amongst different processors. At this stage the second input file is provided with the code which already gives the decomposed layout of ranks and elements.

IP 2 : <rank> <geid1, geid2, geid3, geid4, …..geidN>

In the dynamic workload environment, a way to parallelize this problem of evaluating number of particles in a processor per time stamp is to implement a dictionary with element id as the key and number of particles as the value. This data is readily available in IP 1. Thus for each geid(i) corresponding to each process per time stamp, the values of number of particles can be accumulated over the number of elements. If we consider the dictionary to be “dic” and key to be geid(i), then an accumulator “acc” can be defined for M process and N elements as :

Pseudocode:

```
for each process p in 0 to M-1
{
    for each element i in 0 to N-1
    {
        acc=acc+ dic(geid(i))
    }
}
```

Complexity per process is observed to be linear and equal to the total number of elements i.e O(N).

From an implementation perspective, the task can be decomposed based on Input data decomposition and assigned to processors using MPI (MPI file handlers). This process can be further optimized by using a hybrid programming model (MPI + OpenMP, MPI+ CUDA).

Movement Detection

Once the tool has finished the first task of analyzing the number of particles per rank for a given time step it writes this data in an output file which looks like this:

<rank> <gpid1, gpid2, gpid3> <ts>

The tool then moves on the next stage of evaluating the movement of particles across processors as shown in Fig 3. This step is defined as analyzing the particle trace. This definition is important as it will be used to discuss the synthetic trace in the next section.

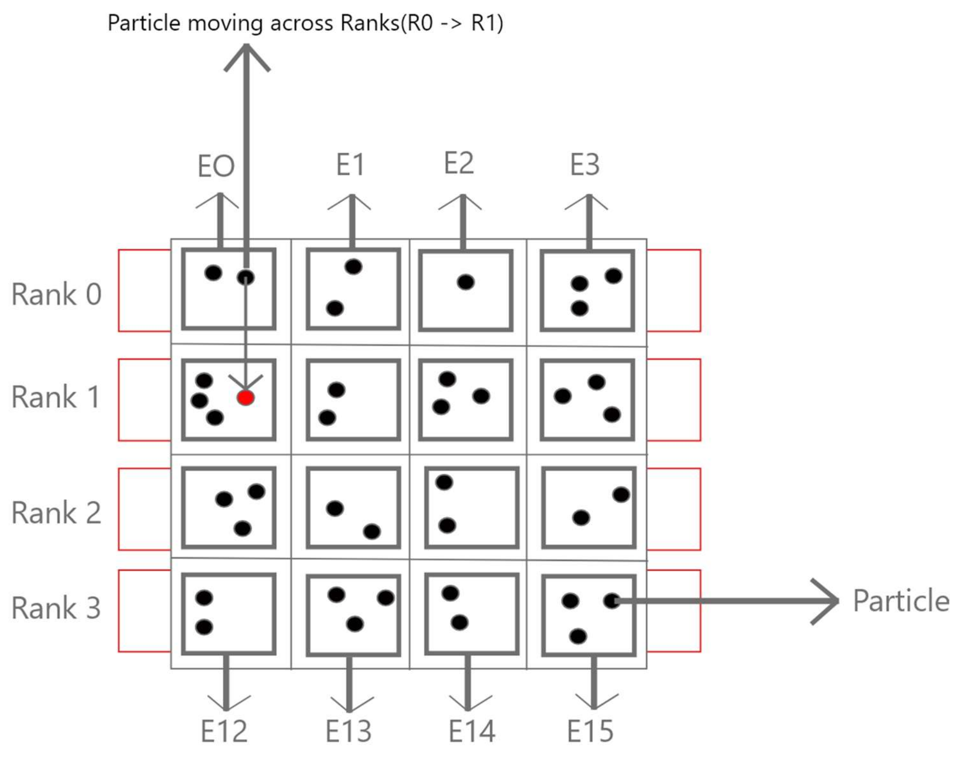


Fig-3 Movement Trace across Ranks

This movement can be detected by implementing a 2D dictionary called 'movement'. This dictionary can take 3 keys: rank1, rank2 and timestep. This information is extracted by the output of the mapping problem by parsing it for two consecutive time steps. If a movement is detected for the same gpid combination across different ranks (r1->r2) in the consecutive time steps, then the value corresponding the key combination: r1, r2, Ts is incremented. When repeated over the entire mapping problem file, a trace of all the particles is analyzed.

