The Design and Implementation of Scalable Parallel Haskell

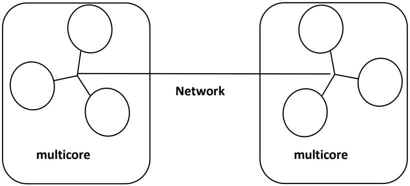
Malak Aljabri, Phil Trinder,and Hans-Wolfgang Loidl

MMnet'13: Language and Runtime Support for Concurrent Systems
Heriot Watt University

May 8, 2013

Parallel Architectures

- Parallel architectures are increasingly multi-level e.g. clusters of multicores.
- A hybrid parallel programming model is often used to exploit parallelism across the cluster of multicores e.g. using MPI + OpenMP.
- Managing two abstractions is a burden for the programmer and increases the cost of porting to a new platform.



Glasgow Parallel Haskell (GpH) Implementations

- Identify parallelism, do not control it.
- Parallelism is supported internally by the language implementation.

Two main GpH implementations:

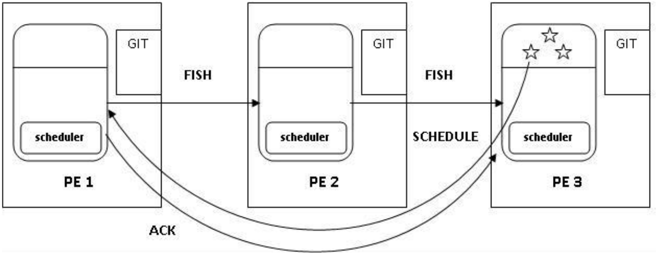
1. GHC-SMP: shared memory.
2. GHC-GUM: distributed memory.

- Both implementations use different but related runtime environment (RTE) mechanisms.
- Good performance results can be achieved on shared memory architectures and on networks individually, but a combination of both, for clusters of multi-cores, is lacking.

Work Distribution in GHC-GUM

Existing Load Balancing:

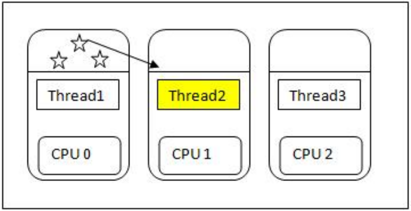
1. Searching for Local Work.
2. Searching for Remote Work.



Work Distribution in GHC-SMP

Existing Load Balancing:

- Spark pools are implemented as bounded work-stealing queues.
- A work-stealing queue is a lock-free data structure.
- The owner can push and pop from one end of the queue without synchronization.
- Other threads can steal from the other end of the queue.



GUMSMP

- A multilevel parallel Haskell implementation for clusters of multicores.
- Integrates the advantages of the two GpH implementations.
- Provides improvements for **automatic load balancing**.
- **The main potential benefits of GUMSMP are:**
  - Providing a scalable model.
  - Efficient exploitation of the the specifics of distributed and shared memory on different levels of the hierarchy.
  - Providing a single high-level programming model.

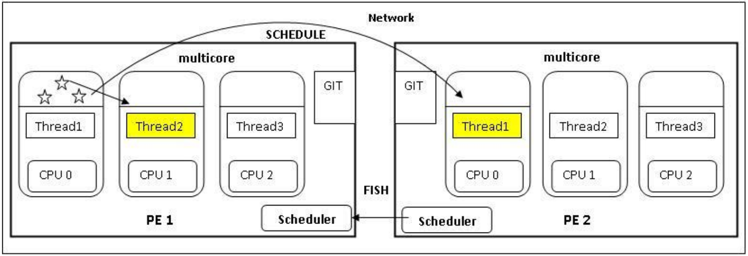
GUMSMP Design Overview

- **Memory Management:** the same virtual shared heap as GHC-GUM.
- **Communication:** the same mechanism implemented in GHC-GUM.
- **Load Balancing:** the combination of GHC-SMP and GHC-GUM mechanisms.

GUMSMP Work Distribution Mechanism

New Load Balancing:

- Work distribution of GUMSMP is hierarchy aware.
- It uses a work-stealing algorithm, through sending FISH message, on networks (inherited from GHC-GUM).
- Within a multicore it will search for a spark by directly accessing spark pools (inherited from GHC-SMP).



GUMSMP Design Objectives

- **Even but asymmetric load balancing** Important to maintain even load distribution, but *accept imbalances* as the communication cost increases.
- **Mostly passive load distribution** Essential to maintain *passive load distribution*, but switch to active in some cases e.g high-watermark.
- **Effective latency hiding** The system must be designed so that communication cost is not in the critical path of cooperating computations.

GUMSMP

Current Implementation

- We achieved the basic functionality on a limited number of PEs.
- With 3 PEs, 5 cores each we have:

Sparks \ PEs	PE 1	PE2	PE3
Global sparks	1	1	2
Local Sparks	13	10	16

Conclusion

- The design of the new multi-level parallel Haskell implementation GUMSMP is presented.
- Designed for high-performance computation on networks of multi-cores.
- The design focuses on flexible work distribution policies.
  - Even but asymmetric load balancing.
  - Mostly passive load distribution.
  - Effective latency hiding.
- The main benefits:
  - Scalable model.
  - Efficient exploitation of distributed and shared memory on different levels of the hierarchy.
  - Single programming model.

GUMSMP

Thanks for Listening ..