Adventures in Vectorizing the Community Land Model

Forrest M. Hoffman
Climate and Carbon Research Institute (CCRI)
Oak Ridge National Laboratory
P.O. Box 2008
Oak Ridge, Tennessee 37831–6008
forrest@climate.ornl.gov

Mariana Vertenstein
National Center for Atmospheric Research (NCAR)

Hideyuki Kitabata
Central Research Institute of Electric Power Industry (CRIEPI)

James B. White III, Patrick Worley, and John Drake
Oak Ridge National Laboratory

Matthew Cordery
Cray Inc.

Abstract

Described here are the extensive efforts of the authors to modify the Community Land Model for vectorization on the Earth Simulator in Japan and the Cray X1 at Oak Ridge National Laboratory. This paper follows experimental results presented at the Cray Users Group (CUG) Meeting in 2003 (White, 2003). Presented here are the technical details of the old and new internal data structures, the required code reorganization, and the resulting performance improvements. Additionally, performance and scaling of the final Community Land Model Version 3 (CLM3) on the IBM Power4, the Earth Simulator, and the Cray X1 are compared.

1 Introduction

The Community Land Model (CLM) is a single column (snow-soil-vegetation) biogeophysical model of the land surface. Written in Fortran 90, CLM can be run offline (*i.e.*, run in isolation using stored atmospheric forcing data), coupled to an atmospheric model (*e.g.*, the Community Atmosphere Model), or coupled to a climate system model (*e.g.*, the Community Climate System Model) through a flux coupler (*e.g.*, Coupler 6). When coupled, CLM exchanges fluxes of energy, water, and momentum with the atmosphere.

Originally developed as the standalone Common Land Model (Dai et al., 2003), the code was significantly modified for integration with the Community Atmosphere Model (CAM) (Bonan et al., 2002) and the Community Climate System Model (CCSM) (Kiehl and Gent, 2004) in 2001 and 2002. This prior development work targeted cache-based scalar multi-processor computer platforms and resulted in code which would not vectorize. The availability of the Earth Simulator in Japan and the Cray X1 at Oak Ridge National Laboratory spawned renewed interest in running climate models—including CLM—on vector architectures. This lead to the need to rewrite the model to provide good performance on both vector and scalar architectures in 2003.

The primary obstacles to vectorization were the layout of the internal data structures and the organization of the processing loops in CLM2.1. The internal data structures were based on a hierarchy of pointers to derived data types each containing scalar quantities. This layout resulted in large, unpredictable strides among variable elements. The model had many high level loops over various grid and subgrid units. In each loop, science subroutines were called for each grid or subgrid unit member. The loops within the science subroutines were all short and contained negligible work.

Early vectorization experiments carried out separately by Kitabata on the Earth Simulator and White on the Cray X1 demonstrated significant performance improvement by eliminating the hierarchy of derived data types in favor of traditional Fortran arrays and by pushing long grid and subgrid unit loops "down" into science subroutines where they were then typically interchanged with short loops (White, 2003). While these experiments were being carried out, a strategy for rewriting the CLM code was being developed to meet the requirements put forth by the research community. Researchers wanted a single CLM code that would run well on both vector and scalar architectures while maintaining the hierarchical nature of the internal data structures. The use of conditionally compiled code blocks (using `#ifdef` preprocessor macros) was to be minimized, and the code modifications could not reduce the performance on the existing scalar platforms.

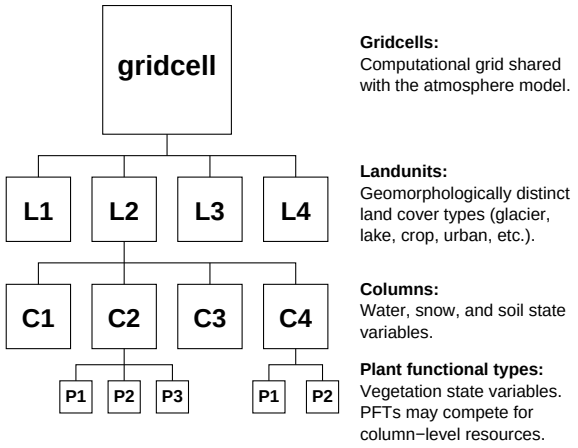

Figure 1: The CLM subgrid hierarchy

2 Data Structures

2.1 Hierarchy of Grid Scales

The horizontal land surface heterogeneity in CLM is represented by a nested subgrid hierarchy composed of gridcells, landunits, columns, and plant functional types (PFTs) as shown in Figure 1. Each gridcell can have a different number of landunits, each landunit can have a different number of columns, and each column can have multiple PFTs. Gridcells represent the computational grid which is shared with the atmospheric physics.

The landunit, the first subgrid level, is intended to capture the broadest spatial pattern of subgrid heterogeneity. It serves primarily to distinguish physical soil properties. Specific landunits include glacier, lake, wetland, urban, and vegetated. The column captures variability in soil and snow state variables within a landunit. Water and energy states and fluxes are tracked at the column level. The PFT, the third subgrid level, captures the characteristic biophysical and biogeochemical functions of broad categories of vegetation and bare soil. Up to four out of 15 possible PFTs differing in physiology and structure may be contained within a single column.

