A New Flexible Coupler for Earth System Modeling developed for CCSM4 and CESM1.

Anthony P. Craig, tcraig@ucar.edu
Climate and Global Dynamics Division, National Center for Atmospheric Research, Boulder, Colorado, USA

Mariana Vertenstein
Climate and Global Dynamics Division, National Center for Atmospheric Research, Boulder, Colorado, USA

Robert Jacob
Mathematics and Computer Science Division, Argonne National Laboratory. Argonne, Illinois, USA

Abstract:

The Community Climate System Model (CCSM) has been developed over the last decade, and it is used to understand past, present, and future climates. The latest versions of the model, CCSM4 and CESM1, contain totally new coupling capabilities in the CPL7 coupler that permit additional flexibility and extensibility to address the challenges involved in earth system modeling. An integral part of CPL7 is the implementation of a coupling architecture that takes a completely new approach with respect to the high-level design of the system. CCSM4 now contains a top-level driver that calls model component init, run, and finalize methods through specified interfaces. The top-level driver allows the model components to be placed on relatively arbitrary hardware processor layouts and run sequentially, concurrently, or mixed. The new coupling architecture continues to provide "plug and play" capability of data and active components, but improvements have been made to the memory and performance scaling of the coupler that support much higher resolution configurations. CCSM4 now scales better to higher processor counts, and the ability to handle global resolutions up to one-tenth degree is demonstrated.

Introduction:

Background

Climate modeling consists of complex multi-physics applications and is one of today's high performance computing challenges. The Community Climate System Model (CCSM) is a coupled, state-of-the-art global climate model

consisting of four fundamental physical components: an atmosphere model, a surface land model, an ocean model, and a sea ice model. The Community Earth System Model (CESM) adds a new land-ice component, land and ocean biogeochemistry functionality, an atmospheric chemistry model, and a capability for the atmospheric component to span a larger range of altitudes. The CCSM and CESM components can be “data” components or “active” dynamical components. Each dynamical component typically contains both fluid-dynamics solvers and detailed parameterizations to compute the internal and external forcing terms that arise from such diverse phenomena as the passage of radiation through the atmosphere, the release of latent heat by phase changes of water, and the effects of friction and unresolved turbulent scales.

As the CCSM/CESM components evolve in time, they periodically exchange boundary data via the use of coupling software. This coupling software supports communication of data between components, interpolation of data between different component grids, merging of fields from several “source” components to a “destination” component, and the production of various diagnostics among other things. CCSM3 (Collins et al., 2006a) used the CPL6 coupling software (Craig et al., 2005). In what follows, we present an overview of the high-level design and performance of the new CPL7 coupling software that forms an integral part of CCSM4 and CESM1. In the following discussion, the name “CCSM4” or “CPL7” will be used to describe capabilities in both CCSM4 and CESM1.

Couplers are a key component of climate models. In particular, coupling of models normally involves at least three different aspects: the coupling architecture, the communication infrastructure, and the coupling methods. The coupling architecture is generally associated with the overall control of the system, including the temporal sequencing of the model components. The communication infrastructure supports data transfer between components and depending on the overall design, can be implemented from a high level driver through subroutine calls or from within the model components directly. Finally, coupling methods provide capabilities such as mapping between different component grids, merging from several source components to a destination component, computation of physical quantities such as fluxes, or the computation of diagnostics. Every coupled climate model application needs to address these aspects, and they are generally implemented using a combination of community tools and custom-built software.

Coupling architectures can be distinguished by four basic features: whether data is sent through a “hub” or communicated directly between components, whether communication of data is handled via a top-level driver or from calls directly in the components, how the coupled model components are configured on hardware processors and whether a system is run as a single executable or as multiple executables. CCSM has always had, and continues to have, a hub design. To couple models developed separately into a single application, CCSM

developed, over the past fifteen years, coupling architectures that permitted coupling with minimum modification to component models.

It is important to distinguish between temporal concurrency and processor concurrency. Component models can be run on mutually exclusive processors, but if they are sequenced temporally in the implementation, a concurrent processor layout would actually run sequentially. Two models will run concurrently across processors only if they are run on unique sets of processors and if there are no temporal dependencies between the components that will limit concurrency.

Single executable designs do not automatically imply that models are running on the same processors sequentially. In general, components can be laid out in relatively arbitrary processor groups with single executable systems. On the other hand, multiple executable systems normally imply that the model components are running concurrently on unique processors sets since hardware systems generally disallow multiple executables from sharing the same processors and interleaving.

A particular coupled climate model implementation is fundamentally driven by architectural choices related to component sequencing, coupling frequency and lags, concurrency, scientific requirements, job launching, component integration, and interoperability. For example, the Parallel Climate Model (PCM) (Washington et al., 2000) incorporated a high level driver that supported sequential execution of components in a single executable. In this case, the coupler was a driver that called components via subroutine interfaces and the sequencing and coupling operations were custom built around particular scientific needs. On the other hand, the Oasis coupler (Valcke et al., 2006a) is a more generic coupler component designed to support coupling of multiple executable components that communicate via calls to a shared coupling interface called PRISM (Valcke, 2006b). Oasis supports both coupling through a hub and direct coupling between components. The design of these two couplers is fundamentally different, yet they both meet the scientific needs of their community. Other examples of climate model couplers are FMS (http://www.gfdl.noaa.gov/fms), FOAM (Jacob et al., 2001), PALM (http://www.cerfacs.fr/globc/PALM_WEB) and FLUME. Other examples of coupling libraries are MCT (Larson et al., 2005), ESMF (Hill et al., 2004), and the Distributed Data Broker (Drummond et al., 2001).

Motivation

Prior to CCSM4, CCSM operated as a multiple executable system where all models ran concurrently over disjoint sets of hardware processors and where each component model was a separate binary program. The components started independently and communicated to the coupler at regular intervals via send and

receive methods called directly from within each component. The coupler acted as a central hub coordinating data exchange, implicitly managing lags and sequencing, and executing coupler operations such as mapping (interpolation) and merging. In practice the coupler sequencing was difficult to understand because of the embedded communication calls in components. In addition, although special efforts were made in CCSM3 to maximize the amount of concurrency, the multiple executable concurrent design limited model throughput in some configurations. In addition, the CCSM3 design made porting and debugging challenging on occasion largely because of the multiple executable job launch. Finally, the prior CCSM implementations were not consistent with an ability to couple using the Earth System Model Framework (ESMF).

