A Feasibility Study on Porting the Community Land Model onto Accelerators Using Openacc

D. Wang
Climate Change Science Institute
Oak Ridge National Lab
Oak, Ridge, TN, USA

W. Wu
Department of Computer Science
University of Tennessee
Knoxville, TN, USA

F. Winkler, W. Ding, O. Hernandez
Computer Science and Mathematics Division Oak Ridge National Lab Oak, Ridge, TN, USA

***Abstract*—As environmental models (such as Accelerated Climate Model for Energy (ACME), Parallel Reactive Flow and Transport Model (PFLOTRAN), Arctic Terrestrial Simulator (ATS), etc.) became more and more complicated, we are facing enormous challenges regarding to porting those applications onto hybrid computing architecture. OpenACC emerges as a very promising technology, therefore, we have conducted a feasibility analysis on porting the Community Land Model (CLM), a terrestrial ecosystem model within the Community Earth System Models (CESM)). Specifically, we used automatic function testing platform to extract a small computing kernel out of CLM, then we apply this kernel into the actually CLM dataflow procedure, and investigate the strategy of data parallelization and the benefit of data movement provided by current implementation of OpenACC. Even it is a non-intensive kernel, on a single 16-core computing node, the performance (based on the actual computation time using one GPU) of OpenACC implementation is 2.3 time faster than that of OpenMP implementation using single OpenMP thread, but it is 2.8 times slower than the performance of OpenMP implementation using 16 threads. On multiple nodes, MPI_OpenACC implementation demonstrated very good scalability on up to 128 GPUs on 128 computing nodes. This study also provides useful information for us to look into the potential benefits of "deep copy" capability and "routine" feature of OpenACC standards. We believe that our experience on the environmental model, CLM, can be beneficial to many other scientific research programs who are interested to porting their large scale scientific code using OpenACC onto high-end computers, empowered by hybrid computing architecture.**

Keywords*—OpenACC; Climate Modeling; Community Land Model; Functional Testing; Performance Analysis; Compiler-assisted Analysis

I. Introduction

As the environmental models (such as Acclerated Climate Model for Energy (ACME), Parallel Reactive Flow and Transport Model (PFloTran), Arctic Terristrial Simulator (ATS), etc.) became more and more complicated, we are facing enormous challenges regarding to porting those applications onto hybrid computing architecture. OpenACC emerges as a very promising technology. In the paper, we present our feasibilty study on porting the Community Land Model (CLM) within the Community Earth System Models using OpenACC. Over the past several decades, researchers have made significant progress in developing high fidelity earth system models to advance our understanding on earth system, and to improve our capability of better projecting future scenarios [1]. The Community Earth System Model is one of the US leading earth system models. CESM is being actively developed under the "Accelarated Climate Model for Energy (ACME)" project to support Department of Energy's climate and environmental research. Within the CESM framework, the CLM is designed to understand how natural and human changes in ecosystem affect climate [2]. The model represents several aspects of the land surface including surface heterogeneity and consists of submodels related to land biogeophysics, the hydrologic cycle, biogeochemistry, human dimensions, and ecosystem dynamics. Currently, the offline CLM simulation system contains of more than 1800 source files and over 350,000 lines of source code. It is well known that the software complexity of the Community Land Model becomes a barrier for rapid model improvements and validation, as well as efficient code porting to next generation HPCs [3,4].

The main purposes of our efforts shown in this paper include: 1) Test data parallelel schemes based on current CLM high level dataflow using a simple non-computing instentive function, 2) Investigate the usefullness of selective copy implemention within on CLM simulation. 3) Evaluate the benefit and cost of porting CLM on accleartors using OpenACC. Specifically, this paper presents detailed information in following sections. We first provide a overview of CLM software structure and dependancy, which leads to our effort of scientific function testing system development. Using our our scientific function testing system, we have extracted one computational kernel out of the whole system, and design and computational experiment for our model potring practices as well as the model compuittional performance evaluations, using both OpenMP and OpenACC.

II. CLM Software Dependency, Data Structure and Workflow

The software system of the global offline CLM includes physical earth system components, such as the CLM, data atmosphere (a proxy atmosphere model, which reads in atmospheric forcings to drive the CLM), stub ocean, stub ice and stub glacier. It contains an application driver to configure the parallel computing environment and the whole simulation system (physical earth system components and flux coupler between those components). It also includes several shared software modules and utilities, such as a flux coupler and its APIs to individual earth system component, parallel IO and performance profiling libraries [4,5]. The schematic diagram of the CLM software structure is shown in Figure 1. It is clear that

the CLM simulation is highly dependent on other components, such as the flux coupler and the data atmosphere.