Biophysical processes are simulated for each subgrid unit independently, and prognostic variables are maintained for each subgrid unit. Processes related to soil and snow require PFT-level properties to be aggregated up to the column level. Aggregation is usually accomplished by computing a weighted sum for each quantity over all PFTs within a column. Similarly, different PFTs compete for the resources tracked at the column level. A complete description of the biophysical processes simulated by CLM is available in the *Technical Description of the Community Land Model (CLM)* (Oleson et al., 2004).

This hierarchical subgrid organization is reflected in the data structures used in the model code. In CLM2.1, the hierarchy was implemented as arrays of pointers to derived data types at each subgrid level. The flux and state variables were implemented as scalars in every instance of a derived data type. While this object-style layout is reasonable for cache-based scalar platforms, it is not conducive to vector processing. After considerable discussion and experimentation on the Cray X1, a new organization was developed which would retain the hierarchy in the data structures while allowing for loop vectorization.

Using a testing methodology developed in the vectorization experiments performed by White, the new proposed data structures were added to the existing code. A vectorized version of the most costly code branch, the `Biogeophysics1()` routine which calls the computationally intensive `CanopyFluxes()` subroutine, was also added to the code. In this experiment, data were copied from the original data structures into the new data structures, and then the original `Biogeophysics1()` subroutine was called, followed by the vectorized version of the subroutine which used the new data structures. For each time step in the model run, after the non-vector and vector subroutines were called, the results contained in the old and new data structures were compared to ensure only round-off differences.

Timing utilities included in the CLM code were used to measure performance differences between the original and the new vector versions of the code branch. Using the proposed hierarchical data structures, the performance improvement matched that of prior vectorization experiments (White, 2003). The strategy for further vectorization was to use the same testing methodology for each code branch in turn to ensure that model results were equivalent at each time step and to monitor performance improvement. As vectorization progressed, the code would be tested by Kitabata on the Earth Simulator and NEC platforms to ensure similar performance gains

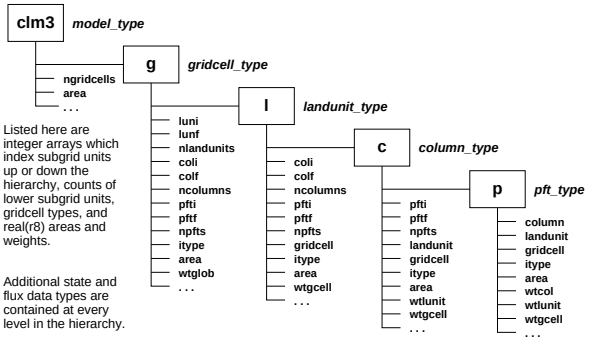

Figure 2: The new CLM3 data structure hierarchy

on these systems. The new data structures and vectorization strategy were subsequently presented to and approved by the Land Model Working Group and NCAR researchers in the summer of 2003.

The new data structures now in CLM3 consist of derived data types for each subgrid unit as shown in Figure 2. Each grid and subgrid unit data type in the hierarchy contains a number of physical and chemical state and flux data types. These data types typically contain real arrays for the state and flux variables which were previously represented as scalars in multiple instances of data types. Vertical heterogeneity is represented by a single vegetation layer, 10 layers for soil, and up to five layers for snow, depending on the snow depth. Multi-layer quantities are stored as two-dimensional real arrays. Using real arrays at every level in the hierarchy maximizes opportunities for contiguous memory access, thereby providing significantly better performance on vector architectures. All arrays contained in derived data types are implemented as Fortran 90 pointers. Memory for these arrays is allocated dynamically during model initialization.

Each of these grid and subgrid data types also contains arrays of integers which serve as array indices to the higher subgrid levels or as initial and final bounds on the lower subgrid levels. For example, as shown in Figure 2, the column-level data type contains `landunit` and `gridcell` integer arrays which refer to the appropriate landunit and gridcell for a given column. The column-level data type also contains integer arrays named `pfti`, `pftf`, and `npfts` which refer to the first and last PFTs and the total number of PFTs, respectively, for a given column. Each subgrid data type also contains real arrays for surface areas and area weights at all higher grid levels.

2.2 Decomposition and Clumps

CLM uses MPI (the Message Passing Interface) for distributed memory parallelism and uses OpenMP for shared memory parallelism. On the Cray X1, the model can also stream across processing units (called Single Streaming Processors, or SSPs) within a Multi-Streaming Processor (MSP) based on Cray Streaming Directives (CSDs) included in the new code. In order to provide performance portability across a wide range of current and future computer architectures, a decomposition strategy partially implemented in CLM2.1 was fully developed in CLM3.

When compiled for distributed memory parallelism (with the preprocessor macro `SPMD` defined), each MPI process will create an instance of the data structures shown in Figure 2 containing only the subset of data assigned to that process. A cache-friendly blocking structure is superimposed on the data structure hierarchy for improved computational efficiency. This blocking structure implicitly controls the vector length of most computations. Gridcells are grouped into blocks (called "clumps") of nearly equal computational cost, and these clumps are subsequently assigned to MPI processes.

The computational cost of a gridcell is approximately proportional to the number of PFTs contained within it. However, since computational cost for some PFTs is higher than for others and since similar PFTs tend to cluster geographically, balancing the workload across MPI processes requires a more complex scheme than simply assigning contiguous blocks of gridcells to clumps. To minimize the potential for load imbalance, gridcells are assigned in cyclic (or round robin) fashion to a pre-determined number of clumps. The clumps are then assigned in cyclic fashion to available MPI processes. This scheme has proven to sufficiently distribute gridcells of various costs among MPI processes, yielding very good parallel load balancing characteristics for most process counts and surface datasets.

