

CLIMATE SIMULATION USING TIGHTLY COUPLED PROGRAM AND DATABASE MODULES

David R. Mikesell
John L. Pfaltz

Department of Computer Science
University of Virginia
Charlottesville, Virginia 22904-4740, U.S.A

ABSTRACT

We report our experience of tightly coupling a global change simulation of the terrestrial carbon cycle with an object-oriented database. By tightly coupled, we mean an interleaving of computational and data access statements within the same module so that intermediate model values are easily represented in persistent storage. This implementation approach opens up the “black box” character of simulation and has led to the discovery of errors within the simulation itself.

1 INTRODUCTION

In this paper we describe a global change simulation that has been implemented by tightly coupling the computational code modeling the phenomena with a database system. We believe that this approach to simulation offers significant benefits, even though it imposes some additional costs.

By “tightly coupled” we mean a simulation whose computational component repeatedly accesses data residing in the database component, and which in turn freely stores its intermediate values back in the database. Thus the database provides an intermediary storage capacity in much the same manner as the operating system’s “heap” or “free store” facility. Such intermediate values are not private to a single code module, but may be accessed by multiple modules. Because the database is persistent; it remains even after the simulation is complete and thus provides a “trace” of the simulation behavior.

We assume that the simulation involves relatively complex computation. For the purposes of this paper, we will also assume that the simulation is “spatio-temporal”, that is we are modeling some phenomena occurring in time over a two-dimensional surface. While we believe our approach is valid for a much wider class of simulations, our direct experience has been with these.

In Section 2, we briefly describe the Spring model of global change and its overall computational structure followed in Section 3, by a brief description of its implementation using the ADAMS database system. Section 4 details some of our experiences which lead us to recommend this approach; Section 5 details the associated costs.

2 A MODEL OF THE CARBON CYCLE

With the recent emphasis on greenhouse gases and global warming (Fletcher 2000; Mann, Bradley, and Hughes 1998; Morrissey and Justus 2000), there has been effort to understand the underlying “carbon cycle” through simulation (Parton, McKeown, Kirchner, and Ojima 1992; Ryan, McMurtrie, Agren, E. R. Hunt Jr., Aper, Friend, Rastetter, and Pulliam 1997; Woodward, Smith, and Emanuel 1995). These models describe the net binding of carbon into plant material and thus removing it from the atmosphere. The amount of carbon, so fixed, is called “net primary production”, or *npp*. Determining a realistic value of *npp* is thus an essential step in modeling global warming as a whole.

Just as a model of global warming is comprised of several independent modules, such as “net primary production” and “industrial emissions”, so too is a simulation of *npp* comprised of separate modules that simulate “biological activity”. These assume a specific “leaf area index” which is based on estimated atmospheric parameters such as “rainfall”, “temperature” and “hours of day light”, as well as soil parameters such as “nitrogen content”. Base parameters are usually observed from distributed sensors, or estimated by various interpolation methods. Intermediate parameters such as “leaf area index” (which denotes the ratio of leaf area to underlying ground area) are typically computed for a specific spatial element at a specific time of year. From these parameters, and others, “assimilation (the total amount of carbon entering a plant) and “net primary production” are finally calculated.

None of this is unusual. Mathematical descriptions of microscopic biological processes can seldom be combined into a closed form mathematical system that describes the overall macroscopic behavior. So we combine them by means of a simulation. Logically, the simulation is a “black box” which, given specific input values, generates corresponding output values. The values of the principal parameter(s) sought by the simulation are written to files (commonly using ASCII or netCDF format) which may be ordered with respect to spatial location, or with respect to time, in whatever manner seems most reasonable. Alternatively, these output values could be stored in a database. Increasingly, simulations are executed *over* a database, but these need not be closely coupled. In any case, the final output can be then visualized by any one of a number of freely available tools. such as NCAR Graphics (NCAR Scientific Computing Division 1998) or *GMT*, Generic Mapping Tools (Wessel and Smith 1998). Visualization tools have been integrated into the earlier versions of the CENTURY system as well (Parton, McKeown, Kirchner, and Ojima 1992). Figure 1 displaying calculated net primary production over the United States for a single year is a representative visual example that we have generated using *GMT* (It is much more striking in color).

