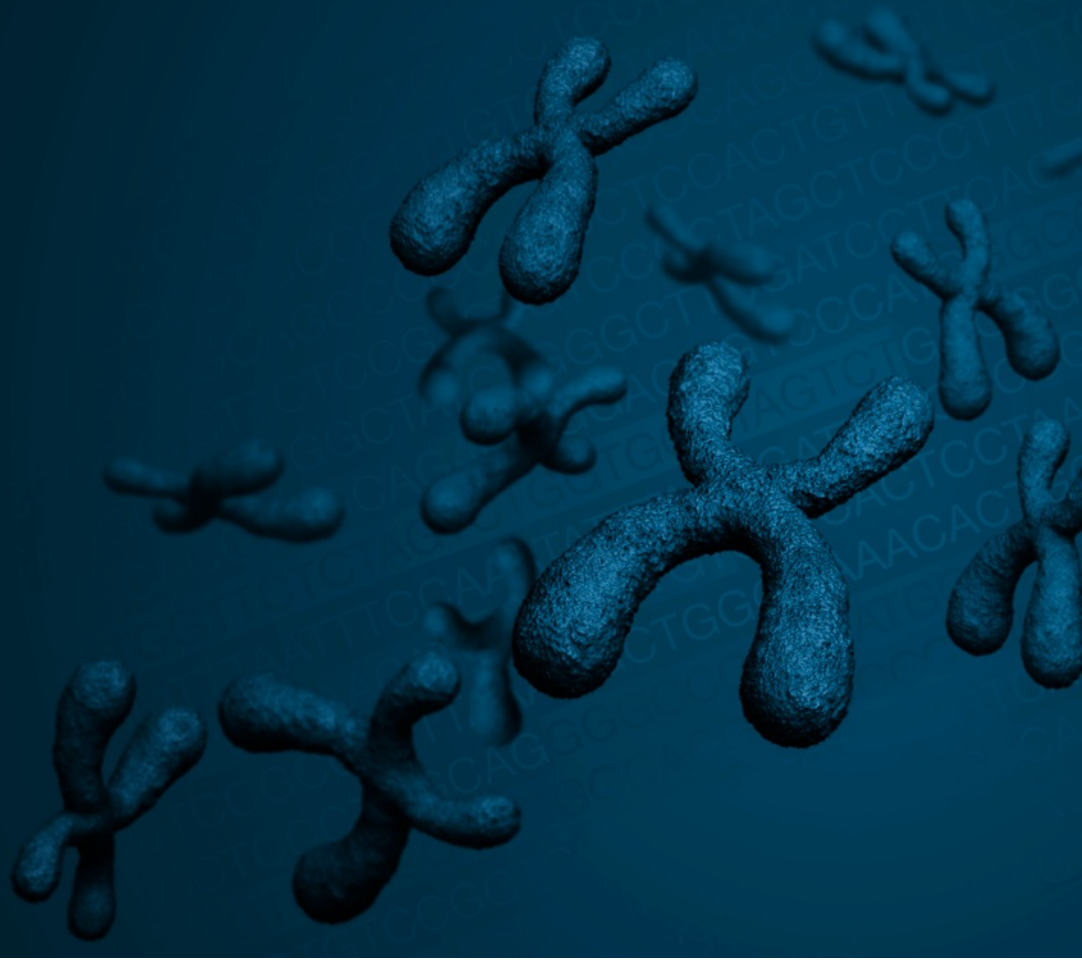
MPI/multi-threaded software in eHive/LSF



Ensembl retreat 2018

Mateus Patricio

EMBL-EBI

Ensembl

MPI – Message Passing Interface

- MPI interface
  - Provides synchronization between processes
- Implementations
  - MPICH
  - Open MPI (do not confuse it with OpenMP)
  - Always run your code with the same implementation used to compile it.
  - Binders available for Perl, Python, R, Ruby, Java …
- Why?
  - Parallel computing

Please clone this for the session material

- https://github.com/mateuspatricio/ensembl-retreat-2018
- Our material is coded in C++
- We will also be writing some eHive pipeline_configs to run our MPI code on the farm

Building blocks of the MPI interface

- Send and Receive are the basic operations of MPI
  - Just like addition and subtraction in mathematics

- Process P1 needs to send data to process P2:
  1. P1 packs all the data into a buffer
  2. The communication device routes the message to the desired location
  3. P2 acknowledges the delivery

MPI_Send(...

one-2-one

```
MPI_Send(
    void* data, // data buffer
    int count, // amount of data to be sent
    MPI_Datatype datatype, // MPI_INT, etc
    int destination, // rank
    int tag,    // used to control deliveries
    MPI_Comm communicator)
```

