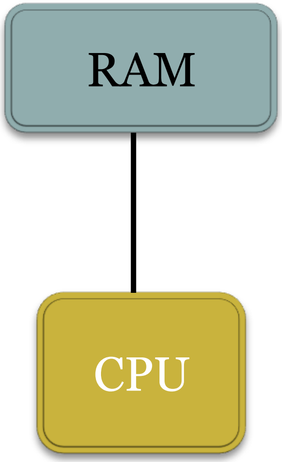
Helios: Heterogeneous Multiprocessing with Satellite Kernels

Ed Nightingale, Orion Hodson, Ross McIlroy, Chris Hawblitzel, Galen Hunt

MICROSOFT RESEARCH

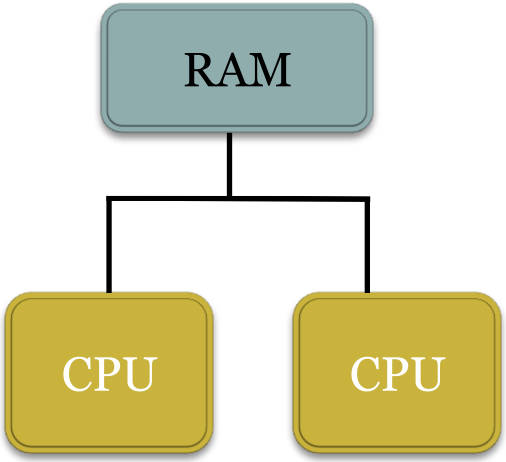
Once upon a time…



Single CPU

- Hardware was homogeneous

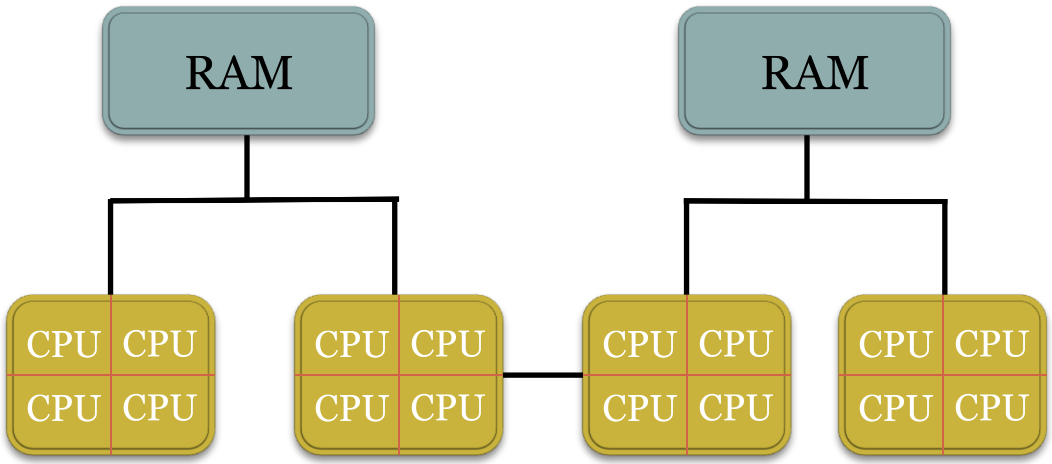
Once upon a time…



SMP

- Hardware was homogeneous

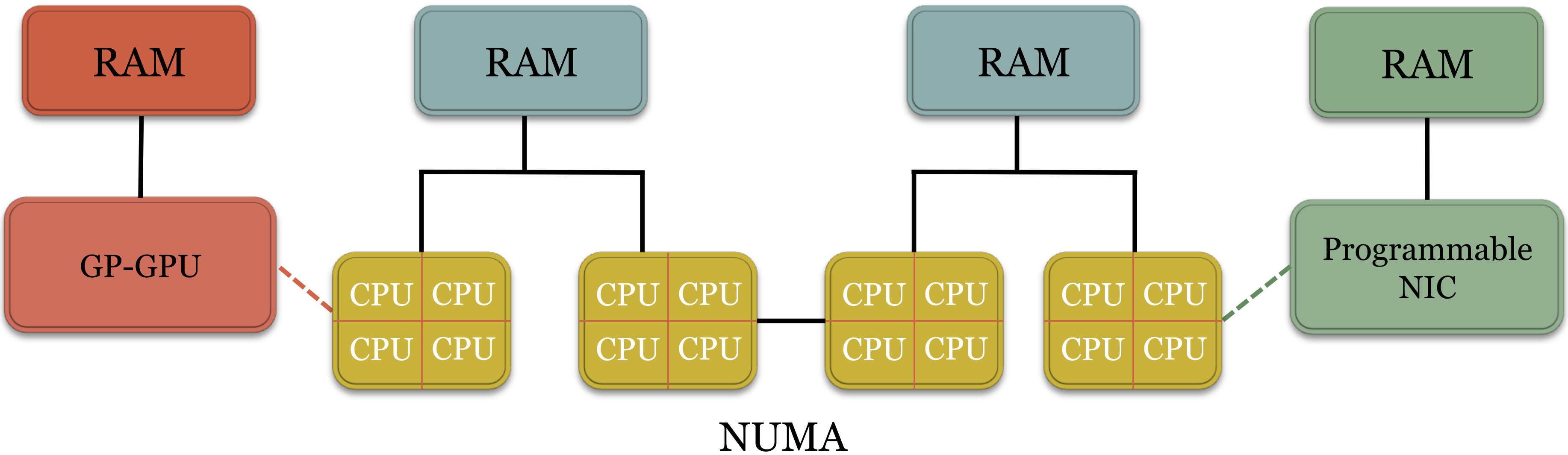
Once upon a time…



NUMA

- 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

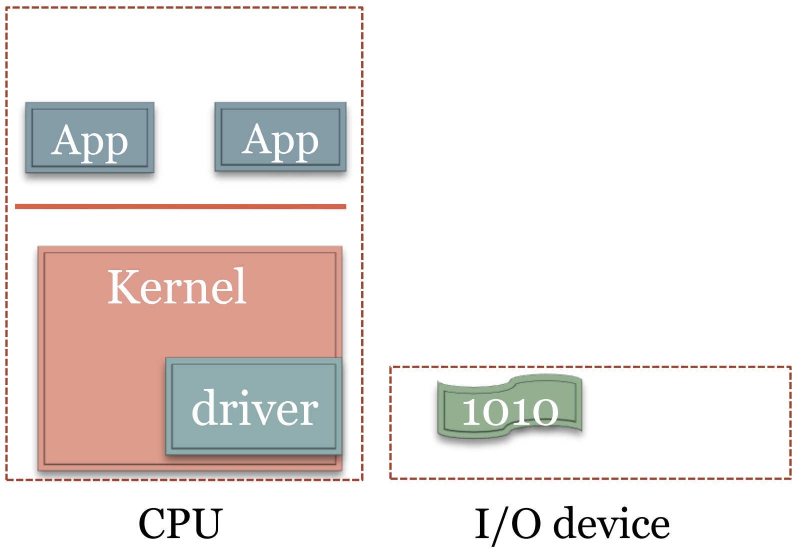
- Helios offloads processes with zero code changes
  - Entire networking stack
  - Entire file system
  - Arbitrary applications

- Improve performance on NUMA architectures
  - Eliminate resource contention with multiple kernels
  - Eliminate remote memory accesses

