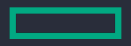
Configuring Huge Pages on LLNL MI300A Systems

El Capitan COE

October 22nd, 2025

Outline

- How to make use of Huge Pages
  - THP
  - HugeTLB (libhugetlbfs)
  - Troubleshooting
- Case Studies
- Caveats



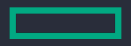
Introduction to Huge Pages

- Physical and virtual memory are partitioned into chunks called pages
  - All physical memory pages are of a fixed size – 4KB
  - Virtual memory pages can vary in size
    - **Huge pages** are virtual pages larger than physical pages – 2MB, 1GB

- The page table of a process maps its virtual pages to physical pages
  - Translation Lookaside Buffer (TLB) – hardware-based page table cache

- Huge pages may provide performance improvements to HPC applications
  - Fewer entries in page tables → faster traversal, lower memory footprint
    - Ex. **A 2MB page table entry corresponds to 512 4KB entries**
  - Reduce page faults and TLB misses

- Huge pages may incur in performance penalty due to memory fragmentation
  - Prone to waste space if only small amounts of nearby memory locations are accessed



Using THP to get Huge Pages

- **Transparent Huge Pages (THP)** is a mechanism from the Linux kernel where virtual memory pages are automatically promoted/demoted into “huge page” sizes
  - Kernel automatically tries to assign huge pages to processes with large, contiguous, anonymous memory regions
  - El Capitan systems supports 2MB THP
  - https://docs.kernel.org/admin-guide/mm/transhuge.html

- To increase the probability of the kernel using THP to map a memory region
  - Allocation must be at least as large as a huge page
  - Memory region must be aligned to huge page size

1. Configure Flux allocation with THP
   - Interactive `flux alloc … --setattr=thp=always <command>`
   - Batch `flux batch … job.sh ; #flux: --setattr=thp=always`
2. Run job

```
flux run … <command>
```

NOTE: THP will be enabled for all commands executed in the Flux allocation

Using HugeTLB (libhugetlbfs) to get Huge Pages

- **HugeTLB** is a mechanism where huge pages are reserved in a persistent pool for application use
  - Huge page pool resides in a special mounted filesystem of type *hugetlbfs*
  - Application needs to be configured to make use of HugeTLB
  - https://docs.kernel.org/admin-guide/mm/hugetlbpage.html

> **NOTE**: There is a single HugeTLB pool per compute node which is shared among all the node's processes

- **libhugetlbfs** is a library that interacts with the *hugetlbfs* filesystem
  - Automatically intercepts malloc calls via the *morecore* API
  - El Capitan systems have */lib64/libhugetlbfs.so*

> **NOTE**: The capability of libhugetlbfs is targeted at system allocated memory, not hipMalloc'ed memory

Using libhugetlbfs to get Huge Pages

WARN: Do not use Cray huge pages modules `module load craype-hugepages2M` nor set other `HUGETLB` environment variables as they may interfere with this guidance.

1. Set up linker flags and build application

```
<compiler> … -lhugetlbfs <sourcefile>
```

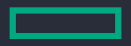
2. Configure Flux allocation with the *hugetlbfs* filesystem
   - Interactive `flux alloc … --setattr=hugepages=512GB <command>`
   - Batch `flux batch job.sh ; #flux: --setattr=hugepages=512GB`

3. Run application with libhugetlbfs configured via environment variables

Environment variable	Description
`HUGETLB_DEFAULT_PAGE_SIZE=2M`	directs libhugetlbfs to use this size-specific *hugetlbfs* filesystem
`HUGETLB_MORECORE=yes`	enable/disable runtime libhugetlbfs malloc hooks
`HUGETLB_VERBOSE=2`	controls verbosity of libhugetlbfs (2 reports errors and warnings)

   - COE recommends setting environment variables directly to the Flux command via the `--env=` option

```
flux run … --env=HUGETLB_MORECORE=yes <command>
```