Clumps not only define the workload for an MPI process, they also serve to block data for shared memory parallelism when using OpenMP or streaming on the Cray X1. The number of clumps per MPI process is determined by the parallel configuration of the model at run time, but it may be set explicitly by setting the `clump_pproc` namelist variable to the desired number of clumps per process. When run serially or with MPI-only parallelism, one clump per process is used. When OpenMP is enabled, the number of clumps per process is set to the maximum number of OpenMP threads available. On the Cray X1 when OpenMP is disabled, CSDs are interpreted by the compiler in place of OpenMP directives, and the number of clumps per process is set to four to take maximum advantage of the four SSP units on an MSP.

2.3 Filters

In addition to clumps, another set of structures, called "filters," was added to better support vectorized processing of columns and PFTs. Filters group like columns or PFTs based on their process-specific categorization and are used for indirect addressing into the main data structure hierarchy. Filters are created for snow, non-snow, lake, non-lake, and bare soil columns and PFTs for each clump of gridcells. Most filters are initialized once, but the snow and non-snow filters must be reconstructed as snowfall and melting occur.

3 Code Reorganization

Loops over columns and PFTs previously located in the top level `driver` routine were moved down into science subroutines to provide opportunities for vectorization. In CLM3, the highest level loops in the `driver` routine run over clumps for each MPI process and provide for OpenMP and Cray Streaming parallelism. Science subroutines, called within these loops, are passed local clump bounds for gridcells, landunits, columns, and PFTs as needed. Relevant filters, in the form of counts and vectors of array indices, are also passed as needed to science subroutines.

Figure 3 shows a portion of a high level loop from the `driver` routine. First, the number of clumps assigned to the process is obtained and stored in `nclumps`. The subsequent loop over all clumps is wrapped with OpenMP and Cray Streaming directives to support shared memory parallelism. Within the loop, the bounds for gridcells, landunits, columns, and PFTs are obtained for the clump being processed by calling `get_clump_bounds()`. Then a science subroutine, `Hydrology1()`, is called and passed the column and PFT bounds as well as the non-lake filters for columns and PFTs for the clump being processed. Additional science subroutines are subsequently called within the same loop. The `driver` routine consists primarily of two such high level loops which call most of the science subroutines used by the model.

Within science subroutines, vector loops run over grid or subgrid units. Other short loops run over snow and soil levels within a column or PFT. In most

```
nclumps = get_proc_clumps()
!$OMP PARALLEL DO PRIVATE (nc,begg,endg, &
!$OMP & begl,endl,begc,endc,begp,endp)
!CSD$ PARALLEL DO PRIVATE (nc,begg,endg, &
!CSD$ & begl,endl,begc,endc,begp,endp)
do nc = 1,nclumps
  call get_clump_bounds(nc, begg, endg, &
       begl, endl, begc, endc, begp, endp)
    .
    .
    .
  call Hydrology1(begc, endc, &
       begp, endp, &
       filter(nc)%num_nolakec, &
       filter(nc)%nolakec, &
       filter(nc)%num_nolakep, &
       filter(nc)%nolakep)
    .
    .
    .
end do
!$OMP END PARALLEL DO
!CSD$ END PARALLEL DO
```

Figure 3: Example high level loop in the `driver` routine

cases, the vector loops were inserted into the short loops for vectorization. Many loops were split into multiple loops and temporary local arrays were used to "carry" data between them. Many of the vector loops use filters for indirect addressing of relevant columns and PFTs. Since arrays in data structures are implemented as pointers, compilers can not determine if vector dependencies exist. As a result, compiler directives are required in order to obtain loop vectorization.

Figure 4 shows an example of a filter loop within a science subroutine. First, local pointers are created to shorten the notation used in equations. The subsequent loop over all non-lake columns is preceded by Cray X1 and Earth Simulator compiler directives. The first directive tells the Cray X1 compiler that the loop is concurrent, meaning it may be streamed and vectorized. The second directive tells the Earth Simulator compiler that no vector dependencies exist in the loop. Within the loop, the column index is obtained from the non-lake column filter vector, the appropriate landunit index is obtained from the column's landunit vector, and the appropriate gridcell index is obtained from the column's gridcell vector. Next, the landunit type and the ground temperature of the column are checked.

```
! Assign local pointers to derived type
! members (landunit-level)
clandunit => clm3%g%l%c%landunit
itype     => clm3%g%l%itype
! Assign local pointers to derived type
! members (column-level)
cgridcell => clm3%g%l%c%gridcell
t_grnd    => clm3%g%l%c%ces%t_grnd
h2osno    => clm3%g%l%c%cws%h2osno
snowdp    => clm3%g%l%c%cps%snowdp
snowage   => clm3%g%l%c%cps%snowage

!dir$ concurrent
!cdir nodep
do f = 1, num_nolakec
  c = filter_nolakec(f)
  l = clandunit(c)
  g = cgridcell(c)
    .
    .
    .
  if (itype(l) == istwet .and. &
      t_grnd(c) > tfrz) then
      h2osno(c) = 0._r8
      snowdp(c) = 0._r8
      snowage(c) = 0._r8
  end if
    .
    .
    .
end do
```

Figure 4: Example filter loop in a science subroutine

If the landunit contains water and the ground temperature is above freezing, three variables are initialized to zero. Other computations are usually performed within such loops.