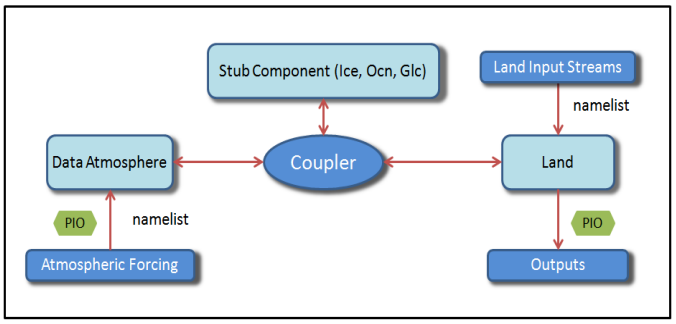

Fig. 1. Software configuration of a global offline CLM simulation that shows a strong coupling with other earth system components. Several earth system model components are listed, including a land model (Land), a data atmospheric model (Data Atmosphere), stub sea ice model (Ice), ocean model (Ocn) and glacier model (Glc)

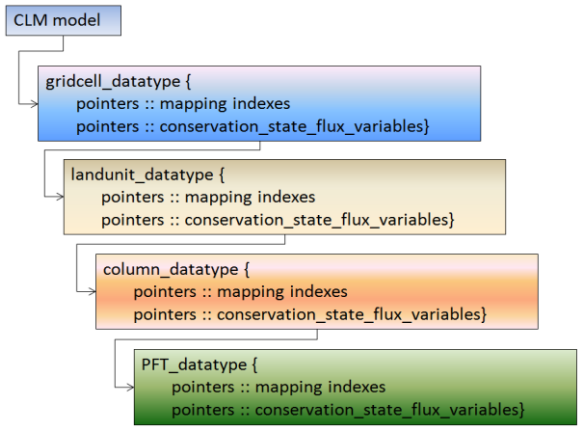

Fig. 2. Hierarchical, derived data structure to represent the heterogeneity of the CLM landscape surface

The key data structure of CLM is a globally accessible derived data type, designed to represent the heterogeneity of landscape surface. Figure 2 shows the CLM data structure in the memory. Each layer of the data structure contains two groups of variables: 1) mapping indexes to represent the spatial connections between those four layers: gridcell, landunit, column, and PFT; 2) derived datatype to store physical data associated with each layer including energy, water, momentum, flux etc.

In the CLM, each gridcell, landunit, soil column, and PFT has a unique ID number. Those multiple level ID numbers are used to create the mapping indexes between those hierarchical landscape surface data structures. The computational domain partition depends on the total number of gridcells across the whole landscape. A static domain-partitioning scheme is implemented in the CLM, so the number of PFTs, soil columns, landunits, and gridcells are fixed on each process during the simulation, most important, there is no cross-domain communication at each of the layered landscape data structure. In another word, CLM, at current stage, is a very good candidate for data parallelism using GPU. Furthermore, a web-based visual analytic system has been developed to explore CLM software structure, an improvement from our previous visual analytics [6]. It provides much needed interface for CLM software structure exploration and further benefits model interpretation and new module development (URL: http://cem-base.ornl.gov/CLM_Web/CLM_Web.html).

III. Software Dependence and Scientific Fucntioanl Testing System

It is obvious that porting the whole CLM simulation system onto accelerator is a very challenging task, considering the complexity of the code itself and more important, the high software dependency on the variety of external math libraries and other earth system components. However, we have implemented an automatic ecosystem function testing system, which is able to extract a specific subroutine/module from CLM and to generate a standalone functional test module for the given subroutine/module. It is a significant improvement from our previous effort on function test platform [7]. Using this testing system, we have successfully tested most ecosystem modules, and it can be extended to all submodels in CLM or even CESM. Originally, it is designed to create direct linkages between site measurements and key ecosystem functions within CLM. It provides much needed integration interfaces for both field experimentalists and ecosystem modelers to improve the model's representation of ecosystem processes within the CESM framework without large software overhead. For the completeness of this paper, we briefly describe the functional testing system here, shown in Figure 3.

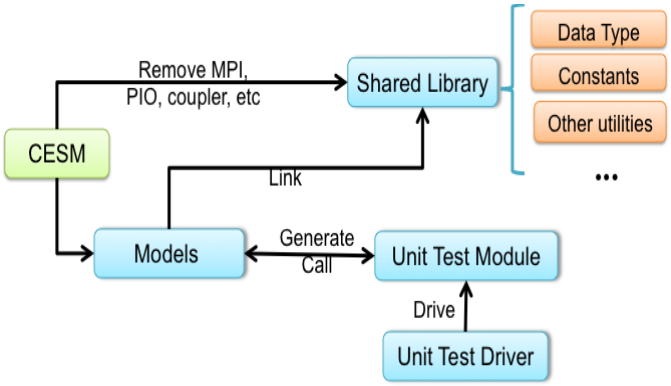

Fig. 3. The software structure of ecosystem function test system. "Shared Library" component contains CLM key data structure and other shared software utilities. For a given CLM function (a single or a group of subroutine(s) in the "Models" component) our system can generate a corresponding test module, located in "Unit Test Module" component, which in turn, driven by "Unit Test Driver"

