CSCI 4/5576: The Random Logistic Map

Amy Le (5576)
Long Tat (4576)
Emily Bertelson (4576)
Kristina Entzel (4576)

December 17, 2014

1 Abstract

The purpose of this investigation was to explore and simulate a spatially random logistic map using a dynamic load balancer on Janus, the CU supercomputer. The recursive nature of the map prevents the individual fixed point iterations from being parallelized, but a set of iterations maybe load balanced over many cores. The original simulation was written in serial code in MATLAB. Our solution has modified the original version for efficiency, speed, and scalability. We implemented our simulation in C++ and present our results in terms of a histogram of observed periodic orbits and a bifurcation diagram. Single core optimization techniques, such as SIMD loop vectorization and function inlining, as well as using a dynamic load balancer for more efficient work distribution were applied. The HDF5 file format was used to store the simulation results in a better archival format. The benchmarking (weak scaling study) results imply the best speedup and efficiency is gained when invoking the load balancer on one node (12 cores), although we tested our simulation over 16 nodes (192 cores). Improvements to this project include optimizing the post-simulation data processing.

2 Introduction

The Logistic map is a quadratic recursive equation on the domain [0,1]. It is a popularly studied topic in nonlinear dynamics and has applications in population modeling. There is one parameter in the expression, r, which can take any value in the range [0,4].

$$x_{n+1} = r x_n (1 - x_n)$$

For values of $r \in [3.5, 4]$, the system experiences the onset of chaos. Between $r \in [0, 3.5]$, we observe stable periodic orbits.

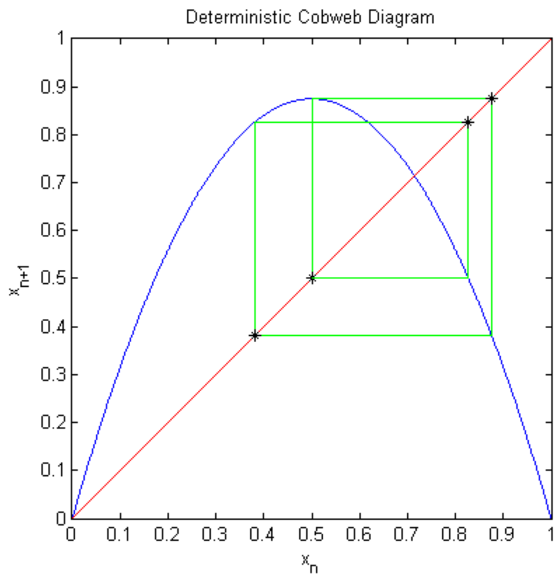

Figure 1: Deterministic Logistic Map (blue) for $r = 3.2$. There is a stable period 4 orbit. The order of the period is calculated by counting the number of crossings (green) on the line $x_{n+1} = x_n$ (red).

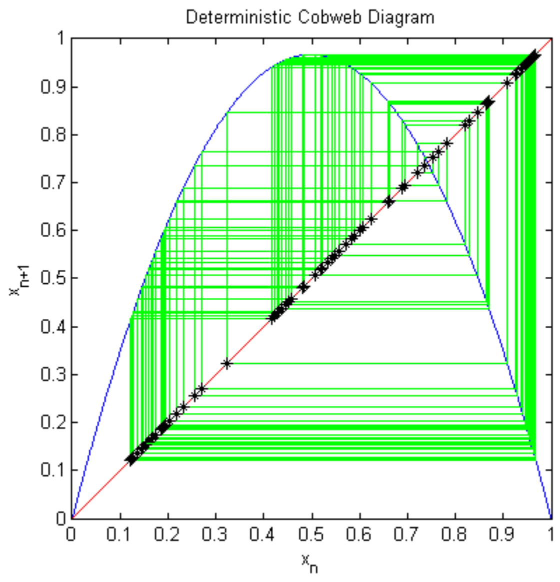

Figure 2: Deterministic Logistic Map (blue) for $r = 3.8$. There is a no stable orbit.

There are two ways to vary the deterministic map. We can simulate the parameter r as a function of time or space. The existing literature explore the notion of randomness in time, so we explore r as a function of space [1].

