Repast Computing Model

Lecture: Parallel Programming (IN 2147)
Prof. Dr. Michael Gerndt

Author
Mantosh Kumar, 03662915
June 22, 2016

Repast HPC History

- **Repast:** Recursive Porous Agent Simulation Toolkit
- **First version:** Repast HPC 1.0 beta (2010)
- **Developed by:** Argonne National Laboratory's Decision and Information Sciences Division

A free open source toolkit

Repast HPC Introduction

- An agent based modeling and simulation framework for high performance distributed computing platforms
- Written in C++
- Uses MPI for parallel operations
- Supports parallelizing the simulation world at prominent modeling
- Supports various projection methodologies (shared grid, shared network and shared continuous space) independently or in combination

Why Repast HPC?

- Agent Based Models (ABM) are becoming large
- Most of the other open source ABM frameworks do not support parallelization

- Repast HPC offers parallel platform to ABM models

- Easy to use: Communication and synchronization are handled automatically by Repast HPC framework

ABM + MPI = Repast HPC

Target Application

- Large scale agent based simulation model
- Social simulation
- A simulation amenable to a parallel computation

Aim: ***Realization of next generation ABM systems explicitly focused on large-scale distributed computing platforms***

Repast HPC Simulation Components (Example: Zombie model)

- Agent Class
  - Implemented as C++ classes
  - Attribute of class = Agent's state
  - Method of class = Agent's behavior

```
class Zombie : public repast::relogo::Turtle {
public:
    Zombie(repast::AgentId id, repast::relogo::Observer* obs) :
    repast::relogo::Turtle(id, obs) {}

    virtual ~Zombie() {}
    // zombie behavior executed every iteration
    void step();
    void infect(Human* human);
};
```

Repast HPC Simulation Components
(Example: Zombie model)

- Package-type code: Contains minimal amount of agent state necessary to copy an agent from one process to another one

```
struct AgentPackage {
    template<class Archive>
    void serialize(Archive& ar, const unsigned int version) {
        ar & id;
        ar & proc;
        ar & type;

        ar & infectionTime;
        ar & infected;
    }

    int id, proc, type;

    int infectionTime;
    bool infected;

    repast::AgentId getId() const {
        return repast::AgentId(id, proc, type);
    }
};
```

Repast HPC Simulation Components (Example: Zombie model)

- Model class: Responsible to initialize the simulation

```
class Model {

private:

    int rank;

public:
    repast::SharedContext<ModelAgent> agents;
    repast::SharedNetwork<ModelAgent, ModelEdge>* net;
    repast::SharedGrids<ModelAgent>::SharedWrappedGrid* grid;
    repast::DataSet* dataSet;

    Model();
    virtual ~Model();
    void initSchedule();
    void step();
}
```

Repast HPC Simulation Components (Example: Zombie model)

- A main function: Initializes the MPI and Repast HPC environments, creates the model class and triggers the Time Scheduler(API: SimulationRunner)

```
int main(int argc, char **argv) {
    mpi::environment env(argc, argv);
    std::string config = argv[1];
    std::string props = argv[2];

    RepastProcess::init(config);
    runZombies(props);

    RepastProcess::instance()->done();
    return 0;
}
```

Repast HPC Simulation Components (Example: Zombie model)

- **main:runZmbies:** uses *SimulationRunner* to run the simulation
- **SimulationRunner :** Repast HPC API for Time Scheduler

```
void runZombies(std::string propsFile) {
    Properties props(propsFile);
    SimulationRunner runner;
    runner.run<ZombieObserver, Patch>(props);
}
```

