Toolkit Development for Parallelized Agent Based Wealth Distribution Models

O.J.Scholten, R.J.Spick, K.A.Hawick

The School of Engineering and Computer Science, University of Hull, Cottingham Road, Hull, Yorkshire HU6 7RX

Objectives

How can high performance computing capability be harnessed to run multi-parameter agent based wealth distribution models, and scientific workflow be streamlined?

- Create a framework for execution, analysis, and visualization of models.
- Explore parallelism as a means of running multiple models simultaneously.
- Evaluate existing libraries and frameworks for parallelism across compute nodes.
- Present early findings and visualizations.

Introduction

Agent based wealth distribution models have been explored for several decades, with early work by Dragulescu[1], Goswami [2] and others. Until recently simulations at realistic scales have not been possible without significant abstraction. It would therefor be useful to create a framework capable of executing several realistic sized models in parallel, automatically collating their results. These results could then be analyzed and visualized, applying relevant economics principles [6] and visualization techniques.

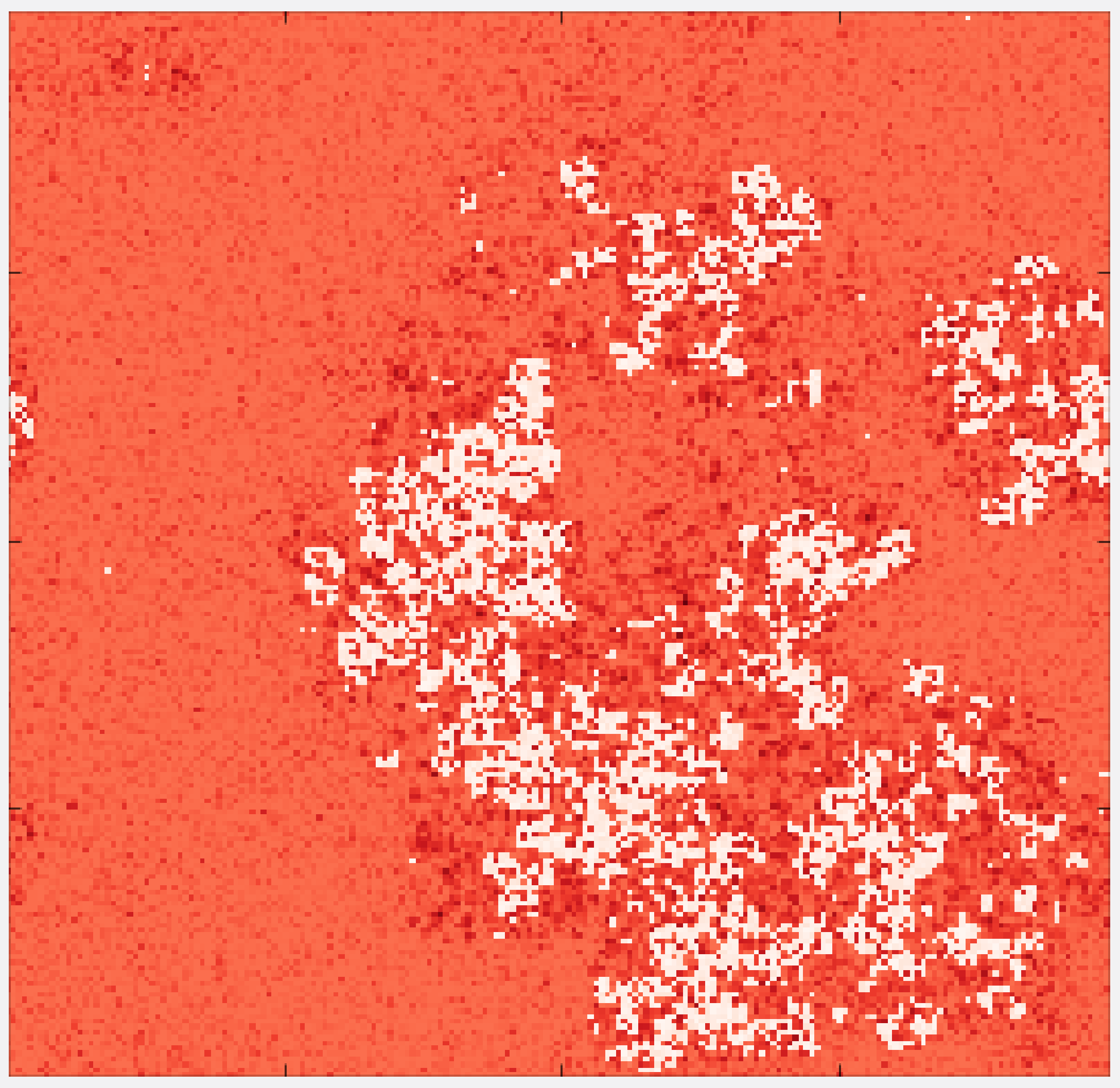
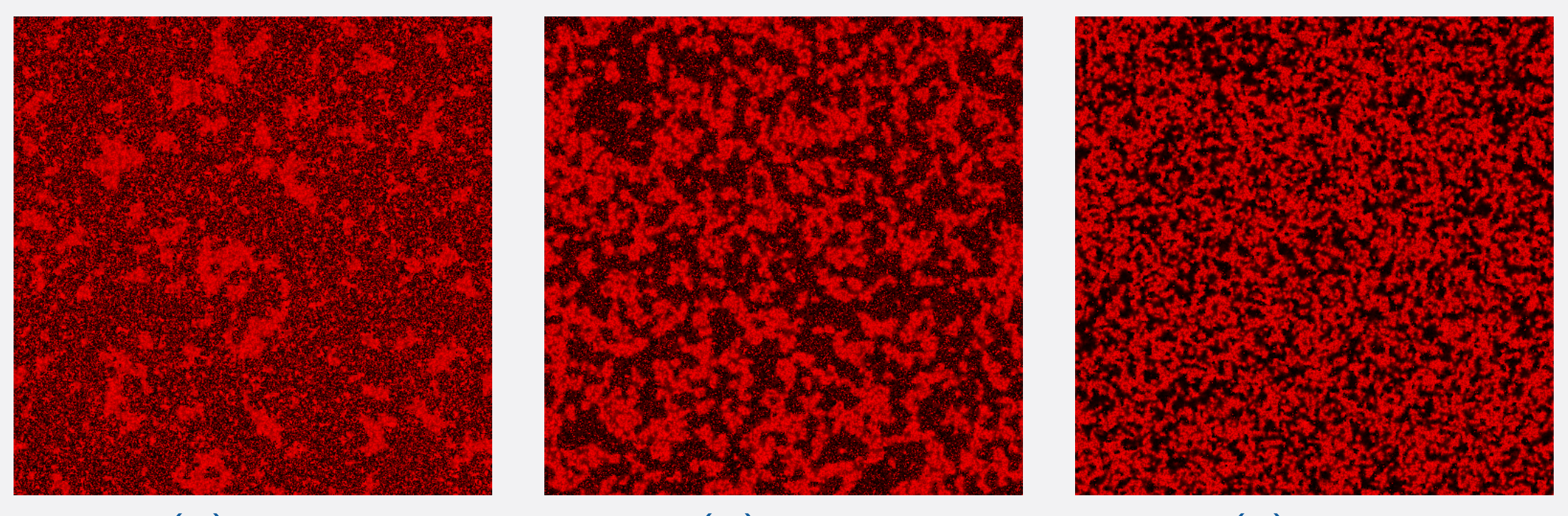


Figure 1: Early visualization of wealth based segregation using launcher style framework with Matplotlib, 200x200 world size.

Early Tools

As preparation for deployment to VIPER, multiple test versions of bespoke agent based modelling frameworks have been developed for use on local machines. These vary greatly in terms of their overall architectures and the languages/libraries they incorporate. Earliest work included a wrapper style program (C++) where each model would be called as a standalone program by another 'launcher' script. Whilst this structure worked in a local machine context it could not easily be scaled for deployment on VIPER given its architecture. The programs were then consolidated into a single unified framework, with parameters set in the code as opposed to through arguments.



(a) 0.29 (b) 0.36 (c) 0.43

Figure 2: Varying satisfaction values with no trading, 500x500 world size.

Figure 1 shows the first image created using Matplotlib(Python) of wealth based segregation using our current (C++) framework. Visualization by this method proved relatively slow, prompting the exploration of more efficient tools.