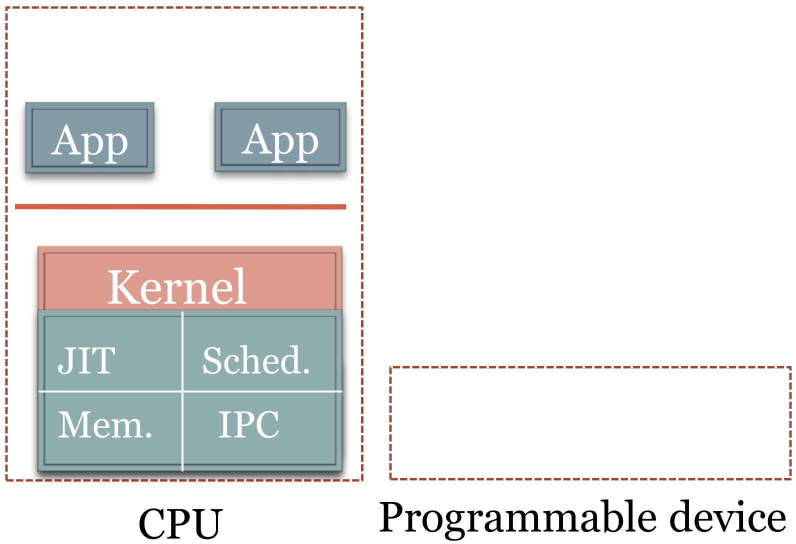
Outline

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

Driver interface is poor app interface

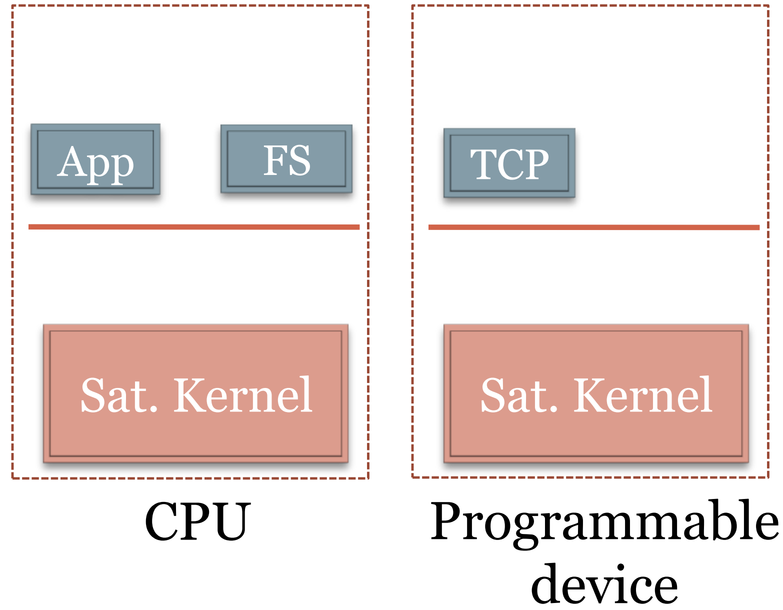


Driver interface is poor app interface



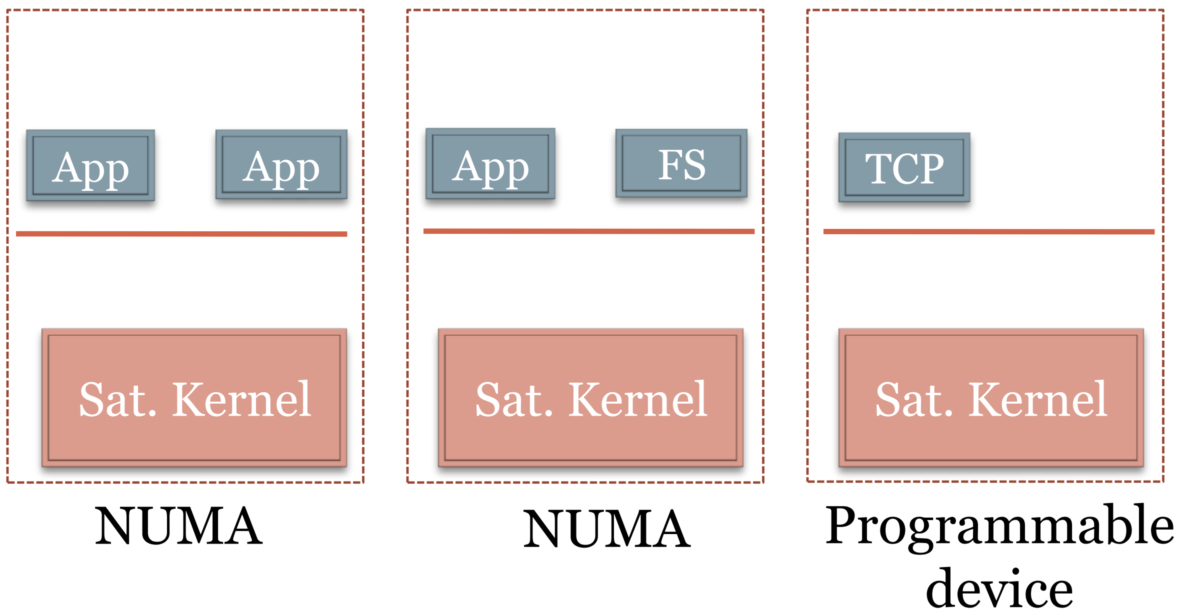
- Hard to perform basic tasks: debugging, I/O, IPC
- Driver encompasses services and runtime…an OS!

Satellite kernels provide single interface



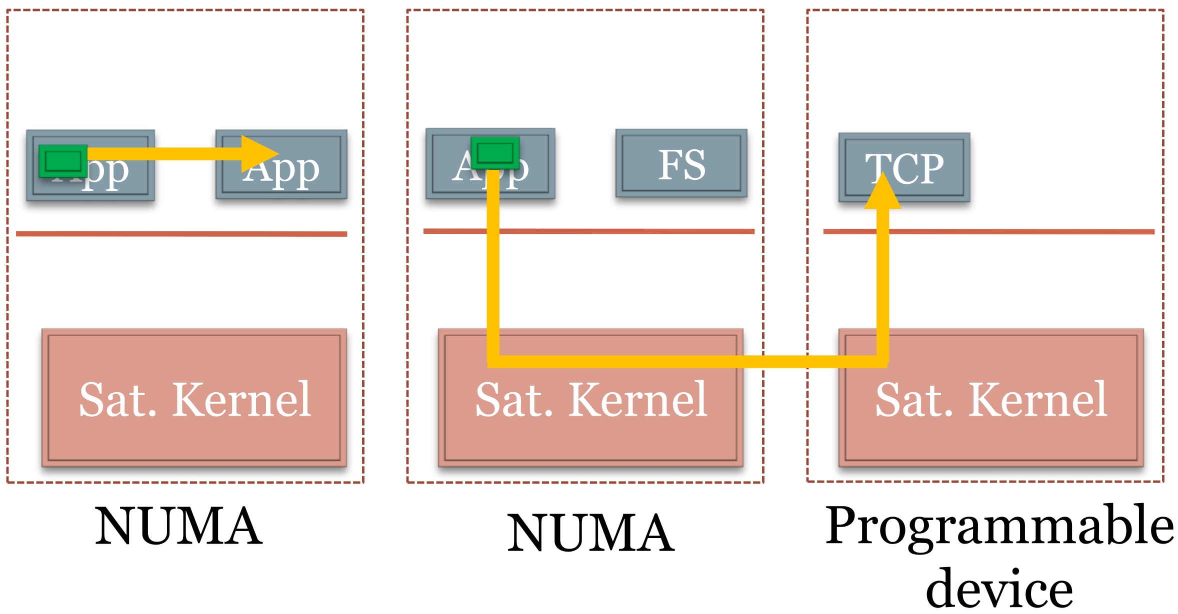
- Satellite kernels:
  - Efficiently manage local resources
  - Apps developed for single system call interface
  - μkernel: Scheduler, memory manager, namespace manager

Satellite kernels provide single interface