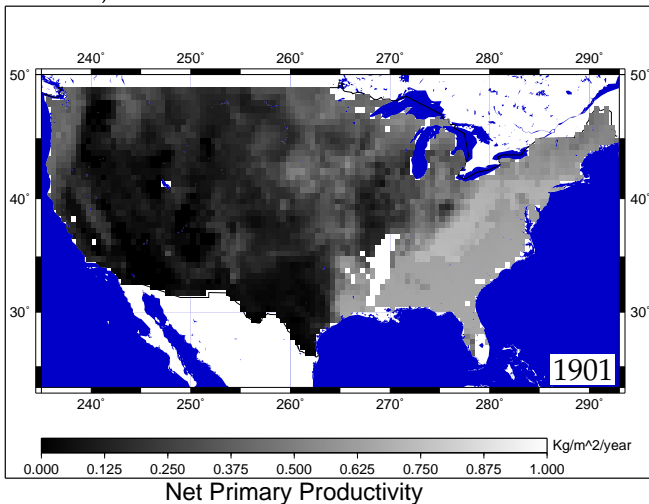


Figure 1: Typical display of annual net primary production, *npp*.

The particular model of the carbon cycle that we have implemented, and illustrated in Figure 1, is the “Spring” model (Emanuel, King, and Post 1994; Woodward, Smith, and Emanuel 1995), an evolving model created by the Environmental Science Department at the University of Virginia. The authors have undertaken the task of porting a file based C implementation of the Spring simulation model to an integrated C++

version using an object-oriented database. This task is essentially complete, and this paper describes some of the observations made in route. To capture the spatial component we have elected to use a pixel-like representation. Each land element has 0.5° by 0.5° resolution. Where higher resolution is required, these dimensions can be halved, or quartered. Our overall temporal resolution is a *year*. Temporal resolution can also be sharpened; but, for reasons not germane to this paper, it is much more difficult.

3 DATABASE IMPLEMENTATION

Our simulation has been implemented using an object-oriented database system, called ADAMS, that has been developed at the University of Virginia (Pfaltz 1993; Pfaltz and French 1993). The runtime operators of ADAMS are invoked through an imbedded language. That is, ADAMS statements are embedded directly within procedures written in a host programming language, such as C, C++, Pascal or Fortran. All of the following examples will be snippets of code copied directly from the simulation. Where they are not self-explanatory, we will provide some supplementary description. But, it is not our intention to describe either the ADAMS database syntax or the Spring global change model in detail. It is their interactive coupling that we wish to focus on. We will, however, observe that all ADAMS statements are delimited by `<< ... >>`.

Since ADAMS is object-oriented, each stored item must belong to a class. Figure 2 illustrates the `LAND_ELEMENT` class. (By convention, we capital-

```
<< LAND_ELEMENT isa CLASS
  having site = { lon, lat, loc_name, elevation }
  having soil = { soil_depth, field_cap, wilt_pt,
                bulk_d, soil_mass, ws_o }
  having intervals = { intervals } >>
```

Figure 2: Declaration of a *land element* CLASS in ADAMS

ize class names, and set object instance names in lower case.) The attributes specific to a *land_element*, that is *longitude*, *latitude*, *location_name* (if any), *elevation*, *average soil depth*, *field capacity*, *wilting point*, *bulk density*, *soil mass* and *initial soil moisture* (*ws_o*) have been separated into two collections called *site* and *soil* respectively.

