CharmPy:
Charm++ with Python

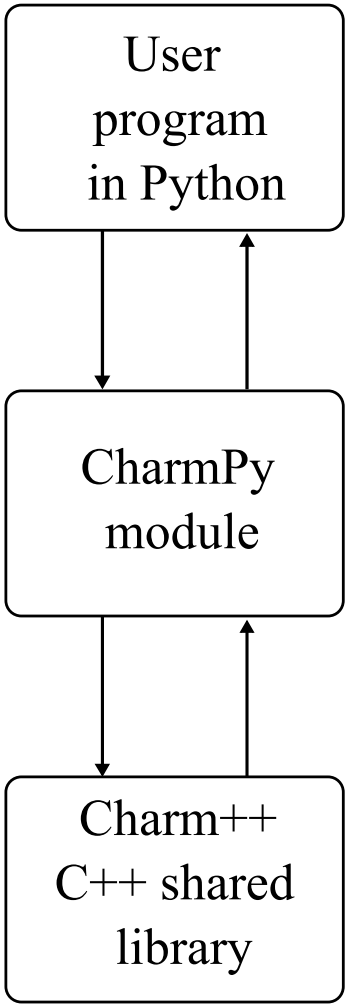
Juan Galvez

PPL UIUC PARALLEL PROGRAMMING LABORATORY

ILLINOIS
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

What is CharmPy?

- Write Charm++ programs in Python
- Core runtime functionality in C++



Why Charm++ for Python?

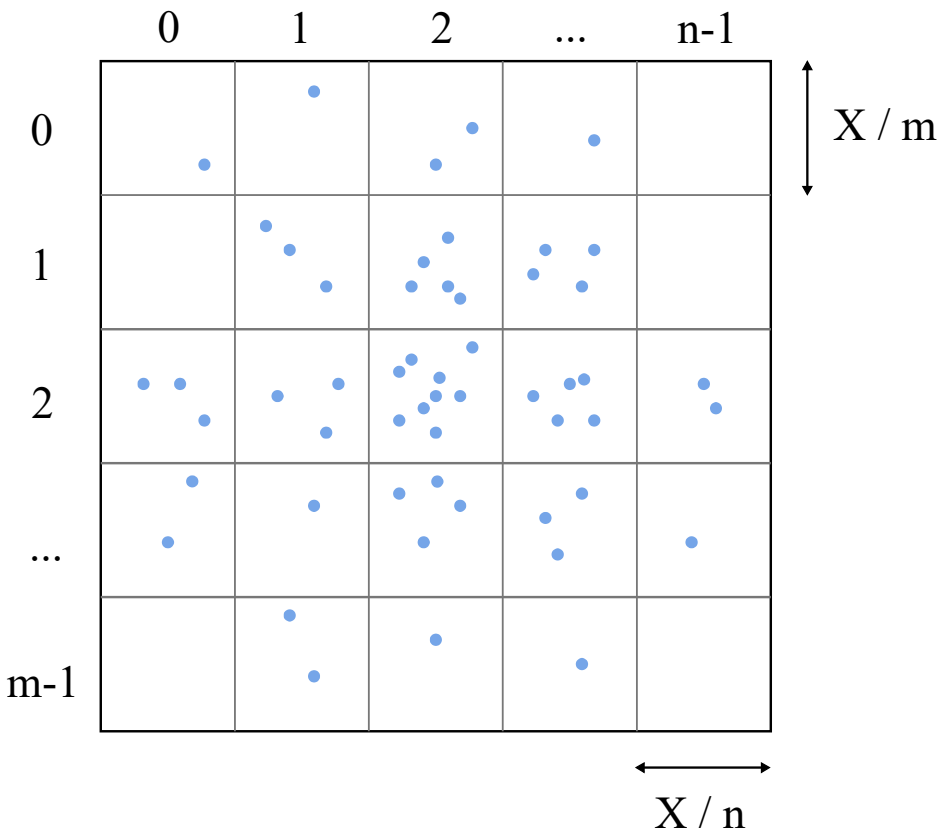
- CharmPy specific features
  - No .ci file
  - Entry methods can receive any type of data (including custom types) without special declaration
  - Any entry method can be target of reductions and callbacks receiving any type of data
    - No special tag, declaration, no CkReductionMsg
  - No need to use or define Charm++ message types
  - No need to write PUP (serialization) routines
  - Streamlined and easy-to-use API

Why Charm++ for Python?

- Productivity (typically much less code)
  - Python has powerful high-level language features and extensive set of libraries
  - Fast prototyping and testing
- Integration with rich set of Python libraries (visualization, numerical, scientific, machine learning, data sets...)
- Performance can be comparable to C/C++ depending on techniques used

Particle example

- 2D $X \times X$ box decomposed into cells → 2D ($m \times n$) chare array
- Varying number of particles per cell
- Each iteration, particles move random distance d < cell size in any direction
- Overdecomposition: multiple cells (chares) per core
- Load balancing every L iterations
- Full program is one file (97 lines)
- Runs on a supercomputer



Particle example

- Particle objects
  - Will be sent between chares (no pup method needed)

```
class Particle:
  def __init__(self, x, y):
  ...
```

- Readonlies container (global object)
  - Objects in container will be broadcast to all processes after mainchare constructor. Examples:

```
ro = readOnlies()
ro.SIM_BOX_SIZE = (100.0, 200.0)
ro.hi = “hello world”
ro.test_particle = Particle(21.2, 45.0)
```

Particle example

- Mainchare constructor

```
class Main(Mainchare):
  def __init__(self, args):
    ro.mainProxy = self.thisProxy
    ro.cellProxy = charm.CellProxy.ckNew((12,12)) # 12x12 array
    ro.cellProxy.run()   # invokes ‘run’ on every cell
```

- Cell (entry method invocation)

```
def run(self):
  ...
  for nb in self.neighbors: # nb is 2-tuple (elem index)
    ro.cellProxy[nb].updateNeighbor(self.iteration,
                                    outgoingParticles[nb])

# where outgoingParticles[nb] is Python list of Particle
# objects to send to neighbor nb
```

Particle example

- Entry methods and “when” construct
  - Entry method invoked when first argument equals member variable

```
@when(“iteration”)
def updateNeighbor(self, iter, particles):
    self.particles += particles
    ...
```

- Reductions

```
self.contribute(len(self.particles), Reducer.max,
                Main.collectStats, ro.mainProxy)

Entry method in Main class:
def collectStats(self, result): print result
```

Performance

- NumPy + Numba
  - Numba can compile array-oriented and math Python code
  - Python can act as high-level language with critical code being non-interpreted (compiled)
  - Numba can also generate GPU code
- CharmPy compatible with other Python interpreters (e.g. PyPy that uses JIT)
- Easy to run C/C++ code, solutions to run Fortran code also exist

Thank you

Questions?

git clone https://charm.cs.illinois.edu/gerrit/charmpy