Why Open MPI?

How to Enable/Access Open MPI in ADMB.

How is Open MPI Used in ADMB Now?

How it Works

Development and Future Directions

Investigation of Parallel Processing Using Open MPI

Derek Seiple

ADMB Developers Workshop

March 2012
Previous Parallelization Efforts

Previous effort was made to run ADMB with pthreads.
Previous Parallelization Efforts

Previous effort was made to run ADMB with pthreads.

Implemented a thread pool, but it didn’t work.
Previous effort was made to run ADMB with pthreads.

Implemented a thread pool, but it didn’t work.

Discovered that there were static global variables.
Open MPI is an open source Message Passing Interface library which must be installed.
Why Open MPI is the Solution

Open MPI is an open source Message Passing Interface library which must be installed.

Open MPI gets around this by having separate memory (processes).
Why Open MPI is the Solution

Open MPI is an open source Message Passing Interface library which must be installed.

Open MPI gets around this by having separate memory (processes).

A master process spawns slaves (copies) each containing its own memory.
Open MPI is the Solution

Open MPI is an open source Message Passing Interface library which must be installed.

Open MPI gets around this by having separate memory (processes).

A master process spawns slaves (copies) each containing its own memory.

The master and slaves then communicate the important pieces of information.
Why Open MPI?

How to Enable/Access Open MPI in ADMB.

How is Open MPI Used in ADMB Now?

How it Works

Development and Future Directions

The Building Blocks

Master

int is_master(void)

void send_int_to_slave(int i, int _slave_number)

void send_double_to_slave(const double v, int _slave_number)

void send_ivector_to_slave(const ivector& v, int _slave_number)

void send_dvector_to_slave(const dvector& v, int _slave_number)

Slave

int is_slave(void)

void get_int_from_slave(int &i, int _slave_number)

void get_double_from_slave(int _slave_number)

dvector get_dvector_from_slave(int _slave_number)

void send_int_to_master(int i)

void send_double_to_master(const double v)

void send_dvector_to_master(const dvector& v)

--> void get_int_from_master(int &i)

--> double get_double_from_master(void)

--> ivector get_ivector_from_master(void)

--> dvector get_dvector_from_master(void)
## The Building Blocks

<table>
<thead>
<tr>
<th>Master</th>
<th>Slave</th>
</tr>
</thead>
<tbody>
<tr>
<td>int is_master(void)</td>
<td>int is_slave(void)</td>
</tr>
<tr>
<td>void send_int_to_slave(int i,</td>
<td>--&gt; void get_int_from_master(int &amp;i)</td>
</tr>
<tr>
<td>int _slave_number)</td>
<td>double get_double_from_master(void)</td>
</tr>
<tr>
<td>void send_double_to_slave(const double v,</td>
<td>ivector get_ivector_from_master(void)</td>
</tr>
<tr>
<td>int _slave_number)</td>
<td>--&gt; dvector get_dvector_from_master(void)</td>
</tr>
<tr>
<td>void send_ivector_to_slave(const ivector&amp; v,</td>
<td></td>
</tr>
<tr>
<td>int _slave_number)</td>
<td></td>
</tr>
<tr>
<td>void send_dvector_to_slave(const dvector&amp; v,</td>
<td></td>
</tr>
<tr>
<td>int _slave_number)</td>
<td></td>
</tr>
<tr>
<td>void get_int_from_slave(int &amp;i,</td>
<td>&lt;-- void send_int_to_master(int i)</td>
</tr>
<tr>
<td>int _slave_number)</td>
<td>double get_double_from_slave(int _slave_number)</td>
</tr>
<tr>
<td></td>
<td>&lt;-- void send_double_to_master(const double v)</td>
</tr>
<tr>
<td></td>
<td>dvector get_dvector_from_slave(int _slave_number)</td>
</tr>
<tr>
<td></td>
<td>&lt;-- void send_dvector_to_master(const dvector&amp; v)</td>
</tr>
</tbody>
</table>
All portions of code containing Open MPI functionality should be enclosed in the macro USE_ADMPI.
All portions of code containing Open MPI functionality should be enclosed in the macro USE_ADMPI.

```c
#if defined(USE_ADMPI)
...
// MPI code
...
#endif
```
All portions of code containing Open MPI functionality should be enclosed in the macro `USE_ADMPI`.

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

Must pass `-DUSE_ADMPI` (in g++) to compiler.
Open MPI recommends using one of their “wrapper compilers” e.g. mpicxx.
Open MPI recommends using one of their “wrapper compilers” e.g. `mpicxx`.

The configure script handles all of this (on Linux for now).
Open MPI recommends using one of their “wrapper compilers” e.g. mpicxx.

The configure script handles all of this (on Linux for now).

To build:

```
make --directories scripts/configure
./configure --enable-mpi
make
```
Use of -master and -nslaves

Program will only run in parallel if you specify:

- `master`
- `master -nslaves <num_slaves>`
Use of -master and -nslaves

Program will only run in parallel if you specify:

- master
- master -nslaves <num_slaves>

Example

./program -master

will run with one master process and one slave process.
Use of -master and -nslaves

Program will only run in parallel if you specify:

- master
- master -nslaves <num_slaves>

Example

./program -master

will run with one master process and one slave process.

