Lab #10: The Mandelbrot Set with MPI

PCSE 2015

1 How to Run TAU to make a Trace

1. You need to connect to Stampede with X11 tunneling enabled.
2. You may use `ssh -Y username@stampede.tacc.utexas.edu` where the `-Y` is important.
3. Start an idev session. It must use the development queue! (If you have a reservation for PCSE, you must say NO!)
4. Go to your PCSE repository. Make sure the repository is up to date.
5. Go to `pcse/ps/Labs/10-MPI-mandel/code/c` (or `f` for Fortran folks).
6. Set environment variable `MY_EXE` to your executable code name (`mandel_serial`, `mandel_bulk`, or `mandel_collective`), e.g. `export MY_EXE=mandel_serial`
7. Take a look at the `traceview` Bash script:

```
#!/usr/bin/env bash


if [ -z ${MY_EXE+x} ]; then
  echo "MY_EXE must be set! Exiting"
  exit -1
fi

ml reset
ml tau
make clean
make ${MY_EXE}
make idevrun EXECUTABLE=${MY_EXE}
make tau EXECUTABLE=${MY_EXE}
jumpshot taulog_${MY_EXE}.slog2
```

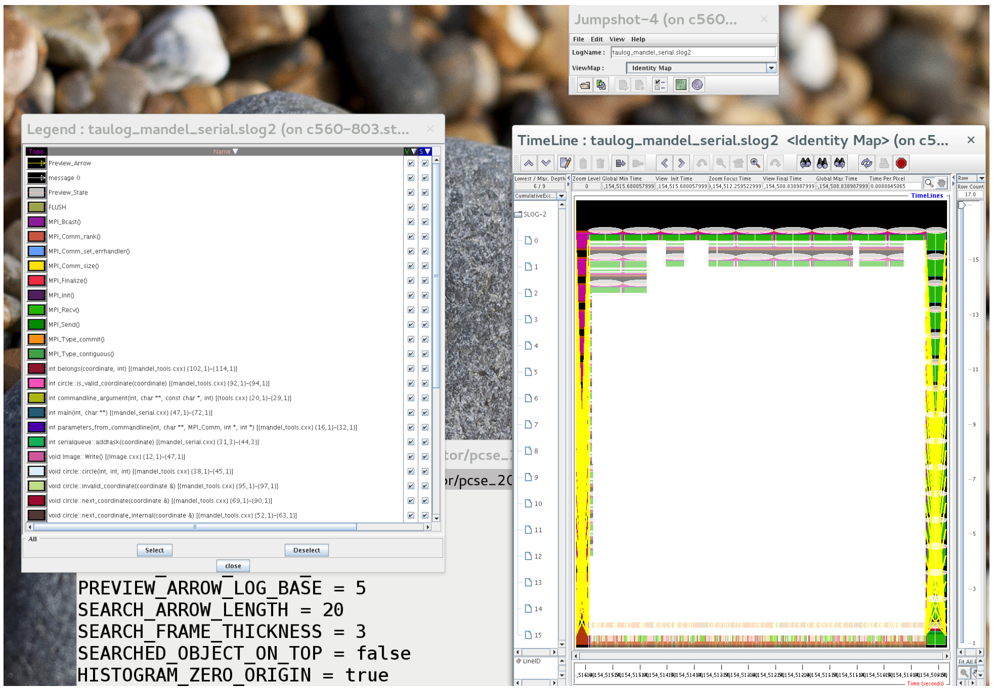
8. We are:
   (a) Making sure `MY_EXE` is set to something
   (b) Resetting modules and then loading TAU
   (c) Issuing a `make clean` in our mandelbrot directory
   (d) Using TAU compiler wrappers to compiler our code with the `make $MY_EXE` command

(e) Running the generated executable using `ibrun` and several TAU environment variables

(f) Postprocessing the TAU output from the created directory to create our `.slog2` Trace output

(g) Running the jumpshot program giving it the `*.slog2` file

9. Now that you have a small idea of what is going on, go ahead and run `./traceview`
10. If everything is set up correctly, you will get a window popping up on your machine:



