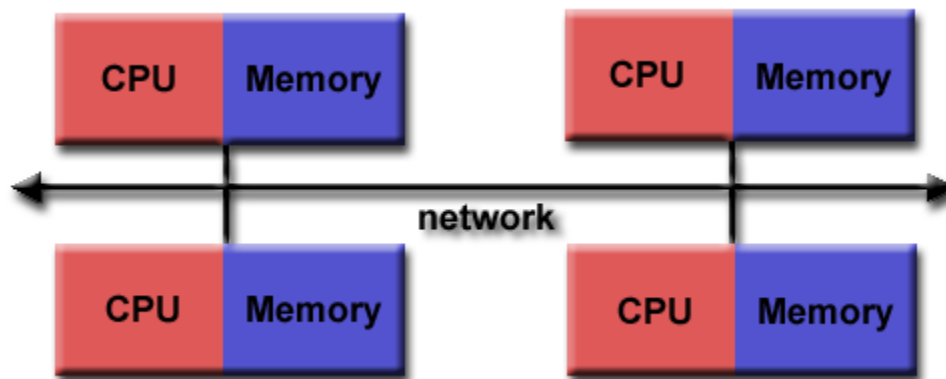

Laborator 7

MPI și Visual Studio C++.

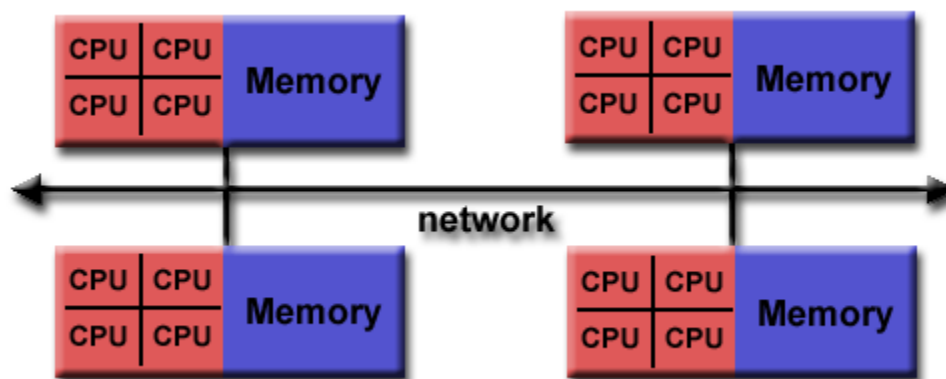
Comunicația între secțiunile paralele de calcul.

Ce este MPI?

MPI (Message Passing Interface Standard) este un standard ce specifică modalitatea de implementare a bibliotecilor de calcul paralel pentru modelele de comunicație prin mesaje pentru arhitecturi multi-calculator. Dacă inițial modelul multi-calculator implica doar arhitecturi monoprosesor:



ulterior funcționalitatea a fost extinsă și pentru sisteme multi-multiprosesor:



astfel la ora actuală se pot implementa algoritmi de calcul paralel utilizând comunicația prin mesaje și la nivel de multiprosesor mecanismele de distribuție a efortului de calcul fiind transparente pentru programator și scalabile de la o singură mașină multiprosesor la sisteme distribuite de mari dimensiuni.

Compilarea și execuția unui program MPI sub Visual Studio

Pentru a compila programe ce utilizează mecanisme MPI cu ajutorul Visual Studio C++ este necesară instalarea componentei Microsoft MPI (ultima versiune este versiunea 8):

Microsoft MPI v8

<https://www.microsoft.com/en-us/download/details.aspx?id=54607>

Suplimentar, proiectele ce folosesc componenta Microsoft MPI trebuie configurate pentru a vedea calea către fișierele binare necesare în procesul de compilare dar și în procesul de execuție. Un ghid amănunțit al modului de configurare al proiectului se găsește la adresa:

