Написать программу для расшифровки зашифрованного текста, используя MPI и OpenMP.

Получаемые программой параметры (command line arguments):

1) Длина ключа шифрования (количество битов ключа)
2) Файл, содержащий зашифрованный текст
3) Файл, содержащий предполагаемые слова в тексте (каждое слова с новой строки)

Данные аргументы обрабатываются рут процессом (нулевым) MPI. Третий аргумент является необязательным. В случае, если третий аргумент не получен, программа должна использовать по умолчанию файл, содержащий большое количество английских слов (описание дальше).

Результатом работы программы будет ключ шифрования и расшифрованный текст. Оба они будут выведены как standard output.

Метод шифрования:

Ключ шифрования является серией битов, длинной k. Незашифрованный текст (plaintext) делится на подстроки длинной k. Затем выполняется функция XOR для каждой под-стоки с ключом шифрования. Результат является зашифрованным текстом (ciphertext).

Для простоты можно предположить, что k равен 4-м или кратно 8-ми. То есть k = 4, 8, 16, …

Что бы расшифровать текст, выполняется похожий процесс: зашифрованный текст делится на подстроки длинной k и затем выполняется оператор XOR с ключом шифрования.

Пример:

Ключ шифрования 1001 и текст “no cigar” (без кавычек):

Предположим, что текст представлен в формате ASCII. В коде ASCII каждый символ отображается с помощью 7 битов с добавлением восьмого бита слева (в представленном примере самый левый бит в кодировке каждого символа является нулем. В некоторых средах этот бит будет битом четности).

В нашем примере это будет выглядеть следующим образом (используем числа в формате HEX для удобства):

```
plain text:      no  cigar .
ascii:        6E 6F 20 63 69 67 61 72 2E
```

Что бы зашифровать, выполним функцию XOR с 1001 (9 в шестнадцатеричном отображении)

```
plaintext:    6E  6F  20  63  69  67  61  72  2E
key:          99  99  99  99  99  99  99  99  99
              ============================
ciphertext:    F7  F6  B9  FA  F0  FE  F8  EB B7
```

Как взломать код?

Воспользуемся методом “brute force”:

Идея заключается в попытках расшифровать данный зашифрованный текст с помощью каждого из возможных ключей длинной k (программа получает значение k как один из параметров). Если получаемый результат выглядит как правильный английский текст, значит мы нашли подходящий ключ. Нужно будет дать определение тому, что является “правильным английским тестом”. Идея заключается в том, что текст является правильным, если в нем присутствуют предполагаемые\ожидаемые слова. Например, если известно, что тема текста – экономика, то предположительно в нем будут присутствовать такие слова как: “bank”, “budget”, “dollars”, “unemployment”. Файл, содержащий предполагаемые слова, будет дан программе как третий аргумент (command line argument). Так как это аргумент является необязательным, по умолчанию нужно использовать файл, содержащий большое количество слов на английском (подразумевая общий текст на английском). В Линукс можно найти такой файл (https://en.wikipedia.org/wiki/Words_(Unix)).

В случае, если группа “предполагаемых” слов мала, то можно просто проверить находится ли данное слово в этой группе. Если же группа велика, предпочтительно использовать структуру данных (например hash table), позволяющую выполнить проверку на наличие более оптимально и быстро. Предположим, что первая строка в файле предполагаемых слов включает в семя несколько предполагаемых слов.

Если длинна ключа k, то возможны 2 в степени k различных ключей. Важно обратить внимание, что если k является большим числом, то использовать данный метод не практично и не реально, потому что может понадобиться слишком много времени для решения данной задачи. Поэтому, при разработке программы, используйте короткие ключи.

В качестве оптимизации программы, можно добавить, что если программа находит подходящий ключ, чтоб она не продолжала перебирать оставшиеся варианты ключей.

Сдача проекта:

Код должен включать в себя комментарии и пояснения. Так же должно быть приложено объяснение (полстраницы-страница) использования и разделения работы между MPI, OpenMP, CUDA. Объяснение как проверяются “предполагаемые слова” (вероятное использование библиотечных функций).