Procedia Computer Science

Volume 29, 2014, Pages 1515–1524

ICCS 2014. 14th International Conference on Computational Science

Toward Better Understanding of the Community Land Model within the Earth System Modeling Framework

Dali Wang[a,*], Joseph Schuchart[b], Tomislav Janjusic[a], Frank Winkler[a], Yang Xu[c] , Christos Kartsaklis [a]

[a]*Oak Ridge National Labortory, Oak Ridge, TN 3783, USA*
[b]*Dresden University of Technology, 01062, Dresden, Germany*
[c] *The University of Tennessee, Knoxville, TN 37966, USA*
{wangd, janjusict, winklerf, kartsakisc}@ornl.gov; joseph.schuchart@tu-dresden.de; yxu30@utk.edu
* Corresponding author. Email: wangd@ornl.gov, Tel.: +1 8652418679; fax: +1865 5749501.

Abstract
One key factor in the improved understanding of earth system science is the development and improvement of high fidelity models. Along with the deeper understanding of biogeophysical and biogeochemical processes, the software complexity of those earth system models becomes a barrier for further rapid model improvements and validation. In this paper, we present our experience on better understanding the Community Land Model (CLM) within an earth system modelling framework. First, we give an overview of the software system of the global offline CLM simulation. Second, we present our approach to better understand the CLM software structure and data structure using advanced software tools. After that, we focus on the practical issues related to CLM computational performance and individual ecosystem function. Since better software engineering practices are much needed for general scientific software systems, we hope those considerations can be beneficial to many other modeling research programs involving multiscale system dynamics.

Keywords: Legacy Scientific Software Application, Community Earth System Model, Community Land Model, Global Variables, High Performance Computing, Software Profiling and Debugging

1 Introduction

Over the past several decades, researchers have made significant progress in developing high fidelity earth system models to advance our understanding on the Earth systems, and to improve our capability of better projecting future scenarios [1]. The Community Earth System Model (CESM) is a flagship model for US Department of Energy's climate and environmental research. Within CESM, the Community Land Model (CLM) is the active component to simulate surface energy, water, carbon, and nitrogen fluxes and state variables for both vegetated and non-vegetated land surfaces [2]. The whole CESM simulation is reconfigurable, which provides a great flexibility to the community to design their own computational experiments. In our study, CESM has been configured into an offline

Selection and peer-review under responsibility of the Scientific Programme Committee of ICCS 2014

© The Authors. Published by Elsevier B.V.
doi:10.1016/j.procs.2014.05.137

global community land model simulation, which includes a data atmosphere model, an active land model and stub models for ocean, ice and glacier. Scientifically, this configuration uses historical climate forcing to drive active land component simulation, and provides unique capability to verify and calibrate land modeling activities with observational datasets.

Considering the general interests of ICCS, we focus on several software engineering approaches to better understand the offline Community Land Model system within the unified Community Earth System Model framework. First, we review the overall software system of the global offline Community Land Model simulation. Second, we present our approach to better understand the software structure and data structure using advanced software tools. After that, we focus on the issues related to scientific software performance, including both computational performance and most important, functional level performance. Due to the ultimate scientific goals and the nature of scientific modelers, the development of scientific software systems differs significantly from the development of other software systems, such as business information systems. We believe the software tools and scientific approaches described in this paper can be beneficial to many other research programs involving large scale, legacy modeling systems.

2 Overall Configuration of the Community Land Model

Within the CESM framework, the CLM is designed to understand how natural and human changes in ecosystems affect the climate. The model represents several aspects of the land surface including surface heterogeneity and consists of submodels related to land biogeophysics, the hydrological cycle, biogeochemistry, and ecosystem dynamics. 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 [3]. 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.

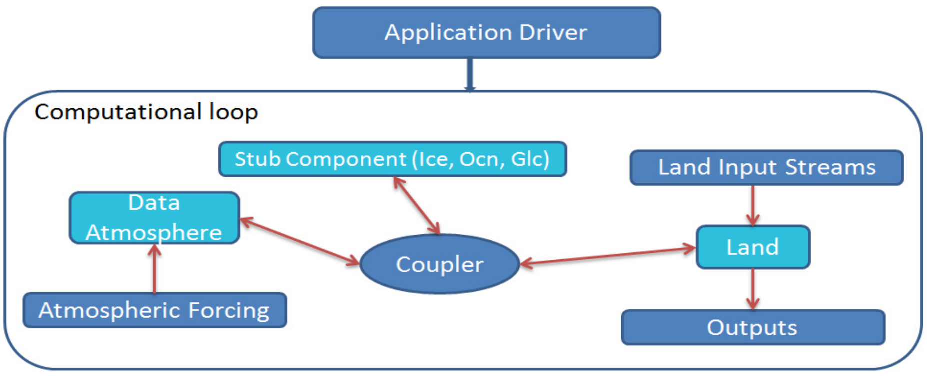

Figure 1: Software configuration of the CLM simulation that shows the strong coupling of the earth system components.

