We would like to thank the anonymous referee's constructive comments. As a result, in our opinion, the revised manuscript is improved with respect to the original. An item-by-item response to the comments is presented below. The manuscript was revised accordingly with changes/modifications presented below as well.

*==========

Responses to the anonymous referee's comments:

Anonymous Referee #1
Received and published: 31 January 2015

In this article, the authors port the WRF YSU PBL scheme onto NVIDIA Tesla K40 CPU. With some optimizations, the GPU code gets a good speedup comparing with CPU-only code. I have some concerns about the paper.

(1) First, the NVIDIA Tesla K40 GPUs is the state-of-art accelerator. But the Intel Xeon E5-2603 is not. The comparison may be a little unfair.

The Intel Xeon E5-2603 is the CPU we have here.

(2) Second, the baseline CPU code has not been well tuned. At least the optimization with height dependence release can also be exploited on CPU.

We ran the CPU-based code on one CPU core using exactly the same optimization along with height dependence release as the GPU-based code. It took 1204 ms, which corresponded to speedup of 1.5x.

We added one sentence with a little modification in the third paragraph of Section 5 "Summary and future work" to reflect this issue. The whole paragraph becomes:

"Using one NVIDIA Tesla K40 GPU in the case without I/O transfer, our optimization efforts on the GPU-based YSU PBL scheme can achieve a speedup of 193× with respect to one CPU core, whereas the speedup for one CPU socket (4 cores) with respect to one CPU core is only 3.5×. We also ran the CPU-based code on one CPU core using exactly the same optimization along with height dependence release as the GPU-based code, and its speedup is merely 1.5x as compared to its original Fortran counterpart. In addition, we can even boost the GPU-based speedup to 360× with respect to one CPU core when two K40 GPUs are applied; in this case, one minute of model execution on dual Tesla K40 GPUs will achieve the same outcome as six hours of execution on a single core CPU."

(3) Third, using share memory is a general technique on GPU. According the Section 4.2, you just simply use more L1 cache to get better performance. Have you tried to use share memory for the better locality?

We have tried "cudaFuncCachePreferShared" for using more shared memory as opposed to "cudaFuncCachePreferL1" for using more L1 cache. It was found that the GPU runtime with "cudaFuncCachePreferShared" is almost the same as that with turning off "cudaFuncCachePreferL1" for this scheme.

In other words, the GPU runtimes for using "cudaFuncCachePreferShared" or turning off "cudaFuncCachePreferL1" are liasted in Table 1:

	GPU runtime	Speedup
Non-coalesced	36.0 ms	50.0x
Coalesced	34.2 ms	52.6x

And, the GPU runtimes for using "cudaFuncCachePreferL1" are presented in Table 2:

	GPU runtime	Speedup
Non-coalesced	34.3 ms	52.5x
Coalesced	33.0 ms	54.5x

We added one sentence to the last paragraph in Section 4.2 to address this issue. The whole paragraph becomes:

"Starting with the first CUDA C version of the YSU PBL scheme, the computing performances with L1 cache command was found to be better than that without this command, while the latter performance was noticed to be almost the same as that using "cudaFuncCachePreferShared" command. This suggests that usage of more L1 cache helps to speed up the CUDA C programs for this scheme. The GPU runtime and speedup are summarized in Table 2 after L1 cache command "cudaFuncCachePreferL1" is launched."