Declarative systems architecture: a quantitative approach (AQUA) Final Report

Simon Peyton Jones, University of Glasgow

January 24, 1997

1 Summary

This report summarises the achievements of AQUA project, which ran from February 1993 to October 1996.

The goal of the project was *"to begin the quantitative feedback process in the field of declarative systems architecture"*. Specifically, the project concerned *"compilers and runtime resource management for non-strict functional languages such as Haskell, on both sequential and parallel platforms"*[1].

A secondary, but important, goal articulated in the proposal was to contribute to the maturing of functional programming languages, helping to move them from the laboratory and into the hands of users.

The project has been a very exciting one:

- We have met essentially all the goals described in the original proposal (Sections 2-4).
- In the area of parallel functional programming we were able to join forces with the Parade project and a research student to go well beyond the originally-proposed work (Section 5).
- In several other areas, not mentioned in the original proposal at all, we made important progress (Section 6).

However good a research programme, it is largely wasted if its results are not well presented in widely-read conferences and journals. We are pleased that we were able to maintain a high rate of publication (about ten papers each year), and to publish in premier conferences (Principles of Programming Languages, Programming Languages Design and Implementation, Functional Programming and Computer Architecture, Parallel Architectures and Languages Europe) and journals (Transactions on Programming Languages and Systems, Science of Computer Programming, Lisp and Symbolic Computation), as well as numerous workshops.

Throughout, project publications are cited numerically, thus [12], and are listed at the end of each section. Publications by others, or published before the start of AQUA, are cited thus (Jarvis [1966]), and appear at the end as usual.

[1] Quotes are from the original proposal

2 Quantitative measurements

"We will produce and publish quantitative information about the dynamic behaviour of substantial and realistic functional programs, both sequential and parallel. We will use this information to improve the optimisations and transformations used by the Glasgow Haskell Compiler, and to improve its runtime system, on both GRIP and sequential workstations".

The Glasgow Haskell Compiler uses program transformation as its main optimisation paradigm, so we invested a lot of effort in characterising a useful set of transformations and measuring their effectiveness. Using our `nofib` benchmark suite (Section 4) we were able to make detailed measurements on a large number (between 20 and 70, depending on the experiment) of actual applications or application kernels. We published several substantial papers on this topic [1–7]. In addition, Santos's PhD thesis is largely devoted to detailed measurements of transformation strategies [4], and Gill's PhD thesis quantifies the effect of the short-cut deforestation transformation [5]. These papers, and the theses, are backed by mountains of measurements. For example, for just one paper: we built 14 special versions of the Prelude libraries (145MB of disc space); then we built each of 70 benchmark program in 16 different ways (1,415MB in disk space); then we ran each of the resulting 1120 programs several times, averaging its run time and other statistics; then we summarised the resulting mound of numbers into a manageable form.

We also made measurements of the memory-system behaviour of lazy languages. In particular, we demonstrated that, contrary to popular belief, generational garbage collection is highly effective for lazy evaluation [2]. GHC has used a generational garbage collector ever since.

Measurements of parallel systems are covered in Section 5.

2.1 Publications

[1] SL Peyton Jones, WD Partain, *"Measuring the effectiveness of a simple strictness analyser"*, in "Functional Programming, Glasgow 1993", ed K Hammond and JT O'Donnell, Workshops in Computing, Springer Verlag, 1993, pp201-220.

[2] PM Sansom and SL Peyton Jones *"Generational garbage collection for Haskell"*, Proc Functional Programming Languages and Computer Architecture (FPCA93),

Copenhagen, ACM Press, June 1993, pp106-116.

[3] AJ Gill and SL Peyton Jones *"Cheap deforestation in practice: an optimiser for Haskell"*, 13th World Computer Congress 94, Vol 1, eds B Pehrson and I Simon, Elsevier Science (North-Holland), 1994, pp581-586.

[4] A Santos, *"Compilation by transformation in non-strict functional languages"*, PhD thesis, Department of Computing Science, Glasgow University, Sept 1995.

