Concurrent GARTH (Genetic AlgoRiTHms) using C++11

A Concurrent Framework for Genetic Algorithms

J. Caleb Wherry
Virginia Tech
Department of Engineering Science & Mechanics

Outline

I. Introduction & Motivations

II. Application Area

III. Genetic Algorithms & Framework

~~IV. Concurrent Methods & Implementation~~ (Timing)

V. Results

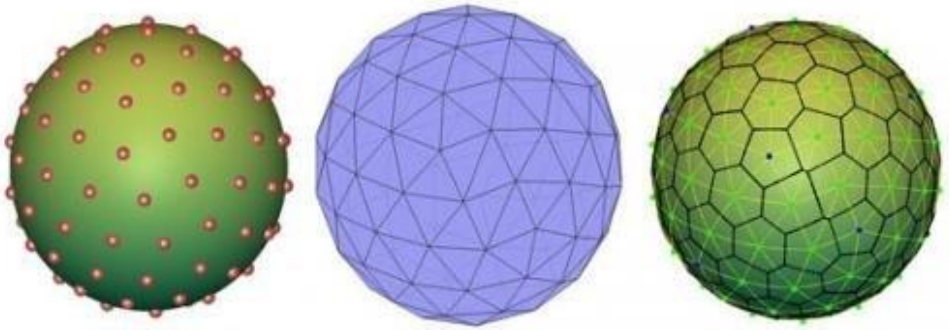
VI. Future Work

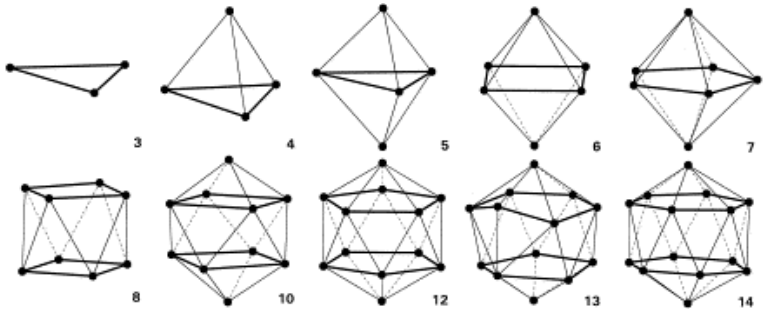
Intro & Motivations

- Problem Statement:
  - Create a cross-platform Genetic Algorithm framework using native C++11 features
- Goals:
  - Integrate concurrent principles wherever possible
  - Apply framework to real world problem
  - Test cross-compiler / cross-platform C++11 standard support
    - Specifically new native thread support
  - Performance benchmark of framework against sequential GAs
- Software Engineering Notes:
  - Open Source (MIT License)
    - http://github.com/calebwherry/Concurrent-GARTH
  - Cmake build system
    - Cross-compiler/cross-platform development
  - Python wrapper for cross-platform testing
  - Library Unit Tests (Google Test)
  - Travis-CI (Continuous Integration)