However, many of the relevant attributes/parameters of any *land_element* have a temporal component. For example, *relative humidity* is dependent on the time of year. The *intervals* attribute in a *land_element* object is a link to a set of time interval data associated with that *land_element*. Figure 3 illustrates the class of `T_INTERVAL`.

```

<< T_INTERVAL isa CLASS
  having time = { i_start, i_end, i_duration }
  having climate = { phot_flux_d, day_len, temp,
                    rel_hum, wnd_spd, precip,
                    atm_co2 }
  having soilchem = { soil_c, soil_n }
  having canopy = { lai, assimilation, assim_bot,
                   npp }
  having moisture = { precip, evapotranspir }
  having holdridge = { HZone, HLatRegion, HAltBelt }
  having links_to = { location, previous, next }
>>

```

Figure 3: Declaration of a *time interval* CLASS in ADAMS

Besides the obvious attributes denoting the *start*, *end*, and *duration* of the interval, there are 18 biologically significant attributes separated into *climate*, *soilchem*, *canopy*, *moisture* and *Holdridge* categories. In addition, we have the link *location* to the associated spatial element, and links to the *previous* and *next* interval in this sequence.

Each biological parameter, for example *soil_n*, is a REAL attribute. Moreover, since ADAMS is object-oriented, and since a parameter such as *soil_n* is itself an object, we may associate attributes with these parameters, as in Figure 4.

```

<< soil_n.p_full_name <- "Soil Nitrogen Density" >>
<< soil_n.p_units <- "g/m^2" >>
<< soil_n.p_reference <- "Spring documentation" >>

```

Figure 4: Metadata associated with the *soil_n* parameter (or attribute)

Here we see that *soil_n* denotes “soil nitrogen density”, and is measured in terms of “grams per square meter” with a defining reference in the current Spring documentation. As we refine our metadata the *preference* will contain page numbers and other more specific data. The provision of metadata within a simulation, that can be interrogated by the executing code, is one example of tight coupling.

Figure 5 constitutes a second example of what we mean by tight coupling. This main simulation loop consists of ADAMS statements embedded within executable C++ code statements.

The name *ti* denotes an ADAMS object. The variables *year*, *startYear* and *endYear* are C++ variables. The next line invokes *load_sim_env*, which is an ADAMS procedure to input initial parameters from the database into the Spring model *interval* data structures. It accepts both the ADAMS objects *land_element* and *ti*, as well as C++ variables *cn*, *n*, *e* and *year* (delimited by *\$*) as its parameters. This

```

<< for_each land_element in land_set do
  << ti instantiates_a T_INTERVAL >>
  year = startYear;
  while (year <= endYear)
    { // load data into simulation structures
  << invoke load_sim_env (land_element, ti $ cn,
                        m, e, year $) >>
      // check for error condition
  if ((e->long != -9999) && (e->lat != -9999))
    {
      cn->lai = bisect_lai(LAI_TOL, LAI_MAX, cn,
                        m, e);
  << invoke store_parm_t(ti, lai $ cn->lai $) >>
      npp = 0.012 * npp_val(ann_a_val( cn, m, e),
                          cn->ml, e->c->temp);
  << invoke store_parm_t(ti, npp $ cn->npp $) >>
    }
  else
    { // didn't load the data correctly.
      cout << "error: load_sim_env_structs\n" ;
      goto dead;
    }
  year++;
}
<< end_for_each >>

```

Figure 5: Database operations embedded in computational code

is “tightly coupled” compiled code. The C++ driver program processes persistent database names as if they were in its own namespace (Combining two separate name spaces is the most difficult aspect of tightly coupled computing (Pfaltz 1993). ADAMS does not provide a completely seamless joining — but it does seem to be fairly effective anyway). Consequently, the ADAMS data space can be easily treated as an extension of the simulation’s own data space.

4 ADVANTAGES OF TIGHT COUPLING

We gave a quick overview of the underlying Spring model in Section 2 where we hinted at the interactions between various portions of the underlying model. Figure 6, which we have taken from (Shugart 1998), illustrates how macroscopic biological behavior can be inferred from several individual microscopic behaviors as determined through controlled experiments in the field, or a laboratory. And here we are only determining the photosynthesis rate limits, W_c , W_j and J . There are similar interactions between many of the 24 biological parameters that are now represented in the database.