The following equation is the fixed point iteration that the code completes, where $R(x)$ is calculated by manipulating a random number generator.

$$x_{n+1} = R(x_n)x_n(1 - x_n)$$

The exact details of how to calculate $R(x)$ are outlined below.

$$\begin{aligned}
\ln(R(x)) &= \xi(x) \\
\xi(x) &= \ln(r) + 2\sum_{n=1}^{N} a_n \cos(2\pi n x) - b_n \sin(2\pi n x) \\
a_n, b_n &\sim Unif(-M_n, M_n) \\
M_n &= \sqrt{1.5 S_n} \\
S_n &= \alpha e^{-L|n|} \\
\alpha &= \sigma^2 \tanh(L/2) \\
\sigma &< \ln(4/r)\frac{\tanh(L/4)}{\sqrt{1.5\tanh(L/2)}}
\end{aligned}$$

Where $L \in (0, 1)$ represents the correlation length (and is fixed for each simulation) and $r \in [0, 4]$ is also fixed for each simulation.

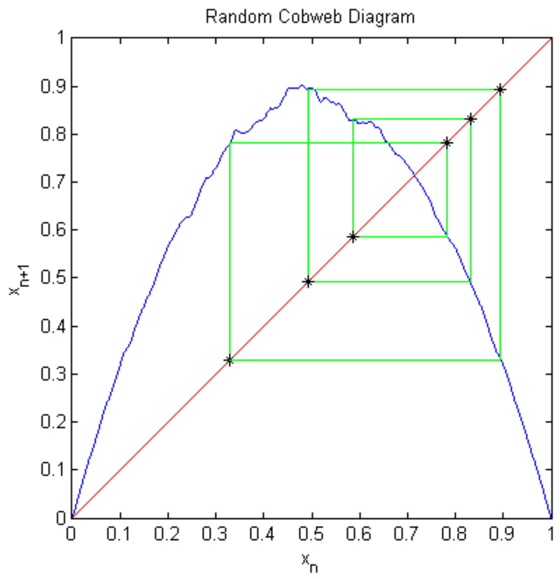

Figure 3: One instance of a random logistic map (blue). The map has converged to a stable period 6 orbit (green). Notice the “wiggliness” in the parabola shape.

Other instances of the random map would vary from this realization, due to the random nature of the map.

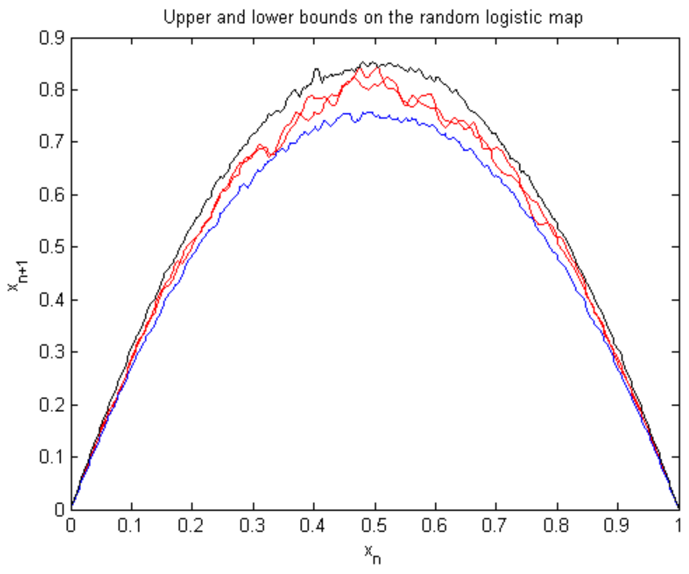

Figure 4: A coarse demonstration of the upper (black) and lower (blue) bounds of the random logistic map. Sample realizations are shown in red.

Since the map can take on a range of values for any given position in space, it would be useful to characterize some of its properties. In particular, we will be studying the stability of the map, which includes locating fixed points and generating bifurcation diagrams. The two main goals are:

1. Find the expected number of order p periodic orbits for a the random map ($p = 1, 2, 3, ...$)
   (a) For an initial starting value $x_0 \in [0, 1]$ and a specific random function $R_0(x)$, iterate until you find the fixed point(s), x_i^* associated with $R_0(x)$.
   (b) Classify the fixed points in terms of a period p orbit.
   (c) Each processor should take a different initial x_0 and report whether the initial condition led to finding a unique stable orbit
      - The processors should be properly load balanced.
      - As each processor finishes its work, it will write its results to an HDF5 file (parallel i/o)
   (d) Repeat the above steps for a large number of different random maps $R_i(x)$, $i = 0, 1, 2, ...\bar{N}$ in order to find the expected number of order p periodic orbits for the random map.
2. Create a set valued bifurcation diagram [4]
   (a) For many values of $r \in [0, 4]$, and a fixed random function $R_0(x)$, plot the locations of the periodic orbits as a function of r. A period p orbit will have p corresponding

x values as its orbit locations (e.g. a period 1 orbit will have 1 fixed point, a period 2 orbit will have 2 fixed points, and so on).

