UJIAN TENGAH SEMESTER

PEMROSESAN DATA TERDISTRIBUSI

IF655-D

Dikerjakan oleh :

Andre Budiman

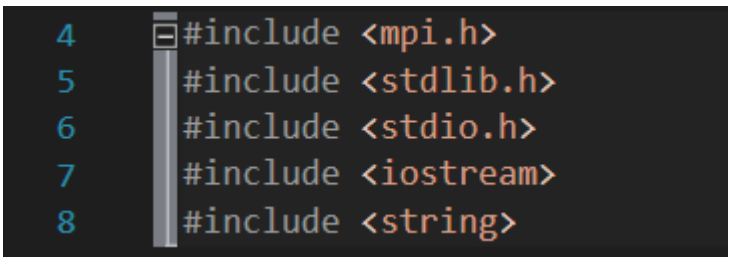
00000032851

UNIVERSITAS MULTIMEDIA NUSANTARA
TANGERANG
2020

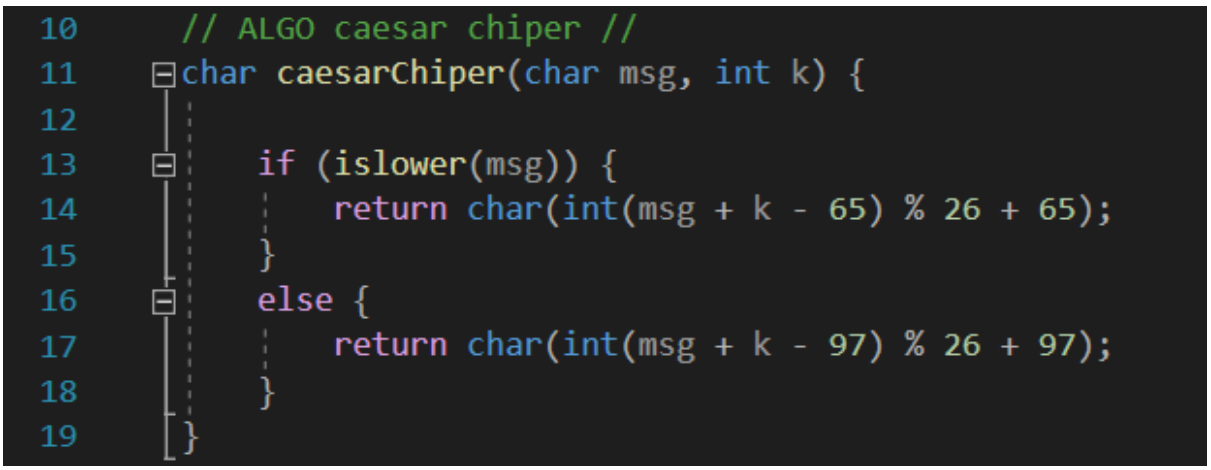
1. Buatlah sebuah program sederhana (tidak lebih dari 500 *lines of code*) yang menyelesaikan sebuah permasalahan *computer science* dengan memanfaatkan pemrosesan / komputasi paralel **[50]**. (CPMK 3-7)

POTONGAN SOURCE CODE & EKSEKUSI PROGRAM MENGGUNAKAN ALGORITMA CAESAR CIPHER UNTUK PARALLEL PROGRAMMING

A. Library yang dipakai

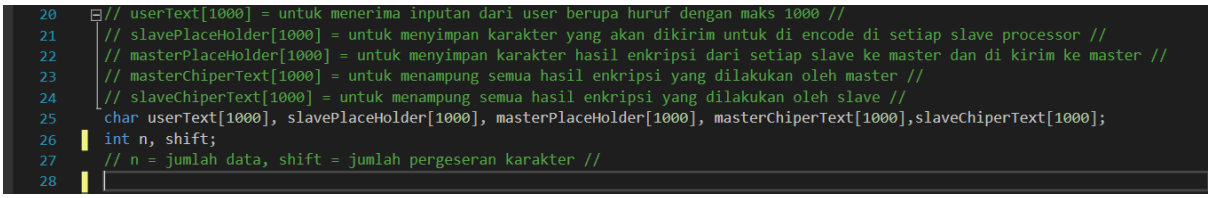


B. Algoritma Caesar Cipher



Algoritma ini akan menerima 2 paramter, yaitu: “msg” dan “k”. “msg” akan berfungsi untuk menerima character yang akan di enkripsi oleh sistem, dan “k” adalah jumlah pergeseran yang akan dilakukan pada karakter yang telah diinput.

