Library Support for Resource Constrained Accelerators

Brock-Nannestad, Laust; Karlsson, Sven

Publication date:
2014

Document Version
Publisher's PDF, also known as Version of record

Library Support for Resource Constrained Accelerators

Laust Brock-Nannestad and Sven Karlsson
Technical University of Denmark

Motivation

- Accelerators provide attractive power/performance trade offs
- Existing programming models treat them as offload devices
- We want accelerator cores as **first class citizens**
  - Execute your application directly from the accelerator
  - Spawn parallelism **from** accelerator
  - But retain convenience of a complete system: libraries, file I/O, ...

Contributions

- A new runtime for accelerators
  - Accelerator drives execution – `main()` on accelerator
  - **Low overhead** – moving C library to host keeps runtime small
- Implementation on x86-64 and ARM + Epiphany (Parallella)
- Evaluation using SPLASH-2; comparison to libgomp
- Measure impact on object code size by minimizing runtime

Design

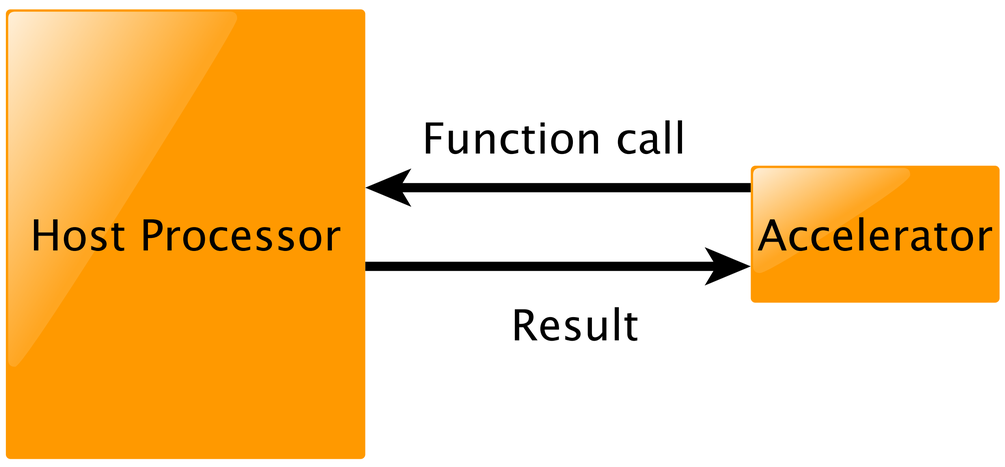

Figure : System architecture

- `main()`/master thread executes on accelerator
- Decisions on spawning parallelism taken **locally**
  - fast, low latency
- Accelerator **may** invoke host
  - Complex decisions or function calls (C library)
- Host assists accelerator, not vice versa

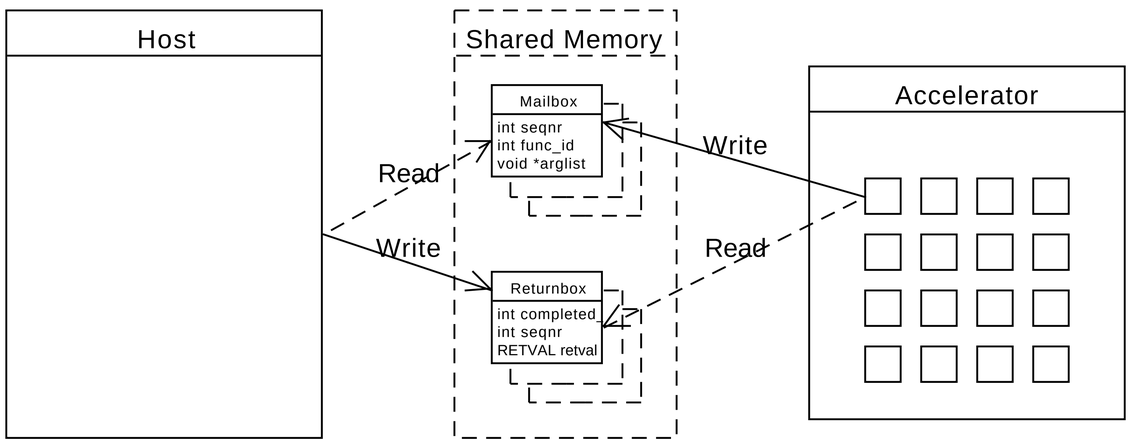

Figure : Mailboxes in shared memory

- Host and accelerator communicate through shared memory
- Mailbox data structures used for communication
- Encode function call identity, arguments, and results