[5] AJ Gill, *"Cheap deforestation for non-strict functional languages"*, PhD thesis, Department of Computing Science, Glasgow University, Jan 1996.

[6] SL Peyton Jones, WD Partain, and A Santos, *"Let-floating: moving bindings to give faster programs"*, Proc ACM International Conference on Functional Programming (ICFP96), Philadelphia, May 1996.

[7] SL Peyton Jones, A Santos, *"A transformation-based optimiser for Haskell"*, Science of Computer Programming, to appear.

3 Profiling

"We will extend the existing Glasgow Haskell system with a suite of profiling tools that will make it significantly more useful to users. We will use these profiling tools to improve the performance of compiler itself, and will use this experience to refine the tools."

The idea of cost-centre profiling, outlined in the proposal, turned out to be a real winner. The ubiquitous presence of higher order functions and lazy evaluation make accurate and predictable profiling a surprisingly subtle business. Our main breakthrough was to develop a formal model for profiling, and to use that as the basis for the implementation. This work led to Sansom's PhD thesis [9], a publication in POPL [11], and a subsequent substantial paper in TOPLAS [12].

Cost-centre profiling is fully implemented in GHC, and has proved highly effective in practice. The idea has been refined in interesting further work at Durham (Jarvis [1966]; Morgan & Jarvis [1995]).

We used the profiler extensively to profile the compiler, discovering thereby some amazing performance bugs [8]. We also discovered some important shortcomings in the profiler, that led to entirely non-obvious refinements in the formal model. (The details can be found in [12].)

3.1 Publications

[8] PM Sansom *"Time profiling a lazy functional compiler"*, in "Functional Programming, Glasgow 1993", ed K Hammond, JT O'Donnell, Springer-Verlag Workshops in Computing, 1994, pp 252-264.

[9] PM Sansom *"Execution profiling for non-strict functional languages"*, PhD thesis, Technical Report FP-1994-09, Department of Computing Science, University of Glasgow, Sept 1994.

[10] SL Peyton Jones and D Wakeling *"Compilers and profiling tools"*, in "Applications of functional programming", ed Runciman and Wakeling, UCL Press 1995.

[11] PM Sansom and SL Peyton Jones *"Time and space profiling for non-strict higher-order functional languages"*, Proc ACM Symposium on Principles of Programming Languages, San Francisco, Jan 1995.

[12] PM Sansom and SL Peyton Jones, *"Formally-based profiling for higher-order functional languages"*, ACM Transactions on Programming Languages and Systems 19(1), Jan 1997.

4 The `nofib` suite

"We will produce a publicly-available suite of Haskell benchmark programs, to serve as a common base for comparing implementations".

During the course of the AQUA project we produced and distributed the `nofib` suite of benchmark programs. It is divided into three sections:

- The *imaginary* suite consists of 10 "toy" programs, such as factorial, tak, digits-of-e, and other programs of 20 lines or so. One can draw absolutely no conclusions from the performance of such programs, but they are nevertheless useful because they sometimes show up deficiencies in a compiler in high relief.
- The *real* suite consists of 26 real application programs. Each was written by an application programmer (not ourselves) with a view to getting a particular job done (rather than benchmarking). Programs in the real suite range in size from a few hundred lines to about ten thousand lines.
- The *spectral* suite lies between the real and imaginary suites. It consists of 45 program "kernels", such as Fast Fourier Transform or a Boyer-Moore theorem prover. These are not real applications, but they are notably more realistic than the imaginary suite.
- A *parallel* suite that contains a dozen parallel Haskell programs (see Section 5).

The `nofib` suite is continually growing; whenever we can we get applications from our users and add it to the suite. The whole suite is publicly available by FTP, and it played a major role in our performance measurements (Section 2).

5 Parallelism

The original proposal made it clear that AQUA was to target parallel platforms as well as sequential ones. At that time our main parallel platform was GRIP, a specialised multiprocessor whose hardware was designed to support graph reduction (Peyton Jones, Clack & Salkild [1989]). Designed over ten years ago, GRIP's technology is now very dated, and meanwhile multiprocessors have become a commodity item.