4 Vector Performance

Preliminary vectorization of CLM was completed in October 2003. The new vectorized model has a smaller memory footprint. The new data structures simplify history updates and reduce the complexity and number of MPI gathers and scatters. The new vectorized model runs 25.8 times faster than the CLM2.1 code on the Cray X1 and even runs 1.8 times faster on the IBM Power4 in offline mode. To gauge overall performance of the new CLM3 and determine the optimum run-time configuration, timing tests were performed on the IBM Power4, the

Earth Simulator, and the Cray X1. The timing tests consisted of a series of 30 day offline runs at T85 resolution varying both processor counts (with and without OpenMP) and the clumps-per-process tuning parameter. The source code used for the tests was that tagged `clm2_deva_51` in the NCAR CVS repository.

CLM3 may be built with a variety of options depending on the computer architecture and processor configuration to be used. Both MPI and OpenMP may be independently enabled or disabled. With both MPI and OpenMP disabled, the model can run serially on a single processor. With MPI enabled and OpenMP disabled, the model runs in distributed memory mode across a number of nodes. With MPI disabled and OpenMP enabled, the model can run on a single shared memory symmetric multiprocessor (SMP) node. With both MPI and OpenMP enabled, CLM3 runs in hybrid distributed/shared memory mode across a number of SMP nodes.

4.1 Cray X1

On the Cray X1, CLM3 can be built with CSDs enabled; however, CSDs are used only around loops which also use OpenMP directives. As a result, OpenMP can be enabled only when CSDs are not. When compiling with OpenMP (ignoring CSDs), the compiler will still multi-stream concurrent loops in science subroutines where possible. Calls to these science subroutines are typically contained within the high level loops in the `driver` routine where the OpenMP directives and CSDs are located. Slight modification of these loops allows simultaneous use of OpenMP and CSDs. The performance impacts of CSDs versus OpenMP on the Cray X1 are described below.

For the Cray X1 with OpenMP disabled, the number of MPI processes is equal to the number of MSPs used. Four threads were used for all tests with OpenMP enabled on the Cray X1. The curves in the performance graphs presented here are all colored by the total number of processors used in the run. For the Cray X1, a processor refers to one MSP. The clumps-per-process tuning parameter was varied between 1 and 32 for all performance tests.

Figure 5 shows total run times for CLM3 on the Cray X1 with both OpenMP and CSDs disabled for 1 to 128 MSPs. The number of MPI processes is equal to the number of MSPs, except that MPI was disabled for the 1 MSP case. For this test, streaming across SSPs occurs only in concurrent loops within science subroutines. The best performance is obtained when clumps per process is set to one, since this setting maximizes the vector length. Scaling beyond 16 MSPs is poor, but run time continues to drop through 64 MSPs. Performance for 128 MSPs is no better than for 64 MSPs.

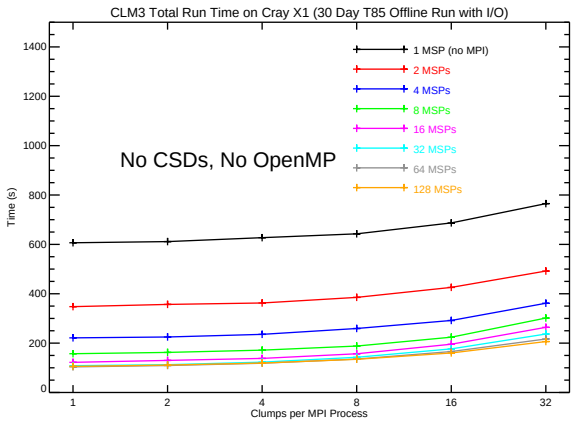

Figure 5: Curves show total run times for CLM3 on the Cray X1 with OpenMP and CSDs both disabled for 1 to 128 MSPs.

Figure 6 shows total run times for CLM3 on the Cray X1 with OpenMP disabled and CSDs enabled for 1 to 128 MSPs. The number of MPI processes is equal to the number of MSPs, except that MPI was disabled for the 1 MSP case. For this test, streaming across SSPs occurs at the high level loops in the `driver` routine. The best performance is obtained when clumps per process is set to four since this setting maximizes the vector length for each of the four SSPs. Scaling beyond 32 MSPs is poor, but run time continues to drop through 64 MSPs. Performance for 128 MSPs is just slightly worse than for 64 MSPs.

Figure 7 shows total run times for CLM3 on the Cray X1 with OpenMP enabled and CSDs disabled for 4 to 256 MSPs. Four threads were used per MPI process, and the number of MSPs shown is equal to the product of the number of MPI processes and the number of threads. MPI was disabled for the 4 MSP case. For this test, streaming across SSPs occurs only in concurrent loops within science subroutines. The best performance is obtained when clumps per process is set to four, since this setting maximizes the vector length for each of the four shared memory MSPs. Scaling beyond 64 MSPs is poor, but run time continues to drop through 128 MSPs.

