Modules to teach parallel computing using Python and the LittleFe Cluster

[Extended Abstract]

José Ortiz-Ubarri
Computer Science Department
University of Puerto Rico, Rio Piedras
jose.ortiz@hpcf.upr.edu

Rafael Arce-Nazario
Computer Science Department
University of Puerto Rico, Rio Piedras
rafael.arce@upr.edu

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.

1. 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.

Besides preparing modules for learning MPI4py and for setting up the LittleFe cluster with Disco [2], we also have designed a set of easy–to–deploy hands–on educational modules for the LittleFe educational Cluster, which can be used to introduce students to concepts such as: the map–and–reduce and master–slave models, load distribution, and scaling. The programming exercises use the Python MPI4py and Disco (map–reduce) libraries to significantly deemphasize attention to low–level details and allow time to complete solutions to more elaborate problems. Each module is constructed around a real–world or research problem to showcase the importance of parallel computing and to motivate the students. Our growing list of modules includes: two modules for teaching and practicing MPI distributed memory programming concepts and a module to practice MapReduce basics.

2. MODULE 1: LEARNING MPI DISTRIBUTED MEMORY PROGRAMMING THROUGH COSTAS ARRAYS

The first module uses the enumeration of Costas arrays to introduce the Master/Worker paradigm using MPI4py. The classic (non–parallel) solution to this problem uses a backtracking algorithm to list all permutations that meet the Costas property [1]. As expected, the time complexity of such a solution grows factorially with the size of the sequence. A parallel solution is easy to explain and implement: let the master compute all permutations up to a given position, then distribute these sub–permutations to be completed independently by the workers. Figure 1 shows the search of Costas sequences of size 6. The initial permutations represent a small portion of the total computation, thus parallelization achieves almost linear speedup.

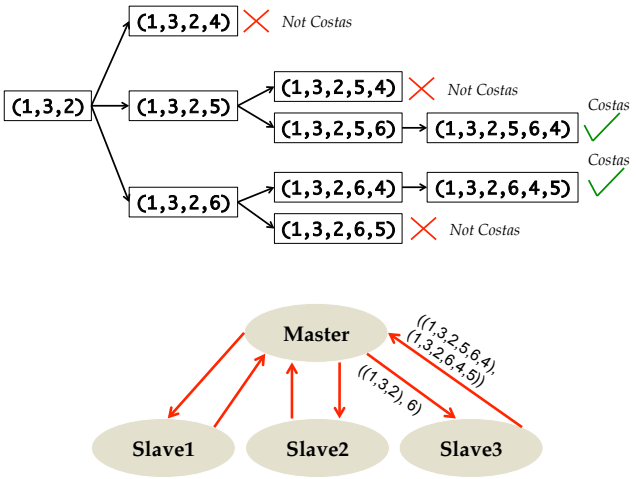

Figure 1: (a) Computation of Costas arrays using backtracking, e.g., computational branch of the Costas arrays of size 6 that starts with subpermutation (1,3,2). (b) Master/worker diagram of Costas problem. Master sends subproblems to slaves. Slaves return the solution to the subproblems.

3. MODULE 2: LEARNING MPI DISTRIBUTED MEMORY PROGRAMMING THROUGH PASSWORD CRACKING

The second module uses *password cracking* as a the motivating application. 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, e.g., 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. Figure 2 illustrates 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 and distributed among the compute nodes. The exercise can be enhanced by having students search for more complex password schemes.

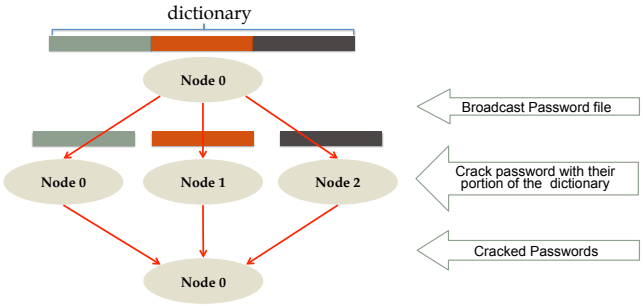

Figure 2: Map/Reduce diagram of the password cracking problem. Node 0 broadcasts the password file to all nodes. The dictionary is divided in approximately the same size for each node. Cracked passwords are sent back to the master.

Another module to teach how to solve problems by dividing the problem in subproblems of approximately the same size was developed based in the Latin Squares problem. The Latin Squares enumeration problem is similar to the Costas enumeration problem discussed in Section 2, in that both are constraint satisfaction problems that can be solved using backtracking. The main difference is that in the Latin Squares problem a mater/slave solution is not necessary due the distribution of solutions throught the search space. Both modules are presented to students to develop a discussion on when to choose one method over the other.

4. MODULE 3: LEARNING MAPREDUCE BASICS THROUGH NETFLOW DATA ANALYSIS

The motivating application for this module is network data processing. In computer networks, aggregation of traffic data per host helps to identify anomalies 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 [3] 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. Simplified code for this application is presented in Figure 3, wherein (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.

```
def map(line, params):
  # Split the data into an array:
  # data[0] is the source, data[1] is destination
  # data[5] is the octet

  data = line.split()
  yield data[0], int(data[5])
  yield data[1], int(data[5])


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

Figure 3: Map and reduce functions for basic NetFlow data processing

5. CONCLUSIONS AND FUTURE WORK

In this work we presented some modules that are used to introduce students to parallel computing concepts with Python libraries that significantly deemphasize attention to low–level inter-process communication details and allow time to complete solutions to more elaborate problems. We hope to expand the list of modules to introduce parallel computing concepts with problems that are of interest or application to different STEM areas.

6. ACKNOWLEDGEMENTS

We thank the NSF-CPATH award CCF-0938995 for supporting this work. The educational modules discussed in this work were developed as part of the requirements for the recipients of the LittleFe educational computer cluster.

7. REFERENCES

[1] R. A. Arce-Nazario and J. R. Ortiz-Ubarri. Enumeration of costas arrays using GPUs and FPGAs. In *Reconfigurable Computing and FPGAs (ReConFig), International Conference*, pages 462–467. IEEE, 2011.

[2] S. Papadimitriou and J. Sun. Disco: Distributed co–clustering with map–reduce. ICDM, 2008.

[3] V. Valluri, M. Djernaes, G. Sadasivan, and B. Claise. Cisco systems netflow services export version 9. http://www.ietf.org/rfc/rfc3954.txt.