Synthetic Trace

Synthetic Trace is the most important aspect of this tool as this has the final application. The whole idea of a dynamic workload prediction tool is to analyze and predict the total particles and their trace for a larger scaled version of input element framework. The scaling is provided to the tool as input. Therefore, if the current element layout is a 4x4 grid, for a scaling factor of 'S=2', the layout becomes 4x8 i.e 32 elements. This is shown in Fig 4. However, an interesting question comes up in this scenario, where does the particle reside now? If the particle initially was in element Ei and now the element is scaled to {Ei0, Ei1}, where does the particle go? The tool tackles this problem by using the rand() function and randomly allocating the particle in one of the element. This is done with the assumption that the overall movement won't be affected by random allocation as in general case, we have millions of particles.

Once this trace is obtained, we now have a new layout of elements with a new allocation of particles. Therefore, if required, the mapping and movement dictionaries and outputs have to be updated. After doing this the tool now has a "synthetic trace" of how the particle would behave if the element layout is scaled.

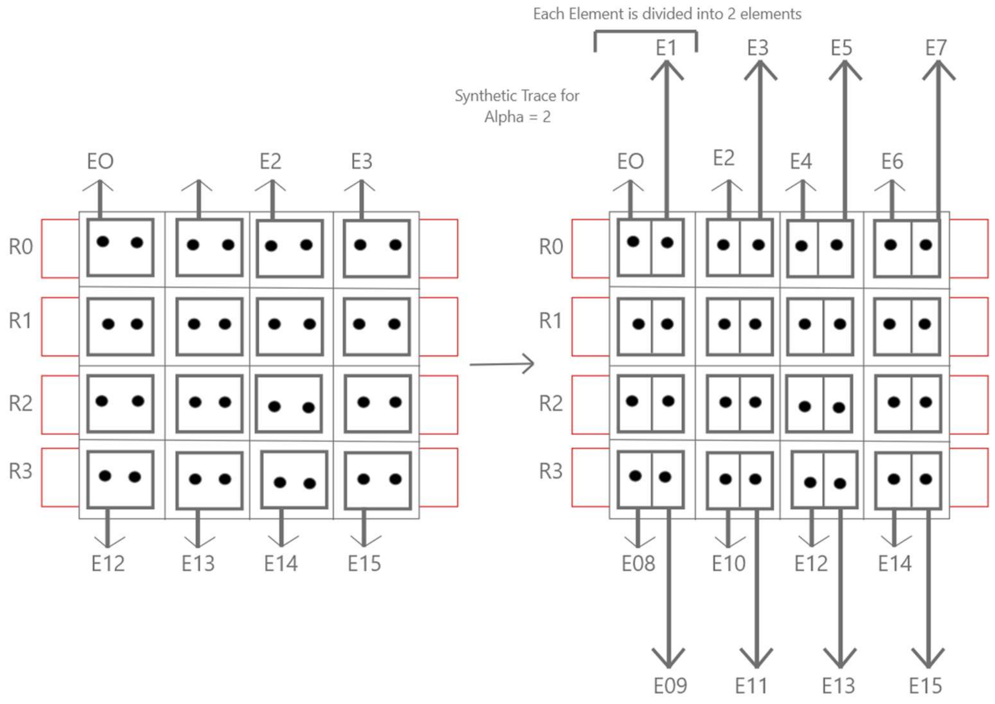

Each Element is divided into 2 elements
Synthetic Trace for Alpha = 2
EO
E2
E3
R0
R1
R2
R3
E12
E13
E14
E15
E1
E3
E5
E7
EO
E2
E4
E6
E08
E10
E12
E14
E09
E11
E13
E15

