Hard Real-time Scheduling for Parallel Run-time Systems

Peter Dinda  Xiaoyang Wang  Jinghang Wang  Chris Beauchene  Conor Hetland

Prescience Lab  Department of EECS  Northwestern University

pdinda.org  presciencelab.org
Paper in a Nutshell

- HPC node OS as an RTOS
  - Isolation in time-shared environment
  - Resource control with commensurate performance
  - Coordination via time instead of via synchronization
    - Barrier removal example

- Hard real-time threads in Nautilus kernel
  - Despite x64
  - ~10 us resolution (Xeon Phi KNL)

- Thread group scheduling and coordination
  - ~3 us synchronization for 255 threads (Phi)

- Publicly available codebase
Outline

• Motivation
  – Prior work on soft RT scheduling of distributed machines
  – Modern machines and interesting runtimes

• What is hard real-time?
  – Liu model

• Implementation in Nautilus
  – Threads
  – Groups

• Performance evaluation
  – Limits (mostly on KNL)
  – Fine-grain BSP benchmark

• Conclusions and future work
Experiences with Soft Real-time

• VSched soft RT scheduler extension for Linux
• Consolidation of interactive and batch VMs
• Time-sharing of distributed memory parallel applications on a cluster with performance isolation and control
  – Coordinated scheduling (i.e., gang scheduling based on time) so BSP applications achieve resource-commensurate performance

B. Lin, P. Dinda, *VSched: Mixing Batch and Interactive Virtual Machines Using Periodic Real-time Scheduling*, SC 2005
Can This Apply Within a Node?

• Increasingly interesting target
  – Growing CPU count: Phi now at 256; NUMA, ...
• OS noise concerns continue
• Much finer granularity scheduling and coordination needed
  – OpenMP loops and tasking
  – NESL VCODE model (abstract vector machine)
• New opportunity: substitute timing for synchronization
  – Example: potential barrier removal
What is Hard Real-time?

• Formal admission control process
  – Based on work and deadlines
  – Scheduler can say no
• Scheduler engine \textit{guarantees} all deadlines
• Limitations
  – Scheduler overheads
  – Context switch overheads
• Our system: threads on a NUMA node
What is Hard Real-time?

• Aperiodic threads
  – Have priority
  – Always admitted

• Periodic threads
  – Phase, period, slice (deadline=period)
  – Selective admission (RMA tests)

• Sporadic threads
  – Phase, size, deadline, aperiodic priority
  – Selective admission (EDF tests)

Nautilus as the Basis for an RTOS

• Nautilus: kernel framework for constructing hybrid run-times (HRTs) on x64
  – No userspace, simple address translation, single address space, streamlined primitives, NUMA, ...
  – 15-40% speedup over Linux for Legion run-time

• Particularly salient for an RTOS:
  – No page faults, only capacity TLB misses
  – Deterministic path length in drivers and all core functionality
  – Steerable interrupts

K. Hale, C. Hetland, P. Dinda, Multiverse: Easy Conversion Of Runtime Systems Into OS Kernels Via Automatic Hybridization, ICAC 2017
Local (per-CPU) Scheduler

- Complex, but core concept is Eager EDF
- Admission control in thread context
- Reservations for aperiodic, sporadic, missing time...
- Interrupt control
- ns-resolution time based on cycle counter (ARAT, ConstantTSC)
  - Scheduling interrupts: APIC timer (TSC deadline mode if available)
The Curse of Missing Time

• Unaccounted time within kernel itself
  – Scheduler overhead, context switch overhead, etc.

• Deliberate, nondeterministic, unaccounted time due to System Management Interrupts (SMIs)
  – Firmware-level interrupts
  – Higher privilege than kernel or even VMM
  – Cannot be turned off
  – Like an alien abduction from the scheduler’s perspective
    • “My clock just jumped forward 10 us!”

• Our approach to both:
  (a) Reservations, (b) Eager Earliest Deadline First

Global (per-node) Scheduler

- Local scheduler coordination via
  - Time (mostly)
  - Interrupts (sparingly)
  - No global locking

- Interrupt steering and segregation
  - Interrupt-free CPUs see only scheduling-related interrupts
  - Interrupt-laden CPUs have careful interrupt control

More Surprises  Few Surprises
Group Scheduling

• **Local schedulers’ clocks synchronized**
  – Variance <1000 cycles (<1 us) over 256 CPUs on Phi

• **Thread groups and group admission control**
  – Main element is admission control done in parallel
  – All or nothing

• **Phase correction** to coordinate initial thread arrival on all involved local schedulers

