EXPLORING PARALLEL PROCESSING OPPORTUNITIES IN AERMOD

George Delic *
HiPERiSM Consulting, LLC, Durham, NC, USA

1. INTRODUCTION

HiPERiSM Consulting, LLC, has a mission to develop (or enhance) software and improve performance on current and future computers for legacy Air Quality Models (AQM). One such model is the U.S. EPA's AERMOD developed by the U.S. EPA Office of Air Quality Planning and Standards (OAQPS), Emissions Monitoring and Analysis Division (EMAD), at the U.S. EPA in Research Triangle Park, North Carolina, U.S.A [1]. The purpose of this presentation is to examine the code structure with a view to the potential for a thread-parallel implementation and to provide quantitative evidence of parallel scaling in a simple task farming experiment. While no code modification has been performed the results with the serial version of AERMOD-HPC suggest good potential for performance enhancement on multi-core and many-core processors such as the Intel Xeon and Intel Xeon Phi processors, respectively.

2. TEST BED ENVIRONMENT

2.1 Hardware

The hardware systems chosen were the platforms at HiPERiSM Consulting, LLC, shown in Table 2.1. Each of the 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.

2.2 Episodes studied

Two benchmarks are used in this analysis (Case 2 and Case 5 from [2]). Case 5 has 1001 sources and 916 receptors with as many as 8760 meteorological hours and is the longest running benchmark. Case 2, a much shorter example, is used for profiling.

Table 2.1. CPU platforms at HiPERiSM Consulting, LLC

Platform	Node20	Node21
Operating system	SuSE Linux 13.2	SuSE Linux 13.2
Processor	Intel™ IA32 (E5-2698v3)	Intel™ IA32 (E5-2698v3)
Coprocessor	4 x Intel Phi 7120	4 x Intel Phi 5110
Peak Gflops (SP/DP)	589 (SP)	589 (SP)
Power consumption	135 Watts	135 Watts
Cores per processor	16	16
Power per core	8.44 Watts	8.44 Watts
Processor count	2	2
Total core count	32	32
Clock	2.3 GHz	2.3 GHz
Bandwidth	68 GB/sec	68 GB/sec
Bus speed	2133 MHz	2133 MHz
L1 cache	16x32 KB	16x32 KB
L2 cache	16x256 MB	16x256 KB
L3 cache	40 MB	40 MB

3. BENCHMARKS

The benchmark of Case 5, 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 Table 3.1. The resulting number of tasks were then launched concurrently as serial AERMOD runs on the respective processor and coprocessor targets. Note that node20 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.

* *Corresponding author:* George Delic, george@hiperism.com

Table 3.1. Task separation based on available core count

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

A detailed discussion of results is presented in Section 5 after the examination of the AERMOD code to demonstrate the motivation for these benchmark experiments.

4. CODE EXAMINATION

4.1 AERMOD memory demand

The most significant beneficial property of the serial version of AERMOD is the small amount of memory required at runtime. This suggest that many multiple tasks could be spawned to run concurrently on the same platform without seriously exhausting memory capacity. This is attractive because 1st generation Intel Phi processors are limited to 16 Gigabytes of memory, and as many as 240 separate AERMOD tasks may execute on a single 60 core MIC processor.

4.2 Serial code in AERMOD

AERMOD is a serial code that has a heavy memory foot print [2]. The total number of memory instructions is voluminous and the load balance of memory instructions per floating point operation is ~20. As a memory bound model, a high TLB instruction cache miss rate leads to processor pipeline stalls that leave the arithmetic processor waiting on data. This is due in part to numerous procedure calls (of short duration) and voluminous miss-predicted branch instruction rates. None of these characteristics are corrected in this exercise, but an examination with the Intel VTune profile tools helps to identify hot-spots as a guide to where performance improvement work could begin. Examples are shown in Figs. 1 and 2 for Case 2 where subroutines ANYAVG, IBLVAL, and SIGZ account for nearly half the runtime.

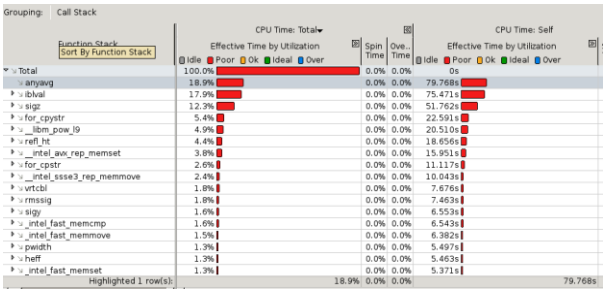

Fig. 1 A VTune profile of Case 2 shows that some 49% of the total time in spent in three procedures: ANYAVG, IBLVAL, and SIGZ.