VIPER Architecture

Development of the framework depended on the architecture of the target system (VIPER), however it can also run on other systems with similar architectures. The VIPER cluster run at the University of Hull operates as 180 compute nodes - each with 2x14-core Broadwell E5-2680v4 processors along with several other components which do not affect the function of the framework. The cluster runs CentOS, a popular linux distribution, with Fortran and C++ compilers readily available making C++ a clear choice.

With the above specifications considered, the framework takes advantage of inter-node communication and parallelization across processors in the same node. One popular parallelization library (OpenMP) was used to great affect on single nodes however it could not utilise the distributed potential of the HPC. Another popular parallelization library (OpenMPI) which emphasizes communication between nodes in a system has been implemented to great effect. The framework has been designed following a master-slave approach, with a single master process managing collation of the data and file system interactions. Each other slave thread is responsible for some fraction of the overall computational effort and executes a variant of a model with some desired parameters, saving its output as an image.

Findings

By utilizing the parallel potential of VIPER we have been able to run identical models with multiple parameter variants. Outputs in figure 2 show how altering the satisfaction parameter in Schelling's model of segregation [3][4] impact the size and nature of the neighbourhoods after reaching equilibrium.

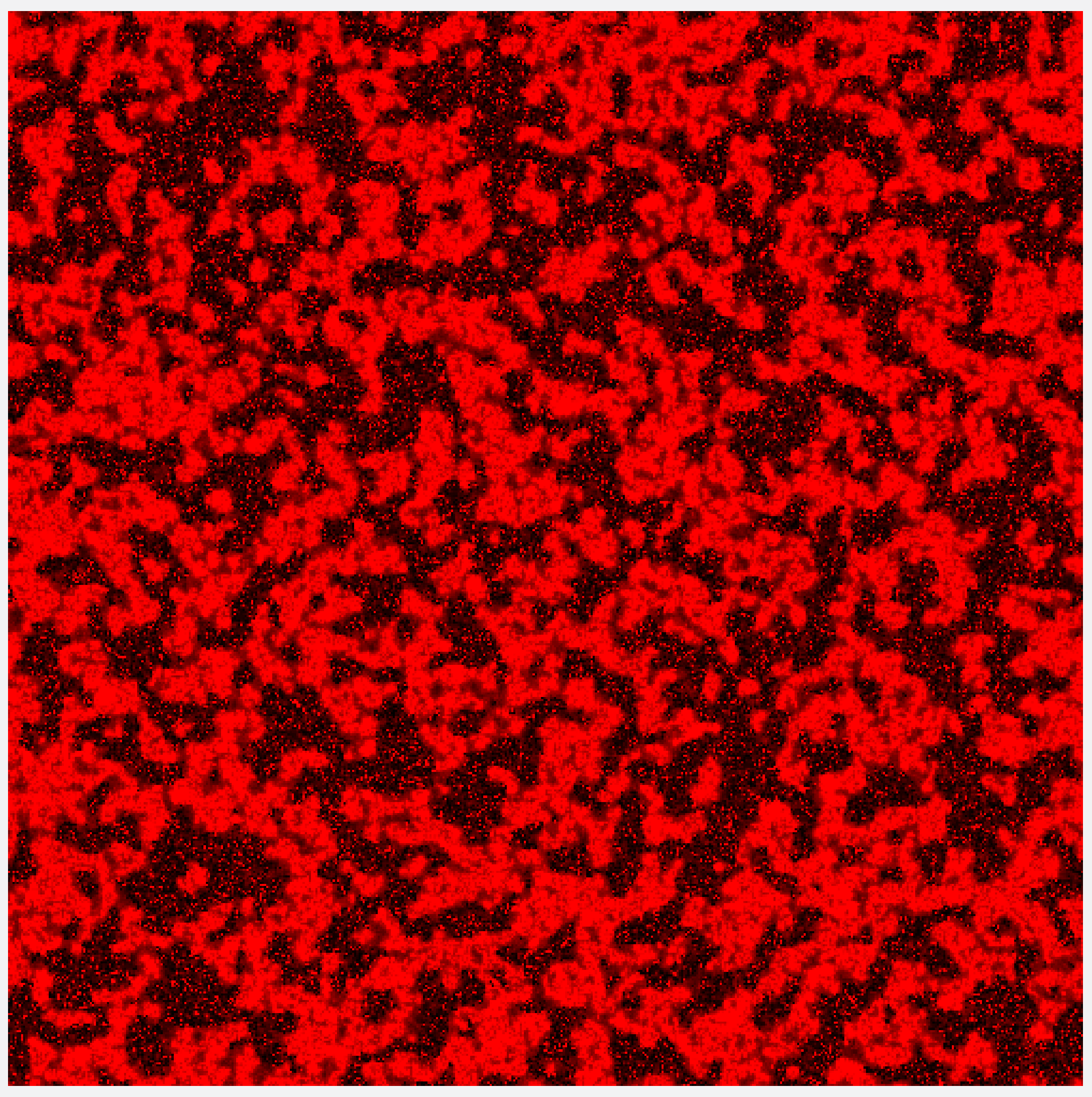


Figure 3: Satisfaction of 0.36 with 12 percent wealth traded per interaction, 500x500 world size.

Figure 3 shows the effect of adding spatial trading mechanics to the model where neighbourhoods were most visually segregated (2b). It is clear that this new mechanic increases the inequality of each neighbourhood (colour intensity), with larger areas of similar agents gathering in the same time duration. As with the satisfaction variable, the traded fraction of each agent's wealth was also run across many processors with different values. A value of 12 percent shows the most visually dissimilar neighbourhoods, further work in this area will be to perform formal statistical tests to quantify results.

Figure 4 shows a similar value for the trading fraction with a lower satisfaction variable. It is clear that the neighbourhoods are much larger than those with no trading (2a) yet inequalities appear less intense than those in figure 3. During our work we also discovered that increasing the trading fraction past a certain point also increased the overall noise in the final image. This phenomena can likely be countered by localizing the satisfaction mechanics and will be another key area of future investigations.

Visualization

Visualization began as primitive python scripts run locally, this transitioned into integration of the CImg library for simple image manipulation integrated into the execution of the framework on VIPER. This progression led to a more streamlined workflow, further reducing the time taken from inception to visualization of models.