- Satellite kernels:
  - Efficiently manage local resources
  - Apps developed for single system call interface
  - μkernel: 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

/fs
/dev/nic0
/dev/disk0
/services/TCP
/services/PNGEater
/services/kernels/ARMv5

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

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

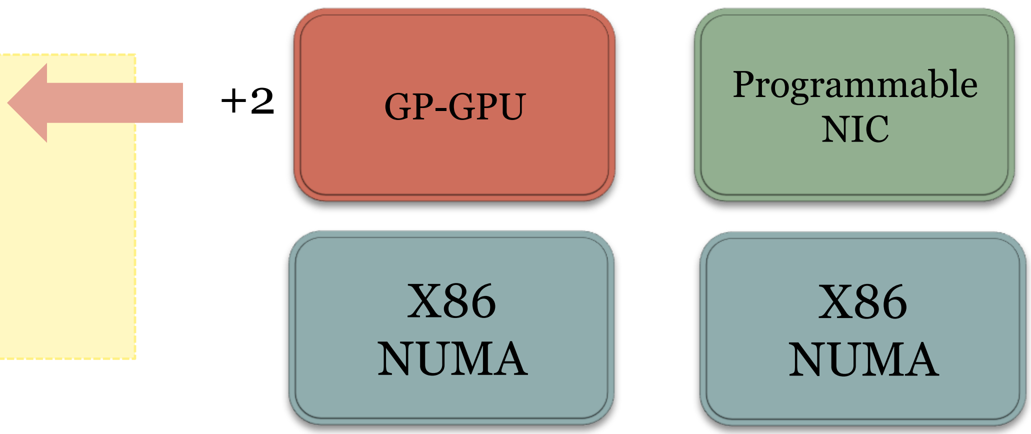
```
<?xml version="1.0" encoding="utf-8"?>
<application name=TcpTest" runtime=full>
 <endpoints>
  <inputPipe id="0" affinity="0"
   contractName="PipeContract"/>
  <endpoint id="2" affinity="+10"
   contractName="TcpContract"/>
 </endpoints>
</application>
```

