0.1 Further Work

There a few areas in the project that need to be looked at, ant behaviour could be more sophisticated and the problems regarding random placement of elements within the simulation world. Further work would resolve these issues and look for ways to increase the amount of parallelism generated by the current solution. Currently the simulation has a fixed partitioning system, future work would aim to make this dynamic to ensure better load balancing. Although graph partitioning algorithms generating equally sized graph segments with respect to vertex weight are NP-Complete [**?**], there exist heuristics which can produce desirable results. If the absense or presence of an ant at a node on the graph is considered a node's weight, it may be possible to use algorithms to partition the graph according to density. Algorithms such as Markov clusters or Voronoi graphs could allow for the generation of such partitions within a simulation world's nodes. This would result in the benefit of more ants being processed within the center of quadrants rather than on the edge of quadrants, as less parallelism is produced at the quadrant edges.

The project could be also expanded to make use of more than one processor by way of distributed computing. Extension the actor model could be implemented in a distributed fashion.

It would also be very interesting to develop similar algorithms in an imperative language like C++ and compare the complexity of the code as well as run time speeds. As this could further justify the choices made throughout this project or possibly call them into question. Another variation on the implementation which would be desirable to produce in further work would be incorporate distibutive programming via the actor model to increase scalability. This could be achieved by making use of the third party Haskell package Remote, sometimes called Cloud Haskell. Each core available on each machine in the network would then be assigned a quadrant and movement between quadrants could be handled entirely by message passing or each processor on the network could be assigned its own group of quadrants similar to a world graph and operate a divide and conquer strategy internally. Again, a comparison between these extensions would allow for the current implementation to be critiqued and analysed further.

During the final weeks of development the visualization of the simulation was revisited in order to produce a more user friendly and easier to view simulation. An effort was made to produce a GUI that would pass simulation settings and initialize a simulation without the need to recompile. Also a seprate window was run on a seperate thread with OpenGL capabilities in order to make use of colour when representing pheremone concentration. These efforts however are incomplete and it would also be nice to see this implemented fully in continued work.

0.1.1 Evaluative Conclusion

Aspirations of the projects artifact where very high throughout the initial phase of the project and the undertaking of some of these plans in a new language was possibly an over estimation of what could be achieved within the time frame the project allowed. However, as the realities of what was being undertaken increased through research and increased understanding throughout the implementation phase these expectations were adjusted. When taking this into consideration, the outcomes of the project were being developed in a previously unknown programming language.

During the earlier stages of the project it was necessary to revise the project plan as changes had occured to the design. The original plan allocated more time than needed for collision detection as in the final design collision detection involved very straight forward quieries into the datastructure. Also time spent on ant behaviour and and producing a visual element was dramatically reduced

Throughout the design process of this project many changes were made influenced not only by development but by research into the problem area of parallelism and simulations and through learning the purely functional programming language Haskell.

There were feasible alternative solutions to the approach the design and development of this project has taken. The choice of programming language arguably slowed the development process however, the use of Haskell allowed for the use of flexible progress with a language like scala.

hangs over 12 by 12

The final product allows for the generation of simulation worlds of varying size, these worlds are composed of many quadrants which divide the simulation problem into partitions. The various simulation elements, the ants, nest and food, all stored in sepeate datastructures, can be randomly placed at locations in the world when provided coordinates. On running the simulation, ants behaviour is to move in the direction of the surrounding cell with the highest pheromone level. Parallelsm is also introduced into the process of resolving ant movements between quadrants, by organizing a process order and for elements which can't be parallelized and grouping those which can the groups can. The product also allows for the manipulation of pheromones so pheromone levels can lower over time. Finally the results of each simulation step can be printed out to the screen

Printing the world Monad use of IO

There have been many lessons learned over the course of this dissertation project, a deeper understanding around the topics of parallelism and concurrency has been gained. Models

If the graph had been an instance of Functor one could have mapped over its nodes with a function that checked the node was on the desired edge before processing it. However this may have been more difficult to read.

Now that the project is has grown in size it is possible to look at programming from a differet point of view that functional programming has provided.

Functional programming languages are not the

The future of computing depends on parallelism (for efficiency), distribution (for scale), and verification (for quality). Only functional languages support all three naturally and conveniently [existential type]; other programming paradigms don.

A greater understanding of concurrency and parallelism,

somethings have beeen significantly enhanced (reasons).

Incomplete however the project has demonstrated an algorithm with potential to expand to Using libraries like debug trace was ineffective

Developed the capacity to program in functional paradigm This has been put to use in a project utilizing a purely functional paradigm