3 Software Structure of the CLM

3.1 The CLM Call Graph

The whole CLM modeling system consists of more than 1800 source files and over 350,000 lines of source code. Call-graph analysis is a widely used tool for understanding the interactions between different parts of an application. Different tools exist for both static and dynamic call path generation and analysis. The well-established gprof (and later improved approaches) tool records profiling information at runtime and presents users with a textual representation of the dynamically recorded call paths and their runtimes [4, 5]. The Vampir trace visualization tool also offers a textual representation of call graphs extracted from trace data [6]. Moreover, the Scalasca toolset puts a strong focus on call-path profiling [7]. However, none of these tools offer a graphical representation of the call graph to aid the user in gaining a complete overview of the application. In [8], the authors present prototypes that display dynamic relations between code parts in graphical displays, so-called Execution Murals. However, these displays do not include any information about the performance or runtime of the individual components. Furthermore, in contrast to the dynamic call graph generation, static call graph generation tools analyze the source code of a program to extract procedure call relations [9, 10]. Also due to their static nature, those tools cannot incorporate any information on the performance and runtime of the components.

In our study, we use advanced profiling and tracing software, the Vampir framework [11] and the Tuning and Analysis Utilities (TAU) [12], to collect necessary information for constructing the CLM call graph. The Vampir framework is being developed at the Center for Information Technology and High Performance Computing (ZIH) at Dresden University of Technology. It consists of the open source instrumentation and measurement tool VampirTrace (for recording profiling and application trace information) and the commercial Vampir (for trace visualization and analysis). TAU is an open source profiling and tracing toolkit for performance analysis of parallel programs that is being developed at the University of Oregon. Nowadays, many advanced compilers (such as the Cray compilers) provide capabilities for automatic compiler instrumentation. However, this straightforward approach produces too much trace data and the overhead is too high to make a meaningful statement on the performance of the CLM. Therefore, we use the TAU instrumentation within VampirTrace to prevent the complete source files from being instrumented. More detailed information on how to configure the Vampir framework for this purpose can be found in a previous paper [13].

The Vampir framework provides functionality to generate call graphs once the trace information is collected. This graph is limited to textual representation inside of Vampir. Herein, we present a new way to enable users to study the runtime structure of the CLM call graph. A Python script was developed to build the CLM call graph in a standard graph format, such as the Graph Modeling Language (GML). The trace is read through the OTF Python API that is modeled after the original C programming interface, which uses a callback mechanism to deliver definitions and events to the reader. The script extracts only the relevant information, e.g., function and process definitions and enter/exit events, from the trace data and modifies the graph on every event. For managing the graphs, the Python networkx package is employed for easy and efficient graph operations such as inserting nodes and edges and modifying their weights. A new node is inserted every time a new function is encountered. A new edge is inserted for each call into a subroutine. The exclusive function runtime is added to the weight of a node and the weight of the corresponding edge is incremented. We used Gephi (www.gephi.org), a multi-platform tool, for interactive graph visualization and exploration. Gephi is especially designed to handle large graphs and supports multiple algorithms for node-placement and metric computation. Moreover, Gephi has a modular design that allows for easy extension using modules, many of which are provided by an active user community.