Recent advances in physics algorithms in CCSM4 components, including an updated and improved atmospheric boundary layer scheme and new radiation and surface albedo algorithms, require that components be coupled more frequently than in the past for stability reasons. In addition, as resolutions increase, component timesteps decrease and coupling frequencies tend to increase. With the recent need for higher frequency coupling, limitations in the CCSM3/CPL6 capability to overlap work in concurrent execution became increasingly critical.

In the past, CAM (Collins et al., 2006b) and CLM (Bonan et al., 2002; Oleson et al., 2010) “stand-alone” coupled systems have been publicly available from NCAR as sequentially coupled single executable climate models with all components running on the same grid. They are extensions of the prior CCM and LSM release models that have existed at NCAR since the 1980s. The CAM “stand-alone” model was an atmosphere, land-surface, data-ocean (using prescribed sea surface temperatures), and thermodynamic-only sea ice (using prescribed ice coverage) coupled system. The CLM “stand-alone” model consisted of data-atmosphere and land-surface coupled components. Over the past decade, these “stand-alone” models were released and supported in parallel to CCSM with overlapping resources from NCAR. For several reasons including code unification, reduced maintenance, and expanded features such as an ability to run on a single processor, CCSM4 developers wanted to duplicate several features associated with the CAM and CLM “stand-alone” systems.

With CPL7 in CCSM4, a completely new approach has been taken with respect to the high-level architecture and design of the system. The system is now controlled by a top-level driver that runs on all processors, and components are run via calls to standard component subroutine interfaces that run on all or arbitrary subsets of hardware processors. This overall effort was undertaken for several reasons including ability to better support new science, desire to support fully sequential and single processor integration, migration to a single executable system for ease in porting and use, increased flexibility of component layouts on hardware processors to improve performance over a wider range of problems, ability to support coupling via Earth System Model Framework (ESMF)

James White 12/10/10 2:15 PM
Comment: The following seems like an overview. The “Motivation” subsection seems to have ended. Maybe add an “Overview” subsection here? Though that might be awkward since a different “Overview” subsection is next.

superstructure, and targeting much higher resolutions and higher processor counts with improved performance and memory scalability.

CCSM4 has greatly expanded the flexibility of component layouts by moving to a single executable system that continues to support concurrent processor layouts but also supports components running sequentially or in mixed sequential/concurrent mode. This new wider layout choice gives users more flexibility in optimizing load balance and efficiency for any simulation configuration. CCSM4 was also rewritten to significantly reduce memory use and improve memory scaling. Previous CCSM versions were run mostly at global resolutions of one to five degrees, and memory usage was not a constraint. CCSM4 now supports the ability to run global one-tenth degree resolutions using tens of thousands of processors on massively parallel machines with relatively limited memory available per processor, especially given the problem size.

In what follows, we describe the CPL7 design in CCSM4 in more detail, with a focus on sequencing, interfaces, infrastructure, memory scaling, and processor layout. Some performance results will also be presented.

Design:

Overview

CCSM4/CPL7 is built as a single executable with a single high-level driver that runs on all processors, calls components, handles sequencing, manages component layout on hardware processors, and communicates data between components. The driver calls all model components via common and standard interfaces. The driver also directly calls coupler methods for mapping (interpolation), rearranging, merging, flux calculation, and diagnostics. In CCSM4, the model components and the coupler methods can run on subsets of all the processors. In effect, the CPL6 sequencing and hub attributes have been migrated to the driver level while the CPL6 coupler operations such as mapping and merging are done as if a separate coupler component existed in CCSM4.

CCSM4 supports both data and active components models. In general, an active component needs data from and provides data to the coupler. Data components, on the other hand, normally only read forcing data from external data files and then provide this data to the coupler. In CCSM4, like CCSM3, the atmosphere, land, and sea ice models are always tightly coupled to better resolve the diurnal cycle. It is important to point out that atmosphere/ocean fluxes are computed in the coupler (and not in the ocean component) at the same frequency that the atmosphere, land and sea ice models communicate. Similarly, the diurnal cycle of ocean surface albedo is also computed in the coupler for use by the atmosphere model. This coupling is typically hourly, although at higher resolutions it can be more frequent. Since the ocean model is not responsible for

computing atmosphere/ocean fluxes, it is typically coupled once or a few times per day. The looser ocean coupling frequency means the ocean state and response is lagged in the system. This also permits the ocean model to run efficiently on a mutually exclusive set of processors from the atmosphere, land, sea-ice and coupler components. There is an option in CCSM4 to run the ocean tightly coupled without any lags, but this is normally used only when running with data ocean components.

Depending on the resolution, hardware, run length and physics, a CCSM4 run can take several hours to several months of wall time to complete. Runs typically encompass model simulation times of decades or centuries, with the model typically running between one and fifty simulated years per wall clock day. CCSM4 has exact restart capability, and the model is typically run in individual one year or multi-year chunks. CCSM4 also has automatic resubmission and automatic long-term data archiving capability.

Sequencing and Concurrency

In CCSM4, the component processor layouts and MPI communicators are derived from simple user-specified namelist input. Presently, there are seven basic MPI processor groups in CCSM4. These are associated with the atmosphere, land, ocean, sea ice, land ice, coupler, and global groups, although others can be added easily. Each of the basic MPI groups can be associated with different processors, and a user can overlap MPI groups on hardware processors relatively arbitrarily. If any processor groups overlap each other in at least one processor, then those components will run sequentially. Each processor group is described at runtime using three scalar variables: the number of MPI tasks, the number of OpenMP threads per MPI task, and the global MPI task rank of the root process for that group. For example, a layout where the number of MPI tasks is 8, the number of threads per MPI task is 4, and the root process is MPI task 16 would create a processor group that consisted of 32 hardware processors, starting at global MPI task number 16, and it would contain 8 MPI tasks. The driver derives all MPI communicators at initialization and passes them to the component models for use. This input information is used both to set MPI groups and to set batch and job launching parameters.

