문제 3: Boyer-Moore 알고리즘 기반의 거대 문자열 패턴검색 최적 병렬화 (학부: 40점, 대학원: 30점)

문제 개요 및 Boyer-Moore 알고리즘 설명

문자열 패턴 검색 알고리즘은 검색 대상문자열 (Target) 에서 패턴문자열 (Pattern) 을 검색하는 알고리즘으로서, 디스크 전체의 파일에서 특정 바이러스 패턴을 검색해야하는 Virus search engine, 생명과학분야의 유전자 패턴검색, 실시간으로 들어오는 네트워크의 패킷을 분석하는 네트워크 보안, Linux의 'grep' 명령어, 문서편집 프로그램의 '찾기' 기능 등 여러분야에서 활용되고 있다. 다양한 문자열 패턴검색 알고리즘 중 Boyer-Moore (BM) 알고리즘은 고속 문자열 패턴검색 알고리즘으로 다음과 같은 과정을 통해 진행된다.

(1) 검색 대상문자열 (Target) 과 검색할 패턴문자열 (Pattern) 을 입력받는다.
(2) 전처리과정을 통해 입력받은 Pattern을 분석하여 두 가지의 패턴 이동규칙 (Shift rules)- bad-character shift, good-suffix shift - 을 계산한다.
(3) Target의 문자와 Pattern의 문자를 비교하여 불일치시 과정 (1)에서 미리 계산한 패턴 이동규칙을 이용해 패턴을 이동시키며 패턴검색을 진행한다.
(4) Target과 Pattern의 문자비교는 Pattern의 오른쪽에서 왼쪽으로, Pattern의 이동은 Target의 왼쪽에서 오른쪽으로 진행된다.

검색과정의 예)

- Target:

G	C	A	T	C	G	C	A	G	A	G	A	G	T	A	T	A	C	A	G	T	A	C	G

- Pattern:

G	C	A	G	A	G	A	G

- Shift rules (전처리 결과):

bad-character shift (BCS):

character	A	C	G	other characters
shift	1	6	2	8

good-suffix shift (GSS):

pattern index	0	1	2	3	4	5	6	7
shift	7	7	7	2	7	4	7	1

- 문자 불일치시

  MAX([BCS[Target character]-비교완료한 글자수], GSS[Pattern index]) 만큼 패턴 이동

- 패턴 발견시

  GSS[0] 만큼 패턴 이동

- 패턴 이동 과정

1

G	C	A	T	C	G	C	A	G	A	G	A	G	T	A	T	A	C	A	G	T	A	C	G
G	C	A	G	A	G	A	G																

1. Target[A] != Pattern[G]

➔ Pattern 1 이동 (BCS[A] == GSS[7] → 1)

2

G	C	A	T	C	G	C	A	G	A	G	A	G	T	A	T	A	C	A	G	T	A	C	G
	G	C	A	G	A	G	A	G															

1. Target[G] == Pattern[G]
2. Target[A] == Pattern[A]
3. Target[C] != Pattern[G]

➔ Pattern 4 이동 (BCS[C]-2 == GSS[5] → 4)

3

G	C	A	T	C	G	C	A	G	A	G	A	G	T	A	T	A	C	A	G	T	A	C	G
					G	C	A	G	A	G	A	G											

1. Target[G] == Pattern[G]
2. Target[A] == Pattern[A]
3. Target[G] == Pattern[G]
4. Target[A] == Pattern[A]
5. Target[G] == Pattern[G]
6. Target[A] == Pattern[A]
7. Target[C] == Pattern[C]
8. Target[G] == Pattern[G]

➔ Pattern 발견

➔ Pattern 7 이동 (GSS[0] → 7)

4

G	C	A	T	C	G	C	A	G	A	G	A	G	T	A	T	A	C	A	G	T	A	C	G
												G	C	A	G	A	G	A	G				

1. Target[G] == Pattern[G]
2. Target[A] == Pattern[A]
3. Target[C] == Pattern[G]

➔ Pattern 4 이동 (BCS[C]-2 == GSS[5] → 4)

