JFP **15** (3): 351–352, 2005. © 2005 Cambridge University Press
doi:10.1017/S0956796805005496 Printed in the United Kingdom

Special Issue
High Performance Parallel Functional Programming

P. W. TRINDER
School of Mathematics and Computer Science, Heriot-Watt University, Riccarton, Edinburgh EH14 4AS, Scotland
(*e-mail:* trinder@macs.hw.ac.uk)

Engineering high-performance parallel programs is hard: not only must a correct, efficient and inherently-parallel algorithm be developed, but the computations must be effectively and efficiently coordinated across multiple processors. It has long been recognised that ideas and approaches drawn from functional programming may be particularly applicable to parallel and distributed computing (e.g. Wegner 1971). There are several reasons for this suitability. Concurrent stateless computations are much easier to coordinate, high-level coordination abstractions reduce programming effort, and declarative notations are amenable to reasoning, i.e. to optimising transformations, derivation and performance analysis.

After a long gestation the potential of functional techniques for parallel and distributed computing is now being realised in practice, and this special issue outlines contributions in several areas. Declarative techniques are being used to construct significant parallel and high-throughput systems, e.g. real-time image analysis and high-end telecoms systems. Indeed, it can be argued that high-performance systems now rival prototyping, formal reasoning and education as areas where functional technology has the greatest impact. A comprehensive survey of parallel functional technologies can be found in Hammond & Michaelson (1999), and a survey of parallel and distributed Haskells in Trinder *et al.* (2002).

Erlang is the most widely used functional language for engineering high-throughput fault-tolerant systems. Erlang was designed for, and has had most impact in, the telecoms sector, but is now spreading to other sectors like banking. The Erlang AXD301 switch, or telephone exchange, is arguably the largest and most significant functional program ever constructed. An early version comprised approximately 500K lines of new Erlang code, 300K lines of mostly-reused C and 8K lines of Java. The system executes on up to 32 processors and was written by a team peaking at 50 software engineers (Blau *et al.*, 1999). Symptomatic of Erlang's commercial success, it has proved very hard to find commercial engineers with time to contribute to an academic journal. However the paper by Gulias *et al.* describes the design, engineering and performance of VoDKA, an Erlang system capable of meeting the substantial real-time performance requirements of delivering video on demand using a cheap commodity cluster.

Engineering any sizable software requires a methodology. Systematic development is especially important for high throughput systems, where performance objectives are often at odds with simplicity and elegance. In systematic development, performance prediction enables optimisation. The paper by Luke and George describes a rule-based methodology giving predictable performance, illustrated with case studies.

For decades there has been sustained interest in the design, implementation and evaluation of parallel functional languages. The research has been greatly facilitated in the last decade when their sophisticated implementations became architecture independent, i.e. no longer tied to a specific parallel machine. Two such languages are represented here. The paper by Loogen *et al.* describes Eden, an extension of Haskell with processes, together with its formal semantics, skeleton-oriented programming methodology, implementation, and some case studies. The paper by Grelck describes SAC, a functional subset of C designed for efficient parallel array manipulation, and covers both optimisation and performance evaluation.

Parallel and distributed functional programming is likely to remain a vibrant research area as our increasingly networked hardware and middleware environments raise new challenges, many of which the functional paradigm is well-placed to address. For example Computational Grids offer enormous amounts of computing power on heterogeneous platforms distributed over wide area networks, hence requiring sophisticated and dynamic management that is hard to provide in a low-level paradigm. Functional languages have enabled high level abstractions over parallel coordination that have become accepted in the parallelism community, e.g. algorithmic skeletons (Cole, 1988). Analogous abstractions are now required for distributed and mobile coordination, and some distributed abstractions are already under development, e.g. Erlang behaviours (Armstrong, 2003). Being able to reason readily about an aspect like coordination enables interaction to be optimised, increases reliability and facilitates refactoring. The strong tradition of reasoning about parallel coordination (Skillicorn, 1992) is a good basis for addressing the new challenges of reasoning about distributed and mobile coordination.

References

Armstrong, J. (2003) *Making Reliable Distributed Systems in the Presence of Software Errors.* PhD thesis, Department of Microelectronics and Information Technology, Royal Institute of Technology, Stockholm, Sweden.

Blau, S., Rooth, J., Axell, J., Hellstrand, F., Buhrgard, M., Westin, T. and Wicklund, G. (1999) AXD 301: A new generation ATM switching system. *Computer Networks*, **31**(6), 559–582.

Cole, M. C. (1988) *Algorithmic Skeletons: Structured Management of Parallel Computation.* PhD thesis, University of Edinburgh. (Also published in book form by Pittman/MIT, 1989.)

Hammond, K. and Michaelson, G. (1999) *Research Directions in Parallel Functional Programming*. Springer-Verlag.

Skillicorn, D. B. (1992) *Parallelism and the Bird-Meertens Formalism.* Kingston, Ontario: Department of Computing and Information Science, Queen's University.

Trinder, P. W., Loidl, H.-W. and Pointon, R. F. (2002) Parallel and Distributed Haskells. *J. Funct. Program.* **12**(4&5), 469–510.

Wegner, P. (1971) *Programming Languages, Information Structures and Machine Organisation.* McGraw-Hill.