Improving HIRLAM Scalability by Asynchronous GRIB File Handling

Jan Boerhout
Sun Microsystems
September 13, 2001

Abstract

The more processors are used when a highly parallel application, the more significant the impact of the sequential I/O part. HIRLAM is an example of such an application. The time spent reading and decoding GRIB input files as well as encoding and writing GRIB output files remains constant, whereas the compute time decreases almost linearly with the number of processors used. The scalability of HIRLAM is improved substantially by separating the I/O handling part from the numerical part of the program.

Introduction

The HIRLAM I/O handling part (real disk I/O time and GRIB code processing time) has been identified as a bottleneck, which limits the scalability of the program. In order to solve this problem, the I/O part has been isolated from the computational model part, resulting in much better parallel efficiency.

This article describes how it was done, presenting details with respect to method, implementation and resulting performance.

The HIRLAM program used is the SHMEM version of release 4.9.1, as ported to a Sun Enterprise 10000 server (further referred to as E10000) with 64 processors and the Solaris Operating Environment.

Method

In the HIRLAM main program a new subroutine HGS_INIT is called (before SHMEM initialization), which starts a second process inside the HIRLAM context, but outside the SHMEM context (it does not have a PE number). This second process is called the *HIRLAM Gribfile Server* or *HGS*.

The original process runs the HIRLAM model and starts with initializing SHMEM and calling main subroutine HLPROG. While the SHMEM environment is being initialized, HGS prereads the first two input GRIB files, decodes the data and stores the resulting field data in a new shared memory segment. This first HGS action is nicely hidden behind the SHMEM initialization: both take about the same amount of time.

Once HIRLAM is ready to use the data from the first input files, these data are already located in proper machine representation in the corresponding shared memory segments. The input data are copied from these segments into the corresponding field arrays.

All processors (or PE's in SHMEM terminology) retrieve their part of the input data in parallel, reducing the time it takes to copy the data to a negligable amount.

HGS continuously reads the next boundary input files ahead of time, making sure that the data will be available in time.

Once the first input files needed have been read as described, the next actions of HGS are the preparations for the first output files (history and post-processing files). Shared memory segments are created and initialized (paged-in) for these files.

Output files are written to GRIB files in a similar fashion. The HIRLAM GRIB output routines have been modified such, that output data to be GRIB-encoded are copied to a shared memory segment, which has been prepared by HGS. Each PE copies its local output data to the appropriate part of the shared memory segment, in parallel, similar to the input method. Because the output at this point is simply a data copy operation, executed in parallel, output is also handled very fast.

Once HIRLAM has written all output data for a particular file to the related shared memory segment, HGS is triggered to start the engrib and store operations, which will continue in parallel with the HIRLAM model execution.

When the model rereads the output data (between the HIRLAM initialization and forecast phases), it will reread the data from the shared memory segment, so there is no delay due to unnecessary time-consuming encode and subsequent decode operations.

Implementation

The central mechanism used for HGS is the standard Unix System V shared memory support. It consists of

- a number of system calls to create, attach, detach and delete memory segments, which are accessible to other processes, depending on read/write/execute protection bits
- similar routines supporting semaphores and message queues (not used in HGS)
- a number of commands to delete or to list details of existing shared memory segments (and semaphores, etc.)

Please note that the use of shared memory segments for HGS is independent of the SHMEM implementation. Because of its superior scalability, the SHMEM version of HIRLAM was chosen for the HGS implementation. There are no reasons, why this method would not work with MPI.

The GRIB handling and I/O sections of the HIRLAM code are quite esoteric. Tampering by non-experts has a high risk of introducing subtle, hard to find bugs. Therefore, care has been taken to:

- limit the number of code modification locations to a minimum;
- only change the field data flow at the highest possible level;
- keep the lower level code sections responsible for field I/O unchanged.

HGS is activated by calling the subroutine HGS_INIT from the main Fortran program. This routine forks off a new process. The parent process returns to the main program, which subsequently calls SHMEM_INIT and the top HIRLAM subroutine HLPROG. The child process serves as HGS process, which subsequently reads the NAMPRC namelist (providing the global grid dimensions needed to call the I/O routines) and alternatingly prereads input files and writes output files throughout the intialization and forecast execution phases of the model itself.