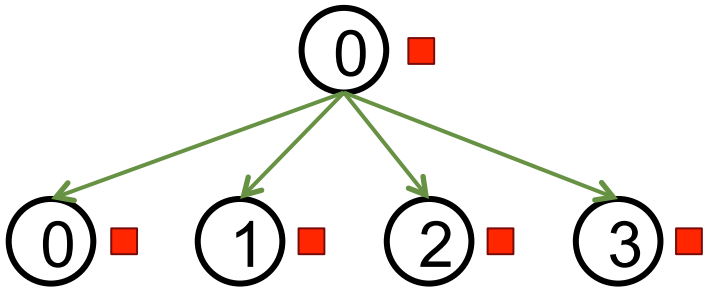
MPI_Recv(...

one-2-one

```
MPI_Recv(
    void* data, // data buffer
    int count, //amount of data to be received
    MPI_Datatype datatype, // MPI_INT, etc
    int source, // origin of data
    int tag,
    MPI_Comm communicator, // MPI_COMM_WORLD
    MPI_Status* status)
```

MPI_Bcast(...

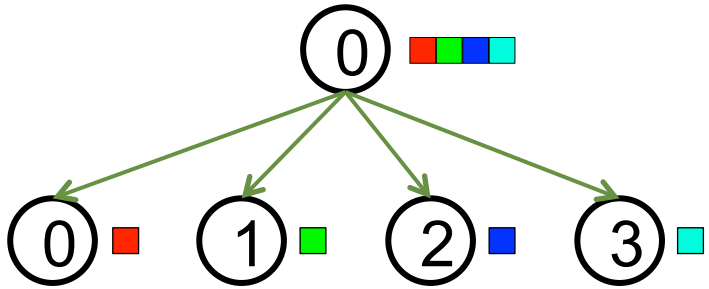
one-2-many




```
MPI_Scatter(
    void* buffer,
    int count,
    MPI_Datatype send_datatype,
    int root, //rank of broadcaster
    MPI_Comm communicator)
```

MPI_Scatter(...

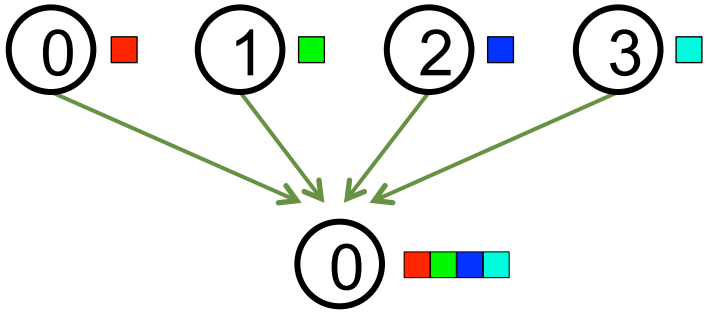
one-2-many




```
MPI_Scatter(
    void* send_data,
    int count,
    MPI_Datatype send_datatype,
    void* recv_data,
    int recv_count,
    MPI_Datatype recv_datatype,
    int root,
    MPI_Comm communicator)
```

MPI_Gather(...

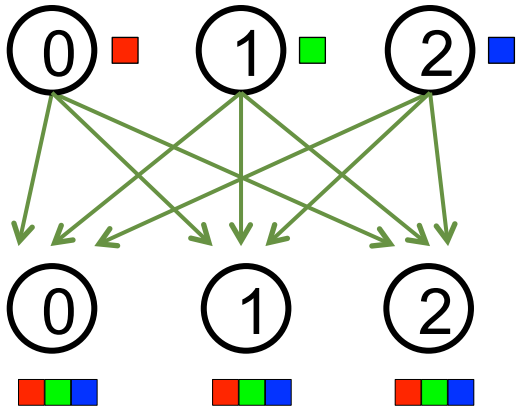
many-2-one




```
MPI_Gather(
    void* send_data,
    int send_count,
    MPI_Datatype send_datatype,
    void* recv_data,
    int recv_count,
    MPI_Datatype recv_datatype,
    int root,
    MPI_Comm communicator)
```

MPI_Allgather(...

many-2-many




```
MPI_Allgather(
    void* send_data,
    int send_count,
    MPI_Datatype send_datatype,
    void* recv_data,
    int recv_count,
    MPI_Datatype recv_datatype,
    MPI_Comm communicator)
```

Our example

- Define an array of integers
- Populate it with integers (i)

```
int* data = new int[array_size];
    long expected_sum = 0;
    for(int i=0; i<array_size; i++) {
        data[i] = i;
    }
```

Send

- Send different parts of the array to each client processes

```
for (int destination_id=1; destination_id<numprocs; destination_id++){
    MPI_Send(&chunksize, 1, MPI_INT, destination_id, tag_chunksize,
MPI_COMM_WORLD);

    MPI_Send(&data[offset], chunksize, MPI_INT, destination_id,
tag_data, MPI_COMM_WORLD);

    offset = offset + chunksize;
}
```

