Exploring parallel processing opportunities in AERMOD

George Delic, Ph.D., HiPERiSM Consulting, LLC

Introduction

The HiPERiSM experience in working with two major environmental models suggests that opportunities for enhanced parallelism have not been sufficiently explored or developed. One such model is AERMOD [1] developed by the U.S. EPA's Office of Air Quality Planning and Standards (OAQPS). The purpose of this report is to examine code structure and expose the potential for a thread parallel implementation. Evidence of potential performance gains are demonstrated in a simple task farming experiment using the (unmodified) source code. The testbed environment included many- and multi-core processors such as the Intel Xeon and Xeon Phi processors.

Test Bed Environment

The hardware systems chosen were the platforms at HiPERiSM Consulting, LLC, where each of two nodes host two Intel E5v3 CPUs with 16 cores each. In addition, each node has four 1st generation Intel Phi co-processor many integrated cores (MIC) cards with 60 and 59 cores for the respective models. With four MIC cards per node, and 4 threads per MIC core, the total available thread count is 960 and 944, for the respective nodes. This report implemented the Intel Parallel Studio® suite (release 16.0) using options for either host CPU or Phi coprocessor.

A Task Farming Experiment

The benchmark of Case 5 [2], with 916 receptors, was partitioned into separate (independent) tasks by distributing the number of receptors into separate AERMOD input data streams. This partitioning scheme is shown in the table. The resulting number of tasks were then launched concurrently as serial AERMOD runs on the respective processor and coprocessor targets. Note that the node has two CPUs with 16 cores each (together denoted as "host"), whereas the there are four attached Phi cards (denoted as Phi, or MIC) with a total of 240 cores available. The runtimes shown in Figs 1 are based on averages in the case of multiple tasks. Inspection of Fig.1 shows that a 60 task partition of Case 5 on one Phi co-processor out-performs a single core task on the host. However, it also shows that the Phi times for any task collection do not out-perform those of the host with 30 tasks. The longest runtimes are for the 1-core result on either processor with a ratio of ~460 for Phi versus host. However, the 240 task result on the Phi is less than 4 times longer than the 30-core result on the host as shown by the ratio of Fig.2. This suggests that after complete parallelization, where redundant work would be removed, a reversal could be expected.

There is variability in runtimes when receptors are partitioned into separate tasks. The individual runtimes for the example of the 240 tasks on the Phi processor corresponding to 60, 120, and 240 cores are shown in Fig. 3. The separate curves correspond to 1, 2, and 4 Phi processors, respectively. The horizontal axis counts the number of tasks. For more than one Phi processor the task times are concatenated with the division at the cusp(s). The rising trend on each Phi card is due, in part, to earlier tasks receiving more resources on each coprocessor than later ones. Also the differences in receptor data is a contributing factor with some tasks completing before others.

Results

Figure 1: Wall clock time (in seconds) versus task count on host and 1, 2, and 4 Intel MIC co-processors in a log-log plot. This is for Case 5 partitioned by receptors into 1, 30, 60, 120, and 240, separate AERMOD runs as in Table 1.

Figure 2: This is the ratio of Phi times from Fig.1 divided by the best host time corresponding to 30 host cores. The horizontal scale is the task count on the Phi processor(s).

Figure 3: For the example of 240 separate tasks in the table, this shows the time (in minutes) versus each of 240 separate AERMOD tasks. The separate curves correspond to utilization of 60 cores (1 Phi card), 120 cores (2 Phi cards), and 240 cores (4 Phi cards).

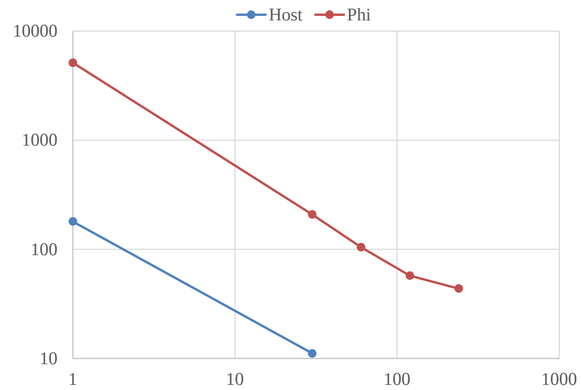
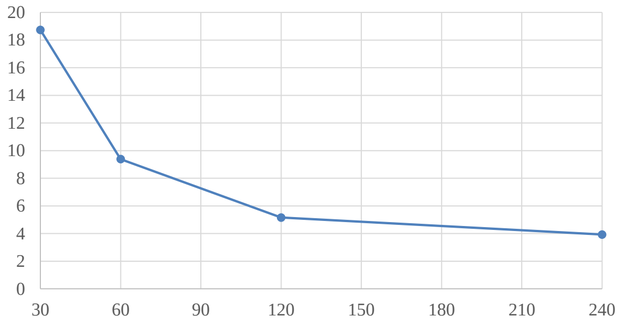