How to set up and run MS MPI using MS Visual Studios

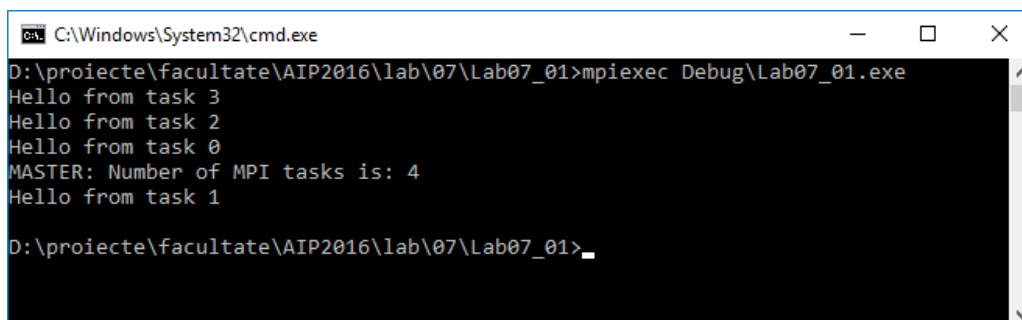
<http://www.math.ucla.edu/~mputhawala/PPT.pdf>

După urmarea instrucțiunilor de configurare a proiectului se va testa următorul program:

```
#include "mpi.h"
#include <stdio.h>
#define MASTER 0

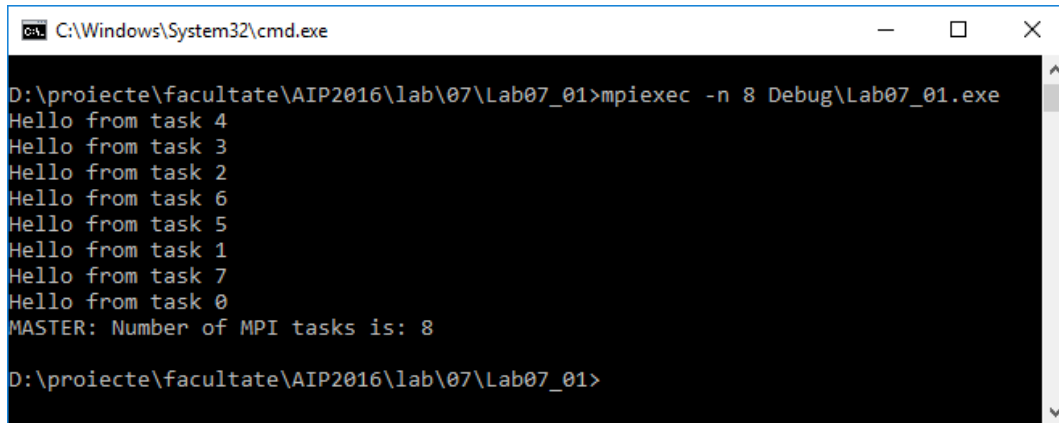
int main(int argc, char *argv[])
{
    int numtasks, taskid, len;
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &numtasks);
    MPI_Comm_rank(MPI_COMM_WORLD, &taskid);
    printf("Hello from task %d\n", taskid);
    if (taskid == MASTER)
        printf("MASTER: Number of MPI tasks is: %d\n", numtasks);
    MPI_Finalize();
    return 0;
}
```

Execuția programului va genera următoarea ieșire (pe un procesor cu 4 nuclee):



```
C:\Windows\System32\cmd.exe
D:\proiecte\facultate\AIP2016\lab\07\Lab07_01>mpiexec Debug\Lab07_01.exe
Hello from task 3
Hello from task 2
Hello from task 0
MASTER: Number of MPI tasks is: 4
Hello from task 1
D:\proiecte\facultate\AIP2016\lab\07\Lab07_01>
```

