Accelerating the Optimized Direct SSA with Dynamic Compilation

Tuomas Laakkonen

Introduction

The Optimized Direct Method (ODM) [1] is a derivative of Gillespie's algorithm. It is more efficient as it eliminates unnecessary propensity calculations. To illustrate this we use the Brusselator, a common model of oscillating reactions, as an example.

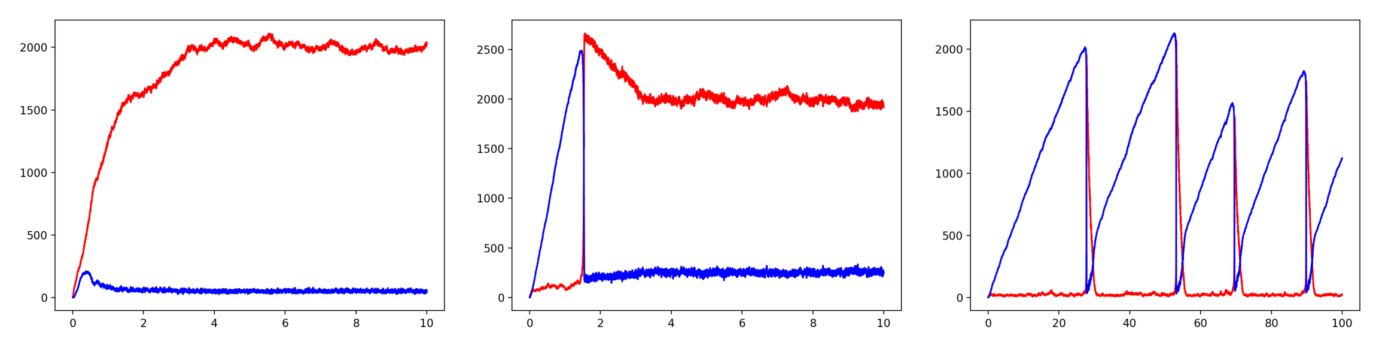

Figure 1: Three different modes of the Brusselator, showing the limit cycle behaviour.

A dependency graph shows what propensities need to be updated when a given reaction occurs. By traversing this graph in the simulation loop, the ODM only updates what is required.

