Streaming Networks for Coordinating Data-Parallel Programs (Position Statement)

Alex Shafarenko

Compiler Technology and Computer Architecture Group, University of Hertfordshire, United Kingdom
A.Shafarenko@herts.ac.uk

Abstract. A new coordination language for distributed data-parallel programs is presented, call SNet. The intention of SNet is to introduce advanced structuring techniques into a coordination language: stream processing and various forms of subtyping. The talk will present the organisation of SNet, its major type inferencing algorithms and will briefly discuss the current state of implementation and possible applications.

Process concurrency is difficult to deal with in the framework of a programming language. If properly integrated into the language semantics, it complicates and often completely destroys the properties that enable the kind of profound optimisations that make compilation of computational programs so efficient. One solution to this problem, which is the solution that this talk will present, is the use of so-called coordination languages. A coordination language uses a readily-available computation language as a basis, and extends it with a certain communication/synchronisation mechanism thus allowing a distributed program to be written in a purely extensional manner. The first coordination language proposed was Linda[Gel85, GC92], which extended C with a few primitives that looked like function calls and could even be implemented directly as such. However an advanced implementation of Linda would involve program analysis and transformation in order to optimise communication and synchronisation patterns beyond the obvious semantics of the primitives. Further coordination languages have been proposed, many on them extensional in the same way, some not; for the state of the art, see a survey in [PA98] and the latest Coordination conference [JP05].

The emphasis of coordination languages is usually on event management, while the data aspect of distributed computations is not ordinarily focused on. This has a disadvantage in that the structuring aspect, software reuse and component technology are not primary goals of coordination. It is our contention that structuring is key in making coordination-based distributed programming practically useful. In this talk we describe several structuring solutions, which have been laid in the foundation of the coordination language SNet. The language was introduced as a concept in [Sha03]; the complete definition, including semantics and the type system, is available as a technical report [Sha06].

The approach proposed in SNet is based on streaming networks. The application as a whole is represented as a set of self-contained components, called

C. Jesshope and C. Egan (Eds.): ACSAC 2006, LNCS 4186, pp. 2–5, 2006.

"boxes" (SNetis not extensional) written in a data-parallel language. SNet deals with boxes by combining them into networks which can be encapsulated as further boxes. The structuring instruments used are as follows:

- Streams. Instead of arbitrary communication, data is packaged into typed variant records that flow in a sequence from their producer to a single consumer.
- Single-Input, Single-Output(SISO) box and network configuration. Multiple connections are, of course, possible and necessary. The unique feature of SNet is that the multiplicity of connection is handled by SNet combinators so that a box sees a single stream of records coming in. The records are properly attributed to their sources by using types (which include algebraic types, or tagged, disjoint unions). Similarly, the production of a single stream of typed records by a box does not preclude the output separation into several streams according to the type outside the box perimeter.
- Network construction using structural combinators. The network is presented as an expression in the algebra of four major combinators (and a small variety of ancillary constructs): serial (pipelined) composition, parallel composition, infinite serial replication (closure) and infinite parallel replication (called index splitter, as the input is split between the replicas according to an "index" contained in data records). We will show that this small nomenclature of tools is sufficient to construct an arbitrary streaming network.
- Record subtyping. Data streams consist of flat records, whose fields are drawn from a linear hierarchy of array subtypes[Sha02, SS04]. The records as wholes are subtyped since the boxes accept records with extra fields and allow the producer to supply fewer variants than the consumer has the ability to recognise.
- Flow inheritance. Due to subtyping, the boxes may receive more fields in a record than they recognise. In such circumstances flow inheritance causes the extra fields to be saved and then appended to all output records produced in response to a given input one[1]. Flow inheritance enables very flexible pipelining since, on the one hand, a component does not need to be aware of the exact composition of data records that it receives as long as it receives sufficient fields for the processing it is supposed to do; and on the other, the extra data are not lost but passed further down the pipeline that the components may be connected by.
- Record synchronizers. These are similar to I-structures known from dataflow programming. SNet synchronisers are typed SISO boxes that expect two records of certain types and produce a joint record. No other synchronisation mechanism exists in SNet, and no synchronisation capability is required of the user-defined boxes.
- The concept of network feedback in the form of a closure operator. This connects replicas of a box in a (conceptually) infinite chain, with the input