• **Same constraints on all local schedulers results in gang scheduling** of the group of threads
  – Without explicit communication
Local Scheduler Synchronization on Phi

Local Schedulers calibrate to within 4000 cycles (3us) of each other

8 CPU example, 256 CPU similar
Code Measures

• Scheduler: ~5000 LoC (C)
  – Also includes work-stealing, thread pools, garbage collection support, and tasks

• Groups and Group Scheduling: ~1000 LoC (C)

• Other changes: ~2000 LoC (C+Assembly)
  – Low-level CPU-state maintenance / context switch
  – Additional thread states
  – Assorted
Test machines

• Phi
  – Supermicro 5038ki ("Colfax KNL Ninja")
  – Intel Xeon Phi 7210 ("Knight’s Landing")
    • 64 cores, 4 hardware threads per core
    • 1.3 GHz
    • 16 GB MCDRAM, 96 GB DRAM
    • All throttling/burst behavior disabled in BIOS

• R415
  – Dell R415
  – AMD 4122
    • 2 sockets, 8 cores/threads total,
    • 2.2 GHz
    • 16 GB DRAM
    • All throttling/burst behavior disabled in BIOS
Validation Through External Monitoring

Phi with parallel port attached to oscilloscope period=100us, slice=50us, phase=0
It is important to realize that an individual Phi CPU is quite slow, with infeasible timing constraints. Each curve represents a different CPU, the scheduling overheads will be lower in terms of both cycles and real time, as can be seen for the R415 in Figure 5(b).

In steady state operation, once a group of threads has been admitted, deadlines are missed by only small amounts. These are quite low, as shown previously. The time costs of group admission control to allow admission of threads involve the scheduling pass itself, while the rest is spent in interrupt processing and the context switch. These lower overheads in turn make possible even smaller schedulable periods (\(\mu s\)). About half of the overhead is spent in timeout and the next thread’s arrival can be processed in the same arrival, and the timeout, although these can overlap (one thread’s admission). It is also instructive to see what happens beyond the edge of feasibility.

In Figures 8 (Phi) and 9 (R415) we show the average and variance of miss times both for feasible and infeasible timing constraints. For feasible timing constraints, the miss times are of course always zero. For infeasible timing constraints, the miss times are generally quite small compared to the constraint. Note again that, in normal operation, infeasible constraints are filtered out by admission control.

In our system, only the overheads of the local schedulers matter. In steady state operation, once a group of threads has been admitted, deadlines are missed by only small amounts. These lower overheads in turn make possible even smaller schedulable periods (\(\mu s\)). About half of the overhead is spent in timeout and the next thread’s arrival can be processed in the same arrival, and the timeout, although these can overlap (one thread’s admission).

It is also instructive to see what happens beyond the edge of feasibility. For infeasible constraints, which are usually not admitted, deadlines are missed by only small amounts. These lower overheads in turn make possible even smaller schedulable periods (\(\mu s\)). About half of the overhead is spent in timeout and the next thread’s arrival can be processed in the same arrival, and the timeout, although these can overlap (one thread’s admission).

Limits on Phi: \(\sim 10\mu s\)

![Limits on Phi: \(\sim 10\mu s\)](image)
Controlled Miss Behavior on Phi

![Graph showing controlled miss behavior on Phi](image)
Limits on R415: ~4us

Figure 6: Local scheduler deadline miss rate on Phi as a function of period (µs) and slice (% of period). Once the period and slice are feasible given scheduler overhead, the miss rate is zero.

Figure 7: Local scheduler deadline miss rate on R415 as a function of period (µs) and slice (% of period). Once the period and slice are feasible given scheduler overhead, the miss rate is zero.

It is important to realize that an individual Phi CPU is quite slow, both in terms of its clock rate and its superscaler limits. On faster individual CPUs, the scheduling overheads will be lower in terms of both cycles and real time, as can be seen for the R415 in Figure 5(b).

These lower overheads in turn make possible even smaller scheduling constraints, as can be seen for the R415 in Figure 7. Here, the edge of feasibility is about 4 µs.

It is also instructive to see what happens beyond the edge of feasibility, when deadlines misses occur because the timing constraint is simply not feasible given the overhead of the scheduler. In Figures 8 (Phi) and 9 (R415) we show the average and variance of miss times both for feasible and infeasible timing constraints. For feasible timing constraints, the miss times are of course always zero. For infeasible timing constraints, the miss times are generally quite small compared to the constraint. Note again that, in normal operation, infeasible constraints are filtered out by admission control.

