HermitCore – A Low Noise Unikernel for Extrem-Scale Systems

Stefan Lankes[1], Simon Pickartz[1], Jens Breitbart[2]

[1]RWTH Aachen University, Aachen, Germany

[2]Bosch Chassis Systems Control, Stuttgart, Germany

E.ON Energy Research Center | RWTH AACHEN UNIVERSITY

Agenda

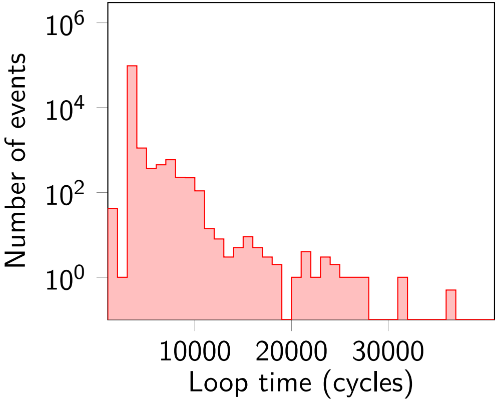
- Motivation
- Challenges for the Future Systems
- OS Architectures
- HermitCore Design
- Performance Evaluation
- Conclusion and Outlook

Motivation

- Complexity of high-end HPC systems keeps growing
  - Extreme degree of parallelism
  - Heterogeneous core architectures
  - Deep memory hierarchy
  - Power constrains
    - $\Rightarrow$ Need for scalable, reliable performance and capability to rapidly adapt to new HW
- Applications have also become complex
  - In-situ analysis, workflows
  - Sophisticated monitoring and tools support, etc. . .
  - Isolated, consistent simulation performance
    - $\Rightarrow$ Dependence on POSIX, MPI and OpenMP
- Seemingly contradictory requirements. . .

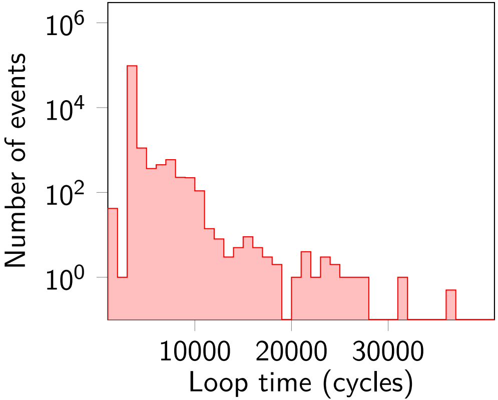
Operating System = Overhead

- Every administration layer has its overhead ⇒ e. g. Hourglass benchmark



Operating System = Overhead

- Every administration layer has its overhead ⇒ e. g. Hourglass benchmark



- How does the HPC / Cloud Community reduce overhead?

How does the HPC community reduce the overhead?

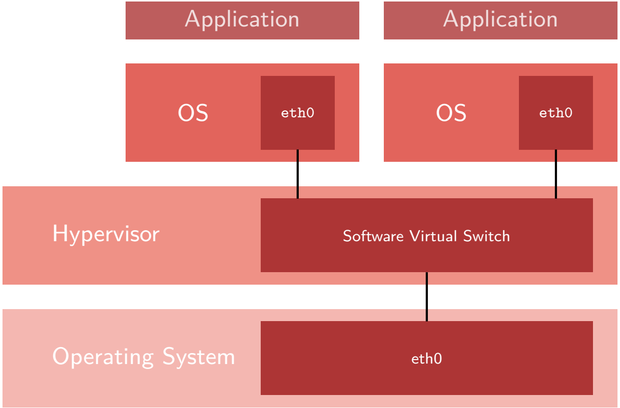
Light-weight Kernels

- Typically taken of an existing fat kernel (e. g. Linux)
- Removalof unneeded features to improve the scalability
- e. g. ZeptoOS