- Use a HDF5 file to store the simulation data and as a source to generate bifurcation diagrams

As the map has an element of randomness to it, many simulations (a large $\bar{N}$) would be required for statistical analysis.

3 Method

The general progression of the project is outlined below:

1. Convert the Matlab code to C++
2. Confirm the C++ versions of the code that we each produce work together correctly by comparing to the serial version
3. Invoke the load balancer to assign an initial condition to each fixed point iteration
4. Benchmark: strong scaling study (speedup and efficiency)

The table below summarizes how we have subdivided the project among ourselves.

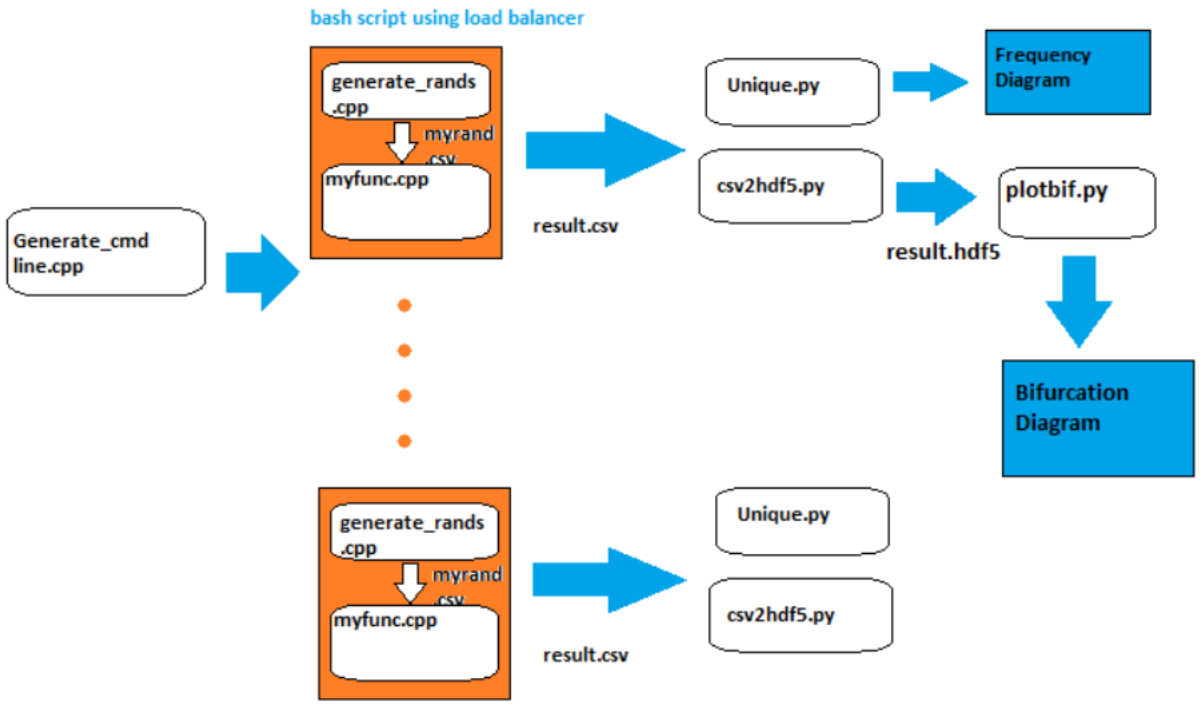

Figure 5: Workflow

3.1 Single Core Optimization

Optimizations implemented in the code conversion:

- Preferential use of the multiply and add operators where possible, since they less expensive than subtract and divide operators
- I used a reduction on the loop that computes the Fourier Series in order to take advantage of the data parallelism with SIMD
- Loop structure was reorganized to take advantage of C++ being row-oriented
- I used inlining in the C++ code to reduce the number of function calls
- The lack of a built in uniform random number generator that generates a random double between two doubles made me create a psudeo random number implementation with the use of `rand` and `srand`.

```
/*Produce a random number in the range [a,b]*/
double rand_draw(double a, double b) {
  double random = ((double) rand()) / (double) RAND_MAX;
  double diff = b - a;
  double r = random * diff;
  return a + r;
}
```