C. Inisialisasi Variabel yang akan dipanggil



D. Program Utama

```
30  //Program Utama
31  int main(int argc, char* argv[])
32  {
33      // RANK = Process ID || SIZE = Banyak Process
34      int rank, size, elementsPerProcess, n_elements_recieve, receiveSlave;
35      double elapsed_time;
36
37      MPI_Status stat;
38
39      // inisialisasi MPI (step awal)
40      MPI_Init(&argc, &argv);
41
42      // mencari proses ID dan berapa banyak proses yang dimulai
43      MPI_Comm_rank(MPI_COMM_WORLD, &rank);
44      MPI_Comm_size(MPI_COMM_WORLD, &size);
45      elapsed_time = -MPI_Wtime();
46
```

```
47      // process utama
48      if (rank == 0) {
49          do {
50              std::cout << "Enter Words: "; // input userText //
51              std::cin >> userText;
52              fflush(stdout);
53              std::cout << "Enter the number of shifts : "; // input shift //
54              std::cin >> shift;
55              fflush(stdout);
56
57              n = strlen(userText); // mengukur panjang userText //
58              elementsPerProcess = n / size; // memecah task yang harus dikerjakan per processor //
59
60              // error handling untuk jumlah karakter kurang dari jumlah processor //
61              if (n < size) {
62                  printf("The number of characters is less than the number of processors, please re-enter the word\n\n\n");
63              }
64          } while (n < size);
65      }
66
67      // broadcast ke semua processor ketika terjadi pergeseran karakter//
68      MPI_Bcast(&shift, 1, MPI_INT, 0, MPI_COMM_WORLD);
69
70      // broadcast panjang n (userText) ke semua processor //
71      MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
```

E. Core Process

```
73    //CORE PROCESS / MASTER PROCESS
74    if (rank == 0) {
75        int idx, x;
76        if (size > 1) {
77            // membagi task ke slave
78            for (x = 1; x < size - 1; x++) {
79                idx = x * elementsPerProcess;
80                // komunikasi antara master ke slave
81                MPI_Send(&elementsPerProcess, 1, MPI_INT, x, 0, MPI_COMM_WORLD);
82                MPI_Send((void*)&userText[idx], elementsPerProcess, MPI_CHAR, x, 0, MPI_COMM_WORLD);
83            }
84
85            // task yang harus dikerjakan oleh slave processor //
86            idx = x * elementsPerProcess;
87            int elements_left = n - idx;
88
89            // komunikasi antara master ke slave
90            MPI_Send(&elements_left, 1, MPI_INT,x, 0, MPI_COMM_WORLD);
91            MPI_Send((void*)&userText[idx], elements_left, MPI_CHAR, x, 0, MPI_COMM_WORLD);
92        }
93
94        // enkripsi master process
95        for (x = 0; x < elementsPerProcess; x++) {
96            printf("processor %d have the data of %c\n", rank, userText[x]);
97            masterChiperText[x] = caesarChiper(userText[x], shift);
98        }
99        printf("master chiper : %s", masterChiperText);
100   }
```