> The fork(2) feature has been used to create an independent HGS process. An alternative method would be the creation of an extra thread for HGS within the rank 0 HIRLAM process, either by the operating system's native thread support or by the Posix *pthread* library.

Input handling

HGS calls the unmodified subroutine GETGRB to read the input files. The model itself also calls this subroutine from the usual location in the HIRLAM call tree, part of which is presented in figure 1. The shaded blocks represent unmodified routines, whereas the white blocks indicate which routines are modified to accomodate HGS.

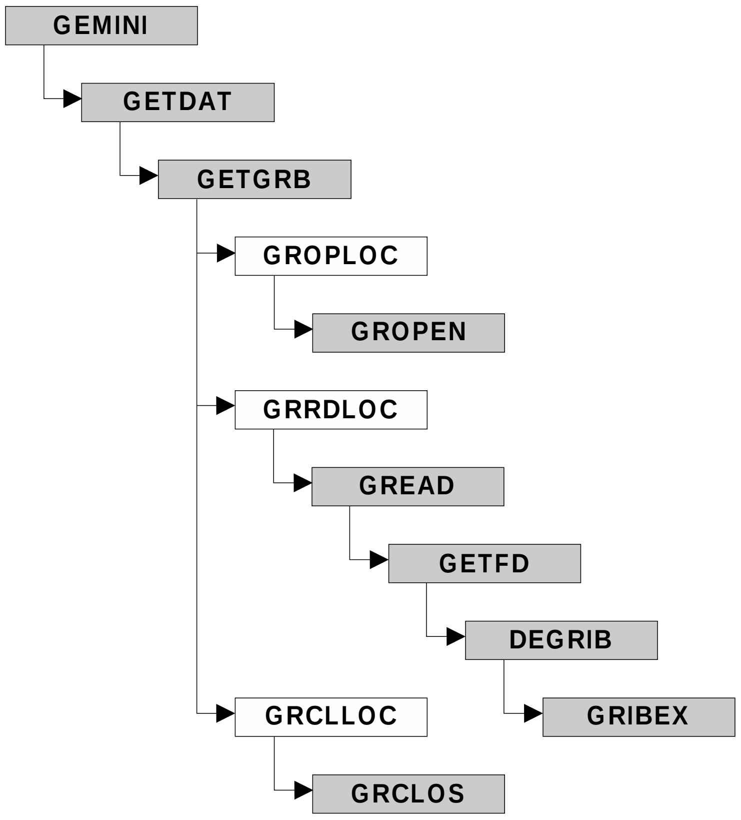

Fig 1 Partial GRIB input call tree

The functionality of the "white" subroutines depends on the context of the calls, as explained below.

When called from HGS prereading the input files, the original HIRLAM code is executed to open, read and close the GRIB input file by calling the unmodified input routines GROPEN, GREAD and GRCLOS, shown in figure 1. In addition, HGS creates a shared memory segment, saves a copy of the input field to a record in this segment.

When called from HIRLAM, the proper shared memory segment is opened and all HIRLAM processes read their local field simultaneously from the global field data located in the segment. The original GRIB input routines are not called.

Some of the input files are read multiple times. Each time the data are copied from the shared memory segment. Once an input file has been read for the last time, the shared memory segment is closed, returning its memory pages to the operating system.

Output handling

HGS treats output similarly. Before HIRLAM produces output, HGS creates a shared memory segment for the next output file to be written. The original partial output call tree is presented in figure 2.