5.4 Group admission control costs

In steady state operation, once a group of threads has been admitted in our system, only the overheads of the local schedulers matter. These are quite low, as shown previously. The time costs of group operation are born solely when the group of threads is admitted, namely for the algorithm in Section 4.3. Figure 10 breaks down the absolute costs of the major steps of the algorithm: group join, leader election, distributed admission control, and the final barrier.
Figure 6 shows the local scheduler miss rate on the Phi. Here, the x-axis represents the slice as a fraction of the period, and the y-axis shows the miss rate. On this kind of graph, we expect a zero miss rate. As the graph shows, the transition point, or the “edge of feasibility,” is for a period of about 10 us.

Once the period and slice are feasible given the scheduler overhead, the miss rate is zero. For infeasible timing constraints, the miss times are generally zero. For feasible timing constraints, the miss times are of course always zero. For infeasible constraints, which are usually not admitted, deadlines are missed by only small amounts.

In Figures 8 (Phi) and 9 (R415) we show the average and variance of miss times both for feasible and infeasible timing constraints. The figures demonstrate that the miss times are quite small compared to the constraints. Note again that, in normal operation, infeasible constraints are filtered out by admission control to allow admission of threads into the system.

In steady state operation, once a group of threads has been admitted, deadlines are missed by only small amounts. These are quite low, as shown previously. The time costs of group join, leader election, distributed admission control, and the absolute costs of the major steps of the algorithm: group join, leader election, distributed admission control, and the final barrier. These lower overheads in turn make possible even smaller scheduling overheads.

It is also instructive to see what happens beyond the edge of feasibility. Figure 7 shows the local scheduler deadline miss rate on R415 as a function of period (µs) and slice (% of period). Once the period and slice are feasible given the scheduler overhead, the miss rate is zero. On the other hand, a sharp disconnect: for too small of a period or slice, or too large of a period (as we expect given the overhead measurements).

In Figures 8 (Phi) and 9 (R415) we show the average and standard deviation of miss times for feasible schedules on R415. For infeasible constraints, which are usually not admitted, deadlines are missed by only small amounts.

Controlled Miss Behavior on R415

<table>
<thead>
<tr>
<th>Slice (% of period)</th>
<th>Miss time (us)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td>0</td>
</tr>
<tr>
<td>30</td>
<td>0</td>
</tr>
<tr>
<td>40</td>
<td>0</td>
</tr>
<tr>
<td>50</td>
<td>0</td>
</tr>
<tr>
<td>60</td>
<td>0</td>
</tr>
<tr>
<td>70</td>
<td>0</td>
</tr>
<tr>
<td>80</td>
<td>0</td>
</tr>
</tbody>
</table>
Fine-grain BSP Microbenchmark

each thread in group:

for (i=0; i<N; i++) {

    local_compute(granularity)

    optional_barrier();

    write_to_neighbor(granularity)

    optional_barrier();
}

Barrier overhead grows with shrinking granularity
Barriers could be removed if the threads ran in lock-step
We ran a parameter study using the microbenchmark and our scheduler's interaction with the benchmark. All threads are enough in all cases to allow us to see the steady state behavior of the parameter space where the compute and communicate phases start to avoid race conditions on the elements. With a group real-time scheduling scheme, this property may be feasible to provide via timing.

We considered both aperiodic and periodic group constraints. The entire parameter space explored consists of 1,037,852 combinations of period and slice plotted.

Figure 13 clearly shows that this is indeed the case. Here, we can readily understand simply by considering the extremes of the entire parameter space explored. The observations suggest that the period (\(T\)) and slice (\(S\)) chosen, benchmark execution rate matches the time resources given. Regardless of the specific period (\(T\)) and slice (\(S\)), we expect the application to operate at 50% of its top speed, not slower. Our prior work also showed how to make this possible in a distributed environment for relatively coarse granularities using soft real-time scheduling with feedback control. Does our hard real-time scheduling scheme with timing isolation, combined with other readily available resource utilization (utilization is almost certain to avoid any blocking on a descheduled thread.

The group hard real-time scheduling scheme can keep application resources with commensurate performance. That is, if constraints of a periodic real-time constraint can be used to control timing isolation is limited to timing, as we previously described. Does our hard real-time scheduler for providing resource control with commensurate performance: Can we emulate iterative computation on a discrete domain, modeled as a memory machine?

Figure 13: Resource control with commensurate performance for Phi, 255 thread BSP benchmark, granularity 128, 900 combinations of period and slice plotted.
6.4 Barrier removal

Because of the lock-step execution across CPUs that our scheduler imposes, there are no timing constraints themselves for some of the combinations. In the benchmark without barriers, execution rate roughly matches the time resources given. However, when barriers are present, the overall task execution time becomes similar to the resources allocated. As the granularity shrinks, proportionate cost increases.