As mentioned in the introduction, there are two aspects that determine whether component models run concurrently. The first is whether unique chunks of work are running on distinct processor groups. The second is the sequencing of this work in the driver. As much as possible, the CCSM4 driver sequencing has been implemented to maximize the potential amount of concurrency of work between different components. Ideally, in a single coupling step, the forcing for all models would be computed first. The models could then all run concurrently, and then the driver time would advance. However, scientific requirements such as the coordination of surface albedo and atmospheric radiation as well as boundary layer stability impose constraints on the coupling lags. Figure 1 shows the

maximum amount of concurrency currently allowed in the CCSM4 driver for a fully active system configuration. More concurrency is technically possible, but the scientific constraints impose a limitation on the coupling between the atmosphere model and the land and sea-ice models such that the atmosphere model must be run sequentially with both of those surface models. Again, figure 1 does not necessarily represent the optimum processor layout for performance for any configuration, but provides a practical limit to the amount of concurrency currently supported in the system. It is important to point out that, with CCSM4, results are within numerical round-off regardless of the layout of components on processors. Results are not bit-for-bit identical in this case because some physical components introduce round-off level changes when changing processor counts.

There is a loss of concurrency in CCSM4 relative to CCSM3. In CCSM4, the model run methods are called from the driver, and the coupling from the overall system perspective looks like “send to component, run component, receive from component”. In CCSM3, the coupling was done via direct calls from inside each component and the send and receive calls could be interleaved with work such that the model run method appeared to have two phases, one between the coupling send and receive, the other between the receive and send. Although that allowed greater concurrency in CCSM3, it was also designed for concurrent-only execution. In CCSM4, multiple run phases were not implemented to simplify the model and provide an opportunity for greater interoperability. This choice was made with the full understanding that there would be some potential loss of model concurrency in specific cases. However, it was felt that the additional flexibility of allowing models to run in a mixed sequential/concurrent system would overcome any performance degradation due to a potential loss of concurrency.

Component Interfaces

The CCSM4 component model interfaces are based on an initialize, run, and finalize approach with consistent arguments across different component models. Although the standard CCSM4 interface arguments currently consist of Fortran and Model Coupling Toolkit (MCT) (Larson et al., 2005) datatypes, an alternative version is available which is compatible with the ESMF superstructure. As part of initialization, the grid and component decomposition information is passed from the component back to the driver. The driver/coupler acquires all information about resolution, configurations, and processor layout at run-time from either input files or from communication with components. The physical coupling fields are passed through the interfaces in the initialize, run, and finalize phases. The run interface also contains a clock that specifies the current time and the run length for the model.

Initialization of the system in CCSM4 is relatively straight-forward. First, the varied MPI communicators are setup in the driver. Then the component

initialization methods are called on the appropriate processor groups, and grid and decomposition information is passed back to the driver. Once the driver has the grid and decomposition information from the components, various rearrangers and mappers are initialized that will move data between processors, decompositions, and grids as needed at the driver level. No distinction is made in the coupler implementation for sequential versus concurrent execution. In general, even for cases where two components have identical grids and processor layouts, often their decomposition is different for performance reasons. In cases where the grid, decomposition, and processor layout are identical between components, the mapping or rearranging operation will degenerate to a local data copy.

The sequencing of the driver run loop for the CCSM4 configuration with coupler, land, sea ice, atmosphere, and ocean components is shown in figure 2. This order of operations is hard-wired in the CCSM4 driver and is based on scientific constraints first and performance optimization second. The letter “C” in figure 2 indicates a coupling operation where data is communicated between the coupler component and another component. This step introduces a cross-component dependence, where one component will normally wait for the other component before communication proceeds. The letter “R” in figure 2 indicates independent work carried out by that component. Coupling to the atmosphere, land, and sea ice models occurs every step through the run loop, while coupling to the ocean normally occurs less frequently. Figure 2 shows that if components are run concurrently, the ocean will begin first, followed by the land and sea ice. The atmosphere model will not start until data from the sea ice and land models is communicated to the coupler, used to compute coupling data for the atmosphere model, and then sent to the atmosphere component. The ocean is always able to integrate concurrently with all other components in the design. Figure 2 provides the detailed implementation that limits the total amount of concurrency shown in Figure 1. The coupler is constantly computing between communication steps, and some of this work is overlapped with other components. Generally, the coupler work and communication costs are small compared to the active model run times.

Additional Software and Parallel IO

CPL6 was built using the Model Coupling Toolkit (MCT) infrastructure library, and CPL7 continues to rely heavily on MCT. MCT supports many critical coupling needs such as managing data parallelism and performing parallel remapping. The high-level data types in CPL7 consist largely of MCT data types, and MCT is used for all rearranging and mapping of data between processors, decompositions, and grids. Although the CPL7 coupling architecture is dependent on MCT infrastructure, the high level superstructure is designed to be compatible with ESMF (Collins et al., 2005), and CCSM4 components are ESMF compliant. In addition, the CCSM project continues to evaluate the use of ESMF infrastructure as a complement to the MCT capabilities. Mapping weights are still

generated offline using the SCRIP package (Jones, 1998). In order to minimize the memory footprint, mapping weights are read into CCSM4 using a method that reads and distributes the weights in reasonably small chunks.

CCSM4 is targeting much higher resolutions than any previous versions. Efforts have been made to reduce the memory footprint and improve memory scaling in all components with a goal of being able to run the fully coupled system at one-tenth degree global resolution on tens-of-thousands of processors, with each processor having as little as 512 MB of memory. This target limits the number of non-distributed arrays that can be allocated on any single processor to just a few at any time and has led to some significant component refactoring especially in initialization. In addition, the memory required to carry out traditional serial I/O at higher resolutions is unacceptable, and serial I/O performance at higher resolutions can be a significant bottleneck. In response to the I/O limitiation, the development of a parallel I/O (PIO) library (Dennis et al, 2011b), based on NetCDF, pNetCDF (www.mcs.anl.gov/parallel-netcdf), and MPI-IO has been underway within the CCSM community to address I/O performance and memory usage in the model. All model components are currently using the PIO software extensively, and use of PIO permits CCSM4 to run at resolutions that were not previously possible due to memory limitations.

Performance, Scaling, and Load Balance

To target scaling to tens of thousands of processors, developers for all components have worked at improving performance scaling via changes to algorithms, infrastructure, and decompositions. In particular, decompositions using shared memory blocks, space filling curves (Dennis et al., 2011a), and all three spatial dimensions have been implemented to varying degrees in order to increase parallelization and improve scalability in all components.