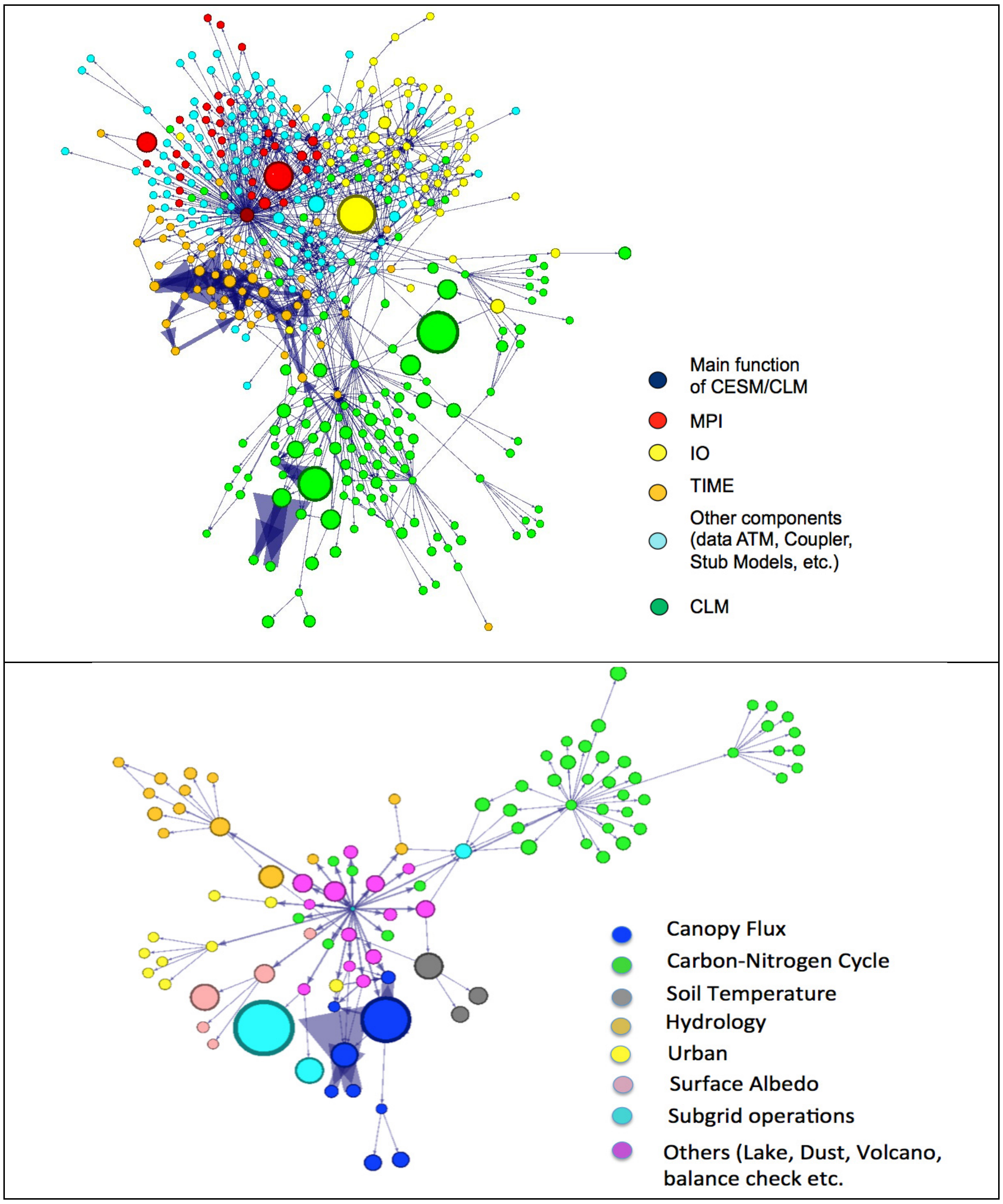
Main function of CESM/CLM
MPI
IO
TIME
Other components (data ATM, Coupler, Stub Models, etc.)
CLM
Canopy Flux
Carbon-Nitrogen Cycle
Soil Temperature
Hydrology
Urban
Surface Albedo
Subgrid operations
Others (Lake, Dust, Volcano, balance check etc.

Figure 2 The software structure of the CLM. Each circle represents an individual subroutine, the area of each circle shows the time spent on the subroutine. Each directed edge indicates the procedure of a subroutine invocation with the weight of the arrows linearly representing the number of invocations. The top graph shows the subroutines related to the CLM simulation within the CESM framework. It is obvious that the software overhead of CLM is significant, such as MPI, IO and time management is significant. The green colored part in the top graphs is the actual CLM model. The bottom graph represents all the submodels within CLM, which includes Canopy fluxes, Carbon-Nitrogen Cycle, Hydrology, Urban, etc.)

3.2 Key Data Structure of the CLM Landscape Surface

Landscape surface is the basic data structure for CLM model development and software engineering. Inside the CLM, the land surface is represented by globally accessible, hierarchical, derived data structures. The landscape surface is represented by four layers of derived data structures: gridcell, landunit, column, as well as Plant Functional Type (PFTs). A gridcell contains landunits, which represent the geomorphical feature of the landscape surface. The primary landunit types include glacier, lake, wetland, urban, and vegetated portion. Each landunit contains different types of columns, such as soil, lake, and wetland. The vegetated portion of a gridcell is represented as soil column, which is further divided into patches of PFTs, each with its own leaf and stem area index and canopy height. All layers of the landscape surface data structure (such as gridcell, landunit, column, and PFT) are declared as globally accessible datatypes, which can be modified by a variety of ecosystem functions.

