PARallel Database Engine (Parade) Final Report

Kevin Hammond (St Andrews University),
Simon Peyton Jones (Glasgow University)
Phil Trinder (The Open University)

December 19, 1997

1 Summary

This report summarises the achievements of the EPSRC Parade project (GRJ/53348), which ran from June 1994 to September 1997. We have made contributions in the following three areas.

- **Parallel Language Support (Section 2).** We have implemented a robust, portable runtime system and compiler support for Glasgow Parallel Haskell. The system supports a complete parallel development environment providing an integrated simulator and sophisticated profiling tools in addition to physical implementations on a variety of parallel machines.

  Despite sustained research interest in parallel functional programming, our implementation is one of the first to be made publicly available, and to be widely used outside the research group that developed it.

- **Large-scale data-intensive parallel programming (Section 3).** In collaboration with groups at Durham University and the Centre for Transport Studies, London we have developed and measured several large parallel data-intensive programs. We have demonstrated wall-clock speedups for both text-book and, more significantly, real problems on both distributed-memory and shared-memory architectures.

- **Parallel Programming (Section 4).** Motivated by our studies of large-scale data-intensive programs, we have developed *evaluation strategies*, a new programming abstraction that cleanly separates code describing the algorithm from that controlling the parallel behaviour. Evaluation strategies can be used to describe a wide range of control and data parallel programming paradigms, are user-extensible, can be nested or composed arbitrarily, and provide a simple framework for reasoning about parallel program behaviour.

  Based on our experiences with using evaluation strategies, we are developing a methodology for constructing large parallel functional programs based on a combination of simulation, profiling and performance measurement on real machines.

A major focus of this project is its emphasis on parallel *programming* rather than parallel *implementations*. In sharp contrast to the majority of projects in this area we have progressed beyond simply obtaining a working parallel implementation to studying the problems associated with producing real parallel programs written in a functional style, and manipulating large quantities of data.

We have published widely during the course of the project, producing almost 30 Journal, Conference, and Workshop papers, edited several workshop proceedings and contributed to a number of books. Several other papers are in preparation. We interact with many groups both nationally and internationally, and have recently been instrumental in establishing a special interest group within Scotland. Encouraged by free, public availability, our software and techniques are being used and further developed by groups worldwide. Our papers are available electronically from the URL below:

```
http://www.dcs.gla.ac.uk/fp/software/gransim/
      GranSim-GUM-papers.html
```

In the following text, 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 Parade, are cited thus, [PCSH87] and appear at the end as usual.

The remainder of this report is structured as follows. Sections 2 to 4 describe our technical contributions. Sections 5–6 describe other project achievements. Finally Section 7 describes the project management structure and identifies external links and collaborations.

2 Parallel Language Support

Several software components were constructed to enable the research. The most important of these are GUM, a robust parallel functional language implementation, and GranSim, a parallel simulator with sophisticated visualisation tools [4]. These software components have found widespread use both in the UK and elsewhere in the world including Australia, Brazil, Germany, Japan, Slovakia and the United States.

The GUM runtime system was constructed in the first 14 months of the project to support Glasgow Parallel Haskell (GPH), the implementation language for the data-intensive programs. It was constructed in cooperation with the EPSRC AQUA project. GUM is robust and portable, being based on the PVM communications harness available on many multiprocessors. As a result it is available for both shared-memory (Sun SPARCserver multi-processors) and distributed-memory (IBM SP2 and Connection Machine

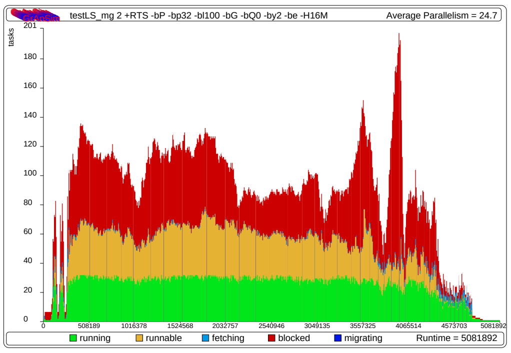

Figure 1: Overall activity profile of a linear system solver

CM5) parallel architectures as well as networks of workstations. GUM delivers wall-clock speedups relative to the best available sequential compiler technology [5]. GUM was developed using experience gained from the EPSRC funded GRIP project [PCSH87], through an interim PVM-based version of GRIP [2]. GUM is freely available for public access as part of the Glasgow Haskell Compiler distribution, and we have encouraged others to both develop and port it. As a result a group in New Mexico have ported GUM to the CM5, and groups in Australia, Brazil and Germany are working on an IBM SP2 port and an improved load-balancing mechanism.

