# Documentatie lab 3

**Hodosi Attila - g234/1**

**16.11.2022**

**Analiza cerintelor:**

Se considera doua numere mari reprezentate sub forma unui tablou de cifre, intregi fara semn, in care cifra cea mai nesemnificativa este pe prima pozitie. Se cere adunarea acestor numere folosind programe bazate pe MPI.

**Cazuri de testare:**

1) Numar 1 = “123456789123456789” = Numar2
2) N_1=1000 si N_2=1000 (random digits)
3) N_1=100 si N_2=100000 (random digits

**Idee generala:**

**Varianta 1:**

Procesul 0 citeste din fisiere chunkuri aproximativ egale cu cel mult 1 elment diferenta.

```
chunk = MAX / (nr_procese - 1);
rest = MAX % (nr_procese - 1);
indexStart = 0;

fin1 >> n_1;
fin2 >> n_2;

for (int i = 0; i < nr_procese - 1; i++) {
    indexEnd = indexStart + chunk;
    if (rest > 0) {
        indexEnd++;
        rest--;
    }

    for (int j = indexStart; j < indexEnd; j++) {
        if (j >= n_1) {
            a[j] = 0;
        }
        else {
            fin1 >> a[j];
        }

        if (j >= n_2) {
            b[j] = 0;
        }
        else {
            fin2 >> b[j];
        }
    }
```

Dupa ce citeste un chunk il distribuie la urmatorul proces liber care va incepe sa efectueze adunarea cifrelor de pe acel chunk.

```
int nr_elemente = indexEnd - indexStart;
MPI_Send(&indexStart, 1, MPI_INT, id_proces_curent, 0, MPI_COMM_WORLD);
MPI_Send(&nr_elemente, 1, MPI_INT, id_proces_curent, 0, MPI_COMM_WORLD);

MPI_Send(a + indexStart, nr_elemente, MPI_SHORT, id_proces_curent, 0, MPI_COMM_WORLD);
MPI_Send(b + indexStart, nr_elemente, MPI_SHORT, id_proces_curent, 0, MPI_COMM_WORLD);
```

```
MPI_Recv(&indexStart, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
MPI_Recv(&nr_elemente, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);

MPI_Recv(a + indexStart, nr_elemente, MPI_SHORT, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
MPI_Recv(b + indexStart, nr_elemente, MPI_SHORT, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);

short carry = 0;
for (int i = indexStart; i < indexStart + nr_elemente; i++) {
    c[i] = (a[i] + b[i] + carry) % 10;
    carry = (a[i] + b[i] + carry) / 10;
}
```

La final fiecare proces trimite carryul procesului urmator, care va actualiza rezultatul in mod corespunzator.

```
if (id == nr_procese - 1) {
    c[MAX] = carry;
}
else {
    MPI_Send(&carry, 1, MPI_SHORT, id + 1, 0, MPI_COMM_WORLD);
}

if (id == 1) {
    carry = 0;
}
else {
    MPI_Recv(&carry, 1, MPI_SHORT, id - 1, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
}
if (carry) {
    int i = indexStart;
    while (carry && i < indexStart + nr_elemente)
    {
        c[i] = (c[i] + carry) % 10;
        carry = (c[i] + carry) / 10;
    }
}
```

In final procesul 0 va primi toate chunkurile si va avea rezultatul final.

```
if (id == nr_procese - 1) {
    MPI_Send(c + indexStart, nr_elemente + 1, MPI_SHORT, 0, 0, MPI_COMM_WORLD);
}
else {
    MPI_Send(c + indexStart, nr_elemente, MPI_SHORT, 0, 0, MPI_COMM_WORLD);
}
```

```
indexStart = 0;
rest = MAX % (nr_procese - 1);
id_proces_curent = 1;
for (int i = 0; i < nr_procese - 1; i++) {
    indexEnd = indexStart + chunk;
    if (rest > 0) {
        indexEnd++;
        rest--;
    }
    int nr_elemente = indexEnd - indexStart;
    if (id_proces_                                    {
        MPI_Recv(c                                  e + 1, MPI_SHORT, id_proces_curent, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    }
    else {
        MPI_Recv(c + indexStart, nr_elemente, MPI_SHORT, id_proces_curent, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    }

    indexStart = indexEnd;
    id_proces_curent++;
}
```
(local variable) int nr_elemente
Search Online

**Varianta 2:**

Procesul 0 va citi cele doua numere si calculeaza numarul de elemente pe care la va primi fiecare proces.

```
if (id == 0) {
    readN1(a, MAX_LOCAL);
    readN2(b, MAX_LOCAL);
    nr_elem = MAX_LOCAL / nr_procese;
}
```

Se face broadcast la numarul de elemente si se distribuie elementele corespunzator intre procese.

```
MPI_Bcast(&nr_elem, 1, MPI_INT, 0, MPI_COMM_WORLD);


short* a_local = new short[nr_elem];
short* b_local = new short[nr_elem];
short* c_local = new short[nr_elem];

MPI_Scatter(a, nr_elem, MPI_SHORT, a_local, nr_elem, MPI_SHORT, 0, MPI_COMM_WORLD);
MPI_Scatter(b, nr_elem, MPI_SHORT, b_local, nr_elem, MPI_SHORT, 0, MPI_COMM_WORLD);
```

Ficare proces calculeaza suma cifrelor si trimite mai departe carryul la procesul urmator, care va actualiza rezultatul in mod corespunzator.

```
if (id == nr_procese - 1) {
    MPI_Send(&carry, 1, MPI_SHORT, 0, 0, MPI_COMM_WORLD);
}
else {
    MPI_Send(&carry, 1, MPI_SHORT, id + 1, 0, MPI_COMM_WORLD);
}

if (id == 0) {
    MPI_Recv(&carry, 1, MPI_SHORT, nr_procese - 1, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
}
else {
    MPI_Recv(&carry, 1, MPI_SHORT, id - 1, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
}

c[MAX] = 0;
if (carry) {
    if (id == 0) {
        c[MAX] = carry;
    }
    else {
        int i = 0;
        while (carry && i < nr_elem)
        {
            c_local[i] = (c_local[i] + carry) % 10;
            carry = (c_local[i] + carry) / 10;
            i++;
        }
    }
}
```

La final se aduna toata chunkurile si se formeaza rezultatul.

```
MPI_Gather(c_local, nr_elem, MPI_SHORT, c, nr_elem, MPI_SHORT, 0, MPI_COMM_WORLD);
```

**Comparatia generala in ms pe 4 procese:**

| N1 | N2 | Secvential | V1 | V2 |
|---|---|---|---|---|
| 18 | 18 | 1,59 | 20,93 | 20,98 |
| 1000 | 1000 | 16,07 | 26,13 | 26,55 |
| 100 | 100000 | 857,26 | 264,277 | 265,315 |

- N1, N2 numarul de cifre

**Concluzie generala:**

Daca avem numere mari cu putine cifre e mai eficient sa folosim varianta secventiala, insa daca avem foarte multe cifre ne ajuta variantele distribuite pe procese.