- Affinity easily edited by dev, admin, or user

Platform Affinity

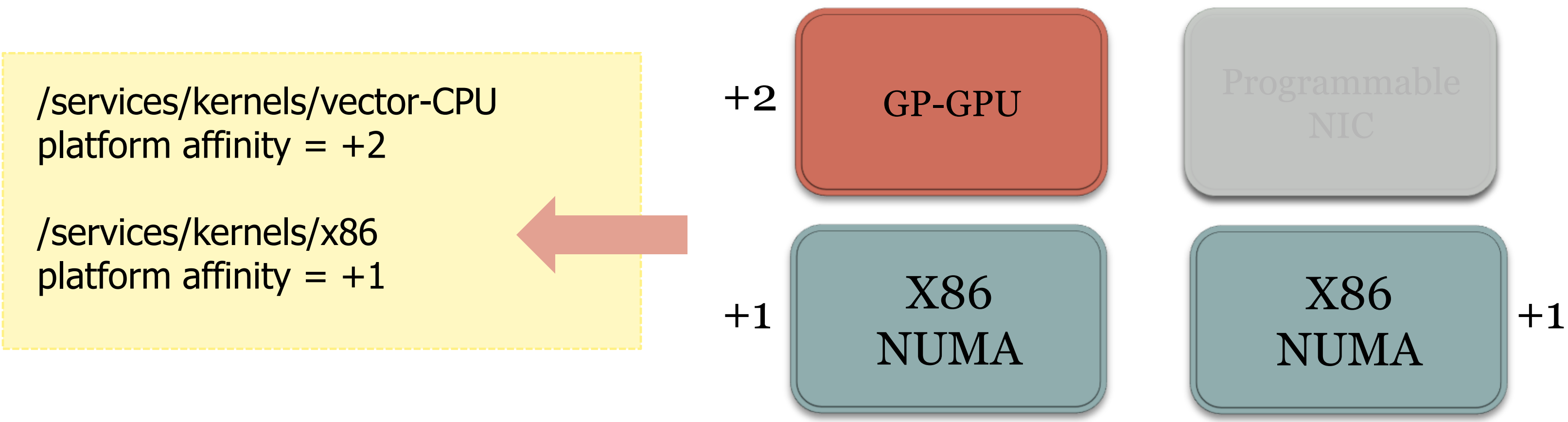
/services/kernels/vector-CPU
platform affinity = +2

/services/kernels/x86
platform affinity = +1



- Platform affinity processed first
- Guarantees certain performance characteristics

Platform Affinity



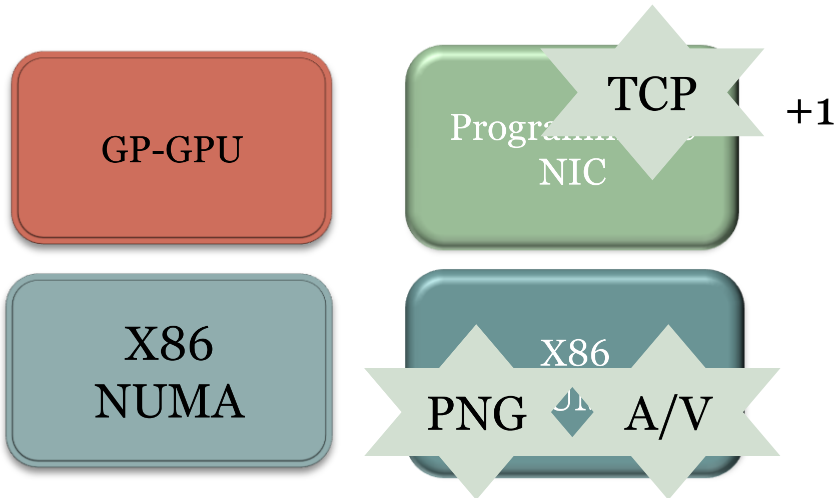
- Platform affinity processed first
- Guarantees certain performance characteristics