In all three cases, the best performance is obtained when vector lengths are maximized. On average, performance is improved by 20% using clump-

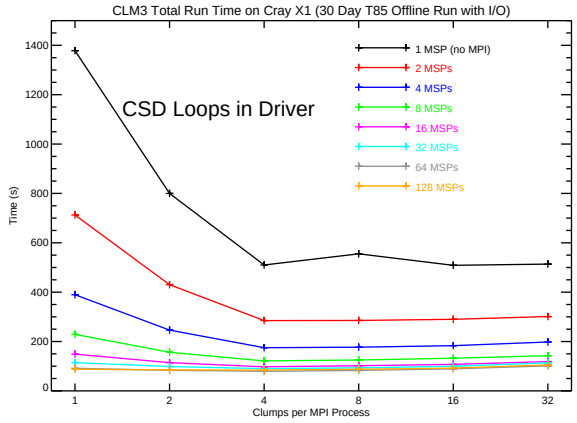

Figure 6: Curves show total run times for CLM3 on the Cray X1 using Cray Streaming Directives around high level loops in the `driver` routine for 1 to 128 MSPs.

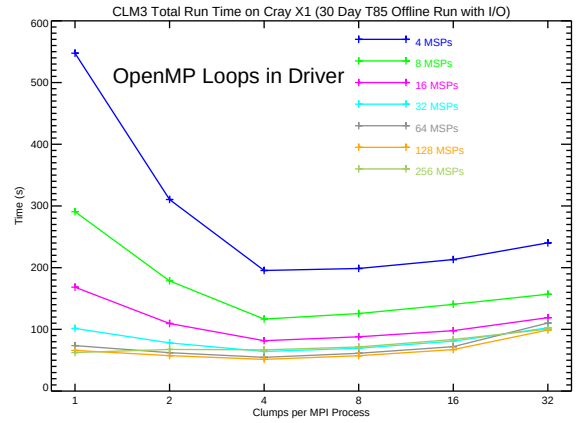

Figure 7: Curves show total run times for CLM3 on the Cray X1 using OpenMP directives around high level loops in the `driver` routine for 4 to 256 MSPs. Four threads were used per MPI process, so the number of processors shown is equal to the product of the number of MPI processes and the number of threads.

ing and enabling CSDs around high level loops to explicitly utilize the four SSPs on each MSP. However, better scaling is obtained by using OpenMP instead of CSDs for the same high level loops in the `driver` routine. MPI performance and problem size limit distributed memory scaling to 64 MSPs, but adding shared memory processors via OpenMP improves scaling to 128 MSPs (*i.e.*, 32 MPI processes and 4 OpenMP threads). It appears that the compiler does a good job of multi-streaming loops in science subroutines (thereby keeping the SSPs sufficiently busy). This fact, combined with the reduced MPI communications, explains why using OpenMP to parallelize high level loops results in better performance beyond 8 MSPs.

In an effort to achieve additional performance on the Cray X1, the high level loops in the `driver` routine were slightly modified so that OpenMP and Cray Streaming could be used simultaneously. The code modification is shown in the listing in Figure 8. For this case, the optimum number of clumps per process is 16: four for the four OpenMP threads times four for the four SSPs.

As shown in Figure 9, this change slightly improves the performance of the OpenMP-only case up through 64 MSPs. Beyond 64 MSPs, the addition of CSD loops within OpenMP loops does not further improve model performance. The overall performance improvement from using OpenMP and CSDs simultaneously was smaller than expected because the vector lengths become too short (*i.e.*, the problem size is too small) and the overhead of MPI communications limits further scaling. More importantly, this result is seen as confirmation that the automatic streaming of concurrent loops within science subroutines is very good. The SSPs are kept busy most of the time.

Profiling experiments were performed to further investigate performance characteristics of CLM3 on the Cray X1. Table 1 contains a complete profile report of CLM3 running on a single MSP with CSDs enabled. This report provides a good overview of the routines that consume the largest amount of run time. The most costly routine is `gettimeofday` which is used by the timing utilities. It is known that `gettimeofday` is particularly expensive on the Cray X1. Running with timers disabled improves model performance by 10–15%. The second routine, `%__rtor_vv`, appears to be a real-to-real vector memory copy routine. The third routine, `canopyfluxes`, is the most expensive scientific subroutine. This routine is known to represent the majority of the calculations in the present version of CLM, and it has

```
!$OMP PARALLEL DO PRIVATE (gnc,nc_beg, &
!$OMP & nc_end,nc,begg,endg,begl,endl, &
!$OMP & begc,endc,begp,endp)
do gnc = 1,nclumps/4
  nc_beg = (gnc - 1) * 4 + 1
  nc_end = min(nc_beg+3,nclumps)
!CSD$ PARALLEL DO PRIVATE (nc,begg, &
!CSD$ & endg,begl,endl,begc,endc, &
!CSD$ & begp,endp)
  do nc=nc_beg,nc_end
    call get_clump_bounds(nc, begg, endg, &
         begl, endl, begc, endc, &
         begp, endp)
       .
       .
       .
    call Hydrology1(begc, endc, &
         begp, endp, &
         filter(nc)%num_nolakec, &
         filter(nc)%nolakec, &
         filter(nc)%num_nolakep, &
         filter(nc)%nolakep)
       .
       .
       .
  end do
!CSD$ END PARALLEL DO
end do
!$OMP END PARALLEL DO
```

Figure 8: Modified clump loop from `driver` providing both OpenMP and Cray Streaming directives

