

Reducing OS noise using offload driver on Intel® Xeon Phi™ Processor

Grzegorz Andrejczuk grzegorz.andrejczuk@intel.com

Jaroslaw Kogut jaroslaw.kogut@Intel.com

Intel Technology Poland

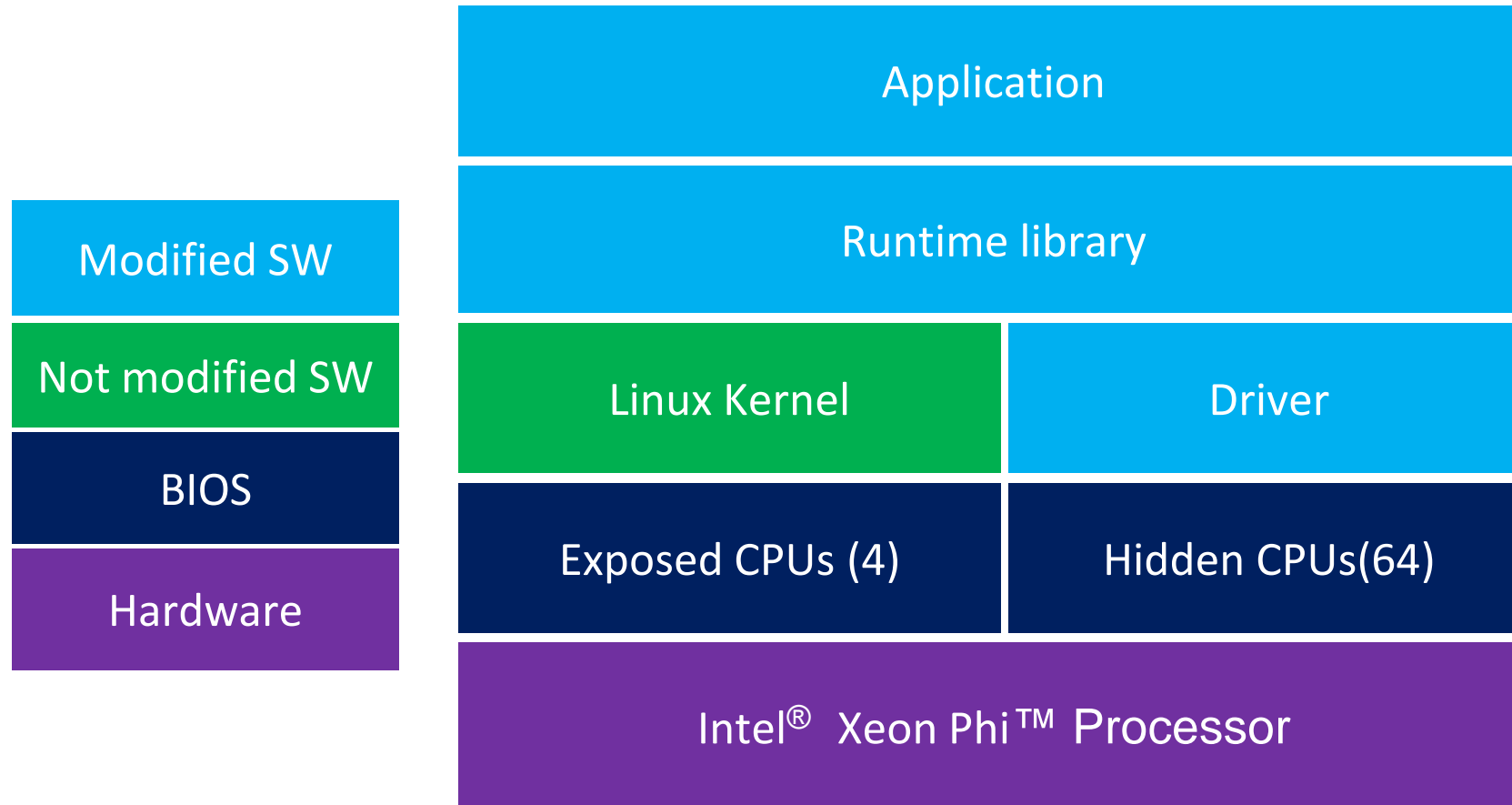
Introduction

Irregularities caused by events like timer interrupts are already recognized as a problem for highly parallel applications running on multi-core processors. Linux tickless kernel is becoming a standard option on HPC compute nodes, but it does not eliminate OS noise completely.