CCSM4 load balance refers to allocating and laying out model components on processors to optimize performance and minimize idle time. CCSM4 performance, load balance, and scalability are constrained by the problem size, complexity, and multiple model character of the system. Within the system, each component has its own scaling characteristics, and variation in scaling of a component occurs because of internal load imbalance, decomposition capabilities, or communication patterns. Component performance can also vary as the model integrates forward in time. This occurs because of seasonal variability of the cost of physics in models, changes in performance during an adjustment (spin-up) phase, and temporal variability in calling certain model operations like radiation, dynamics, or I/O. Within these constraints, a load balance configuration with acceptable idle-time and reasonably good throughput is nearly always possible to configure with CCSM4. CCSM4 has significantly increased the flexibility of the possible processor layouts, and this has resulted in better load balance configurations compared to previous CCSM versions.

In practice, load-balancing CCSM4 involves a number of considerations such as which components are run, their absolute resolution, and their relative resolution; cost, scaling and processor count capabilities for each component; and internal load imbalance within a component. It is often best to load balance the system with all significant run-time I/O turned off because this generally occurs infrequently (typically one timestep per month in CCSM4), is best treated as a separate cost, and can bias interpretation of the overall model load balance. The ability to use OpenMP and the performance of OpenMP threading in some or all of the system is dependent on the hardware/OS support as well as whether the system supports running all MPI and mixed MPI/OpenMP on overlapping processors for different components. Finally, the processor layout, whether sequential, concurrent, or some combination of the two, can be varied. Typically, a series of short test runs is done with the desired production configuration to establish a reasonable load balance setup for the production job. CCSM4 provides some post-run analysis of the performance and load balance of the system to assist users in improving the processor layouts.

Results:

In this section, performance scaling results will be presented for four different coupler kernels. Then some full model results will be presented to show how the layout of components on hardware processors impacts overall model performance. The four kernels that will be discussed are a field merge on the ocean grid, an atmosphere-ocean flux calculation, a rearrangement of ocean data between two different decompositions, and an atmosphere to ocean mapping. These kernels represent the most common CCSM coupler operations. Results will be presented from three different hardware platforms, bluefire, jaguarpf, and intrepid and at two different model resolutions, “f19_g16” and “f05_t12”.

James White 12/10/10 2:40 PM
Comment: I think if computer names as proper nouns, so I would use “Bluefire”, “JaguarPF” (or “Jaguarpf”), and “Intrepid”.

Bluefire is an IBM SP6 at the National Center for Atmospheric Research (NCAR) with 4096 4.7 GHz processors and 4 GB of memory per processor. Processors are grouped into 32-way nodes, and each processor supports up to 4 FLOPS (floating point operations) per clock cycle. The interconnect uses an InifiniBand switch, and each node has eight 4X DDR links. Bluefire supports simultaneous multi-threading (SMT) which allows 64 MPI tasks to be assigned to each node, and SMT mode was used for all timing results on bluefire. Jaguarpf is a Cray XT5 at the National Center for Computational Sciences (NCCS) at Oak Ridge National Laboratory (ORNL) that has 18,688 nodes of dual hex-core AMD processors running at 2.6 Ghz with 16 GB of memory per node. The total number of processors available is 224,256, and the interconnect uses a SeaStar 2+ router. Intrepid is an IBM Bluegene/P at Argonne National Laboratory (ANL) with 163,840 850 MHz PowerPC 450 processors. There are 4 processors per shared memory node, 1024 nodes per rack, and each node has 2 GB of memory.

The “f19_g16” is a moderate resolution grid and consists of a nominally two degree atmosphere/land grid with 144 x 96 horizontal gridpoints coupled to a nominally one degree ocean/ice grid with 320 x 384 gridpoints. The “f05_t12” is a high resolution configuration with 576 x 384 gridpoints on a nominally half degree atmosphere/land grid coupled to a nominally one-tenth degree global resolution ocean/ice configuration with 3600 x 2400 horizontal grid points. All cases were run with shared memory parallelism (OpenMP) turned off, and one MPI task was placed on each processor with the following exceptions. On bluefire, the IBM SP6, SMT mode was used as indicated above. On intrepid at the high resolution “f05_t12” configuration, one MPI task was assigned to each 4-way node due to memory limitations.

Tests were carried out on all machines in a production environment. As a result, some variability in timings was observed. Cases were rerun as needed to try to understand the variability better, and generally, best times are shown. All cases were carried out as 20 day runs using a dead model configuration without any history or restart I/O. All timers were isolated with MPI_BARRIER calls, and the time on the slowest task is presented. Over each 20 day integration, each kernel was called 960 times, and the time was summed over all the calls over and each MPI task. The times presented in the plots are seconds per simulated model day. All results are plotted in log/log format, and each plot uses the same horizontal and vertical axis scales for easier comparison. In addition, the same linear scaling reference line is included in each plot, and the horizontal axis is the number of MPI tasks used by the coupler in the testcase.

Figure 3 shows performance of the ocean field merge kernel from the CCSM coupler. This kernel is responsible for merging various fields on the ocean grid from different components. This operation is trivially parallel (there is no communication), and the kernel is primarily memory access limited. At the higher “f05_t12” resolution, the performance on all platforms is remarkably similar with near linear scaling to over 1000 processors on all machines. In constrast, the performance scaling of the moderate “f19_g16” resolution case is near linear up to about 100 processors on all machines. Above about 100 processors, performance tails off on bluefire and intrepid, while jaguarpf performance tails off at about 1000 processors.

Figure 4 shows the performance scaling of the atmosphere/ocean flux calculation. This calculation is also trivially parallel and is computed on the ocean grid. Compared to the ocean merge kernel, this computation is dominated by mathematical operations including add, multiply, divide, min/max, log, exponential, and square root. The number of FLOPS per memory load in this kernel is relatively high, and compared to figure 3, the scaling is near linear on all machines to at least 1000 processors even at the moderate resolution. The absolute performance of this kernal on bluefire and jaguarpf is nearly identical,

but intrepid runs about 10 times slower than the other two machines on this highly FLOP intensive metric.

James White 12/10/10 3:19 PM
Comment: Why 10x slower? Because you are only using one processor per node? This seems too much slower.