In our study, we use a debugger, Allinea DDT (*www.allinea.com)*, to better understand the hierarchical data structure. Allinea DDT is a commercial debugger developed by Allinea Software of Warwick, United Kingdom, primarily used for debugging parallel MPI or OpenMP programs. As most debugger, DDT offers sophisticated functions such as running a program step by step (single-stepping) and stopping (breaking). It also features a complete memory debugging tool which can be used to detect memory leaks, or reading and writing beyond the bounds of arrays. Using DDT, it is easy to understand the memory allocation of those hierarchical data structures. For example, we can launch the CLM on a single computing node (with 16 CPUs). Within DDT, we can see how the computational domain has been partitioned across multiple MPI processes. 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. Figure 3 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.

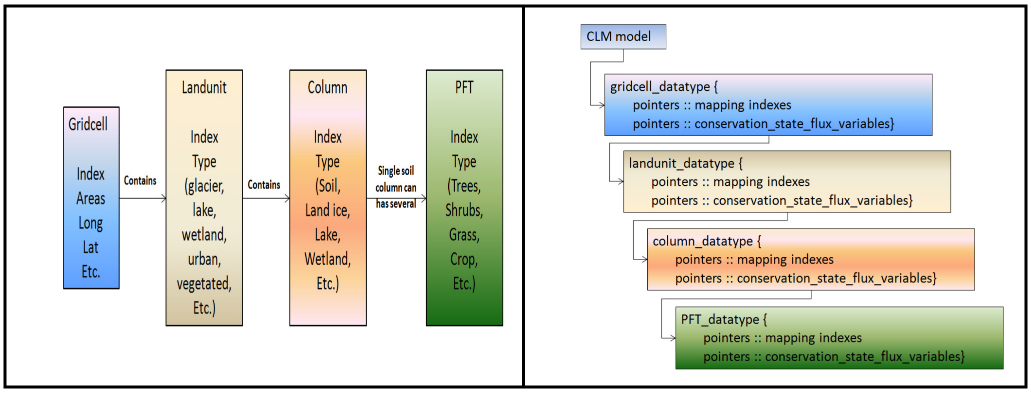
Gridcell
Index
Areas
Long
Lat
Etc.
Contains
Landunit
Index
Type
(glacier,
lake,
wetland,
urban,
vegetated,
Etc.)
Contains
Column
Index
Type
(Soil,
Land ice,
Lake,
Wetland,
Etc.)
Single soil column can has several
PFT
Index
Type
(Trees,
Shrubs,
Grass,
Crop,
Etc.)
CLM model
gridcell_datatype {
pointers :: mapping indexes
pointers :: conservation_state_flux_variables}
landunit_datatype {
pointers :: mapping indexes
pointers :: conservation_state_flux_variables}
column_datatype {
pointers :: mapping indexes
pointers :: conservation_state_flux_variables}
PFT_datatype {
pointers :: mapping indexes
pointers :: conservation_state_flux_variables}

Figure 3: Hierarchical, derived data structure to represent the heterogeneity of the CLM landscape surface. Left graph shows the conceptual data layout. Right graph shows the schematic presentation of the CLM landscape surface data structure view in DDT

4 Computational Characteristics of the CLM

In this section, we focus on the computational performance of the CLM software system. 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 three month simulation is designed to investigate the overall computation and communication ratio and the computational intensity. We use 240 cores (with MPI processes) for the CLM, data atmospheric model and the flux coupler. The stubs for ocean, ice and glacier are executed on one core. Some simulation characteristics are shown in Figure 4. The top graph in Figure 4 shows the general pattern of the CLM simulation on the first 10 processes of the total 240 processes. The green color represents the execution of user application functions, the red color represents MPI activities, and the yellow color represents I/O activities. There is a clear synchronization pattern during the simulation, caused by the I/O and flux communication between CLM and the coupler. The bottom left graph is a pie chart to demonstrate the ratio between computation (application), communication (MPI), and others (I/O, and Vampir overhead). The chart shows that the computation takes around 51% of the time, and around 48 % of the time is spent on communication. The blue color represents less than 1% (percentage data not shown). The bottom right graph shows the computational intensity of CLM, which was determined using PAPI (Performance Application Programming Interface, icl.cs.utk.edu/projects/papi) on one process (#10). The peak Floating-point Operations Per Second (FLOPS) is around 200 million FLOPS and mostly comes from the flux calculation related to the coupler. The average performance rate is around 100 million FLOPS. It is obvious that the current CLM presents a low computation intensity and most important. Those properties are important parts of the ongoing software engineering plan for the CLM.