Fig 4 Synthetic Trace Element Distribution for Alpha = 2

6. Related works:

This project is based on [1]. The tool does its implementation in a serial manner while we have extended it to parallel implementation. There has been a lot of work done in relation to particle in cell algorithm and their implementations using high performance computing. With the emergence of new architectures every few years application developers are often hesitant in developing codes with the fear of their code becoming archaic after the development of new architectures. To cope with these changes *Victor k. Decyk and Tajendra V. Singh* [2] have come up with a set of new adaptable particle in cell algorithms that gives assurance to developers by finding some key features in the development of these new aforementioned architectures. The first observation made was that the architectures in future will generally be hierarchical with the lowest level of hierarchy being dominated by some tightly coupled SIMD cores which will be used for evolution in lock step, the next higher level of hierarchy will be that of loosely coupled SIMD cores with slower memory and final the highest level of hierarchy will be that of accumulation of these lower levels with message passing.

The adaptive algorithm generally deals with plasma physics applications which involves three points, 1) The deposition of particles on a grid, 2) Solving a differential equation to find the electric and magnetic fields on the grid, 3) Finding out the acceleration of particles on the grid by solving newtons equations. Traditionally the codes have been made parallel by keeping the particles and their interacting fields in the same nodes however only coarse-grained decomposition could be achieved by this. However, to achieve the fine-grained domain decomposition some particles have been kept ordered and all the particles which would interpolate are stored together.

Paulett C. Liewer [3] of the California Institute of technology also came up with a general concurrent algorithm for a particle (GCPIC) and has been the golden standard in cell simulation code. In this algorithm the physical domain of the particle in simulation was decomposed into a set of sub domains and these sub domains were then distributed among the different nodes on a machine. The most critical bottleneck of the codes involving the upgradation of the locations and the speeds of the particles were 100% efficient when increased with the increase in the number of cores used for the computation of the resources showing an extremely strong scaling. The push times and the times to deposit forming the most intensive areas of operation were also seen to evolve at a 94-97% efficiency. [4] These modifications done on Plasma physics particles gave a strong foundation in the research and parallelization of the particle in cell algorithms. The simulations were carried out on a JPL Mark III Hypercube parallel computer system.

Dynamic workload prediction is also a key part of our project and finds many applications especially in the development of multi staged multi core chips. In the cases of dynamic workload prediction several advances have been made to allow for the development of low powered, highly reliable chips. Due to uneven load balancing several hot-spots often develop in multi core chips leading to high degradation of certain parts of chips eventually leading to the failure of the whole system. *Monir Zaman, Ali Ahmadi and Yiorgos Markis* [5]. The approaches to workload prediction have historically been generally reactive (in response to the device reaching a threshold) or proactive (predictive in some sense leading to prevention of future failures). The approach by this trio involves an SVM (Support Vector Machine) which instead of reducing the training error minimizes the generalized error bound which has led to good generalization performance this technique of SVR has been adapted to the workload prediction problem and utilized to get good thermal prediction in multi core chips without Dynamic Voltage Frequency Scaling.

7. Scope:

The scope of this project can be defined as the number of numerous optimizations that can be carried out on the tool. Once the MPI implementation is done, the data gets decomposed into processors, however since this is an embarrassingly parallel implementation, the computation by each processor can be either concurrently done by threads using MPI + OpenMP or it can also be accelerated on GPUs using CUDA+MPI. Both programming models will be implemented in this project. Even further optimizations are done to remove the parallel overheads. Contention is the most likely overhead candidate because while evaluating movement detection, two timestamps are accessed. For example, processor one can be evaluating ts1 , therefore to detect a movement it has to access t2 , while a computation could also be using ts2. This contention scenario will rise for every timestamp other than ts1. Memory pattern restructuring will be used to tackle to this contention by copying the required timestamps locally in each processor. This could lead to higher memory consumption which shall be explored for larger input sizes and a tradeoff will be determined.