The idea

- Intel® Xeon Phi™ Processors are divided into host CPUs (visible by OS) and hidden CPUs initialized from the driver module (not visible for OS).
- The memory space is shared between host and hidden CPUs.
- The solution does not require any Linux kernel modifications.
- We are using simple scheduling which simulates the data level parallelism (DLP) approach - doing the same work on different data using all available hidden CPUs.

Solution Architecture



Environment setup

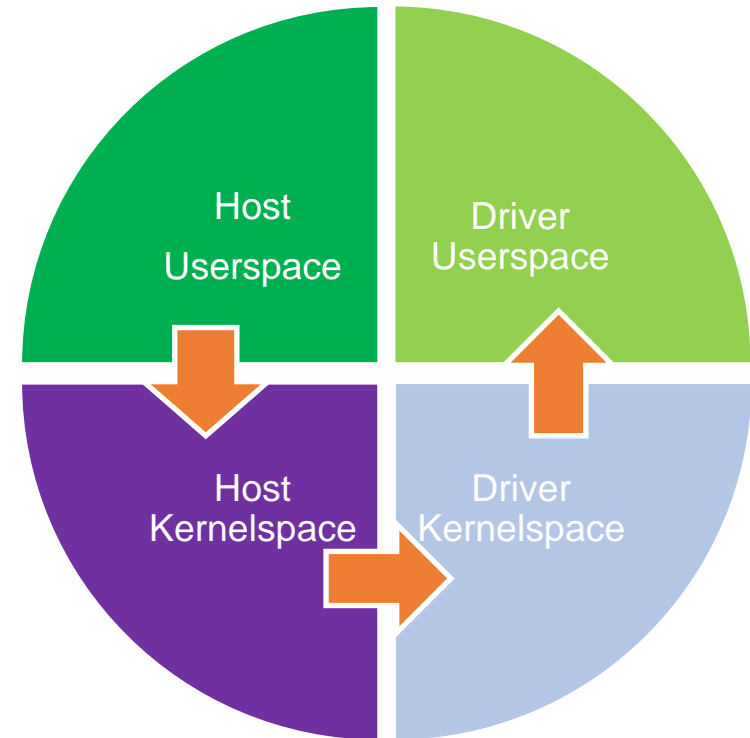
1. Hide a subset of processors by removing appropriate entries from the MADT ACPI table
2. Put the modified ACPI table into initrd
3. Boot Linux kernel with the modified initrd
4. Load the driver
5. Boot the hidden CPUs by specifying their APIC IDs in SYSFS

Initialization control flow

1. Application initializes the runtime library.
2. The runtime library reserves the hidden CPUs and uses the driver to request to execute its code there.
3. The driver initializes the hidden CPUs and starts executing the *"idle"* loop.
4. The hidden CPUs wait for the work from Application.