As shown in Figure 3, the testing system contains a "Shared Library" component, which includes modules commonly used by most of CLM functions, such as key data structure (*clm_type*), physical/chemical/ecological constants (*clm_const*, and *pft_const* etc.) and other utilities (such as String manipulation functions). The "Models" component contains most of software subroutines and modules related to ecosystem functions in CLM, which are exactly the same as the ones in CLM. In order to increase the software system's portability on computing platforms, we have decoupled CLM connections with other CESM components, such as Coupler and Atmosphere, and we have removed several external

libraries and component (such as MPI, NetCDF, PIO and Coupler) from original source code using proxy libraries or component. The key data structure used by CLM (*clm_type*) is still kept as the same as before, so data used for CLM can be directly used in testing program. "Unit Test Module" contains a set of automatically generated unit test modules for given ecosystem functions of interest. "Unit Test Driver" does initialization job to ensure memory location of variables used by unit test module is allocated, it also executes unit test module and verifies the testing results.

IV. Case Study Configuration

As we mention in the previous section, CLM is a very complicated modeling system, it will be a very challenging task to rewrite the majority code using CUDA APIs, therefore, we are more towards to the high-level derivative approach using OpenACC [8]. In this study, we focus on the test of data parallel schemes based on current CLM high level dataflow using a simple non-computing intensive function using one specific ecosystem function (kernel), CNGResp. Within CLM, the CNGResp Module is designed to update all the growth respiration fluxes (the prognostic carbon state variables) at each timestep. Specifically, the schematic procedure of CNGResp Module is shown in Figure 4.

1. define local pointer to the global arrays within CLM_type
2. assign local pointer to derived type arrays (input)
3. assign local pointer to derived type arrays (output)
4. loop though pfts to update leaf and fine root grown respiration

Fig. 4. Schematic procedure of CNGResp Module within CLM

Totally, there are 19 global variables within *clm_type* are used as input datastreams and 18 global variables are used as both input and output datastreams. The computational experiments are configured using similar settings for half-degree offline CLM simulation. Specifically, our landscape surface data structure contains 62482 gridcells, 83935 landunits, 135628 soil columns, and 1101228 plant function types. The workflow of our computational experiment is designed as follow: In each timestep, we copy all the global variables (both input and output datastreams, total 37 arrays) onto the GPU memory, then we break all the loop into parallel computation on GPU cores, and copy back these datastreams. Specifically, we first copied the user defined hierarchical data structure, and around 300 MB data onto GPU memory. Then, after all the computation is done among the CUDA cores, all the data are copied out of the GPU to the corresponding CPU memory locations.

This experiment gives a good opportunity to investigate the usefulness and efficiency of selective copy implementation of one individual function within CLM simulation. Because the CNGResp module is a non computing intensive kernel, which can be used as a benchmark case to evaluate the benefit and cost of poring other CLM kernels on accelerators using similar OpenACC features.

V. MPI_OpenMP Implementation

Since in most cases, CLM is configured to run with the data of global earth, therefore, in this section, we present a way to wrap the CNGResp Model with MPI and OpenMP to maximum parallel computation. According to Fig 3, the top level of CLM data structure is grid cell. Each grid cell is independent, so a generic method is to parallel the program by grid cell; each MPI operates on one or more grid cells. However, in this particular CNGResp Model, the subroutine operates on the column level, so instead of dealing with grid cell and land unit, we only have to parallelize column. Assume the number of column is C, and number of MPI process is NP, then each MPI process operates on C/NP columns. Since most current CPUs have multiple cores, we use OpenMP to parallelize computation on each MPI process. If there are NT OpenMP threads, so each OpenMP operates on C/NP/NT columns. CNGResp Model does not access all the pfts in each column, so each OpenMP thread has its private pft filter variable to get access to the filtered pfts. Algorithm 1 shows the pseudo code of MPI-OpenMP implementation of CNGResp Model

Algorithm 1 MPI_OpenMP based partition pseudo code of CNGResp Model

```
column_per_MPI = C/NP
column_per_OpenMP = column_per_MPI/NT
steps = days * 24 * 2
begin_index = 1 + mpi_rank*column_per_MPI
!$OMP PARALLEL NUM_THREADS(NT)
PRIVATE(tid, pft_filter, num_pft, end_index)
      tid = omp_get_thread_num()
end_index = begin_index + (tid+1)*chunk_size
DO i = 1, steps
   DO j = begin_index + tid*column_per_OpenMP,
begin_index + (tid+1)* column_per_OpenMP
      call get_pft_filter(pft_filter, num_pft)
      call CNGResp(num_pft, pft_filter)
   END DO
END DO
!$OMP END PARALLEL
```

