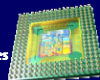
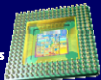


Novel Computational & Execution Models – Declarative Languages and Transactions

Ian Watson & Mikel Lujan
School of Computer Science
University of Manchester
UK

Supported by EPSRC Grant EP/E036368/1



Motivation

Usual arguments about the problem of programming multi-core

Transactions may be an answer to removing the complexity of locks

But most proposals are still in the context of threaded conventional languages

- Threads still need to be explicit
- Threads are imperative with state meaning that exploitation of anything other than the explicit thread level parallelism is hard

Work in the 80s & 90s showed how declarative languages could lead to simple parallel execution (also current use of map/reduce)

But acceptance was hindered by lack of shared state

The Addition of Transactions

Microsoft work on Haskell plus the STM Monad have shown how shared state can be incorporated into a declarative language in a way which maintains much of the declarative flavour but results in a much more expressive language.

This is at a language level, we need to consider the implications for an underlying parallel computation model.

We also need to see how this may influence the underlying hardware architecture.

Haskell is a lazy language and the current implementations of concurrent Haskell are built on top of a lazy abstract machine. Strict languages (although less 'pure') will probably lead to greater efficiency.

Potential Benefits

Explicit parallelism can be added to declarative languages without compromising their properties

But implicit parallelism can also be easily exploited across a wide range of granularity

‘General Purpose’ declarative models still have a requirement for globally shared memory but the coherence requirements can be relaxed significantly.

The addition of transactions places more requirements on the memory system, but does not require fine grain coherence.

Major benefits for scalable memory systems?

Starting Points

Significant amount of work done in the 80s & 90s on reduction machine architectures for the implementation of declarative languages.

Manchester has much experience in this area – from Dataflow through Flagship to the European Declarative System (EDS)

Have been developing both hardware and software Transactional Memory systems.

The hardware work has been aimed specifically at systems with relaxed requirements for coherence.

Also previous language work, notably a major contribution to the SISAL project and an experimental language called UFO.

Some Warnings/Reservations

Note that shared memory is essential for general purpose programming

Don't be seduced by distributed message passing solutions – Erlang is an interesting language, with particular application in certain domains, but **NOT** the answer to general purpose multi-core programming.

We need to be pragmatic

- Don't insist on too much purity
- We need languages and features that people can understand!

But maybe not too pragmatic – a language such as F# has probably already gone too far to make parallel execution easy.