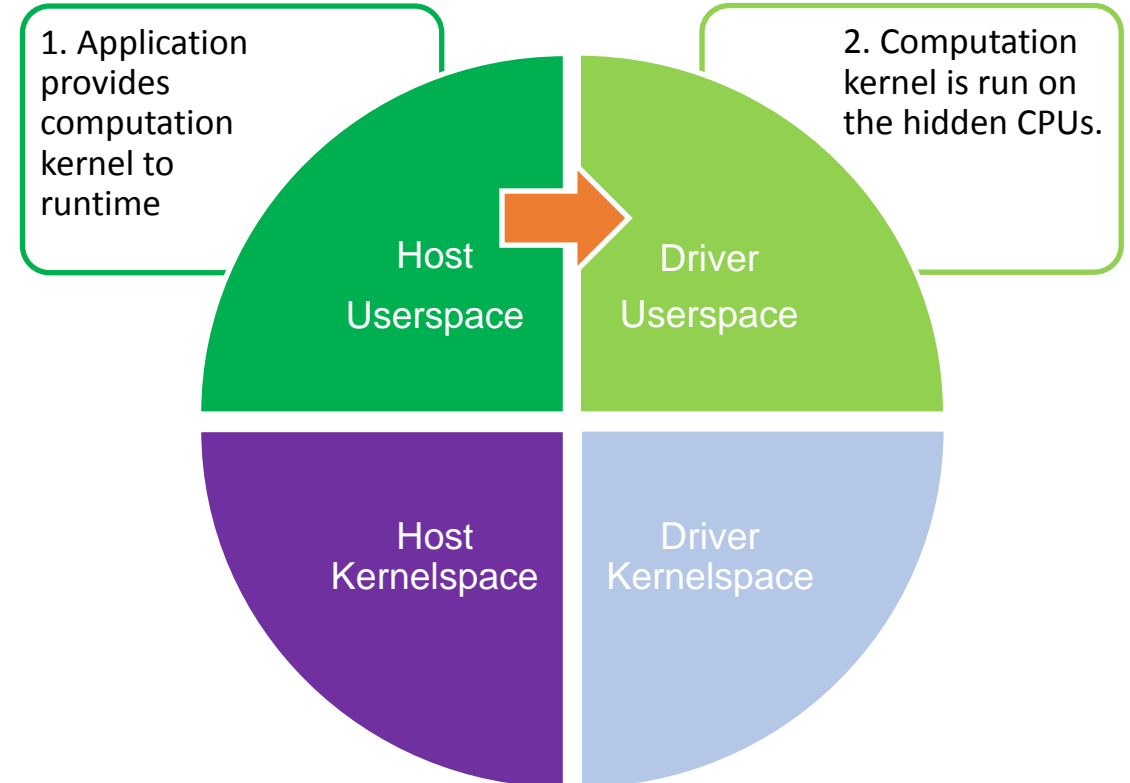
Work:

- pointer to function
- pointer to data



Offload control flow

1. Application offloads the work to hidden CPUs
Register values are preserved as they would be in a normal function call
2. The hidden CPUs get the work and run it
This happens without any interaction with the OS
3. When the work is done the hidden CPUs wait (using Ring 3 MWAIT) while the application gets the results
API supports synchronous and asynchronous calls



Runtime API example

```

#include "rt.h"

/* Run on hidden CPU */
void fun(void *param)
{
    int *d = (int*)param;
    while (*d--);
}

int main()
{
    int loops = 100;
    struct rt_ctx *ctx;
    struct rt_work work = {fun, &loops};

    if (ctx = rt_init()) {
        /* synchronous offload */
        rt_run(ctx, &loops);

        /* loops equals 0 at this point */
        rt_exit(ctx);
    }
}

```

initialization

execution

shutdown

Performance - setup

Hardware:

- Intel® Xeon Phi™ Processor 7250, 68 cores @ 1.4GHz, 16 GB MCDRAM Flat/Quadrant
- 6 x 32 GB DDR4
- First 4 cores (16 CPUs) visible to OS
- Remaining CPUs are hidden

Software:

- RedHat* 7.3 + 4.11 kernel
 - Configured with 7800 MCDRAM hugepages
- Intel® Composer XE 2017.4.056

Benchmarks:

- The STREAM Benchmark used as reference
- Modified STREAM using hugepages and `__assume_aligned(64)` compiler hint
- Runtime compiled to run on host CPUs and compiled to run on hidden CPUs
- Common problem size - 1.4 GiB
- All measurements on 4-67 CPUs