VI. OpenACC Directive and Implementation

OpenACC is a directive-based language extension for Fortran, C, and C++, that facilitates the simple and effective use of accelerators (e.g., GPUs) without sacrificing portability for non-accelerator systems. The Oak Ridge Leadership Computing Facility (OLCF) has made a strategic investment in OpenACC for the Titan system and applications are starting to use it. However OpenACC is a very young specification. Application scientists at ORNL have already identified a number of extensions to OpenACC that would significantly enhance its expressiveness and usability in their applications. Looking further forward, towards ExaScale computers, we see trends towards node-level environments with heterogeneous compute resources, and more complex memory environments. Extending OpenACC to support such environments, with task-based execution, the ability to control placement of data in memory, and interoperability with other prominent node-level programming models will smooth the path for today's

applications to make the transition to new ExaScale architectures, as well as preparing them to for the jump to next-generation programming models and languages. One of the important missing features that OpenACC needs is to support levels of memory copy. Data structures such as "struct" and "STL" are common used in C and C++ programming, but users are not allowed to copy those encapsulated data into GPU memories directly. Manually restructuring code to avoid dereference is a common way which is used to handle those cases. However this is a non-trivial process, especially when the code contains multiple levels of dereference. At current stage, a very straightforward method was adopted in this study to evaluate the efficiency of OpenACC copy by using our real scientific application. Specifically, we use copy function to copy both the data structure, and all the input datastreams for CNGResp module from the *clm_type* data structure in CPU, and break the computational do loop and map those computation onto GPU cores. After the computation on GPU, we use the copy function to move these output datastreams out from GPU and updates the data values within *clm_type* in CPU memory.

VII. MPI_OpenACC Implementation

Parallel partition scheme within our MPI_OpenACC Implementation is very similar to that within MPI_OpenMP. Assume the number of column is C, and number of MPI process is NP, then each MPI process operates on C/NP columns. Since each computing node has a GPU, we use OpenACC to parallel computations on each MPI process. If there are NT OpenACC threads, then each OpenACC operates on C/NP/NT columns. Algorithm 2 shows the pseudo code of MPI-OpenACC implementation of CNGResp Model. The significant parts of the code include 1) explicit data structure copy (copy in), and 2) explicit data value copy (both in and out). Due to the limitation of current PGI implementation, we are not able to use the "deep copy" and "routine" feature, but we are very confident that our program will greatly benefit from these two features once they are implemented, because we can use the "routine" feature to make the code structure more concise, and most importantly, we can use the "deep copy" capability to copy only these input variables on and output variables off the GPU memory.

Algorithm 2 MPI_OpenACC based partition pseudo code of CNGResp Model

```
column_per_MPI = C/NP
steps = days * 24 * 2
begin_index = 1 + mpi_rank*column_per_MPI
end_index = column_per_MPI*(mpi_rank+1)
!$ACC DATA COPYIN(struct) COPY(members)
DO i = 1, steps
!$ACC KERNELS
!$ACC LOOP INDEPENDENT
  DO j = begin_index, end_index
     Compute CNGResp
  END DO
END DO
!$ACC END KERNELS
!$ACC END DATA
```

VIII. Computational Experiments and Scalability Analysis

In this section, we investigate performance impact of MPI-OpenMP and MPI-OpenACC implementation. The experiment case used in benchmark is a fixed size problem since size of the landscape surface data structure is already given. Therefore, strong scalability experiment is the best option to present the performance speedup by using MPI_OpenMP and MPI_OpenACC. In a strong scalability experiment, the problem size is fixed, while the number of OpenMP thread is increased.

A. Computational Platform

The computational platform used in this research is the Cray XT6 Titan supercomputer at the National Center for Computational Sciences (NCCS) at Oak Ridge National Laboratory (ORNL). Titan uses 16-core AMD Opteron central processing units (CPUs) in conjunction with NVidia Tesla K20X GPUs. It uses 18,688 CPUs paired with an equal number of graphics processing units (GPUs) to perform at a theoretical peak of 27 PetaFLOPS.

A center-wide Lustre file system provides 5 PB of disk space for all NCCS computing resources. The broach configuration of K20X GPU are listed as following: Processor clock, 732 MHz; Memory clock, 2.6 GHz; Memory size 6 GB; Memory I/O 384-bit GDDR5; and Memory configuration 24 pieces of 64M ×16 GDDR5 SDRAM. According to the online K20X GPU document [9], the peak double precision floating point performance (board) can reach 1.31 teraflops, and the memory bandwidth for board (ECC off) can reach 250 GBytes/sec. In our study, we used PGI FORTRAN compiler (version 14.7.0), Cray-mpich (version 6.3.0), OpenMP (version 3.1) and CUDAtoolkit (version 5.5.20-1.0402.7700.8.1).

B. Single Node (Shared Memory System)

In shared memory system benchmark, we demonstrate the performance impact by using OpenMP to parallel computation. Figure 5 presents the strong scalability performance of MPI-OpenMP implementation on 1 node. In ideal strong scaling, a program is considered to scale linearly if the speedup (in terms of work units completed per unit time) is equal to the number of processing elements used. While in our case, we are not able to achieve linear speedup when the number of threads varies from 1 to 16.

