Parallel programming in Madagascar

Chenlong Wang
Why parallel?

- Time & Money
- Non-local resource
- Parallel Hardware
HPC structure
Outline

Parallel calculation in Madagascar with

- OpenMP
- MPI
- Pscons

Obtain demos

`git clone https://github.com/Chenlonw/MadagascarSummer2017.git`
OpenMp
Installation

OpenMP is a feature of the compiler and its parallel calculation is based on the shared-memory

In Madagascar, you can do parallel programming with the OpenMP either internally or externally

- `#include <omp.h>` plus `-fopenmp`
- `sfomp`
Internal usage

hello.c:

```c
#include <omp.h>
#include <stdio.h>

int main()
{
    #pragma omp parallel
    {
        printf("Hello world from %d of %d\n", omp_get_thread_num(), omp_get_num_threads());
    }
    return 0;
}
```

compiler

gcc hello.c -o hello -fopenmp / icc hello.c -o hello -openmp
The default scoping of variable is **shared** unless specified

- `#pragma omp parallel {}`
- `#pragma omp parallel for`
- Reduction
```c
#include <omp.h>
#include <stdio.h>

int main()
{
    #pragma omp parallel
    {
        #pragma omp master
        {
            printf("Hello world from %d \n", omp_get_thread_num());
        }
        #pragma omp single  // Threads will meet barrier automatically
        {
            printf("Hello world from %d \n", omp_get_thread_num());
        }
        #pragma omp barrier
        printf("Number of threads %d \n", omp_get_num_threads());
    }
}
```

Exercise 1

Code inside a parallel region will be copied to several copies and executed by different threads.

- Only responsible for master thread
- Only be executed by a single thread
- All threads will stop here until the slowest one finished
Exercise 2

```
#include <omp.h>
#include <stdio.h>
#include <math.h>

int main ()
{
    long i;
    long n=100000;

    double X[n],Y[n];

    for (i=0;i<n;i++)
        X[i]=(double)i;

    double wstart = omp_get_wtime();

    #pragma omp parallel for
    for (i=0;i<n;i++)
        Y[i]= sin(X[i])+cos(X[i]);

    printf("Time %f\n",omp_get_wtime()-wstart);
}
```

gcc sin.c -fopenmp -lm -o sin

<table>
<thead>
<tr>
<th>With OpenMP</th>
<th>Time 0.00328</th>
</tr>
</thead>
<tbody>
<tr>
<td>Without OpenMP</td>
<td>Time 0.01896</td>
</tr>
</tbody>
</table>
Exercise 3

\[ \pi = \int_0^1 \frac{4.0}{1 + x^2} \, dx \]

With Reduction  **Time 0.073934**

Without Reduction  **Time 2.564278**

```c
#include <omp.h>
#include <stdio.h>

int main()
{
    long i;
    double x, mypi, sum = 0.0;
    double vpi = 3.141592654;
    double tol = 0.0000001;
    long nsteps = 10000000;
    double step, diff, wend;

    double wstart = omp_get_wtime();

    step = 1.0 / (double) nsteps;

    // #pragma omp parallel for private (x) reduction(+:sum)
    // #pragma omp parallel for private (x)
    for (i = 0; i < nsteps; i++)
    {
        x = (i + 0.5) * step;
        #pragma omp atomic
        sum += 4.0 / (1.0 + x * x);
    }

    mypi = step * sum;
    wend = omp_get_wtime();

    diff = mypi - vpi;
    if (diff < 0) diff *= -1.0;

    if (diff > tol)
        printf("Error in pi: %f\n", mypi);
    else
        printf("PI %1.10f\n", mypi);

    printf("Time: %f\n", wend - wstart);
}
```
In Madagascar

```
grep "pragma omp" $RSFSRC/*/*/M*.c |
awk -F ':' '{ print $1 }' |
uniq
```

139 standalone programs (approximately 11% of Madagascar programs) were using OMP on the last check (2014-02-09)
external usage

