A CASE FOR TRANSFORMING PARALLEL RUNTIMES INTO OS KERNELS

Kyle Hale

Peter Dinda

halek.co
v3vee.org
presciencelab.org
xstack.sandia.gov/hobbes
HOBBES
xstack.sandia.gov/hobbes

Palacios
An OS Independent Embeddable VMM
v3vee.org/palacios
THE CURRENT OS/RUNTIME MODEL

**User mode**
- Parallel app
- Parallel runtime

**Kernel mode**
- General-purpose kernel
- Node HW
THIS MODEL HAS SOME ISSUES
ARE PROVIDED KERNEL ABSTRACTIONS THE RIGHT ONES?

user mode

runtime

I’d like to pin memory to a specific PFN range please

kernel mode

general OS

 NOT ALWAYS
ARE PROVIDED KERNEL ABSTRACTIONS THE RIGHT ONES?

user mode
runtime

kernel mode
general OS

I’d like to never be interrupted please
NOPE

NOT ALWAYS
I’d like to set up some custom page mappings please

Uh no

user mode
runtime

kernel mode
general OS
RESTRICTED ACCESS TO HARDWARE

user mode

runtime

I’d like to interrupt another processor please

kernel mode

general OS

HA!
What are the consequences?
What are the consequences?

WORKAROUNDS & COMPROMISES
What are the consequences?

WORKAROUNDS & COMPROMISES

DUPLICATED FUNCTIONALITY
If runtime had

we could mitigate these issues
If runtime had FULL HARDWARE ACCESS, we could mitigate these issues.
If runtime had

FULL HARDWARE ACCESS

CONTROL OVER KERNEL ABSTRACTIONS

we could mitigate these issues
THE CURRENT OS/RUNTIME MODEL

- **user mode**
  - parallel app
  - parallel runtime

- **kernel mode**
  - general-purpose kernel
  - node HW
OUR PROPOSED MODEL:
THE HYBRID RUNTIME (HRT)

user mode

kernel mode

parallel app

hybrid runtime

node HW
OUR PROPOSED MODEL: THE HYBRID RUNTIME (HRT)

user mode

kernel mode

parallel app

hybrid runtime

node HW

Mashup of OS and runtime
OUR PROPOSED MODEL: THE HYBRID RUNTIME (HRT)

user mode

kernel mode

The runtime **IS** the kernel, built within a kernel framework

parallel app

hybrid runtime

node HW
OUR PROPOSED MODEL: 
THE HYBRID RUNTIME (HRT)

The runtime IS the kernel, built within a kernel framework. Everything is in kernel space.
OUR PROPOSED MODEL: THE HYBRID RUNTIME (HRT)

The runtime **IS** the kernel, built within a kernel framework

Everything is in **kernel space**

HRT has **full** access to the hardware
OUR PROPOSED MODEL: THE HYBRID RUNTIME (HRT)

**user mode**

**kernel mode**

- parallel app
- node HW

**HRT can control HW access**

**HRT can pick its own abstractions**
OUR PROPOSED MODEL:
THE HYBRID RUNTIME (HRT)

user mode

kernel mode

parallel app

HRT

MORE POWER!
We built a kernel framework to support HRTs
We built a kernel framework to support HRTs
We built a kernel framework to support HRTs

We ported an existing, complex parallel runtime
We built a kernel framework to support HRTs

We ported an existing, complex parallel runtime
We built a kernel framework to support HRTs

We ported an existing, complex parallel runtime

We ported our framework to cutting-edge many-core hardware
We built a kernel framework to support HRTs

We ported an existing, complex parallel runtime

We ported our framework to cutting-edge many-core hardware
We built a kernel framework to support HRTs

We ported an existing, complex parallel runtime

We ported our framework to cutting-edge many-core hardware

We evaluated our port on a standard HPC benchmark
We built a kernel framework to support HRTs

We ported an existing, complex parallel runtime

We ported our framework to cutting-edge many-core hardware

We evaluated our port on a standard HPC benchmark
XEON PHI + NAUTILUS + LEGION + HPCG