However, we did observe the performance increase (computation time decrease) when more OpenMP threads has been used. It is because the operation contained in CNGResp subroutine is mostly floating point operation. The experiment case is to simulate 30 days (1 iteration simulates plant growth respiration in 30 minutes). There is billions of floating point operations in total. Since the AMD CPU contains 16 cores, when more OpenMP threads have been used, less computation work was assigned on each CPU core, therefore, less time has been used for those floating point operations on each CPU core. For example, when 16 OpenMP threads were used, the computation time of each core is less than 13.3 second, which gave out a speedup number of 6.3, as shown in Figure 5.

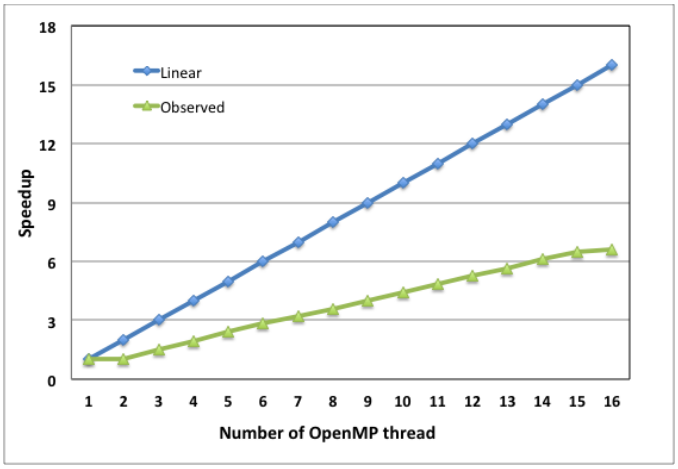

Fig. 5. Strong Scalability of CNGResp Model (MPI_OpenMP) running on a single node of Titan machine using one MPI process. The speedup varies by the number of OpenMP threads on the single computing node

C. Multiple Nodes (Distributed Memory System)

In the distributed memory test case, each MPI process occupies one computation node, and OpenMP is used to obtain more parallelism within each MPI process. Table 1 presents the strong scalability of CNGResp Model running in multiple Titan nodes. On each computing node, there is one MPI process with 16 OpenMP threads. The computation time of sequential implementation (using one MPI process and 1 OpenMP thread on a single node) is around 88.9 seconds, which is used as the benchmark performance for the speedup number calculation.

TABLE I. STRONG SCALABILITY OF CNGRESP MODEL (MPI_OPENMP) RUNNING ON UP TO 128 NODES OF TITAN MACHINE. ON EACH COMPUTING NODE (WITH 16 CPU CORES), ONE MPI PROCESS AND 16 OPENMP THREADS WERE USED. THE SPEEDUP WAS CALCULATED AGAINST THE COMPUTATION TIME OF SEQUENTIAL IMPLEMENT (1 MPI AND 1 OPENMP THREAD) ON SINGLE COMPUTING NODE (88.9 SECONDS)

# of Nodes (Each has 16 cores)	Time (s)	Speedup (16 OpenMP threads)
1	13.26	6.7
2	6.22	14.3
4	3.07	29.0
8	1.52	58.6
16	0.73	121.9
32	0.41	217.1
64	0.22	404.5
128	0.12	741.7

As shown in Table 1, the model demonstrated a good scalability up to 128 nodes, in which up to 128*16 = 2048 CPU cores were used for computation. In the simulation using single computing node, the maximum computation time on each core is less than 13.26 seconds. While in the simulation using 128 nodes, the maximum computation time on each CPU core is less than 0.12 second.

Similarly, we have conducted the scalability experiment on those Titan nodes, using MPI_OpenACC implementation. Table 2 shows the strong scalability of CNGResp Model running on Titan with the OpenACC implementation. For the comparison, the speedup number of OpenACC is also calculated against the computation time of sequential code on CPU (that is around 89 seconds). Two facts worthy mentioning here: (1) the computational time of OpenACC implementation on single node (38.7 seconds) is faster than that of single thread OpenMP implementation (88.9 seconds), but it is slower than that of 16-thread OpenMP implementation (13.4 seconds). (2) Due to the limitation of current deep copy feature, we have to use the standard copy function to move all input and output global variable and the data structure on and off the GPU. More detailed results were shown in next section. Also from the coding perspective, we found the OpenACC implementation is very straightforward, so that we think automatic instrumentation of these OpenACC directives into the CLM source is feasible and have great potentials for further CLM parallel code development.

TABLE II. STRONG SCALABILITY OF CNGRESP MODEL (MPI_OPENACC IMPLEMENTATION) RUNNING ON UP TO 128 NODES OF TITAN MACHINE, EACH MPI PROCESS USES ONE K20C GPU. THE SPEEDUP IS CALCULATED AGAINST THE COMPUTATION TIME OF SEQUENTIAL CODE ON SINGLE CPU CORE (88.9 SECONDS)

# of GPUs	Time (s)	Speedup
1	38.7	2.3
2	20.76	4.3
4	10.56	8.5
8	5.5	16.4
16	2.98	30.2
32	1.72	52.3
64	1.05	85.7
128	0.74	121.6

As shown in Table 2, the model demonstrated a very good scalability up to 128 nodes, in which up to 128 GPUs were used for computation. In the simulation using single computing node, it took 38.7 second to finish all the computation using one GPU, that gave out the speedup number of 2.3. While in the simulation using 128 nodes, the maximum computation time on each GPU is less than 0.74 second, giving out a speedup number of 121.6.

