Design

Desgin techniques There is no standard way to model programs in Haskell and many different methods are used to carry an idea through from concept to code. Some programmers use modelling techniques similar to the Unified Modeling Language (UML) [Which is geared heavliy towards object orientated languages whith its use of class diagrams/Commonly used in object-orientated languages such as Java] While other developers use concept maps and data flow diagrams to provide a general picture of the design before they begin programming. However, because Haskell is a strongly and statically typed language it is a common practise to start constructing a program by working out the data types needed to complete the task. Then implement the functions for working with the newly created data types. Finally culminating in writing modules that bring together all the functions in a structured way. The approach to modelling however, differs from project to project.

Unlike imperative languages such as Java and C++ Haskell does not use classes to create objects and then send messages between objects. Rather it uses Declarative and Data-driven methods to process data. Haskell doesn't strictly speaking do any of these; because it is the programmer that does things using Haskell. However the structure of the language makes it more natural for imperative or Object Oriented programmers to take a different approach when programming in Haskell.

It is critically important to use the right technique or tool for the job at hand.

The classes used by Haskell are similar to those used in other object-oriented languages such as C++ and Java. However, there are some significant differences:

Haskell separates the definition of a type from the definition of the methods associated with that type. A class in C++ or Java usually defines both a data structure (the member variables) and the functions associated with the structure (the methods). In Haskell, these definitions are separated. The class methods defined by a Haskell class correspond to virtual functions in a C++ class. Each instance of a class provides its own definition for each method; class defaults correspond to default definitions for a virtual function in the base class.

Haskell does not support the C++ overloading style in which functions with different types share a common name. The type of a Haskell object cannot be implicitly coerced; there is no universal base class such as Object which values can be projected into or out of. C++ and Java attach identifying information (such as a VTable) to the runtime representation of an object. In Haskell, such information is attached logically instead of physically to values, through the type system. There is no access control (such as public or private class constituents) built into the Haskell class system. Instead, the module system must be used to hide or reveal components of a class. (Gentle introduction to Haskell Version 98)

Data types in Haskell and then quick checks [Method::Type-¿ Type -¿ Type - Method x y = undefined - quickCheck] – reference

UML is not limited to OOP. Even though it's founded in OOA/D it can be adapted a long way. The data structures are typically determined by the operations one wishes to run on them, so the usefulness of a diagram ultimately lies with t depends on the kind of UML model do you wish to make

Parametric polymorphism's also remarkably effective at helping strip out the irrelevant (use of type variables in)

Points to consider when desiging a parallel Algorithm Granularity Load Balancing Dependancy - Dependancy Graphs - http://msdn.microsoft.com/en-us/library/ee847415.aspx Scheduling

The partitioning phase of a design should produce one or more possible decompositions of a problem. Before proceeding to evaluate communication requirements, Ian Foster [Designing and Building Parallel Programs 1995 by Ian Foster] reccomends the use of the following checklist to ensure that the design has no obvious flaws. Generally, all these questions should be answered in the affirmative.

Does your partition define at least an order of magnitude more tasks than there are processors in your target computer? If not, you have little flexibility in subsequent design stages. Does your partition avoid redundant computation and storage requirements? If not, the resulting algorithm may not be scalable to deal with large problems. Are tasks of comparable size? If not, it may be hard to allocate each processor equal amounts of work. Does the number of tasks scale with problem size? Ideally, an increase in problem size should increase the number of tasks rather than the size of individual tasks. If this is not the case, your parallel algorithm may not be able to solve larger problems when more processors are available. Have you identified several alternative partitions? You can maximize flexibility in subsequent design stages by considering alternatives now. Remember to investigate both domain and functional decompositions. [Ian Foster]

High level parallel design (pmap) discussion. Map Reduce Actors Embarassingly Parallel Replicable Repository Pipeline A series of ordered but independent computation stages need to be applied on data, where each output of a computation becomes input of subsequent computation

'par 'seq' annotations and other forms of denoting parallelism.

-Predicted code flow of the application On the function/method level. More detailed than that.

Data flow

Representing Source code

(Talk about the non parallel

Then the parallel version

Split into Epochs of design

Mind map)

Alternate designs Everthing is based on message passing ants are grouped into oiiks and processed in batches by threads/ processors. Each processor gets representation of the world and sends requests to all other processors to check if it can move its ant in a particular direction, this design is possible more suitable for Erlang

Representation of the Simulation world - Split into Epochs of design

The design of the simulation went through many iterations.

The list based Worlds design free movement The initial concept for the representation of the simulation world was list based. Several lists would be produced for everything in the world, for an example a list of Ants currently in the simulation, a list of Food currently in the simulation and a list of the surfaces in the simulation. Each element in the lists would hold a location value and ———————— The initial concept for the data representation of the simulation world was to produce a list for every type of item in the world. For instance there would be an Ant List, a Food List a Surface Type List. Each item in the list would also be given a location to represent where it was in the simulation world. Each list could then be processed in parallel. Copies of each list would be made for lists which needed other lists in order to compute their changes. Once each list had updated the old lists would be discarded and copies of the new lists would once again be made for the next simulation step. The locations held by each item would have an x and y value, and information radius or a second x and y in order to calculate a bounding box.

List based Worlds location tags This approach was similar to the previous idea, using lists to process objects of similar types. However placing a restriction that x and y coordinates must be integers and each location is the size of one ant. This concept influenced all following design decisions, making it trivial to reason about collision detection.

Graph based World Hold all

Graph based Worlds non parallel

Graph based Worlds parallel

Types The types that will be involved in modeling the simulation of an Ant colony are as follows.

```
Ant           :: Ant
Pheremone     :: Double
Food          :: Double
Nest          :: Bool
AntQuadrant   :: Graph -> Holds- Possible Ant
PherQuadrant  :: Graph -> Holds- Pheremone Level
FoodQuadrant  :: Graph -> Holds- Possible Food
NestQuadrant  :: Graph -> Holds- Possible NestArea
AntWorld      :: Graph -> Holds- Many Ant Quadrants
PherWorld     :: Graph -> Holds- Many PherQuadrants
FoodWorld     :: Graph -> Holds- Many FoodQuadrants
NestWorld     :: Graph -> Holds- Many NestQuadrants
```

Key Functions The functions responsible for the flow of the program are

stitchUpAntQuads - Sequentially processes pairs of edge-nodes on a Graph (to possibly move Ants between AntQuadrants). stitchUpPherQuads - Sequentially processes pairs of edge-nodes on a Graph (to manage the spread of Phermone between PherQuadrants).

processAntQuadrant - Moves each Ant within a quadrant. processPherQuadrant - Evaporates the Pheremone

Main Given Starting parameters create a World. -Parameters

```
Size of Quadrant :: Int
Size of World :: Int
Amount of Ants :: Int
Amount of Food :: Int
```

-Initialize random nest area proportional to the amount of Ants. -Randomly position Ants around the Nest Area -Randomly position Food (not In Food Area) -Initialize all PherQuadrants as 0 except for Food Quadrants which maintain 10.0

Simulation Loop

In parallel -stitchAntQuads -stitchPherQuads

In parallel - processAntQuads - processPherQuads

Display

ReQuading Graph Density calculating densest points NP-hard?

When graphs are split at densest points More serial calculations.

QUICK CHECK NOTES