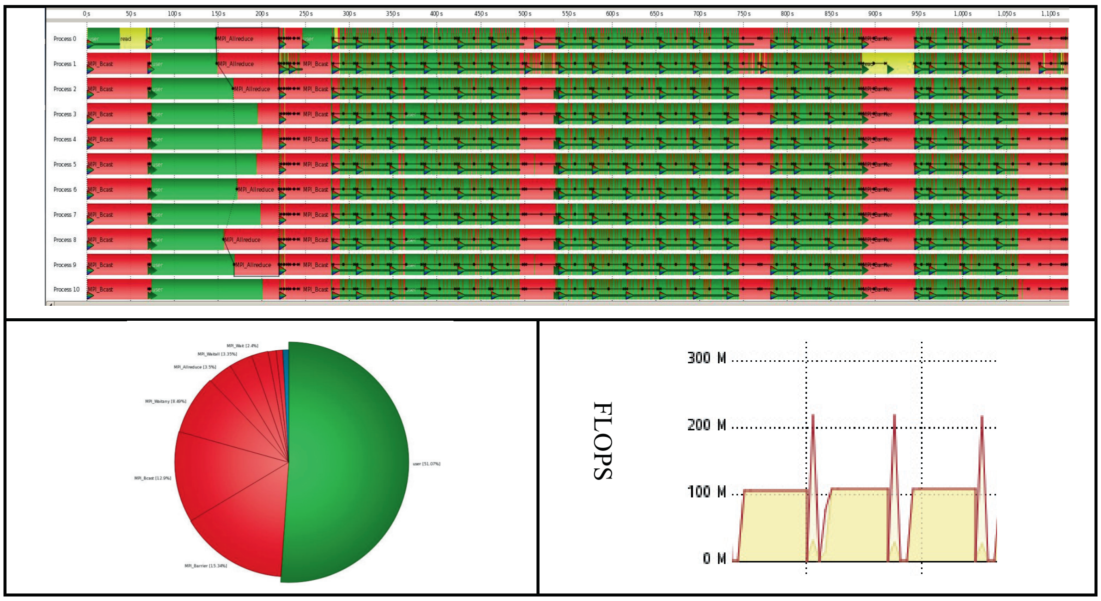

Figure 4: Computational characteristics of the global offline CLM simulation. Top: Computation and communication patterns, Bottom left: Communication/Computation ratio, Bottom right: Computational intensity in FLOPS using PAPI

5 Better Understanding of CLM Ecosystem Function

As a scientific application, it is vital to get the fundamental processes correct in the CLM ecosystem simulations [15,16]. Likewise it is important to investigate new theories of ecosystem function and new process representations within the context of ecosystem sciences. Herein, we present our efforts to understand the CLM individual ecosystem functions via the investigation of scientific context and memory access patterns.

5.1 Investigation of Scientific Context of CLM Ecosystem Function

As mentioned in the previous section, the CLM contains several submodels related to land biogeophysics, the hydrologic cycle, biogeochemistry, and ecosystem dynamics. Within each submodel, the software is generally organized by software modules or subroutines based on natural system functions, for example, radiative fluxes, carbon-nitrogen cycles, momentum and heat flux, soil temperature, hydrology, and photosynthesis, etc. It is generally not easy to develop rapid knowledge on scientific context around specific CLM functions. The landscape surface is the key data structure for the CLM model. After the CLM model initialization, a subset of the landscape surface is allocated in the memory of each computing process. The majority of data arrays and derived datatypes within the hierarchical CLM landscape surface data structure are declared as globally accessible. Therefore, for each individual CLM function (module or subroutines), there are two groups of input and output parameters. In this article, for the clarification, we call all the parameters directly defined by individual function itself as *explicit function parameters*, and parameters which are embedded within the globally accessible landscape surface data structure as *implicit function parameters*.