Multi-Kernels

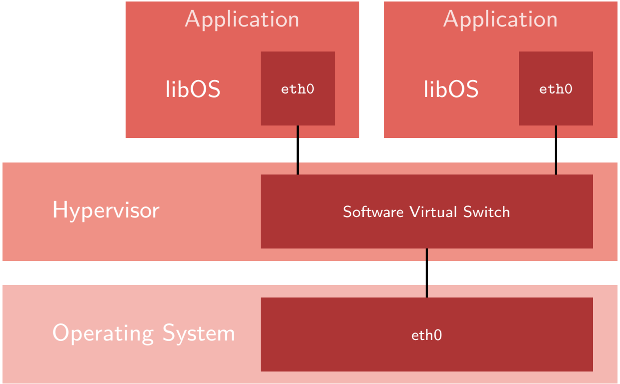
- A specialized kernel runs side-by-side to full-weight kernel (e. g. Linux)
- Applications of the full-weight kernels are able to run on the specialized kernel.
- The specialized kernel catch every system call and delegate them to the full-weight kernel
  - Binary compatible to the full-weight kernel
- Examples: mOS, McKernel

OS Designs for Cloud Computing – Usage of Common OS



- Two operating systems to maintain a single computer?
- Double Management!
- Why does a Cloud base on a Multi-User-/Multi-Tasking OS?

OS Designs for Cloud Computing – LibraryOS



- Now, every system call is a function call ⇒ Low overhead
- Whole optimization of an image possible (including the library OS)
  - Link Time Optimization (LTO)
- Removing unneeded code ⇒ reduces the attack vector

Related Unikernels

- Rump kernels[1]
  - Part of NetBSD ⇒ e. g., NetBSD's TCP / IP stack is available as library
  - Strong dependencies to the hypervisor
  - Not directly bootable on a standard hypervisor (e. g., KVM)
- IncludeOS[2]
  - Runs natively on the hardware ⇒ minimal Overhead
  - Only 32 bit support to avoid the overhead of paging ⇒ uncommon in HPC
- MirageOS[3]
  - Designed for the high-level language OCaml ⇒ uncommon in HPC

[1] A. Kantee and J. Cormack. "Rump Kernels – No OS? No Problem!". In: ; login: 2014.
[2] A. Bratterud et al. "IncludeOS: A Resource Efficient Unikernel for Cloud Services". In: 7th Int. Conference on Cloud Computing Technology and Science. 2015.
[3] A. Madhavapeddy et al. "Unikernels: Library Operating Systems for the Cloud". In: 8th Int. Conference on Architectural Support for Programming Languages and Operating Systems. 2013.

HermitCore Design

- Combination of the Unikernel and Multi-Kernel to reduce the overhead
  - The same binary is able to run
    - in a VM (classical unikernel setup)
    - or bare-metal side-by-side to Linux (multi-kernel setup)
- Support for dominant programming models (MPI, OpenMP)
- Single-address space operating system
  - No TLB Shootdown

HermitCore Design

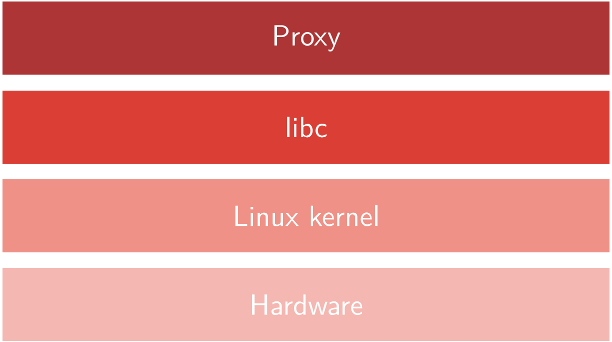
Classical Unikernel Setup

- Applications are able to boot directly within a VM
  - Tested with Qemu / KVM
  - Tested with uhyve (experimental KVM-based Hypervisor)
    - Qemu emulates more than HermitCore needs ⇒ large setup time
    - uhyve reduce the boot time (from 2 s to ~30 ms)
  - Amazon Web Services (available soon!)