But, how can one verify that the basic formulas have been correctly written? How can one verify that they are combined correctly? Even small errors can grow with cumulative simulations.

The traditional construction of environmental simulations is that of a functional “black box”. Given various input parameters, one derives corresponding output

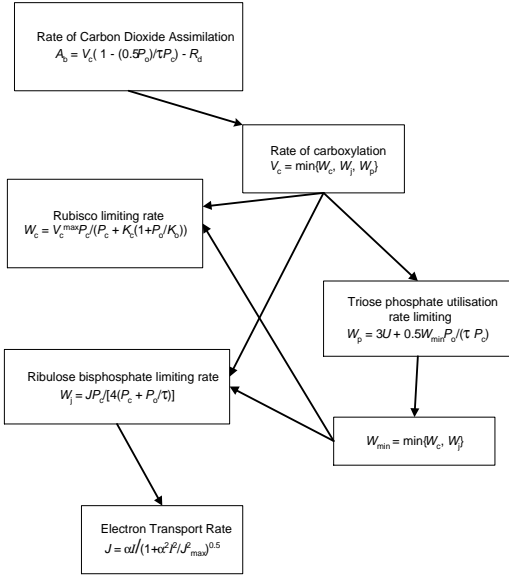


Figure 6: Interaction of computations modeling photosynthesis rate limitation within the Spring model

parameters. By tightly coupling the simulating black box with a database, we can easily store intermediate parameters for subsequent analysis. Snapshots of any of them can be taken and displayed in the manner of Figure 1. One can determine if these “look right”. Because so many visualization products are C++, or C, driven, it is easy to insert appropriate ADAMS statements into the visualization tool to select the parameters, or combinations of parameters, to be displayed. In the file based incarnation of the Spring simulation this was exceedingly difficult. Special programs had to be written to extract the appropriate values from the files in the correct order. Because it was difficult, it was not done. In effect, by closely coupling the code with the database we provide the global change researcher with a kind of debugging tool that permits examination of an executing simulation, in much the same way that programmers can trace variables in an executing program.

The benefits of this approach was made apparent early in our investigation of the Spring model. After executing the model over a year-long dataset, we examined the values of some of the internal model variables. An examination of the parameter *assim_bot*, which stores carbon assimilation values for the bottom canopy layer, showed that, in some cases, the value for this parameter exceeded the value stored in *assim*, the sum of the carbon assimilation values for *all* canopy layers. By using simple queries over our database, it was easy to determine that this occurred only in those cases where the leaf area index value (*lai*) was less than

1. This indicated that there was a fundamental error in the canopy module code which made itself evident in the boundary case when *lai* was less than 1 (see Figure 7).

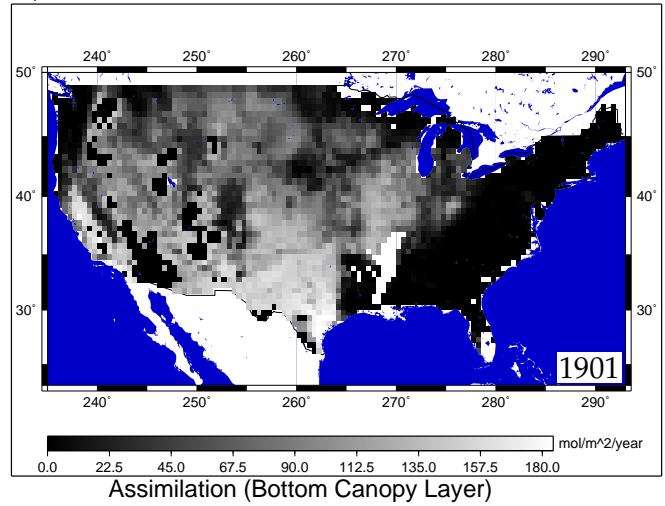


Figure 7: Carbon assimilation in the lowest canopy layer, *assim_bot*, showing unexpectedly high results in western US desert areas.