[1] This is a conceptual view; in practice the data fields are routed directly to their consumers, thanks to the complete inferability of type in SNet.

data flowing to the head of the chain and the output data being extracted on the basis of fixed-point recognition. The main innovation here is the proposal of a type-defined fixed point (using flow inheritance as a statically recognisable mechanism), and the provision of an efficient type-inference algorithm. As a result, SNet has no named channels (in fact, no explicit channels at all) and the whole network can be defined as a single expression in a certain combinator algebra.

The talk will address the following issues. We will first give an overview of stream processing pointing out the history of early advances [Kah74, AW77, HCRP91], the semantic theory [BS01] and the recent languages [Mic02]. Then the concepts of SNet will be introduced, focusing in turn on: overall organisation and combinators, type system and inference algorithms, concurrency and synchronisation, and the binding for a box language. Finally a sketch of a complete application in the area of plasma simulation using the particle-in-cell method will be provided.

Work is currently underway to implement SNet as a coordination language for a large EU-sponsored Integrated Project named "EATHER" [Pro], which is part of the Framework VI Advanced Computing Architecture Initiative. University of Hertfordshire is coordinating the software side of the project; if time permits, the talk will touch upon the progress achieved to date.

References

[AW77] E. A. Ashcroft and W. W. Wadge. Lucid, a nonprocedural language with iteration. *Communications of the ACM*, 20(7):519–526, 1977.

[BS01] M Broy and G Stefanescu. The algebra of stream processing functions. *Theoretical Computer Science*, (258):99–129, 2001.

[GC92] D Gelernter and N Carriero. Coordination languages and their significance. *Communications of the ACM*, 35(2):96–107, Feb. 1992.

[Gel85] David Gelernter. Generative communication in linda. *ACM Trans Program. Lang Syst.*, 1(7):80–112, 1985.

[HCRP91] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The synchronous data-flow programming language LUSTRE. *Proceedings of the IEEE*, 79(9):1305–1320, September 1991.

[JP05] Jean-Marie Jacquet and Gian Pietro Picco, editors. *Coordination Models and Languages. 7th International Conference, COORDINATION 2005, Namur, Belgium, April 20-23, 2005*, volume 3454 of *Proceedings Series: Lecture Notes in Computer Science, Vol. 3454 Jacquet, Jean-Marie; Picco, Gian Pietro (Eds.) 2005, X, 299 p., Softcover Lecture Notes in Computer Science*. Springer Verlag, 2005.

[Kah74] G Kahn. The semantics of a simple language for parallel programming. In L Rosenfeld, editor, *Information Processing 74, Proc. IFIP Congress 74. August 5-10, Stockholm, Sweden*, pages 471–475. North-Holland, 1974.

[Mic02] Michael I. Gordon *et al.* A stream compiler for communication-exposed architectures. In *Proceedings of the Tenth International Conference on Architectural Support for Programming Languages and Operating Systems, San Jose, CA. October 2002*, 2002.

[PA98] G A Papadopoulos and F Arbab. Coordination models and languages. In *Advances in Computers*, volume 46. Academic Press, 1998.

[Pro] The AETHER Project. http://aetherist.free.fr/Joomla/index.php.

[Sha02] Alex Shafarenko. Coercion as homomorphism: type inference in a system with subtyping and overloading. In *PPDP '02: Proceedings of the 4th ACM SIGPLAN international conference on Principles and practice of declarative programming*, pages 14–25, 2002.

[Sha03] Alex Shafarenko. Stream processing on the grid: an array stream transforming language. In *SNPD*, pages 268–276, 2003.

[Sha06] Alex Shafarenko. Snet: definition and the main algorithms. Technical report, Department of Computer Science, 2006.

[SS04] Alex Shafarenko and Sven-Bodo Scholz. General homomorphic overloading. In *Implemntation and Application of Functional Languages. 16th International Workshop, IFL 2004, Lübeck, Germany, September 2004. Revised Selected Papers.*, LNCS'3474, pages 195–210. Springer Verlag, 2004.