Review of GdH August 1999 - February 2000

Robert F. Pointon

February 21, 2000

Abstract

In the project for the Glasgow Distributed Haskell (GdH), we look at progress made so far, and plans for the future. It should be noted that this project is currently due to end February 2001.

1 Introduction

We look at an overview of the GdH project in terms of design, implementation, applications & evaluations, extensions, and potential papers.

2 Design

The design phase of GdH is complete, and we have decided on the features want to introduce to the langauge, the approach to be taken and the primitives we need to add.

2.1 Features

Remote Thread Creation the ability to create explicit threads on other processing elements (PEs).

Distributed I/O through the ability to place threads and thus perform remote I/O, allowing the unique resources and attributes of each PE to be utilised.

Fault Tolerance within pure functional code recovery is attempted, whilst within non-pure I/O code exceptions will be raised thus allowing programmable fault recovery.

Dynamic PE Network. Fault tolerance will mean that PEs can be rejected and reinitilised, this will be taken further allowing the ability for PEs to added or deleted from the running program.

Backward Compatability with existing Haskell programs will be maintained including the centralised concepts of Glasgow Parallel Haskell (GpH) having a distinguished *main* thread and therefore a main thread PE, likewise, Concurrent Haskell specifing program termination as occuring with the finishing of the main thread.

2.2 Approach

Firstly we need mechanisms to identify the current PE and what other PEs are available.

Threads and MVars have been identified as immobile resources, stuck on a particular PE. All other immobile resources (ie file, foreign objects pointers) should be protected by MVars. Thus we require a mechanism to identify the owning PE of these immobile things.

Remote PEs will be accessed via a synchronous remote evaluation mechanism. This will allows the remote creation of threads and performing of I/O.

Fault Tolerance is noted as an important requirement, and the new exception mechanisms of Concurrent Haskell will play a major role. *TO DO: Issues of automatic recovery will be delayed until later in the project.*

2.3 Primitives

The actual low-level primitives needed follow naturally from the 'Approach'.

PE identification :

```
allPEId :: IO [PEId]
myPEId :: IO PEId
```

Immobile resource identification :

```
threadOwner :: ThreadId -> IO PEId
mvarOwner :: MVar a -> IO PEId
```

Remote evaluation mechanism :

```
reval :: IO a -> PEId -> IO a
```

3 Implementation

Little work has occured on the implementation given that GUM and GpH are still somewhat unstable. What has been implemented is the skeleton of the communication code for the `reval`, all of `allPEId`, `myPEId` and some library code for building higher-level primitives such as `rforkIO`.

TO DO: `reval` *requires the implemention of a new closure type, and code for handling the blocking of threads, etc. Also the creation of the temporary remote thread.*

`threadOwner` *requires some rewritting of the standard libraries as the ThreadID should be unique across all PEs, and the simplest solution is to make it a tuple of the existing ThreadId and the owning PEId.*

`mvarOwner` *requires code to access the GALA tables to check if it is global and if so then which PE owns it.*

Remaining work is the rewritting of various I/O libraries to check if MVar/Threads are local or remote and to apply `reval` *appropriately.*

4 Applications and Evaluation

Several applications have been proposed, the actually evaluation would involve a comparison with the same application written in different langauges, possibly GpH, Eden and JAVA.

Web Server Hopefully utilising distributed I/O - one machine with external network connection, others with dedicated disks. Fault tolerance is a must.

Kiteck Factory Simulation A client server type application, where the clients chat via a message board and receive the simulation information from the server.

Speculative Parallel Evaluation Having explict remote threads, means that unlike GpH we have more control over the parallelism, work and data placement. This could lead on to an interesting comparison between GpH and GdH.

5 Extensions

Anti-PEId The use of PEId is considered rather low level, and some sort of registry system would be nice. PEs would have their capabilities recorded so that work can be distributed via calls refering to the PE with the lowest load, or IP address xxx.yyy.zzz, or cluster xxyz... This should be implementable as library functions using the current primitives.

6 Papers

The sort of papers we look to write in the near future:

The design of GdH Which we should be able to write now.

The implementation and Evaluation of GdH Once we have completed the basic implementation and written a few mini-applications then hopefully we will have a better feel for the language's features.

Exceptions in GdH ? Once the asynchronous exceptions and basic fault detection has been implemented then hopefully we can say some intelligent things about GdH maybe we can compare it to other languages with exceptions and/or fault recovery.

Fault Tolerance in distributed shared memory functional languages We might come up with some neat ideas for fault recovery for the pure functional side.