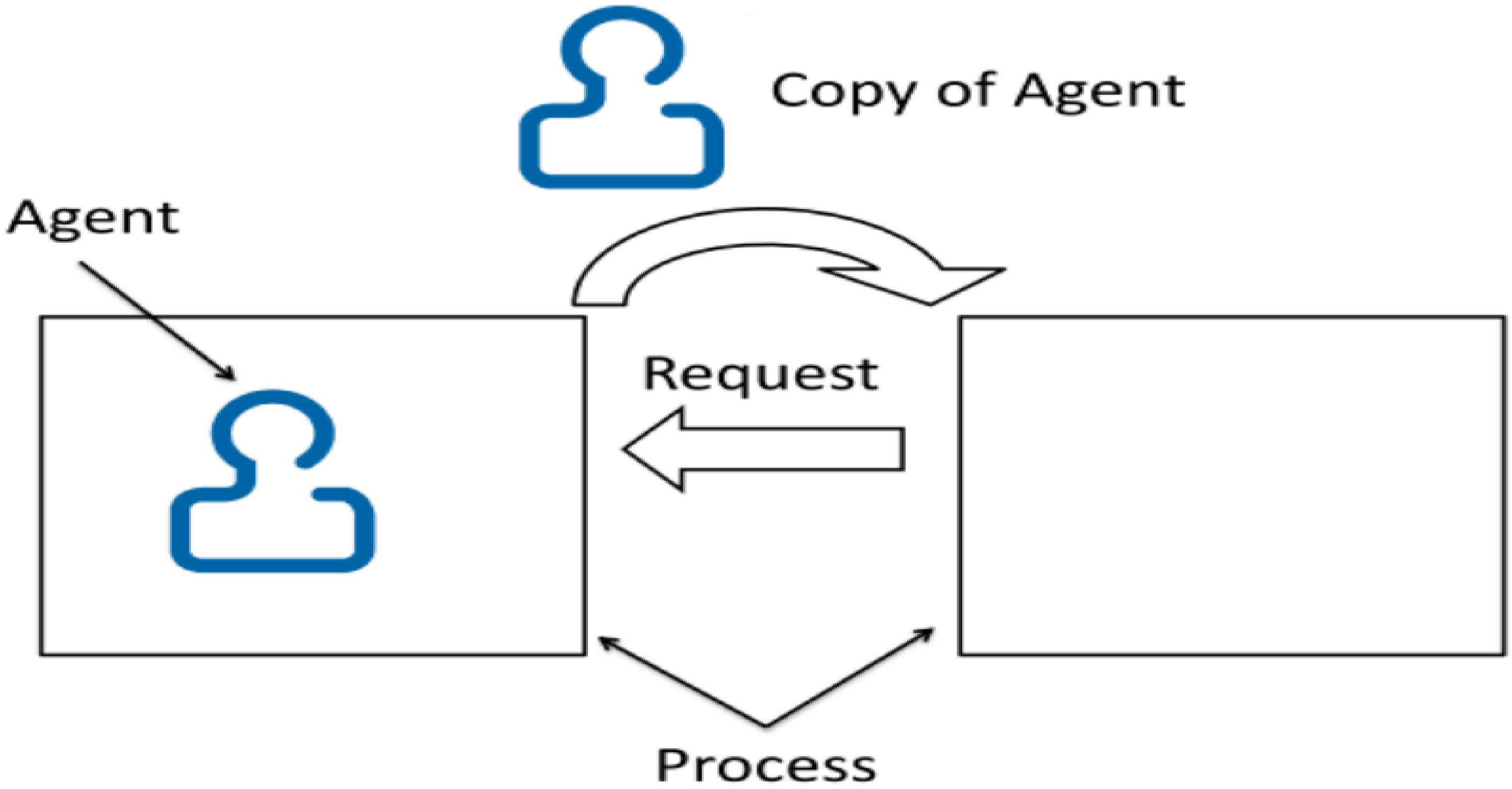
A Simulation Step (Example: Zombie model)

```
void ZombieObserver::go() {
    // get the zombies and call step on them
    AgentSet<Zombie> zombies;
    get(zombies);
    zombies.ask(&Zombie::step);

    // get the humans and call step on them
    AgentSet<Human> humans;
    get(humans);
    humans.ask(&Human::step);

    // perform cross process synchronization
    initSynchronize();
    synchronizeTurtleStatus<AgentPackage> (*this, *this);
    synchronizeTurtleCrossPMovement();
    synchronizeBuffers<AgentPackage> (*this, *this);

    if (_rank == 0) {
        std::cout << RepastProcess::instance()->
            getScheduleRunner().currentTick()  << std::endl;
    }
}
```

- Scheduled to execute every tick of simulation
- Gets AgentSets of Zombies & Humans
- Call step method on them

- **Synchronization**
  - Initialization call
  - Sync agent status
  - Migrate any moved agents
  - Sync buffers between processes

How agent info is shared across processes



Source: [4]

Repast HPC Features

- Synchronization scheduling of events
- Global data collection
- Automatic management of agent interactions across processes
- Offers a suite of features that specifically facilitate creating, executing and evaluating agent-based models

Repast HPC does much of the parallel programming of an agent simulation

Repast HPC Optimization Observations

- Global data collection feature of Repast HPC is a major performance bottleneck
- Execution time scales almost linearly with the number of cores for mainly independent agents
- Speedup drops significantly in case of highly interdependent agents
- Gathering information from agents consumes lot of time
- Less interdependent agents are better ⇒ Less Communication

References

1) Collier, N., Repast HPC Manual. 1-43 (2013)
2) Presentation source code (https://gitlab.lrz.de/ga78boj/hpcLabProjectRepastHPC)
3) Original Source Code (https://github.com/scamicha/SCC15HPCRepast)
4) Segawa, S. Kin, S. et.al. Implementation of Massive Agent Model Using Repast HPC and GPU. 1-5 (2014)

Thank You!

Questions?