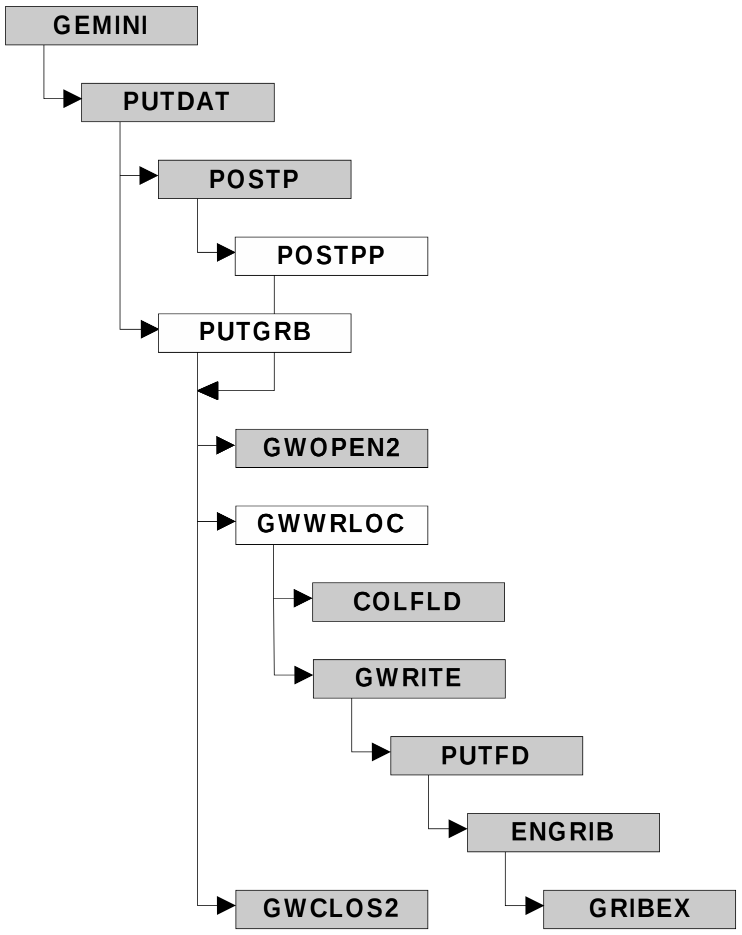

Fig 2 Partial GRIB output call tree (original)

The routines POSTPP and PUTGRB have been modified such, that instead of calling GWOPEN2 and GWCLOS2 corresponding HGS versions are called, whereas GWWRLOC has been modified to call HGS_GWRITE.

Figure 3 shows the HGS version of the partial output call tree. Routine HGS_GWOPEN2 opens the output file's shared memory segment, which HGS has already created beforehand. The DDR and other file parameters are saved in the segment's administration structure.

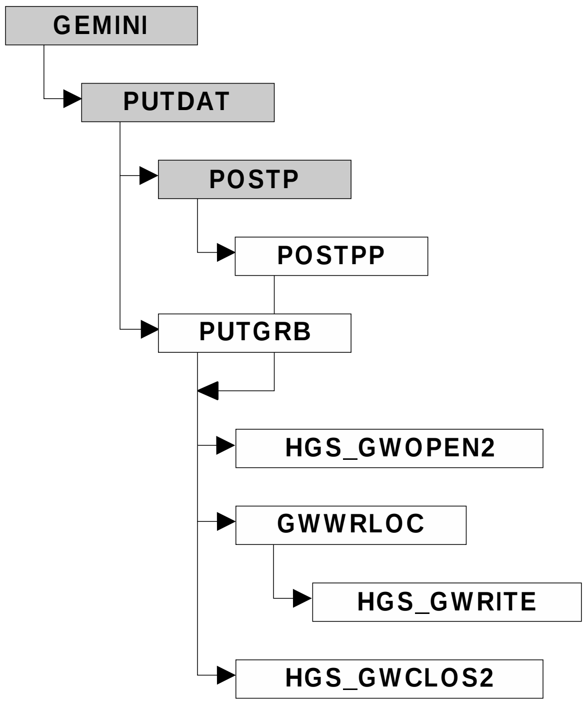

Fig 3 Partial GRIB output call tree (modified)

GWWRLOC (called for each field) saves an output field to the shared memory segment. It first calls the new HGS_GWRITE routine, which keeps track of the output records stored in the shared memory segment, returning the address of the next record, without actually copying any data. The address returned is used by GWWRLOC to copy the local field rows to the right positions in the global field rows stored in the shared memory segment. All HIRLAM processes perform this local to global field copy operation in parallel.