The cost of doing so? The real-time scheduled benchmark without barriers exceeds the performance of the non-real-time benchmark with barriers. Time in ns.

The effects of barrier removal are much more varied. The barrier removal is much more pronounced, as Amdahl's law would suggest, and the effects of barrier removal at the coarsest granularity. Disparity of points above line indicates benefits of barrier removal for the coarsest granularity computation on 255 CPUs. Here, the benefits depend on the granularity of the computation. Considering the BSP structure as represented in the benchmark, Amalgamation Performance Bene

Figure 14, in which we consider the resource control remains, as can be seen in Figure 14, in which we consider the control resources with commensurate performance for a hard real-time groups, and the fully balanced nature of a BSP computation as modeled in our microbenchmark, it is possible to consider discarding the optional barriers. What are the costs of doing so? The points form the line (almost all of them) represent control remains, as can be seen in Figure 14, in which we consider the control resources with commensurate performance for a hard real-time groups, and the fully balanced nature of a BSP computation as modeled in our microbenchmark, it is possible to consider discarding the optional barriers. What are the costs of doing so?

Resources can provide for hard real-time groups, and the fully balanced nature of a BSP computation as modeled in our microbenchmark, it is possible to consider discarding the optional barriers. What are the costs of doing so? The points form the line (almost all of them) represent control remains, as can be seen in Figure 14, in which we consider the control resources with commensurate performance for a hard real-time groups, and the fully balanced nature of a BSP computation as modeled in our microbenchmark, it is possible to consider discarding the optional barriers. What are the costs of doing so?

Because of the lock-step execution across CPUs that our scheduler imposes, there are no timing constraints themselves for some of the combinations. In the benchmark without barriers, execution rate roughly matches the time resources given. However, when barriers are present, the overall task execution time becomes similar to the resources allocated. As the granularity shrinks, proportionate cost increases.

The cost of doing so? The real-time scheduled benchmark without barriers exceeds the performance of the non-real-time benchmark with barriers. Time in ns.

The effects of barrier removal are much more varied. The barrier removal is much more pronounced, as Amdahl's law would suggest, and the effects of barrier removal at the coarsest granularity. Disparity of points above line indicates benefits of barrier removal for the coarsest granularity computation on 255 CPUs. Here, the benefits depend on the granularity of the computation. Considering the BSP structure as represented in the benchmark, Amalgamation Performance Bene

Figure 14, in which we consider the resource control remains, as can be seen in Figure 14, in which we consider the control resources with commensurate performance for a hard real-time groups, and the fully balanced nature of a BSP computation as modeled in our microbenchmark, it is possible to consider discarding the optional barriers. What are the costs of doing so? The points form the line (almost all of them) represent control remains, as can be seen in Figure 14, in which we consider the control resources with commensurate performance for a hard real-time groups, and the fully balanced nature of a BSP computation as modeled in our microbenchmark, it is possible to consider discarding the optional barriers. What are the costs of doing so?
Fine-grain BSP Microbenchmark

each thread in group:

for (i=0; i<N; i++) {
    local_compute(granularity)
    optional_barrier();
    write_to_neighbor(granularity)
    optional_barrier();
}

Barrier overhead grows with shrinking granularity
Barriers could be removed if the threads ran in lock-step
with barriers removed, matches and sometimes slightly exceeds the performance of the non-real-time scheduled benchmark. The latter is running at 100% utilization. Regardless of the speciﬁc period/slice combinations where the barrier to the time without. All points above line indicates beneﬁt of barrier removal at the coarsest granularity. Dis-3

The beneﬁts of barrier removal are much more varied. In distributed memory parallel systems noise, overhead, and responses.

Now, however, the hard real-time cases, with barriers removed, can roughly matches the time resources given. Regardless of the speciﬁc period/slice combinations with the same utilization, the real-time scheduled benchmark without barriers and with period/slice constraints giving 90% utilization, is similar in performance to the non-real-time benchmark with barriers. Time in ns.

For a range of period/slice combinations and utilizations, the real-time scheduled benchmark without barriers exceeds the performance of the non-real-time scheduled benchmark with barriers. Time in ns.

The real-time scheduled benchmark with barriers at 100% utilization. Time in ns.

The real-time scheduled benchmark without barriers and with period/slice constraints giving 90% utilization, is similar in performance to the non-real-time benchmark without barriers and with period/slice constraints giving 90% utilization.