Figure 5 documents the scaling performance of a rearrange operation on the ocean grid. There are no FLOPS in this kernel, just memory access and an MCT rearrange of data between an ocean decomposition and an ice decomposition of the same grid. This rearrange is nearly an all-to-all communication of data between processors, and is performance limited by the communication cost. Scaling is generally sub linear for both resolutions across the full range of cases tested. The scaling tails off significantly above about 100 processors at the moderate “f19_g16” resolution with jaguarpf becoming slower (rolling over) above about 100 processors. At the higher “f05_t12” resolution, scaling is somewhat better, but it still flattens out at about 500 processors on bluefire, it flattens out at about 1000 processors on intrepid, and it rolls over at about 500 processors on jaguarpf.

Figure 6 shows the atmosphere to ocean mapping performance in CCSM. This kernel is a mixture of communication and multiply-add operations (Jacob et al., 2005) and is carried out using an MCT mapping method. In this case, the mapping weights are always distributed based on the ocean decomposition at initialization. When the kernel is called, the atmosphere data is rearranged to the ocean decomposition such that, for every processor, the atmosphere data required for mapping to the local ocean grid is available. In many cases, atmosphere data is rearranged to more than one ocean processor. When the rearrangement is complete, atmosphere data is locally interpolated to the ocean grid. Overall, the performance of this kernel is similar to the performance of the rearrange kernel in figure 5 indicating the importance of the rearrange step in the overall cost of this operation. Scaling starts to go flat or even turns over at about 100 processors for all machines at the moderate “f19_g16” resolution. The “f05_t12” scales well to about 1000 processors, except on jaguarpf where scaling flattens dramatically at about 500 processors.

Figures 7 provides some insight into the performance implication of different processor layouts for a specific CCSM coupled run. Figure 7 shows the overall performance of each component schematically for an “f19_g16” moderate resolution fully prognostic case running on a total of 128 processors. (a) is a fully concurrent layout of components on processors, (b) is a fully sequential layout where all components are run on all processors, (c) is fully sequential except the ocean model is run concurrently with the rest of the system, and (d) is a more general mixed sequential/concurrent layout. In all cases, the total number of processors used is 128 and the idle times, represented by gray boxes, was minimized as much as possible. The total run time for each of the four configurations is 33.5, 20.8, 21.8, and 19.1 seconds per model day for the (a) concurrent layout, (b) sequential layout, (c) sequential plus ocean concurrent layout, and (d) mixed sequential/concurrent layout respectively. Except for the (a) fully concurrent layout that has significant idle time because of the

James White 12/10/10 3:21 PM
Comment: Where do the times in the figure come from?

sequencing limitations of the atmosphere, land, and ice models in the driver, the total performance of the varied processor layouts is relatively close.

The concurrent layout in Figure 7 cannot be compared to the CCSM3 runtime because the concurrency is completely different. As was discussed in the design section, the CCSM3 coupling was implemented with two run phases that allowed greater overlap of work between components. In CCSM4, each component has just one run phase, and this significantly reduces the amount of potential concurrency and increases the runtime of a purely concurrent layout in CCSM4 compared to what would have been attained in CCSM3. However, the increased flexibility of CCSM4 processor layouts generally results in increased performance in nearly all configurations compared to the CCSM3 concurrent-only implementation.

James White 12/10/10 3:22 PM
Comment: Do you have measurements supporting this?

In general, Figure 7 demonstrates some of the potential component layouts on processors available to users of CCSM4. The optimal layout and processor counts for each component for any given case depends heavily on the scaling performance of each component and the resolution of each component. In general, the effectiveness of running models concurrently is a trade-off between the idle time created by concurrent execution versus the generally sublinear scaling of components as processor counts are increased.

Summary:

The latest version of the CPL7 coupler for CCSM4 and CESM1 has been described. Since the CCSM3 release, the implementation has been refactored significantly. CCSM4 now contains a top-level driver that calls model component init, run, and finalize methods through consistent interfaces. It also coordinates the sequencing and MPI groups of the components to provide greater flexibility in processor layouts. Many changes were also made to improve the memory and performance scaling of the coupler, and the CCSM4 coupling overhead has been reduced compared to CCSM3. Most of this effort has been implemented using the MCT coupling library, but ESMF compliant interfaces are also available for CCSM4 components.

CCSM4 now scales better to higher processor counts compared to CCSM3, and CCSM4 is able to handle much higher resolution configurations compared to CCSM3. Scaling curves in the CPL6 paper (Craig et al., 2005) were presented to 32 processors for a T42 (nominally three degree) atmosphere/land case coupled to a nominally one degree ocean/ice resolution. In this paper, results are presented for cases up to one-tenth degree global resolutions on processor counts up to 10,000. In particular, the CCSM4 has been run with active components at one-quarter degree atmosphere and land global resolutions coupled to a one-tenth degree ocean and ice global resolution (Dennis et al.,

2011c). This could not have been achieved with the CCSM3/CPL6 coupling implementation.

The performance of individual coupling kernels has been presented. As expected, trivially parallel operations scale well to high processor counts. In particular, the atmosphere/ocean flux kernel scaled almost linearly on all machines. The memory intensive merge kernel scaled well, and the communication intensive rearrange and mapping kernels generally scaled to moderate task counts. Several factors need to be considered when choosing the coupler processor count and layout for overall CCSM4 load balance and performance. The coupler cost is generally small compared to other components, and it is often run sequentially with other components, particularly the atmosphere model. As a result, the coupler is often placed on available processors based on load balance and optimization of the other components first. In general, the coupler will probably run faster on more processors within reason. However, based upon the scaling of the communication kernels on jaguarpf, there is the possibility that the coupler will slow down on some machines as processor counts are increased, so it is important to evaluate the effect of increasing processors on coupler performance before carrying out a long run.

Some general statements could also be made about the relative performance of the different hardware architectures used in this study. On a per processor basis, bluefire was consistently faster than either jaguarpf or intrepid for the coupler timings in general. The memory intensive merge kernel timings were similar for all machines especially in the regime where there are many gridcells per processor. But the computation intensive flux calculation was 10x slower on intrepid compared to either bluefire or jaguarpf. This should be somewhat expected based on the relative processor speed. The performance scaling of the communication intensive kernels on all machines tailed off at higher processor counts with jaguarpf performing worst at the higher processor counts. It is unclear whether the communication scaling behavior on jaguarpf is inherent to the hardware, associated with the local system setup, or related to the diverse load on the machine at any given time.

Finally, the flexibility to vary the processor layout in CCSM4 has been demonstrated. The CCSM4 model's ability to support sequential, concurrent, or mixed layouts provides much greater flexibility with respect to overall load balance and performance optimization compared to CCSM3.