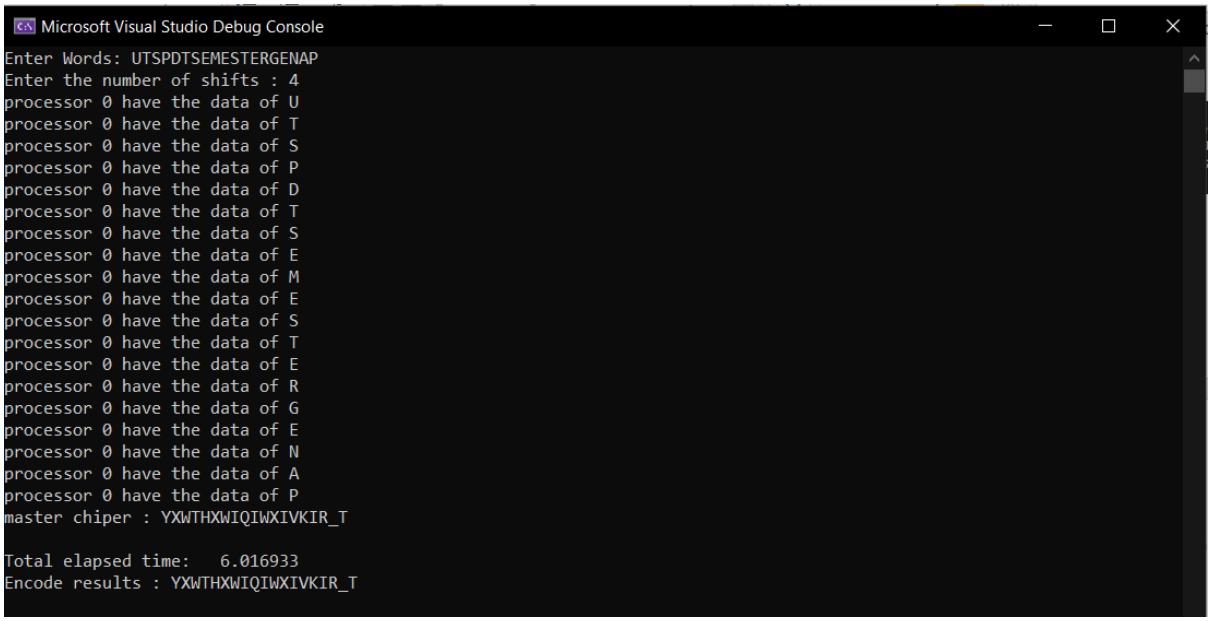
F. Slave Process

```
102   // slave process
103   else {
104       // mendapat kiriman dari master processor
105       MPI_Recv(&n_elements_recieve, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, &stat);
106       MPI_Recv(&slavePlaceHolder, n_elements_recieve, MPI_CHAR, 0, 0, MPI_COMM_WORLD, &stat);
107
108       // enkripsi yang dilakukan oleh slave untuk setiap task yang di dapat"
109       for (int x = 0; x < n_elements_recieve; x++) {
110           printf("processor %d have the data of %c\n", rank, slavePlaceHolder[x]);
111           slaveChiperText[x] = caesarChiper(slavePlaceHolder[x], shift);
112       }
113
114       // mengirim hasil enkripsi dari slave ke master //
115       MPI_Send((void*)&slaveChiperText, n_elements_recieve, MPI_CHAR, 0, 99, MPI_COMM_WORLD);
116       MPI_Send(&n_elements_recieve, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
117       printf("slave chiper : %s\n", slaveChiperText);
118   }
```