Multi-Kernel Setup

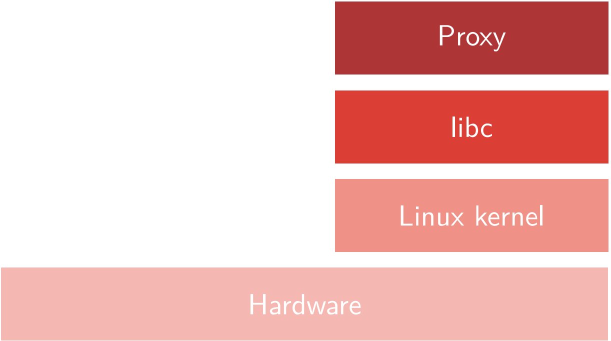
- One kernel per NUMA node
  - Only local memory accesses (UMA)
  - Message passing between NUMA nodes
- One FWK (Linux) in the system to get access to a broader driver support
  - Only a backup for pre- / post-processing
  - Critical path should be handled by HermitCore

Booting HermitCore



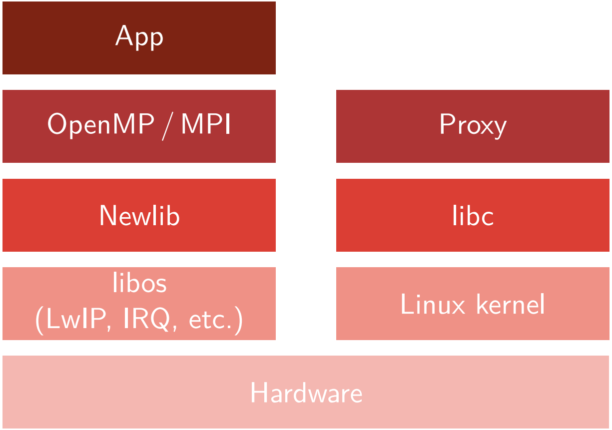
- On the detection of a HermitCore app, a proxy will be started.

Booting HermitCore



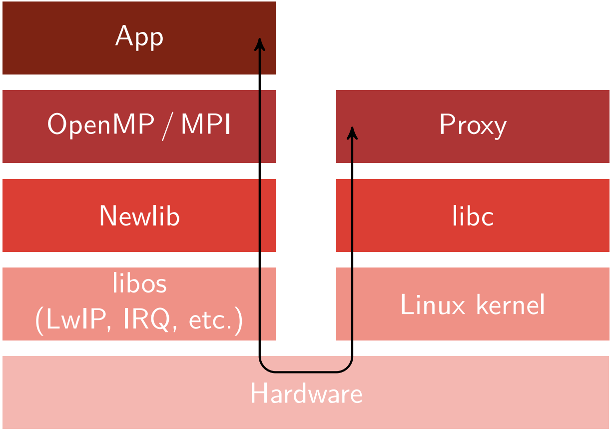
- On the detection of a HermitCore app, a proxy will be started.
- The proxy unplugs a set of cores.

Booting HermitCore



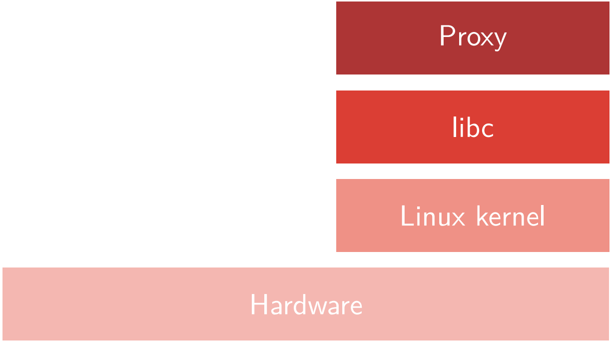
- On the detection of a HermitCore app, a proxy will be started.
- The proxy unplugs a set of cores.
- Triggers Linux to boot HermitCore on the unused cores.

Booting HermitCore



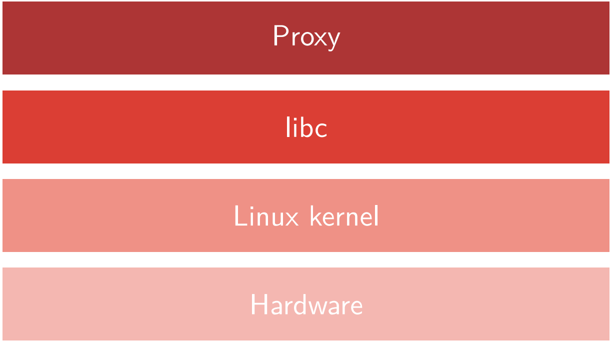
- On the detection of a HermitCore app, a proxy will be started.
- The proxy unplugs a set of cores.
- Triggers Linux to boot HermitCore on the unused cores.
- A reliable connection will be established.

