Institution of Information and Technology

University of Dhaka

A Parallel Clustering Algorithm with KnuthMorrisPratt (KMP) Algorithm

Author:
Chinmoy Acharjee
Rafiul Islam Ratul

Supervisor:
Dr. B M Mainul Hossain

April 15, 2018

1 Introduction

In computer science, string searching is one of the most common problems where it tries to find whether a string is a sub-string of another particular string or not. There are several algorithms to solve this problem. These algorithms are mostly used in text edit processing, image processing, literature retrieval, natural language recognition, spell checking, WWW search engines, computer virus signature matching, data compression, DNA sequence matching and many other fields.

The most common string searching algorithm is Knuth–Morris–Pratt (KMP) algorithm. It is a prefix based searching algorithm faster than normal string searching algorithm like 'Naive String Searching Algorithm'. The algorithm uses degenerating property (pattern having same sub-patterns appearing more than once in the pattern) of the pattern and improves the worst case complexity to O(n).

We can make this algorithm faster if we make it parallelly executable. As it is linear time complexity algorithm, it will consume more time if the text data is big. So our goal is to make this algorithm more efficient. If we run the KMP algorithm on multiple computers to handle larger data, it will be less time-consuming. By doing that, it can be used in large-scale applications.

To achieve parallelism in KMP algorithm, we will use MPI clustering programming. MPI is a message-passing model of distributed memory. This paper will show how KMP algorithm runs on multiple computers or multiple processors simultaneously using MPI clustering. This paper presents a simulation and statistical analysis on KMP both sequentially and parallelly.

2 Related Works

1. KMP Algorithm: This algorithm was conceived by Donald Knuth and Vaughan Pratt and independently by James H. Morris in 1977. Knuth, Morris and Pratt discovered first linear time string-matching algorithm by analysis of the naïve algorithm. It keeps the information that naive approach wasted gathered during the scan of the text. By avoiding this waste of information, it achieves a running time of O(m + n). The implementation of Knuth-Morris-Pratt algorithm is efficient because it minimizes the total number of comparisons of the pattern against the input string. The prefix-function-

- preprocesses the pattern to find matches of prefixes of the pattern with the pattern itself.
- is defined as the size of the largest prefix of P[0..j 1] that is also a suffix of P[1..j].
- also indicates how much of the last comparison can be reused if it fails.
- It enables avoiding backtracking on the string 'S'.

2. Concept of MPI: The Message Passing Interface Standard (MPI) is a message passing library standard based on the consensus of the MPI Forum, which has over 40 participating organizations, including vendors, researchers, software library developers, and users. The goal of the Message Passing Interface is to establish a portable, efficient, and flexible standard for message passing that will be widely used for writing message passing programs. MPI is the first standardized, vendor independent, message passing library. The advantages of developing message passing software using MPI closely match the design goals of portability, efficiency, and flexibility. MPI is not an IEEE or ISO standard, but has in fact, become the "industry standard" for writing message passing programs on HPC platforms.

MPI (Message Passing Interface) is a specification for the developers and users of message passing libraries. By itself, it is not a library but rather the specification of what such a library should be. MPI primarily addresses the message-passing parallel programming model where data is moved from the address space of one process to that of another process through cooperative operations on each process.

Reasons for Using MPI: Standardization - MPI is the only message passing library that can be considered a standard. It is supported on virtually all HPC platforms. Practically, it has replaced all previous message passing libraries.

Portability - There is little or no need to modify the source code when the application is ported to a different platform that supports (and is compliant with) the MPI standard.

Performance Opportunities - Vendor implementations should be able to exploit native hardware features to optimize performance. Any implementation is free to develop optimized algorithms.

Functionality - There are over 430 routines defined in MPI-3, which includes the majority of those in MPI-2 and MPI-1.

Availability - A variety of implementations are available, both vendor and public domain.

The standards of MPI are as follows:

- Point-to-point communication
- Collective operations
- Process groups
- Communication contexts
- Process topologies
- Bindings for FORTRAN 77 and C
- Environmental management and inquiry

- Profiling interface

3. MPI Funtions: Although MPI is a complex and multifaceted system, a wide range of problems can be solved using just six of its functions.

Here are some basic funnctions are given below:
MPI_INIT : Initiate an MPI computation.

MPI_FINALIZE : Terminate a computation.

MPI_COMM_SIZE : Determine number of processes.

MPI_COMM_RANK : Determine my process identifier.

MPI_SEND : Send a message.

MPI_RECV : Receive a message.

3. Message Passing Process of MPI: MPI is a mainly established to divide work among some computers and minimize the calculation or computing time of data operation. If we divide the tasks among some computers then one computer need to communicate with other computer to give data and receive data. For this reason, MPI is based on the message passing, whose function is to exchange information, coordinate and control the implementation steps with the definition of program grammar and semantics in the core library by sending messages between the concurrent execution parts.

All MPI program include "mpi.h" header, by including this header in a program we can use all library function which we need to write a MPI code. We initialize all process by MPI_Init() function and after finishing tasks we terminate all process by MPI_Finalize() function.

3 Refereces

http://cs.indstate.edu/ kmandumula/presentation.pdf
https://www.ics.uci.edu/ eppstein/161/960227.html