./program -master -nslaves <num_slaves>

will run with one master process and <num_slaves> slave processes.
How is Open MPI Used in ADMB Now?

The Hessian calculation for standard ADMB models (Thanks to Dave).

Example: `./catage -master -nslaves 2`

The master does the minimization. Slave(s) wait for master then split up the Hessian calculation.

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
...

Development and Future Directions
How is Open MPI Used in ADMB Now?

The Hessian calculation for standard ADMB models (Thanks to Dave).
How is Open MPI Used in ADMB Now?

The Hessian calculation for standard ADMB models (Thanks to Dave).

**Example**

```
./catage -master -nslaves 2
```

The master does the minimization. Slave(s) wait for master then split up the Hessian calculation.
How is Open MPI Used in ADMB Now?

The Hessian calculation for standard ADMB models (Thanks to Dave).

Example

./catage -master -nslaves 2

The master does the minimization. Slave(s) wait for master then split up the Hessian calculation.

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
...

Separable Models: With a slight change to the tpl file.

PROCEDURE_SECTION
    int j=0;
    for (int i=1;i<=nh;i++)
    {
        fun(i,j,u(i),log_theta1,beta);
    }

SEPARABLE_FUNCTION void fun( int i,int & j ,const prevariable& ui,
                             const prevariable& log_theta1, const dvar_vector& beta)

    f += 0.9189385 + 0.5*square(ui); // N(0,1) likelihood contribution from u’s
    ...  
    for (ii=1;ii<=nump(i);ii++)
    {
        j++;
        dvariable log_lambda=beta(0)+beta(1)*TRT(j)+beta(2)*CARD(j)+log(gi);
        dvariable lambda=mfexp(log_lambda);
        f += lambda*S(j) - log_lambda;
    }
    ...

How is Open MPI Used in ADMB Now?

Separable Models: With a slight change to the tpl file.

PROCEDURE_SECTION

```cpp
separable_bounds(sb,1,nh);
for (int i=sb->indexmin();i<=sb->indexmax();i++)
{
    int j=0;
    for(int k=1;k<i;k++)
    {
        j+=nump(k);
    }
    fun(i,j,u(i),log_theta1,beta);
}
```

SEPARABLE_FUNCTION void fun( int i,int & j ,const prevariable& ui,
const prevariable& log_theta1, const dvar_vector& beta)

```cpp
f += 0.9189385 + 0.5*square(ui); // N(0,1) likelihood contribution from u's
... 
for (ii=1;ii<=nump(i);ii++)
{
    j++;
    dvariable log_lambda=beta(0)+beta(1)*TRT(j)+beta(2)*CARD(j)+log(gi);
    dvariable lambda=mfexp(log_lambda);
    f += lambda*S(j) - log_lambda;
}
... 
```
What is separable_bounds and what does it do?

separable_bounds(var, lb, ub) is a macro that properly calls ad_separable_manager class.
What is separable_bounds and what does it do?

Separable bounds is a macro that properly calls ad_separable_manager class.

Ad_separable_manager handles the looping over the separable calls.
This is both if we run in parallel or normally.
What is separable_bounds and what does it do?

separable_bounds(var,lb,ub) is a macro that properly calls ad_separable_manager class.

ad_separable_manager handles the looping over the separable calls.
This is both if we run in parallel or normally.

To be added:
Will add a flag so that if you don’t put in separable_bounds you can still use Open MPI features in separable models.
The Laplace Approximation Calculator

```c
int mpi_minimizer_flag=1;
#if defined(USE_ADMPI)
if (ad_comm::mpi_manager)
{
    if (ad_comm::mpi_manager->is_slave())
    {
        mpi_minimizer_flag=0;
    }
}
#endif
...
while (fmc.ireturn>=0)
{
    if (mpi_minimizer_flag)
        fmc.fmin(f,x,g);
    mpi_set_x_f_ireturn(x,f,fmc.ireturn);
    if (fmc.ireturn>0)
    {
        ...
        g=(*lapprox)(x,f,this);
        ...
        if (lapprox->init_switch==0)
        {
            if (f<fmc.fbest)
            {
                lapprox->ubest=lapprox->uhat;
            }
        }
    }
}
```
Why Open MPI?

How to Enable/Access Open MPI in ADMB.

How is Open MPI Used in ADMB Now?