GranSim is a flexible simulator for GpH. It is highly parameterised in order to allow idealistic simulation as well as accurate modelling of the wide range of architectures that are potentially supported by GUM. Unlike most comparable simulators it measures execution in machine cycles and uses an optimising compiler (GHC), thus achieving a very accurate simulation.

We have used GranSim in three ways. Firstly, we have used it during two consecutive parallel program development stages: an initial idealised simulation is followed by accurate simulation modelling the final target architecture. Secondly, we have used GranSim to prototype runtime-system designs including various graph packing schemes and mechanisms for latency hiding [6]. Our simulations have allowed us to take concrete steps to improve the GUM runtime-system. Finally, GranSim implements a set of visualisation tools that provide detailed activity profiles, granularity information, and information about the parallel distribution of resources. The same set of visualisation tools can be used for both GranSim and GUM.

Like GUM, GranSim is distributed with the Glasgow Haskell Compiler. It is currently being used by groups in York, Durham, St Andrews, Amsterdam, and Kyoto. Some of the visualisation tools have also been adapted for the use in other simulators.

2.1 Publications

[1] Hammond K., Loidl H.-W., Mattson J.S. Jr., Partridge A.S., Peyton Jones S.L., and Trinder P.W., "GRAPHing The Future", *Proc. 6th Intl. Workshop on the Implementation of Functional Languages*, Norwich, England (September 1994).

[2] Loidl H.-W. and Hammond K., "GRAPH for PVM: Graph Reduction for Distributed Hardware", *Proc. 6th Intl. Workshop on the Implementation of Functional Languages*, Norwich, England (September 1994).

[3] Hammond K., Mattson J.S. Jr., Partridge A.S., Peyton Jones S.L., and Trinder P.W., "GUM: a portable Parallel Implementation of Haskell", *Proc. of the 7th. Intl. Workshop on Implementation of Functional Languages*, Bøastad, Sweden (September 1995).

[4] Hammond K., Loidl H.-W., and Partridge A.S., "Visualising Granularity in Parallel Programs: A Graphical Winnowing System for Haskell", *Proc. HPFC'95 — High Performance Functional Computing*, Denver, Colorado, (April 1995), pp. 208–221.

[5] Trinder P.W., Hammond K., Mattson J.S. Jr., Partridge A.S., and Peyton Jones S.L., "GUM: a Portable Parallel implementation of Haskell", *Proc. Programming Languages Design and Implementation (PLDI '96)*, Philadelphia, USA, (May 1996).

[6] Loidl H.-W., and Hammond K., "Making a Packet: Cost-Effective Communication for a Parallel Graph Reducer", *Proc. 8th Intl. Workshop on the Implementation of Functional Languages*, Bonn, Germany, (September 1996), Springer-Verlag LNCS 1268, pp. 184–199.

3 Data-intensive Parallel Programming

The broad objective of the Parade project was to investigate data-intensive programming in a parallel functional language. This area has not been heavily studied to date in the parallel community, and our results thus represent some of the first attempts to parallelise such programs. Our initial target application was transaction-processing. However, it rapidly became apparent that this application did not utilise the strengths of the GpH runtime system that was constructed during the first phase of the project (dynamic data and task placement, dynamic load management, computational power, a rich type system and equational reasoning). We therefore chose instead to investigate complex parallel database queries, since such queries exploit much of the power of a parallel functional language.

