Using the Cray Multi-Tier Memory Analysis Tool

Cray Inc.

This document provides instructions and guidance on how to use the Cray multi-tier memory analysis tool for selectively allocating memory bandwidth-sensitive arrays into KNL's MCDRAM using the Cray compiler HBM allocation directives. The goal of this tool is to assist the user with more effectively using the different tiers of memory on KNL, resulting in reduced overall application execution time.

When deciding how to best use KNL's MCDRAM for an application, first determine if selective allocation into MCDRAM is right for your application by trying the following experiments from least to most effort.

1) Run in quad/cache mode

2) Run in quad/flat mode, but don't allocate anything into MCDRAM (helps determine how sensitive the program is to MCDRAM)

3) Check to see if program fits in MCDRAM

   a) Use `numactl --membind=1` to force all allocations into MCDRAM

4) Use CCE, CrayPAT and Reveal to selectively allocate bandwidth-sensitive arrays into MCDRAM

The memory analysis tool is available for use on Cray XC systems with Haswell/Broadwell[1] or KNL processors. There are pros and cons associated with using the tool on the different processors, which are discussed in this document.

Minimum software versions needed to use the memory analysis tool:

- cce/8.5.6
- perftools-base/6.4.4

The following steps should be used to collect memory traffic information on the different supported processors.

[1] Note that although the tool is supported on Haswell/Broadwell, the recommendations from the tool for selective allocation are still for KNL, as KNL alone has MCDRAM. The support for Haswell/Broadwell is simply to allow the use of more comprehensive counters by the tool.

Collecting Memory Traffic Data for Bandwidth Sensitivity Analysis

1) Load the software and enable a data collection experiment.

```
$ module load PrgEnv-cray

$ module load craype-haswell
     or
$ module load craype-mic-knl

$ module load perftools-base
$ module load perftools-lite-hbm
```

2) Build the application dynamically with a CCE program library

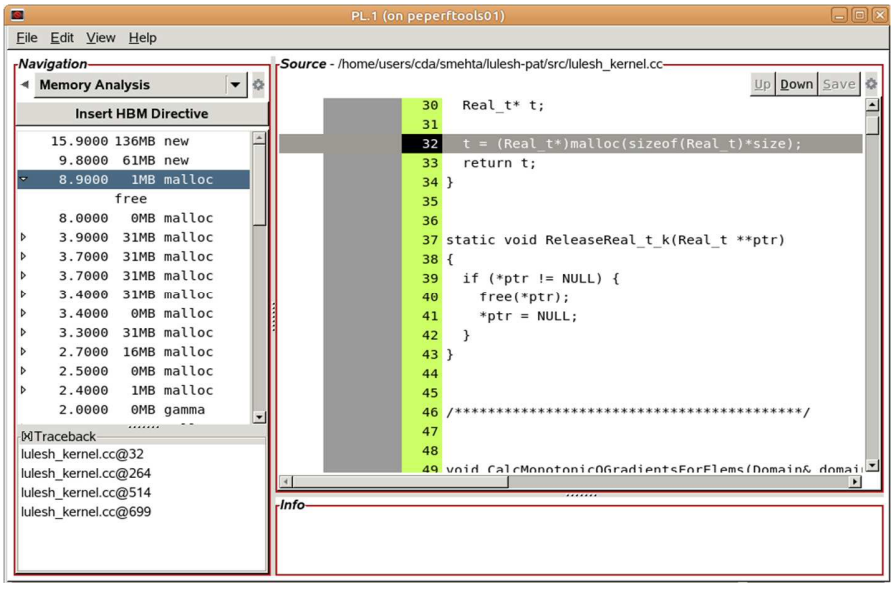
```
$ cc –dynamic –h pl=/full_path/my_pl.pl … my_program.c
```

3) Run the binary, **using all of the cores on the node,** which pushes the bandwidth utilization to the max. If you are running on KNL, run on nodes configured in **`split` mode** (25% cache). The benefits of this are discussed later in this document.

4) Run Reveal to review results, find data allocation sites, and insert CCE HBM allocation directives into source. The list of bandwidth-sensitive arrays is also available in the `*.rpt` file.

```
$ reveal my_pl.pl my_program.ap2
```

The following screen shot shows the ranked list of arrays that are candidates for allocation into MCDRAM. Clicking on an array will display the allocation site in the source.



Using Haswell or Broadwell Nodes for Data Collection

Pros:
The statistics obtained on Haswell are more likely to reliably track memory traffic since the prefetcher's activity will not affect the statistics collected.

Cons:
Haswell is not KNL. TLB misses (which result from strided accesses) are likely to be more expensive on KNL than Haswell due to the absence of an L3 cache. In order to prevent this performance degradation in KNL, it may be beneficial to initially run in `split` mode (as recommended above), where the nodes are configured with 25% cache.

Using KNL Nodes for Data Collection

Pros:
The statistics are collected on the architecture where the final application will be run. The application sees the exact cache sizes and TLB sizes, which has the potential to be more accurate.

Cons:
A fairly important drawback of collecting statistics on KNL is that the memory traffic due to the hardware prefetcher is untracked, which can cause anomalies in the reported data. If a memory bandwidth sensitive data structure is prefetched, it will rank lower in the list of bandwidth-consuming arrays than it should, because the untracked prefetching hides the accesses. (Remember that prefetching only helps latency and not bandwidth.) This means that some arrays that should be placed in HBM may be missed.