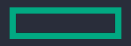
Using THP and libhugetlbfs in Conjunction

THP and **libhugetlbfs** can be used together

```
flux alloc … --setattr=thp=always --setattr=hugepages=512GB <command>
```

- libhugetlbfs takes precedence because it intercepts allocation calls prior to the Linux kernel's THP logic
- Some system allocators do not go through glibc's malloc hooks thus bypassing libhugetlbfs
  - This largely depends on the specific allocation API

- These mechanisms are targeted at system allocated memory
  - Works with C++ and Fortran
  - Applies to OpenMP target offload when used with unified shared memory
  - hipMalloc already requests 2MB huge pages regardless if these mechanisms are enabled



Check if Huge Pages are enabled in a Flux Allocation

- **THP**
  - Enabled if `[always]`, disabled if `[never]`

```
cat /sys/kernel/mm/transparent_hugepage/enabled
        [always] madvise never
```

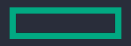
  - To view the configured huge page size

```
grep Hugepagesize /proc/meminfo
        Hugepagesize:   2048 KB
```

- **HugeTLB (libhugetlbfs)**
  - Enabled if a *hugetlbfs* mount point exists, else no output is shown

```
grep hugetlbfs /proc/mounts
        hugetlbfs /dev/hugepages hugetlbfs rw,relatime,pagesize=2M 0 0
```

  - The huge page size is specified in the "`pagesize`" field



Check if Huge Pages are used by MPI

- For MPI, the NIC counter summaries indicate if huge pages were utilized

  `MPICH_OFI_CXI_COUNTER_REPORT=2` (or higher)

  https://cpe.ext.hpe.com/docs/latest/mpt/mpich/intro_mpi.html#mpich-ofi-environment-variables

```
0: MPICH Slingshot CXI Counter Summary:
0: Counter                              Samples          Min          (/s)         Mean          (/s)          Max          (/s)
0: atu_cache_evictions                        2           12           0.2           22           0.4           31           0.6
0: atu_cache_hit_base_page_size_0             8      2813498      50241.0      2815816      50282.4      2816806      50300.1
0: atu_cache_hit_derivative1_page_size_0      8     14132798     252371.4     14136255     252433.1     14144239     252575.7
```

- The presence and non-zero value of `atu_cache_hit_derivative1_page_size_0` counter is an indicator that allocations backed by huge pages are used throughout MPI.

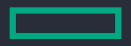
libhugetlbfs Diagnostics

- libhugetlbfs has options that can increase its verbosity
  - Useful for verifying that huge pages are being used and diagnosing issues
- The environment variable `HUGETLB_VERBOSE` controls verbosity

Verbosity	Report type
1	only errors
2	errors and warnings
3	same as level-2 plus extra information • check if libhugetlbfs set up the malloc *morecore* hooks • check if huge pages are used during allocations
4 or 99	same as level-3, output is prefixed with hostname and TID

NOTE: If too many warnings are generated by libhugetlbfs, you can disable warnings and set verbosity to only report errors.

- Reports extra debug information via `HUGETLB_DEBUG=yes`
  - Also provides the same information as with `HUGETLB_VERBOSE=4`
- You do not need to relink your application with a different library
  - Simply set these prior to running your application



libhugetlbfs Sample Output

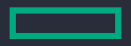
- libhugetlbfs is enabled

```
INFO: setup_morecore(): heapaddr = 0xc00000
```

- An allocation is making use of huge pages from HugeTLB pool
  - libhugetlbfs allocation size doesn't necessarily needs to match the requested size
  - libhugetlbfs Multiple allocations can be used to satisfy a single request

```
INFO: hugetlbfs_morecore(3129344) = …
INFO: heapbase = 0xc00000, heaptop = 0xc00000, mapsize = 0, delta=3129344
INFO: Attempting to map 4194304 bytes
INFO: … = 0xc00000
```