Acknowledgements:

A. Craig and M. Vertenstein have been supported by U.S. Department of Energy grants DE-FC02-97ER62402 and XX. M. Vertenstein has also been supported by the National Science Foundation grant AGS-0856145. R. Jacob been

supported by the Office of Science (BER), U.S. Department of Energy under contract DE-AC02-06CH11357. The authors which to thank the National Center for Atmospheric Research, the Oak Ridge National Laboratory, and the Argonne National Laboratory for access to and assistance with computing resources.

References:

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

Collins, N., G. Theurich, C. DeLuca, M. Suarez, A. Trayanov, V. Balaji, P. Li, W. Yang, C. Hill, and A. da Silva (2005). Design and Implementation of Components in the Earth System Modeling Framework. International Journal of High Performance Computing Applications. 19(3):341-350.

Collins, W.D., C.M. Bitz, M.L. Blackmon, G.B. Bonan, C.S. Bretherton, J.A. Carton, P. Chang, S.C. Doney, J.J. Hack, T.B. Henderson, J.T. Kiehl, W.M. Large, D.S. McKenna, B.D. Santer and R.D. Smith (2006a). The Community Climate System Model Version 3 (CCSM3). Journal of Climate. 19(11):2122-2143.

Collins, W.D., P.J. Rasch, B.A. Boville, J.J. Hack, J.R. McCaa, D.L. Williamson, B.P. Briegleb, C.M. Bitz, S-J. Lin and M. Zhang (2006b). The Formulation and Atmospheric Simulation of the Community Atmosphere Model Version 3 (CAM3). Journal of Climate. 19(11):2144-2161.

Craig, A.P, R.Jacob, B. Kauffman, T. Bettge, J. Larson, E. Ong, C. Ding, and Y. He (2005). CPL6: The New Extensible High Performance Parallel Coupler for the Community Climate System Model. International Journal of High Performance Computing Applications. 19(3):309-328.

Dennis, J.M., D Bailey, E. Hunke (2011a). A Probabilistic Weighted Space-Filling Curve Partitioning for Spatial Load Imbalance. International Journal of High Performance Computing Applications. Submitted.

Dennis, J.M., J. Edwards, R. Loy, R. Jacob, A. Mirin, A.P. Craig, M. Vertenstein (2011b). An Application Level Parallel I/O Library for Earth System Models. International Journal of High Performance Computing Applications. Submitted.

Dennis, J.M., M. Vertenstein, P. Worley, A. Mirin, A.P. Craig (2011c). An Ultra-High-Resolution Capability in the Community Climate System Model. International Journal of High Performance Computing Applications. Submitted.

Drummond, L. A. , J. Demmel, C. R. Mechoso, H. Robinson, K. Sklower and J. A. Spahr (2001). A Data Broker for Distributed Computing Environments. Lecture Notes in Computer Science. 2073:31-40.

Hill, C., C. DeLuca, V. Balaji, M. Suarez and A. da Silva (2004). Architecture of the Earth System Modeling Framework. Computing in Science and Engineering. 6(1).

Jacob, R., C. Schafer, I. Foster, M. Tobis, and J. Anderson (2001). Computational Design and Performance of the Fast Ocean Atmosphere Model, Version One. Proc. 2001 International Conference on Computational Science, eds. V. N. Alexandrov, J. J. Dongarra, C. J. K. Tan, Springer-Verlag, pp. 175-184.

Jacob, R., J. Larson, and E. Ong (2005). MxN Communication and Parallel Interpolation in CCSM3 Using the Model Coupling Toolkit. International Journal of High Performance Computing Applications, 19(3):293-307,

Jones, P.W. (1998). A User's Guide for SCRIP: A Spherical Coordinate Remapping and Interpolation Package. Los Alamos National Laboratory, Los Alamos, NM.

Larson, J., R. Jacob and E. Ong (2005). The Model Coupling Toolkit: A new Fortran90 Toolkit for Building Multiphysics Parallel Coupled Models. International Journal of High Performance Computing Applications. 19(3):277-292.

Oleson, K., D.M. Lawrence, G. B. Bonan, M. Flanner, E. Kluzek, P. Lawrence, S. Levis, S. Swenson, P. Thornton, A. Dai, M. Decker, R. Dickinson, J. Feddema, C. Heald, F. Hoffman, J. Lamarque, N. Mahowald, G. Niu, T. Qian, J. Randerson, S. Running, K. Sakaguchi, A. Slater, R. Stockli, A. Wang, Z. Yang, and X. Zeng (2010). Technical description of version 4.0 of the Community Land Model (CLM). NCAR Technical Note, NCAR/TN-478+STR.

Valcke, S., E. Guilyardi and C. Larsson (2006a). PRISM and ENES: A European approach to Earth system modelling. Concurrency Computat.: Pract. Exper. 18(2):231-245.

Valcke, S. (2006b). OASIS3 User Guide (prism_2-5). CERFACS Technical Report TR/CMGC/06/73, PRISM Report No 3, Toulouse, France. 60 pp.

Washington, W.M., J.W. Weatherly, G.A. Meehl, A.J. Semtner Jr., T.W. Bettge, A.P. Craig, W.G. Strand Jr., J.M. Arblaster, V.B. Wayland, R. James and Y. Zhang (2000). Parallel climate model (PCM) control and transient simulations. Climate Dynamics. 16(10/11):755-774.

Figures:

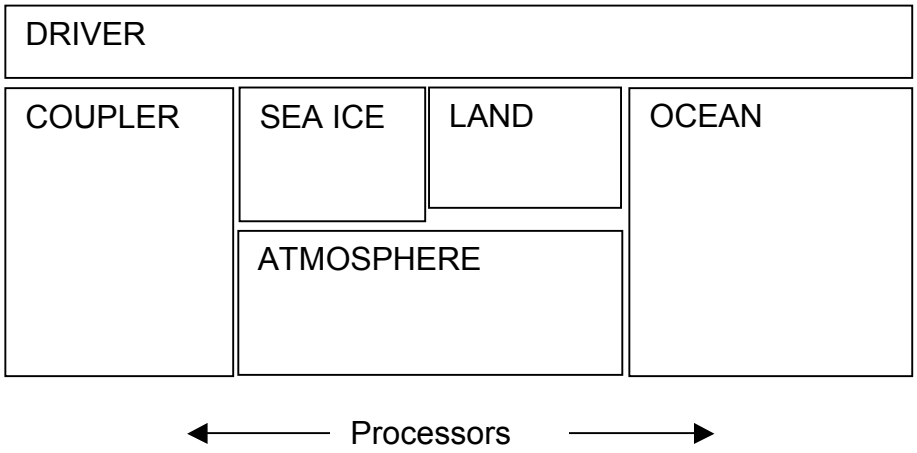

Figure 1: CCSM4 Concurrency capability based on scientific constraints.