Allocating Select Arrays in MCDRAM

CCE HBM directives based on memory traffic data can be inserted either manually or through Reveal. In order for the directives to take effect, the user must link dynamically and load the `cray-memkind` module. To check if the directives have actually taken effect, set the CCE environment variable, `CRAYMEM_DEBUG`, to '`1`' or '`2`'. This will produce messages at run-time showing allocation information similar to the following:

```
posix_memalign(allocator="bandwidth" size=…)
```

"`bandwidth`" refers to an allocation in MCDRAM. An allocator marked as "`normal`" implies that the arrays specified on the HBM directive were allocated in DDR. Allocating to DDR can occur if the application was not built dynamically, or if the hardware does not support or is not configured to use MCDRAM.

Note that use of the directives will convert a stack allocation to a heap allocation, which can have significant overhead as compared to a stack allocation.

Running on KNL with HBM Allocation Directives

After inserting the desired allocation directives, we recommend trying two modes on KNL. First, run in `split` mode (25% cache) to see if the use of selective allocation into MCDRAM performs better than pure `cache` mode. This will better mimic Haswell with the L3 cache. The application still has 75% of MCDRAM available for allocation, and this may help programs that have pronounced TLB misses, as they will go to cache rather than to the slower DDR. (See more discussion in the Further Guidance section.) . If there is no improvement, try running in `flat` mode with the inserted allocation directives, and check for overall application performance improvement.

An application may not see a performance improvement over pure `cache` mode if its problem size/memory footprint is small, or if there are many important arrays that cannot all fit into the limited HBM space.

Q&A

Q: What software is required to use the memory analysis tool?
A: The minimum software required to use the tool is:

```
cce/8.5.6
perftools-base/6.4.4
```
(Reveal is available with perftools-base/6.4.4)

Q: What about stores?
A: The memory traffic information collected on either Haswell or KNL only includes loads. Even though stores are not tracked, data stored is typically used (read) later in the program, at which point the accesses are tracked. In most cases, if the loop where an array is written is bandwidth intensive, then the loop(s) where it is later read is bandwidth-intensive, too. If the user examines the code and finds sites that are identified as hot by load tracking, and sees stores into arrays that are not in the object load profile (because there weren't a significant number of loads from those arrays that missed cache either there or at other sites), an additional experiment can be performed on Haswell or Broadwell by setting the following runtime environment variable and re-executing the instrumented binary from the `perftools-lite-hbm` experiment:

```
$ export PAT_RT_PERFCTR=hbm_stores
```

Q: What is the difference between '`sample %`' and '`weight %`' in the .rpt file?
A: The '`sample %`' gives the percentage of total memory requests attributed to a particular array, and the '`weight %`' uses the same number and weighs it by a measure of the bandwidth consumption of that array. That is, if the '`weight %`' of an array is larger than '`sample %`', it implies that that array is being used in a bandwidth-intensive region of

the application code, and is therefore ranked higher (by weight) to indicate its importance for allocation in HBM.

Q: Does the memory tracking incur a lot of overhead?
A: Good statistics are obtainable from the `perftools-lite-hbm` experiment with a very small percentage overhead, and there is typically a minimal increase in execution time over non-instrumented programs. Occasional high overhead has been observed, however, where time is spent in the tool that can end up skewing results. This overhead is being investigated and will be addressed in a future release. Minimal tracking overhead can be confirmed by checking the percent of samples dedicated to memory tracking in pat_report output. If the user sees that the number of samples collected in the non-user code dedicated to memory tracking is more than 5%, then the overhead is too high and recommendations from the tool should be ignored.

Q: How does the memory tracking scale to larger nodes?
A: Since the overall overhead is typically very low, this tool can be used on applications that run on large numbers of nodes.

Further Guidance

- If an application has large strides, there may be more page table walks. Since KNL does not have an L3 cache, this amounts to memory accesses. There can be up to 3 memory accesses for every page table walk, which can be very detrimental to performance. In such cases, running in cache mode has a natural advantage since the memory accesses are serviced by the cache is much better performing than DRAM access. In such cases, it is beneficial to have at least 25% of MCDRAM configured as cache (i.e. the split mode on KNL) and use the remaining HBM for static allocations.

- It may happen that upon adding directives, even though the array in concern may be getting allocated in HBM, the performance degrades. This may be due to a stack allocation getting converted to a heap allocation (after the addition of the directive). This happens with local variables inside a routine. In such cases, either the local variables should not be allocated in HBM or if they are, they should be declared as global variables such as inside a module in Fortran.

- Running the `perftools-lite-hbm` experiment on either Haswell or KNL provides good feedback on memory bandwidth sensitive objects in a program, and the user may use either. However, if the user knows that a code is prefetch friendly (i.e. is dominated by streaming accesses), running the memory tracking experiment on Haswell or Broadwell is recommended. If the user knows that a code has strided accesses and is not prefetch friendly in the critical loops, collecting statistics on KNL is recommended. Based on early experience with the tool, we have found that collecting data on Haswell or Broadwell to be slightly more useful. Of course, the best may be to collect statistics on both Haswell and KNL and use the combined results to get the most complete picture.