Speedup over Linux

Legion Processor Count (Cores)
XEON PHI + NAUTILUS + LEGION + HPCG
XEON PHI + NAUTILUS + LEGION + HPCG

11% average speedup
NAUTILUS

user mode

kernel mode

parallel app

runtime

threads  sync.  paging  events  HW info  bootstrap  timers  IRQs  console

Hardware
NAUTILUS

user mode

kernel mode

parallel app

runtime

Hardware

threads sync. paging events HW info bootstrap timers IRQs console

Nautilus primitives & utilities (HRT can use or not use any of them)
Nautilus primitives & utilities (HRT can use or not use any of them)
NAUTILUS

user mode

---

kernel mode

parallel app

runtime

threads  sync.  paging  events  HW info  bootstrap  timers  IRQs  console

Hardware

HRT
NAUTILUS

user mode

kernel mode

parallel app

runtime

threads sync. paging events HW info bootstrap timersIRQs console

Hardware

Kernel
MINIMAL LIGHTWEIGHT PRIMITIVES

FULL HARDWARE ACCESS

VERY FAST BOOT TIMES
LIGHTWEIGHT PRIMITIVES

EXAMPLE: THREADS

x86_64 Opteron: 64 cores, 4 sockets, 8 numa zones, 128GB RAM
LIGHTWEIGHT PRIMITIVES

EXAMPLE: THREADS

x86_64 Opteron: 64 cores, 4 sockets, 8 numa zones, 128GB RAM
LIGHTWEIGHT PRIMITIVES

EXAMPLE: THREADS

x86_64 Opteron: 64 cores, 4 sockets, 8 numa zones, 128GB RAM
LIGHTWEIGHT PRIMITIVES

EXAMPLE: THREADS

x86_64 Opteron: 64 cores, 4 sockets, 8 numa zones, 128GB RAM
FULL HARDWARE CONTROL

EXAMPLE: INTERRUPT CONTROL
very simple modification: give runtime control over interrupts in its task scheduler
very simple modification: give runtime control over interrupts in its task scheduler

→ modest speedups
FULL HARDWARE CONTROL

EXAMPLE: INTERRUPT CONTROL

very simple modification: give runtime control over interrupts in its task scheduler

→ modest speedups

MUCH more to come here
in addition to Legion, we have 2 other high-level, parallel runtimes running as HRTs

**NESL**: VCODE interpreter running as HRT

**NDPC**: home-grown, *co-designed* HRT
INTEGRATING THE HRT WITH A LEGACY OS
THE HYBRID VIRTUAL MACHINE
Regular OS (ROS)

- parallel app
- parallel runtime
- general OS

Legacy functionality from the Regular OS via the HVM

Hybrid Virtual Machine (HVM)

- hybrid runtime
- specialized virtualization model
LINUX FORK + EXEC  ~ 714µs

HVM + HRT CORE BOOT  ~ 61µs
LINUX FORK + EXEC $\sim 714\mu s$

HRT boot is CHEAP!

