Prof. Dr. R. Loogen, M. Dieterle,
T. Horstmeyer, B. Pickenbrock

Philipps Universität Marburg

CEFP
4TH CENTRAL-EUROPEAN FUNCTIONAL PROGRAMMING SCHOOL - JUNE 14-24
2011
BUDAPEST
EÖTVÖS LORÁND UNIVERSITY

Central European Functional Programming School, June 2011: Eden Lab Session, Friday, June 16, 9–11am

Example programs are provided via the Eden web page

http://www.informatik.uni-marburg.de/~eden

Navigate to CEFP!

Working with the Eden system

General: For extended descriptions on Eden, the Eden compiler, the skeleton library, the trace viewer and additional material visit the Eden web pages (see above).

Compile and run Eden programs: You have to use the command '`edenenv`' to set the necessary environment variables on your lab machines. Use `ghc --make` with the option `-parmpi` (or `-parpvm`). Eden programs have to import the module `Control.Parallel.Eden`.
Eden programs work on basis of the parallel middleware MPI (or PVM, respectively). Eden programs use the RTS-options `MPI@<hostfile>` to define the hosts of the parallel machine (default is `mpihosts`), `-N<n>` to set the number of virtual machines (default is 1 VM per host), `-qQ<X>M` to set the size of the message buffer and `-H<X>M` for the initial heap-size. If a program crashes, you probably have to increase the message buffer.

Runtime-tracing of Eden programs: Compile a program with the option `-eventlog` and start it with the runtime option `-ls` to create a runtime profile which can be visualized with the Eden trace viewer EdenTV. A sample MPI hostfile is provided on the web page.

Exercises

1 Parallel `map-reduce` in Eden

(a) Develop a parallel version of the following higher-order Haskell function

```
mapReduce :: (a → b) → (b → b → b) → b → [a] → b
mapReduce mapF redF n list = foldl redF n (map mapF list)
```

Assume that the parameter function `redF` is associative and that `n` is the neutral element of `redF`.

(b) Use your parallel map-reduce skeleton to parallelise the program `sumEuler.hs`.

(c) Run your parallel program on i machines, where $i \in \{1, 2, 4, ...\}$ (runtime option `-N`i).
Use the Eden trace viewer to analyse the parallel program behaviour.

see next page

2 **Load Balancing**
Write parallel versions of the Julia-Set program using

(a) the `parMap` skeleton.

(b) the `farm` skeleton and the un-/shuffle function for task distribution.

(c) the `workpoolSorted` skeleton.

Use EdenTV to analyse the runtime behavior of the different versions.

3 **Matrix multiplication according to Gentleman**
Write a parallel version of the matrix multiplication program `gentleman.hs` given on the Eden pages. The program uses a torus skeleton. Write your own parallel torus skeleton (with Remote Data or simply using communication through the caller process) or use the skeleton of the latest skeleton library version (not installed, but available on the Eden pages).

In a simple version, all matrix elements are calculated by a different process. In a more sophisticated version, processes manage sub-matrices. The algorithm works similar when each process multiplies sub-matrices and sends the sub-matrices to their neighbors.

You can use

```
spawnPss :: (Trans a, Trans b) ⇒ [[Process a b]] → [[a]] → [[b]]
```

to instantiate a matrix of processes and

```
fetchRDss :: Trans a ⇒ [[RD a]] → [[a]]
```

to fetch a matrix of Remote Data.