Application Area

Thomson Problem





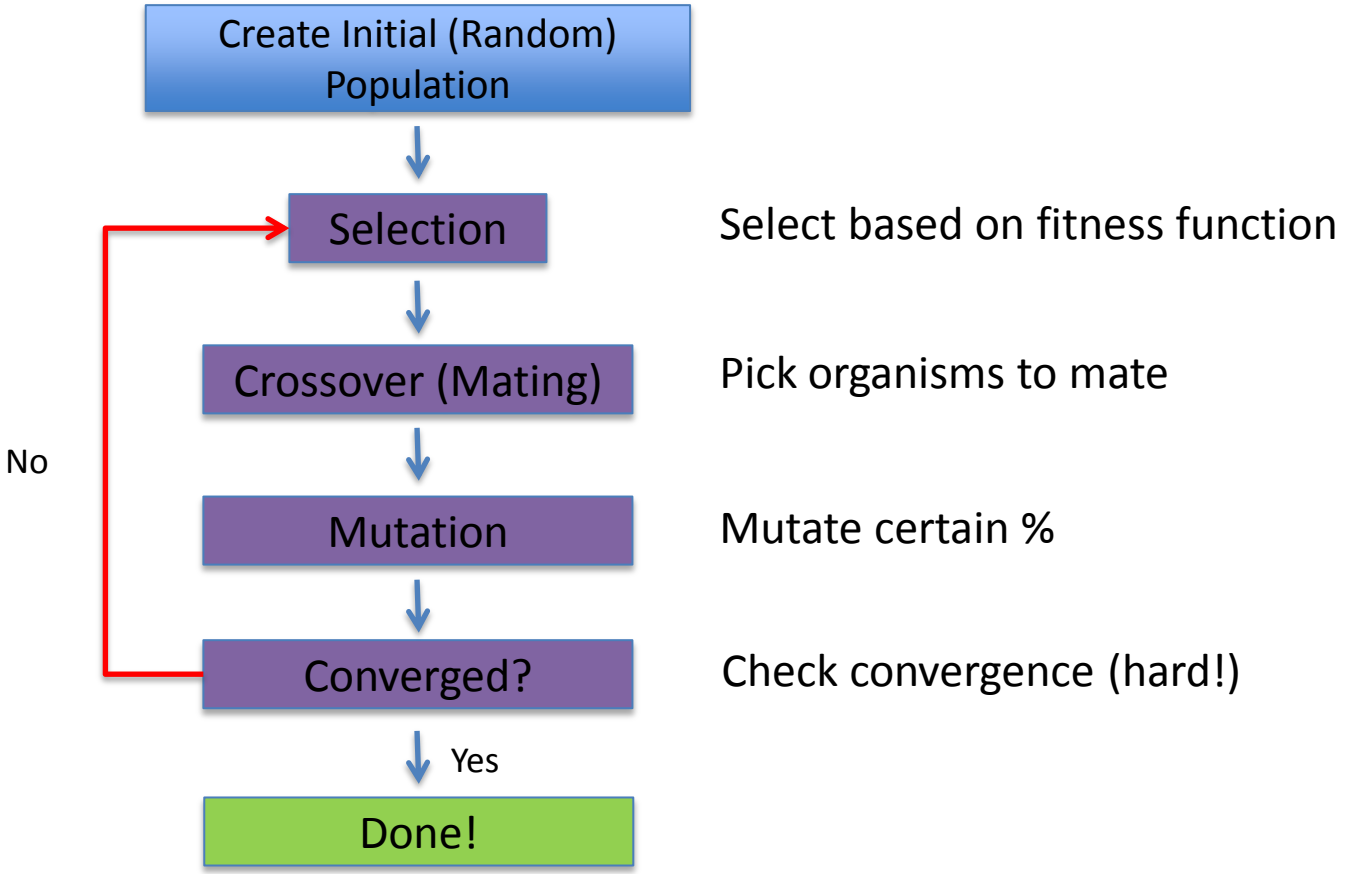
Minimize Electrostatic Energy Between All Charged Particles

