CompSci 590.03: Introduction to Parallel Computing

Homework 3 - Stencil Pattern (10/05/15)

Lecturer: Alvin R. Lebeck *Scribes: Mengke Lian, Kai Fan, Chaoren Liu*

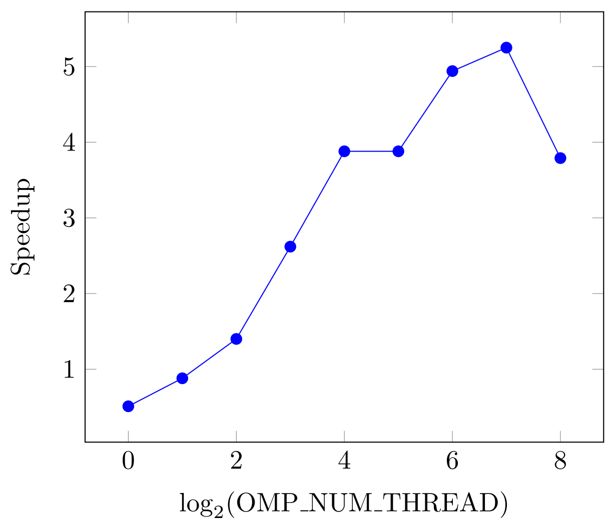
Contribution distribution:

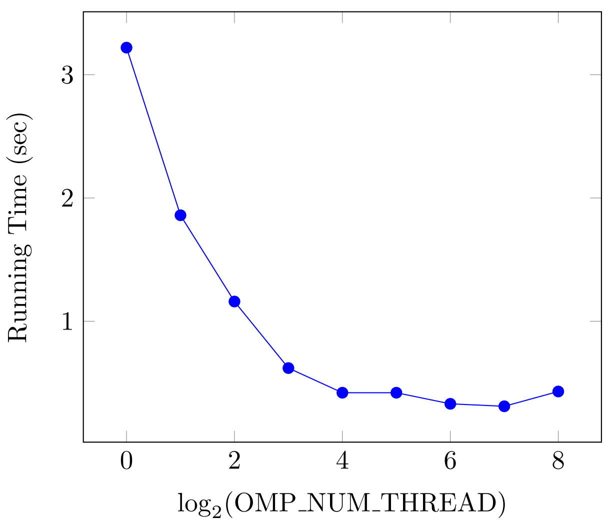
- Mengke Lian: implemented the stencil pattern for OpenMP, offload and CUDA. In CUDA implementation, I used 2D shared array. However, compared to result of other members, it seems that 1D shared array is slightly faster.

- Kai Fan: implemented the stencil pattern in three frameworks. I also rewrote the function `filter` in more maintainable manner, with inner loop by getting rid of redundant code. In addition, I implemented the function `filter` by considering the most outer boundary pixel, which is omitted in the provided serial code.

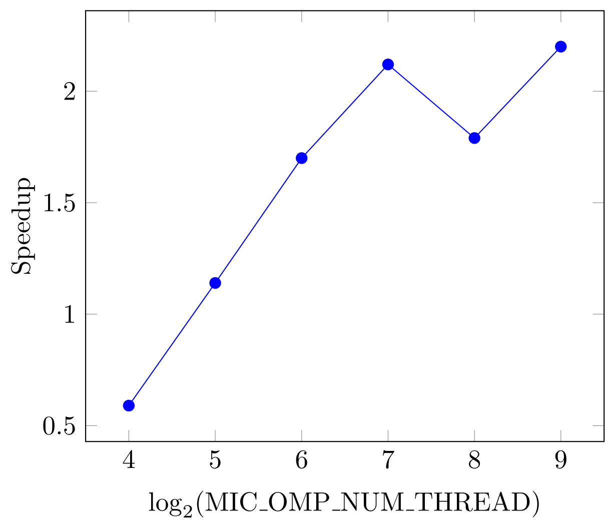
- Chaoren Liu: I implemented the stencil pattern with cuda. I used the shared memory model discussed in class. The cummulative error is the same as the serial code.