Constructing the large data-intensive programs described below has required a synthesis of experience, tools and techniques. We have built on our experience of large-scale parallel functional programming [8] and of functional database programming languages [13]. The programs are measured and run on the GUM/GranSim platforms described above. Section 4 describes the mechanism and techniques that were developed in order to engineer the programs.

Demonstrator. We initially chose a simple example to demonstrate the feasibility of using GpH for data-intensive programs. The problem chosen was the bill-of-material, or parts explosion, problem from Date's textbook [Da76]. The demonstrator gives good simulated results, but unfortunately the Sun SPARCserver machine available is heavily used. On four processors the program achieves a *relative* speedup of 2.6 over a single-processor, and an *absolute* wall-clock speedup of 2.2 over the optimised sequential code [15].

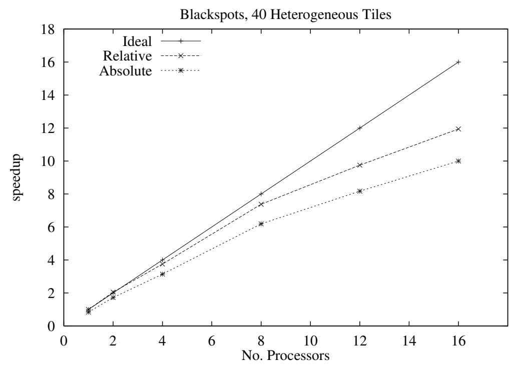

Figure 2: Speedups on a Workstation Network

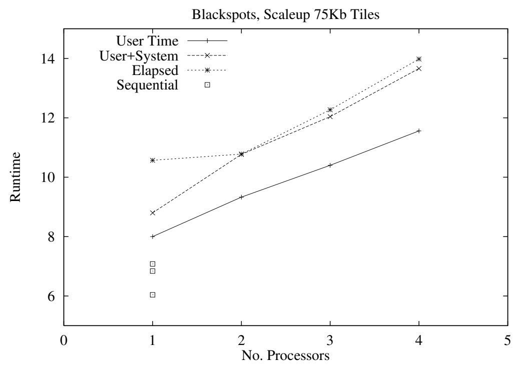

Figure 3: Scaleup on Sun SPARCserver

Accident Blackspots. Cooperating with a group at the London Centre for Transport studies we have constructed a parallel program to locate road traffic accident blackspots from police reports. This group already had an implementation [WH96] in the PFL database programming language [SP91]. However, our compiled sequential Haskell implementation is an order of magnitude faster than this interpreted PFL implementation.

Several different means of parallelising the program were investigated before a satisfactory version was obtained by partitioning the data spatially into a number of overlapping tiles. The parallelism is coarse-grained as a single task locates the blackspots in each tile. The program has been measured on both the shared-memory Sun SPARCserver and on a network of Sun 4/15 workstations. On both architectures it achieves

- **Good Speedups** compared with optimised code produced by the 'best sequential Haskell compiler in the world'. Figure 3 is a speedup graph for the workstations.
- **Satisfactory Scaleups.** Figure 3 is a scaleup graph for the SPARCserver illustrating that increasing the quantity of data processed in line with the number of processors that are available does not significantly degrade performance.

The dynamic nature of a parallel functional language gives our implementation several advantages over the conventional coarse-grained implementation technique of statically allocating a fixed set of tiles to each processor. Our implementation uses a shared file-system to dynamically allocate tiles to processors from a single pool. This results in good performance even when tile size, time-required to process a tile and processor speed all vary. It also makes the program independent of the number of tiles. Because the model of parallelism is abstract the program is portable, and even the difference between shared and distributed memory architectures is transparent, subject to the availability of a shared file system.

Lolita. We have cooperated with the EPSRC DEAR project to parallelise the Lolita natural language processor. Lolita has been developed at Durham University over the past 10 years and is large, comprising 60K lines of Haskell and a few thousand lines of C. Lolita is data-intensive in accessing large, read-only, data structures that typically occupy around 16Mb. It also generates large internal data structures, e.g. for each sentence in the text to be analysed Lolita generates a forest of possible parses and a forest of possible meanings.