Huge page mapping succeeded

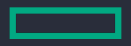


Troubleshooting libhugetlbfs

Common warnings you may encounter while using libhugetlbfs

Configuration	Warning message	Description	Solution
`flux alloc …`	Hugepage size 2097152 unavailable	A hugetlbfs filesystem of page size 2MB is unavailable	Use Flux option `--setattr=hugepages=512GB`
`HUGETLB_MORECORE=no`	HUGETLB_MORECORE=no, not setting up morecore	libhugetlbfs is linked but not enabled	Set environment variable `HUGETLB_MORECORE=yes` when running application
`HUGETLB_MORECORE=yes`	New heap segment map at 0x10e41200000 failed: Cannot allocate memory	During an allocation event, either: • A hugetlbfs filesystem is unavailable • Insufficient huge pages available in hugetlbfs' pool	N/A Allocation falls back to default 4KB page size.
`HUGETLB_MORECORE=yes`	libhugetlbfs:WARNING: Heap shrinking is turned off	Many huge page allocations have succeeded, leaving insufficient space for more. libhugetlbfs can attempt to shrink the heap to allow more space but shrinking is disabled.	

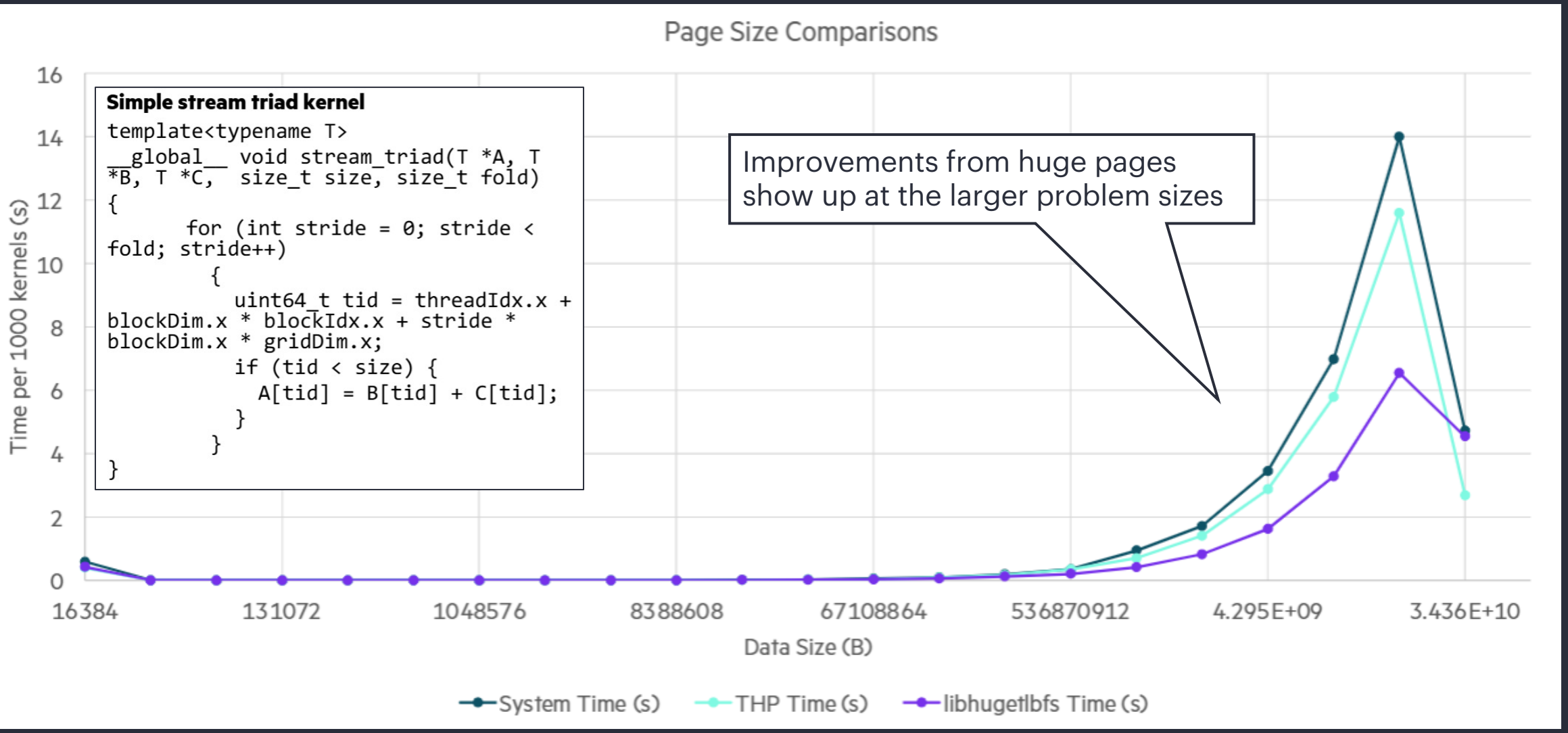
Contact the COE if you are unsure on how to troubleshoot specific issues