Later, we had run spatial displays of most of the biological parameters enumerated in Figure 3. All seemed quite reasonable. We began creating time sequenced (from 1901 to 1994) spatial videos of *npp*. It varies considerably from year to year, especially in the western Appalachians and Ohio basin. This surprised us; we would have thought the process would have been more constant. But, it could be so. Still, we investigated this phenomena a bit further. We discovered that *lai* (leaf area index) also fluctuated widely from year to year. It was closely correlated with *precip* (annual rain fall). This makes some sense; but we know that mature trees, which constitute the major biota in this region, do not drastically change their leaf area from year to year. The Spring model does not appropriately model biological stability. Because of our observations, the authors of the Spring model are re-examining their fundamental equations.

Because in a closely coupled system, where many of the intermediate parameters are captured in a database, it is easy to visualize their distribution and the change over time, one achieves the kind of “debugging” capability that we have described. But, the visualization benefits are only part of the story. We asserted that *lai* is closely correlated with *precip*. This is visually true. It can also be measured. It is almost trivial to include your favorite spatial, and/or temporal, correlation algorithm, such as those discussed in (Gottwald, Richie, and Campbell 1992; Kelley, Barry, Clapp, and

Rodriguez 1998), and then apply it to those parameters which one extracts from the database. This ability to facilitate further scientific investigation seems almost more important than simple debugging.

5 COSTS OF TIGHT COUPLING

As one might expect, the debugging and analysis facility that we have described does not come without a cost. Accessing input data via a database can require multiple disk accesses per data item due to index lookups, while flat files can be read sequentially. Furthermore, there is a substantial one time cost to load the data into the database. Measuring the total performance cost, however, is problematic. System load, secondary storage subsystem performance, and the efficiency of the implementation all influence the magnitude of this cost. However, we have found that, on the average, our implementation on a dual processor Sun Enterprise-250 compute server using ADAMS runs at about 60–70% of the speed of the flat file implementation. This timing overhead becomes less onerous when the simulation is distributed over several processors to achieve concurrent execution. We have observed superlinear speed-up, and near linear scale-up, in a distributed, shared nothing version of ADAMS (Pfaltz, Haddleton, and French 1998).

Since ADAMS fully indexes each attribute stored in the database, and since the ADAMS internal representation of numerical values is 12 bytes long (compared to typically 8 bytes for a C language *double* value), there is also an additional expense in terms of secondary storage. The storage requirement for the 94 year simulation described here using ADAMS is approximately 3.2 GB. If the same data were simply stored as flat files, the storage requirement would be about 0.8GB. Given the ever-decreasing cost of secondary storage, we believe that a factor of four increase in storage requirements is a small price to pay for the added functionality of a database.

An additional cost is a potential increase in maintenance requirements of the model code. When a simulation is coded over flat files, so long as the input/output statements are unchanged, its internal logic can pretty much be changed at will. But when the database code is intertwined with the model code, any major changes to the model code may need to be propagated to the database. For example, exchanging a module which uses the parameter *incident solar radiation* with one that uses *photon flux density* instead, would require that any reads from, or writes to, the *incident solar radiation* value stored in the database be changed to reflect the new parameter. Additionally, if *incident solar radiation* was not included in the database schema, then of course

it would need to be added. The model maintainer needs to be mindful of the effects of model revisions on the database. But, since the database statements are part of the simulation code, this is difficult to overlook.

6 SUMMARY

A simulation becomes more tightly coupled with a database as more of its internal variables are represented in the data space as opposed to local stack space or system heap space. As we have seen in Section 5, this imposes additional costs on the simulation. It will execute more slowly and require more persistent storage.