5

G	C	A	T	C	G	C	A	G	A	G	A	G	T	A	T	A	C	A	G	T	A	C	G
																G	C	A	G	A	G	A	G

1. Target[G] == Pattern[G]
2. Target[C] == Pattern[A]

➔ Pattern 7 이동 (BCS[C]-1 < GSS[6] → 7)

➔ 검색 종료

문제 설명

본 문제에서는 Boyer-Moore 알고리즘을 이용한 거대 문자열의 패턴검색을 병렬화한다. Target은 지정된 거대 문자열 파일(9,082,011,648자, 약 8.46GB ASCII text file)을 사용하며 Target 파일의 경로와 Pattern은 프로그램 실행 시 매개변수로 전달한다. 수행시간 측정구간에서는 Target 파일을 메모리로 로드하는 시간(File I/O 시간)과 전처리 과정 시간 (BSS 및 GSS 계산시간)은 포함하지 않으며 검색수행시간과 병렬화에 필요한 추가적인 처리시간만을 측정한다.

Main.c(.f90)는 Boyer-Moore 알고리즘을 이용하여 문자열을 검색하는 과정을 구현한 코드이다. 해당 코드에는 기본적인 MPI 코드 구조가 구현되어있으나 병렬화되지 않은 순차코드이다. Main.c(.f90) 코드에서 패턴검색은 Boyer-Moore.c(.f90)의 do_search(...) 함수를 호출하여 수행한다. Main.c(.f90)의 일정부분을 수정하여 병렬화를 완성하고 그에 따른 성능향상을 보여라.

병렬화 Hint: 본 문제의 병렬화과정은 거대 문자열을 분할하여 rank들에게 할당하고 각 rank들은 할당된 문자열을 독립적으로 동시에 검색하는 방식으로 진행한다.

최적화 Hint: Boyer-Moore 알고리즘의 특성상 그 수행시간은 문자열의 길이보다 문자열의 내용에 의존적이다. 따라서 모든 rank들이 같은 길이로 나누어진 문자열을 할당받아 검색을 수행하여도 나누어진 문자열의 내용은 모두 다르기 때문에 rank간의 검색수행시간에 차이가 발생하게 된다. 즉 rank간의 load unbalance가 발생하게 되어 작업을 마치고 다른 rank를 기다리면서 컴퓨팅 자원을 낭비하는 유휴 rank가 생기기 쉽고, 이는 컴퓨팅 자원 활용율의 감소로 이어진다. 따라서 이를 최소화하기 위한 성능최적화 방안을 고민해보자.

참고 및 유의 사항

1. 아래의 사항들을 위반하거나 조건을 만족하지 못하는 경우 0점 처리한다.
2. 병렬코드의 수행결과 (패턴을 찾은 회수) 가 주어진 순차코드의 결과와 정확히 일치하지 않을 경우 정확성 미달로 성능을 인정하지 않는다.
3. 병렬코드의 성능은 답안 제출 이후 미리 정해놓은 패턴(비공개)을 입력하여 측정한다.
4. 순차코드의 수정 불가한 부분이나 주어진 알고리즘을 수정하는 행위, 다른 문자열 패턴 검색 알고리즘을 사용하거나 다른 공개코드를 사용하는 행위는 금지한다.
5. 병렬화를 진행하면서 발생되는 추가적인 모든 작업들 (변수선언, 메모리 할당/해제, 통신시간 등)은 반드시 시간 측정 구간 안에서 수행한다 (순차 코드 참고).
6. 학부의 경우 1노드 20코어, 대학원의 경우 4노드 32코어를 모두 사용한 결과로만 채점한다.
7. Boyer-Moore.c 파일 및 Makefile 은 수정할 수 없다.

8. SIMD나 어셈블리 레벨의 최적화는 허용하지 않으며 모두 일반적인 코드로 작성되어야 한다.
9. OpenMP, pThread 등을 활용한 Thread 레벨의 병렬화는 일체 허용하지 않는다.
10. 제출한 코드의 성능이 순차코드의 성능보다 나쁠 경우 결격처리한다.
11. 기타 불분명한 사항은 Staff에 문의한다.