Running Erlang on the Parallella

Magnus Lång\textsuperscript{1} Kostis Sagonas\textsuperscript{1,2}

\textsuperscript{1}Department of Information Technology  
Uppsala University

\textsuperscript{2}School of Electrical and Computer Engineering  
National Technical University of Athens

\textit{Erlang User Conference, 2015}
The Parallella

- An ARM-based single board computer
- Main feature: the Epiphany co-processor

Figure: Adapteva’s Parallella

\(^1\)Image copyright Gareth Halfacree
The Epiphany

- 32-bit general-purpose RISC
- Many, very simple cores
  - No out-of-order execution
  - No caches
- Each core has 32KB of SRAM
- Network on Chip: Memory space is divided in $64 \times 64$ 1MB sections

Figure: The Epiphany Architecture
Our goal

- Build a modified ERTS that will run Erlang code on co-processors
- Processes are explicitly spawned on the Epiphany
  - Current limitation: one process per core
- Run existing code with minimal modification
- Possible use cases
  - Lower power consumption of Erlang workloads
  - Reserve processor throughput for Erlang processes
Why run Erlang on the Epiphany?

- Experiment with Erlang on low-power devices
  - 64-core Epiphany edition has more FLOPS/W than best contemporary GPU (NVIDIA Kepler)
  - Pity to use such devices only via low-level languages
- Erlang is a natural fit!
  - Concurrent programming model
  - Distribution
  - Fault tolerance
Programmung model

1> P1 = erlang:spawn(...).
2> link(P1).
3> P1 ! self().
4> receive pong -> ok end.
5> exit(P1, plz).

1> P2 = epiphany:spawn(...).
2> link(P2).
3> P2 ! self().
4> receive pong -> ok end.
5> exit(P2, plz).

Figure: Code for the Epiphany works like any other Erlang

Imposed limitations change how programs should be structured:

- Number of processes
- Amount of memory
Q: Do I just change `spawn(X)` into `epiphany:spawn(X)`?
   A: Sometimes that is sufficient, sometimes not

The limitations need to be considered
   For process count: Use an arbitrator
High-level structure

- Master-slave structure
- Both are built on the same code

Figure: System overview
High-level structure

- Master-slave structure
- Both are built on the same code

**Figure: VM closeup**
Synchronous and asynchronous communication

- Syscalls: GC, some built-in functions
- Message buffers: messages, memory management, etc...

**Figure**: Master-slave communication
- Code is not loaded into both systems automatically
- `code_server` makes sure they run the same version

**Figure:** Loading of threaded code
Performance gotchas

- Syscalls will never be fast
- Rule of thumb: “Does it access any global state?”
- `atom ! Message is a syscall`
What next?

- **Performance**
  - We need to fit code & data in SRAM
  - The solution is called HiPE
  - In the future, you will need to HiPE-compile your hotpath
Summary

Current status

- A modified Erlang Runtime System that runs Erlang on the Epiphany co-processor
- Runs existing Erlang code with minimal modification