White Paper

How to Use Huge Pages to Improve Application Performance on Intel® Xeon Phi™ Coprocessor

Introduction

To get good performance for executions on the Intel® Xeon Phi™ coprocessor, huge memory pages (2MB) are often necessary for memory allocations on the coprocessor. This is because large variables and buffers are sometimes handled more efficiently with 2MB vs. 4KB pages. With 2MB pages, TLB misses and page faults may be reduced, and there is a lower allocation cost. This article describes how to enable 2MB pages for offload executions and native executions.

Enable huge pages for offload executions

The Intel® compiler offload runtime allocates memory with 2MB pages when the size of allocation exceeds the value of the MIC_USE_2MB_BUFFERS environment variable. For example, users can specify memory allocations with size bigger than or equal to 64KB to be allocated in huge pages, by setting MIC_USE_2MB_BUFFERS=64K. Below is an excerpt from the Intel® Compiler XE 2013 User and Reference Guide:

`MIC_USE_2MB_BUFFERS`	Use 2M pages for (size > MIC_USE_2MB_BUFFERS). Pointer-based variables whose runtime length exceeds the value of this variable will be allocated in large pages. Set this variable to *integer*`B|K|M|G|T` where: • `B` = bytes • `K` = kilobytes • `M` = megabytes • `G` = gigabytes • `T` = terabytes For example: `MIC_USE_2MB_BUFFERS=64K`

Enable huge pages for native executions

It needs more work to enable 2MB pages for applications that entirely run on the coprocessor. The uOS on the coprocessor is based on Linux kernel 2.6.x, which does not include huge pages as a standard feature. Huge pages must be enabled manually, by using the mmap system call or the libhugetlbfs library. This article provides recipes for both approaches.

Use the mmap system call

The code sample below shows one way to define two functions, malloc_huge_pages and free_huge_pages, to be used for memory allocation and de-allocation on the coprocessor. Users shall use these functions to replace the malloc/free system functions in their code.

```c
#include <assert.h>
#include <stdlib.h>
#include <sys/mman.h>

#define HUGE_PAGE_SIZE (2 * 1024 * 1024)
#define ALIGN_TO_PAGE_SIZE(x) \
    (((x) + HUGE_PAGE_SIZE - 1) / HUGE_PAGE_SIZE * HUGE_PAGE_SIZE)

void *malloc_huge_pages(size_t size)
{
    // Use 1 extra page to store allocation metadata
    // (libhugetlbfs is more efficient in this regard)
    size_t real_size = ALIGN_TO_PAGE_SIZE(size + HUGE_PAGE_SIZE);
    char *ptr = (char *)mmap(NULL, real_size, PROT_READ | PROT_WRITE,
                   MAP_PRIVATE | MAP_ANONYMOUS |
                   MAP_POPULATE | MAP_HUGETLB, -1, 0);

    if (ptr == MAP_FAILED) {
            // The mmap() call failed. Try to malloc instead
            ptr = (char *)malloc(real_size);
            if (ptr == NULL) return NULL;
            real_size = 0;
    }

    // Save real_size since mmunmap() requires a size parameter
    *((size_t *)ptr) = real_size;

    // Skip the page with metadata
    return ptr + HUGE_PAGE_SIZE;
}

void free_huge_pages(void *ptr)
{
    if (ptr == NULL) return;

    // Jump back to the page with metadata
    void *real_ptr = (char *)ptr - HUGE_PAGE_SIZE;
    // Read the original allocation size
    size_t real_size = *((size_t *)real_ptr);

    assert(real_size % HUGE_PAGE_SIZE == 0);

    if (real_size != 0)
            // The memory was allocated via mmap()
            // and must be deallocated via munmap()
            munmap(real_ptr, real_size);
    else
            // The memory was allocated via malloc()
            // and must be deallocated via free()
            free(real_ptr);
}
```

By default, the uOS running on Intel® Xeon Phi™ coprocessor will allocate memory with huge pages on as-needed basis via the overcommit mechanism. To check that overcommit is enabled for huge pages execute this command from the host:

```
ssh mic0 cat /proc/sys/vm/nr_overcommit_hugepages
```

The output should be a single non-zero number which represents the limit on huge pages the kernel can allocate. The overcommit mechanism may fail if there isn't enough free memory. Another option, rather than relying on the overcommit mechanism, is to pre-allocate a pool of huge pages on the coprocessor. For example, issue this command from the host:

```
ssh mic0 "echo 800 > /proc/sys/vm/nr_hugepages"
```

Using the libhugetlbfs library

libhugetlbfs is a library providing easy access to huge pages of memory. It is a wrapper for the hugetlbfs file system. Using this approach, dynamically linked applications that call the regular malloc/calloc/realloc to requests memory allocation will get huge pages automatically. There is no need to change the code or recompile. libhugetlbfs is an open source project with the GNU LESSER GENERAL PUBLIC LICENSE. It can be downloaded from: https://sourceforge.net/project/showfiles.php?group_id=156936.