8. Programming Methodologies:

We will be converting the provided Java code into a serial code and providing optimization on the serial code using OpenMPI, OpenMP and CUDA methodologies. OpenMPI is a message passing paradigm which assumes partitioned address spaces and supports explicit parallelization. It provides point to point communication, collective operations, process groups and profiling interface. MPI uses commands like MPI_SEND and MPI_RECEIVE for communication between the different cores. A common methodology used for optimization is to make one of the processors as the Master and the others as Slave and allocate the work through the master to different nodes. Once the work has been fulfilled by the slave nodes, they communicate the results back to master which accumulates them and produces the final output product. OpenMP is shared address space paradigm which divides the task in hand into threads and allows for multiple threads to run on a single core, it consists of several directive and clauses which allows for optimizations of the code to be implemented. "pragma_omp" is one of the directives the spawn's different threads to be used on a rank for parallel processing. CUDA (Compute Unified Device Architecture) is a programming model an API Interface. It is used for programming NVIDIA GPU's to be used for acceleration. The basic programming structure of CUDA consists of host memory allocation, device memory allocation, transferring data to the GPU, launch the CUDA kernel, collect results from the GPU and deallocating the memory in the end.

9. Task Division:

Code Evaluation

- Responsibility: All members
- Start Date: 03/18/2019
- End Date: Ongoing

Java to C++ Conversion

- Responsibility: Kshitij Raj
- Start Date: 03/22/2019
- End Date: 03/29/2019

MPI Implementation:

- Responsibility: All members
- Start Date: 03/30/2019
- End Date: 04/06/2019

MPI + OpenMP Implementation:

- Responsibility: Devansh Gulati
- Start Date: 04/07/2019
- End Date: 04/11/2019

MPI + Cuda Implementation

- Responsibility: Kaustubh Shukla
- Start Date: 04/12/2019
- End Date: 04/17/2019

