Memory Exercise

Adrian Jackson

24 October 2016

1 Introduction

In this set of exercises we will investigate the memory available on the Knights Landing (KNL) processors. We will investigate the different memory configurations (cache and flat modes), and the different static methods for accessing MCDRAM. We will also look at programming approaches that can be used to exploit MCDRAM in flat mode.

2 Basic memory modes

To investigate the memory setup and the basic memory modes on the KNL we are going to use the well-known STREAMS benchmark for measuring memory bandwidth. You should have already download the source code we are going to be using in the introductory exercise, but if you didn't you can download from:

https://www.archer.ac.uk/training/course-material/2016/11/161101_KNL_EPCC/Exercises/MemorySource.tar.gz

You should be able Unpack it with the command `tar zxvf MemorySource.tar.gz`

The example code can be found in `MemorySource/*/`, where * is either C or Fortran, according to your preference. There is also a small application, `xthi.c`, in `MemorySource/xthi` which we will use to investigate thread placement on the KNL.

2.1 Cache mode

Use the Makefile to compile the code. Run the compiled application using the batch system using the batch script provided (i.e. `qsub runstream.sh`). We are looking to run on a single node that is configured in cache mode. The application is currently configured to use ~2GB of memory. Run in cache mode using 256 threads (the provided run script should already do this) and see what performance you get.

Now, alter the code, changing the size of the array used in the calculations. This is the variable `n=100000000` in the Fortran code (in `stream_d.f`) or `# define N 100000000` in the C code (in `stream_d.c`). Try the following values:

Size of N	Size of array in GB	Average memory bandwidth
100000000	~2.3	
1000000000	~23	
2000000000	~46	
3000000000	~68	

Remember, you'll need to re-compile your code between each change of the variable to ensure the new array size is used.

2.2 Flat mode

Now we want to investigate the performance we can get controlling the MCDRAM manually. The first thing we will do will be to target MCDRAM using the bulk memory policy approaches (i.e. through `numactl`).

Modify the batch submission script you are using to target KNLs that are configured in quad flat mode, i.e. change:

```
#PBS -l select1:aoe=quad_100
```

to:

```
#PBS -l select=1:aoe=quad_0
```

Now re-run the tests you performed in cache mode:

Size of N	Size of array in GB	Average memory bandwidth
100000000	~2.3	
1000000000	~23	
2000000000	~46	
3000000000	~68	

Note, we are now using only main memory (also known as DRAM or DDR). How does the bandwidth compare to that of cache mode?

To target MCDRAM we are going to use the numactl program. Change the aprun command in your submission script from this:

```
aprun -n 1 -j 4 -d $OMP_NUM_THREADS -cc depth ./stream_d
```

To this:

```
aprun -n 1 -j 4 -d $OMP_NUM_THREADS -cc depth numactl -m 1 ./stream_d
```

Change the size of the array you are using to the initial size (100000000), re-compile, and run. This will force all memory used by the application to be allocated in the MCDRAM. What performance do you get now?

Now increase the size of the array to 1000000000 and re-run. What happens? To fix these issues we can move from the fixed memory choice to a preferred memory choice, i.e.

```
aprun -n 1 -j 4 -d $OMP_NUM_THREADS -cc depth numactl -p 1 ./stream_d
```

Re-run with your submission script updated and see what performance you get.

As we are using preferred memory mode then allocates that can be satisfied in the MCDRAM will be done there, and if there is not enough space the allocate will target main memory. In this case the `a` and `b` arrays will be allocated in MCDRAM and `c` array in main memory.

Now we going to try to manually use the MCDRAM from your application. The Cray has a module to enable you to use this, you should do the following:

```
module load cray-memkind
```

Now alter the code to use either `hbw_malloc`/`hbw_free` (for C) or the Intel `fastmem` directives (Fortran). Try allocating them all on MCDRAM, or just some of them. What happens if you increase the size of array and you run out of MCDRAM?