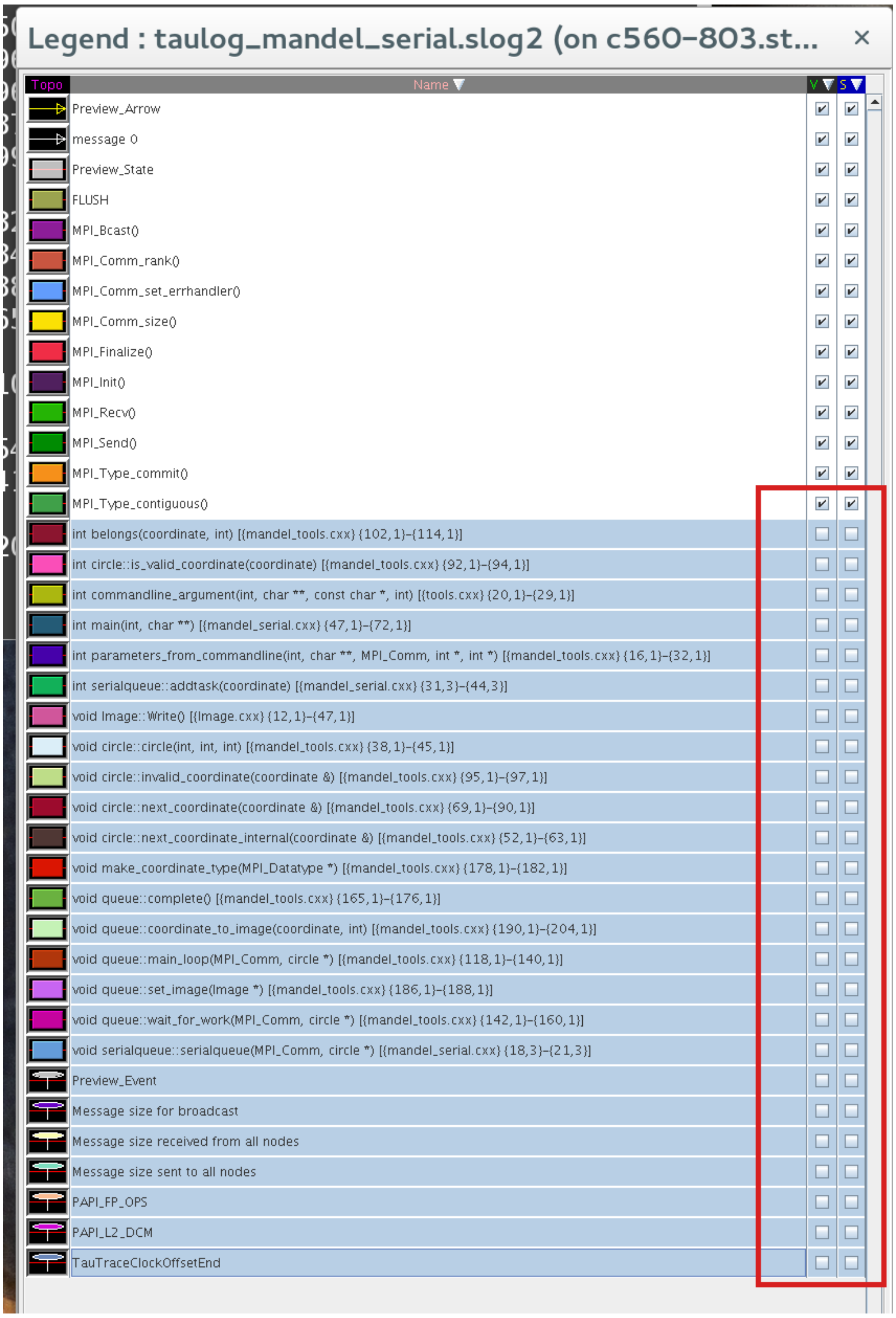
Jumpshot-4 (on c560...
File Edit View Help
LogName : taulog_mandel_serial.slog2
ViewMap : Identity Map
TimeLine : taulog_mandel_serial.slog2 <Identity Map> (on c5...
SLOG-2
TimeLines
Select
Deselect
close

Legend : taulog_mandel_serial.slog2 (on c560-803.st...