Positive Affinity

/services/TCP
communication affinity = +1

/services/PNGEater
communication affinity = +2

/services/antivirus
communication affinity = +3



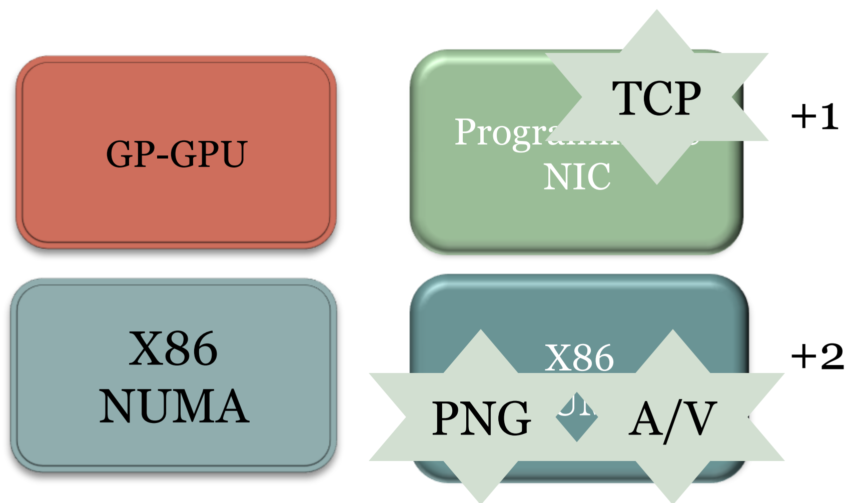
- Represents ‘tight-coupling’ between processes
  - Ensure fast message passing between processes
- Positive affinities on each kernel summed

Positive Affinity

/services/TCP
communication affinity = +1

/services/PNGEater
communication affinity = +2

/services/antivirus
communication affinity = +3



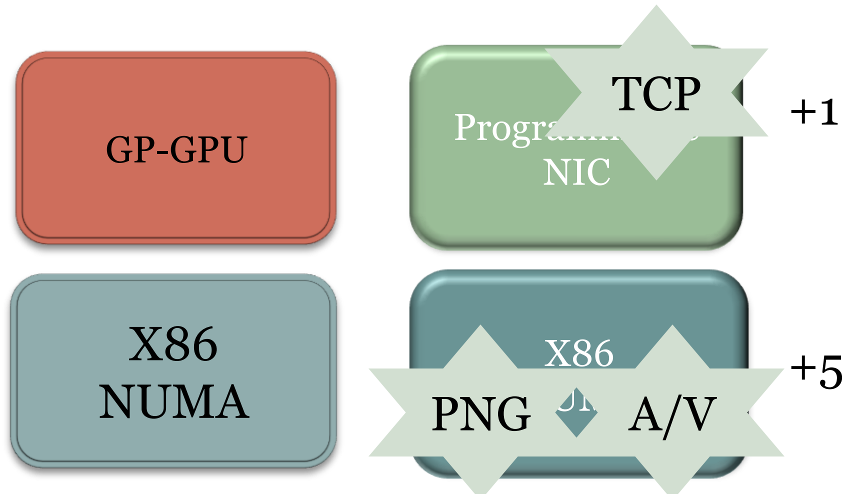
- Represents ‘tight-coupling’ between processes
  - Ensure fast message passing between processes
- Positive affinities on each kernel summed

Positive Affinity

/services/TCP
communication affinity = +1

/services/PNGEater
communication affinity = +2

/services/antivirus
communication affinity = +3

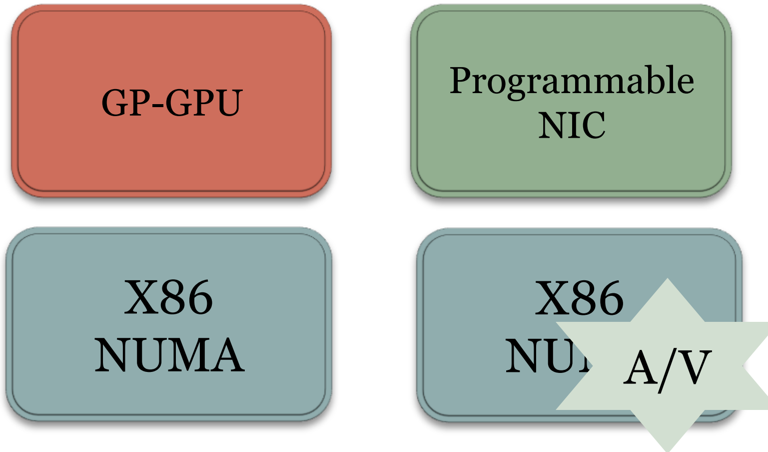


- Represents ‘tight-coupling’ between processes
  - Ensure fast message passing between processes
- Positive affinities on each kernel summed

Negative Affinity

/services/kernels/x86
platform affinity = +100

/services/antivirus
non-interference affinity = -1

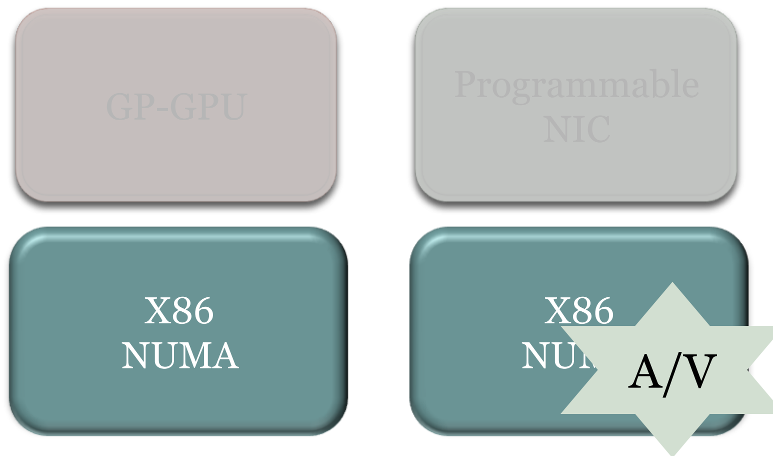


- Expresses a preference for non-interference
  - Used as a means of avoiding resource contention
- Negative affinities on each kernel summed