If the input data is supposed to be parallel, *sfomp* command splits the data along the given axis and runs it through parallel threads.

```
OpenMP wrapper for embarassingly parallel jobs.

sfomp < in.rsf > out.rsf split=ndim join=axis

<table>
<thead>
<tr>
<th>int</th>
<th>join=axis</th>
<th>axis to join</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>split=ndim</td>
<td>axis to split</td>
</tr>
</tbody>
</table>
```

sfomp sfsMOOTH rect1=5 rect2=5 < in.rsf > out.rsf
Smoothing Mona

Mona Lisa

Smoothened without OpenMP

Smoothened with OpenMP

Mona Lisa

Without OpenMP

With OpenMP
Smoothing Mona

Run smoothing function with parallel threads

\[
\text{Flow('patch','mona','patch w=200,200 p=3,3 | put n3=9 n4=1')}
\]

\[
\text{Flow('patch2','patch','smooth rect1=%d rect2=%d repeat=2' \}
\%
\text{(rect,rect), split=[3,'omp'])}
\]

\[
\text{Flow('pmona2','patch2','put n3=3 n4=3 | patch inv=y weight=y')}
\]

Split the input data along the 3\textsuperscript{rd} axis
MPI
Installation

MPI (Message-Passing Interface) is dominant framework for parallel processing including distributed-memory system. Several implementations (such as Open MPI and MPICH are available)

In Madagascar, you can do parallel programming with the MPI either internally or externally as well
Internal usage

MmpiHelloworld.c:

```c
#include <mpi.h>
#include <rsf.h>
#include <stdio.h>
#include <math.h>

int main(int argc, char *argv[]) {
    int myid, numprocs;
    int namelen;
    char processor_name[MPI_MAX_PROCESSOR_NAME];

    sf_init(argc, argv);

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &myid);
    MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
    MPI_Get_processor_name(processor_name, &namelen);

    fprintf(stderr, "Hello World! Process %d of %d on %s\n", myid, numprocs, processor_name);
    MPI_Finalize();
    exit(0);
}
```

from rsf.proj import *

if not WhereIs('mpirun'):
    sys.stderr.write("\nNo MPI.\n")
    sys.exit(1)

NP = int(ARGUMENTS.get('NP', '4'))

Flow('out', None,
    '''
    sfmpiHelloworld -hostfile=hostfile
    ''', np=NP, stdin=0, stdout=-1)
End()
MPI_COMM_WORLD is a handle points to all the resource

- MPI_Send and MPI_Recv
- MPI_Datatype
Exercise 1

Mpihellocommu.c:

Send a message from master to slave

```c
#include <mpi.h>
#include <rsf.h>
#include <stdio.h>
#include <string.h>
#include <math.h>

int main(int argc, char *argv[])
{
    int myid;
    char message[30];
    MPI_Status mpi_status;

    sf_init(argc, argv);
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &myid);
    if (myid == 0)
    {
        strcpy(message, "Hello processor 1!");
        MPI_Send(message, strlen(message), MPI_CHAR, 1, 99, 
                  MPI_COMM_WORLD);
    }
    else if(myid == 1)
    {
        MPI_Recv(message, 30, MPI_CHAR, 0, 99, 
                  MPI_COMM_WORLD, &mpi_status);
        fprintf(stderr, "received: %s\n", message);
    }

    MPI_Finalize();
    exit(0);
}
```
Exercise 2

Mmpihello.c:

```c
... /* Input vectors in memory */
a = sf_floatalloc (n1);
b = sf_floatalloc (n1);

/* How many vectors per CPU */
nc = (int)(n2/(float)ncpu + 0.5f);
c = sf_floatalloc2 (n1, nc);

/* Starting position in input files */
sf_seek (ain, n1*cpuid*esize, SEEK_CUR);
sf_seek (bin, n1*cpuid*esize, SEEK_CUR);
for (i = cpuid; i < n2; i += ncpu, k++) {
    /* Read local portion of input data */
    sf_floatread (a, n1, ain);
sf_floatread (b, n1, bin);
    /* Parallel summation here */
    for (j = 0; j < n1; j++)
        c[k][j] = a[j] + b[j];
    /* Move on to the next portion */
sf_seek (ain, n1*(ncpu - 1)*esize, SEEK_CUR);
sf_seek (bin, n1*(ncpu - 1)*esize, SEEK_CUR); }

if (0 == cpuid) {
    /* Collect results from all nodes */
    for (i = 0; i < n2; i++) {
        k = i / ncpu; /* Iteration number */
j = i % ncpu; /* CPU number to receive from */
    if (j) /* Receive from non-zero CPU */
        MPI_Recv (&c[k][0], n1, MPI_FLOAT, j, j, 
                MPI_COMM_WORLD, &mpi_stat);
sf_floatwrite (c[k], n1, cout);
    }
    sf_fileclose (cout);
} else {
    /* Send results to CPU #0 */
    for (i = 0; i < k; i++) /* Vector by vector */
        MPI_Send (&c[i][0], n1, MPI_FLOAT, 0, cpuid, 
                MPI_COMM_WORLD);
}
sf_fileclose (ain); sf_fileclose (bin);
MPI_Finalize ();
return 0;
```