Fig. 1

Number of tasks	Number of receptors per task	Target processor and MIC count
1	916	Host and 1x Phi
30	~ 31	Host and 1 x Phi
60	~15	1 x Phi
120	~8	2 xPhi
240	~4	4 x Phi

Fig. 2



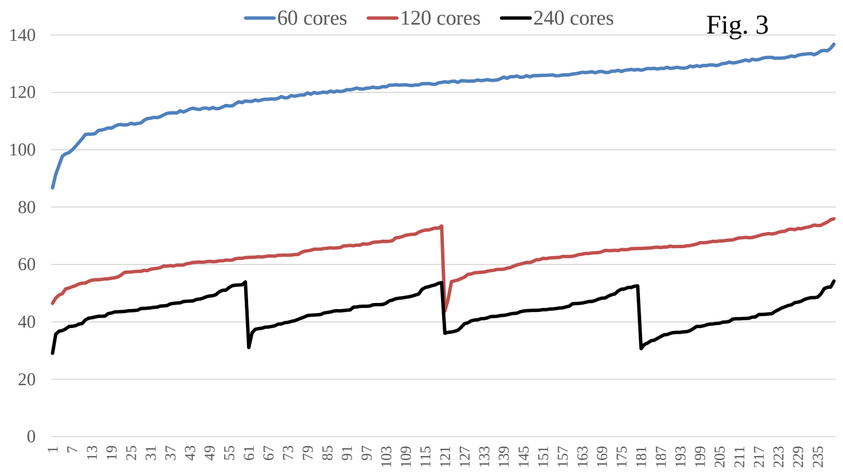


Fig. 3

Parallelization issues

Source and receptor loops

The AERMOD model does have implicit loops that could be parallelized. There are 27 source loops and 38 receptor loops that are typically nested inside the source loops. Any compute intensive receptor loop is a target for thread parallelization and would involve a less complex level of effort than attempting to parallelize on a source loop that contains it

Call tree and the receptor loop

The call tree structure in AERMOD inside potential parallel regions, such as the receptor loop in PCALC, is complex. Some procedures contain deeper procedure call trees. In fact the call tree in the PCALC receptor loop is of the order of four levels deep.

Variables in the receptor loop

Nearly all variables in the call tree contained by the receptor loop are global while a few are local. If the receptor loop is to be modified for thread parallel form, then all variables contained therein, for caller and callee, need to be classified as either shared of private to avoid memory corruption when multiple threads are active in a thread team

Variable categories for the Phi

In addition to the thread parallel requirements, variables contained in the parallel region need to be listed by type for offload to the Phi. There are three categories to list in separate declarations: "in", "inout", or "nocopy".

I/O operations in AERMOD

The AERMOD code contains numerous formatted I/O including some 120 read statements and 1760 write statements. Those that are inside potential thread parallel regions are best hoisted outside it. This would require some memory for storage of data intended for deferred output outside the parallel region. However, this approach would enhance performance by avoiding thread synchronization in the parallel region

Level of effort to parallelize

The code features itemized above imply some considerable level of effort to create thread parallel regions and implies significant code modification and debug effort.

Conclusions

Quantitative evidence of parallel scaling with AERMOD in a simple task farming experiment was demonstrated. While no source code modification was performed, the results with the serial version of AERMOD suggest good potential for performance enhancement on platforms with multiple cores. Theses results motivate an exploration of how best to modify AERMOD for parallel performance.

References

[1] AERMOD is available at the U.S. EPA Technology Transfer Network, Support Center for Regulatory Air Models, (https://www.epa.gov/scram/)

[2] Delic, G. and A. R. Srackangast, 2007: contribution to 6th Annual CMAS Conference, Chapel Hill, NC, October 1-3, 2007, (https://www.cmascenter.org/conference/2007/agenda.cfm)