Negative Affinity

/services/kernels/x86
platform affinity = +100

/services/antivirus
non-interference affinity = -1



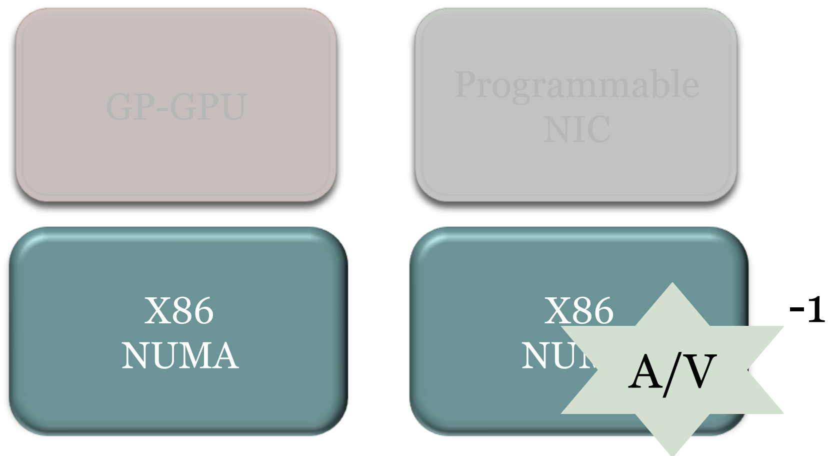
- Expresses a preference for non-interference
  - Used as a means of avoiding resource contention

- Negative affinities on each kernel summed

Negative Affinity

/services/kernels/x86
platform affinity = +100

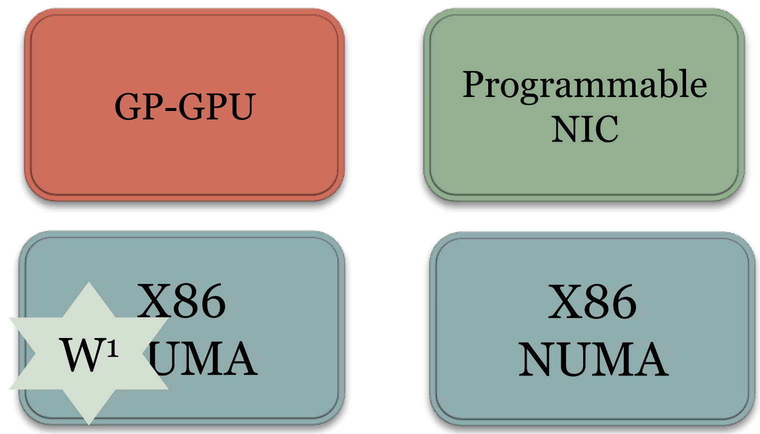
/services/antivirus
non-interference affinity = -1



- Expresses a preference for non-interference
  - Used as a means of avoiding resource contention
- Negative affinities on each kernel summed