Implicit, mediul de execuție MPI va genera un număr de fire de execuție egal cu numărul de procesoare din sistem. Pentru a modifica acest lucru se poate adăuga parametrul *-n* la execuția mediului *mpiexec*:



```
C:\Windows\System32\cmd.exe

D:\proiecte\facultate\AIP2016\lab\07\Lab07_01>mpiexec -n 8 Debug\Lab07_01.exe
Hello from task 4
Hello from task 3
Hello from task 2
Hello from task 6
Hello from task 5
Hello from task 1
Hello from task 7
Hello from task 0
MASTER: Number of MPI tasks is: 8

D:\proiecte\facultate\AIP2016\lab\07\Lab07_01>
```

În cadrul unui program MPI secțiunea de calcul paralel NU este delimitată de instrucțiunile *MPI_Init()* și *MPI_Finalize()*. Spre deosebire de OpenMP, unde paralelizarea se realizează doar pe anumite secțiuni ale programului, un program MPI este executat integral în paralel de mai multe ori. Puneți o instrucțiune *printf* înainte de *MPI_Init()* și observați efectul. Funcțiile *MPI_Comm_size* și *MPI_Comm_rank*, utilizate în program, permit preluarea numărului de task-uri concurente și a identicatorului fiecărui task.

Comunicația între secțiunile paralele de calcul. Calcularea numărului Pi.

Pornind de la problema dată în laboratorul 2 (calculul numărului Pi) să se realizeze o implementare paralelă utilizând MPI.

```
#include "stdafx.h"
#include "mpi.h"
#define MASTER 0

int main(int argc, char *argv[])
{
    int i, num_steps = 200000000;
    double step, x, pi, sum = 0.0;
    step = 1.0 / (double)num_steps;
    int numtasks, taskid, len;
    MPI_Status status;
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &numtasks);
    MPI_Comm_rank(MPI_COMM_WORLD, &taskid);
    printf("MPI task %d has started...\n", taskid);
```

```

int interval = num_steps / numtasks;
for (i = taskid*interval; i < (taskid+1)*interval; i++) {
    x = (i + 0.5)*step;
    sum = sum + 4.0 / (1.0 + x*x);
}

if (taskid != MASTER) {
    MPI_Send(&sum, 1, MPI_DOUBLE, 0, 1, MPI_COMM_WORLD);
}
else {
    double sum_buf;
    for (i = 1; i < numtasks; i++) {
        MPI_Recv(&sum_buf, 1, MPI_DOUBLE, i, 1,
                MPI_COMM_WORLD, &status);
        sum += sum_buf; }
    pi = step * sum;
    printf("pi=%f\n", pi);
}
MPI_Finalize();

return 0;
}

```

Având în vedere că în cazul modelului de calcul paralel multi-calculator (modelul utilizat de MPI) nu există memorie partajată / variabile partajate între firele de execuție paralele singura modalitatea de comunicație între acestea sunt mesaje. Programul folosește funcțiile *MPI_Send (&buf,count,datatype,dest,tag,comm)* și *MPI_Recv (&buf,count,datatype,source,tag,comm,&status)* pentru a implementa comunicația între firele de execuție paralele.

Rulați programul și observați comportamentul la execuție cu un număr divers de fire de execuție:

```

Select C:\Windows\System32\cmd.exe
D:\proiecte\facultate\AIP2016\lab\07\Lab07_02>mpiexec -n 16 Debug\Lab07_02.exe
MPI task 4 has started...
MPI task 1 has started...
MPI task 2 has started...
MPI task 5 has started...
MPI task 6 has started...
MPI task 7 has started...
MPI task 9 has started...
MPI task 8 has started...
MPI task 10 has started...
MPI task 11 has started...
MPI task 12 has started...
MPI task 13 has started...
MPI task 14 has started...
MPI task 3 has started...
MPI task 15 has started...
MPI task 0 has started...
pi=3.141593
D:\proiecte\facultate\AIP2016\lab\07\Lab07_02>

```

Completați programul cu diverse mesaje care să confirme că fiecare fir de execuție realizează în mod corect calculele alocate. Încercați să implementați o evaluare de performanță (bazată pe timpul de execuție) astfel încât să comparăm cazul secvențial și cel paralel.