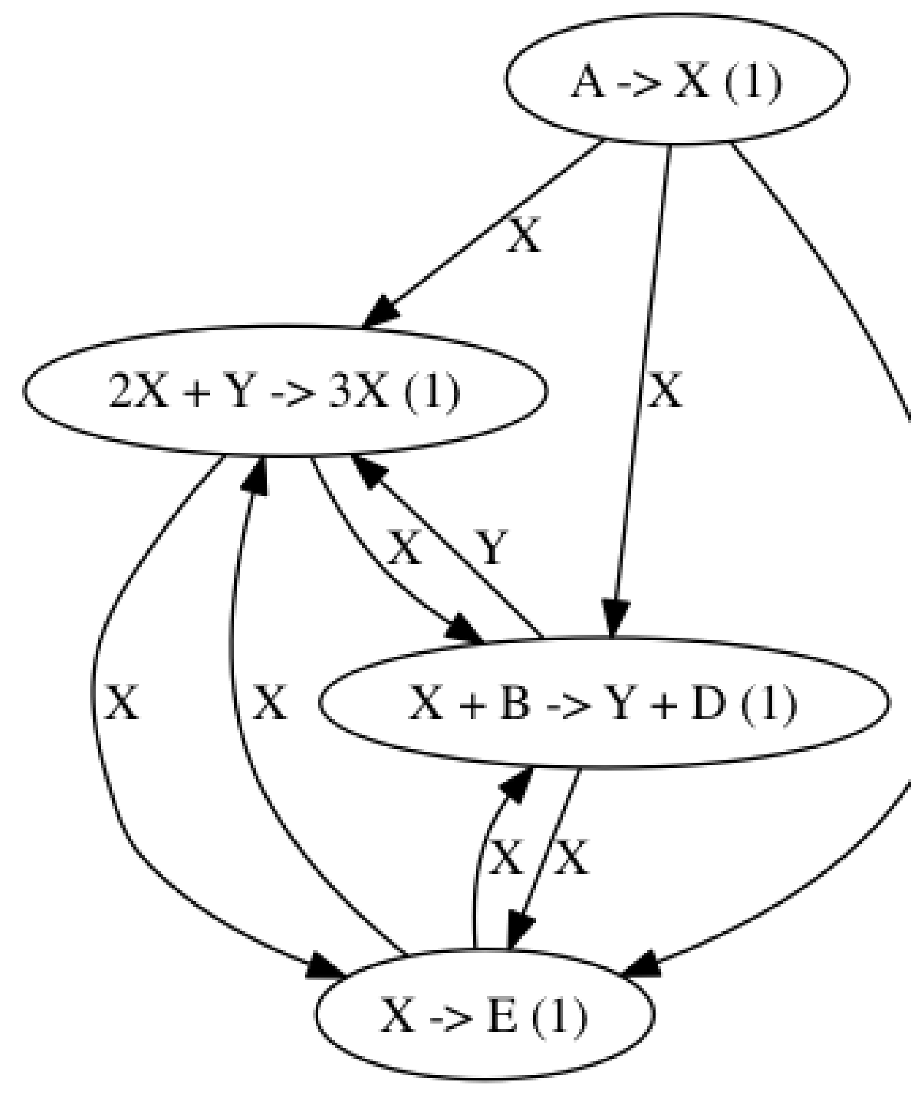

Figure 2: The dependency graph of the Brusselator. The labels of the edges indicate which species has caused the dependency.

However, in practice traversing this graph can be slow [1].

Method

To improve the performance of the ODM, we eliminated the graph traversal step from the simulation loop. We did this by constructing a function which updates the state and propensity given the reaction that occurs. The function hardcodes which values need to be updated for each reaction, and so the simulation can operate without using the dependency graph. The function is generated using the LLVM framework [2] - this is handled in stages: first, we construct a tree representation of the propensity function of each reaction, and then translate it into LLVM's intermediate representation.