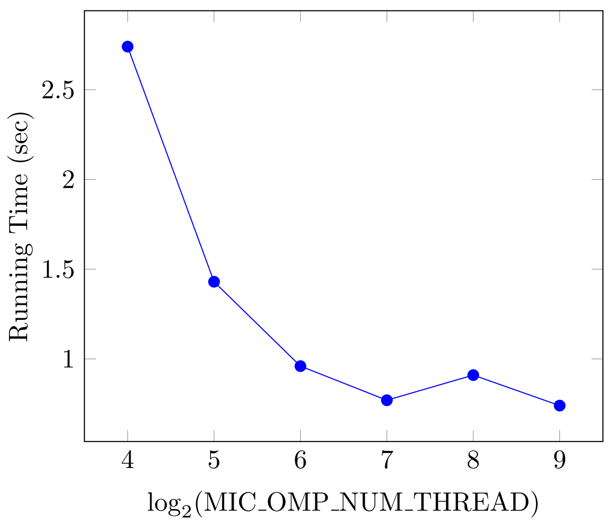
The provided serial code for running `bigimg.bmp` is about 1.629s. The curves for OpenMP only program is shown below:



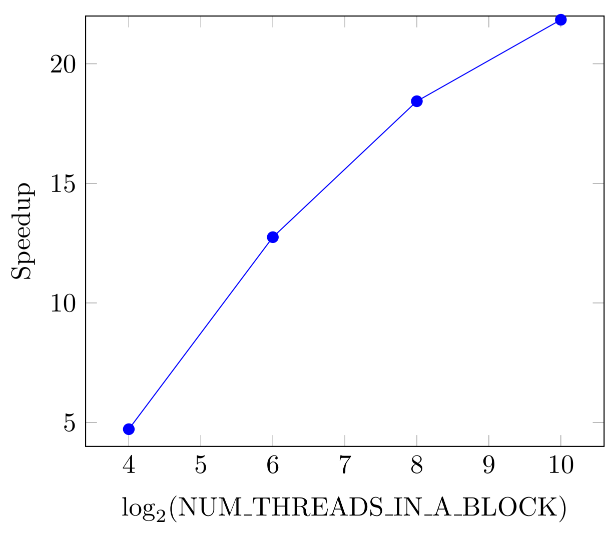


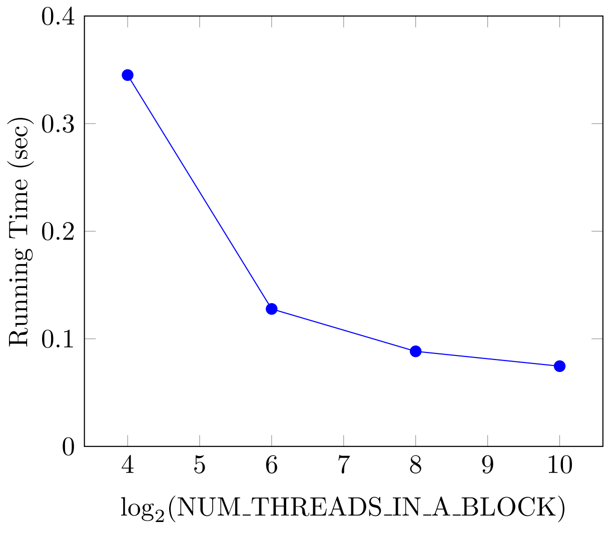
For curves of offload programs, note that when x tick is 9, it means the `MIC_OMP_NUM_THREADS` is default, not 2^9.





The curves for CUDA programs;





Discussion

In OpenMP + offload implementation, we can see it is slower than running OpenMP only program natively. This is perhaps because that the accelerator XeonPhi is of older version than the computer. Also notice that when `MIC_OMP_NUM_THREADS` is 128 or default, the performance is better than 256. One possible reason is the number of available threads on XeonPhi is between 128 and 256, if we set `MIC_OMP_NUM_THREADS` to be 256, there are more scheduling issues.

In cuda implementation, we divided the picture into several 2-dimensional blocks (for example 16 x 16). Within each block, there are 256 threads in this case. To avoid the duplicate memory access on the same pixel, we used the shared memory model with one layer apron around the block. When we load data into shared memory, each thread load one pixel if the pixel of the thread is not on the block boundary. If the pixel of the thread is on the block boundary, it will also load the nearby pixel into the apron. In this way, each pixel is used multi-times in evaluating the stencil. The speedup is inceasing when increasing the number of threads in the block. The more threads in a block, the better the shared memory works to save the duplicate memory access.