Due to the complexity of the CLM software and automatically machine configuration, it usually takes almost half an hour to rebuild the software system completely and it is not straightforward to investigate the runtime values of those global parameters using the traditional print functions. Therefore, debuggers, such as the Allinea DDT, can offer great assistance to better understand the scientific context of function parameters (both explicit and implicit). For example, using the debugger, we can easily browse and visualize the values of both explicit and implicit function parameters of any target CLM function. It is also very valuable to have basic statistical information (range, mean, medium, deviation, etc.) of those parameters to determine the data quality in the scientific context. In our study, we also developed a set of scripts to extract and annotate the interrelationships among key CLM ecosystem function calls, their subroutine explicit parameters and implicit parameters (global variables). Based on that information, we can construct a graph, which summarizes the interrelationships among all the function calls, subroutine explicit parameters and global variables. The structure of the graph of *CanopyFluxes* submodel is shown in Figure 5. In this figure, the blue circles represent the ecosystem function calls, such as *Stomata*, *FrictionVelocity* and *CanopyFlux,* etc. The green circles represent the explicit parameters associated with each function, which can be further grouped as inputs and outputs. The yellow circles represents the implicit parameters of each ecosystem functions. Implicit parameters (global variables) are further grouped into three categories: *read_only, modified*, and *write_only*. *Read_only* are global variables used as input parameters only. *Modified* are global variables whose value is changed by the subroutine and their values depend on their values at previous timestep. *Write_only* are global variables whose value is changed by the subroutine, but their values are independent of their values at previous timestep. Interestingly, we found that there are several global variables which are defined as explicit parameters of function calls. Moreover, we also found dead-code (fragments of previous implementations), local data pointer are defined for easy access to global variables but never used during the routine. Red circles are used to show all of those global variables in the graph. The visual structure representation of CLM ecosystem functions is an invaluable tool for CLM model developers to understand the CLM code.

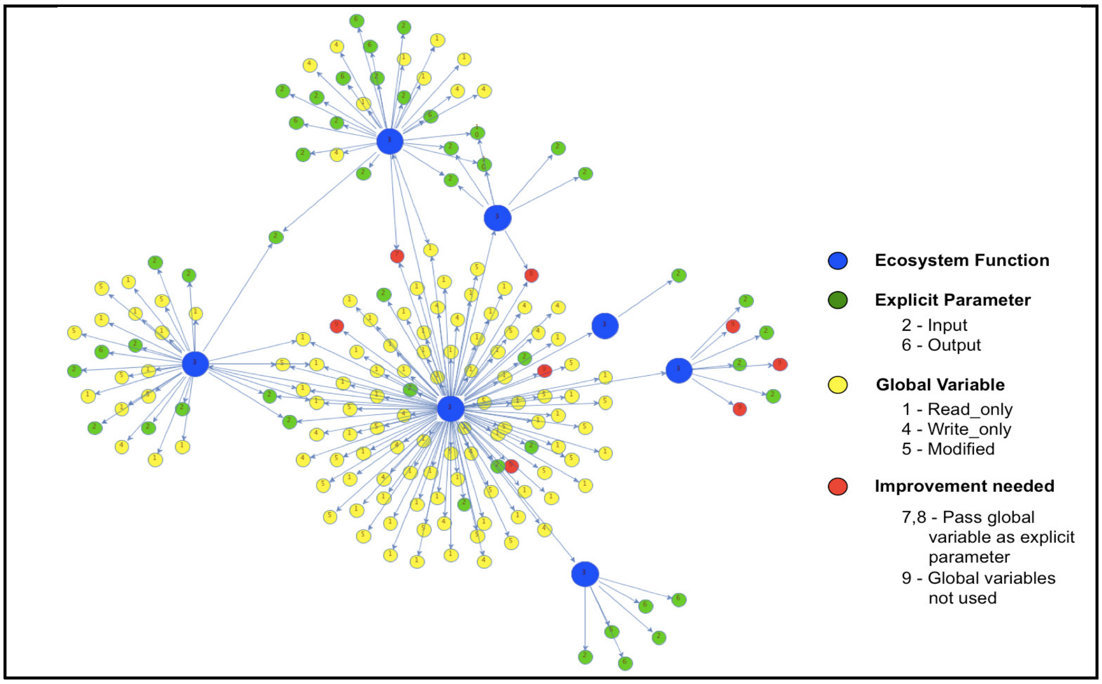

Figure 5: Graph visualization of functional calls (blue) as well as their explicit parameters (green) and implicit parameters/global variable (yellow) within *CanopyFluxes* submodel.