Booting HermitCore



- On the detection of a HermitCore app, a proxy will be started.
- The proxy unplugs a set of cores.
- Triggers Linux to boot HermitCore on the unused cores.
- A reliable connection will be established.
- By termination, the cores are set to the HALT state.

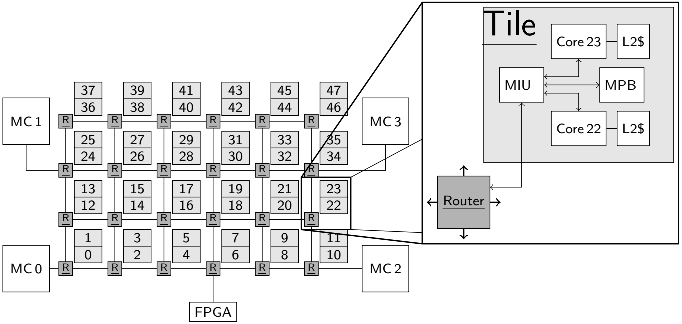
Booting HermitCore



- On the detection of a HermitCore app, a proxy will be started.
- The proxy unplugs a set of cores.
- Triggers Linux to boot HermitCore on the unused cores.
- A reliable connection will be established.
- By termination, the cores are set to the HALT state.
- Finally, reregistering of the cores to Linux.

Runtime Support

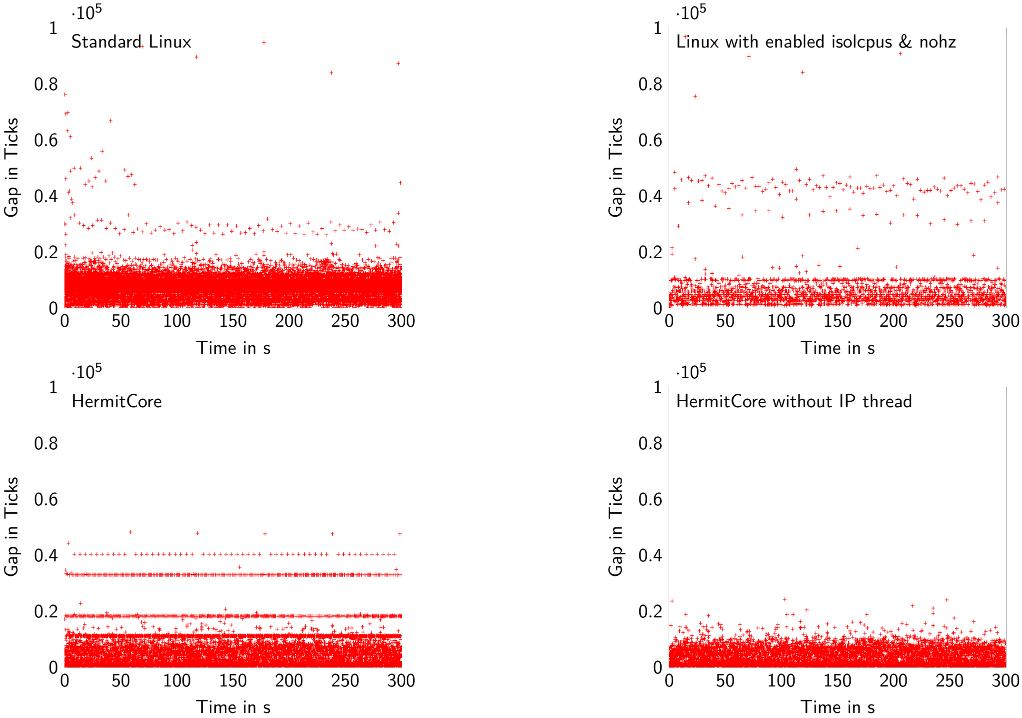
- SSE, AVX2, AVX512, FMA,. . .
- Full C-library support (newlib)
- HBM support similar to memkind
- IP interface & BSD sockets (LwIP)
  - IP packets are forwarded to Linux
  - Shared memory interface
- Pthreads
  - Thread binding at start time
  - No load balancing ⇒ less housekeeping
- OpenMP via Intel's Runtime
- iRCCE- & MPI (via SCC-MPICH)
- Full support for the Go runtime