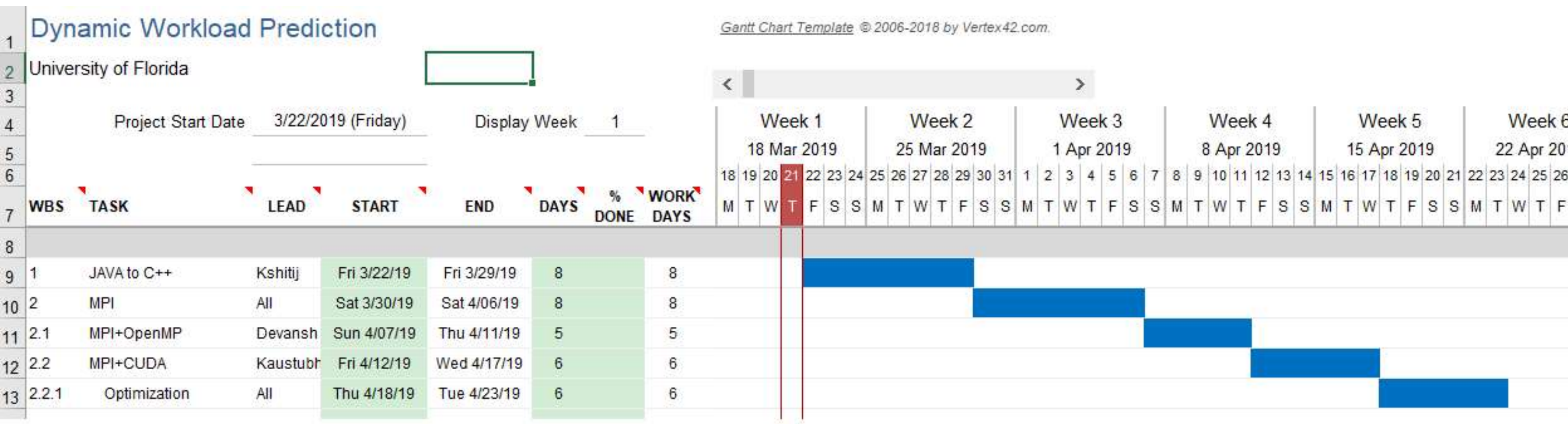
Optimization

- Responsibility: All members
- Start Date: 04/18/2019
- End Date: 04/23/2019

Report & Presentation Preparation

- Responsibility: All members
- Start Date: 04/24/2019
- End Date: 04/08/2019

10. Time and Work Schedule:



Dynamic Workload Prediction

University of Florida

Gantt Chart Template © 2006-2018 by Vertex42.com

Project Start Date 3/22/2019 (Friday) Display Week 1

Week 1 18 Mar 2019 | Week 2 25 Mar 2019 | Week 3 1 Apr 2019 | Week 4 8 Apr 2019 | Week 5 15 Apr 2019 | Week 6 22 Apr 20

WBS	TASK	LEAD	START	END	DAYS	% DONE	WORK DAYS
1	JAVA to C++	Kshitij	Fri 3/22/19	Fri 3/29/19	8		8
2	MPI	All	Sat 3/30/19	Sat 4/06/19	8		8
2.1	MPI+OpenMP	Devansh	Sun 4/07/19	Thu 4/11/19	5		5
2.2	MPI+CUDA	Kaustubh	Fri 4/12/19	Wed 4/17/19	6		6
2.2.1	Optimization	All	Thu 4/18/19	Tue 4/23/19	6		6

Fig 5 Gantt Chart

11. Facilities to be used:

We will be using HiperGator for our project. HiperGator consists of 916 compute nodes of 2 Intel E5-2698v3 (2.3GHz) processors with 32 cores each and 128GB of memory for a total of 29,312 cores and 117,248 Gb.[6] 8 processors of 28 cores each are connected to nVidia Tesla K80 GPU, 12GB Device RAM, 8 processors of 28 cores each are connected to Intel Xeon Phi 5100 Co-processor, 8 processors of 28 cores each are connected to nVidia Grid K2 GPU and 8 processors of 28 cores each are connected to nVidia Tesla M2090 GPUs, 6GB Device RAM. For MPI optimizations we will be using the processors depending on the divisions of the timestamp divisions of the projects, for CUDA optimizations we will be using Nvidia Tesla K80 GPU (this depends upon the smallest threads that we can divide the rank work) and for OpenMP we will be using threads on.

12. References:

[1] A Report on Particle Workload prediction tool – Sai Chenna

[2] Particle-in-Cell algorithms for emerging computer architectures Viktor K. Decyka,∗ , Tajendra V. Singhb a Department of Physics and Astronomy, University of California, Los Angeles, CA 90095-1547, USA b Institute for Digital Research and Education, University of California, Los Angeles, CA 90095-1547, USA

[3] A General Concurrent Algorithm for Plasma Particle-in-Cell Simulation Codes PAULETT C. LIEWER Jet Propulsion Laboratory, California Institute of Technology, Pasadena, California 91109 AND VIKTOR K. DECYK Physics Department, University of California at Los Angeles, Los Angeles, California 90024 Received June 23, 1988; revised December 14, 1988

[4] Workload Characterization and Prediction: A Pathway to Reliable Multi-core Systems Monir Zaman∗, Ali Ahmadi∗ and Yiorgos Makris∗ ∗Department of Electrical Engineering, The University of Texas at Dallas, Richardson, TX 75080, USA

[5] Particle-in-cell simulations with charge-conserving current deposition on graphic processing units Xianglong Kong,⇑ , Michael C. Huang, Chuang Ren, Viktor K. Decyk Department of Mechanical Engineering, University of Rochester, Rochester, NY 14627, USA b Laboratory for Laser Energetics, University of Rochester, Rochester, NY 14627, USA Department of Electrical and Computer Engineering, University of Rochester, Rochester, NY 14627, USA Department of Physics and Astronomy, University of Rochester, Rochester, NY 14627, USA Department of Physics and Astronomy, University of California Los Angeles, Los Angeles, CA 90095, USA

[6] https://help.rc.ufl.edu/doc/UFRC_Help_and_Documentation