Topo	Name	V	S
▷	Preview_Arrow	✓	✓
▷	message 0	✓	✓
▨	Preview_State	✓	✓
▨	FLUSH	✓	✓
🟪	MPI_Bcast()	✓	✓
🟥	MPI_Comm_rank()	✓	✓
🟦	MPI_Comm_set_errhandler()	✓	✓
🟨	MPI_Comm_size()	✓	✓
🟥	MPI_Finalize()	✓	✓
🟪	MPI_Init()	✓	✓
🟩	MPI_Recv()	✓	✓
🟩	MPI_Send()	✓	✓
🟧	MPI_Type_commit()	✓	✓
🟩	MPI_Type_contiguous()	✓	✓
🟥	int belongs(coordinate, int) [{mandel_tools.cxx} {102,1}-{114,1}]	✓	✓
🟪	int circle::is_valid_coordinate(coordinate) [{mandel_tools.cxx} {92,1}-{94,1}]	✓	✓
🟨	int commandline_argument(int, char **, const char *, int) [{tools.cxx} {20,1}-{29,1}]	✓	✓
🟦	int main(int, char **) [{mandel_serial.cxx} {47,1}-{72,1}]	✓	✓
🟪	int parameters_from_commandline(int, char **, MPI_Comm, int *, int *) [{mandel_tools.cxx} {16,1}-{32,1}]	✓	✓
🟩	int serialqueue::addtask(coordinate) [{mandel_serial.cxx} {31,3}-{44,3}]	✓	✓
🟪	void Image::Write() [{Image.cxx} {12,1}-{47,1}]	✓	✓
⬜	void circle::circle(int, int, int) [{mandel_tools.cxx} {38,1}-{45,1}]	✓	✓
🟩	void circle::invalid_coordinate(coordinate &) [{mandel_tools.cxx} {95,1}-{97,1}]	✓	✓
🟥	void circle::next_coordinate(coordinate &) [{mandel_tools.cxx} {69,1}-{90,1}]	✓	✓
🟫	void circle::next_coordinate_internal(coordinate &) [{mandel_tools.cxx} {52,1}-{63,1}]	✓	✓

```
PREVIEW_ARROW_LOG_BASE = 5
SEARCH_ARROW_LENGTH = 20
SEARCH_FRAME_THICKNESS = 3
SEARCHED_OBJECT_ON_TOP = false
HISTOGRAM_ZERO_ORIGIN = true
```

11. Try only displaying only MPI routines. Highlight all other routines on the legend; right-click and choose `toggle selected` for both columns of check boxes:

