Running ADMB Models in Parallel with Open MPI

Derek Seiple

1 Open MPI

Open MPI is an open-source Message Passing Interface (MPI) that allows parallel computing. It is intended for use in high power computation. It is the parallelization tool used in ADMB.

2 Preparing ADMB for use with Open MPI

In order to take advantage of parallelization within ADMB, Open MPI must first be installed on the computer(s) the model will be run on. ADMB must also be properly ocnfigured to make use of the Open MPI capabilities within.

As of now the only way to take advantage of the parallelization is to compile form source using the correct arguments to enable it. ADMB can be compiled as normal using the configure script and no parallelization features will be accessable. All of the portions of code that make use of, or reference to, Open MPI features are enclosed in the macros:

```
#if defined(USE_ADMPI)
...
// MPI code
...
#endif
```

This gives the user the option of compiling either with or without Open MPI capability.

If the user does want to take advantage of parallelizastion within ADMB, then the source can be copiled in much the same way it is done now withone exception; `--enable-mpi` must be pased to the configure script. So the entire build process consists of:

```
make --directories scripts/configure
./configure --enable-mpi
make
```

Once compiled, all of the features necessary to run a program in parallel will be accessible. For now this only works on Linux. It will be extended to all supported platforms.

3 Running models with Open MPI

Once ADMB is compiled correctly, and you have a model that is written for paralleliation (see sections 4, 5, and 6), then you may run that model in parallel by passing one of the following command line options:

- `-master` - Runs program with one master process and one slave process.
- `-master -nslaves <num_slaves>` - Runs program with one master process and `<num_slaves>` slave processes.

Once again, specifying one of these command line arguments is neseccary in order for the program to run in parallel. This is so the user can develope a single model that can be run both serially as well as in parallel without having to recompile or maintain separate tpl files.

4 ADMB-MPI Functions for Developers

All of the calls to Open MPI are handled and encapsulated in the class named `admpi_manager`. The relavent instance of this class is called `mpi_manager` which is declared in the global class `ad_comm`. Therefore calls can be made to any of `mpi_manager`'s methods from anywhere within ADMB by

```
ad_comm::mpi_manager->method_you_wish_to_call();
```

Two calls of great importance are:

```
int admpi_manager::is_master(void);
int admpi_manager::is_slave(void);
```

which will tell you if you are in a slave or a master process. So to check if you are in the master process, for example you would call something like

```
if (ad_comm::mpi_manager)
{
  if (ad_comm::mpi_manager->is_master())
    {
      // master executes this.
    }
    else
    {
      // slaves executes this.
    }
}
else
{
  // program was not run with the -master command line argument
}
```

The above code block is very important because certain functions supplied by `admpi_manager` can only be called by the master process, while other can only be called by the slave process. If this is not followed, the program will crash with an Open MPI error message. In addition it shows how program execution is handled if the command line argument `-master` is not used. Also, for each function that is executed by the master, the corresponding slave function must be called and vice versa. See the table on page 2 to see what functions must be called by the master and which must be called my the slave. Functions in the same table row must be used together; the naming is intended to be intuative.

Functions that can only be called by the master	Functions that can only be called by a slave
`void send_int_to_slave(int i, _slave_number)`	`void get_int_from_master(int &i)`
`void send_double_to_slave(const double v, int _slave_number)`	`double get_double_from_master(void)` d
`void send_ivector_to_slave(const ivector& v, int _slave_number)`	`ivector get_ivector_from_master(void)`
`void send_dvector_to_slave(const dvector& v, int _slave_number)`	`dvector get_dvector_from_master(void)`
`void get_int_from_slave(int &i, int _slave_number)`	`void send_int_to_master(int i)`
`double get_double_from_slave(int _slave_number)`	`void send_double_to_master( const double v)`
`dvector get_dvector_from_slave( int _slave_number)`	`void send_dvector_to_master( const dvector& v)`

For example if you want all of the slaves to send a dvector back to the master the following block of code can be used.

```
if (ad_comm::mpi_manager->is_master())
{
  //get dvectors from slaves
  for(int si=1;si<=ad_comm::mpi_manager->get_num_slaves();si++)
  {
    dvector received = ad_comm::mpi_manager->get_dvector_from_slave(si);
  }}
}
else
{
  //send dvector to master
  ad_comm::mpi_manager->send_dvector_to_master(sent);
}
```

This example demonstrates a couple of important things. First it shows that there is a built-in function

```
ad_comm::mpi_manager->get_num_slaves()
```

that can be called from within a slave or the master that tells us how many slaves were spawned. Notice how the master uses that to loop through and receive a dvector from each of the slaves. This also showes that each slave is assigned a unique slave ID starting from 1 upto the number of slaves. Lastly, this block of code will get executed by each of the slaves and therefore each slave will send a dvector to the master. Each of those calls must be received by the master, so `get_dvector_from_slave` must be called multiple times.

5 Use with Hessian calculation

Some standard ADMB models have large Hessians that take a long time to compute. Using parallelization can be used to help reduce the amount of time it takes the computation takes.

For example take the catage example located in the ADMB examples folder. By executing it with the `-master` flag the Hessian portion will be computed in parallel. Only the master does the minimization portion of the program; the slave(s) wait for master to finish the minimization. Afterwards both the master and slave(s) work together by splitting up the rows that each will compute. This can be seen during execution as the Hessian estimation phase may output something that look like

```
Estimating row 1 out of 38 for hessian
Estimating row 20 out of 38 for hessian
Estimating row 2 out of 38 for hessian
Estimating row 3 out of 38 for hessian
Estimating row 21 out of 38 for hessian
...
```

instead of doing it in order one at a time.

6 Use with Separable Models

Effort is being made for ADMB with random effects to be able to use the Open MPI parallel capabilities. As of this writting a first iteration has been implemented for separable models.

A typical separable model will have a procedure section like this one

```
PROCEDURE_SECTION
  ...
  for (i=1;i<=nh;i++)
  {
    fun(i,j,u(i),log_theta1,beta);
  }
```

where `fun` is the separable function. The program will loop through calling the separable function in order. The calculation that take place in the separable function call are for the most part independant from each other, so the order in which they are called is not important as long as all calls are made and the proper values are accumulated properly. To do this in parallel we have to tell the master and slaves how to split up these calculations.

There is a class `ad_separable_manager` that automatically handles this. The only thing the user has to do is change the above procedure section to the following

```
PROCEDURE_SECTION
  ...
  separable_bounds(sb,1,nh);
  for (i=sb->indexmin();i<=sb->indexmax();i++)
  {
    fun(i,j,u(i),log_theta1,beta);
  }
```

The line `separable_bounds(sb,1,nh);` is a macro that automatically sets up and declares the instance of `ad_separable_manager` that gets used. `sb->indexmin()` and `sb->indexmax()` will automatically handle the bounds that the master and slaves will use. This separable bounds class is robust enough that the model will run correctly even if it is not started in parallel mode. Of course model specific features will have to be acounted for when developing a model.

7 What still needs to be done

There is a lot of work that still needs to be done. The following is a breif overview of things that still need to be finished. Other suggestions are welcome.

- Provide support to other operating systems and compilers.
- Streamline the configuration/build process so Open MPI can be enabled on all platforms easily.
- Add clustering option so the slave processes can run on a separate machine and communicate over a network (distributed computation).
- Expand on the types of models that can be run in parallel.
- Optimize parallelization to improve program running times.