The Design of GUMSMP: a Multilevel Parallel Haskell Implementation

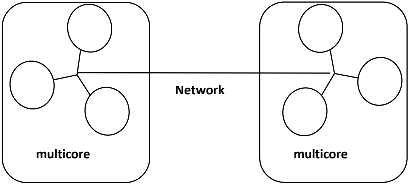
Malak Aljabri, Phil Trinder,and Hans-Wolfgang Loidl

Heriot Watt University

September 14, 2012

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.
- The Main Goal: Providing efficient control of hierarchical architectures using GpH.



GpH(Glasgow Parallel Haskell)

- Semi-explicit parallel Haskell.
- Parallelism is expressed by two primitives added to the Haskell program: par and pseq.
- Example:

```
parfib :: Int -> Int
parfib n | n <= 1    = 1
         | otherwise = runEval $ do
                         x <- rpar (parfib (n-1))
                         y <- rseq (parfib (n-2))
                         return (x + y)
```

GpH(Glasgow Parallel Haskell)

- **Evaluation strategies:** polymorphic and higher order functions controlling parallelism.
- Potentially add extensions to refine placement e.g *parBound*.
- Tow main implementations :
  - GHC-SMP - shared memory.
  - GHC-GUM - distributed memory.

GUMSMP

- A multilevel parallel Haskell implementation for clusters of multicores.

GUMSMP

- A multilevel parallel Haskell implementation for clusters of multicores.
- Integrates the advantages of the distributed memory *GHC-GUM* and the shared memory *GHC-SMP* implementations.

GUMSMP

- A multilevel parallel Haskell implementation for clusters of multicores.
- Integrates the advantages of the distributed memory *GHC-GUM* and the shared memory *GHC-SMP* implementations.
- Provides improvements for **automatic load balancing**.

GUMSMP

- A multilevel parallel Haskell implementation for clusters of multicores.
- Integrates the advantages of the distributed memory *GHC-GUM* and the shared memory *GHC-SMP* implementations.
- Provides improvements for **automatic load balancing**.
- **The main potential benefits of GUMSMP are:**
    - Provides a scalable model.

GUMSMP

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

GUMSMP

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

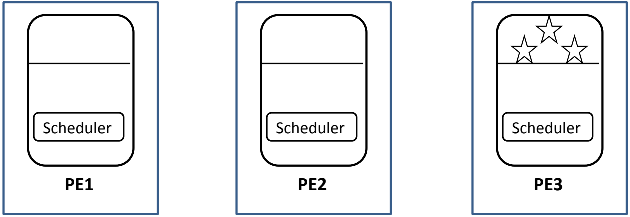
GUMSMP Design Overview

- **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(using the hierarchy-aware policy).

Work Distribution in GHC-GUM

Load Balancing:

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

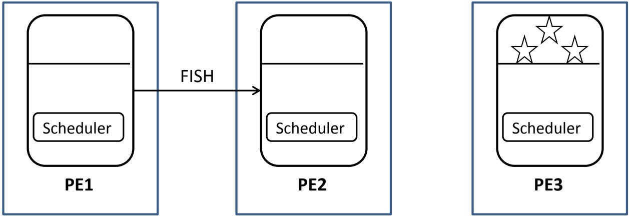


PE1 needs work

Work Distribution in GHC-GUM

Load Balancing:

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

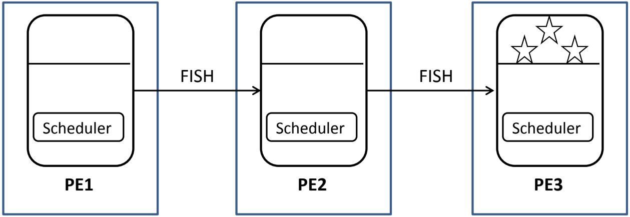


PE1 sends fish message

Work Distribution in GHC-GUM

Load Balancing:

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

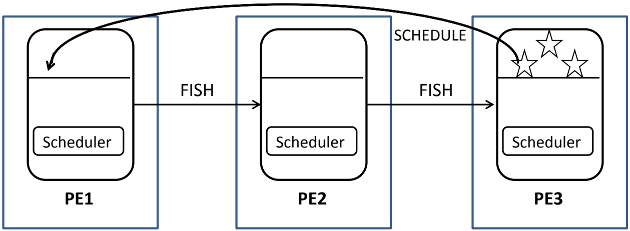


PE2 forwards the message

Work Distribution in GHC-GUM

Load Balancing:

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

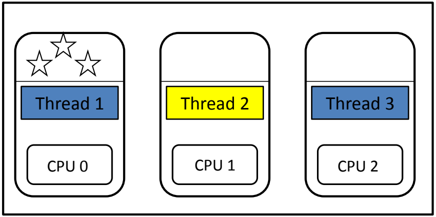


PE3 sends work to PE1

Work Distribution in GHC-SMP

Load Balancing:

- Processor's Spark Pool is implemented as a bounded work-stealing queue.
- 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.

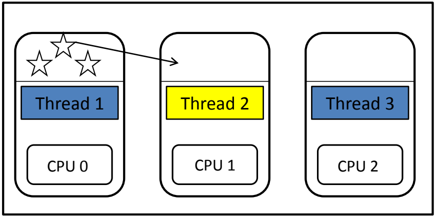


PE1 creates 'spark thread' to get work

Work Distribution in GHC-SMP

Load Balancing:

- Processor's Spark Pool is implemented as a bounded work-stealing queue.
- 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.

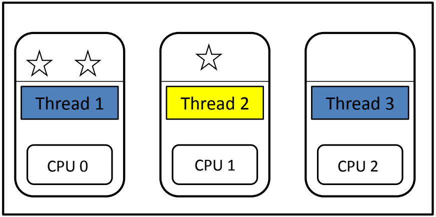


The 'spark thread' steals spark

Work Distribution in GHC-SMP

Load Balancing:

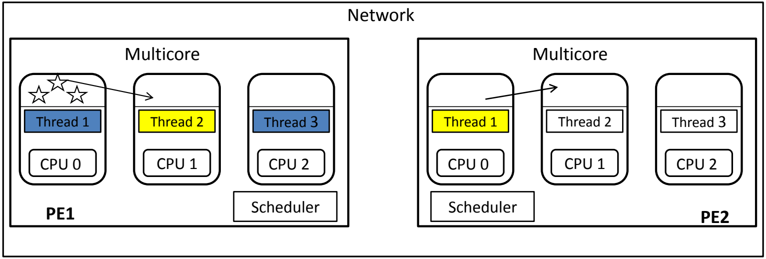
- Processor's Spark Pool is implemented as a bounded work-stealing queue.
- The owner can push and pop from one end of the queue without synchronization.
- Other threads can steal from the other end of the queue.



Evaluates the spark

GUMSMP Work Distribution Mechanism

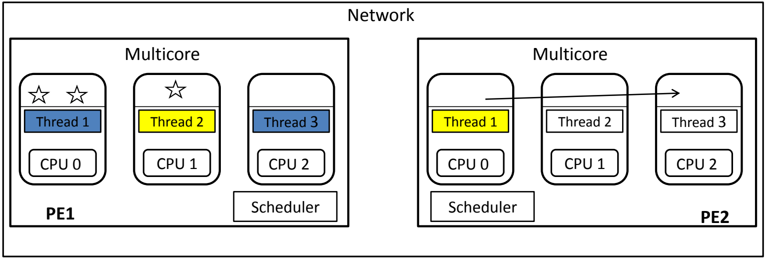
- Work distribution of GUMSMP is hierarchy aware.
- It uses a work-stealing algorithm, through sending FISH message, on networks (inherited from GHC-GUM).
- Within a multicore it will search for a spark by directly accessing spark pools (inherited from GHC-SMP).



PE2 has no local work

GUMSMP Work Distribution Mechanism

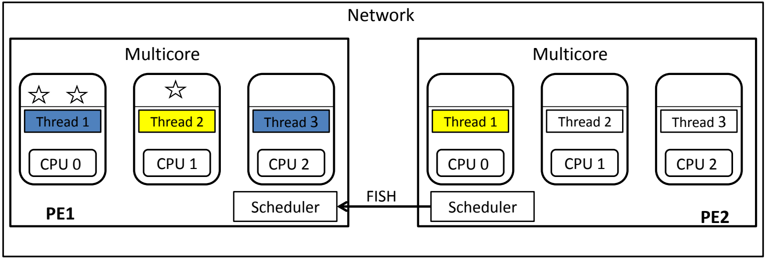
- Work distribution of GUMSMP is hierarchy aware.
- It uses a work-stealing algorithm, through sending FISH message, on networks (inherited from GHC-GUM).
- Within a multicore it will search for a spark by directly accessing spark pools (inherited from GHC-SMP).