3.2 Load Balancer

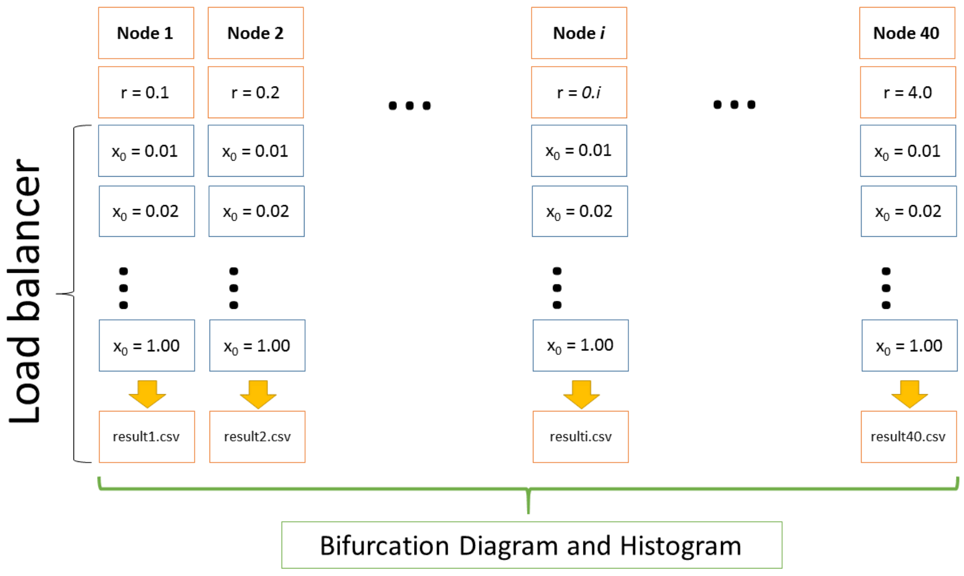

Figure 6: Load balancer.

We researched types of load balancers [5]. We found there are many strategies for load balancing [6].

3.3 HDF5

A structure for the file has been devised:

```
group "$r$"
    group "$L$"
        group "$p = 1$"
            dataset
                $(x_1)$
                $(x_2)$
                $(x_3)$
                ...
        group "$p = 2$"
            dataset
                $(x_{11}, x_{12})$
                $(x_{21}, x_{22})$
                ...


group "$L$"
    dataset
        $(x_1, r, p = 1)$
        $(x_1, x_2, r, p = 2)$
        ...
        $(x_1, x_2, ...x_k, r, p = k)$
```

From there, HDF5 compatibility will be implemented in code where relevant [2] [3].

