Scientific Research Publishing

Journal of Computer and Communications, 2024, 12, 102-106
https://www.scirp.org/journal/jcc
ISSN Online: 2327-5227
ISSN Print: 2327-5219

Script-Based GPU-Ready ELM Development for Continuous Code Integration

Peter Schwartz*, Dali Wang*, Fengming Yuan, Peter Thornton

Oak Ridge National Laboratory, Environmental Sciences Division, Oak Ridge, USA
Email: *schwartzpd@ornl.gov, *wangd@ornl.gov, yuanf@ornl.gov, thorntonpe@ornl.gov

How to cite this paper: Schwartz, P., Wang, D.L., Yuan, F.M. and Thornton, P. (2024) Script-Based GPU-Ready ELM Development for Continuous Code Integration. *Journal of Computer and Communications*, **12**, 102-106.
https://doi.org/10.4236/jcc.2024.125007

Received: April 10, 2024
Accepted: May 24, 2024
Published: May 27, 2024

Abstract

Designing and optimizing complex scientific code for new computing architectures is a challenging task. To address this issue in the E3SM land model (ELM) development, we developed a software tool called SPEL, which facilitates code generation, verification, and performance tuning using compiler directives within a Function Unit Test framework. In this paper, we present a SPEL extension that leverages the version control system (e.g., Git) to autonomous code generation and demonstrate its application to continuous code integration and development of the ELM software system. The study can benefit the scientific software development community.

Keywords

E3SM Land Model, GPU Code Porting, Continuous Code Integration, SPEL, Scientific Software Refactorization

1. Introduction

State-of-the-art Earth system models (ESM) provide critical information on climate change. There are several fully-coupled ESMs, including the Energy Exascale Earth System Model (E3SM) that uses code optimized for the US Department of Energy's (DOE) advanced computers [1]. The E3SM Land Model (ELM) is an integral part of the E3SM framework, simulating the interactions between terrestrial land surfaces and other Earth system components. ELM has been instrumental in understanding hydrologic cycles, biogeophysics, and dynamics of terrestrial ecosystems [2].

In the development of large-scale, ultrahigh-resolution ELM (uELM), the researchers have implemented the GPU-ready ELM code using OpenACC on Summit at Oak Ridge National Laboratory [3]. To facilitate the automatic port-

*Both authors contributed equally.

ing of individual ELM modules, the researchers have developed a software tool named SPEL [4]. SPEL works within a Functional Unit Testing framework [5] and contains a collection of functions to parse ELM code, generate code segments, analyze dataflow, and support code verification. This paper presents the extension of SPEL functions to support continuous ELM integration based on code change information harvested from Git commits.

2. SPEL Extension to Support Continuous ELM Code Development

ELM uses GitHub to facilitate its full-cycle software development, including design, development, quality assurance, deployment, and maintenance. Currently, SPEL provides functions for ELM code analysis and GPU code generation, as well as optimization (**Figure 1**). The SPEL software package consists of four major functions. The basic function involves parsing and code generation, initializing ELM, capturing module parameters, and generating Fortran code. The unit test creation function is responsible for creating a unit test driver, managing data input and output, and executing code. The GPU porting function handles GPU data regions, offloading to the GPU, and performance tuning. Lastly, the verification function is used for verifying testing results.

Our objective is to expand SPEL's capabilities to support continuous integration of ELM code. To achieve this, we have developed a script-based function that follows a two-step workflow. The first step involves analyzing and processing information from the Git commit output of the CPU code base. The second step involves calling SPEL's functions to generate GPU-ready code for new modifications. This function skips all unchanged OpenACC segments, examines the structure and scope of code modifications, and generates code with appropriate directives (clauses) for these altered sections. The workflow of using SPEL GitUtil is also shown in **Figure 1**.

The ELM code can undergo three major types of change:

1) Change to land subgrid datatypes (such as the addition of topographic units) and associated ELM processes. This modification impacts many ELM processes, including Canpoy FLux, Ecosystem Dynamics, and Soil Temperature.