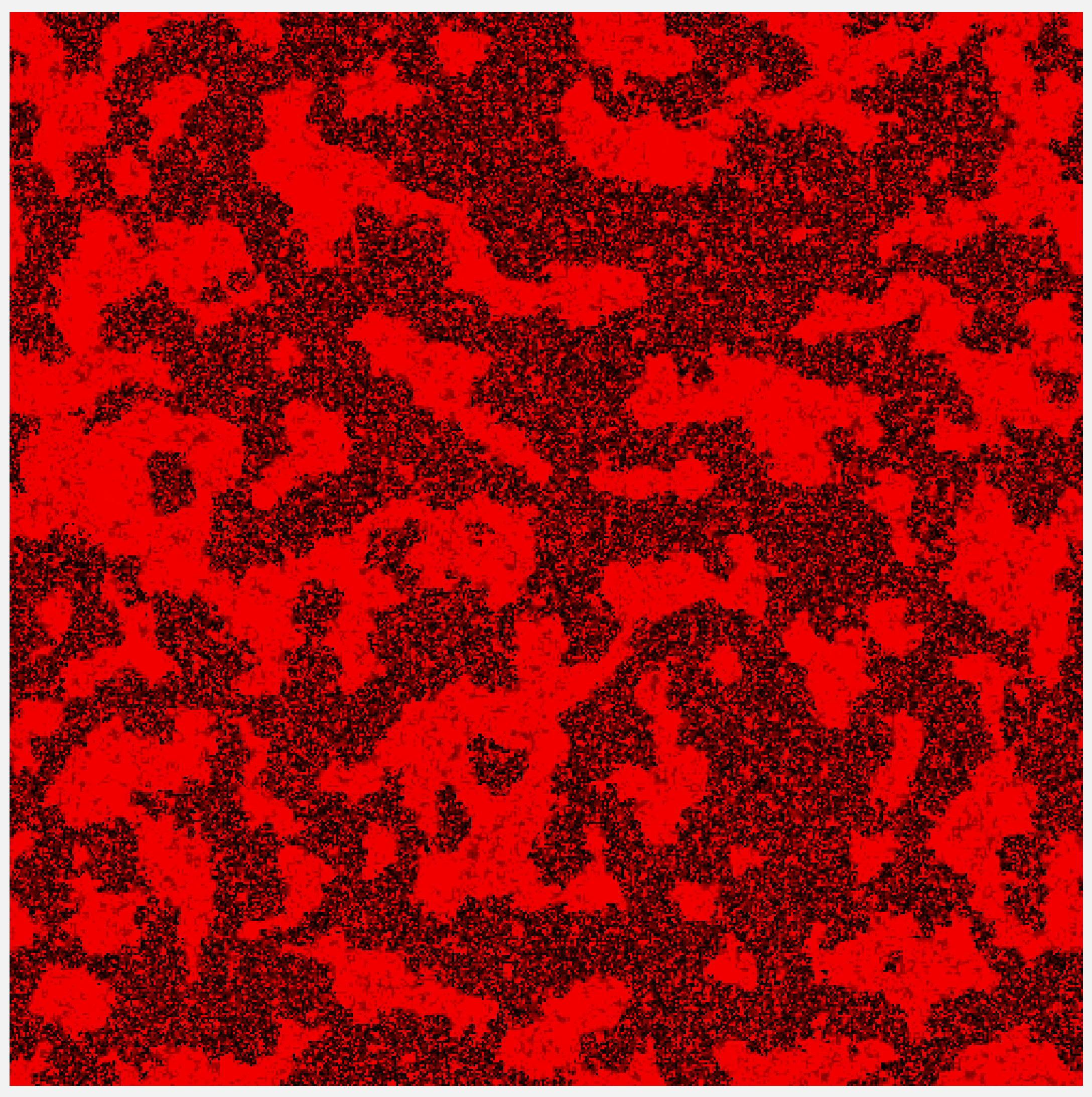


Figure 4: Satisfaction of 0.29 with 13 percent wealth traded per interaction, 500x500 world size.

Conclusion

Parallelisation frameworks are not new in the field of complex systems research, however a bespoke solution as presented in this poster acts as a springboard for future projects and represents an increase in the rate at which models can be developed and analyzed. This shortening of the development life cycle means future projects with a complex systems focus can be created in a matter of minutes as opposed to days, other researchers will also be able to develop their models in a much shorter time frame using our framework.

Future work in this area will include adding features and functionality whilst performing more rigorous testing with an emphasis on quality from a software engineering perspective. The software developed as part of this project also has the potential to be repurposed for use in other academic disciplines, inviting collaboration between departments and research groups.

References

[1] Dragulescu, A. and Yakovenko, V.M., 2000. Statistical mechanics of money. The European Physical Journal B-Condensed Matter and Complex Systems, 17(4), pp.723-729.