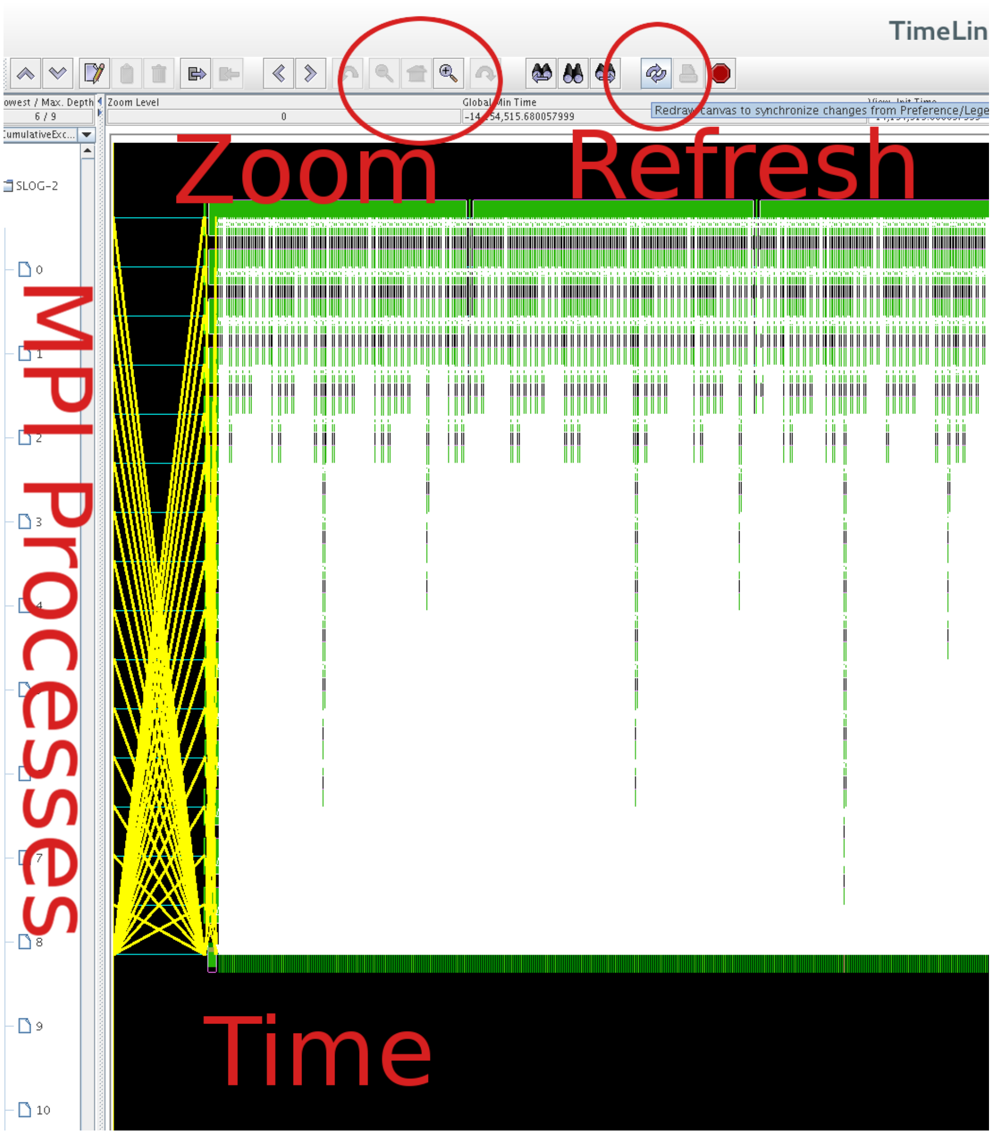


Legend : taulog_mandel_serial.slog2 (on c560-803.st...