IX. SYSTEMATIC PERFORMANCE ANALYSIS

In order to get more detailed information on the OpenACC implementation, we used Vampir toolkit (www.vampir.eu) to trace and analyze detailed performance matrix on GPU [10].

The Vampir toolkit consists of the runtime measurement system Score-P [11], and the performance analysis tool Vampir [12]. Score-P is a new convenient measurement infrastructure for collecting performance data. It supports the developer with instrumentation and allows detailed logging of program execution for parallel applications using message passing (MPI), threads (OpenMP, Pthreads), and offloading to accelerators (OpenACC and CUDA). Score-P provides two commonly used techniques to investigate the performance behavior of parallel applications: Profiling and Tracing. Profiling is based on aggregating performance data, which allows a statistical view on a program run such as number of

invocations or accumulated time of functions or messages. Unlike profiling, the tracing approach does not summarize events. Tracing records all events of an application run that are of interest for later examination together with the time they occurred and a number of event type specific properties. A trace file contains all recorded events in a chronological order and therewith allows a time line representation of the program execution. Trace files generated by Score-P can be analyzed with Vampir in a post-processing step. Vampir is a performance analysis tool that offers intuitive parallel event trace visualization with many displays showing different aspects of the parallel performance behavior. Vampir provides interactive zooming and browsing to show either a broad overview or details of the program behavior. Different timeline displays show application activities and communication along a time axis. Statistical displays provide quantitative results for arbitrary portions of the timelines. Powerful zooming and scrolling allows to pinpoint the real cause of performance problems. Vampir is designed to be an easy to use tool, which enables developers to quickly display the program behavior at any level of detail.

Since tracing causes some instrumentation overhead we used profiling to get an overview of accumulated timing information of MPI, user regions, and CUDA kernel executions generated by OpenACC directives. This was very useful to determine the ratio of GPU to host computation. For a more advanced performance analysis we used tracing to visualize the dynamic runtime behavior in Vampir at any level of detail. Using tracing we have recorded exact time stamps for all GPU related events such as kernel execution on the assigned CUDA streams, fixed CUDA kernel metrics (threads per kernel, memory usage), host-device data transfers and synchronization, and GPU idle time. The Vampir analysis of the generated trace files helped us to understand and enhance the OpenACC implementation at scale by using different OpenACC directive combinations which have impact on CUDA kernel executions, host-device data transfers and synchronization.

X. Result Discussions

In this section, we focus on the analysis of trace files for these computational experiments with OpenACC implementation.

Figure 6 shows the master timeline of the simulation on single Titan node. The MPI_Barrier, shown at time mark (11.00 second), presented the completion of model initialization. The data copy preparation started right after the MPI_Barrier, and finished at the time mark of 11.25 second, when data copy started. The actual GPU computation started at the time mark of 11.34 second, therefore, the total time of data movement (320 MB) took only 0.09 seconds. The total GPU computation time was around 38.70 seconds.

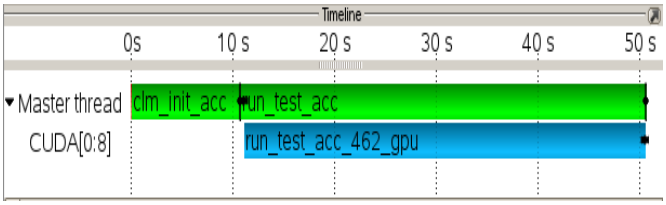

Fig. 6. Vampir Timeline information on CNGResp Model running on single Titan node with one MPI process (Master thread) and one K20x GPU (CUDA[0:8])

The CNGResp kernel was automatically renamed after the caller function "run_test_acc" plus the line number (#462) where OpenACC kernel directive was defined.

Figure 7 shows the trace information on our simulation using 4 MPI processes and 4 GPUs on 4 Titan nodes, including master timeline, function summary, message information, as well as a close-up look at the data copy before the GPU computation. Again, the MPI_Barrier, started and finished between the time mark of 2.83 second and 2.86 seconds, presented the completion of model initialization (total time of 11.23 second on all 4 nodes).

The data copy preparation started right after the completion of MPI_Barrier (at 2.86 second), and all finished before the time mark of 3.08 second. Therefore, on 4 nodes, totally about 0.88 second was used for data preparation, data copy (including some extra ideal time on each node). The actually data copy operation only took about 0.02 second on each node. The actual GPU computation started around the time mark of 3.08 second, and the total time of GPU computation took about 39.83 seconds. Totally, around 600 MB data have been moved in and out of GPU devices.