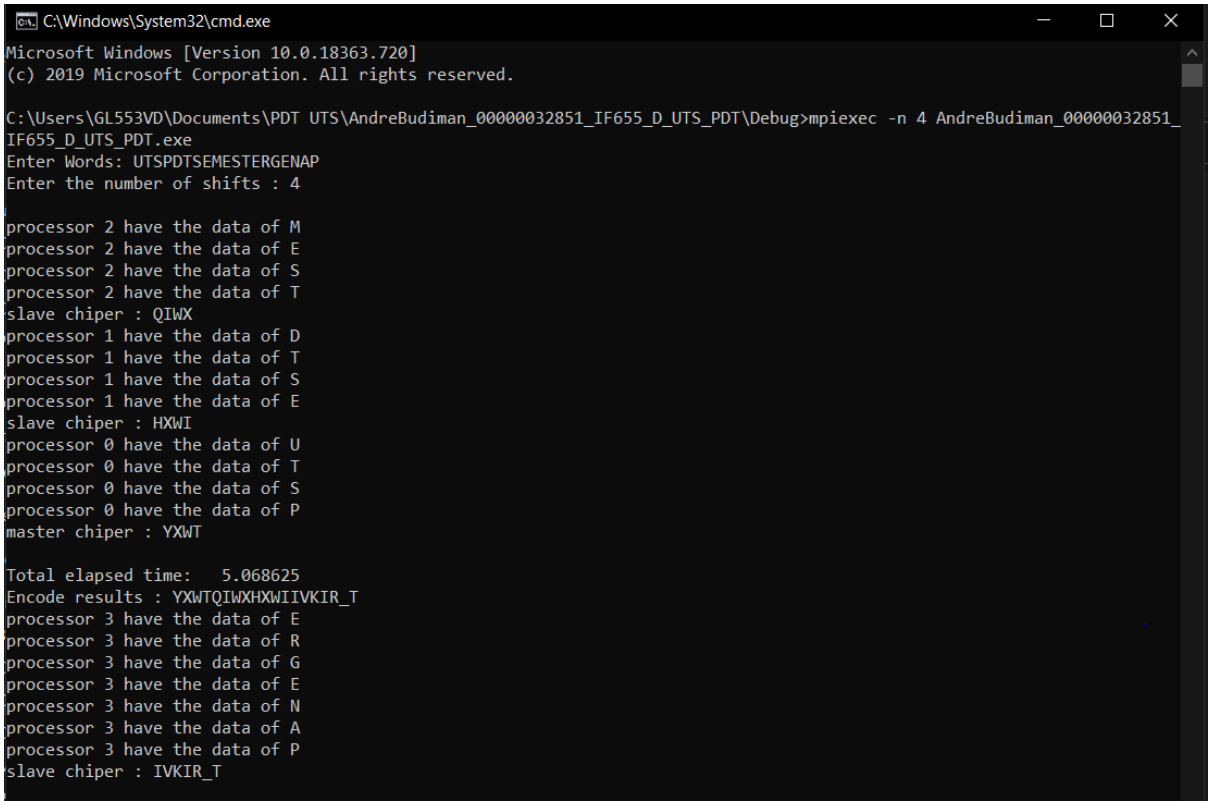
G. Output Elapsed Time sampai dengan End Code

```
120   // print waktu yang dibutuhkan sistem //
121   if (rank == 0) {
122       elapsed_time += MPI_Wtime();
123       printf("\n\nTotal elapsed time: %10.6f\n", elapsed_time);
124
125       // menerima semua enkripsi dari setiap slave processor //
126       for (int x = 1; x < size; x++) {
127           MPI_Recv(&receiveSlave, 1, MPI_INT, MPI_ANY_SOURCE, 0, MPI_COMM_WORLD, &stat);
128           MPI_Recv(&masterPlaceHolder, receiveSlave, MPI_CHAR, MPI_ANY_SOURCE, 99, MPI_COMM_WORLD, &stat);
129
130           // memisahkan hasil enkripsi yang dihasilkan oleh slave process //
131           strcat_s(masterChiperText, sizeof(masterPlaceHolder), masterPlaceHolder);
132
133       }
134       // menggabungkan semua hasil enkripsi (slave + master processor) //
135       printf("Encode results : %s \n", masterChiperText);
136   }
137   // membersihkan semua status MPI sebelum keluar dari process
138   MPI_Finalize();
139   return 0;
140   }
```

H. Eksekusi Program (1 Processor)



I. Eksekusi Program (4 Processor)



Kesimpulan yang dapat diambil adalah semakin banyak processor yang bekerja, waktu yang dibutuhkan oleh sistem untuk mengerjakan suatu task akan semakin cepat, dapat dilihat perbedaan elapsed time yang dimiliki oleh sistem dengan 1 processor dan 4 processor untuk memproses kata **"UTSPDTSEMESTERGENAP",** karena dengan adanya 4 processor, task akan dipecah dan dikirimkan ke slave process.