2.2.1 memkind on non-Cray systems

If you are running these exercises on a non-Cray system we have provided a version of memkind in the source you can use. To do this you will need to alter your makefile, changing:

```
LDFLAGS=
```

To:

```
LDFLAGS=-lmemkind
```

You will also need to setup access to memkind by running this on the command line:

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:~/MemorySource/memkind/lib
```

2.3 Thread placement

Now we're going to look at thread placement on the KNL and what affect this can have on memory performance. You can use the xthi program to investigate thread placement. If you run this it will print out how many threads are used and what core they are running on. Remember that the KNL's we are using have 256 logical cores, 64 physical cores, and the first 64 logical cores (numbers 0-63) map to the physical cores, with hyperthreads on each core mapping in a round robin fashion for the remaining logical cores (i.e. physical core 0 has virtual cores 0,64,128,192, physical core 1 has virtual cores 1,65,129,193, etc…).

2.3.1 Non-Cray System

The default thread placement on KNLs that are using Intel's KMP affinity setup is scattered. This means it places threads on the physical cores first, then if there are more threads to be placed it places them on the hyperthreads until everything is filled up. You can change this to compact, where all hyperthreads on the first physical core are filled, then the next physical core is used, etc…, or to more explicit thread placement.

You can investigate the placement settings by running `xthi`. Compile and run `xthi` (there is `makefile` provided to build `xthi`), using various numbers of threads to see where threads are placed by default.

Now we want to investigate thread placement on memory performance. We are going to run the streams benchmark (you should use the original array size) on a KNL configured in flat mode using a range of different thread counts. Initially we are not using MCDRAM. Firstly run with 1 thread and see what memory performance you get. Now increase to 4, 8, 16, and 64 threads. What performance do you see?

Now try running on 4 threads but this time set this environment variable as well:

```
export KMP_AFFINITY=granularity=fine,proclist=[0,16,32,48],explicit
```

What impact does this have on performance? Try 8 threads and this environment variable:

```
export KMP_AFFINITY=granularity=fine,proclist=[0,1,16,17,32,33,48,49],explicit
```

Now, try running using the MCDRAM instead (using `numactl –m 1` to force the memory to be allocated on MCDRAM). Start with 1 thread and normal thread placement, then try 4, 16, and 64 threads. Then try specifying placement like in the examples above. Does it make any difference to the performance for MCDRAM? Remember, this may be a product of cluster mode, so may be different for different cluster modes.

2.3.2 Cray System

The default thread placement on the Cray XC40 KNLs is controlled through aprun. This means, by default, it places threads on the physical cores first, then if there are more threads to be placed it places them on the hyperthreads until everything is filled up.

You can investigate the placement settings by running `xthi`. Compile and run `xthi` (there is `makefile` provided to build `xthi`), using various numbers of threads to see where threads are placed by default.

Now we want to investigate thread placement on memory performance. We are going to run the streams benchmark (you should use the original array size) on a KNL configured in flat mode using a range of different thread counts. Initially we are not using MCDRAM. Firstly run with 1 thread and see what memory performance you get. To do this you will need to change your batch script, setting `OMP_NUM_THREADS=1`.

Now increase to 4, 8, 16, and 64 threads. What performance do you see?

Now try running on 4 threads but this time set this environment variable as well:

```
export KMP_AFFINITY=disabled
```

This turns of the Intel or OpenMP default binding and allows the Cray aprun command to control thread binding. Also changing your aprun line to this:

```
aprun –n 1 –cc 0,x,16,32,48 ./stream_d
```

Note: the **x** in the above command is required to fix an issue with the Intel compiler and the Cray aprun command. If you use the Cray compile this **x** is not required.

What impact does this have on performance? Try 8 threads and this aprun line:

```
aprun –n 1 –cc 0,x,8,16,24,32,40,48,54 ./stream_d
```

Now, try running using the MCDRAM instead (using `numactl –m 1` to force the memory to be allocated on MCDRAM). Start with 1 thread and normal thread placement, then try 4, 16, and 64 threads. Then try specifying placement like in the examples above. Does it make any difference to the performance for MCDRAM? Remember, this may be affected by the cluster mode in use, so may be different for different cluster modes.