Here are the steps to use the libhugetlbfs library:

- Download the source code of the library. Build it with the Intel C/C++ compiler with the support for Intel® Xeon Phi™ coprocessor. This can be done by adding a "-mmic" flag to the CC64 variable in the Makefile, and then invoking the make command:

  ```
  make ARCH=x86_64 CC64=icc libs BUILDTYPE=NATIVEONLY
  ```

  The built library libhugetlbfs.so is found in the obj64 directory in the source tree.
- Using a utility like scp, upload libhugetlbfs.so and its dependences to the coprocessor, including libsvml.so, libintlc.so.5, libintlc.so, and libimf.so. On the coprocessor side, put them in a path that is included in the LD_LIBRARY_PATH env-variable.
- Mount the libhugetlbfs file system on the coprocessor:

  ```
  ssh mic0 "mkdir -p /mnt/hugetlbfs"
  ssh mic0 "mount -t hugetlbfs none /mnt/hugetlbfs"
  ```

- For dynamically linked applications, set the LD_PRELOAD env-variable before the execution:

```
ssh mic0 "LD_LIBRARY_PATH=/path/to/lib HUGETLB_MORECORE=yes \
LD_PRELOAD=/path/to/lib/libhugetlbfs.so ./a.out"
```

- For statically linked applications, they need to be recompiled with special options for the compiler: "-Wl,--whole-archive,libhugetlbfs.a,--no-whole-archive". Then, it can be launched by, for example:

```
ssh mic0 "HUGETLB_MORECORE=yes ./a.out"
```

- If you also want .BSS, .DATA and .TEXT sections to be placed on huge pages for your dynamically linked application you need to pass special options to compiler: "-Wl, -T./libhugetlbfs/ldscripts/elf_x86_64.xBDT -L ./libhugetlbfs/obj64/". Then, launch the execution on the coprocessor:

```
ssh mic0 "LD_LIBRARY_PATH=/path/to/lib HUGETLB_MORECORE=yes ./a.out"
```

- You can check whether memory allocations occur on the coprocessor during application execution using the following command:

```
ssh mic0 "cat /proc/meminfo | grep HugePages"
```

- The output will include lines like (from the Linux kernel documentation):

```
HugePages_Total: vvv
HugePages_Free:  www
HugePages_Rsvd:  xxx
HugePages_Surp:  yyy
Hugepagesize:    zzz kB
```

  Where:
  - HugePages_Total is the size of the pool of huge pages.
  - HugePages_Free is the number of huge pages in the pool that are not yet allocated.
  - HugePages_Rsvd is the number of huge pages for which a commitment to allocate from the pool has been made, but no allocation has yet been done. Reserved huge pages guarantee that an application will be able to allocate a huge page from the pool of huge pages at fault time.
  - HugePages_Surp is the number of huge pages in the pool above the value in /proc/sys/vm/nr_hugepages. The maximum number of surplus huge pages is controlled by /proc/sys/vm/nr_overcommit_hugepages.

Notes

- The mmap method discussed above applies to C/C++ programs only. FORTRAN programs that run natively should use the libhugetlbfs library to get huge pages for memory allocations.

Huge pages in Intel® MKL Automatic Offload

Automatic Offload is a unique offloading model provided by Intel® Math Kernel Library. This feature allows computationally intensive Intel® MKL functions to take advantages of both the host processor and the coprocessor automatically and transparently, without the need of compiler offload pragmas or code changes. In the latest Intel® MKL release (version 11.0), only a small set of level 3 BLAS functions, like GEMM, TRMM, and TRSM, support this model. Programs that involve only Automatic Offload do not need to use any of these methods to enable huge pages. The Intel® MKL runtime system already takes care of allocating memory with 2MB pages.

Notices

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

A "Mission Critical Application" is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL'S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS' FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL OR ITS SUBCONTRACTOR WAS NEGLIGENT IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL PRODUCT OR ANY OF ITS PARTS.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined". Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.
Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or go to: http://www.intel.com/design/literature.htm

Optimization Notice

http://software.intel.com/en-us/articles/optimization-notice/

Performance Notice

For more complete information about performance and benchmark results, visit www.intel.com/benchmarks