2. Buatlah rancangan metodologi desain Foster untuk program tersebut (*partitioning, communication, agglomeration, mapping*) **[50]**. (CPMK 1-2)

A. Partitioning

Partitioning merupakan **Domain Decomposition**. Proses utama (master) akan membaca inputan userText dan jumlah task yang diinput oleh user, jumlah task ini berpengaruh terhadap berapa banyak task yang harus dikerjakan oleh sistem (sebanyak n), n sendiri adalah panjang kata yang akan di enkripsi oleh sistem dengan menggunakan notasi "Big O", jadi setiap karakter yang diinput oleh user merupakan sebuah primitive task, contoh: "UTSPDT" memiliki 6 primitive task.

B. Communication

Dalam Communication, dilakukan 2 buah komunikasi, yaitu: Local Communication dan Global Communication. Dalam communication, master process akan mengirimkan character yang telah diinput oleh user untuk di kerjakan oleh setiap process dan jumlah characternya, ketika proses pengiriman ini, kita dapat menggunakan "MPI_Send" untuk mengirimkan characternya, agar slave dapat menerima character yang dikirimkan oleh master process, kita dapat menggunakan "MPI_Recv" agar slave process dapat menerimanya. Selain "MPI_Send" dan "MPI_Recv", kita juga dapat menggunakan "MPI_Bcast" untuk melakukan broadcast agar processor yang berjalan mengetahui jumlah pergeseran character dan panjang katanya.

```
77          // membagi task ke slave
78          for (x = 1; x < size - 1; x++) {
79              idx = x * elementsPerProcess;
80              // komunikasi antara master ke slave
81              MPI_Send(&elementsPerProcess, 1, MPI_INT, x, 0, MPI_COMM_WORLD);
82              MPI_Send((void*)&userText[idx], elementsPerProcess, MPI_CHAR, x, 0, MPI_COMM_WORLD);
83          }
84
85          // task yang harus dikerjakan oleh slave processor //
86          idx = x * elementsPerProcess;
87          int elements_left = n - idx;
88
89          // komunikasi antara master ke slave
90          MPI_Send(&elements_left, 1, MPI_INT,x, 0, MPI_COMM_WORLD);
91          MPI_Send((void*)&userText[idx], elements_left, MPI_CHAR, x, 0, MPI_COMM_WORLD);
92      }
93
```

Komunikasi serta membagi task dari master ke slave

```
102     // slave process
103     else {
104         // mendapat kiriman dari master processor
105         MPI_Recv(&n_elements_recieve, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, &stat);
106         MPI_Recv(&slavePlaceHolder, n_elements_recieve, MPI_CHAR, 0, 0, MPI_COMM_WORLD, &stat);
107
```

Slave process menerima kiriman task dari master processor

```
67      // broadcast ke semua processor ketika terjadi pergeseran karakter//
68      MPI_Bcast(&shift, 1, MPI_INT, 0, MPI_COMM_WORLD);
69
70      // broadcast panjang n (userText) ke semua processor //
71      MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
```

Broadcast pergeseran karakter dan panjang dari userText

C. Agglomeration

Agglomeration adalah proses yang menggabungkan antar primitive task agar menjadi 1 task yang akan dikerjakan oleh processor yang diminta oleh user.

Contoh: karakter yang diterima adalah “UTSPDTSEMESTERGENAP”, pada tahap proses ini, processor akan membagi karakter, digunakan 4 processor, sehingga akan terjadi 4 pembagian task. Processor 0 (Master) untuk karakter “UTSP”, processor 1-3 (Slave) untuk karakter “DTSE”,”MEST”, dan “ERGENAP”.

```
94          // enkripsi master process
95          for (x = 0; x < elementsPerProcess; x++) {
96              printf("processor %d have the data of %c\n", rank, userText[x]);
97              masterChiperText[x] = caesarChiper(userText[x], shift);
98          }
```