[2] Goswami, S. and Sen, P., 2014. Agent based models for wealth distribution with preference in interaction. Physica A: Statistical Mechanics and its Applications, 415, pp.514-524.

[3] Clark, W.A., 1991. Residential preferences and neighborhood racial segregation: A test of the Schelling segregation model. Demography, 28(1), pp.1-19.

[4] Schelling TC. Dynamic models of segregation. Journal of mathematical sociology. 1971 Jul 1;1(2):143-86.

[5] Raicu, I., Foster, I.T., Zhao, Y.: Many-task computing for grids and supercomputers. In: Proc Workshop on Many-Task Computing on Grids and Supercomputers. Austin, TX, USA (17 November 2008)

[6] Inoue, J.I., Ghosh, A., Chatterjee, A. and Chakrabarti, B.K., 2015. Measuring social inequality with quantitative methodology: analytical estimates and empirical data analysis by gini and k indices. Physica A: Statistical Mechanics and its Applications, 429, pp.184-204.

[7] Wang, K., Zhou, X., Chen, H., lang, M., Raicu, I.: Next generation job management systems for extreme-scale ensemble computing. In: Proc 23rd Int Symposium on High-performance, parallel and distributed computing (HPDC). pp. 111–114. Vancouver, BC, Canada (23-27 June 2014)

[8] Liu, J., Pacitti, E., Valduriez, P., Mattoso, M.: A survey of data-intensive scientific workflow management. J. Grid Computing 13, 457–493 (2015)