Routine HGS_GWCLOS2 marks the segment as ready and writes its identification number (same as Fortran unit number) to the synchronization pipe telling HGS that it may encode the data saved in the segment and write the results to the GRIB output file. HIRLAM does not wait for HGS for this to finish.

HGS had already been waiting for the "ready" signal. It reads the segment's identification number and calls C function `hgs_engrib_and_store()` , which uses the original HIRLAM routines GWOPEN2, GWRITE and GWCLOS2.

Reusing output files

Between the two initialization and the forecast phases the files written by one phase is input by the next. Some files are even read repeatedly. The HGS version avoids unnecessary encoding/decoding, because the memory segments are preserved between the phases. Though this only improves the initialization process, not the forecast itself, the gain is big enough to be noted.

The current HGS version waits until all output files have been created before it terminates. If the programs following the HIRLAM run process the output files in sequence, some more time would be saved by terminating the main program while the HGS process is still writing the last output file.

> It has been noted that a new initialization method is currently being developed by the HIRLAM group, which eliminates the need to store and reread the initialization results.

Current implementation status

The current HGS implementation is fully functional, but one shortcut has been taken. The number of input files, output files, the logical unit numbers used and the sequence of initialization phases as defined in the HIRLAM benchmark have been hard-coded into the HGS process.

The NAMDEV namelist input data should be interpreted for all phases for HGS to preread input files and store the output files according to the scheme defined in the namelist information. This is a relatively straight-forward effort, which will be completed as part of ongoing cooperation between the HIRLAM consortium, KNMI and Sun Microsystems.

How many processors

There are several possibilities to decompose the HIRLAM grid into subgrids in two dimensions. It is customary to choose a number of subgrids in each dimension (called NPROCX and NPROCY), such that the product is the number of processors available on the compute server. On a 64 processor system one would choose 8 in each dimension, each processor handling one subgrid.

If one takes one processor less, the numbers for NPROCX and NPROCY would be 7 and 9. Though one processor less would mean that the same amount of work would take longer to complete, in practice this is not always the case. It turns out, that freeing one processor for operating system duties not only compensates for the loss in compute power, but even results in a modest performance gain.

The processor not used for computations, being idle most of the time, is ideally suited to run the HGS process.

Performance

Table 1 summarizes the performance as measured on a Sun Enterprise 10000 Server (or

E10000 for convenience) with a processor clock frequency of 400MHz and an external cache size of 8MB per processor. The times are in hours:minutes:seconds for a full 48 hours forecast range, including the two initialization phases and all output files as specified in a recent HIRLAM version 4.9.1 benchmark.

Table 1 Performance HIRLAM with HGS

Processors	*Runtime*	*Speedup*	*Ideal*	*Efficiency*
17	02:15:22	1.00	1.00	
33	01:09:51	1.94	1.94	100%
64	00:37:55	3.57	3.76	95%

In all 3 cases one of the processors runs the HGS process.

The parallel efficiency of 95% on 64 processors is very high, considering

- that this is a complete run, including setup and I/O
- the amount of I/O and a time-consuming GRIB data conversion (total size of input and output data in GRIB format: 300 MB input and 630MB output).

The best runtime without HGS is 0:44:15 (on 63 processors), more than 6 minutes or 17% longer.

The benchmark's grid dimensions are 406 x 324 points and 31 levels, the time step was 360 seconds and the forecast range was 48 hours, history and postprocessing files were produced every 6 hours, boundary files were read in around the same time steps.

Conclusions

It has been demonstrated, that the time spent in the GRIB file I/O processing part of HIRLAM can be hidden completely behind the computational part. The method used is based on a reroute of the field data flow, instead of modifications in the delicate GRIB encoding and decoding program parts.

The scalability of the HIRLAM program as a whole is improved significantly. The runtime of a full 48 hours forecast run on a 64 processor Sun Enterprise 10000 server is reduced by 17% due to this optimization technique.