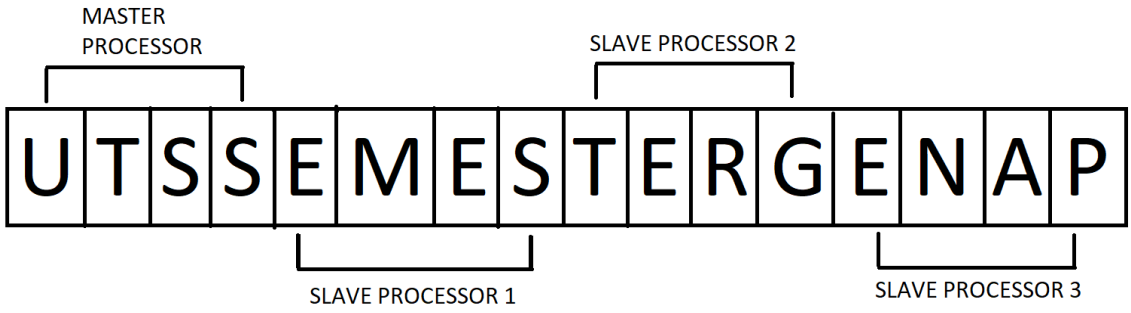
```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.18363.720]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\GL553VD\Documents\PDT UTS\AndreBudiman_00000032851_IF655_D_UTS_PDT\Debug>mpiexec -n 4 AndreBudiman_00000032851_
IF655_D_UTS_PDT.exe
Enter Words: UTSPDTSEMESTERGENAP
Enter the number of shifts : 4

processor 2 have the data of M
processor 2 have the data of E
processor 2 have the data of S
processor 2 have the data of T
slave chiper : QIWX
processor 1 have the data of D
processor 1 have the data of T
processor 1 have the data of S
processor 1 have the data of E
slave chiper : HXWI
processor 0 have the data of U
processor 0 have the data of T
processor 0 have the data of S
processor 0 have the data of P
master chiper : YXWT

Total elapsed time:   5.068625
Encode results : YXWTQIWXHXWIIVKIR_T
processor 3 have the data of E
processor 3 have the data of R
processor 3 have the data of G
processor 3 have the data of E
processor 3 have the data of N
processor 3 have the data of A
processor 3 have the data of P
slave chiper : IVKIR_T
```

D. Mapping

Mapping dilakukan secara dinamis. Proses mapping akan membagi task dan sisa task yang harus dikerjakan. Hal ini dimulai ketika user telah menginput karakter-karakter yang harus dikerjakan oleh processor, lalu processor akan mengecek berapa banyak processor yang dapat bekerja, ketika jumlah processor yang dapat bekerja melebihi 1, maka master processor akan mengirimkannya ke slave processor dengan maksud mempercepat kerja process, hal ini dilakukan juga untuk memotong waktu yang dibutuhkan oleh processor untuk menyelesaikan setiap primitive task. Setiap process yang bekerja juga tidak saling menunggu, atau dengan kata lain, setiap processor dapat bekerja secara bersamaan, hal inilah yang membuat sebuah process yang dipecah dapat menjadi lebih cepat untuk selesai ketika di process lebih dari 1 processor.

Contoh:

Menggunakan 4 processor untuk karakter “UTSSEMESTERGENAP”, terjadi pemecahan primitive task untuk dikirimkan dari master processor ke slave processor, pemecahannya menjadi “UTSS”, “EMES”, “TERG”, dan “ENAP”




```
84
85          n = strlen(userText); // mengukur panjang userText //
86          elementsPerProcess = n / size; // memecah task yang harus dikerjakan per processor //
87
88          // task yang harus dikerjakan oleh slave processor //
89          idx = x * elementsPerProcess;
90          int elements_left = n - idx;
```

Line code 85-86 = pembagian task

Line code 88-90 = sisa task yang harus dikerjakan

Referensi :

J. Quinn, Michael. “Parallel Programming in C with MPI and Open MP Edition 3”.