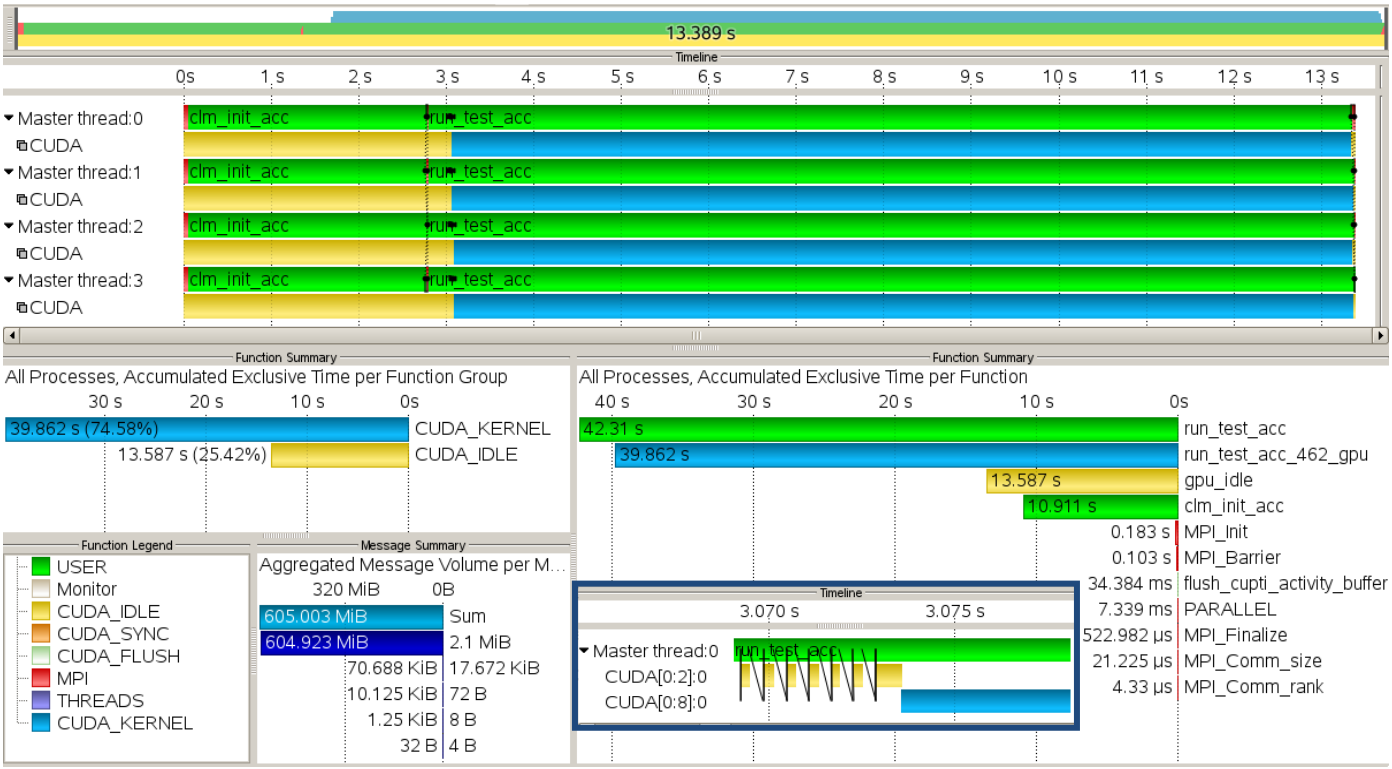

Fig. 7. Vampir visualization of CNGResp Model running on 4 Titan nodes with 4 MPI processes and 4 K20x GPUs.The main timeline is shown in the top panel. A small segment of the data movement is shown in a close-up window on the bottom. One can clearly see that the GPU is only idle when there is no data transfer or kernel execution. The Function Summary (left middle) shows that almost 75% of the GPU is fully occupied. The Function Summary (right middle) shows profile information of all functions. A Function Legend is shown on the left bottom panel. A Message Summary (left bottom) shows that 605MiB of data were copied between host and GPU device

Herein, we also listed some trace information on computational experience across 128 Titan nodes using 128 MPI processes and 128 GPUs. During the simulation, the MPI_Barrier, started and finished between the timestamps of 0.23 second and 0.25 seconds, presented the completion of model initialization (total time of 30.72 second on all 128 nodes). The data preparation and copy operations started right after the completion of MPI_Barrier (at 0.25 second), and all finished before the timestamp of 0.53 second, when the GPU computation starts on all GPUs. Therefore, on 128 nodes, totally about 35.84 second was used for data preparation and data copy (including significant extra ideal time on each node). The actually data copy operation still only took about 0.02 second on each node. The actual GPU computation started around the time stamp of 0.53 second, and ended around the time mark of 1.01 second. Therefore, the total time of GPU computation took about 61.44 seconds. Again, totally, around 600 MB data have been moved in and out of GPU devices.