4.3 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.

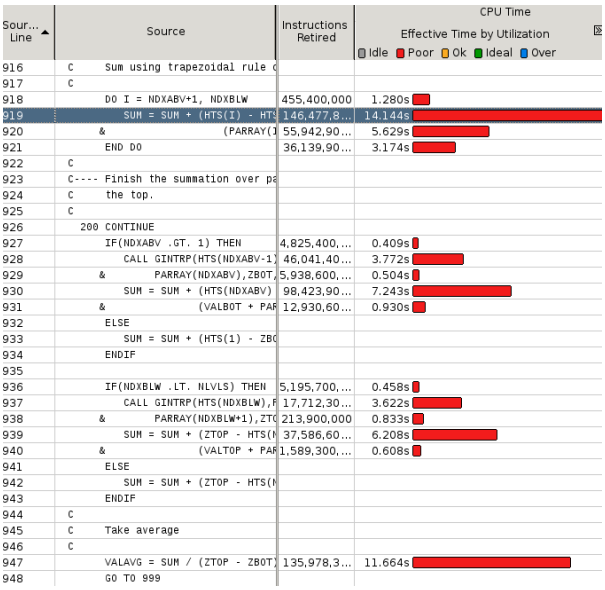

Fig. 2 The example of procedure ANYAVG shows most time is spent in application of the trapezoidal rule.

This structure is repeated in several places depending on the source type. The receptor loop in PCALC (for point sources) is such a candidate. However, thread parallelization over the receptor loop is complicated by several code structure features discussed below.

4.4 Call tree in the receptor loop

The call tree structure in AERMOD inside potential parallel regions, such as the receptor loop in PCALC, is complex. This is shown in Table 4.1 where Those procedures with a "yes" in the last column contain deeper procedure calls. In fact the call tree in the PCALC receptor loop is of the order of four levels deep.

Table 4.1. Call tree in PCALC receptor loop

Procedure called	Module where located	Contains further calls
XYDIST	calc2.f	no
AERCALC	cacl1.f	yes
MEANDR	calc2.f	no
PRMCALC	calc1.f	yes
GAMCALC	calc1.f	yes
EV_SUMVAL	evcalc.f	no
SUMVAL	calc2,f	no
EVALCK	calc2.f	no

4.5 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.

4.6 Variable categories for the Phi

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

"in" for copy from host to MIC,
"inout" for copy back to host from MIC,
"nocopy" local to the MIC.

4.7 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.

4.8 Level of effort to parallelize

A thread parallel version of AERMOD is possible, in principle, by parallelization of the receptor loop. However, the code features itemized above imply some considerable level of effort to create thread parallel regions. This implies significant code modification and debug effort.

5. A TASK FARMING EXPERIMENT

5.1 AERMOD performance

AERMOD runtime results are shown in Figs. 3 and 4. The population of cores on the host is limited at most to 32, so that 30 tasks in this case corresponds to ~31 receptors in each of 30 tasks on either host or Phi processors. The remainder correspond to the tasks listed in Table 3.1. For the Phi case 60 tasks fit one Phi processor, 120 two Phi processors and 240 on four Phi processors. The partitioning could be continued with 480 tasks across the two nodes of Table 2.1, but this was not done in this report. The longest runtimes are for the 1-core result on either processor with a ratio of ~460 for Phi versus host. From Fig. 3 it is seen that scaling with 30 tasks versus 1 task on the host is ~16, and ~32 on the MIC. A result that is expected since the MIC is designed to utilize many cores and threads.

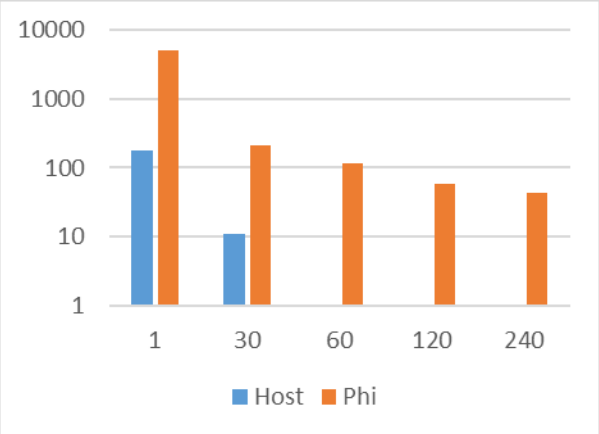

Fig 3. This shows the log of the wall clock time (in minutes) versus the task count on host and 1, 2, and 4 Intel MIC co-processors for Case 5 partitioned by receptors into 1, 30, 60, 120, and 240, separate AERMOD runs as in Table 3.1.