Operating System Micro-Benchmarks

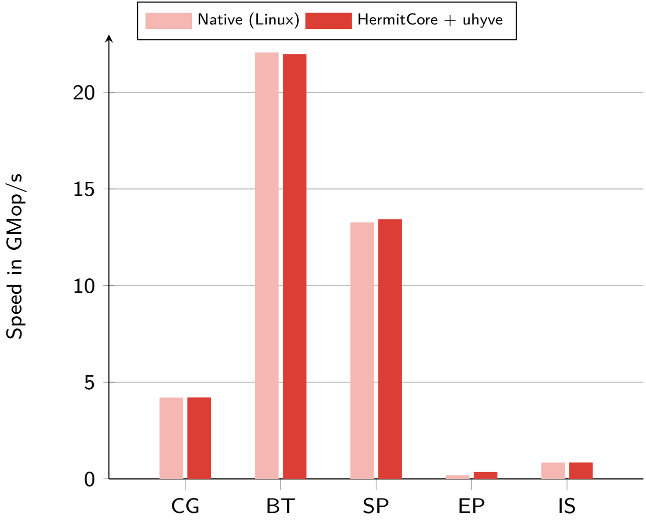
- Test systems
  - Intel Haswell CPUs (E5-2650 v3) clocked at 2.3 GHz
  - Intel KNL (Phi 7210) clocked at 1.3 GHz, SNC mode with four NUMA nodes
- Results in CPU cycles

System activity	**KNL**		**Haswell**	
	HermitCore	Linux	HermitCore	Linux
`getpid()`	15	486	14	143
`sched_yield()`	197	983	97	370
`malloc()`	3 051	12 806	3715	6575
first write access to a page	2 078	3 967	2018	4007



·10^5
Standard Linux
Gap in Ticks
Time in s
·10^5
Linux with enabled isolcpus & nohz
Gap in Ticks
Time in s
·10^5
HermitCore
Gap in Ticks
Time in s
·10^5
HermitCore without IP thread
Gap in Ticks
Time in s

Overhead of VMs – Determined via NAS Parallel Benchmarks (Class B)



E.ON Energy Research Center

RWTH AACHEN UNIVERSITY

Conclusions

- It works! $\Rightarrow$ `https://youtu.be/gDYCJ1DOTKw`
- Binary packages are available
- Reduce the OS noise significantly
- Try it out!

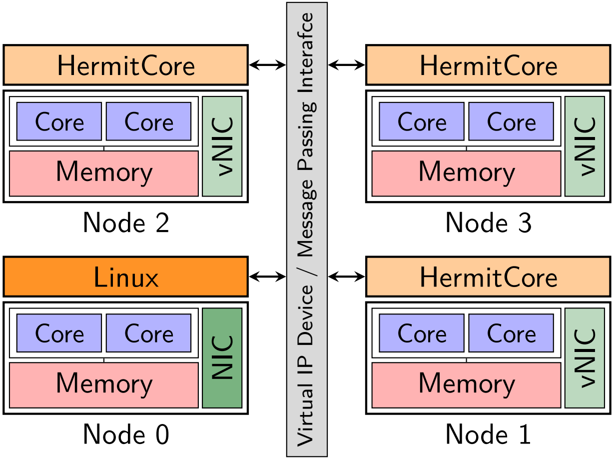
`http://www.hermitcore.org`

Thank you for your kind attention!