PE2 has no local work

GUMSMP Work Distribution Mechanism

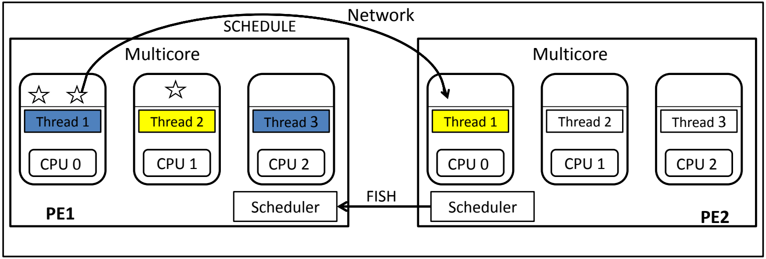
- Work distribution of GUMSMP is hierarchy aware.
- It uses a work-stealing algorithm, through sending FISH message, on networks (inherited from GHC-GUM).
- Within a multicore it will search for a spark by directly accessing spark pools (inherited from GHC-SMP).



PE2 sends Fish message

GUMSMP Work Distribution Mechanism

- 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).



PE1 replay with spark

GUMSMP Design Objectives

- **Hierarchy aware load balancing** Important to maintain even load distribution, but accept imbalances as the communication cost increases.

GUMSMP Design Objectives

- **Hierarchy aware 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.

GUMSMP Design Objectives

- **Hierarchy aware 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 Design Decisions

Spark Placement: where to place a spark, that has been imported from another processor ?

GUMSMP Design Decisions

Spark Placement: where to place a spark, that has been imported from another processor ?

1. Assign it to the spark pool of the first idle processor.
   (+) Keep utilization high.
   (-) Lead to higher fragmentation.

GUMSMP Design Decisions

Spark Placement: where to place a spark, that has been imported from another processor ?

1. Assign it to the spark pool of the first idle processor.
   (+) Keep utilization high.
   (-) Lead to higher fragmentation.
2. **Separate spark pool, dedicated to imported sparks**.
   (+) Keep related piece of work together.
   (+) Useful in some situation e.g no idle processors any more.
   (-) Requires additional stealing step.

GUMSMP Design Decisions

Fishing: when to send a spark requesting message to a remote PE ?

GUMSMP Design Decisions

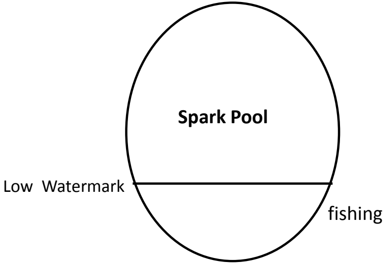
Fishing: when to send a spark requesting message to a remote PE ?

1. Send a message when the PE is idle.
   (-) Might not be idle any more.

GUMSMP Design Decisions

Fishing: when to send a spark requesting message to a remote PE ?

1. Send a message when the PE is idle.
   (-) Might not be idle any more.
2. **Low-Watermark mechanism**.



GUMSMP Design Decisions

Work-offloading: How to process the received work-requesting message ?

GUMSMP Design Decisions

Work-offloading: How to process the received work-requesting message ?

1. Select spark from the processor with largest spark pool.
   (-) Impose additional overheads.
2. **Random**.

Ongoing Work

- The shared memory component of the hybrid system shows performance within 7% of the original GHC-SMP implementation.
- Complete the implementation of the enhanced work distribution policy.
- Assess the quality of the enhanced work distribution policy on hierarchical architectures.

Conclusion

- The design of the new multi-level parallel Haskell implementation GUMSMPis presented.
- Designed for high-performance computation on multilevel architectures e.g. networks of multi-cores.
- The design focuses on flexible work distribution policies.
  - Hierarchy aware 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.

Thank You..