Surprisingly, parallelising Lolita entailed minimal changes to the 300 modules of code. A realistic simulation gives an average parallelism of 3.1, which is satisfactory in view of an initial sequential segment of C-parsing. Wallclock speedups obtained to date on the Sun SPARCserver are disappointing because of Lolita's heavy memory requirements. The current machine has just 256Mb of physical memory, and each copy of the parallel program requires its own 100Mb partition. In consequence even with just two processors physical memory is exhausted and an average parallelism of 1.4 obtains a slowdown of approximately 10%. We aim to achieve better results by using a dedicated machine with a 512Mb physical memory, by making the C re-entrant and hence parallelising it, and by arranging for the processors to share a single copy of the read-only data structures.

Bowing calculations. We are in the process of parallelising a program for automatically calculating the bowing notations used by violinists and other stringed instrument players to cue the direction in which the bow should be moved 7. A sequential version of this program was originally written in Ada. By converting it to Haskell and applying our parallelisation tools, we hope to be able to learn enough about the parallel structure of the algorithm that we can create a parallel version of the original Ada. This will improve the program's performance on complicated musical passages and so permit real time analysis. Such an analysis would be useful to professional musicians playing an unfamilar piece at short notice (e.g. for a film score) as well as to orchestral/chamber players and amateur musicians. Our initial assessment suggests that the program is amenable to the data-oriented techniques we have used elsewhere.

Language support for bulk data. To make it better suited for data-intensive programming, we have investigated ways of improving the bulk type support in Haskell [9,14]. By exploiting Haskell's existing type class mechanism [25] and features proposed for the new definition of standard Haskell such as multi-parameter type classes, we have succeeded in defining general polymorphic operations over bulk classes. This aids abstraction by permitting transparent replacement of one bulk type by another that is more suitable to the task at hand, and increases code reuse by allowing more general data-manipulating definitions to be written.

3.1 Publications

Many of these results are recent and the work is ongoing. We are currently writing papers describing our experiences with both programs, and another distilling our experiences with these and other large programs.

[7] Hall C.V., Hammond K., Loidl H.-W., O'Donnell J.T and Trinder P.W., "Refining a Parallel Algorithm For Calculating Bowings" *Proc. 1997 Glasgow Workshop on Functional Programming*, Ullapool, Scotland, (September 1997).

[8] Hammond K., "Parallel Implementation on GRIP", In [RW95], (1995).

[9] Hammond K., and Trinder P.W., "Database Manipulation in Haskell 1.3", *Proc. 1995 Glasgow Workshop on Functional Programming*, Ullapool, Scotland (July 1995) Springer-Verlag.

[10] Loidl H.-W., "LinSolv: a Case Study in Strategic Parallelism" *Proc. 1997 Glasgow Workshop on Functional Programming*, Ullapool, Scotland, (September 1997).

[11] Loidl H.-W. and Trinder P.W., "Engineering Large Parallel Functional Programs", Submitted to *Proc. IFL'97 — Intl. Workshop on the Implementation of Functional Languages*, St Andrews, Scotland, (September 1997).

[12] Loidl H.-W., Morgan R. and Trinder P.W., "Parallelising a Large Functional Program; or Keeping Lolita Busy", Submitted to *Proc. IFL'97 — Intl. Workshop on the Implementation of Functional Languages*, St Andrews, Scotland, (September 1997).

[13] Paton N., Cooper R., Williams H., and Trinder P.W., *Database Programming Languages*, Prentice Hall (1996).

[14] Peyton Jones S.L., "Bulk types with class", *Proc. 1997 ACM SigPlan Haskell Workshop*, Amsterdam, (June 1997).

[15] Trinder P.W., Hammond K., Loidl H.-W., Peyton Jones S.L., and Wu J., "A Case-study of Data-intensive programs in Parallel Haskell", *Proc. 1996 Glasgow Workshop on Functional Programming*, Ullapool, Scotland (July 1996).

4 Parallel Functional Programming

Our experiences engineering the large data-intensive programs described in the previous section have led us to make several contributions to parallel functional programming. We have made a number of general contributions [16,17], and the following specific contributions.