Performance - STREAM

Reference experiment :

- STREAM Benchmark using OpenMP
- nohz_full=1-271

Experiment 1:

- STREAM using OpenMP
- nohz_full=1-271 isolcpus=1-271

Experiment 2:

- Modified STREAM using OpenMP
- nohz_full=1-271 isolcpus=1-271

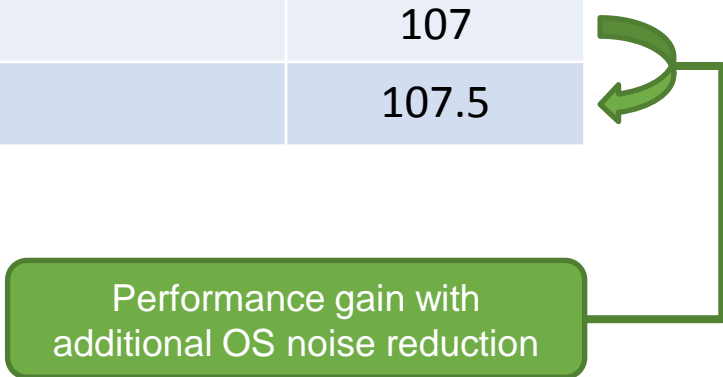
Experiment 3:

- Modified STREAM using runtime compiled to run on host
- nohz_full=1-271 isolcpus=1-271

Experiment 4:

- Modified STREAM using runtime compiled to use hidden CPUs
- nohz_full=1-271 isolcpus=1-271

Measurement	Result [%]
Reference	100
Experiment 1	100
Experiment 2	106
Experiment 3	107
Experiment 4	107.5