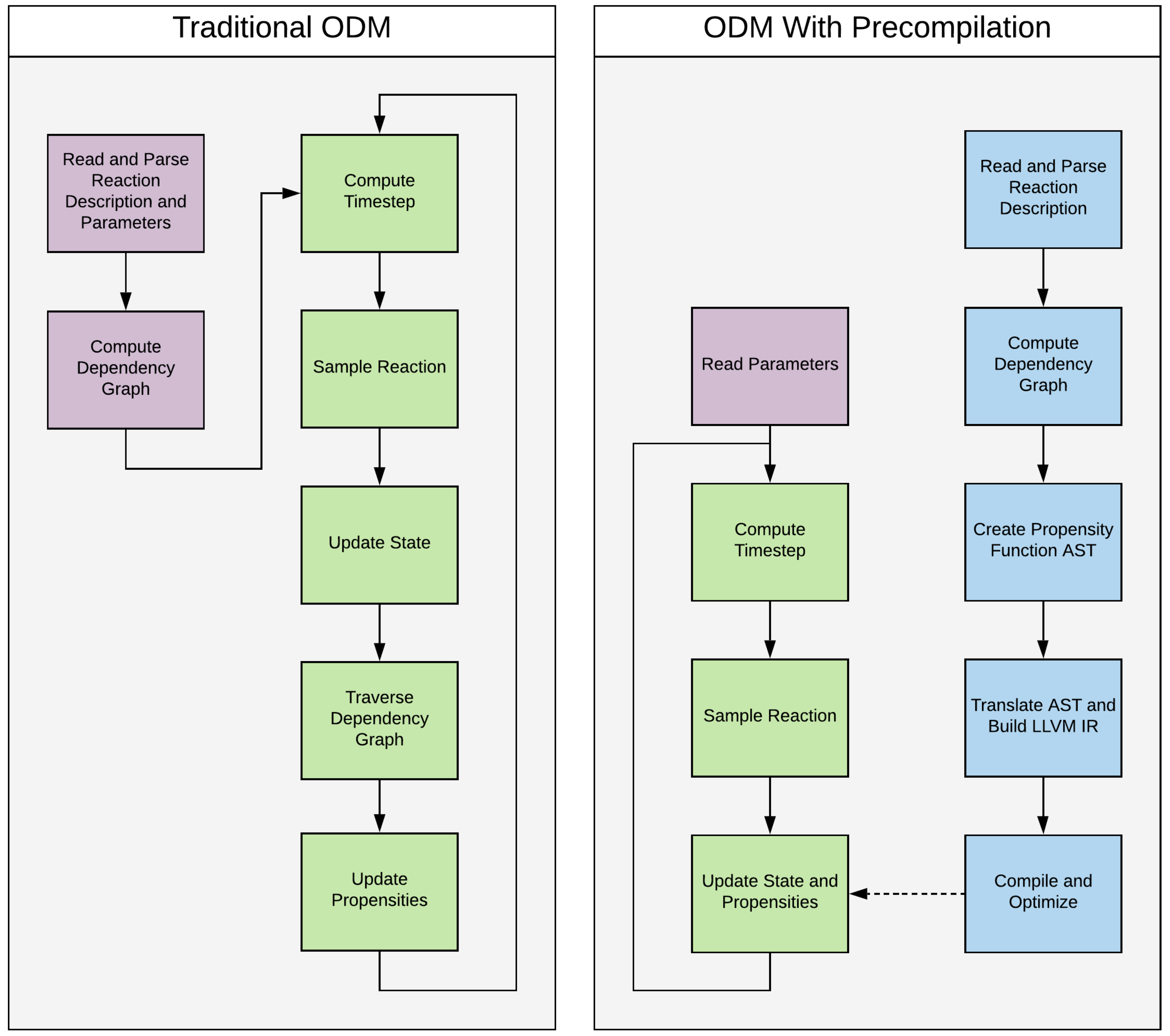

Figure 3: Flowchart describing the differences between the traditional and accelerated ODM. Purple represents program setup, green is the simulation loop and blue are precomputed steps.

The translation looks at the dependency graph to construct the updates for each dependent propensity, and then feeds this to LLVM's optimizer and compiler which produces a function that can be called in the simulation. By examining the provided initial conditions, we can also provide hints to the compiler about which reactions are most frequent to help optimization. We picked LLVM as the intermediate representation is platform independent [2], so our program should work on all platforms.

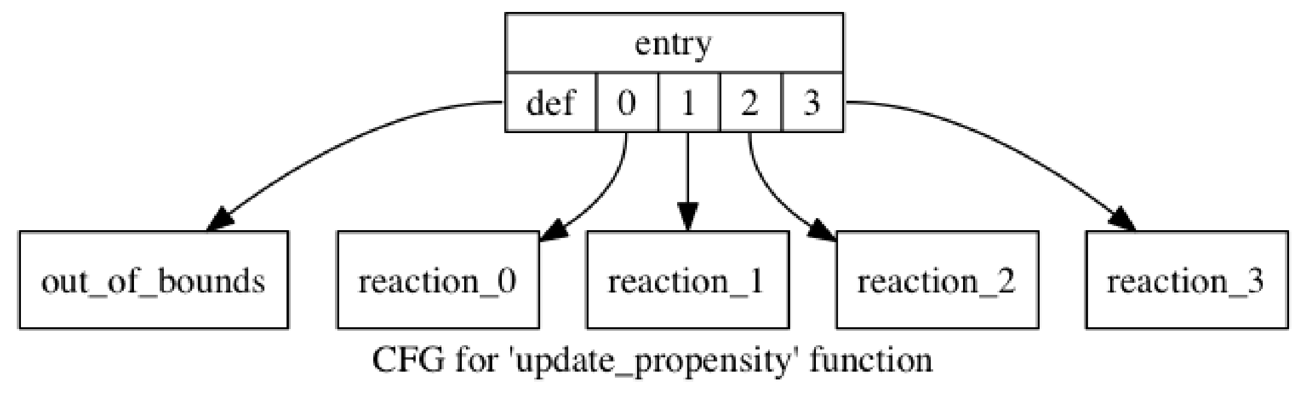

Figure 4: A control flow graph illustrating the structure of the generated propensity update function for the Brusselator. The flat structure is chosen to improve performance as it does not branch.

Results

We compared our implementation with StochKit2 [3], a highly optimized implementation of ODM, using five different models, varying in size from 3 to 610 reactions.