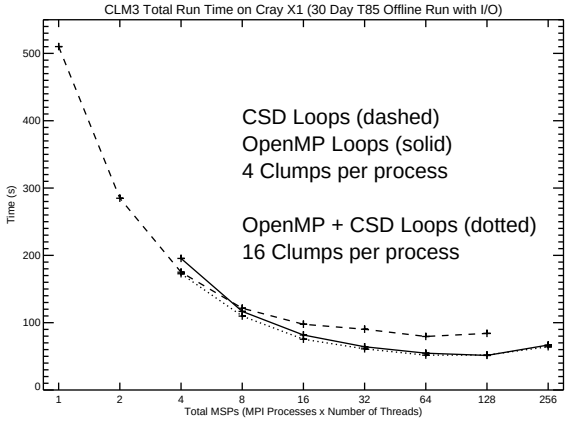

Figure 9: CLM3 total run time on the Cray X1 using CSD loops (dashed), OpenMP loops (solid), and OpenMP with CSDs simultaneously (dotted).

```
   Samp% | Cum.Samp% |  Samp |SSP=0
                              |Function
                              |Caller
|-------------------------------------
| 100.0% |    100.0% | 59330 |Total
|-------------------------------------
|  10.7% |     10.7% |  6368 |gettimeofday
|  10.0% |     20.8% |  5944 |%__rtor_vv
|   7.1% |     27.9% |  4233 |canopyfluxes@canopyfluxesmod_
|   6.7% |     34.6% |  3966 |updateinput@rtmmod_
|   5.6% |     40.1% |  3293 |%__alog
|   5.3% |     45.4% |  3135 |areaovr_point@areamod_
|   5.0% |     50.4% |  2966 |pft2col@pft2colmod_
|   4.6% |     55.0% |  2725 |rtmriverflux@rtmmod_
|   4.3% |     59.3% |  2540 |mkmxovr@areamod_
|   2.7% |     62.0% |  1585 |soiltemperature@soiltemperatur
|   2.6% |     64.6% |  1543 |areaave@areamod_
|   2.4% |     67.0% |  1433 |phasechange@soiltemperaturemod
|   2.3% |     69.3% |  1368 |%__exp
|   1.9% |     71.2% |  1117 |biogeophysics2@biogeophysics2m
|   1.9% |     73.0% |  1115 |stomata@canopyfluxesmod_
|   1.8% |     74.9% |  1084 |frictionvelocity@frictionveloc
|   1.7% |     76.6% |  1019 |soilwater@soilhydrologymod_
|   1.7% |     78.2% |   990 |tridiagonal@tridiagonalmod_
|   1.6% |     79.9% |   970 |update_hbuf_field@histfilemod_
|   1.5% |     81.3% |   866 |soilthermprop@soiltemperaturem
|   1.4% |     82.7% |   822 |combinesnowlayers@snowhydrolog
|   1.2% |     83.9% |   723 |lseek
|   1.2% |     85.1% |   705 |__read
|   1.1% |     86.3% |   672 |__write
|   1.0% |     87.3% |   600 |update_hbuf@histfilemod_
|   0.6% |     87.9% |   383 |surfacerunoff@soilhydrologymod
|   0.6% |     88.6% |   372 |vec2xy@mapxy_
|   0.6% |     89.2% |   357 |%__atan
|   0.5% |     89.7% |   307 |driver_
|   0.5% |     90.2% |   293 |__atan
|   0.5% |     90.6% |   283 |areaovr@areamod_
|   0.5% |     91.1% |   276 |hydrology2@hydrology2mod_
|   0.5% |     91.6% |   269 |drainage@soilhydrologymod_
|   0.4% |     92.0% |   251 |atm_readdata@atmdrvmod_
|   0.4% |     92.4% |   248 |snowwater@snowhydrologymod_
|   0.4% |     92.8% |   224 |dividesnowlayers@snowhydrology
|   0.4% |     93.2% |   222 |makel2a@lnd2atmmod_
|   0.4% |     93.5% |   211 |__sin
|   0.3% |     93.8% |   200 |%__alog10
|   0.3% |     94.2% |   195 |biogeophysics1@biogeophysics1m
|   0.3% |     94.5% |   190 |baregroundfluxes@baregroundflu
|   0.3% |     94.7% |   149 |_ld_read
|   0.2% |     95.0% |   131 |initdecomp@decompmod_
|   0.2% |     95.2% |   130 |_stride_dv
|  Truncated because cumulative % of Samp exceeds 95.
|=====================================
```

Table 1: Profile for a 90 day T85 run using 1 MSP with CSDs

been extensively optimized.

The `updateinput`, `areaovr_point`, `mkmxovr`, and `areaave` routines provide data exchange and interpolation functionality for the River Transport Model (RTM), while `rtmriverflux` runs the RTM. The data exchange and interpolation functions, which have not been vectorized, are used repeatedly when running CLM3 in offline mode; however, they are called only during initialization when run in fully coupled CCSM mode, so effort has not been spent optimizing these routines. The `pft2col` routine accumulates data from the plant functional type (PFT) sub-grid level to the column level.

```
   Samp% | Cum.Samp% |  Samp |SSP=0
                            |Function
                            |Caller
|---------------------------------------
| 100.0% |    100.0% | 82521 |Total
|---------------------------------------
|  13.8% |     13.8% | 11372 |gettimeofday
|   7.8% |     21.6% |  6459 |areaovr_point@areamod_
|   7.1% |     28.7% |  5822 |%__rtor_vv
|   6.4% |     35.1% |  5278 |mkmxovr@areamod_
|   5.3% |     40.4% |  4393 |updateinput@rtmmod_
|   5.3% |     45.7% |  4379 |canopyfluxes@canopyfluxesmod_
|   5.3% |     50.9% |  4334 |MPI_CRAY_gatherv
|   4.9% |     55.8% |  4023 |MPI_CRAY_bcast
|   3.9% |     59.7% |  3181 |rtmriverflux@rtmmod_
|   3.8% |     63.5% |  3176 |pft2col@pft2colmod_
```