A+B=C

Collect results

Splits the data along the slowest axis

Splits the data along the slowest axis
Exercise 2

Mona Lisa

+  

Marmousi

=  

Monamousi
Exercise 2

Mona Lisa + Marmousi = Monamousi
**external usage**

If the input data is supposed to be parallel, `sfmpi` command splits the data along the given axis and runs it through parallel threads.

```
mpirun -np 4 sfmpi split=3 sfsMOOTH rect1=5 rect2=5  input=in.rsf output=out.rsf
```
Smoothing Mona

Mona Lisa

Without MPI

With MPI
Smoothing Mona

Flow('patch','mona','patch w=200,200 p=3,3 | put n3=9 n4=1')

Flow('patch2','patch','smooth rect1=%d rect2=%d repeat=2' \ 
   % (rect,rect), split=[3,'mpi',[0]],reduce='cat',np=4)

Flow('pmona2','patch2','put n3=3 n4=3 | patch inv=y weight=y')

- Run smoothing function with parallel threads
- How many processors involved
- Split the input data along the 3rd axis
- How to reduce (add or cat axis=1)

Madagascar School of Reproducible Computational Geophysics Shanghai 2017
Pscons
Pscons

The SCons can execute the script in parallel by splitting input data or splitting commands.

Unlike the OpenMP or MPI utilities, this has fault tolerance -- in case of a node failing, restarting the job will allow it to complete.

Try

```
pscons / scons -j num
```
Demo1 – splitting input data

```
from rsf.proj import *

Fetch('mona.img','imgs')
Flow('mona','mona.img',
    '''
    echo n1=512 n2=513 in=$SOURCE data_format=native_uchar | dd type=float |
    window n1=512 n2=512 | sftransp | math output="input/1000"
    ''', stdin=0)
Result('mona', 'grey allpos=y title="Mona Lisa" color=b screenratio=1 wantaxis=n'"

Fetch('marmvel.hh','marm')
Flow('mar','marmvel.hh',
    '''
    dd form=native | window j1=1 j2=2 f2=800 n2=512 f1=0 n1=512 |
    scale dscale=0.001 |
    put label1=Depth unit1=km label2=Distance unit2=km | math output="input/4.45"
    '''
)
Result('mar', 'grey allpos=y title="Marmousi" color=b screenratio=1 wantaxis=n'"

Flow('monamousi','mar mona','math b=${SOURCES[1]} output="input+b"', split=[2,256])
Result('monamousi', 'grey allpos=y title="Monamousi" color=b screenratio=1 wantaxis=n'"

End()
```
Demo 1

Mona Lisa + Marmousi = Monamousi
from rsf.proj import *

......

# shot configuration in coordinate
nshot=2
shot0=0
dshot=0.5
Flow('sz',None,'math n1=1 output=\%(sz)g' \%par)
for i in range (1,nshot+1):
    isx = shot0 + (i-1)*dshot
    print isx
Flow('sx-%d'\%(i),None,'math n1=1 output=\%g'\%(isx))
Flow('source-%d'\%(i),'sx-%d sz'\%(i),'cat axis=1 $\{SOURCES[1]\}$')
Flow('erecfield-%d ewavefield-%d'\%(i,i),'elasticwavelet density receivers source-%d ccc'\%(i),
"
    ewefd2d
den=${\{SOURCES[1]\}}
rec=${\{SOURCES[2]\}}
sou=${\{SOURCES[3]\}}
ccc=${\{SOURCES[4]\}}
wfl=${\{TARGETS[1]\}}
dabc=y snap=y verb=y jsnap=\%(jsnap)d jdata=\%(jdata)d
    ssou=y nb=20 nbell=11
"
\%'par)
End()
Demo 2

Two shots  Shot 1  Shot 2
Acknowledge
I would like to acknowledge the HPC group in NTNU gave me lectures on parallel programming (https://www.hpc.ntnu.no/display/hpc/NTNU+HPC+GROUP).

Some of the scripts are modified from the Madagascar official website (http://www.ahay.org/wiki/Parallel_Computing)