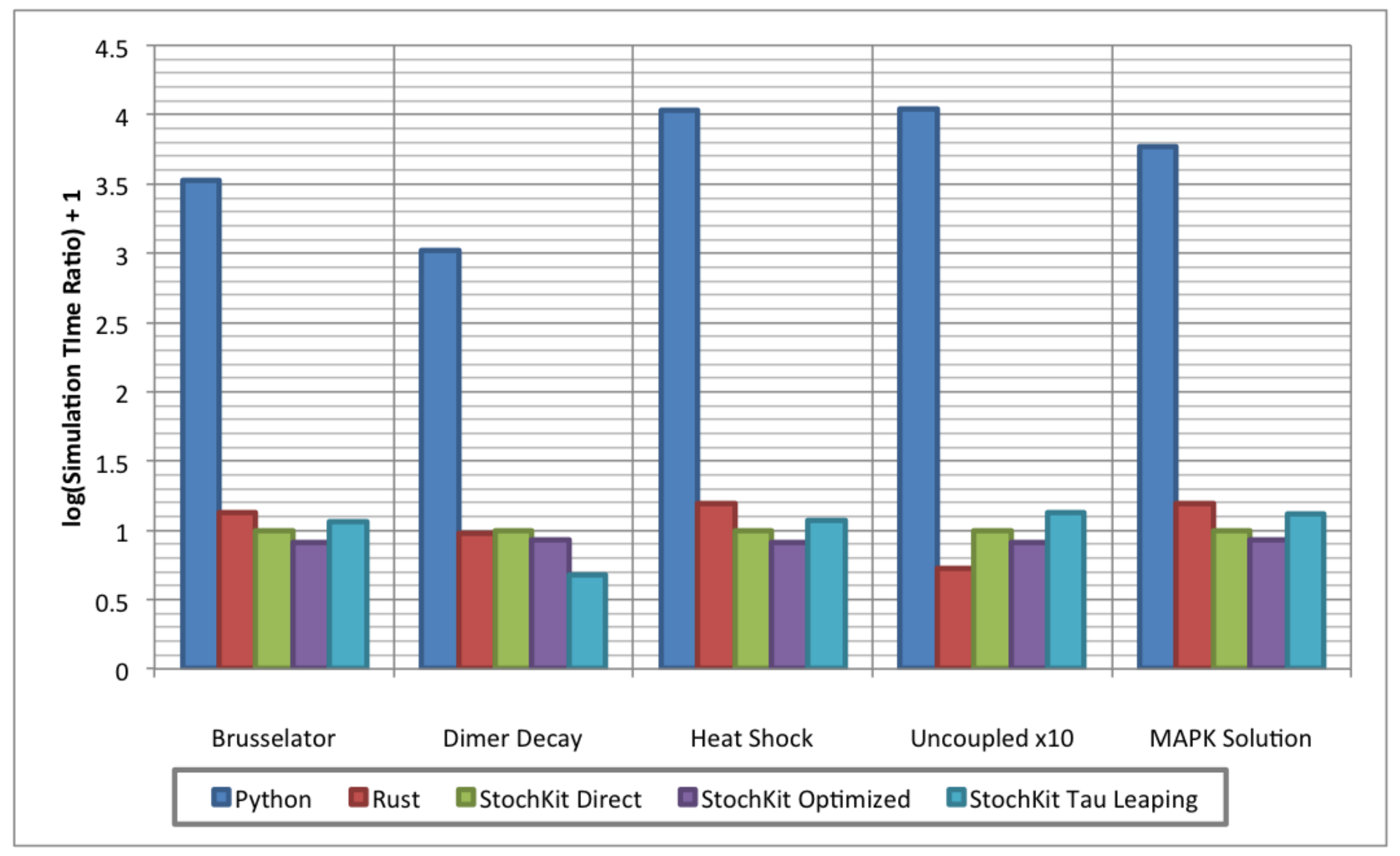

Figure 5: Performance comparison of our program (labelled as Rust) and various alternatives.

We observed similar performance in all models. We also measured the variation of efficiency with the size of the system.