We therefore decided to completely re-implement our parallel implementation of Haskell, targeting it for generic

distributed-memory machines. Specifically, we built our implementation on top of the PVM (Parallel Virtual Machine) library, which is available for just about any parallel machine, whether shared memory or distributed memory. The result is GUM (Graph reduction for a Uniform Memory). GUM includes the whole of the Glasgow Haskell Compiler, largely unchanged, together with a completely new runtime system that supports:

- a shared heap distributed over a collection of disjoint address spaces, together with the necessary machinery to transparently shift data from one address space to another;
- multiple evaluation threads, together with the mechanisms to allocate them to free processors;
- communication and synchronisation mechanisms to allow the threads to communicate with each other.

All of this sounds relatively straightforward but, because functional languages provide a rather high-level view of parallelism, it is a substantial implementation challenge to (a) implement all the necessary pieces, and (b) make them work together well enough to deliver wall-clock speedups over the best sequential compiler technology. Indeed, despite the number of papers that have been written about parallel functional programming, we believe that GUM is the first publicly-released implementation that delivers wall-clock speedups. Our paper about GUM was published at PLDI'96 [18].

Along with GUM we have implemented a suite of profiling tools that allow the programmer to see multiple views of the parallel execution of the program [17]. This is done both by instrumenting GUM itself, and though a uniprocessor simulation of GUM called GranSim. GranSim is simpler to run than GUM, and also allows design choices to be explored — such as changing the number of processors, the packet size, or the network latency. Using GUM and GranSim we have published several papers giving quantitative measurements of the runtime behaviour of parallel Haskell programs [13,16,14,17,20].

The work in this strand of the project has gone well beyond what we originally proposed. We were able to carry out this extra work because (a) it met the aims of the Parade project so we could pool effort between Parade and AQUA, and (b) it fitted the research programme of a research student, Hans-Wolfgang Loidl. These two factors substantially increased the effort available for parallel implementations, to the benefit of all three participants (AQUA, Parade, and Loidl). The publications cited below are therefore joint with Parade.

5.1 Publications

[13] G Akerholt, K Hammond, P Trinder and SL Peyton Jones *"Processing transactions on GRIP, a parallel graph reducer"*, Proc Parallel Architectures and Languages Europe (PARLE93), eds A Bode, M Reeve, G Wolf, Munich, June 1993, p634-647.

[14] K Hammond, JS Mattson, SL Peyton Jones *"Automatic spark strategies and granularity for a parallel functional language reducer"*, CONPAR 1994, Linz, Austria, Springer LNCS 854, Sept 1994, pp521-532.

[15] HW Loidl, K Hammond, *"GRAPH for PVM: Graph Reduction for Distributed Hardware"*, Sixth International Workshop on the Implementation of Functional Languages (IFL'94), University of East Anglia, Norwich, Sept 1994.

[16] HW Loidl, K Hammond, *"On the Granularity of Divide-and-Conquer Parallelism"*, Glasgow Workshop on Functional Programming, Ullapool, Springer-Verlag Workshops in Computing, July 1995.

[17] K Hammond, HW Loidl, A Partridge, *"Visualising granularity in parallel programs: a graphical winnowing system for Haskell"*, Proceedings of High Performance Functional Computing, ed APW Bohm & JT Feo, Lawrence Livermore National Laboratory, April 1995.

[18] PW Trinder, K Hammond, JS Mattson, AS Partridge, SL Peyton Jones, *"GUM: a portable, parallel implementation of Haskell"*, Proc ACM Symposium on Programming Languages Design and Implementation (PLDI'96), Philadelphia, May 1996.

[19] HW Loidl, K Hammond, *"A Sized Time System for a Parallel Functional Language"*, electronic proceedings of the Glasgow Workshop on Functional Programming, July 1996.

[20] HW Loidl, K Hammond, *"Making a Packet: Cost-Effective Communication for a Parallel Graph Reducer"*, in Eighth International Workshop on the Implementation of Functional Languages (IFL'96), Bonn/Bad-Godesberg, Germany, Sept 1996.