Table 2: Top 10 routines for a 90 day T85 run using 2 MSPs with CSDs

```
   Samp% | Cum.Samp% |   Samp |SSP=0
                            |Function
                            |Caller
|---------------------------------------
| 100.0% |    100.0% | 829295 |Total
|---------------------------------------
|  21.9% |     21.9% | 181986 |gettimeofday
|  15.2% |     37.1% | 125741 |MPI_CRAY_bcast
|  13.5% |     50.6% | 112289 |MPI_CRAY_gatherv
|  12.3% |     63.0% | 102158 |areaovr_point@areamod_
|  10.5% |     73.4% |  86886 |mkmxovr@areamod_
|   7.3% |     80.8% |  60740 |rtmriverflux@rtmmod_
|   4.0% |     84.8% |  33180 |areaave@areamod_
|   2.3% |     87.0% |  18867 |atm_readdata@atmdrvmod_
|   1.1% |     88.2% |   9518 |areaovr@areamod_
|   1.1% |     89.3% |   9149 |MPI_Allgatherv
```

Table 3: Top 10 routines for a 90 day T85 run using 32 MSPs with CSDs

Table 2 shows the top 10 routines for the same 90 day run using 2 MSPs with CSDs enabled. MPI is used for communication, and already `MPI_Gatherv` and `MPI_Bcast` are showing up at the 7th and 8th positions in the report. A small portion of this time is attributable to synchronization (*i.e.*, waiting for processes to catch up to the point in the program where the communication occurs); however, it is likely that MPI performance could be improved. CLM3 has very good load balancing, so synchronization should take very little time. `MPI_Gatherv` is used each time step to generate a temperature diagnostic, and it is used to accumulate data for monthly history output. `MPI_Bcast` is used to distribute atmospheric forcing data read from disk by the master process to all processes on a monthly basis.

As can be seen from Table 3, `MPI_Bcast` and `MPI_Gatherv` are the second and third most expensive routines when the model is run on 32 MSPs. Only `gettimeofday` represents more time samples. `MPI_Allgatherv`, used after each run of RTM, appears at number 10. With 32 MPI processes, MPI routines and interpolation for RTM dominate run time. While some additional time may be lost to a slightly larger load imbalance than when using two processes, at 32 MPI processes communication time appears to exceed calculation time.

4.2 Earth Simulator

Similar timing experiments were carried out on the Earth Simulator (ES). Since the ES machine does not use multi-streaming processors, only OpenMP configurations were tested. One set of tests used four OpenMP threads, while the other used eight OpenMP threads. Like the timing tests on the Cray X1, the clumps-per-process parameter was varied from 1 to 32. The results from these timing tests

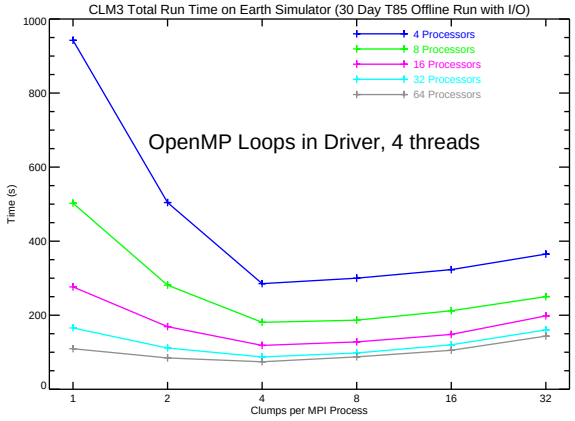

Figure 10: Curves show total run times for CLM3 on the Earth Simulator using OpenMP directives around high level loops in the `driver` routine for 4 to 64 processors. Four threads were used per MPI process, so the number of processors shown is equal to the product of the number of MPI processes and the number of threads.

are shown in Figures 10 and 11. The number of processors shown is equal to the product of the number of MPI processes and the number of threads.

As expected, choosing four clumps per process provides the best performance for the four threads case while eight clumps per process provides the best performance for the eight threads case. On the Earth Simulator, as on the Cray X1, the model scales only to about 64 processors. Using eight OpenMP threads yields slightly better performance than using four threads.

4.3 Performance Comparison

Figure 12 shows a comparison of the best CLM3 total run times for the IBM Power4 using four OpenMP

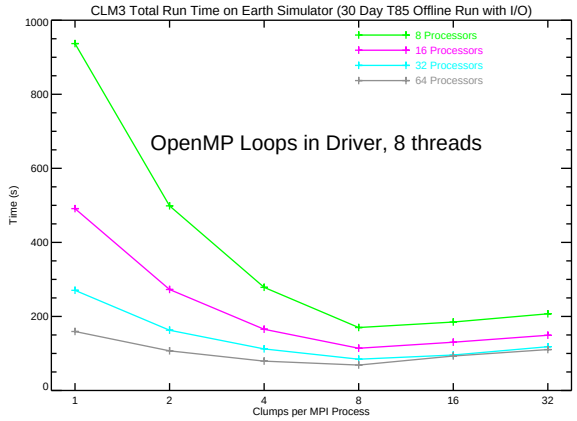

Figure 11: Curves show total run times for CLM3 on the Earth Simulator using OpenMP directives around high level loops in the `driver` routine for 8 to 64 processors. Eight threads were used per MPI process, so the number of processors shown is equal to the product of the number of MPI processes and the number of threads.

threads; the Earth Simulator using both four and eight OpenMP threads; and the Cray X1 using CSDs, OpenMP, and both CSDs and OpenMP simultaneously. The curve for the IBM Power4 shows very good scaling to 32 processors (within a single node) and reasonable scaling to 64 processors. The IBM matches the best Earth Simulator performance and beats the Cray X1 when using only CSDs at 64 processors.