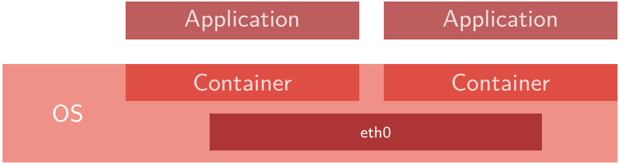
Backup slides

Outlook

- A fast direct access to the interconnect is required
- SR-IOV simplifies the coordination between Linux & HermitCore

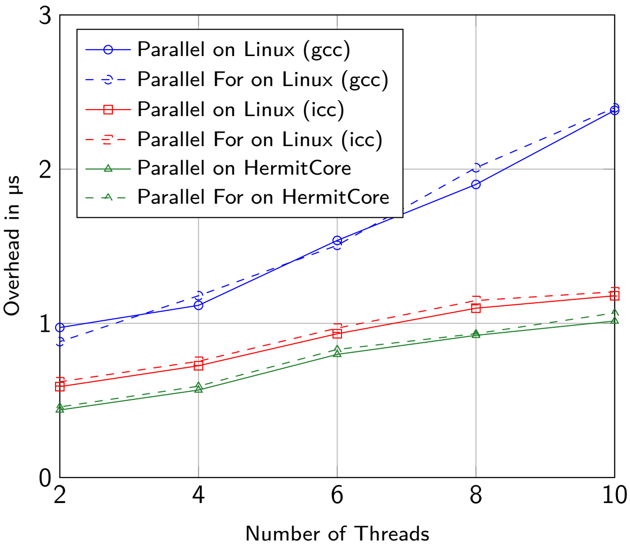


OS Designs for Cloud Computing – Container

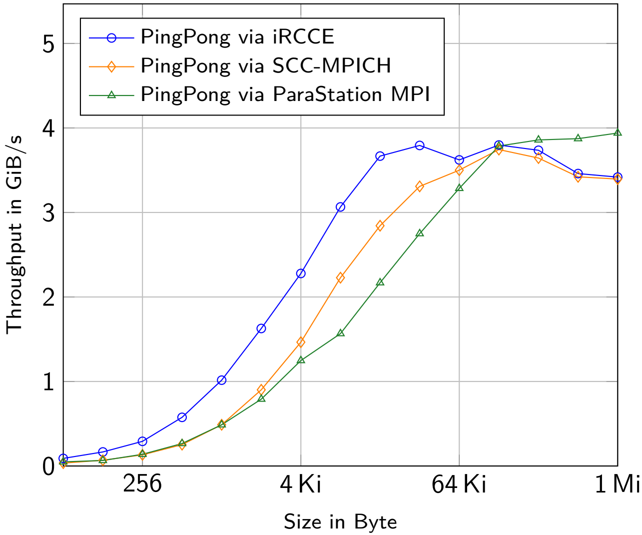


- Building of virtual borders (namespaces)
- Containers and their processes doesn't see each other
- Fast access to OS services
- Less secure because an exploit for the container attacks also the host OS
- Doesn't reduce the OS noise of the host system

EPCC OpenMP Micro-Benchmarks



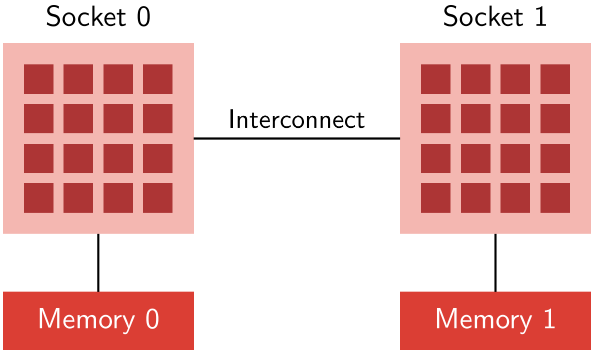
Throughput Results of the Inter-kernel Communication Layer



