Detailed explanation of optimisation strategy along with code

1. Initial model Description:

In the given model, there are two types of agent - Human and Zombie.

The initial zombie model uses the property values to create humans and zombies and moves them to random locations in a plain text dataset using repast functions. Once the created dataset is added to observer, the observer takes care of retrieving agents, calling steps on them, scheduling, migration and synchronization.

With each clock tick, the humans try to find the one of the neighboring grid cell where zombies are minimum. On the other hand the zombies move to that neighbouring grid cell where the count of humans is maximum. Thus zombies are chasing humans and humans are trying to run away from zombies.

2. Optimisation issue with Global data collection:

In initial model, output data is stored in CSV file. When the simulation was run in parallel, the Repast writes the aggregate data across processes to a single file using MPI reduction (Aggregate Data Collection) or each process can also write to the same file in turn. Both are time consuming[1]. In my experiments the MPI_reduce took around 8.45 % of the total execution time [figure included in PPT].

2.1 Why use HDF5:

If we use HDF5 we have certain advantage over using native repast functions. HDF5 is a specialized toolkit for high-performance computing. This library includes a version that permits parallelization (Here we are going to use this to get benefit of concurrent shared memory). In the HDF5 parallel version, each process maps a section of its memory from which to write, and also maps a section of the file to which to write. Each MPI process reads the input for only its portion of the data using collective MPI I/O [2].

2.2 Implementation:

To do this I moved the data to the int structured array and then used HDF5 parallel write function to transfer it to the output file. While configuring HDF5, I had set it to Use collective I/O access to the output file using “H5Pset_dxpl_mpio” function (transfer modes = H5FD_MPIO_COLLECTIVE).

Commit: Link
File name: zombies/ZombieObserver.cpp
Function name: ZombieObserver::snapshot

Additional advantage:

In the initial model, the agents are created using repast API, now with HDF5 which is a kind of native c++ library, we can create the agents which will be faster than repast API.
Commit: Link
File name: zombies/ZombieObserver.cpp
Function name: ZombieObserver::setup

3. Optimisation issue with mixing phases in ABM:

Performance analysis (with scalasca) showed that most of the execution time could be divided in three stages which are when agents

1. get(humans), CountZombiesOnPatch() : Gather information
2. face(winningPatch) : Choose a particular set of behaviors

3. move(.5) : Execute them

Scalasca showed that the first step was consuming the maximum [figure attached in PPT].
The first step can be broken down in multiple steps to reduce its execution time. With the help of scalasca, we had found out that CountZombieOnPatch and CountHumanOnPatch were taking respectively 20.34% and 5.76% of total execution time.

3.1 Observation:

It is to be noted that agents do not modify anything in phase 1 and 2, as they just evaluate potential course of actions depending on existing data. If we separate them from the action's executions then they can be simultaneously executed without risk of collision [3].

3.2 Implementation:

Divided the first step into three further steps:

ZombieObserver::gatherPatches

- Collect all patches in grid
- File name: zombies/ZombieObserver.cpp

ZombieObserver::gatherPopulation

- In every patch get the number of active agents and move this data into an int structured array which will be eventually used by the HDF5 parallel write function to transfer it to the output file.
- File name: zombies/ZombieObserver.cpp

HdfDataOutput<T>:: write

- Writing output data
- File name: zombies/hdf_dataoutput.h

Commit: Link

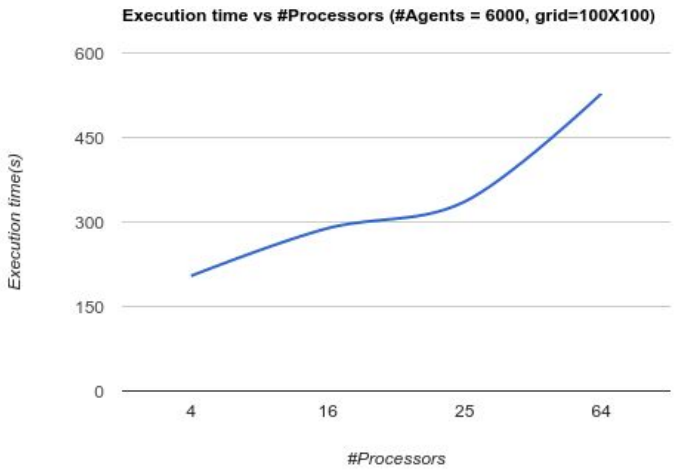
4. Optimisation Observation : Less interdependent agents are better

I tried to visualize the zombie model in a practical way.

"Let us assume there are two rival football supporters in a single closed room with full of beer. But each member of both groups are doing nothing but staring at other rival group members angrily. The atmosphere of the room will be very tensed I guess. So Even if I put more beer bottles and in case a fight breaks up between two groups, it would be a messy situation. Now let us put one TV now and instruct all of them to watch/stare that TV instead. The environment would be better compared to earlier."

I found the nature of zombie problem similar to the above scenario. In this problem both zombies and humans are trying to interact with each other. Instead of this approach If we change the behaviour of humans and zombies then we could hope to get better results. For example we could alter the behavior of human and zombie agents to look for the nearest largest chunk of human population instead of looking for each other, then the interdependence among them could reduce to a great extent. The below graph justifies this observation that for highly interdependent agents the speedup drops significantly.

Less interdependent agents = Less communication



While designing the behavior of the agents, developers should/can decide the nature of agents. They should try to design less interdependent agents.

5. Conclusion:

The main observation was that the global data collection feature of Repast HPC is a major performance bottleneck as it does not allow collecting data independently from individual agents, but only from process. Therefore HDF5 is a better option since it allows data collection from individual agents.

6. Plots

Optimisation issue 2 and 3 are deeply related with each other. Together they solve the issue of global data collection of repast hpc. All the plots correspond to optimisation issues mentioned in above sections 2 and 3 only. At the end of ticks I had calculated the overall run time of the application. So basically this gives an insight in that optimisation.

Setting of scenario:

100 X 100 grid
Human : 5000
Zombie : 1000
Stop at tick: between 20 - 100

The idea of designing less independent agents is just an observation. It was not implemented.

7. Flops

Yes I agree that there is almost non-existing numeric operations in the source code. But the zombies and humans are moving a 0.5 step each iteration. The coordinates are floating point. The only place it happened was where we made them move. I wanted to analyse the impact of this floating point operation on overall performance using scalasca. This is why I was trying to integrate PAPI with the existing system.

8. References:

1. Repast HPC manual

2. J. T. Murphy, “Computational social science and high performance computing: A case study of a simple model at large scales,” in Proceedings of the 2011 Computational Social Science Society of America Annual Conference, Santa Fe, 2011. [Online]. Available: Link
3. SIMUL 2014 : The Sixth International Conference on Advances in System Simulation. [Online]. Available: Link