Helios: Heterogeneous Multiprocessing with Satellite Kernels

Ed Nightingale, Orion Hodson, Ross McIlroy, Chris Hawblitzel, Galen Hunt
Once upon a time...

- Hardware was homogeneous
Once upon a time...

- Hardware was homogeneous
Once upon a time...

- Hardware was homogeneous
Problem: HW now heterogeneous

- Heterogeneity **ignored** by operating systems
- Standard OS abstractions are **missing**
- Programming models are **fragmented**
Solution

- Helios manages ‘distributed system in the small’
  - Simplify app development, deployment, and tuning
  - Provide single programming model for heterogeneous systems

- 4 techniques to manage heterogeneity
  - Satellite kernels: Same OS abstraction everywhere
  - Remote message passing: Transparent IPC between kernels
  - Affinity: Easily express arbitrary placement policies to OS
  - 2-phase compilation: Run apps on arbitrary devices
Results

- Helios offloads processes with **zero** code changes
  - Entire networking stack
  - Entire file system
  - Arbitrary applications

- **Improve** performance on NUMA architectures
  - Eliminate resource contention with multiple kernels
  - Eliminate remote memory accesses
• Motivation
• Helios design
  o Satellite kernels
  o Remote message passing
  o Affinity
  o Encapsulating many ISAs
• Evaluation
• Conclusion
Driver interface is poor app interface
Driver interface is poor app interface

- Hard to perform **basic** tasks: debugging, I/O, IPC
- Driver encompasses services and runtime...an OS!
Satellite kernels provide single interface

- Satellite kernels:
  - Efficiently manage local resources
  - Apps developed for single system call interface
  - μkernel: Scheduler, memory manager, namespace manager
Satellite kernels provide single interface

- **Satellite kernels:**
  - Efficiently manage local resources
  - Apps developed for single system call interface
  - \(\mu\)kernel: Scheduler, memory manager, namespace manager
Remote Message Passing

- Local IPC uses **zero-copy** message passing
- Remote IPC **transparently** marshals data
- Unmodified apps work with multiple kernels
Connecting processes and services

- Applications register in a namespace as services
- Namespace is used to connect IPC channels
- Satellite kernels register in namespace

/fs
/dev/nic0
/dev/disk0
/services/TCP
/services/PNGEater
/services/kernels/ARMv5
Where should a process execute?

- Three **constraints** impact initial placement decision
  1. Heterogeneous ISAs makes migration is difficult
  2. Fast message passing may be expected
  3. Processes might prefer a particular platform

- Helios exports an **affinity** metric to applications
  - Affinity is expressed in application metadata and acts as a hint
  - Positive represents emphasis on **communication** – zero copy IPC
  - Negative represents desire for **non-interference**
Affinity Expressed in Manifests

```
<?xml version="1.0" encoding="utf-8"?>
<application name="TcpTest" runtime="full">
  <endpoints>
    <inputPipe id="0" affinity="0"
contractName="PipeContract"/>
    <endpoint id="2" affinity="+10"
contractName="TcpContract"/>
  </endpoints>
</application>
```

- Affinity easily edited by dev, admin, or user
Affinity Expressed in Manifests

- Affinity easily edited by dev, admin, or user

```xml
<?xml version="1.0" encoding="utf-8"?>
<application name="TcpTest" runtime="full">
  <endpoints>
    <inputPipe id="0" affinity="0"
      contractName="PipeContract"/>
    <endpoint id="2" affinity="+10"
      contractName="TcpContract"/>
  </endpoints>
</application>
```
Platform Affinity

- Platform affinity processed first
- **Guarantees** certain performance characteristics

```
/services/kernels/vector-CPU
platform affinity = +2

/services/kernels/x86
platform affinity = +1
```
Platform Affinity

- Platform affinity processed first
- **Guarantees** certain performance characteristics

/services/kernels/vector-CPU platform affinity = +2
/services/kernels/x86 platform affinity = +1
Positive Affinity

- Represents ‘tight-coupling’ between processes
  - Ensure fast message passing between processes

- Positive affinities on each kernel summed

/services/TCP communication affinity = +1
/services/PNGEater communication affinity = +2
/services/antivirus communication affinity = +3
Positive Affinity

- Represents ‘tight-coupling’ between processes
  - Ensure fast message passing between processes
- Positive affinities on each kernel summed
Positive Affinity

- Represents ‘tight-coupling’ between processes
  - Ensure fast message passing between processes

- Positive affinities on each kernel summed

/services/TCP communication affinity = +1
/services/PNGEater communication affinity = +2
/services/antivirus communication affinity = +3
Negative Affinity

Expresses a preference for **non-interference**
- Used as a means of avoiding resource contention

Negative affinities on each kernel summed

- /services/kernels/x86 platform affinity = +100
- /services/antivirus non-interference affinity = -1
Negative Affinity

- Expresses a preference for **non-interference**
  - Used as a means of avoiding resource contention

- Negative affinities on each kernel summed

---

/services/kernels/x86 platform affinity = +100
/services/antivirus non-interference affinity = -1
Negative Affinity

- Expresses a preference for **non-interference**
  - Used as a means of avoiding resource contention

- Negative affinities on each kernel summed

/services/kernels/x86
platform affinity = +100

/services/antivirus
non-interference affinity = -1
Self-Reference Affinity

/services/webserver non-interference affinity = -1

- Simple scale-out policy across available processors
Self-Reference Affinity

-1

/services/webserver
non-interference affinity = -1

-1

- Simple scale-out policy across available processors
Self-Reference Affinity

/services/webserver
non-interference affinity = -1

-1

• Simple scale-out policy across available processors
Turning policies into actions