[21] PW Trinder, K Hammond, H-W Loidl, SL Peyton Jones, *"Algorithms + Strategies = Parallelism"*, submitted for publication in the Journal of Functional Programming.

6 Other achievements

As well as carrying out the goals of the original proposal, we were able to make substantial progress on several other fronts. Little, if any, of this progress would have been possible without the support of the AQUA project team, and the Glasgow Haskell Compiler.

The Haskell language. We have continued to be major contributors to the Haskell language design and implementation. Haskell is now *the* non-strict functional language. Its continued health depends on the design and implementation efforts of research groups such as ours; GHC is one of only two major Haskell compilers in the world. The major event during the AQUA project was the development of Haskell 1.3, in which significant new features were added to the language.

State monad. At POPL 1992 we published "Imperative functional programming" (Peyton Jones & Wadler [1993]), which showed how to express computations that perform input/output using *monads* (Wadler

[1992]). Since then the monadic approach has become the accepted way to combine state with purely-functional programming.

During 1993/4 we made important further progress with this idea, by showing how to *encapsulate* a stateful computation so that it has a purely-functional interface. We made use of *parametricity* to show that no side effects could "escape" from the computation. This work led to a PLDI paper [25], and a subsequent substantial journal paper [26].

State monads are implemented in GHC; indeed all input/output and array manipulation is done using the state monad.

Deforestation. An exciting development just before the start of AQUA was our discovery of the `foldr/build` law that enabled *short-cut deforestation*, a transformation that removes intermediate lists from functional programs. Gill continued to study the transformation during the AQUA project, under Peyton Jones's supervision and with AQUA support. He discovered important refinements to the basic idea (notably the need for a more sophisticated inlining strategy, and the generalisation of `foldr/build` to `foldr/augment`). Our FPCA'93 paper [23] led to a flurry of papers describing interesting further developments (Hu, Iwasaki & Takeichi [1996]; Launchbury & Sheard [1995]; Takano & Meijer [1995]). Measurements of effectiveness of short-cut deforestation were mentioned earlier [3,5].

Concurrent Haskell. Some application areas require explicit concurrency; for example, a graphical user interface is often best modeled as a number of interacting input/output-performing threads. Being a deterministic language, Haskell cannot directly express such concurrency, which is inherently non-deterministic.

Motivated by this need we designed and implemented a concurrent extension to Haskell, Concurrent Haskell, on top of which Finne has built the Haggis graphical user interface toolkit as part of his PhD research (Finne & Peyton Jones [1995]). We published a paper on Concurrent Haskell at POPL 1996 [27], and our GHC distribution includes a Concurrent Haskell implementation.

Foreign language interface. Languages that cannot inter-work with other better-established languages are doomed to a slow death. One of the advantages of our approach to I/O is that it naturally accommodated a way to call C procedures directly (the so-called `ccall` mechanism). GHC has supported `ccall` since 1993.

When we used the `ccall` mechanism to support a "customer", the SADLI project at the MRC in Edinburgh, we discovered that Haskell might hold a pointer to a `malloc`'d C data structure. This structure could only be freed when Haskell discarded the pointer, an event which is not usually detectable. This led us to implement *foreign-object pointers*, data values for which the Haskell garbage collector would call a finalisation routine when discarding them. Using this technology, SADLI successfully used Haskell in a mixed C/Haskell implementation of cytology screening (Poole [1995]).

More recently, we have developed *Green Card*, a preprocessor for Haskell that generates much of the boilerplate code necessary to marshal data across the interface between Haskell and C. Green Card has been used successfully to generate the Haskell/C interface to provide a Haskell interface for the Windows API on a PC. This work supports a new and exciting collaboration with Microsoft, in which they are using Haskell as an animation modeling language, and GHC as its implementation. We have written an as-yet-unpublished report on Green Card [28].

A typed intermediate language. There is increasing interest in strongly-typed intermediate languages for sophisticated compilers (Morrisset [1995]; Shao & Appel [1995]; Tarditi et al. [1996]). GHC was a pioneer here; since 1990 we have used the second-order lambda calculus as its intermediate language [22]. We are now working on a yet more expressive, yet more economical, intermediate language, Henk, based this time on the *lambda cube* (which contains the second order lambda calculus as a special case). Our paper, which is submitted to the 1997 Types in Compilation workshop, gives the details [29].

