Message Passing Co-Processor

Stefan Kästle
stefan.kaestle@inf.ethz.ch
Systems @ ETH Zurich
Motivation

• Future machines
  – No shared memory
  – No cache-coherence

• Explicit sharing
  – Message passing
Motivation II

• Message passing is a bottleneck
  – Kernel entry
  – Interrupt costs

=> Does not scale with number of channels
Our contribution

• The message passing co-processor (mcp)
  – message passing in **hardware**, one per core?
  – **Offload** message-passing overhead
  – application cores can do useful work

• Describe the **minimal interface** to mcp

• **Prototype** of message passing hardware
  – investigate protocol design space
Part 1:

DESIGN
Minimal interface

• Every application needs own context on mcp
  – Protection!
  – Context mapped to its virtual address space

• Signaling channels
  – Separate channels for each direction, no locks
Minimal interface II

- Protocol to signal:
  - Creation of **new domains**
    - To create per-application state on mcp
  - Creation of **new channels**
    - Lookup of routing information for local channel id
  - **Context-switches**
    - mcp needs to context-switch as well
    - Signal beginning and end of context-switch
Transparent hardware device

• Protocol between mcps heavily depending on:
  – Architecture
  – Interconnect

• Protocol is transparent to user-level apps
  – Exception: flow-control?!

• mcp acts according to interface
Interconnect protocol

• However, we believe we need a reservation based system
  – Due to networking issues
    • starvation, contention, head-of-the line blocking, ..
Networking issues

Interconnect
Per-core buffers

Messages to c2

User-level buffers
Per-channel

c1

c2 (malicious)

c3

Stefan Kästle <stefan.kaestle@inf.ethz.ch>
Networking issues

Messages cannot be sent although receiver has free buffer space

Stefan Kästle <stefan.kaestle@inf.ethz.ch>
Part 2:

IMPLEMENTATION ON THE SCC
Intel SCC Overview

- 48 cores (32Bit x86)
- Groups of 2 cores called **tile**
Intel SCC Overview

• Cores on tile share **message passing buffers**
  
  – 16 KiB
  – Efficient communication between cores on tile
  – Mapped into virtual address space of other cores
mcp implementation on SCC

- Every other core as dedicated mcp
- MPB holds:
  - Per-application buffers (UL)
    - Incoming/outgoing messages to application
  - Inter-core buffers (IC)
    - Messages from other mcps
    - Each mcp has separate buffer -> no locking
  - Signaling channels

<table>
<thead>
<tr>
<th>UL in</th>
<th>UL out</th>
<th>Signaling, ...</th>
<th>IC (4 KiB)</th>
</tr>
</thead>
</table>

Stefan Kästle <stefan.kaestle@inf.ethz.ch>
mcp implementation on SCC II

• Reservation: RTS/CTS based
  – Ready-To-Send (RTS): Sender has message to send
  – Clear-To-Send (CTS): Receiver accepts message
  – Easy to implement
  – Does not require a lot of state in the hardware
  – Optimizations possible (piggy-backing, reservation of several slots .. )
RTS/CTS based protocol

Stefan Kästle <stefan.kaestle@inf.ethz.ch>
RTS/CTS based protocol
Questions?
mcp state

• Each mcp requires state information
  – Channel lookup table: 1 entry per channel
  – RTS counter: 1 per channel
  – CTS storage: 1 Bit per channel
  – Backup memory to context-switch mcps: per-app

Does not scale, limit number of channels per core
RTS/CTS based protocol
RTS/CTS based protocol
RTS/CTS based protocol
RTS/CTS based protocol
RTS/CTS based protocol
Backup 1

• What makes message-passing so expensive:
  – Polling in case of many channels
    • Implement select() in hardware
  – For complex interconnects: flow control?
  – Notifications, e.g. Interrupts
    • Different schemes: only one interrupt for a message burst and only if app is not currently running
    • According to app priorities?
  – Verification of messages, e.g. system call
    • Channel Lookup on mcp