5.2 Analyzing Memory Access Patterns of CLM Function

To analyze CLM's global data-structure access patterns we used the Gleipnir tracing tool [17] built as a plug-in tool for the Valgrind framework (www.valgrind.org). The Gleipnir tool allows the user to trace an application's data accesses and map them to internal structures, or manually inserted identifiers. Gleipnir's unique feature is tracing access patterns at various program granularities (e.g. per process, thread, function, local, global, and heap data structures). Users can use a number of client interface calls, provided by Gleipnir, to tune the instrumentation detail at runtime. External tools supplement the information by visualizing the traces. We can also use the traces for cache simulators to relate data access patterns with cache memory performance. To understand data-structure access pattern variations we traced the CLM application's land component focusing on the *CanopyFluxes* module's *Stomata* function within CLM v.4.0. Our tracing focused on explicit structures because they are the most relevant components of the code. Figure 6 shows an example graph on a single process. The *X-axis* is the number of executed instructions. We chose a granularity of 1 million instructions per tic, thus every bar on the *X-axis* represents 1 million executed instructions. The *Y-axis* is the relative number of data-structure references for every million instructions. The structures are color-coded, which allows us to observe data-structure access pattern during execution. In this example we can observe that most structures are uniformly accessed throughout the function's execution. By applying similar technique to each function, we can establish a benchmark dataset on data access and utilization of each CLM ecosystem functions within current release of CLM. Because there is an ongoing debate within the CLM community on adapting better software data structure to accommodate the mission requirement related to hydrological modeling and vegetation dynamics, etc., comparing access patterns between algorithm changes or CLM updates will help in determining potential access irregularities or unintended behavior for future CLM developments.

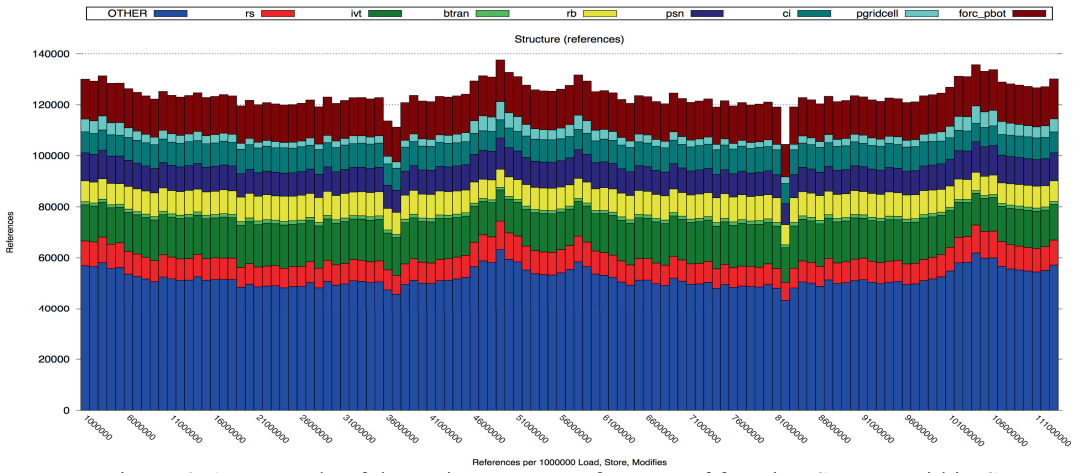

Figure 6: An example of dynamic structure references of function *Stomata* within CLM

6 Conclusions and Future Work

In this paper, we presented our experience in using advanced software engineering approaches, such as software profiling and debugging, to better understand the Community Land Model system within the unified Community Earth System Modeling framework. From the software design perspective, nowadays we have encountered serious challenges in the scientific software applications development due to rapid model development requirements and radical shifts in computing hardware architecture. We are in a great demand of advanced software engineering tools to better understand the fundamentals of software structure and data structure of legacy scientific software applications. A lot of effort is put into the investigation of scientific software performance. In this article, we also presented our efforts to investigate the scientific context as well as the memory access patterns of global CLM landscape surfaces variables associated with individual ecosystem function. The scientific (ecosystem) functionality, which incorporates the state-of-the-science understating of nature and human system, is the most significant and vital fundamentals of scientific software system. We believe that our experience in adapting software engineering approaches for a legacy scientific software system described in this paper can be beneficial to many other research programs involving large scale, legacy modeling systems. Future work will involve with software call-graph generation for each new CLM release as well as graph-based software structure comparison and analysis. The dynamic call-graph generation will be enhanced and extended to better understand the various sequences of function calls which are influence by different environmental conditions. We will investigate possible ways for better visualize dynamic runtime behaviours. We are also in the process of establishing a benchmark database which contains the computation characteristics, scientific context, as well as the global data access patterns of each CLM's ecosystem function .This information will be further integrated into our CLM functional testing platform [15] to serve broad scientific communities.