6.1 Publications

[22] SL Peyton Jones, CV Hall, Hammond, WD Partain, and PL Wadler *"The Glasgow Haskell compiler: a technical overview"*, JFIT Technical Conference, Keele, March 1993, pp249-257.

[23] A Gill, J Launchbury and SL Peyton Jones *"A short cut to deforestation"*, Proc of Functional Programming Languages and Computer Architecture 93, Copenhagen, ACM Press, June 1993, pp223-232.

[24] J Launchbury, *"Lazy imperative programming"*, ACM Workshop on State in Programming Languages (SIPL'93), Copenhagen, June 1993.

[25] J Launchbury and SL Peyton Jones *"Lazy functional state threads"*, Proc ACM Conference on Programming Language Design and Implementation, Orlando, June 1994, pp24-35.

[26] J Launchbury and SL Peyton Jones *"State in Haskell"*, Lisp and Symbolic Computation (special issue on state) 8(4), pp293-341, Dec 1995.

[27] SL Peyton Jones, AJ Gordon, and SO Finne *"Concurrent Haskell"*, Proc ACM Symposium on Principles of Programming Languages, Jan 1996.

[28] SL Peyton Jones and T Nordin, *"Green Card: a foreign language interface for Haskell"*, Technical Report, University of Glasgow, Jan 1997.

[29] SL Peyton Jones and E Meijer, *"Henk: a typed intermediate language"*, submitted to Types in Compilation Workshop (June 1997).

7 Other publications

This section collects other project publications that do not fall naturally into the preceding sections.

[30] CV Hall, K Hammond, SL Peyton Jones, PL Wadler *"Type classes in Haskell"*, European Symposium on Programming (ESOP'94), ed D Sannella, Springer Verlag LNCS 788, April 1994, pp241-256. This paper gives a formal account of Haskell's type classes. Type classes are the most distinctive feature of the Haskell language, and they are also the most challenging technically.

[31] CV Hall, K Hammond, SL Peyton Jones, PL Wadler, *"Type Classes in Haskell"*, ACM Transactions on Programming Languages and Systems 18(2), March 1996, 109-138. This paper is an expanded journal version of [30].

[32] SL Peyton Jones and A Santos *"Compilation by transformation in the Glasgow Haskell Compiler"*, Functional Programming Glasgow 1994, Springer-Verlag Workshops in Computing, eds Hammond, Turner & Sansom, 1995, pp184-204. This workshop paper collects together many of the transformations used in GHC.

[33] SL Peyton Jones, *"Compilation by transformation: a report from the trenches"*, European Symposium on Programming (ESOP'96), Springer Verlag LNCS 1058, Jan 1996, pp18-44. A completely rewritten and expanded version of [32], presented as an invited paper.

[34] SL Peyton Jones, *"Bulk types with class"*, electronic proceedings of the Glasgow Workshop on Functional Programming, July 1996. A new paper, so far published only electronically, discussing the design of a library of functions manipulating *bulk types*, such as sets, bags, lists, finite maps, and so on.

8 Project management and personnel

The project was led by Professor Simon Peyton Jones, with support from Professor Phil Wadler and Dr John Launchbury. We were fortunate to attract two outstanding Research Assistants, Dr Will Partain and Dr Jim Mattson. This group formed the core team, which turn supported, and was enriched by, close interactions with other individuals and groups.

- Several research students: Patrick Sansom, André Santos, Andy Gill, Sigbjorn Finne, and Simon Marlow.
- Dr Kevin Hammond, a research fellow.
- Two research assistants, Dr Phil Trinder and Dr David Turner, both funded by other EPSRC grants (Parade and Linear Types respectively).
- Four short-term appointments: Bryan O'Sullivan and Darren Moffat (vacation students); and Hans Loidl and Alastair Reid (research students). (See Section 9.)
- Phil Broughton, a highly experienced technical project manager at ICL. Phil attended regular project steering-group meetings, and provided an external critical eye on our progress and objectives.
- The Lolita research group at the University of Durham. Lolita is a large natural-language processing toolkit, written entirely in Haskell. This group have become one of our largest users, and we now have a joint research project with them (DEAR).
- A large and growing body of users of both the Haskell language, and the Glasgow Haskell Compiler. We regard our users as an important source of guidance for our future priorities. GHC is used right across the world (Ireland, Hawaii, Germany, Australia, Eastern Europe, Brazil, to name but a few).