GA nomenclature: *Fitness Function*

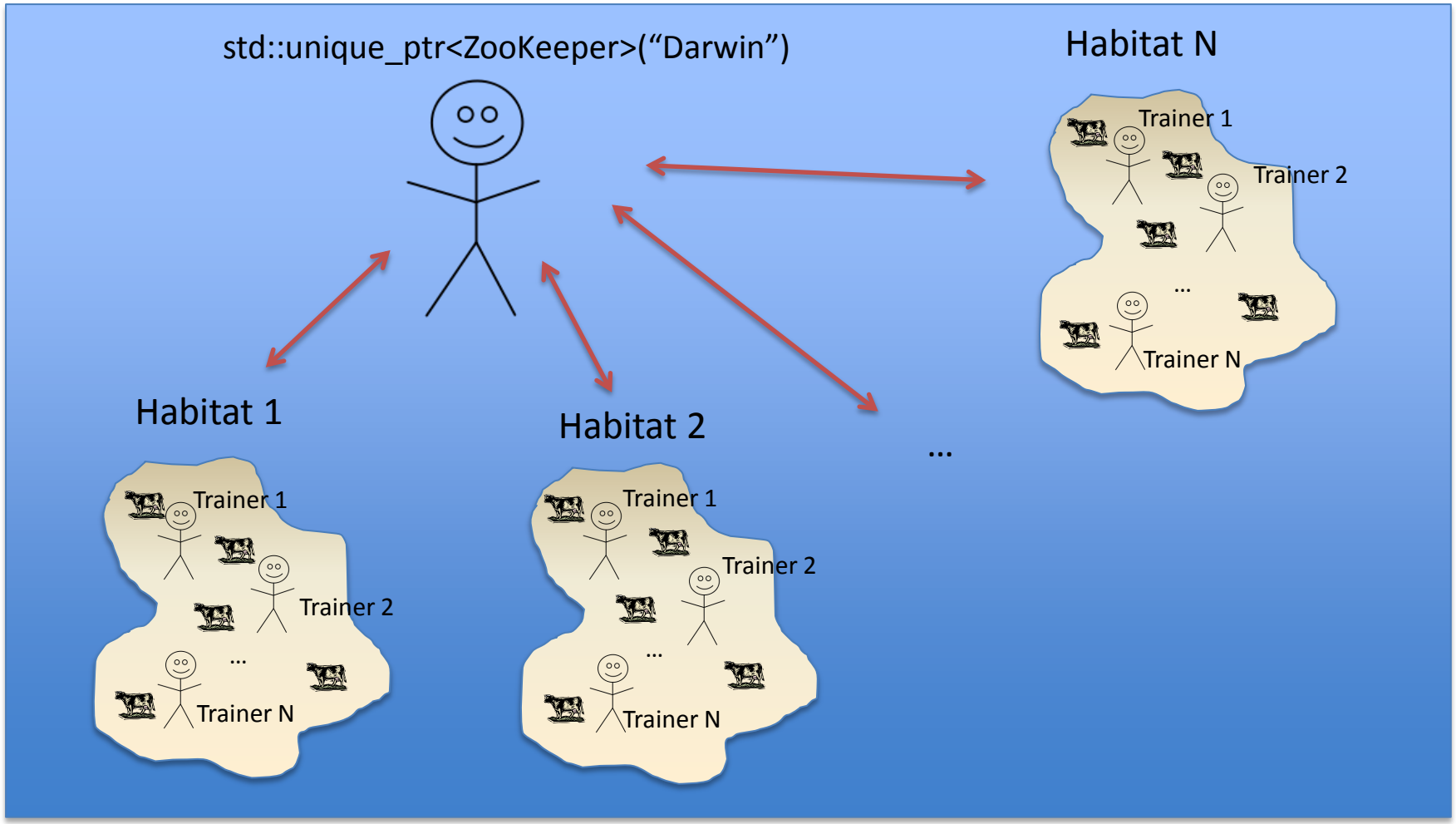
$$U = \frac{q^2}{4\pi\epsilon_0} \sum_{i>j=1}^{n_c} \frac{1}{|\vec{r}_i - \vec{r}_j|}$$

Genetic Algorithms

- Based on principles from Darwin's *Origin of Species*
  - Natural Selection
- Very powerful for non-linear optimization problems
  - Selection & Mutation drive exploration of search space
  - Larger population = better searching of space