2) Change to ELM process parameters (such as the addition of parameters for snowmelt) and associated functions. This type of change requires adjustments in

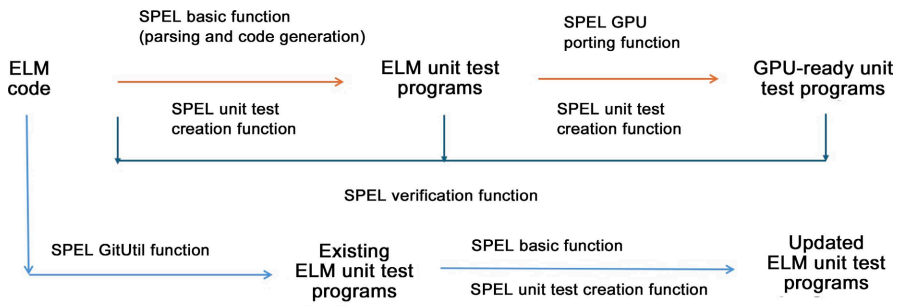


Figure 1. SPEL functions used for uELM code porting within a Functional Unit Testing framework.

GPU data region creation and ELM function kernels and impacts limited ELM processes.

3) Change to individual ELM functions (such as subroutines). This type of change results in data and code changes in specific subroutines. The third type of change is considered as the basic ELM code change.

3. Code Generation for Modifications in a Single ELM Source File

This paper presents a code generation procedure for VerticalProfileMod.F90 that holds routines for the vertical discretization of C and N inputs into decomposing pools. It is an example of the basic ELM code change without new data (variable) introduced. There are four steps:

1) Find the change history of the module (*i.e.*, **git log—follow—VerticalProfileMod.F90**).

2) Select a specific commit (*i.e.*, **067be7cbc99cd349770d4eeb827-0b58f47498dfe**, committed by Michael A. Brunke on June 10, 2020, to add "profile adjustments w/ variable soil in VerticalProfileMod").

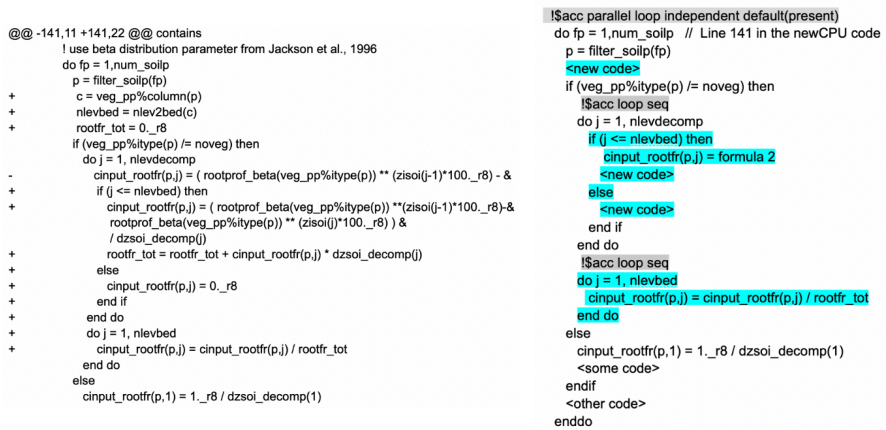
3) Extract the modified file (*i.e.* **git show {commit}**) and save the Git change logs. Four code sections in the VerticalProfileMod.F90 were changed, annotated by special symbols "@@" (Table 1).

4） Develop a new SPEL function (**gitutils**) use **regex** to analyze the Git output file (*i.e.*, list of filenames, changed sections), mark the region of code change in the existing GPU code, remove all the OpenACC directives in the region, modify the code (based on Git commit), then call SPEL functions to generate GPU-ready code only for the changed sections.

For illustration purpose, the first code change section (starting at line 141) (left) and the newly generated GPU-ready code (right) is shown in Figure 2. **Gitutils** first parses the code change section and determines the impacted region of the code change (the outer loop that beyond the texts in the commit log), secondly removes existing **acc** statements in the target region (first two **acc** statements), and then changes the code (the old 11-line code became a new 22-line code with new statements (cyan)), finally uses appropriate SPEL functions to regenerate the GPU code (add three **acc** statements (gray) at the right places).

Table 1. Changed code sections in VerticalProfileMod.F90.

