# Modules to teach parallel computing using Python and the LittleFe Cluster

José R. Ortiz-Ubarri, Rafael A. Arce-Nazario
jose.ortiz@hpcf.upr.edu, rafael.arce@upr.edu
Computer Science Department - University of Puerto Rico Rio Piedras Campus

## Abstract

The ability to design effective solutions using parallel processing should be a required competency for every computing student. However, teaching parallel concepts is sometimes challenging and costly. For such reasons here we present a set of modules to teach parallel computing paradigms using python MPI4py and disco in the LittleFe educational Cluster.

## Introduction

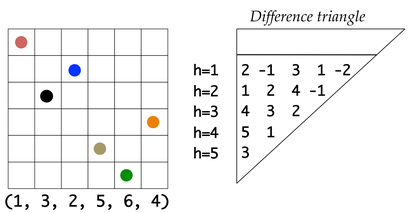
Hands-on programming exercises and code demonstrations/experiments are an essential educational resource for teaching introductory parallel computing concepts. The design of effective and applicable exercises can be challenging to the instructor, especially since many libraries or frameworks for parallel programming require intricate system setups and training students in low-level details to get even the most basic programs to work. In our experiences, low-level details in parallel computation exercises lead, among other things, to student (and instructor) frustration and deter the students from focusing on the essential concepts.

```python
comm = MPI.COMM_WORLD
num_procs = comm.Get_size()
rank = comm.Get_rank()
stat = MPI.Status()

msg = "Hello world, proc %s !",
      % rank

if rank == 0:
    # Master work
    print msg
    for i in range(num_procs - 1):
        msg = comm.recv
          (source=MPI.ANY_SOURCE,
           tag=MPI.ANY_TAG, status=stat)
        print msg

else:
    # Worker work
    comm.send(msg, dest = 0)
```

```c
int main (int argc, char* argv[]){

  const int max_msg_length = 100;
  char message[max_msg_length+1];
  MPI_Status status ;
  int rank, num_procs ;
  int tag ;
  int mpi_error ;

  mpi_error = MPI_Init(&argc, &argv);
  mpi_error = MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  mpi_error = MPI_Comm_size(MPI_COMM_WORLD,
                            &num_procs);

  message = sprintf(message,
                    "Hello world, proc %d!", rank);

  if(rank == 0){
    printf(message) ;
    for(int i = 0; i< num_procs-1; i++){
      mpi_error = MPI_Recv(message, max_msg_length + 1,
                           MPI_CHAR, MPI_ANY_SOURCE,
                           tag, MPI_COMM_WORLD,  &status);
      printf(message);
    }
  }
  else{
    mpi_error_code = MPI_Send( message,
                    strlen(message) + 1,
                    MPI_CHAR, 0, tag, MPI_COMM_WORLD);
  }

  mpi_error = MPI_Finalize();
}
```

## MPI Module 1: Parallel enumeration of Costas arrays

The Costas array enumeration problem consists in finding permutations of size N that meet the distinct difference property: for all integers h, i, and j, with $1 \le h \le n-1$, and $1 \le i,j \le n-h$, $f(i + h) - f(i) = f(j + h) - f(j)$ implies $i = j$.







A naïve approach to find all the Costas arrays of size N, is to generate all N! permutations. The search space is reduced by using a backtracking algorithm that continues searching a computational branch as long as it maintains the Costas property or stop searching in the computational branch otherwise (See Figure 1). But even this appoach grows factorially with N.

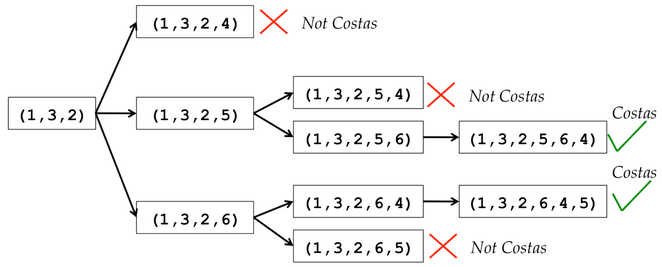



**Figure 1.** Computational branch of Costas arrays of size 6 that start with sub permutation (1,3,2).

We use the Costas array enumeration problem to introduce the Master / Worker paradigm using mpi4py. As illustrated in Figure 2, the master is in charge of generating sub permutations, and the slaves complete the search of Costas arrays with size N that start with the subpermutation sent by the master. Results are sent back to the master.