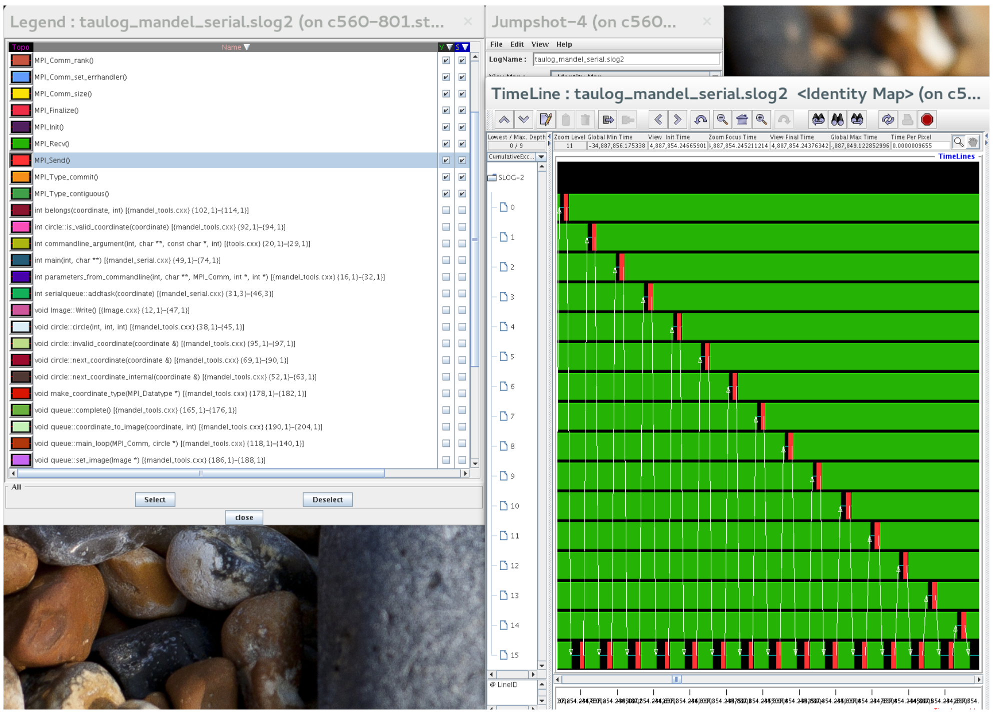
Topo	Name	V	S
→	Preview_Arrow	✓	✓
→	message 0	✓	✓
■	Preview_State	✓	✓
■	FLUSH	✓	✓
■	MPI_Bcast()	✓	✓
■	MPI_Comm_rank()	✓	✓
■	MPI_Comm_set_errhandler()	✓	✓
■	MPI_Comm_size()	✓	✓
■	MPI_Finalize()	✓	✓
■	MPI_Init()	✓	✓
■	MPI_Recv()	✓	✓
■	MPI_Send()	✓	✓
■	MPI_Type_commit()	✓	✓
■	MPI_Type_contiguous()	✓	✓
■	int belongs(coordinate, int) [{mandel_tools.cxx} {102,1}-{114,1}]	☐	☐
■	int circle::is_valid_coordinate(coordinate) [{mandel_tools.cxx} {92,1}-{94,1}]	☐	☐
■	int commandline_argument(int, char **, const char *, int) [{tools.cxx} {20,1}-{29,1}]	☐	☐
■	int main(int, char **) [{mandel_serial.cxx} {47,1}-{72,1}]	☐	☐
■	int parameters_from_commandline(int, char **, MPI_Comm, int *, int *) [{mandel_tools.cxx} {16,1}-{32,1}]	☐	☐
■	int serialqueue::addtask(coordinate) [{mandel_serial.cxx} {31,3}-{44,3}]	☐	☐
■	void Image::Write() [{Image.cxx} {12,1}-{47,1}]	☐	☐
■	void circle::circle(int, int, int) [{mandel_tools.cxx} {38,1}-{45,1}]	☐	☐
■	void circle::invalid_coordinate(coordinate &) [{mandel_tools.cxx} {95,1}-{97,1}]	☐	☐
■	void circle::next_coordinate(coordinate &) [{mandel_tools.cxx} {69,1}-{90,1}]	☐	☐
■	void circle::next_coordinate_internal(coordinate &) [{mandel_tools.cxx} {52,1}-{63,1}]	☐	☐
■	void make_coordinate_type(MPI_Datatype *) [{mandel_tools.cxx} {178,1}-{182,1}]	☐	☐
■	void queue::complete() [{mandel_tools.cxx} {165,1}-{176,1}]	☐	☐
■	void queue::coordinate_to_image(coordinate, int) [{mandel_tools.cxx} {190,1}-{204,1}]	☐	☐
■	void queue::main_loop(MPI_Comm, circle *) [{mandel_tools.cxx} {118,1}-{140,1}]	☐	☐
■	void queue::set_image(Image *) [{mandel_tools.cxx} {186,1}-{188,1}]	☐	☐
■	void queue::wait_for_work(MPI_Comm, circle *) [{mandel_tools.cxx} {142,1}-{160,1}]	☐	☐
■	void serialqueue::serialqueue(MPI_Comm, circle *) [{mandel_serial.cxx} {18,3}-{21,3}]	☐	☐
⊤	Preview_Event	☐	☐
⊤	Message size for broadcast	☐	☐
⊤	Message size received from all nodes	☐	☐
⊤	Message size sent to all nodes	☐	☐
⊤	PAPI_FP_OPS	☐	☐
⊤	PAPI_L2_DCM	☐	☐
⊤	TauTraceClockOffsetEnd	☐	☐

12. In the plot window, the Y-axis represents each MPI process, the X-axis represents time (although TAU's timer isn't currently configured properly). You may

hit the refresh button to update the plot after your legend changes. You may zoom in and out with the zoom menu and also by clicking and dragging with your mouse horizontally in the main window:



13. Zoom in until you can focus on an individual set of MPI send and receive calls. Try updating the color back on the legend menu for `MPI_Send`. See if you can get something like the figure below for `mandel_serial`:


Legend : taulog_mandel_serial.slog2 (on c560-801.st...
Jumpshot-4 (on c560...
TimeLine : taulog_mandel_serial.slog2 <Identity Map> (on c5...
File Edit View Help
LogName : taulog_mandel_serial.slog2
SLOG-2
Select
Deselect
close