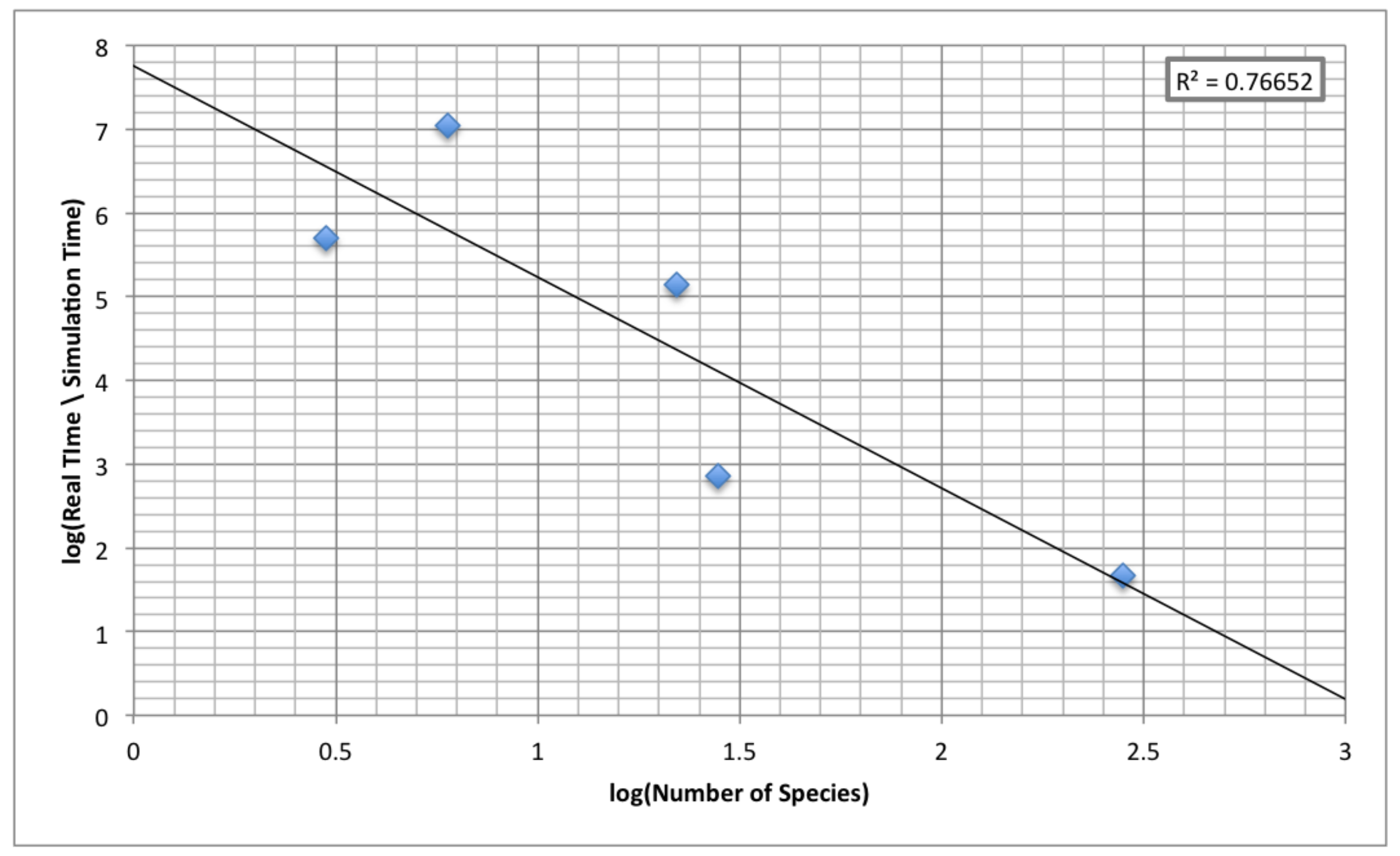

Figure 6: A graph showing how simulation efficiency varies with number of species.

Conclusion

As our implementation performed comparably to StochKit2 we can conclude that dynamic compilation is an effective technique as the rest of the program was not optimized at all. The increasing efficiency for large networks and parameterization also makes this useful for large networks and parameter sweeps. In future this could be extended to handle arbitrary propensity functions.

References

[1] Yang Cao, Hong Li, and Linda Petzold. Efficient formulation of the stochastic simulation algorithm for chemically reacting systems. *The Journal of Chemical Physics*, 121(9):4059–4067, 2004.

[2] Chris Lattner and Vikram Adve. Llvm: A compilation framework for lifelong program analysis & transformation. In *Proceedings of the International Symposium on Code Generation and Optimization: Feedback-directed and Runtime Optimization*, CGO '04, pages 75–, 2004.

[3] Kevin R. Sanft, Sheng Wu, Min Roh, Jin Fu, Rone Kwei Lim, and Linda R. Petzold. StochKit2: software for discrete stochastic simulation of biochemical systems with events. *Bioinformatics*, 27(17):2457–2458, 07 2011.