Changed code section	Purpose
@@ −141,11 +141,22 @@ contains	recalculate fine-root distribution in soil and bedrock
@@ −171,7 +182,12 @@ contains	adjust root profile integration
@@ −181,8 +197,15 @@ contains	adjust leaf and stem profile integration
@@ −270,7 +295,7 @@ contains	adjust error reporting for new bedrock layers



```
@@ -141,11 +141,22 @@ contains
          ! use beta distribution parameter from Jackson et al., 1996
          do fp = 1,num_soilp
            p = filter_soilp(fp)
+           c = veg_pp%column(p)
+           nlevbed = nlev2bed(c)
+           rootfr_tot = 0._r8
            if (veg_pp%itype(p) /= noveg) then
              do j = 1, nlevdecomp
-               cinput_rootfr(p,j) = ( rootprof_beta(veg_pp%itype(p)) ** (zisoi(j-1)*100._r8) - &
+               if (j <= nlevbed) then
+                 cinput_rootfr(p,j) = ( rootprof_beta(veg_pp%itype(p)) **(zisoi(j-1)*100._r8)-&
+                  rootprof_beta(veg_pp%itype(p)) ** (zisoi(j)*100._r8) ) &
+                  / dzsoi_decomp(j)
+                 rootfr_tot = rootfr_tot + cinput_rootfr(p,j) * dzsoi_decomp(j)
+               else
+                 cinput_rootfr(p,j) = 0._r8
+               end if
              end do
+             do j = 1, nlevbed
+               cinput_rootfr(p,j) = cinput_rootfr(p,j) / rootfr_tot
            end do
          else
            cinput_rootfr(p,1) = 1._r8 / dzsoi_decomp(1)
```

```
!$acc parallel loop independent default(present)
do fp = 1,num_soilp  // Line 141 in the newCPU code
  p = filter_soilp(fp)
  <new code>
  if (veg_pp%itype(p) /= noveg) then
    !$acc loop seq
    do j = 1, nlevdecomp
      if (j <= nlevbed) then
        cinput_rootfr(p,j) = formula 2
        <new code>
      else
        <new code>
      end if
    end do
    !$acc loop seq
    do j = 1, nlevbed
      cinput_rootfr(p,j) = cinput_rootfr(p,j) / rootfr_tot
    end do
  else
    cinput_rootfr(p,1) = 1._r8 / dzsoi_decomp(1)
    <some code>
  endif
  <other code>
enddo
```

Figure 2. The input Git commit log (left) and the new GPU-ready code (right) for the first changed code section in VerticalProfileMod.F90.

4. Conclusion

This study successfully extended the SPEL functions to generate GPU-ready code for continuous ELM integration. The exemplar case demonstrated the effectiveness of the GPU-code generation procedure based on the GitHub change logs. This study laid the foundation for more complicated cases via iterative approaches, which will be reported in future publications.

Acknowledgements

This research was supported as part of the Energy Exascale Earth System Model (E3SM) project, funded by the U.S. Department of Energy, Office of Biological and Environmental Research. This research used resources from the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory.

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

[1] Golaz, J.-C., Caldwell, P.M., Van Roekel, L.P., Petersen, M.R., Tang, Q., Wolfe, J.D., Abeshu, G., Anantharaj, V., Asay-Davis, X.S., Bader, D.C., *et al.* (2019) The DOE E3SM Coupled Model Version 1: Overview and Evaluation at Standard Resolution. *Journal of Advances in Modeling Earth Systems*, **11**, 2089-2129. https://doi.org/10.1029/2019MS001870

[2] Burrows, S.M., Maltrud, M., Yang, X., Zhu, Q., Jeffery, N., Shi, X., Ricciuto, D., Wang, S., Bisht, G., Tang, J., *et al.* (2020) The DOE E3SM v1.1 Biogeochemistry Configuration: Description and Simulated Ecosystem-Climate Responses to Historical Changes in Forcing. *Journal of Advances in Modeling Earth Systems*, **12**, e2019MS001766. https://doi.org/10.1029/2019MS001766

[3] Wang, D.L., Schwartz, P., Yuan, F.M., Thornton, P. and Zheng, W.J. (2022) Towards Ultra-High-Resolution E3SM Land Modeling on Exascale Computers. *Computing in Science & Engineering*, **1**, 1-14.

[4] Schwartz, P., Wang, D.L., Yuan, F.M. and Thornton, P. (2022) SPEL: Software Tool for Porting E3SM Land Model with OpenACC in a Function Unit Test Framework. 2022 *Workshop on Accelerator Programming Using Directives* (*WACCPD*), Dallas, USA, 13-18 November 2022, 1-14. https://doi.org/10.1109/WACCPD56842.2022.00010

[5] Wang, D.L., Xu, Y., Thornton, P., King, A., Steed, C., Gu, L.H. and Schuchart, J. (2014) A Functional Test Platform for the Community Land Model. *Environmental Modelling & Software*, **55**, 25-31. https://doi.org/10.1016/j.envsoft.2014.01.015