ARCHITECTURE-TRANSPARENT CONTROL OF PARALLELISM

HARDWARE PARALLELISM CAUSES SOFTWARE CRISIS
We critically rely on increasingly complex and computationally demanding software. Exploiting parallelism is crucial for performance, since modern hardware is massively parallel, heterogeneous, and hierarchical [1].

- Parallel programming is harder than sequential due to complexity of parallelism control
- Mainstream languages put the burden of parallelism control on the programmer, reducing productivity
- Non-deterministic race conditions and deadlocks are exceptionally difficult to detect and correct
- Rapid hardware evolution makes manual code adaptation infeasible
- Heterogeneity requires adaptation and leads to proliferation of different programming models

We need a unified programming model that achieves high performance portability and productivity by adapting to evolving parallel architectures [2]. High-level declarative approaches to structured parallel programming appears most promising.

PARALLEL FUNCTIONAL PROGRAMMING TO THE RESCUE
Functional programming expresses computations in terms of functions that map inputs to outputs and is founded on the lambda calculus, offering the following advantages [2]:

- High level of abstraction increases productivity: programmers declare the algorithm and identify parallelism whilst run-time system manages parallel execution
- Less manual architecture-specific code optimisations improve portability, whilst the run-time system adapts the execution to target architectures for performance
- No race conditions and deadlocks due to the deterministic programming model

The challenge is design and implementation of an extensible, automatic and architecture-transparent parallel execution across heterogeneous and hierarchical parallel architectures to achieve high performance portability.

DESKTOP OF AN ADAPTIVE RUN-TIME SYSTEM
Main goals:
- Implement distributed graph reduction using virtual shared memory view
- Flexibility to adapt to heterogeneous architectures for high performance portability
- Extensibility through modular design (similar to Microkernel and Multikernel approaches)
- Support for decentralised cost-model-based policy control and online profiling

We demonstrate how architectural trends lead to the need for performance portability
We present and justify a declarative high-level semi-implicit approach to parallel programming that is based on adaptive architecture-transparent parallelism control and balances productivity and performance portability
We enhanced the GUM run-time profiling module to monitor per processor activity profiles including thread counts and allocation, among other statistics
Ongoing work is aimed at empirical evaluation of the benefits of cost-model-based architecture-transparent control of parallelism that uses architecture information and system information to adapt to heterogeneous target architectures

FUTURE WORK
- Application classification using the profiling information to identify further relevant cost model parameters, alongside the number and speed of processors, cache and RAM sizes, network latency, memory bandwidth and system load
- Integration of a cost model into the GUM run-time system and improving over the available heuristics that control parallelism
- Empirically evaluating the new run-time system using a set of benchmarks on heterogeneous architectures by measuring run times and calculating speedups on a Beowulf cluster and on Xeon Phi and comparing to baseline run-time system and mainstream languages and programming models such as C-MPI/OpenMP

References

Acknowledgements
This is joint work with Dr Hans-Wolfgang Loidl, Prof Dr Greg Michaelson, Malak Aljabri (all Heriot-Watt University) and Prof Dr Kevin Hammond (University of St Andrews) and is supported by SICSA. The author is grateful to the members of the Dependable Systems Group and to Julian Geddes for inspiring discussions. Design by Sins Robinsen Daves.

PRELIMINARY RESULTS
- We extend the GUM parallel Haskell run-time system [3] with additional online profiling such as thread numbers differentiated by state, spark numbers, and allocation, among others
- We have implemented parallel sparse matrix multiplication to complement the parfib benchmark suite, which we will use for evaluation

The challenge is design and implementation of an extensible, automatic and architecture-transparent parallel execution across heterogeneous and hierarchical parallel architectures to achieve high performance portability.