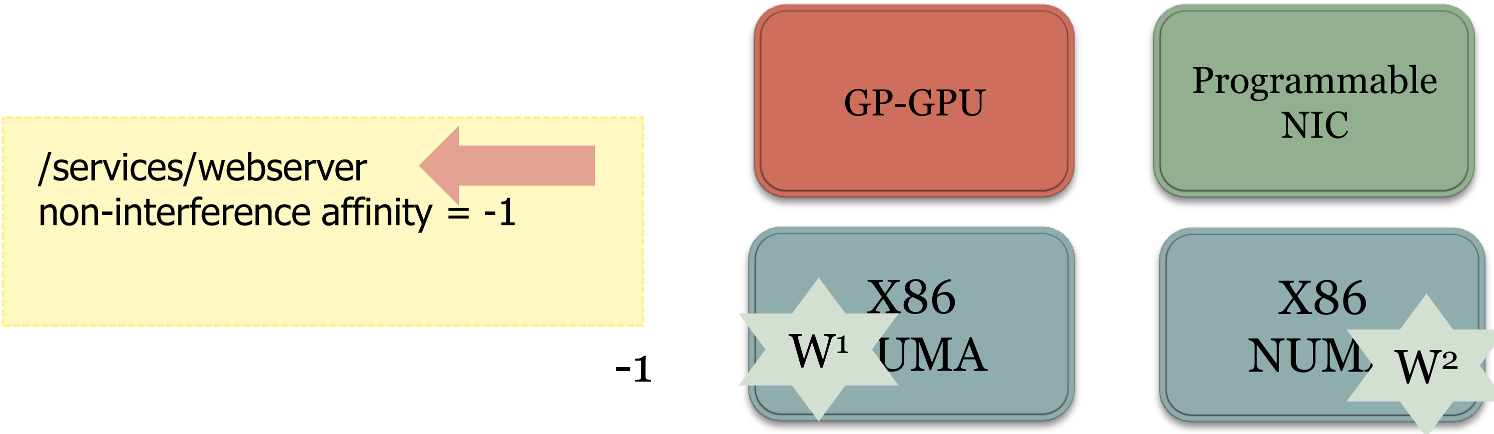
Self-Reference Affinity

/services/webserver
non-interference affinity = -1



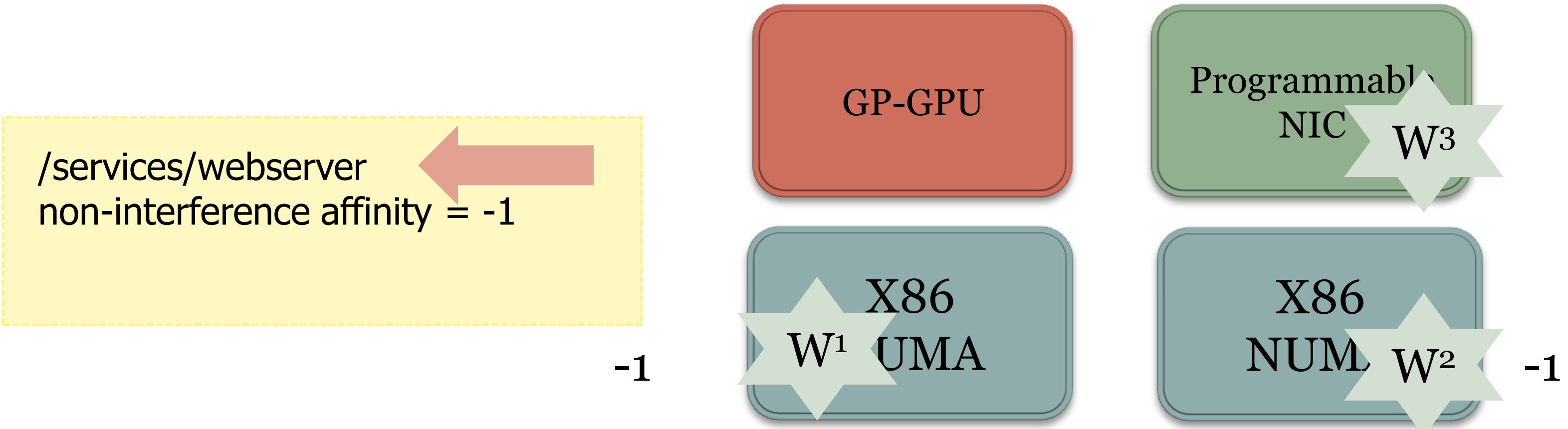
- Simple scale-out policy across available processors

Self-Reference Affinity



- Simple scale-out policy across available processors

Self-Reference Affinity



- 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.

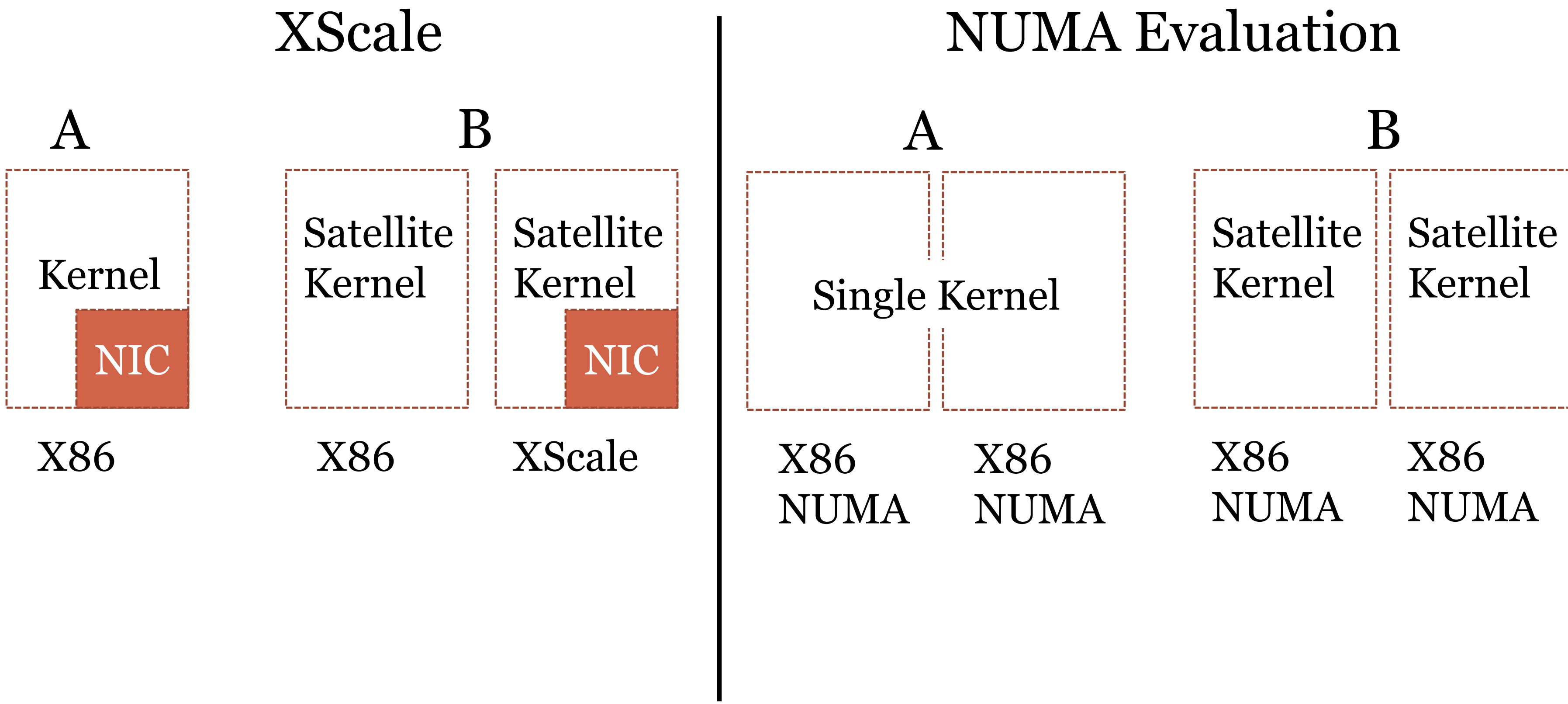
Limitations

- 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



Offloading Singularity applications

Name	LOC	LOC changed	LOM changed
Networking stack	9600	0	1
FAT 32 FS	14200	0	1
TCP test harness	300	5	1
Disk indexer	900	0	1
Network driver	1700	0	0
Mail server	2700	0	1
Web server	1850	0	1