In Figure 15, which is semantically identical, shows the beneﬁts depend on the granularity of the computation. Con-6

sidering the BSP structure as represented in the benchmark, Am-

possible to consider discarding the optional barriers. What are the costs of doing so? Dahl's law tells us that the cost of compute_local_element() period chosen, benchmark execution rate, and performance degradation. These issues com-

erative timing, and ultimately differentiated timing, and ultimately

In Figure 16, which is semantically identical, shows the bene-8

figure, each point represents a single combination of period and slice (there are 900), plus aperiodic con-

resources allocated. As the granularity shrinks, proportionate con-

the performance of the benchmark is cleanly controlled by the time scheduling with 100% utilization. Regardless of the period selected, benchmark execution rate, and performance degradation. These issues com-

erative timing, and ultimately differentiated timing, and ultimately

Aperiodic

Performance Gain

granularity 128

x 10^8

x 10^8

Performance

Gain

Aperiodic
Barrier Removal

Time without Barrier Removal

Time with Barrier Removal

Performance Gain

Aperiodic

granularity 1

Time with the barrier to the time without. All points above the line (almost all of them) represent combinations of period and slice (there are 900). The point compares the time with barriers to the time without. Because of the lock-step execution across CPUs that our scheduler must maintain, as can be seen in Figure 14, in which we consider the performance of the benchmark is running faster without the barrier. The points form a line (almost all of them) represent computations on 255 CPUs. In the performance benchmark with barriers at 100% utilization. Regardless of the period selected, performance benefits from barrier removal are much more varied. Effects of barrier removal are much more varied. These issues combined as scale increases, as do the overheads of using full OS scheduling of CPUs in a shared memory node, not courser grain scheduling of a distributed memory machine.

In distributed memory parallel OS noise, overhead, and responses.

Our work ties to long-running research in several areas, as described below. It is important to keep in mind that we target

7 RELATED WORK

Now, however, the hard real-time cases, with barriers removed, can suggest, and the effects of barrier removal are much more varied. Issues of doing so?

What are the possible to consider discarding the optional barriers. What are the

Amdahl's law tells us that the cost of

possible to consider discarding the optional barriers. What are the

dahl's law tells us that the cost of

Resources allocated. As the granularity shrinks, proportionate con-

performance bene

of a BSP computation as modeled in our microbenchmark, it is

because the overall task execution time becomes similar to

the timing constraints themselves for some of the combinations.

the benchmark without barriers and with period/slice constraints gi-

The benefits of barrier removal at the coarsest granularity. Dis-

The benefits of barrier removal at the coarsest granularity. Dis-

The performance benchmark without barriers exceeds the performance of the non-real-

the real-time scheduled benchmark with barriers. Time in ns.

the real-time scheduled benchmark with barriers. Time in ns.

Figure 14: Resource control with commensurate performance for a

Figure 16, which is semantically identical, shows the bene

nest granularity computation on 255 CPUs Here, the bene

nest granularity with barriers. All period/slice combinations are

nest granularity with barriers. All period/slice combinations are

Figure 15: Bene

Figure 16: Bene

different period/slice combinations with the same uti-

different period/slice combinations with the same uti-

Different period/slice combinations with the same uti-

6.4 Barrier removal

With barriers removed, matches and sometimes slightly exceeds the

with barriers removed, matches and sometimes slightly exceeds the

with barriers removed, matches and sometimes slightly exceeds the

write_remote_element_on()
Related Work

• OS Noise
• Gang scheduling
• Vsched / Coordinated soft RT
  – As in motivation
• Mondragon
• RTVirt
• Tessellation
• Barrelfish
  – Also coordination via time
Ongoing/Future Work

• Further overhead reduction
  – Reduce granularity
• Real-time tasks
• Interrupt-free scheduling
  – Avoid interrupt overheads / Reduce granularity
  – Compiler-based injection of cooperative scheduling calls
• Real-time executive model
  – Scheduling implemented at compile-time as a superloop, as in safety-critical and/or smallest embedded systems
• Custom hardware for scheduling and synchronization
  – Intel HARP / FPGA
Paper in a Nutshell

• HPC node OS as an RTOS
  – Isolation in time-shared environment
  – Coordination via time instead of via synchronization
  – Barrier removal example

• Hard real-time threads in Nautilus kernel
  – Despite x64
  – ~10 us resolution (Xeon Phi KNL)

• Thread group scheduling and coordination
  – ~3 us synchronization for 255 threads (Phi)

• Publicly available codebase
For More Information

• Peter Dinda
  – pdinda@northwestern.edu
  – http://pdinda.org
• Codebase available
  – http://v3vee.org
• Prescience Lab
  – http://presciencelab.org
• Acknowledgements
  – NSF, DOE