Masterpraktikum Scientific Computing

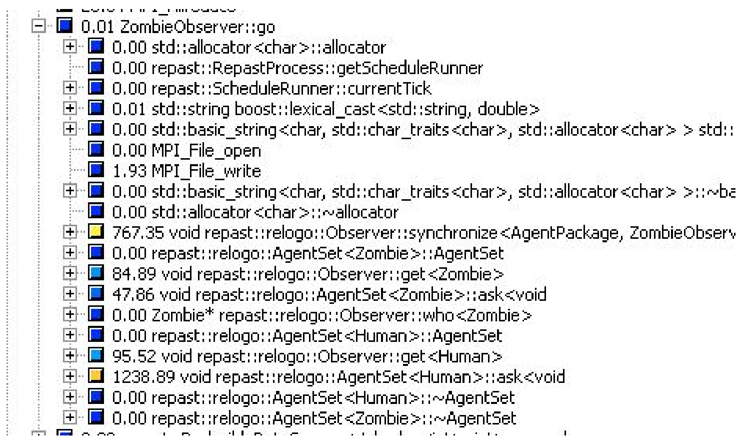
High Performance Computing Solution Sheet

Project: Optimization Strategy

Author
João Trindade (Student number : 03673251)

Analysis

To determine the main hotspots, I used scalasca to profile the code.
First the project was run for 100 ticks 5000 humans 50 zombies per patch. In a 2x2 configuration. With scalasca, the execution time was: 0:09:29



```
⊟ 0.01 ZombieObserver::go
   ⊞ 0.00 std::allocator<char>::allocator
     0.00 repast::RepastProcess::getScheduleRunner
   ⊞ 0.00 repast::ScheduleRunner::currentTick
   ⊞ 0.01 std::string boost::lexical_cast<std::string, double>
   ⊞ 0.00 std::basic_string<char, std::char_traits<char>, std::allocator<char> > std::
     0.00 MPI_File_open
     1.93 MPI_File_write
   ⊞ 0.00 std::basic_string<char, std::char_traits<char>, std::allocator<char> >::~ba
     0.00 std::allocator<char>::~allocator
   ⊞ 767.35 void repast::relogo::Observer::synchronize<AgentPackage, ZombieObserv
   ⊞ 0.00 repast::relogo::AgentSet<Zombie>::AgentSet
   ⊞ 84.89 void repast::relogo::Observer::get<Zombie>
   ⊞ 47.86 void repast::relogo::AgentSet<Zombie>::ask<void
   ⊞ 0.00 Zombie* repast::relogo::Observer::who<Zombie>
   ⊞ 0.00 repast::relogo::AgentSet<Human>::AgentSet
   ⊞ 95.52 void repast::relogo::Observer::get<Human>
   ⊞ 1238.89 void repast::relogo::AgentSet<Human>::ask<void
   ⊞ 0.00 repast::relogo::AgentSet<Human>::~AgentSet
   ⊞ 0.00 repast::relogo::AgentSet<Zombie>::~AgentSet
```

The conclusions were that most of the time is spent in a synchronisation step and in agent action processing. The synchronisation part consists of MPI Messaging The agent processing consists mainly of moving and processing the agent behaviour.

Run for 100 ticks 7500 humans 75 zombies per patch. In 2x2 configuration.
With scalasca, the execution time was 0:13:10
Problem size: 5000 -> 7500 : 50% increase
synchronize: 767.35 -> 882.57 : 15% increase
Observer::get: 84.89 -> 100.76: 18% increase
AgentSet<Zombie>::ask: 47.86 -> 86.17 : 80% increase
Observer:get<Human>: 95.52 -> 116.67 : 22% increase
AgentSet<Human>::ask: 1238 -> 1872: 50% increase

Conclusions: Agentset->ask functions increase the most with the increase in problem size.

After having identified some of the hotspots I decided to verify the output behaviour, to in the future, validate possible changes and improvements.
Project was run for different size matrixes but maintaining the overall project size to see if there are any differences in the output.
Original overall problem size was 20.000 humans and 200 zombies.
per process
5000 humans - 50 zombies - 2 x 2
2222 humans - 22 zombies - 3 x 3
1250 humans - 13 zombies - 4 x 4
800 humans - 8 zombies - 5 x 5

There were considerable differences in the outputs, after 100 ticks the 5 x 5 has more 409 infected than the 2 x 2. That's close to 10% relative to the size of the output.

Improvements

The improvement began by removing code that was not being used. In the file Human.cpp, the distance to move is being calculated, but the action is commented out. Turns out that the calculation of this distance takes considerable time. After also commenting out this code, the speedup is in the order of 5.7 for a 1 million problem size.

Then I decided to continue by parallelizing a simple for-loop with OpenMP's "#pragma omp parallel for". This was done in the turtleOn function in the Observer.h file. The speedup was small but still relevant: up to 1.2 in the larger problem sizes.

The third improvement was to add OpenMP tasks to the ZombieObserver:go function. The two functions that get the Sets of Agents (zombies and humans) can be executed in parallel. This turned out to be an interesting speedup in the larger cases, but a terrible solution for the smaller ones. To fix this, I added a Threshold to the omp task, with the if clause inside the pragma. This was not as successful as expected, even after trying with different thresholds the performance was still bad. I then decided to manually test the threshold and select what code to execute, instead of relying in the openmp "IF" clause inside the pragma. This resulted in the behaviour I was looking for. The speedup was up to 1.7 in the larger cases while not changing in the lower (bellow the threshold) test cases.

Next I decided to apply the same approach I had with the turtleOn (using the omp parallel for) in two other functions: agentset::ask and observer::maxOneOf. In the first case, the performance was worst than without the pragma, maybe it was because of the problem size I was using was already too small to test every improvement at the same time. On the other hand, the maxOneOf function crashed with a seg.fault when with the pragma.

Conclusion

Improving this repast simulation was particularly difficult giving the amount of abstract classes and templates used. They create a layer of complexity that make using the tools we learnt very challenging.
For this, scalasca proved invaluable, it was helpful not only to identify the main hotspots, but also as a simple way of determining the call path in the code execution.
In the end, the result was good, the speedup ranged from around 7 to 2 depending on the problem size, this taking into account the removing of the Human::step() code. If we take this code into consideration the speedup is around 2.5, as show in the Spreadsheet "Improvement Analysis" - 1M Problem size page.