- Priority based algorithm reduces candidate kernels by:
  - First: Platform affinities
  - Second: Other positive affinities
  - Third: Negative affinities
  - Fourth: CPU utilization

- Attempt to balance simplicity and optimality
Encapsulating many architectures

- **Two-phase** compilation strategy
  - All apps first compiled to MSIL
  - At install-time, apps compiled down to available ISAs

- MSIL encapsulates *multiple versions* of a method

- Example: ARM and x86 versions of `Interlocked.CompareExchange` function
**Implementation**

- Based on Singularity operating system
  - Added satellite kernels, remote message passing, and affinity

- XScale programmable I/O card
  - 2.0 GHz ARM processor, Gig E, 256 MB of DRAM
  - Satellite kernel identical to x86 (except for ARM asm bits)
  - Roughly 7x slower than comparable x86

- NUMA support on 2-socket, dual-core AMD machine
  - 2 GHz CPU, 1 GB RAM per domain
  - Satellite kernel on each NUMA domain.
Limitations

• Satellite kernels require timer, interrupts, exceptions
  ○ Balance device support with support for basic abstractions
  ○ GPUs headed in this direction (e.g., Intel Larrabee)

• Only supports two platforms
  ○ Need new compiler support for new platforms

• Limited set of applications
  ○ Create satellite kernels out of commodity system
  ○ Access to more applications
Outline

- Motivation
- Helios design
  - Satellite kernels
  - Remote message passing
  - Affinity
  - Encapsulating many ISAs
- Evaluation
- Conclusion
Evaluation platform

XScale

A
Kernel
X86

B
Satellite Kernel
X86
Satellite Kernel
XScale

NUMA Evaluation

A
Single Kernel
X86 NUMA

B
Satellite Kernel
X86 NUMA

Satellite Kernel
X86 NUMA
Offloading Singularity applications

<table>
<thead>
<tr>
<th>Name</th>
<th>LOC</th>
<th>LOC changed</th>
<th>LOM changed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Networking stack</td>
<td>9600</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>FAT 32 FS</td>
<td>14200</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>TCP test harness</td>
<td>300</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>Disk indexer</td>
<td>900</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Network driver</td>
<td>1700</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Mail server</td>
<td>2700</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Web server</td>
<td>1850</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

- Helios applications offloaded with very little effort
Offloading Singularity applications

<table>
<thead>
<tr>
<th>Name</th>
<th>LOC</th>
<th>LOC changed</th>
<th>LOM changed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Networking stack</td>
<td>9600</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>FAT 32 FS</td>
<td>14200</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>TCP test harness</td>
<td>300</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>Disk indexer</td>
<td>900</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Network driver</td>
<td>1700</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Mail server</td>
<td>2700</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Web server</td>
<td>1850</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

- Helios applications offloaded with **very little** effort
Offloading Singularity applications

<table>
<thead>
<tr>
<th>Name</th>
<th>LOC</th>
<th>LOC changed</th>
<th>LOM changed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Networking stack</td>
<td>9600</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>FAT 32 FS</td>
<td>14200</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>TCP test harness</td>
<td>300</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>Disk indexer</td>
<td>900</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td><strong>Network driver</strong></td>
<td>1700</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Mail server</td>
<td>2700</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Web server</td>
<td>1850</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

- Helios applications offloaded with very little effort
Offloading Singularity applications

<table>
<thead>
<tr>
<th>Name</th>
<th>LOC</th>
<th>LOC changed</th>
<th>LOM changed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Networking stack</td>
<td>9600</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>FAT 32 FS</td>
<td>14200</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>TCP test harness</td>
<td>300</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>Disk indexer</td>
<td>900</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Network driver</td>
<td>1700</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Mail server</td>
<td>2700</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Web server</td>
<td>1850</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

- Helios applications offloaded with **very little** effort
Netstack offload

- Offloading improves performance as cycles freed
- Affinity made it easy to experiment with offloading

<table>
<thead>
<tr>
<th>PNG Size</th>
<th>X86 Only uploads/sec</th>
<th>X86+Xscale uploads/sec</th>
<th>Speedup</th>
<th>% reduction in context switches</th>
</tr>
</thead>
<tbody>
<tr>
<td>28 KB</td>
<td>161</td>
<td>171</td>
<td>6%</td>
<td>54%</td>
</tr>
<tr>
<td>92 KB</td>
<td>55</td>
<td>61</td>
<td>12%</td>
<td>58%</td>
</tr>
<tr>
<td>150 KB</td>
<td>35</td>
<td>38</td>
<td>10%</td>
<td>65%</td>
</tr>
<tr>
<td>290 KB</td>
<td>19</td>
<td>21</td>
<td>10%</td>
<td>53%</td>
</tr>
</tbody>
</table>
**Email NUMA benchmark**

- Satellite kernels improve performance 39%
Related Work

- **Hive** [Chapin et. al. ‘95]
  - Multiple kernels – single system image

- **Multikernel** [Baumann et. Al. ’09]
  - Focus on scale-out performance on large NUMA architectures

- **Spine** [Fiuczynski et. al.’98]
  - **Hydra** [Weinsberg et. al. ‘08]
  - Custom run-time on programmable device
Conclusions

- Helios manages ‘distributed system in the small’
  - Simplify application development, deployment, tuning

- 4 techniques to manage heterogeneity
  - **Satellite kernels**: Same OS abstraction everywhere
  - **Remote message passing**: Transparent IPC between kernels
  - **Affinity**: Easily express arbitrary placement policies to OS
  - **2-phase compilation**: Run apps on arbitrary devices

- Offloading applications with **zero** code changes
- Helios code release soon.