Evaluation Strategies. The process of writing large parallel programs is complicated by the need to specify both the parallel behaviour of the program and the algorithm that is to be used to compute its result. *Evaluation strategies* are lazy higher-order functions that control the parallel evaluation of non-strict functional languages. Using evaluation strategies, it is possible to achieve a clean separation between algorithmic and behavioural code. The result is enhanced clarity and shorter parallel programs [21]. Strategies can express a wide range of parallel programming paradigms, like divide-and-conquer or pipelining. We use evaluation strategies in the data-intensive programs, and encourage their use in all GpH programs. They are being used by groups at Durham, St Andrews and Passau Universities. We are currently developing a framework for reasoning about strategic functions.

Methodology. Based on our experiences during this project we are developing a methodology for engineering large parallel programs [15]. We adopt a top-down approach, commencing with the top-level pipeline and parallelising successive components of the program. Our methodology uses several existing tools, including sequential time profiling [SP97] and GranSim. We have also identified the need for other tools to support the process, which has led to the development of a source-level parallel profiling technique [18] based on the sequential work mentioned above [SP97].

Granularity. The importance of good task granularity in large parallel programs is clearly apparent from the examples we have considered. We have investigated various means of monitoring and both statically and dynamically improving granularity [19, 20], including control mechanisms based on evaluation strategies. This work is ongoing.

4.1 Publications

[16] Hammond K., Mattson J.S. Jr., and Peyton Jones S.L., "Automatic Spark Strategies and Granularity for a Parallel Functional Language Reducer" *Proc. CONPAR '94*, Linz, Austria (September 1994), Springer-Verlag LNCS 854, pp. 521–532.

[17] Hammond K., "Parallel Functional Programming: An Introduction" *Proc. 1st Intl. Symposium on Parallel Symbolic Computation (PASCO '94)*, Hagenberg/Linz, Austria, (September 1994), pp. 181–193, World Scientific.

[18] Hammond K., Loidl H.-W., and Trinder P.W., "Parallel Cost Centre Profiling" Submitted to *Proc. IFL'97 — Intl. Workshop on the Implementation of Functional Languages*, St Andrews, Scotland, (September 1997).

[19] Loidl H-W. and Hammond K., "A Sized Time System for a Parallel Functional Language", *Proc. 1996 Glasgow Workshop on Functional Programming*, Ullapool, Scotland (July 1996).

[20] Loidl H-W. and Hammond K., "On the Granularity of Divide-and-Conquer Parallelism", *Proc. 1995 Glasgow Workshop on Functional Programming*, Ullapool, Scotland (July 1995) Springer-Verlag.

[21] Trinder P.W., Hammond K., Loidl H.-W., and Peyton Jones S.L., "Algorithm + Strategy = Parallelism", *To appear in* The Journal of Functional Programming, (January 1998).

5 Other Achievements

This section enumerates the contributions we have made in areas not immediately within the remit of the Parade project.

- We have been active in establishing SCOFPIG, the SCOttish Functional Parallel-programming Interest Group. This provides a forum for informal research presentation and discussion. Members are drawn from research groups at Glasgow, Edinburgh, Heriot-Watt and St Andrews Universities and possess a strong diversity of interest.
- We hosted a number of national and international researchers during the 1995 Glasgow Research Festival [PT96], when we ran a mini-workshop on functional database technology.
- We have developed a publicly-available set of parallel GpH applications as part of the `nofib` suite of Haskell benchmark programs.
- We have been heavily involved in the continuing design of the Haskell language definition as joint editors of the language report [28,26].
- We have made concrete suggestions aimed at improving the usability of the standard Haskell libraries and their support for bulk data types. We are in the process of creating a repository for new and evolving Haskell libraries.

6 Other Publications

This section lists other publications by members of the project team which are only indirectly relevant to the goals of the project, or which are supportive but do not introduce new research results.

[22] Chan D.K.C. and Trinder P.W., "A Processing Framework for Object Comprehensions", To appear in *Journal of Information and Software Technology.*

[23] Davie A.J.T. and Hammond K., "Functional Hypersheets", *Proc. 1996 Glasgow Workshop on Functional Programming*, Ullapool, Scotland (July 1996).