- Helios applications offloaded with very little effort

Offloading Singularity applications

Name	LOC	LOC changed	LOM changed
Networking stack	9600	0	1
FAT 32 FS	14200	0	1
TCP test harness	300	5	1
Disk indexer	900	0	1
Network driver	1700	0	0
Mail server	2700	0	1
Web server	1850	0	1

- Helios applications offloaded with very little effort

Offloading Singularity applications

Name	LOC	LOC changed	LOM changed
Networking stack	9600	0	1
FAT 32 FS	14200	0	1
TCP test harness	300	5	1
Disk indexer	900	0	1
Network driver	1700	0	0
Mail server	2700	0	1
Web server	1850	0	1

- Helios applications offloaded with very little effort

Offloading Singularity applications

Name	LOC	LOC changed	LOM changed
Networking stack	9600	0	1
FAT 32 FS	14200	0	1
TCP test harness	300	5	1
Disk indexer	900	0	1
Network driver	1700	0	0
Mail server	2700	0	1
Web server	1850	0	1

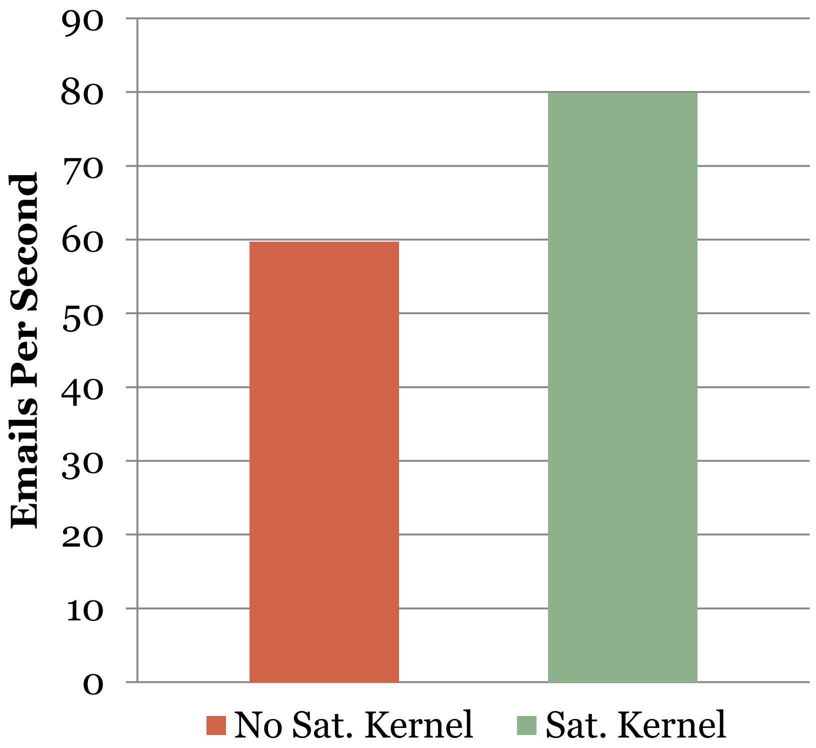
- Helios applications offloaded with very little effort

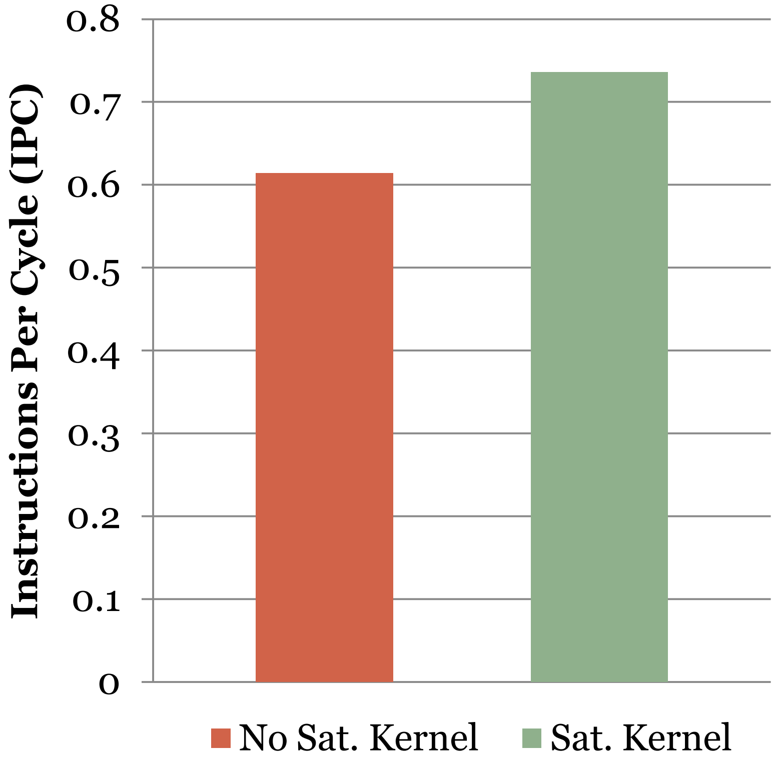
Netstack offload

PNG Size	X86 Only uploads/sec	X86+Xscale uploads/sec	Speedup	% reduction in context switches
28 KB	161	171	6%	54%
92 KB	55	61	12%	58%
150 KB	35	38	10%	65%
290 KB	19	21	10%	53%

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

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 <u>zero</u> code changes
- Helios code release soon.