Topo	Name	V	S
🟥	MPI_Comm_rank()	✓	✓
🟦	MPI_Comm_set_errhandler()	✓	✓
🟨	MPI_Comm_size()	✓	✓
🟥	MPI_Finalize()	✓	✓
🟪	MPI_Init()	✓	✓
🟩	MPI_Recv()	✓	✓
🟥	MPI_Send()	✓	✓
🟧	MPI_Type_commit()	✓	✓
🟩	MPI_Type_contiguous()	✓	✓
🟥	int belongs(coordinate, int) [{mandel_tools.cxx} {102,1}-{114,1}]	☐	☐
🟪	int circle::is_valid_coordinate(coordinate) [{mandel_tools.cxx} {92,1}-{94,1}]	☐	☐
🟨	int commandline_argument(int, char **, const char *, int) [{tools.cxx} {20,1}-{29,1}]	☐	☐
🟦	int main(int, char **) [{mandel_serial.cxx} {49,1}-{74,1}]	☐	☐
🟪	int parameters_from_commandline(int, char **, MPI_Comm, int *, int *) [{mandel_tools.cxx} {16,1}-{32,1}]	☐	☐
🟩	int serialqueue::addtask(coordinate) [{mandel_serial.cxx} {31,3}-{46,3}]	☐	☐
🟪	void Image::Write() [{Image.cxx} {12,1}-{47,1}]	☐	☐
⬜	void circle::circle(int, int, int) [{mandel_tools.cxx} {38,1}-{45,1}]	☐	☐
🟩	void circle::invalid_coordinate(coordinate &) [{mandel_tools.cxx} {95,1}-{97,1}]	☐	☐
🟥	void circle::next_coordinate(coordinate &) [{mandel_tools.cxx} {69,1}-{90,1}]	☐	☐
🟫	void circle::next_coordinate_internal(coordinate &) [{mandel_tools.cxx} {52,1}-{63,1}]	☐	☐
🟥	void make_coordinate_type(MPI_Datatype *) [{mandel_tools.cxx} {178,1}-{182,1}]	☐	☐
🟩	void queue::complete() [{mandel_tools.cxx} {165,1}-{176,1}]	☐	☐
🟩	void queue::coordinate_to_image(coordinate, int) [{mandel_tools.cxx} {190,1}-{204,1}]	☐	☐
🟧	void queue::main_loop(MPI_Comm, circle *) [{mandel_tools.cxx} {118,1}-{140,1}]	☐	☐
🟪	void queue::set_image(Image *) [{mandel_tools.cxx} {186,1}-{188,1}]	☐	☐

14. For `mandel_serial`, we can clearly see the "serialization" of the send and recv's as each process has posted a recv and is waiting for some work.
15. Apply this procedure to generate traces for your versions of the code described in later sections.

2 Mandelbrot set

NOTE: See Section 8.3 of the textbook for more information on this assignment.

If you've never heard the name Mandelbrot set, you probably recognize the picture. Its formal definition is as follows:

> A point c in the complex plane is part of the Mandelbrot set if the series x_n defined by
>
> $$\{x_0 = 0 x_{n+1} = x_n^2 + c$$
>
> satisfies
>
> $$\forall_n : |x_n| \leq 2.$$

It is easy to see that only points c in the bounding circle $|c| < 2$ qualify, but apart from that it's hard to say much without a lot more thinking. Or computing; and that's what we're going to do.

In this set of exercises you are going to take an example program `mandel_serial.cxx` and extend it to use a variety of MPI programming constructs. This program has been set up as a master-worker model: there is one master processor (for a change this is the last processor, rather than zero) which gives out work to, and accepts results from, the worker processors. It then takes the results and construct an image file from them.

2.1 Invocation

The `mandel_serial` program is called as

```
ibrun -np 123 mandel_serial steps 456 iters 789
```

where the `steps` parameter indicates how many steps in x, y direction there are in the image, and `iters` gives the maximum number of iterations in the `belong` test.

If you forget the parameter, you can call the program with

```
mandel_serial -h
```

and it will print out the usage information.