But, random access via the database provides the researcher with the flexibility to easily process the data in any order, not just that which is preordained by a flat file layout. It also supports local spatial subdivision to simulate the phenomena in more detail in critical areas. But, perhaps most important, tightly coupling the simulation to a database opens the door to a greater understanding of precisely what is happening “inside” the simulation. It becomes less of a “black box” as we are now able to examine model intermediate values and their relationship to each other as well as to inputs and outputs. As we have shown through our own experience, this has immeasurable value in model debugging and development. Fast simulations that are incorrect have little value, but even more malefic are systems that are believed to be correct but are not. And it is difficult to show correctness in a complex model without a thorough examination of its internals. Furthermore, this kind of close coupling also opens the door for a scientific analysis of the values of intermediate parameters as well as the specified output. The versatility and transparency offered by this approach can provide the model developer new and invaluable insights into the workings of a simulation, thus enabling the developer to build higher fidelity and more robust models.

7 ACKNOWLEDGEMENTS

This research was supported in part by DOE grant DE-FG05-95ER25254.

REFERENCES

- Emanuel, W. R., A. W. King, and W. M. Post. 1994. *A dynamic model of terrestrial carbon cycling*. Springer Verlag.
- Fletcher, S. R. 2000. *98-2: Global climate change treaty: The Kyoto Protocol*. Washington, D.C.: Congressional Research Service. Made available to the public by the National Council For Science and the Environment, Washington, D.C.

- Gottwald, T. R., S. M. Richie, and C. L. Campbell. 1992. LCOR2 – spatial correlation analysis software for the personal computer. *Plant Disease* 76 (2): 213–215.
- Kelley, R., R. Barry, J. Clapp, and M. Rodriguez. 1998. Spatio-temporal estimation of neighborhood effects. *J. of Real Estate Finance and Economics* 17 (1): 15–33.
- Mann, M. E., R. S. Bradley, and M. K. Hughes. 1998. Global-scale temperature patterns and climate forcing over the past six centuries. *Nature* 392:779–787.
- Morrissey, W. A., and J. R. Justus. 2000. *Ib89005: Global climate change*. Washington, D.C.: Congressional Research Service. Made available to the public by the National Council For Science and the Environment, Washington, D.C.
- NCAR Scientific Computing Division 1998. SCD FY1998 Annual Scientific Report. Technical report, NCAR.
- Parton, W. J., B. McKeown, V. Kirchner, and D. Ojima. 1992. *CENTURY users manual*. Fort Collins, Colorado: Colorado State University.
- Pfaltz, J. L. 1993, April. The ADAMS language: A tutorial and reference manual. Technical Report IPC TR-93-003, Institute for Parallel Computation, University of Virginia. Found at <http://www.cs.virginia.edu/adams>.
- Pfaltz, J. L., and J. C. French. 1993, March. Scientific database management with ADAMS. *Data Engineering* 16 (1): 14–18.
- Pfaltz, J. L., R. F. Haddleton, and J. C. French. 1998, July. Scalable, parallel, scientific databases. In *10th International Conf. on Scientific and Statistical Database Management*, 4–11. Capri, Italy.
- Ryan, M. G., R. E. McMurtrie, G. Agren, E. R. Hunt Jr., J. D. Aper, A. D. Friend, E. B. Rastetter, and W. M. Pulliam. 1997. *Global change: Effects on coniferous forests and grasslands*, Chapter Comparing Models of Ecosystem Function for Temperate Conifer Forests II: Simulations of the Effect of Climate Change. Wiley.
- Shugart, H. H. 1998. *Terrestrial ecosystems in changing environments*. Cambridge University Press.
- Wessel, P., and W. H. F. Smith. 1998. New, improved version of Generic Mapping Tools released. *EOS Trans. Amer. Geophys. U.* 79 (47): 579.
- Woodward, F. I., T. M. Smith, and W. R. Emanuel. 1995. A global land primary productivity and phytogeography model. *Global Biochemical Cycles* 9:471–490.