XI. Conclusions and Future Work

We have demonstrated our objectives, methods and case study to investigate the feasibility of porting CLM key data structure and simplified data flow onto accelerators using the copy feature of OpenACC. It is obvious that there are room for further OpenACC performance improvement, specially related to selective data movement and code rewriting using “routine” feature. Considering the huge software complexity of CLM code, and continuous code changes from active model development, we view the high-level programing derivatives approach using OpenACC of great interest. Based on our previous work (such as on interactive CLM structure exploration and CLM functional testing code generation and data stream identification, compiler analysis and other preliminary CLM code immigration preparations), we think it is the very useful first step to porting CLM onto pre-ExaScale computers using OpenACC approach. There are further investigations needed, specially, those implementations using “deep copy” function and “routine” features. We are also conducting similarity-based analysis for CLM [13,14], which in turn give us more information on porting individual kernels onto GPUs. We believe that our experience on pilot study on porting modular environmental models can be beneficial to many other scientific research programs which adapt high-level programming directives to porting scientific applications on hybrid high-end computers.

Acknowledgment

This research was partially funded by Terrestrial Ecosystem Sciences (TES) Program and Climate Sciences for Sustainable Energy Future (CSSEF) Program under the Biological and Environmental Research (BER), Office of Science of the U.S. Department of Energy (DOE). This research used resources of the Oak Ridge Leadership Computing Facility, located in the National Center for Computational Sciences at Oak Ridge National Laboratory, which is managed by UT-Battelle LLC for the Department of Energy under contract DE-AC05-00OR22725.

REFERENCES

[1] Washington, W. M., C. L. Parkinson, 2005, An Introduction to Three-Dimensional Climate Modeling, University Science Books, 2rn edition.

[2] Oleson, K., Lawrence, D., Gordon, B., Flanner, M., Kluzek, E., Peter, J., Levis, S., Swenson, S., Thornton, P., and Feddema J., 2010, Technical description of version 4.0 of the Community Land Model (CLM).

[3] Wang, D., Post, W., Wilson, B., 2011. Climate Change Modeling: Computational Opportunities and Challenges, IEEE Computing in Science and Engineering, Vol 13(5), pp36-42

[4] Wang, D., Schuchart, J., Janjusic, T., Winkler, F., and Xu, Y. 2014a. Toward Better Understanding of the Community Land Model within the Earth System Modeling Framework, International Conference on Computational Science, Cairns, Australia, 2014,

[5] Domke, J., Wang, D., Runtime Tracing of the Community Earth System Model: Feasibility Study and Benefits, 12th Workshop on Tools for Program Development and Analysis in Computational Science, Omaha, Nebraska, June 2012, procedia CS 9: pp1950-1958, 2012

[6] Xu, Y., Wang, D., Janjusic, T., Xu, X., A Web-based Visual Analytic System for Understanding the Structure of Community Land Model, International Conference on Software Engineering Research and Practice, June, 2014.

[7] Wang, D., Xu, Y., Thornton, P., King, A., Gu, L., Steed, C., Schuchart, J., 2014b. A Functional Testing Platform for the Community Land Model, Environmental Modeling and Software, Vol. 55, pp25-31, 10.1016/j.envsoft.2014.01.015

[8] Oscar Hernandez, Wei Ding, Barbara Chapman, Ramanan Sankaran, Richard L. Graham, Christos Kartsaklis, "Experiences with High-Level Programming Directives for Porting Applications to GPUs". Facing the Multicore - Challenge II. Lecture Notes in Computer Science, Volume 7174/2012, pages 96-107, 2012

[9] Telse K-Series Overview, available online at http://www.nvidia.com/content/tesla/pdf/Tesla-KSeries-Overview-LR.pdf

[10] Dietrich, R., Winkler, F., William, T., Stolle, J., Henschel, R., & Berry, D. K. (2013). A Case Study: Holistic Performance Analysis on Heterogeneous Architectures using the Vampir Toolchain. In *PARCO* (pp. 793-802).

[11] A. Knüpfer, H. Brunst, J. Doleschal, M. Jurenz, M. Lieber, H. Mickler, M. Müller and W.E. Nagel, "The Vampir Performance Analysis Tool-Set", Tools for High Performance Computing, pp 139-155, Springer Verlag, 2008

[12] A. Knüpfer, C. Rössel, D. an Mey, S. Biersdorf, K. Diethelm, D. Eschweiler, M. Gerndt, D. Lorenz, A. D. Malony, W. E. Nagel, Y. Oleynik, P. Saviankou, D. Schmidl, S. Shende, R. Tschüter, M. Wagner, B. Wesarg, F. Wolf: Score-P - A Joint Performance Measurement Run-Time Infrastructure for Periscope, Scalasca, TAU, and Vampir, Proceedings of 5th Parallel Tools Workshop, 2011

[13] Ding, W., Hsu, C. H., Hernandez, O., Chapman, B., & Graham, R. (2013). KLONOS: Similarity-based planning tool support for porting scientific applications. *Concurrency and Computation: Practice and Experience*, *25*(8), 1072-1088

[14] Wei Ding, Oscar Hernandez, and Barbara Chapman. "A Similarity-Based Analysis Tool for Porting OpenMP Applications". In Facing the Multicore-Challenge III, pp. 13-24. Springer Berlin Heidelberg, 2013.