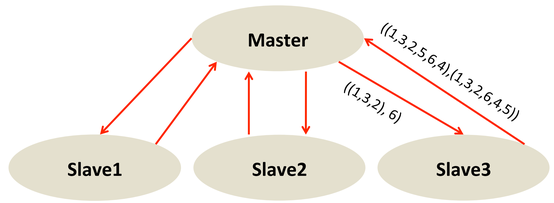



**Figure 2.** Master/worker diagram of Costas problem. Master sends sub problems to slaves. Slaves return the solution to the sub problems.

## MPI Module 2: Parallel password cracking

Password cracking is a method by which an intruder that gains access to an encrypted passwords file can deduce the original (clear text) passwords using a dictionary and brute force. Each word in a dictionary is encrypted and compared to the encrypted passwords in the password file.

We use the password cracking example to teach how to solve problems by dividing the problem in sub problems of approximately the same size among the available compute nodes. In our approach, the complete password file is sent to all the compute nodes (MPI broadcast), and the dictionary file is divided in (approximately) equal parts to the compute nodes.

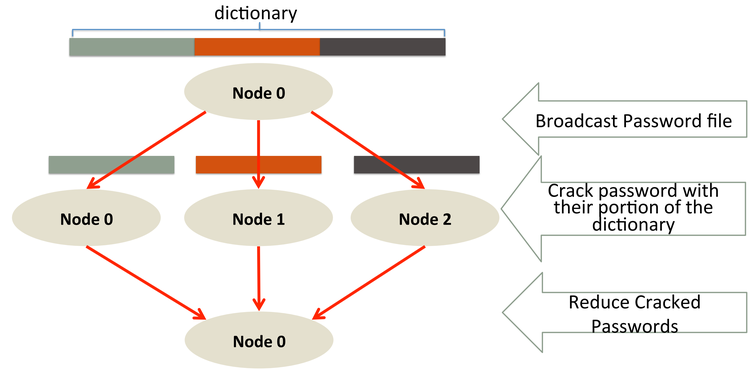



**Figure 3.** Map/Reduce diagram of the password cracking problem. Node 0 broadcasts the password file to all nodes. The dictionary is divided in approximately the same size for each node. Creacked passwords are sent back to the master.

## MapReduce Module 1: Network Traffic

The aggregation of traffic data per host helps to identify anomalies in a computer network such as (1) an IP in use that is not delegated, (2) an IP that is generating more traffic than normal, and (3) an IP that is not generating traffic. NetFlow is a network protocol that aggregates connection information from one host to another over a certain period of time (flow). NetFlow files consist of large quantities of connection information such as IP addresses of the source and destination hosts, and the traffic in each connection (e.g. 5 minutes of UPR network traffic can be as high as 6.5MB, or 363,149 lines of flows). Thus, processing them takes considerable processing effort.

In this module, we use NetFlow data processing as an example of an application that can be accelerated using Disco MapReduce. In the sample code below: (1) the source and destination IPs are mapped by their addresses and valued by their traffic octets (2) the reducer gathers the sorted key-values and computes their aggregated sums.

```python
def map(line, params):
    #Split the data into an array
    data = line.split()
    yield data[0], int(data[5])
    yield data[1], int(data[5])
```

```python
def reduce(line, params):
    for ip, traffic in kvgroup(sorted(iter)):
        yield ip, sum(traffic)
```

## Acknowledgements

We thank the NSF-CPATH award CCF-0938995 for supporting this work.

## References

R. Arce, J. Ortiz, “Enumeration of Costas Arrays in FPGAs and GPUs”. International Conference on ReConFigurable Computing and FPGAs, Cancun, Mexico, 2011.
Python, www.python.org
MPI4py, http://mpi4py.scipy.org/
Disco Project, http://discoproject.org/
Wikipedia, http://en.wikipedia.org/wiki/MapReduce
NetFlows, http://www.ietf.org/rfc/rfc3954.txt