Receive

- Receive the size of the data
- Receive the data

```
MPI_Recv(&chunksize, 1, MPI_INT, SERVER_NODE, tag_chunksize,
MPI_COMM_WORLD, &status);

int* data = new int[chunksize];

MPI_Recv(data, chunksize, MPI_INT, SERVER_NODE, tag_data,
MPI_COMM_WORLD, &status);
```

Reducing results from workers

```
MPI_Reduce(&mysum, &sum, 1, MPI_LONG, MPI_SUM, SERVER_NODE,
MPI_COMM_WORLD);
```

```
MPI_Reduce(
    void* send_data,
    void* recv_data,
    int count,
    MPI_Datatype datatype,
    MPI_Op op,
    int root,
    MPI_Comm communicator)
```

Compiling

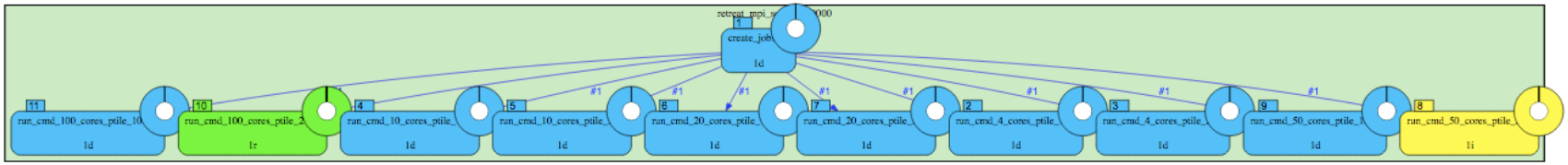
- We will use the Intel C++ compiler
  - Much better performance

- We want to use the MPI implementation supplied by EBI
  - For that we need to prepare our environment
    - Follow up the recipe provided in the git repository (`# "Unload" basic.sh`)
    - Load the MPICH module

- mpic++ -o my_mpi_code my_mpi_code.cc

eHive pipeline

```
init_pipeline.pl retreatMpiSession_conf.pm -pipeline_name retreat_mpi_session_500
```



This is how your running MPI job looks like:

```
JOBID   USER    STAT  QUEUE      FROM_HOST   EXEC_HOST   JOB_NAME   SUBMIT_TIME
8574908 mateus  RUN   mpi-rh7    ebi-cli-001 1*ebi6-124  *1_mpi-2_1 Jun 13 15:34
                                               1*ebi6-128
                                               1*ebi6-130
                                               1*ebi6-126
                                               1*ebi6-033
                                               1*ebi6-102
                                               1*ebi6-029
                                               1*ebi6-115
                                               1*ebi6-022
                                               1*ebi6-122
                                               1*ebi6-082
                                               1*ebi6-044
                                               1*ebi6-006
                                               1*ebi6-055
                                               1*ebi6-175
                                               1*ebi6-067
                                               1*ebi6-144
                                               1*ebi6-053
                                               1*ebi6-138
                                               1*ebi6-089
```

Querying the results

logic_name	input_id	runtime_msec	cpu_sec	lifespan_sec	mem_megs
run_cmd_100_cores_ptile_10	{"array_size" => 500}	5068	46.8285	5.20989	46.4414
run_cmd_20_cores_ptile_1	{"array_size" => 500}	5737	5.2582	5.97581	46.3867
run_cmd_10_cores_ptile_2	{"array_size" => 500}	7906	24.6924	8.13759	46.3789
run_cmd_50_cores_ptile_2	{"array_size" => 500}	7566	10.6468	8.37856	46.3867
run_cmd_20_cores_ptile_5	{"array_size" => 500}	11425	75.6551	11.5699	46.3984
run_cmd_10_cores_ptile_1	{"array_size" => 500}	13493	8.80207	13.6313	46.3906
run_cmd_4_cores_ptile_2	{"array_size" => 500}	13537	26.9854	13.6894	46.3984
run_cmd_50_cores_ptile_10	{"array_size" => 500}	17238	163.227	17.4366	46.3789
run_cmd_4_cores_ptile_1	{"array_size" => 500}	50185	43.7933	50.3312	46.3906

MySQL over MPI

- Compile with MySQL support (within ensembl environment):
  - mpic++ -o distribute_array_mysql -L/usr/include/mysql -lmysqlclient -I/usr/include/mysql distribute_array_mysql.cc

- Now testing it:
  - mpirun -n 4 /homes/mateus/mpi/ensembl-retreat-2018/distribute_array_mysql 10

Questions

Special thanks to Matthieu Muffato

EMBL-EBI

e!Ensembl