Genetic Algorithm Framework



*Habitat = CPU, Trainer = Core/Thread, Cow = Organism

Results

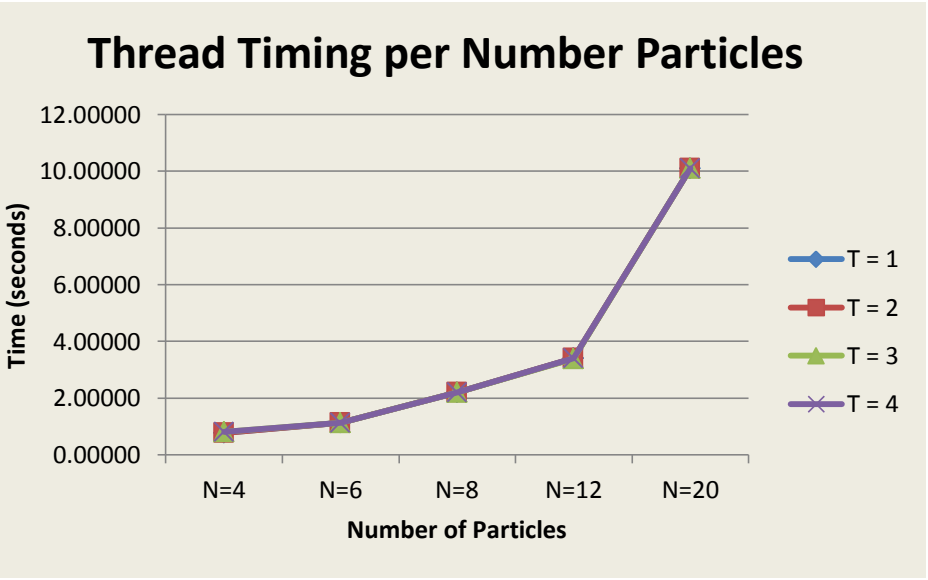
Cross-Compiler / Cross-Platform Results

	Windows 7	Mac OS 10.8	Linux - Ubuntu 12.04 LTE
GNU GCC 4.8	-	✓	✓
Clang 3.4	✓	✓	-
MinGW 4.8	✓	-	-
MSVC 18.0	✓	-	-
Intel 14.0	✓	-	-

Results

Mini-Benchmark Performance Results

Intel® Core™ i5-2500K CPU @ 3.30GHz (4 core)



- Results averaged over 5 runs
- No speedup, wat?!
  - Not using ThreadPool?
  - Overhead of OOP?
  - Bad Design?
  - Simple organism fitness?

Number Particles	T = 1	T = 2	T = 3	T = 4
N=4	0.76831	0.78415	0.80762	0.80893
N=6	1.12857	1.13520	1.13676	1.13939
N=8	2.20581	2.21002	2.20900	2.20811
N=12	3.39890	3.39945	3.39669	3.39931
N=20	10.10100	10.10144	10.10119	10.10109

Future Work

- Deeper scalability testing on hardware with many cores
  - Xeon Phi (57+ cores)
- Fully Implement ThreadPool and Barrier
  - Reason why mini-performance results are not good.
  - Not native to C++11, might be in C++14 or C++17?
- Scalability testing using HPC Systems
  - MPI (not done for this project since not natively supported)
- Memory Testing
  - Does OOP design effect anything? Probably
  - Does overhead of std::shared_ptr<T>, etc effect anything?
- Parallelize other parts of GA
  - Mutation, selection, sorting, etc.

References

[1] M. Herlihy and N. Shavit. The Art of Multiprocessor Programming. Morgan Kaufmann Publishers, 2012.

[2] M. K¨ppen and E. Dimitriadou. Concurrent Application of Genetic Algorithm in Pattern Recognition, pages 868–877. IOS Press, Amsterdam, The Netherlands, The Netherlands, 2003.

[3] T. Pang. Introduction to Computational Physics. Cambridge University Press, Cambridge, 2006.

[4] A. Williams. C++ Concurrency In Action: Practical Multithreading. Manning Publications, 2012.

[5] http://tracer.lcc.uma.es/problems/thomson/img/electr.gif

[6] https://www.maths.unsw.edu.au/sites/default/files/imagecache/default_content/me100im1.jpg

Questions

Questions &| Comments?

N = 12, ~10,000 generations