Driver Operation	CPL	LND	ICE	ATM	OCN
Ocean Prep	R				
CPL to OCEAN communication	C				C
Land Prep	R				
CPL to LAND communication	C	C			
Sea Ice Prep	R				
CPL to ICE communication	C		C		
Ocean Run					R
Ice Run			R		
Land Run		R			
Atm/Ocean Surface Flux Computation	R				
ICE to CPL communication	C		C		
Ice Post	R				
LAND to CPL communication	C	C			
Land Post	R				
Atm Prep	R				
CPL to ATM communication	C			C	
Atm Run				R	
ATM to CPL communication	C			C	
Atm Post	R				
OCEAN to CPL communication	C				C
Ocean Post	R				

Figure 2: Driver Loop Sequencing showing the order of operations at the driver level and concurrency of the coupler, land, sea ice, atmosphere, and ocean components. An “R” indicates work is being done by that component and a “C” indicates the coupler and another component are communicating.

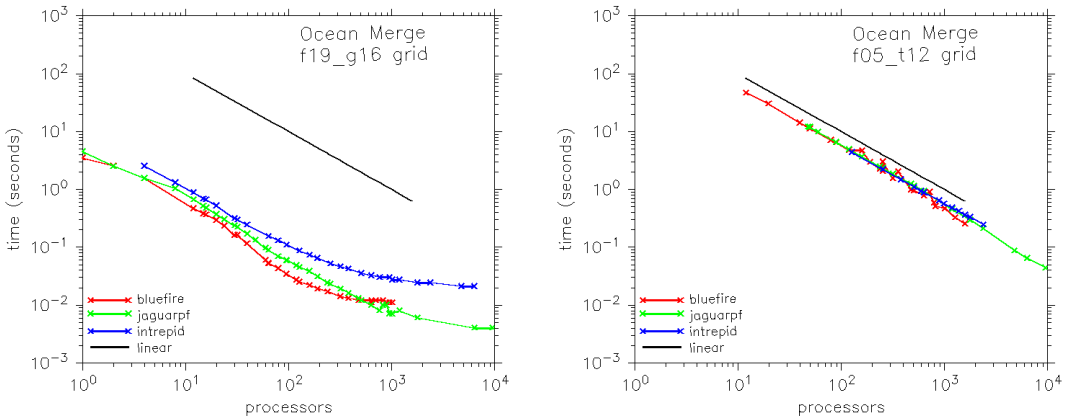

Figure 3: Scaling performance of the ocean merge kernel in the CCSM4 coupler at at two resolutions (“f19_g16” and “f05_t12”) and on three hardware platforms (bluefire, jaguarpf, and intrepid).

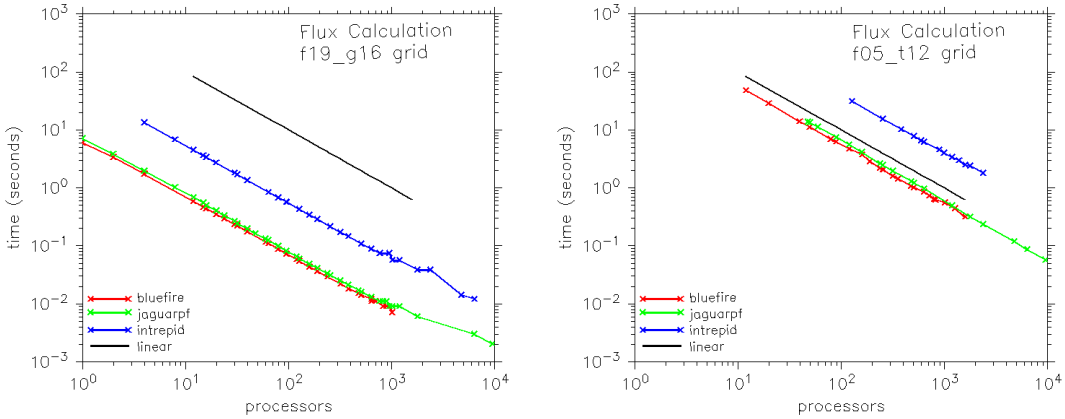

Figure 4: Scaling performance of the atmosphere/ocean flux kernel in the CCSM4 coupler at two resolutions (“f19_g16” and “f05_t12”) and on three hardware platforms (bluefire, jaguarpf, and intrepid).

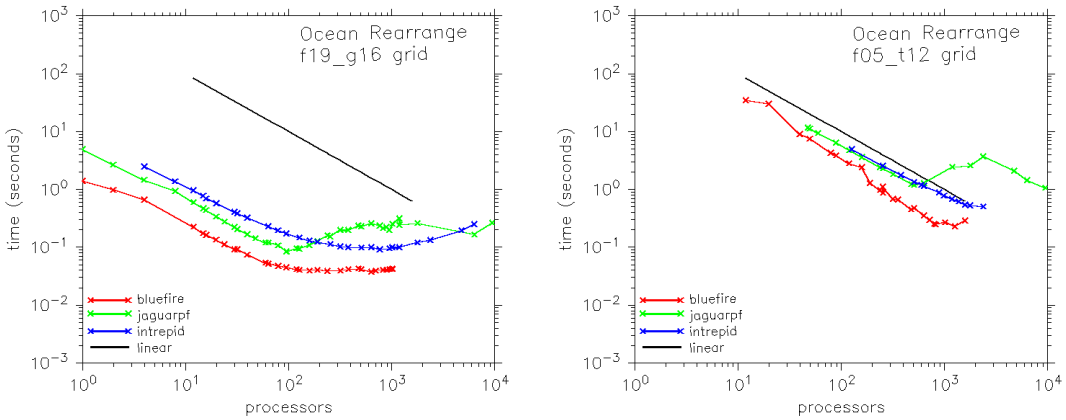

Figure 5: Scaling performance of the ocean to ice rearrange kernel in the CCSM4 coupler at two resolutions (“f19_g16” and “f05_t12”) and on three hardware platforms (bluefire, jaguarpf, and intrepid).

