Referee's Report on "Comparing Id and Haskell in a Monte Carlo Photon Transport Code"

Jeffrey Hammes, Olaf Lubeck, Wim Boehm

Summary

The functionality of a large Fortran code—*MCNP*—developed at Los Alamos National Laboratory, is outlined. This code is used to simulate photon transport, with applications in nuclear engineering and medical physics. A system is modelled by a series of cells having variable shape and material characteristics. A simulated radiation source is introduced. Individual photons are tracked through the cells, as they undergo collision and splitting events. Statistical information on particle flux, etc is accumulated.

The *MCP-Functional* code developed by the authors implements a representative subset of the functionality of the full Fortran code.

The top-level Haskell is displayed. It contains operations for creating source photons; tracking them to produce an event list for each of these photons; tallying, to yield a user-defined summary of events; accumulation of averages and standard deviations; and, finally, printing of results. The producer-consumer relationship between tracking and tallying is emphasised.

The Id code differs from the Haskell in tending to avoid recursion, higher-order functions, partial applications and function composition in favour of a more imperative style with abundant use of loops, and using arrays in favour of lists.

Driven by performance considerations, the implementation order was unusual...

```
Fortran -> ``Impure'' Id -> ``Pure'' Id -> Haskell.
```

To enable high-statistics runs for production *and* debugging purposes, the Id code has been written in an (apparently) almost completely procedural style, with explicit memory management calls. The debugged code was "reverse engineered" to produce a correct (but presumably memory inefficient?) "pure" Id code.

The *MCP-Haskell* code has been derived from the *MCP-Id* code, and apparently only uses laziness "in the small" (it hasn't influenced the overall program structure). The Haskell version has space leaks. It is claimed that this leakage is unavoidable due to sharing of data structures by multiple consumers.

The authors conclude that: "The Monte Carlo photon transport problem coded cleanly in a pure functional way, and MCP-Functional produces results which are statistically comparable with those of MCNP". Resource management issues were the main obstacle to the development of the functional version of the code.

Comments and suggestions

A major worry is the level of procedural detail the authors have had to resort to in order to debug and validate the "functional" Id code, as described in section 6. It was unclear to to what extent the problems are ones of memory management in a functional language, and to what extent they are problems of parallelism peculiar to Id. There is extensive discussion of barriers and synchronisation problems. I assume these problems relate to parallel execution of Id. Some brief overview of the relevant semantics of Id might help to clarify the matter (perhaps in a separate section). In any case the presentation would benefit if the problems intrinsic to functional languages were more clearly factored from those connected with parallelism.

I was worried by the implementation order of the various versions: Fortran, then "impure" Id, then "pure" Id, then Haskell. As the authors note, this is exactly the reverse of the "natural" order, with prototypes developed in the higher level languages, which are then translated to more efficient, more procedural implementations. Has the decision to take an existing large Fortran program and try to reproduce its results skewed the program development process here? Is it an oversimplification to suggest that some of the problems with debugging might have been avoided if the program development process had followed the more conventional route? I understand the authors claim that validation of the final code required high statistics data, but one still suspects that more debugging could be achieved on a component by component basis in high-level protoype versions...

It is stated in section 6.2 that "the evaluation order of MCP-Haskell requires a heap size which grows linearly with the number of source photons." By investing an amount of effort in the Haskell implementation comparable to that evidently expended on the Id version, would it not be possible to plug space leaks? (Of course Haskell doesn't provide direct control over deallocation, but my understanding was that code can be normally rearranged to eliminate leaks.) Perhaps displaying a simplified example code with a similar sharing pattern would make it clearer what the fundamental problem is here?

Other points

Section 3 It would be nice to see the top-level organisation of the tracking

phase (more interesting than tallying?).

In figure 8, the agreement between Fortran and Id results seems too close, given the size of the error bars. If, as suggested, different random number generators were used in the two programs, the runs ought to be statistically independent, and the mean values would be expected differ by about their standard error. The three displayed values are *much* closer than that (if the error bars are indeed standard errors). This needs some explanation. Perhaps MCNP gives over-conservative error estimates (this seems like bad practise—it amounts to throwing away information).

Section 4, first para "local variables are all no more than..." should presumably be "local variable *names* are all no more than...".

Sections 4, 5, 6 More explanation of Id semantics would be useful, where it differs radically from a normal functional language. For example, what is the semantics of an Id barrier—how global is it? What are the M-structure arrays and open lists mentioned in section 6.1?

Section 8 Is it really appropriate to record the directory structure of an anonymous FTP site here? That depends on JFP policy, but I think a simple pointer to the site would be enough.