Case Study: Stream-Triad Kernel

Simple stream triad kernel

```
template<typename T>
__global__ void stream_triad(T *A, T
*B, T *C,  size_t size, size_t fold)
{
       for (int stride = 0; stride <
fold; stride++)
         {
           uint64_t tid = threadIdx.x +
blockDim.x * blockIdx.x + stride *
blockDim.x * gridDim.x;
           if (tid < size) {
             A[tid] = B[tid] + C[tid];
           }
         }
}
```

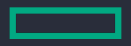


Case Study: MARBL

Simulation of the N210808 National Ignition Facility experiment with 3T multigroup radiation diffusion

- Using default 4KB pages
  - Memory registration cache of the communication layer maxed out causing performance degradation for long runs
    - Too many cache entries
    - Using default max cache size

- Using 2MB huge pages via libhugetlbfs
  - Memory registration cache remained manageable along with stable performance for long runs
    - O(100) fewer entries with 2MB pages versus 4KB pages
    - A 2MB cache entry can correspond up to 512 4KB cache entries



Subtleties and Things to be Aware of

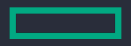
- **If you are explicitly doing mmap calls**
  - Contact the COE if you need explicit control of huge pages requests
  - THP requests huge pages from the kernel using `madvise()`

```
mmap(…, MAP_PRIVATE | MAP_ANONYMOUS, …)
madvise(…, MADV_HUGEPAGE)
```

  - libhugetlbfs requests huge pages from the reservation pool using `mmap()`

```
mmap(…, MAP_PRIVATE | MAP_ANONYMOUS | MAP_HUGETLB | MAP_HUGE_2MB, …)
```

- **If you are explicitly changing malloc settings**
  - Contact the COE if you need to configure malloc settings in your applications
  - libhugetlbfs changes default malloc settings
    - Some allocator options are more appropriate for huge pages
    - Default options are also dynamic
  - Some compilers such as CCE also optimize malloc settings



Recap

- Huge pages may provide performance improvements to HPC applications
  - COE considers the use of huge pages as a best practice on MI300A systems
  - https://hpc.llnl.gov/documentation/user-guides/using-el-capitan-systems/introduction-and-quickstart/pro-tips

THP

- Configure a Flux allocation, then run application

```
flux alloc … --setattr=thp=always <command>
flux run … <command>
```

HugeTLB (libhugetlbfs)

- Link the library when building application

```
<compiler> … -lhugetlbfs <sourcefile>
```

- Configure Flux allocation and job's environment, then run application

```
flux alloc … --setattr=hugepages=512GB <command>
flux run … --env=HUGETLB_DEFAULT_PAGE_SIZE=2M \
           --env=HUGETLB_MORECORE=yes --env=HUGETLB_VERBOSE=2 <command>
```

COE is interested in learning the effects of huge pages in your applications

Thank You

Eduardo Ponce, eduardo.ponce@hpe.com