Acknowledgments

This work was funded by the European Artemis project nr. 295440, Portable and Predictable Performance on Heterogeneous Manycores (PaPP)

Experimental setups

- 1) Dual Socket Xeon 5570 @ 2.93 GHz, Linux 3.2, GCC 4.8.1, glibc 2.17
- 2) Adapteva Parallella w. 16 core Epiphany accelerator
- SPLASH-2 benchmarks LU, FFT, C-runtime library offloaded to host
- Baseline is SPLASH-2 parallelized with GNU libgomp for GCC 4.8.1

Execution time (x86-64)

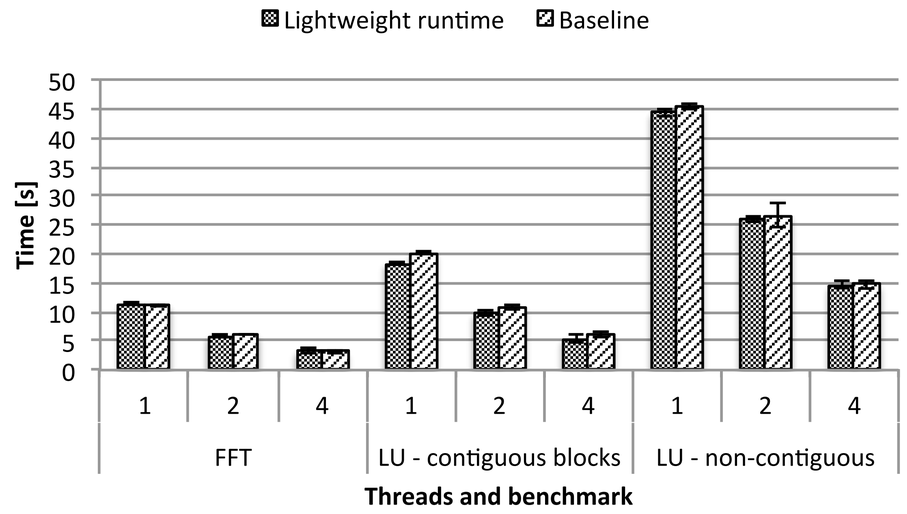

Figure : Execution time and speed up for increasing number of cores on **x86-64**.

Object code sizes and savings

C library function (sym)	Size (bytes)	FFT	LU
`__printf_fp`	9345	X	X
`__sin_sse2`	8569	X	
`__cos_sse2`	5159	X	
`_getopt_internal_r`	4391	X	X
`_int_malloc`	4776	X	X
`_int_free`	2796		
`___memset_sse2`	2705		X
`malloc`	333	X	X
(the rest)		271	565
Total		**35044**	**22115**

Accelerator runtime object code size

Accelerator component	x86-64 (bytes)	Epiphany (bytes)
Host interface	209	96
Locks, Barriers	159	178
Misc	130	228
Bootstrap/Initialization	202	138
Thread creation	49	0
Total	**749**	**640**

Conclusions

- Object code reduction of 66-75% on benchmarks by offloading C library
- Low runtime footprint of 750 bytes (x86-64), 640 bytes (Epiphany)
- Computational code not influenced by remote function call overhead – Performance comparable to GNU libgomp
- Initialization is single threaded. FFT initialization dominated by remote function call overhead (see paper for details)
- Future work
  - Parallella/Epiphany implementation soon to be released
  - Careful memory management – software caching techniques

Further information

Brock-Nannestad, L. and Karlsson, S. "Library Support for Resource Constrained Accelerators." In *Improving OpenMP for Devices, Task, and More*. Springer, 2014, pp.187-201.