S-Net: High-level coordination for the many-core era

Grelck, C.; Scholz, S.-B.; Shafarenko, A.

Publication date
2010
Document Version
Author accepted manuscript
Published in
Intel European Research and Innovation Conference (ERIC'10), Braunschweig, Germany

S-Net: High-Level Coordination for the Many-Core Era

Clemens Grelck[1], Sven-Bodo Scholz[2], Alex Shafarenko[2]

[1] University of Amsterdam, Amsterdam, Netherlands
[2] University of Hertfordshire, Hatfield, United Kingdom

C.Grelck@uva.nl

Summary

S-Net is a coordination language that aims at reconciling the legacy world of sequential software and programming techniques with the hardware reality in the many-core era. This is achieved through a near-complete separation of conventional application programming from concurrency-related concerns. Sequential legacy code as well as new code written in a conventional concurrency-agnostic style is made fit for efficient execution on modern many-core systems through a high-level coordination layer based on stream processing.

1 S-Net in a Nutshell

In the many-core era software must be parallel. Period. This is first and foremost a software engineering problem. Most existing software is sequential, and very few programmers are proficient in parallel programming. In contrast, all modern hardware is parallel and the degree of parallelism grows exponentially. It hardly seems feasible to adapt legacy software using conventional parallel programming techniques, which heavily intertwine algorithmic behaviour with the mere organisation of concurrent execution including problem decomposition, communication and synchronisation.

S-Net [1] is a novel coordination language whose design thoroughly avoids intertwining computational and organisational aspects. Instead, S-Net aims at a complete separation of concern between *application engineering* (i.e. writing or reusing sequential application building blocks) and *concurrency engineering* (i.e. composing them to a parallel application). Consequently, we leave the specification of basic components to conventional languages and let S-Net focus on the interfacing of sequential components and their orderly collaborative interaction.

S-Net views a piece of legacy code as a *box* that is connected to the outside world solely by two typed streams: an input stream and an output stream. A box is characterised by a stream transformer function that maps a single data item from the input stream to a (possibly empty) stream of data items on the output stream. To facilitate dynamic reconfiguration, a box has no internal state and no access to external state. The single-input/single-output (SISO) design of S-Net is essential for the separation of concerns. Only with this restriction can the (legacy) implementation of a box completely abstract from its location in a network of boxes. Multiple input streams would immediately raise the question of synchronisation while multiple output streams would raise the issue of routing.

It is up to the specifically trained *concurrency engineer* to orchestrate a number of boxes to form a parallel application. Thanks to the SISO restriction, streaming networks can be described by algebraic formulae. *Network combinators* take operand boxes or networks and define another SISO network. Thus, streaming networks of asynchronous components are defined inductively. We have identified a total of four network combinators that prove sufficient to construct a large number of network prototypes:static serial and parallel composition of heterogeneous components as well as dynamic serial and parallel replication of homogeneous components.

Efficient implementations automatically map S-Nets to symmetric multithreaded architectures [2] as well as to distributed address space clusters thereof [3].

Publications

[1] C. Grelck, S.B. Scholz, and A. Shafarenko, "Asynchronous Stream Processing with S-Net", *Int. Journal of Parallel Programming* ***38(2)*** (2010).

[2] C. Grelck and F. Penczek, "Implementation Architecture and Multithreaded Runtime System of S-Net", 20th International Symposium on Implementation and Application of Functional Languages, Revised Selected Papers, LNCS, Springer (2010).

[3] C. Grelck, J. Julku, and F. Penczek, "Distributed S-Net", Workshop on Advances in Message Passing , ACM Digital Library (2010).