Lack of programmability

Non-Uniform Memory Access

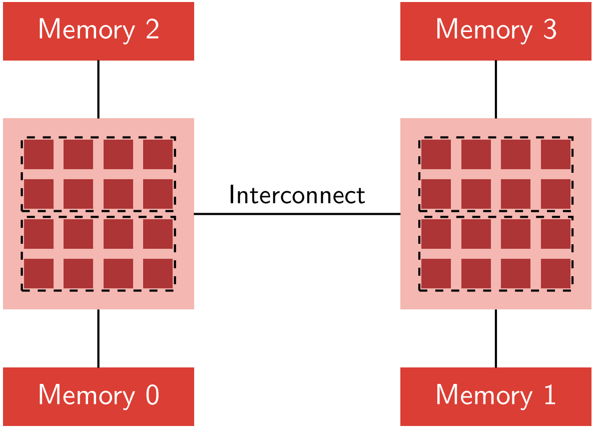
- Costs for memory access may vary
- Run processes where memory is allocated
- Allocate memory where the process resides
- Implications for the performance
  - Where should the applications store the data?
  - Who should decide the location?
    - The operating system?
    - The application developers?



Lack of programmability

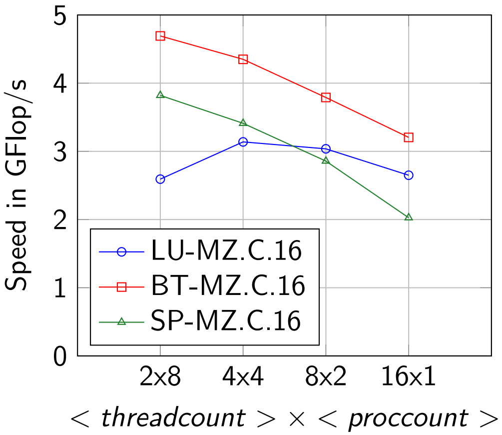
Non-Uniform Memory Access

- Costs for memory access may vary
- Run processes where memory is allocated
- Allocate memory where the process resides
- Implications for the performance
  - Where should the applications store the data?
  - Who should decide the location?
    - The operating system?
    - The application developers?



Tuning Tricks

- Parallelization via Shared Memory (OpenMP)
  - Many side-effects and error-prone
  - Incremental parallelization
- Parallelization via Message Passing (MPI)
  - Restructuring of the sequential code
  - Less side-effects
- Performance Tuning
  - Bind MPI applications on one NUMA node
  - ⇒ No remote memory access



OpenMP Runtime

- GCC includes a OpenMP Runtime (libgomp)
  - Reuse synchronization primitives of the Pthread library
  - Other OpenMP runtimes scales better
  - In addition, our Pthread library was originally not designed for HPC
- Integration of Intel's OpenMP Runtime
  - Include its own synchronization primitives
  - Binary compatible to GCC's OpenMP Runtime
  - Changes for the HermitCore support are small
    - Mostly deactivation of function to define the thread affinity
  - Transparent usage
    - For the end-user, no changes in the build process