2.2 Tools

The driver part of the Mandelbrot program is simple. There is a circle object that can generate coordinates

```
class circle {
 public :
  circle(double stp,int bound);
  void next_coordinate(struct coordinate& xy);
  int is_valid_coordinate(struct coordinate xy);
  void invalid_coordinate(struct coordinate& xy);
```

and a global routine that tests whether a coordinate is in the set, at least up to an iteration bound. It returns zero if the series from the given starting point has not diverged, or the iteration number in which it diverged if it did so.

```
int belongs(struct coordinate xy,int itbound) {
  double x=xy.x, y=xy.y; int it;
  for (it=0; it<itbound; it++) {
    double xx,yy;
    xx = x*x - y*y + xy.x;
    yy = 2*x*y + xy.y;
    x = xx; y = yy;
    if (x*x+y*y>4.) {
      return it;
    }
  }
  return 0;
}
```

In the former case, the point could be in the Mandelbrot set, and we colour it black, in the latter case we give it a colour depending on the iteration number.

```
  if (iteration==0)
    memset(colour,0,3*sizeof(float));
  else {
    float rfloat = ((float) iteration) / workcircle->infty;
    colour[0] = rfloat;
    colour[1] = max((float)0,(float)(1-2*rfloat));
    colour[2] = max((float)0,(float)(2*(rfloat-.5)));
  }
```

We use a fairly simple code for the worker processes: they execute a loop in which they wait for input, process it, return the result.

```
void queue::wait_for_work(MPI_Comm comm,circle *workcircle) {
  MPI_Status status; int ntids;
  MPI_Comm_size(comm,&ntids);
  int stop = 0;

  while (!stop) {
    struct coordinate xy;
    int res;

    MPI_Recv(&xy,2,MPI_DOUBLE,ntids-1,0, comm,&status);
    stop = !workcircle->is_valid_coordinate(xy);
    if (stop) res = 0;
    else {
      res = belongs(xy,workcircle->infty);
    }
    MPI_Send(&res,1,MPI_INT,ntids-1,0, comm);
  }
  return;
}
```

A very simple solution using blocking sends on the master is given:

```
class serialqueue : public queue {
private :
  int free_processor;
public :
  serialqueue(MPI_Comm queue_comm,circle *workcircle)
    : queue(queue_comm,workcircle) {
    free_processor=0;
  };
  /* Send a coordinate to a free processor;
     if the coordinate is invalid, this should stop the process;
     otherwise add the result to the image.
  */
  void addtask(struct coordinate xy) {
    MPI_Status status; int contribution;

    MPI_Send(&xy,2,MPI_DOUBLE,
             free_processor,0,comm);
    MPI_Recv(&contribution,1,MPI_INT,
             free_processor,0,comm, &status);
    if (workcircle->is_valid_coordinate(xy)) {
      coordinate_to_image(xy,contribution);
      total_tasks++;
    }
    free_processor++;
    if (free_processor==ntids-1)
      // wrap around to the first again
      free_processor = 0;
  };
```

Explain why this solution is very inefficient. Make a trace of its execution that bears this out.

2.3 Bulk task scheduling

The previous section showed a very inefficient solution, but that was mostly intended to set up the code base. If all tasks take about the same amount of time, you can give each process a task, and then wait on them all to finish. A first way to do this is with non-blocking sends.

Code a solution where you give a task to all worker processes using non-blocking sends and receives, and then wait for these tasks with `MPI_Waitall` to finish before you give a new round of data to all workers. Make a trace of the execution of this and report on the total time.

You can do this by writing a new class that inherits from `queue`, and that provides its own `addtask` method:

```
class bulkqueue : public queue {
public :
  bulkqueue(MPI_Comm queue_comm,circle *workcircle)
    : queue(queue_comm,workcircle) {
```

You will also have to override the `complete` method: when the circle object indicates that all coordinates have been generated, not all workers will be busy, so you need to supply the proper `MPI_Waitall` call.

2.4 Collective task scheduling

Another implementation of the bulk scheduling of the previous section would be through using collectives.

Code a solution which uses scatter to distribute data to the worker tasks, and gather to collect the results. Is this solution more or less efficient than the previous?