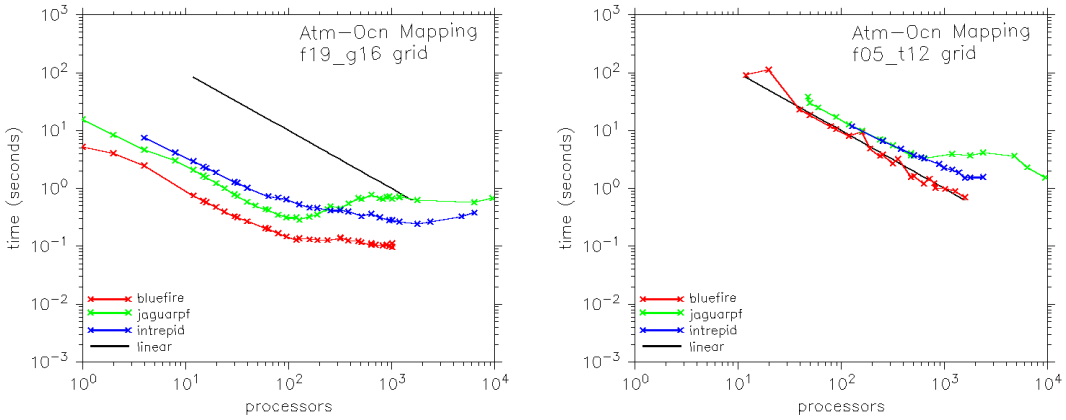

Figure 6: Scaling performance of the atmosphere to ocean mapping kernel in the CCSM4 coupler at two resolutions (“f19_g16” and “f05_t12”) and on three hardware platforms (bluefire, jaguarpf, and intrepid).

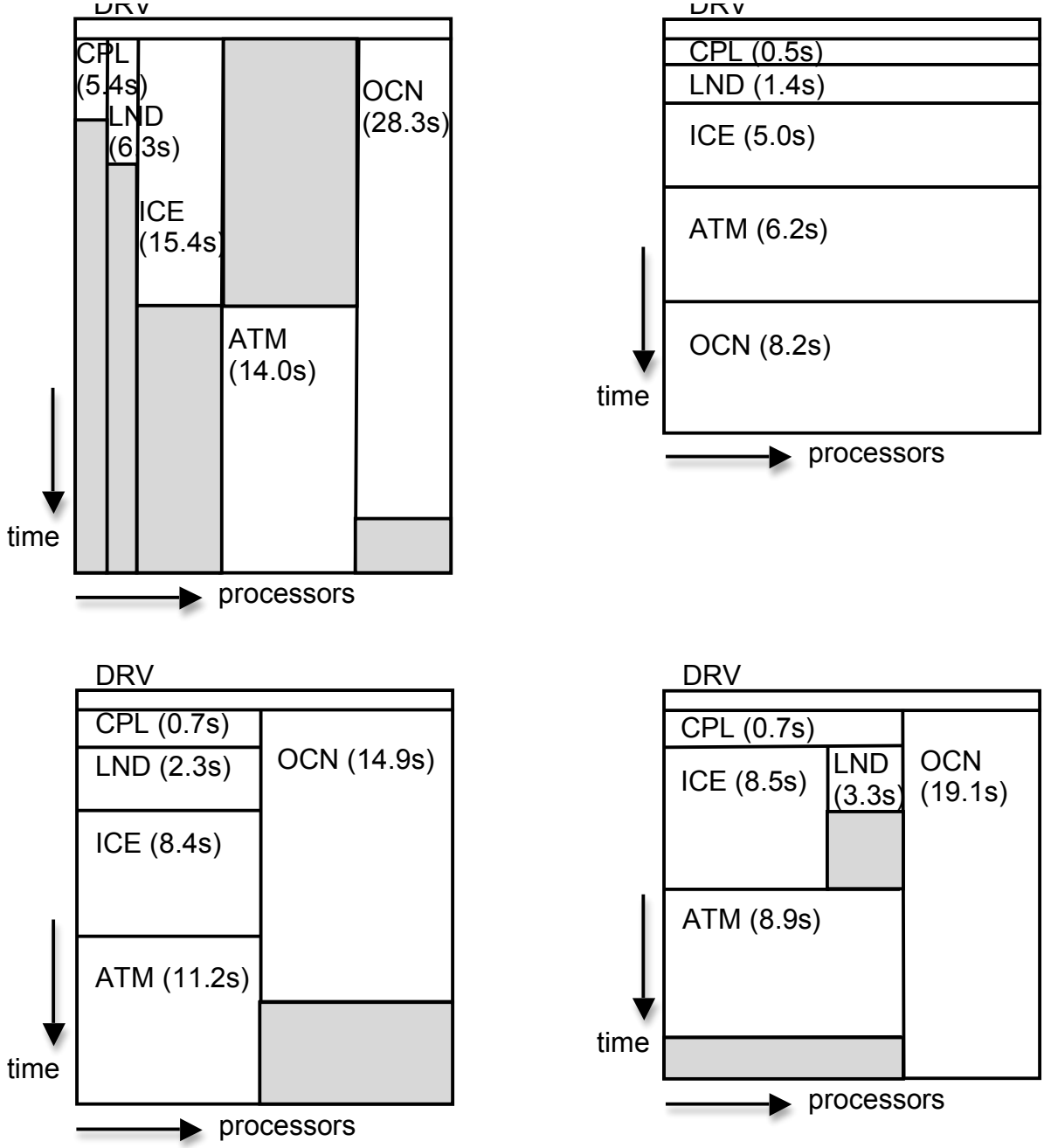

Figure 7: Schematics of CCSM4 component model timing for a fully prognostic "f19_g16" configuration running with four different processor layouts. In all cases, the total number of processors used is 128. Gray boxes represent idle time, and the time for each component in seconds per simulated model day is indicated. The total time for a) the fully concurrent layout is 33.5s; for b) the fully sequential layout is 20.8s; for c) the sequential layout with the ocean running concurrently with other components is 21.8s; and for d) a mixed sequential/concurrent layout is 19.1s.