

Use the following references for this assignment. For a description of the Phi see <http://www.ece.lsu.edu/gp/refs/rahman-phi-book.pdf>. For a description of the Phi instructions see <http://www.ece.lsu.edu/gp/refs/xeon-phi-isa-ref-manual.pdf>. For use of the Intel compiler (used in the homework) visit <https://software.intel.com/en-us/node/459680> or search for “Intel C++ Compiler Reference.”.

Problem 0: This assignment requires a system with a Xeon Phi. If necessary, follow the instructions for class account setup and homework workflow on the course procedures page <http://www.ece.lsu.edu/gp/proc.html>. Be sure to update your copy of the repo with the command “git pull”.

The assignment file, `hw05.cc`, contains three versions of a kernel for computing the sum of sections of a scattered array. Data is in array `data_array`, the element numbers to select are in array `idx_array`. The kernel adds up sections of length `clength`, which is a compile-time constant, and writes the sum to output array `sum_array`. Initially each kernel is nearly identical, they will be modified as part of this assignment.

Compile and run the program. The makefile will generate host and MIC assembly language versions of the code, in files `hw05.s` and `hw05MIC.s` respectively.

The program starts by printing the number of Phis available, then it prints the number of cores on the CPU and Phi. After that it prints the size of the Data and Index arrays.

The program runs each kernel multiple times, in launch configurations with increasing number of threads. (The sample below does not show the entire output.)

```
Number of accelerator devices installed: 1
Num cores (cpu,phi) (12,57)
Data array size:      16384 elements,   64.0 kiB.
Index array size:    16777216 elements, 64.0 MiB.
```

Running kernel `sums_0`.

Num Tds	Time s	Data GB/s	Pct	
28	30850.887	2.31	1.0%	0
56	15776.157	4.52	1.9%	0
84	11402.369	6.25	2.6%	00

The timing for each configuration is printed, along with a possibly incorrect percentage of peak data bandwidth realized by the code. The peak bandwidth is shown as a number and as a line of digits (zeros above). If the percentage, under `Pct`, is 100% that means the kernel is transferring data at the maximum rate (based upon certain assumptions).

If there are any difficulties running the code ask for help. There is nothing to submit for this first problem.

Problem 1: As discussed in class, Phi relies on prefetch (as do CPUs) whereas current NVIDIA processors use a massive amount of threads to hide memory latency.

Notice that in kernel `sums_0` variables such as `dapp->num_pieces` are assigned to local variables. One reason is simply readability: the code is less cluttered with the `dapp->` parts removed. Another reason is compiler timidity. We know that `dapp->num_pieces` won't change, but the compiler can't rule that out. (Don't forget that this is parallel code.) In contrast, the compiler can be sure that `num_pieces` won't change.

Sharp-eyed students might have noticed that one `dapp->` still remains in the loop body in `sums_0`, though `sums_1` is done correctly in that `dapp->` is removed. Run the code and compare the performance of `sums_0` and `sums_1`; `sums_1` should be faster.

Explain the difference by examining the assembler code for the two routines (before modifying `sums_1`). *Hint: `prefetch` is used for store addresses.*

- (a) Based on the assembler code, why does removing `dapp->` make the code run faster?
- (b) How does the presence of `dapp->` impede the code.

Problem 2: Kernel `sums_0` and `sums_1` access array elements in an order appropriate for NVIDIA GPUs.

- (a) Modify the code in `sums_1` so that access ordering is more appropriate for Phi.
- (b) Indicate the performance improvement obtained.

Problem 3: The `ptri512` and `ptrf512` data types contain an attribute that indicates the pointer value is a multiple of 64 bytes (512 bits). (They are declared using a typedef near the top of the file.) The compiler can use this information to generate better code.

- (a) In `sums_1` (the one re-written for the previous problem) use these types in the place or places where they will make a difference.
- (b) There is at least one array for which the special type is not needed (even though the array address has the proper alignment). Identify the array and explain why it is not needed.

Problem 4: The Intel compiler should unroll the `piece` loop, even if the iteration (trip) count is not known at compile time. However, the compiler will not group the loads to `idx_array` at the top of the unrolled loop because of possible aliasing with `sum_array`.

When we encountered a similar problem in CUDA code we used a local array for `sum_array`, removing the fear of aliasing from the compiler's decision making process, and so enabling it to place the `idx_array` loads at the top of the loop.

Based on the Phi microarchitecture, explain why this would not make much of a difference for Phi.

Note: Do not write any code for this problem, just answer the question. Another note: It actually does realize about 10% speedup, perhaps due to other scheduling benefits.

Problem 5: Consider the two ways of accessing array elements, the GPU ordering used in `sums_0` and the Phi ordering in a correctly solved `sums_1`. Answer the questions below for `sums_0` and `sums_1` and take into account the value of `clength`, which is 16, and assume the program is run with the default parameters.

- (a) Explain the difference in performance of the two orderings for access to `idx_array`.
- (b) Explain the difference in performance of the two orderings for access to `data_array`.
- (c) Explain the difference in performance of the two orderings for access to `sum_array`.

Problem 6: One advantage of the Phi over the Kepler is the 512 kiB L2 cache, providing more medium-speed storage.

- (a) Look at the impact on performance when varying the size of the data array. The data array size can be specified with the first argument; the number of elements in the data array is $1024 \times a$, where a is the first argument. The default values is 16, so running `hw05 16` gives the default data size.

(b) Explain at which data-array sizes you expect performance to change, and contrast it with where performance does change.

Problem 7: The code for computing the sum uses multiple vector instructions, which is inefficient compared to the way the sum is found on NVIDIA GPUs. It is inefficient because the number of lanes computing the sum drops by half in each of the four steps. In contrast, each thread in a CUDA warp is doing useful work when computing the sum in the Homework 3 version of this code.

(a) Modify `sums_2` so that the compiler will efficiently use a vector add to compute the sum. This can be done by rearranging elements so that each lane of a vector is from a different piece (value of `piece`), rather than a different component of a piece (value of `i`). (Note: Don't worry if the code runs more slowly than `sums_1`.)

(b) If the code is not as fast, explain why. Do so by examining the assembly code.