The shapes of the Earth Simulator and Cray X1 curves are similar, so the model scales in a similar fashion on both machines. However, the Cray X1 performance numbers are usually better than those from the Earth Simulator. When using only CSDs around the `driver` loops, the Cray X1 beats the Earth Simulator out to 32 processors. At 64 processors, the Earth Simulator does slightly better. On the other hand, when OpenMP is used on the Cray X1 (with or without CSDs), it always beats the Earth Simulator on a per processor basis. Moreover, at low processor counts (16 or fewer), the model performs significantly better on the Cray X1 than on the other two platforms.

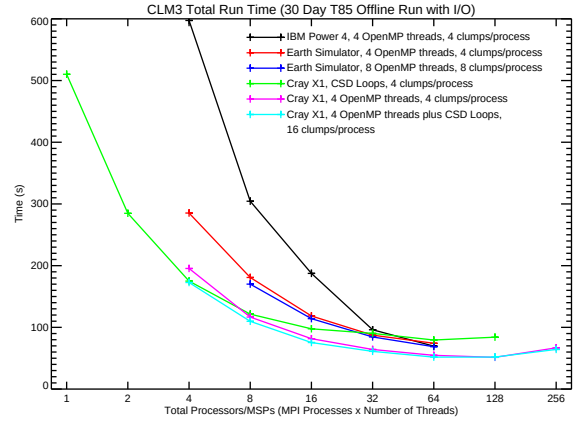

Figure 12: Curves show performance of the Community Land Model (CLM3) run in offline mode on the IBM Power4, the Earth Simulator, and the Cray X1.

5 Conclusions

Vectorization should be considered during all model development to ensure performance portability across computer platforms. Internal data structures can often be designed to meet the goals of researchers and still provide good vector and scalar performance. Writing loops so that they will vectorize is worth the up-front effort since it assures good performance on a wide variety of systems. Even today's personal computers have small vector units which can be harnessed for high performance computational science.

The vector performance of the Community Land Model is reasonable even when run in offline mode. When coupled to the Community Atmosphere Model and the Community Climate System Model, better performance is expected since atmospheric data are passed directly to the land model instead of being read from disk and broadcast to all MPI processes. At 64 processors, the performance of the model is similar on the IBM Power4, the Earth Simulator, and the Cray X1. However, the Cray X1 offers the best performance of all three platforms tested from 4 to 64 processors when OpenMP is used. Moreover, at low processor counts (16 or fewer), the model performs significantly better on the Cray X1 than on the other platforms. When run with CCSM, fewer processors will need to be allocated to the land model.

Vectorizing the Community Land Model required significant changes to the internal data structures and every science subroutine in the model. The process of rewriting the code took about six months to complete, and the planning and related development activities impacted model developers for an entire year. However, the resulting model code performs better on both vector and scalar architectures. These performance improvements will result in more and better research into land surface processes and feedbacks as access to vector platforms widens.

Acknowledgments

This research used resources of the Center for Computational Sciences at Oak Ridge National Laboratory which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE–AC05–00OR22725. Additional resources were provided by the National Center for Atmospheric Research (NCAR) which is sponsored by the National Science Foundation (NSF), the Central Research Institute of Electric Power Industry (CRIEPI) in Japan, Cray Inc., and NEC Corporation.

References

Bonan, G. B., Oleson, K. W., Vertenstein, M., Levis, S., Zeng, X., Dai, Y., Dickinson, R. E., and Yang, Z.-L. (2002). The land surface climatology of the Community Land Model coupled to the NCAR Community Climate Model. *J. Climate*, 15:3123–3149.

Dai, Y., Zeng, X., Dickinson, R. E., Baker, I., Bonan, G. B., Bosilovich, M. G., Denning, A. S., Dirmeyer, P. A., Houser, P. R., Niu, G., Oleson, K. W., Schlosser, C. A., and Yang, Z.-L. (2003). The Common Land Model. *Bulletin of the American Meteorological Society*, 84(8):1013–1023.

Kiehl, J. T. and Gent, P. R. (2004). The Community Climate System Model, version two. *J. Climate*. accepted for publication.

Oleson, K., Dai, Y., Bonan, G., Bosilovich, M., Dickinson, R., Dirmeyer, P., Hoffman, F., Houser, P., Levis, S., Niu, G.-Y., Thornton, P., Vertenstein, M., Yang, Z.-L., and Zeng, X. (2004). Technical Description of the Community Land Model (CLM). Technical Note NCAR/TN-461+STR, National Center for Atmospheric Research.

White, J. B. (2003). An optimization experiment with the Community Land Model on the Cray X1. In *Proceedings of the 2003 Cray Users Group (CUG) Meeting*, Columbus, Ohio.