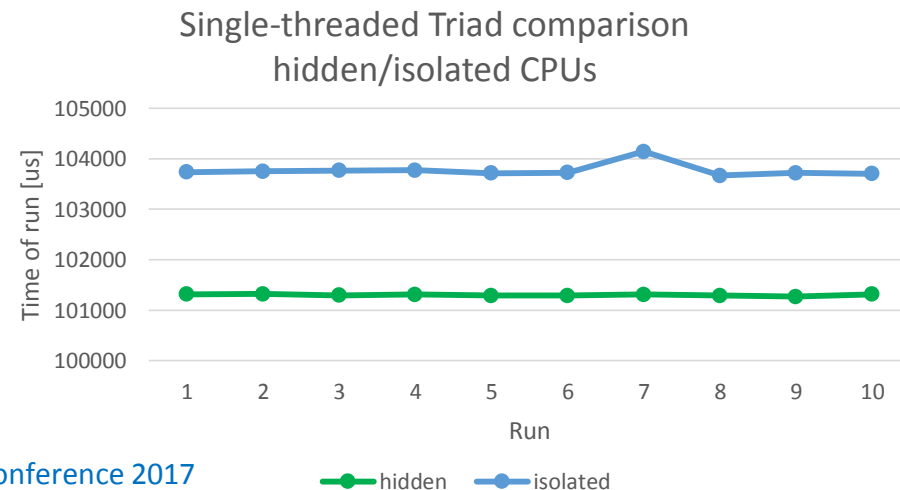
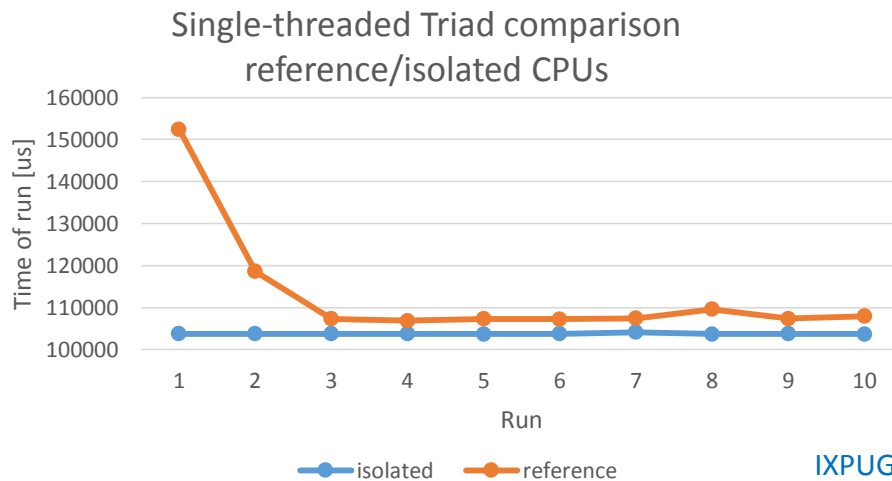
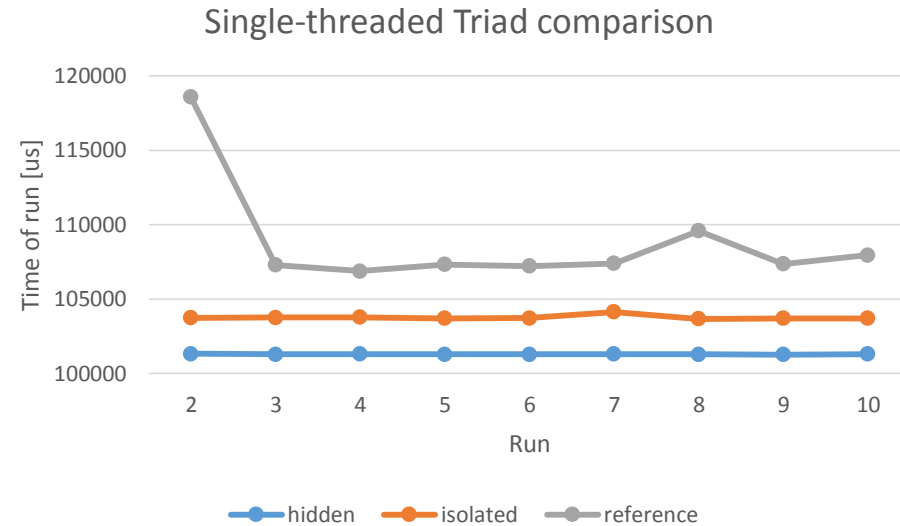
Performance gain with additional OS noise reduction

Performance stability

We ran the STREAM Triad kernel and measured time required to process 1.4 GiB using single CPU

- Hidden CPU
- Isolated CPU (nohz_full isolcpus)
- Reference OS CPU (no modifier)

On the hidden CPU the difference between fastest and slowest execution was 55 us, while for isolated and reference CPU it was 474 us and 45522 us.



Current Limitations

- The hidden CPUs cannot handle page faults.
Memory used by the computation kernel must be backed by physical pages before running it and it can't change during the run.
- System calls cannot be called from the computation kernel
- It is required to use custom synchronization between host and driver

Conclusions and Insights

It works

- The solution is stable
- x86_64 only
- It performs quite well, but there is still place for further optimizations

Place for improvements

- Improve scalability
- Tackle the limitations from the previous slide
- Benchmark more complex workloads

Thank you

The Team:

- Łukasz Odzioba lukasz.odzioba@intel.com
- Łukasz Daniluk lukasz.daniluk@intel.com
- Paweł Karczewski pawel.karczewski@intel.com
- Jarosław Kogut jaroslaw.kogut@intel.com
- Krzysztor Góreczny krzysztof.goreczny@intel.com

Legal Disclaimer & Optimization Notice

INFORMATION IN THIS DOCUMENT IS PROVIDED “AS IS”. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

Copyright © 2016, Intel Corporation. All rights reserved. Intel, Pentium, Xeon, Xeon Phi, Core, VTune, Cilk, and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

Optimization Notice

Intel’s compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804