HVM + HRT CORE BOOT $\sim 61\mu s$
<table>
<thead>
<tr>
<th>RANK</th>
<th>SITE</th>
<th>SYSTEM</th>
<th>CORES</th>
<th>RMAX (TFLOP/S)</th>
<th>RPEAK (TFLOP/S)</th>
<th>POWER (KW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>National Super Computer Center in Guangzhou, China</td>
<td>Tianhe-2 (MilkyWay-2) - TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P NUDT</td>
<td>3,120,000</td>
<td>33,862.7</td>
<td>54,902.4</td>
<td>17,808</td>
</tr>
<tr>
<td>7</td>
<td>Texas Advanced Computing Center/Univ. of Texas, United States</td>
<td>Stampede - PowerEdge C8220, Xeon E5-2680 8C 2.700GHz, Infiniband FDR, Intel Xeon Phi SE10P Dell</td>
<td>462,462</td>
<td>5,168.1</td>
<td>8,520.1</td>
<td>4,510</td>
</tr>
<tr>
<td>18</td>
<td>DOE/SC/Pacific Northwest National Laboratory, United States</td>
<td>cascade - Atipa Visione IF442 Blade Server, Xeon E5-2670 8C 2.600GHz, Infiniband FDR, Intel Xeon Phi 5110P Hewlett-Packard</td>
<td>194,616</td>
<td>2,539.1</td>
<td>3,388.0</td>
<td>1,384</td>
</tr>
<tr>
<td>33</td>
<td>Purdue University, United States</td>
<td>Conte - Cluster Platform SE250s Gen8, Xeon E5-2670 8C 2.600GHz, Infiniband FDR, Intel Xeon Phi 5110P Hewlett-Packard</td>
<td>77,520</td>
<td>776.8</td>
<td>1,341.1</td>
<td>310</td>
</tr>
<tr>
<td>64</td>
<td>Tulip Trading, Australia</td>
<td>C01N - SuperBlade SBI-7127RG-E, Intel Xeon E5-2695v2 12C 2.4GHz, Infiniband FDR, Intel Xeon Phi 7120P Supermicro</td>
<td>160,600</td>
<td>798.3</td>
<td>3,164.5</td>
<td>619</td>
</tr>
<tr>
<td>69</td>
<td>Intel, United States</td>
<td>Endeavor - Intel Cluster, Intel Xeon E5-2697v2 12C 2.700GHz, Infiniband FDR, Intel Xeon Phi 7140</td>
<td>51,392</td>
<td>758.9</td>
<td>933.5</td>
<td>387.2</td>
</tr>
</tbody>
</table>
[root@v-test-t620 nautilus]# philix -d -b weever -k nautilus.bin
XEON PHI + NAUTILUS + LEGION + HPCG

11% average speedup
A CASE FOR TRANSFORMING PARALLEL RUNTIMES INTO OS KERNELS

my website halek.co

our development blog haltloop.com

our lab presciencelab.org

the Hobbes project xstack.sandia.gov/hobbes

follow us here for:
- experience report on building OS for Phi
- philix release (soon)

Kyle Hale  Peter Dinda
BACKUPS
FULL HARDWARE CONTROL

EXAMPLE: INTERRUPT CONTROL

Speedup

Legion Processors (threads)
FULL HARDWARE CONTROL

EXAMPLE: INTERRUPT CONTROL

![Graph showing Speedup vs. Legion Processors (threads)]
user mode

kernel mode

parallel app

hybrid runtime

node HW
Hybrid Virtual Machine (HVM)
This is the performance path, through the HRT.
Regular OS (ROS)

User mode
- parallel app
- parallel runtime
- general OS

Kernel mode
- parallel runtime
- general OS

Legacy functionality from the Regular OS via the HVM
We can boot these things very quickly!
several auxiliary HRTs spawned in less than a millisecond
HPCG IN LEGION ON XEON PHI
HPCG IN LEGION ON XEON PHI

![Graph showing performance of Legion and Processor Counts ranging from 1 to 220 processors. The y-axis represents execution time in seconds, and the x-axis represents the Legion Processor Count (cores). Legend includes Natuibus and Linux.]
port of NESL

- nested data parallel language aimed at vector machines
port of NESL

- nested data parallel language aimed at vector machines

- we can run unmodified NESL programs in our kernel-mode VCODE interpreter
the first co-designed HRT: NDPC

- Nested Data Parallelism in C/C++
- subset of NESL
the first co-designed HRT: NDPC

- Nested Data Parallelism in C/C++
- subset of NESL
- fork/join parallelism over flattened vector processing
the first co-designed HRT: NDPC

- Nested Data Parallelism in C/C++
- subset of NESL
- fork/join parallelism over flattened vector processing
- allows us to explore runtime/kernel co-design
- e.g. smart kernel-mode thread fork
several auxiliary HRTs spawned in less than a millisecond
to get started with your own Xeon Phi prototype kernel:
to get started with your own Xeon Phi prototype kernel:

• follow our blog
• use our tool (philix) to boot it and leverage MPSS stack
to get started with your own Xeon Phi prototype kernel:

• follow our blog
• use our tool (*philix*) to boot it and leverage MPSS stack
to get started with your own Xeon Phi prototype kernel:

- follow our blog
- use our tool *(philix)* to boot it and leverage MPSS stack

find out more @ haltloop.com