How it Works

Development and Future Directions

The Laplace Approximation Calculator

```c
int mpi_minimizer_flag=1;
#if defined(USE_ADMPI)
if (ad_comm::mpi_manager)
{
    if (ad_comm::mpi_manager->is_slave())
    {
        mpi_minimizer_flag=0;
    }
}
#endif
...
while (fmc.ireturn>=0)
{
    if (mpi_minimizer_flag)
        fmc.fmin(f,x,g);
    mpi_set_x_f_ireturn(x,f,fmc.ireturn);
    if (fmc.ireturn>0)
    {
        ...
        g=(*lapprox)(x,f,this);
        ...
        if (lapprox->init_switch==0)
        {
            if (f<fmc.fbest)
            {
                lapprox->ubest=lapprox->uhat;
            }
        }
    }
}
```
The Laplace Approximation Calculator

Code snippet: at end of call to (*lapprox)(x,f,this)

```c
for (i=1;i<=xadjoint.indexmax();i++)
    xadjoint(i)*=scale1(i);
}
...
#if defined(USE_ADMPI)
if (ad_comm::mpi_manager)
{
    if (ad_comm::mpi_manager->sync_objfun_flag)
    {
        if (ad_comm::mpi_manager->is_master())
        {
            //get dvectors from slaves and add into xadjoint
            for(int si=1;si<=ad_comm::mpi_manager->get_num_slaves();si++)
            {
                dvector slave_xadjoint =
                    ad_comm::mpi_manager->get_dvector_from_slave(si);
                xadjoint+=slave_xadjoint;
            }
        }
        else
        {
            //send dvector to master
            ad_comm::mpi_manager->send_dvector_to_master(xadjoint);
        }
    }
}
#endif
return xadjoint;
```
The Laplace Approximation Calculator

Code snippet: at end of call to (*lapprox)(x,f,this)

```c
for (i=1;i<=xadjoint.indexmax();i++)
    xadjoint(i)*=scale1(i);
}
...
#endif
if defined(USE_ADMPI)
if (ad_comm::mpi_manager)
{
    if (ad_comm::mpi_manager->sync_objfun_flag)
    {
        if (ad_comm::mpi_manager->is_master())
        {
            //get dvectors from slaves and add into xadjoint
            for(int si=1;si<=ad_comm::mpi_manager->get_num_slaves();si++)
            {
                dvector slave_xadjoint =
                    ad_comm::mpi_manager->get_dvector_from_slave(si);
                xadjoint+=slave_xadjoint;
            }
        }
        else
        {
            //send dvectors to master
            ad_comm::mpi_manager->send_dvector_to_master(xadjoint);
        }
    }
}
else
{
    //send dvectors to master
    ad_comm::mpi_manager->send_dvector_to_master(xadjoint);
}
#endif
return xadjoint;
```
## Performance Analysis for nested4 example

### Using `time ./nested -nohess <options>`

<table>
<thead>
<tr>
<th></th>
<th>real time</th>
<th>user time</th>
<th>sys time</th>
<th>-master</th>
<th>-master -nslaves 2</th>
<th>-master &gt; out.txt</th>
<th>-master &gt; out.txt</th>
<th>-master &gt; out.txt</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Katsuo Optimized</strong></td>
<td>0m9.303s</td>
<td>0m9.013s</td>
<td>0m0.256s</td>
<td>0m9.649s</td>
<td>0m13.902s</td>
<td>0m9.264s</td>
<td>0m12.946s</td>
<td>0m13.898s</td>
</tr>
<tr>
<td><strong>Katsuo Debug Mode</strong></td>
<td>0m15.144s</td>
<td>0m14.853s</td>
<td>0m0.284s</td>
<td>0m15.488s</td>
<td>0m15.186s</td>
<td>0m15.118s</td>
<td>0m12.483s</td>
<td>0m15.658s</td>
</tr>
<tr>
<td><strong>My Machine Optimized</strong></td>
<td>0m16.023s</td>
<td>0m15.541s</td>
<td>0m0.408s</td>
<td>0m23.620s</td>
<td>0m36.418s</td>
<td>0m14.841s</td>
<td>0m11.981s</td>
<td>0m0.412s</td>
</tr>
<tr>
<td><strong>My Machine Debug Mode</strong></td>
<td>0m22.828s</td>
<td>0m22.009s</td>
<td>0m0.560s</td>
<td>0m28.453s</td>
<td>0m41.708s</td>
<td>0m22.397s</td>
<td>0m27.965s</td>
<td>0m41.055s</td>
</tr>
<tr>
<td><strong>Dave’s Machine Optimized</strong></td>
<td>0m14.405s</td>
<td>0m14.265s</td>
<td>0m0.132s</td>
<td>0m11.404s</td>
<td>0m11.404s</td>
<td>0m11.404s</td>
<td>0m19.945s</td>
<td>0m1.744s</td>
</tr>
<tr>
<td><strong>Dave’s Machine Debug Mode</strong></td>
<td>0m22.828s</td>
<td>0m22.009s</td>
<td>0m0.560s</td>
<td>0m19.949s</td>
<td>0m21.657s</td>
<td>0m21.657s</td>
<td>0m21.657s</td>
<td>0m21.657s</td>
</tr>
</tbody>
</table>
Where do we go from here?

- Provide support to other operating systems and compilers.
Where do we go from here?

- Provide support to other operating systems and compilers.
- Streamline the configuration/build process so Open MPI can be enabled on all platforms easily.
Where do we go from here?

- 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).
Where do we go from here?

- 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.
Where do we go from here?

- 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.
Investigation of Parallel Processing Using Open MPI

Derek Seiple

ADMB Developers Workshop

March 2012