Acknowledgement: This research was funded by the U.S. DOE, Office of Science, Biological and Environmental Research (BER). This research used resources of the Oak Ridge Leadership Computing Facility, located 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 WM, Parkinson*, CL, An Introduction to Three-Dimensional Climate Modeling, University Science Books,* 2005
2. Oleson K, Lawrence D, Gordon B, Flanner M, Kluzek E, Peter J, Levis S, Swenson S, Thornton P, and Feddema J, Technical description of version 4.0 of the Community Land Model (CLM), 2010, http://www.cesm.ucar.edu/models/cesm1.0/clm/CLM4_Tech_Note.pdf
3. Wang, D., Post, W., Wilson, B., Climate Change Modeling: Computational Opportunities and Challenges, *IEEE Computing in Science and Engineering*, 2011, 13(5), 36-42
4. Graham, SL, Peter B. Kessler, and Marshall K. Mckusick. 1982. Gprof: A call graph execution profiler. In *Proceedings of the 1982 SIGPLAN symposium on Compiler construction* (SIGPLAN '82). ACM, New York, NY, USA, 120-126. DOI=10.1145/800230.806987 http://doi.acm.org/10.1145/800230.806987
5. Spivey JM, Fast, Accurate Call Graph Profiling. *Softw: Pract. Exper*., 2004, 34: 249–264. doi: 10.1002/spe.562
6. Müller MS, Knüpfer A, Jurenz M, Lieber M, Brunst H, Mix H, Nagel WE, Developing Scalable Applications with Vampir, VampirServer and VampirTrace. In: *Parallel Computing: Architectures, Algorithms and Applications*. IOS Press, 2008, 15: 637–644
7. Geimer M, Wolf F, Wylie BJN, Abraham E, Becker D, Mohr B, The Scalasca Performance Toolset Architecture. *Concurrency and Computation: Practice and Experience* 2010, 22(6) 702–719
8. Jerding DF, Stasko JT, and Ball T, 1997. Visualizing Interactions in Program Executions. In *Proceedings of the 19th International Conference on Software Engineering* (ICSE '97). ACM, New York, NY, USA, 360-370. DOI=10.1145/253228.253356 http://doi.acm.org/10.1145/253228.253356
9. Murphy GC, Notkin D, Griswold WG, and Lan, ES, 1998. An empirical study of static call graph extractors. *ACM Trans. Software Engineering and Methodology,* 1998, 7(2), 158-191. DOI=10.1145/279310.279314 http://doi.acm.org/10.1145/279310.279314
10. LaToza TD, Myers, BA, Visualizing Call Graphs, *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 2011, 17-124 doi: 10.1109/VLHCC.2011.6070388
11. Knüpfer A, Brunst H. Doleschal J, M. Jurenz, M. Lieber, H. Mickler, M. S. Müller, W. E. Nagel, The Vampir Performance Analysis Tool-Set, in: M. Resch, R. Keller, V. Himmler, B. Krammer, A. Schulz (Eds.), *Tools for High Performance Computing*, Springer Berlin Heidelberg, 2008, pp. 139–155
12. Shende SS, Malony AD, The Tau Parallel Performance System, *The International Journal of High Performance Computing Applications* 2006, 20,287–331.
13. Domke J, Wang D, Runtime Tracing of the CESM: Feasibility Study and Benefits, *Procedia Computer Science*, 2012(9), 1950-1958
14. Hu YF, Efficient, High-quality Force-Directed Graph Drawing, Mathematica Journal 10.1 (2005): 37-71.
15. Wang D, Xu Y, Thornton P, King A, Steed C, Gu L, Schuchart J, A Functional Test Platform for the Community Land Model, Environmental Modeling and Software, 2014, Volume 55, Pages 25-31, 10.1016/j.envsoft.2014.01.015.
16. Wang D, Ricciuto D, Post W, Berry M, Terrestrial Ecosystem Carbon Modeling, *Encyclopedia for Parallel Computing*, Editor, David A Padua, Springer, 2011, DOI 10.1007/978-0-387-09776-4 , 2034-2039
17. Janjusic T, Kavi KM, and Potter B, Gleipnir: A Memory Analysis Tool, *Procedia Computer Science*, 2011 (4), 2058–2067.