[24] Gordon A.D. and Hammond K., "Monadic I/O in Haskell 1.3", *Proc. 1995 Workshop on Future Directions of Haskell*, La Jolla, California, (July 1995).

[25] Hall C.V., Hammond K., Peyton Jones S.L. and Wadler P.L., "Type Classes in Haskell", *ACM TOPLAS*, **18(2)**, pp. 109–138, (April 1996).

[26] Hammond K., Peterson J.C. (eds.), et al. *Haskell 1.4 — A Non-Strict, Purely Functional Language*, (April 1997).

[27] Hammond K., Turner, D.N. and Sansom P.M. (Eds.) *Proc. 1994 Glasgow Workshop on Functional Programming*, Ayr, Scotland (July 1994), Springer-Verlag.

[28] Peterson J.C., Hammond K. (eds.), et al. *Haskell 1.3 — A Non-Strict, Purely Functional Language*, (May 1996).

[29] Trinder P.W. (Ed.), *Electronic Proceedings of the 1996 Glasgow Workshop on Functional Programming*, Ullapool, Scotland (July 1996).

7 Project Management and Collaborations

The project was directed jointly by Dr. Kevin Hammond, Prof. Simon Peyton Jones and Dr. Phil Trinder, the latter being employed as a post-doctoral Research Assistant on the project. Prof. Nic Holt, a senior software engineer with ICL remained in constant touch with the work. The focus of the project was maintained through joint Glasgow-ICL steering group meetings held on a biannual basis.

The individuals above formed the core project team who in turn interacted with the groups and individuals listed below.

- Research students including Hans-Wolfgang Loidl who worked on many of the programs described above as well as significantly enhancing the GranSim simulator.
- Members of the EPSRC AQUA project including Dr. Jim Mattson who played a critically important role in implementing and developing the GUM runtime system and who also provided essential technical support for the Glasgow Haskell compiler in conjunction with the main compiler implementor, Dr. Will Partain.
- Mr. Phil Broughton, a highly experienced technical project manager at ICL. Mr. Broughton 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. This group forms the largest user of our software and has provided the biggest data-intensive program that we have studied to date.
- Other users of GpH worldwide.
- Researchers and implementors at the University of Tasmania (Australia), Los Alamos National Laboratories (USA), Philips-Universität Marburg (Germany), and York University (UK). These groups are in the process of exploiting our work to enable their own research interests.
- The London Centre for Transport Studies, University College London provided us with the traffic accident example: a real application with real data, suitably modified to protect privacy.

References

[Da76] Date C.J., *An Introduction to Database Systems* 4th Edition. Addison Wesley (1976).

[Hal94] Halstead Jr. R., "Self-describing files + smart modules = parallel program visualisation", *Theory and Practice of Parallel Programming* Springer-Verlag LNCS 907, pp. 253–283, (1994).

[PCSH87] Peyton Jones S.L., Clack C., Salkild J. and Hardie M., “GRIP – a high-performance architecture for parallel graph reduction”, *Proc. FPCA 87*, Portland, Oregon, Kahn G. (ed.), Springer-Verlag LNCS, (1987).

[PT96] Peyton Jones S.L. and Thomas M., “The Glasgow Computing Science Research Festival”, TR-1996-2, Dept. of Computing Science, Glasgow University, (January 1996).

[RW95] Runciman C., and Wakeling D. (eds.), *Applications of Functional Programming.* 1995. UCL Press.

[SP97] Sansom P.M. and Peyton Jones S.L., “Formally-based profiling for higher-order functional languages”, *ACM Transactions on Programming Languages and Systems*, **19(1)**, January 1997

[SP91] Small C. and Poulovassilis A.P., “An Overview of PFL”, *Proc. 3rd. Intl. Workshop on Database Programming Languages*, Kanellakis P. Schmidt J.W. (eds) (1991) Morgan Kaufman.

[WH96] Wu J. and Harbird L., “A Functional Database System for Road Accident Analysis”, *Advances in Engineering Software*, **26(1)**, (1996), pp. 29–43.