9 Deployment of resource

The only hiccup in the project came when Jim Mattson, one of the two RAs, left earlier than expected. With the agreement of EPSRC we were able to plug the gap this left by employing several short-term staff, as mentioned above.

We spent more money than expected on travel, by some £4k. The principal investigators are well-known internationally, and we simply underestimated the costs of their travel.

We also spent more than expected under the consumables heading, by some £5k. For some reason, the amount originally awarded was some £5k less than that requested in the proposals, and the sum requested was actually required. One reason for the high figure requested (explicit in the RG2) was the maintenance costs for GRIP's host machine, an outdated but irreplaceable Sun3 server.

10 Conclusion

AQUA has been my main "baby" for the last three years[2]. It has been the focus of my research efforts, and the core of my research programme.

I believe that AQUA has been unusually successful, in both the quantity and quality of research that it has generated within its area. One important reason for this success is that AQUA had enough core effort to support an informal and diverse collection of partners, *to the mutual benefit of all.* I believe this is a good model for research, and one I hope to continue. I am grateful for the EPSRC support that made AQUA possible.

References

S Finne & SL Peyton Jones [Sept 1995], "Composing Haggis," Proc 5th Eurographics Workshop on Programming Paradigms in Graphics, Maastricht.

Z Hu, H Iwasaki & M Takeichi [May 1996], "Deriving structural hylomorphisms from recursive definitions," in *Proc International Conference on Functional Programming, Philadelphia*, ACM, .

[2] I have been involved in other projects as co-investigator, but in each case the project has been led by someone else.

SA Jarvis [1966], "Profiling large-scale lazy functional programs," PhD thesis, Department of Computer Science, University of Durham.

J Launchbury & T Sheard [June 1995], "Warm fusion," in *Proc Functional Programming Languages and Computer Architecture, La Jolla*, ACM.

RG Morgan & SA Jarvis [Apr 1995], "Profiling large-scale lazy functional programs," in *Proceedings of High Performance Functional Computing*, APW Bohm & JT Feo, eds., Lawrence Livermore National Laboratory, .

G Morrisset [Dec 1995], "Compiling with types," PhD thesis, CMU-CS-95-226, Carnegie Mellon University.

SL Peyton Jones, C Clack & J Salkild [June 1989], "High-performance parallel graph reduction," in *Proc Parallel Architectures and Languages Europe (PARLE)*, E Odijk, M Rem & J-C Syre, eds., LNCS 365, Springer Verlag, .

SL Peyton Jones & PL Wadler [Jan 1993], "Imperative functional programming," in *20th ACM Symposium on Principles of Programming Languages, Charleston*, ACM, .

I Poole [March 1995], "Public report of the SADLI project: safety assurance in diagnostic laboratory imaging," MRC Human Genetics Unit, Edinburgh.

Z Shao & AW Appel [June 1995], "A type-based compiler for Standard ML," in *SIGPLAN Symposium on Programming Language Design and Implementation (PLDI'95), La Jolla*, ACM, .

A Takano & E Meijer [June 1995], "Deforestation in calculational form," in *Proc Functional Programming Languages and Computer Architecture, La Jolla*, ACM.

D Tarditi, G Morrisett, P Cheng, C Stone, R Harper & P Lee [May 1996], "TIL: A Type-Directed Optimizing Compiler for ML," in *SIGPLAN Symposium on Programming Language Design and Implementation (PLDI'96), Philadelphia*, ACM.

PL Wadler [Jan 1992], "The essence of functional programming," in *19th ACM Symposium on Principles of Programming Languages, Albuquerque*, ACM, .