Support of compilers beside GCC

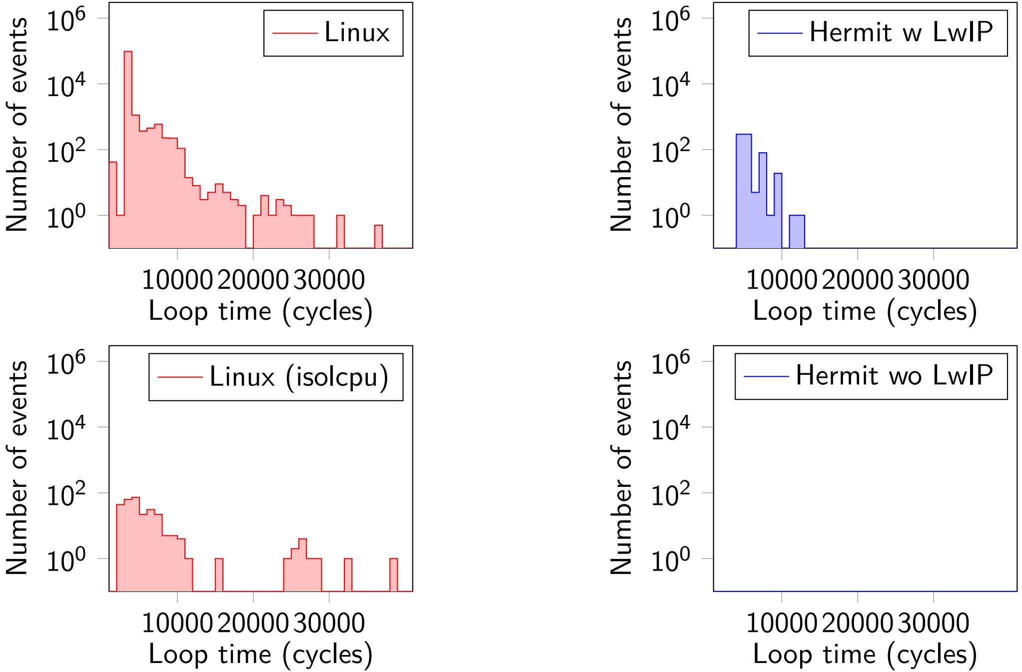
- Just avoid the standard environment (−ffreestanding)
- Set include path to HermitCore's toolchain
- Be sure that the ELF file use HermitCore's ABI
  - Patching object files via elfedit
- Use the GCC to link the binary

```
LD = x86_64-hermit-gcc
#CC = x86_64-hermit-gcc
#CFLAGS = -O3 -mtune=native -march=native -fopenmp -mno-red-zone
CC = icc -D__hermit__
CFLAGS = -O3 -xHost -mno-red-zone -ffreestanding -I$(HERMIT_DIR) -openmp
ELFEDIT = x86_64-hermit-elfedit

stream.o: stream.c
        $(CC) $(CFLAGS) -c -o $@ $<
        $(ELFEDIT) --output-osabi HermitCore $@

stream: stream.o
        $(LD) -o $@ $< $(LDFLAGS) $(CFLAGS)
```

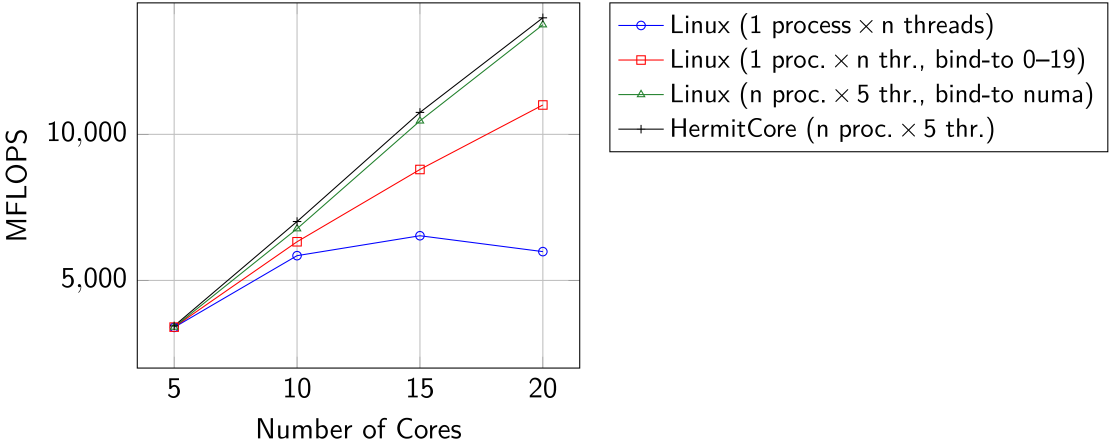
Is the software stack difficult to maintain?

- Changes to the common software stack determined with `cloc`

Software Stack	LoC	Changes
binutils	5 121 217	226
gcc	6 850 382	4 821
Linux	15 276 013	1 296
Newlib	1 040 826	5 472
LwIP	38 883	832
Pthread	13 768	466
OpenMP RT	61 594	324
HermitCore	–	10 597


Linux
Hermit w LwIP
Linux (isolcpu)
Hermit wo LwIP
Number of events
Loop time (cycles)
10^0
10^2
10^4
10^6
10000
20000
30000

Hydro (preliminary results)



Thank you for your kind attention!

Stefan Lankes et al. – slankes@eonerc.rwth-aachen.de

Institute for Automation of Complex Power Systems
E.ON Energy Research Center, RWTH Aachen University
Mathieustraße 10
52074 Aachen

www.acs.eonerc.rwth-aachen.de