4 Results

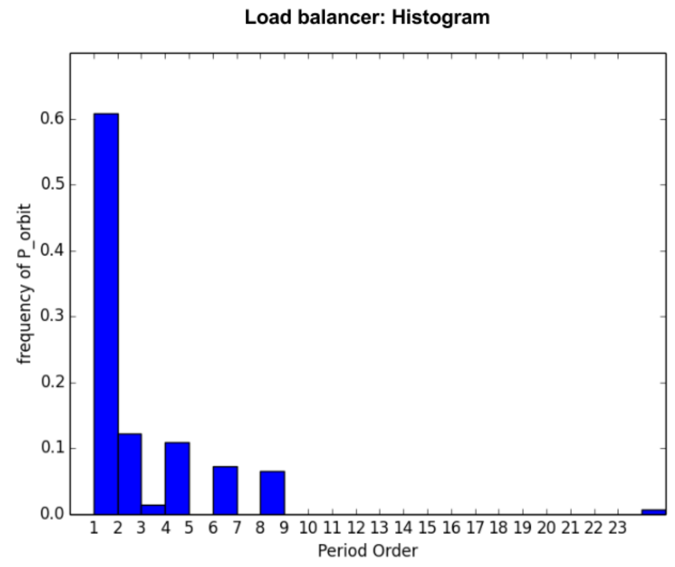

Figure 7: load balanced histogram

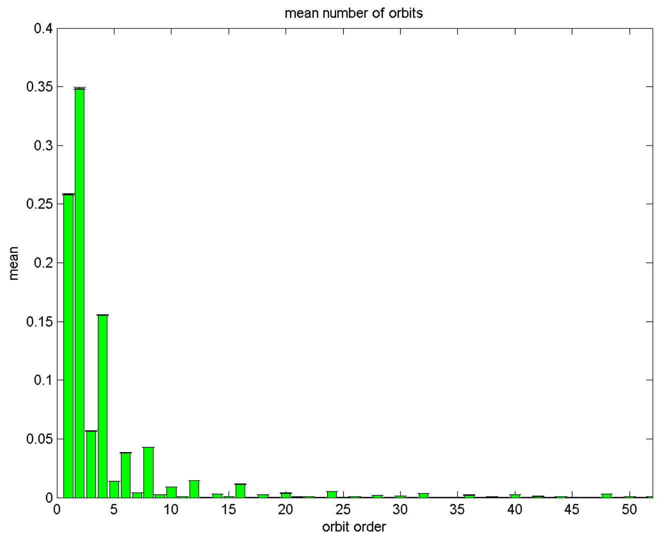

Figure 8: serial histogram

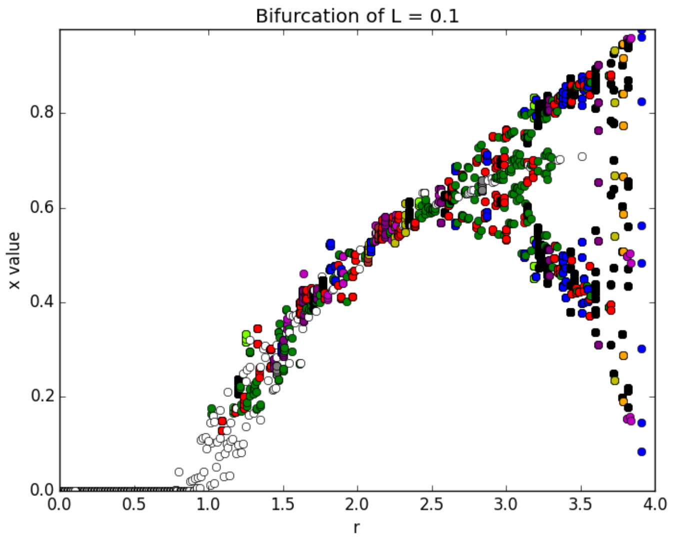

Figure 9: Bifurcation with $L = 0.1$

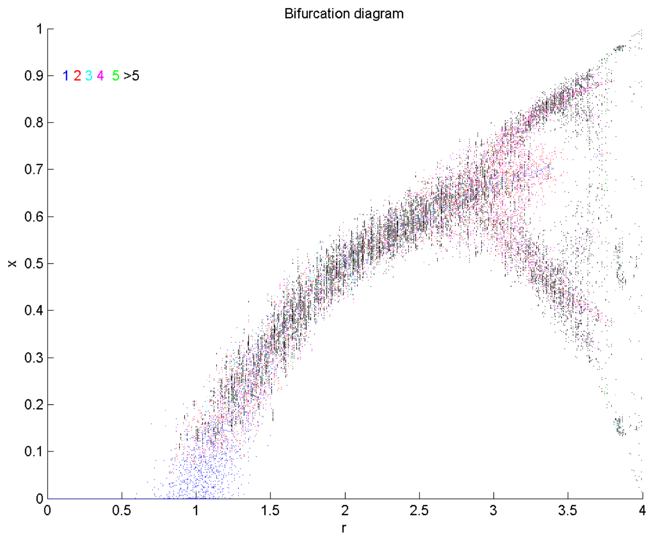

Figure 10: serial bifurcation

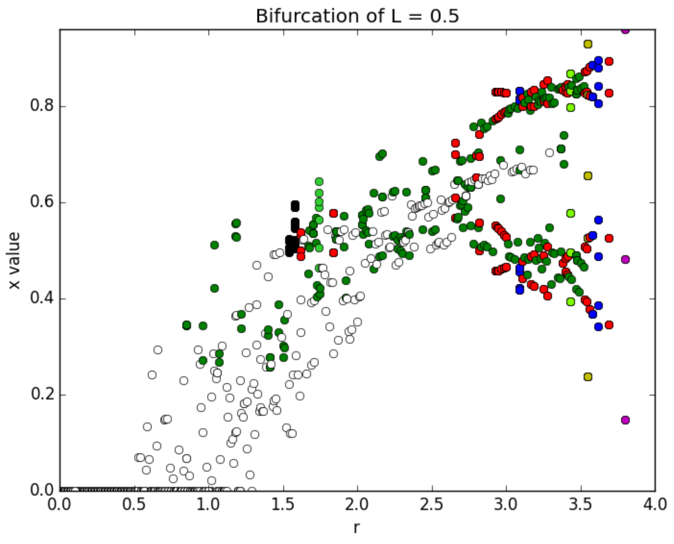

Figure 11: Bifurcation with $L = 0.5$

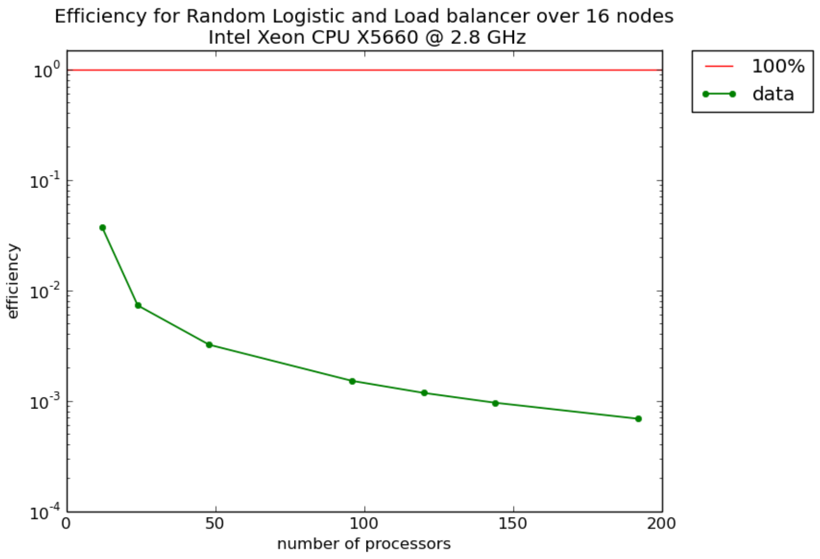

Figure 12: Efficiency

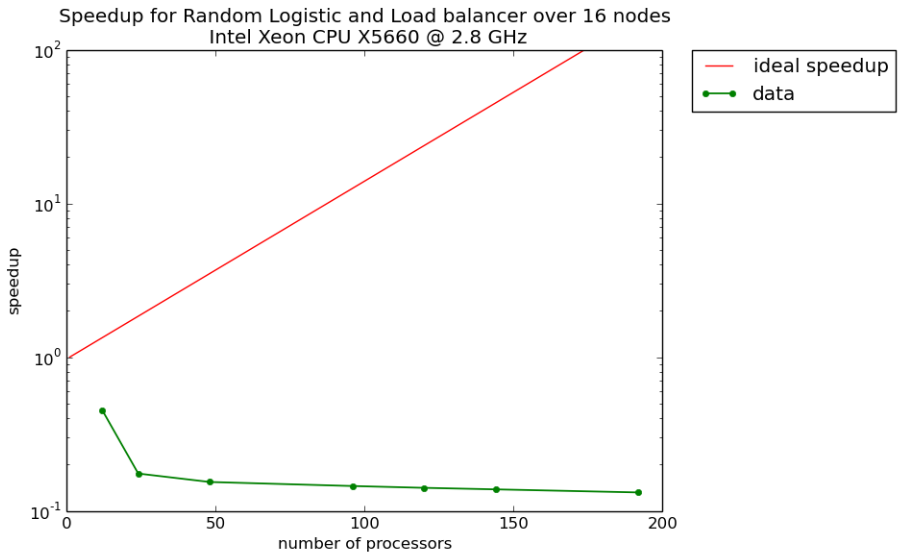

Figure 13: Efficiency

5 Conclusion

5.1 Future Work

References

[1] K.B. Athreya and Jack Dai. Random logistic maps. *Journal of Theoretical Probability*, 13(2):595–608, 2000.

[2] Mike Folk, Gerd Heber, Quincey Koziol, Elena Pourmal, and Dana Robinson. An overview of the HDF5 technology suite and its applications. In *Proceedings of the EDBT/ICDT 2011 Workshop on Array Databases*, AD '11, pages 36–47, New York, NY, USA, 2011. ACM.

[3] The HDF Group. HDF5 user's guide, 2014. http://www.hdfgroup.org/HDF5/doc/UG/index.html.

[4] Jeroen S.W. Lamb, Martin Rasmussen, and Christian S. Rodrigues. Topological bifurcations of minimal invariant sets for set-valued dynamical systems. *Proceedings of the American Mathematical Society*, 2013.

[5] S. Olivier and J. Prins. Scalable dynamic load balancing using UPC. In *Parallel Processing, 2008. ICPP '08. 37th International Conference on*, pages 123–131, Sept 2008.

[6] Marc H. Willibeek-LeMair and Anthony P. Reeves. Strategies for dynamic load balancing on highly parallel computers. *IEEE Transactions on Parallel and Distributed Computing*, 4(4), September 1993.