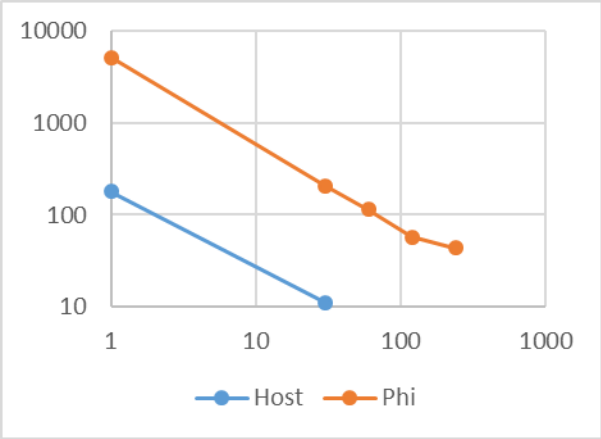

Fig.4.This shows the results of Fig. 3 on a log-log scale to demonstrate the strong scaling as task numbers multiply and compares host and Phi processor times.

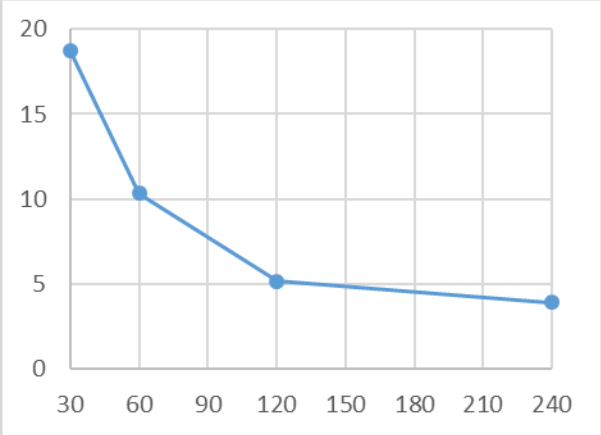

Fig.6.This shows the ratio of Phi times from Fig.3 divided by the best host time corresponding to 30 host cores. The horizontal scale is the task count on the Phi processor(s).

5.2 AERMOD performance on MICs

The runtimes shown in Figs 3 and 4 are based on averages in the case of multiple tasks. 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. 5. The separate curves correspond to 1, 2, and 4 Phi processors, respectively. The horizontal axis counts the tasks.

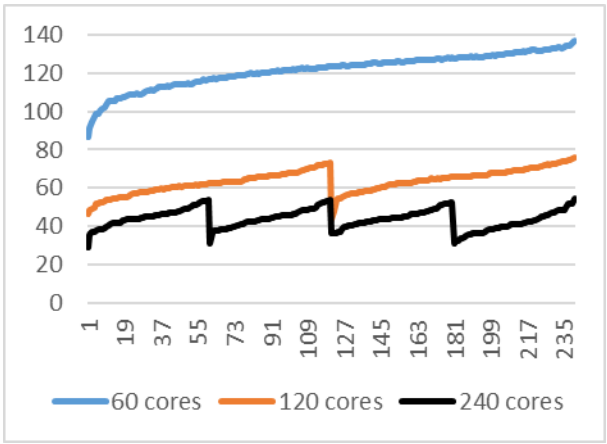

Fig. 5. For the example of 240 separate tasks (Table 3.1), this shows the time (in minutes) versus each of 240 separate AERMOD tasks. The separate curves correspond to utilization of 60 cores (1 Phi card), 120 cores (2 Phi cards), and 240 cores (4 Phi cards).

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.

5.3 Comparing host and MICs

Inspection of Fig.4 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. Nevertheless, the 240 task result on the Phi is less than 4 times longer than the 30-core result on the host as shown in the ratio of Fig.6.

Several factors need to be taken into account in this experiment to understand this outcome:

- Each task is a complete AERMOD execution including replicated input and output operations
- I/O on the Phi requires buffering of all read/write operations over a PCI bus to the host because that is where the hard drives reside

In a thread parallel version of the receptor loop(s) of AERMOD both of these impediments would be ameliorated. Furthermore, since each core on the Phi supports four threads, additional scaling could be uncovered.

As a footnote it is worth noting that these experiments have been conducted with the first generation Phi co-processor. The second generation Phi will be directly coupled to the global memory and storage device so that the I/O buffering over a PCI bus will be absent.

6. 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. These results motivate an exploration of how best to modify AERMOD for parallel performance.

References

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

[2] G. Delic and A.R. Srackangast, 6th Annual CMAS conference, Chapel Hill, NC, October 1-3 2007.