ASC L1

February 22, 2022

1 Instalarea mașinii virtuale WSL

Unul dintre avantajele utilizării mașinilor virtuale de tip WSL (Windows Subsystem for Linux) îl reprezintă posibilitatea de minimiza consumul de resurse folosite de această mașină. Spre deosebire de variantele anterioare, WSL (începând cu jumătatea anului 2021) funcționează și pe distribuțiile Windows Home (10 și 11). Ne vom axa pe acest tip de mașină deoarece instalarea pachetelor MPI în Windows (pentru a le folosi de ex în CobeBlocks) folosește wrapperele Microsoft ce sunt diferite de standardul MPI furnizat de pachetul mpich.

Pentru a rula o astfel de mașină virtuală este necesară instalarea Hyper-V și Virtual Machime Platform. Dacă întâmpinați probleme la instalarea acestora cel mai probabil acest lucru se da torează faptului că extensiile de virtualizare (prezente în majoritatea procesoarelor moderne) nu sunt active și va trebui să intrați în UEFI config (Shift+Restart -> selectați Troubleshoot -> Uefi configuration) și să le activați. Odată instalate la bootarea windows-ului puteți rula instrucțiunea următoare pentru a instala WSL V1.

```
wsl --install
```

După instalare este necesar un reboot și va trebui să setați un username și o parolă pentru mașina virtuală ubuntu instalată implicit. Noi pe această distribuție vom lucra.

Ulterior treceți la instalarea WSL2 descărcând kernel-ul de la adresa (https://wslstorestorage.blob.core.windows.net/wslblob/wsl_update_x64.msi). Va trebui să setați aceasta versiune ca fiind implicită:

```
wsl --set-default-version 2
```

1.1 Integrarea cu VSCode

Dacă nu aveți deja instalat, va fi necesar să descărcați și să instalați VSCode. Odată instalat trebuie activate extensiile Remote-WSL și C/C++ (cel cu intellisense). Acestea vor trebui instalate și în mașina virtuală WSL folosind butonul install on WSL.

Deschide-ți un nou workspace folosind butonul verde din colțul din stânga jos a ferestre VSCode -/- și selectați mașina virtuală existentă. Deschide-ti un terminal în această mașină și instalați pachetele necesare:

```
sudo apt update
sudo apt upgrade
sudo apt install build-essential openmpi-common openmpi-dev libgsl-dev libgsl-dev mpich
```

Ulterior creați un folder în home-ul utilizatorului din WSL în care vom lucra cu VSCode.


```
mkdir ASP
```

În VSCode va trebui să adăugați un folder și veți selecta folder-ul remote creat anterior. Salvați

workspace-ul pentru a-l putea refolosi ulterior.

1.1.1 Testarea integrării C/C++

în folder-ul workspace vom crea un folder test în care vom rula un test simplu c++ pentru verificarea corectitudinii funcționării pentru sistemul nostru. Astfel, în folder-ul respectiv vom crea fișierul test.cpp cu următorul conținut:

```
#include <iostream>
#include <vector>
#include <string>

using namespace std;

int main()
{
                        vector<string> msg {"Hello", "C++", "World", "from", "VS Code", "and the C++ extension!"};

    for (const string& word : msg)
    {
        cout << word << " ";
    }
    cout << endl;
}
```

Cu ocazia scrierii programului puteți verifica și integrarea intellisense. Va trebui să configurăm sistemul de build folosind din meniu ** Terminat -> Configure Default Build Task ** unde veți selecta ca sistem de build **g++** după care puteți face operația de build (**CTRL+SHIFT+B**). Dacă fișierul de ieșire este creat puteți să deschide-ți un Integrated Terminal (**CTRL+P**-> Terminal: New Integrated Terminal) și să îl executați:

```
./test
```

Dacă totul funcționează corect ați compilat primul program.

1.1.2 Un program mai complex

Vom încerca în cele ce urmează să implementăm un program mai complex. Problema propusă este calculul valorilor $ (x, J_0(x))$ pentru funcția Bessel J_0 pe intervalul 0-5 folosind un pas de 0.2. Vom afișa a 7-ea pereche la consolă. Rezultatul va fi scris într-un fișier binar urmând ca datele conținute să fie plasate pe un grafic trasat cu **gnuplot**.

Programul este următorul:

```
#include <stdio.h>
#include <stdlib.h>
#include <gsl/gsl_sf_bessel.h>
#include <math.h>
```



```
#define NMAX 200
```

```c
#define MY_MEM_ERR 1
#define MY_FILE_ERR 2
#define MY_CALL_ERR 3
#define MY_SUCCESS 0

int main(int argc, char ** argv) {
	if (argc != 2) {
		printf("\n Utilizare. %s nume_fisier \n", argv[0]);
		printf("\n nume_fisier este numele fisierului binar de date");
		printf(" ce urmeaza a fi creat");
		printf("\n Programul calculeaza perechile (x, J0(x)) pe un ");
		printf("grid cu pasul 0.2 pe intervalul [0,5]");
		printf("\n si stoceaza aceste perechi in fisierul binar ");
		printf("nume_fisier. La final extrage si afiseaza a 7-a pereche");
		printf("\n\n");
		return MY_CALL_ERR;
	}

	double x, *data, data_set[2];
	int i;
	FILE *fp;

	if ((data = (double *) malloc(2*NMAX*sizeof(double))) == NULL) {
		fprintf(stderr, "%s: nu pot aloca memorie. Ies ... \n", argv[0]);
		return MY_MEM_ERR;
	}

	for (i = 0; i < 2*NMAX; i+=2) {
		x = ((double)i)*5/((double)NMAX)/2.;
		*(data + i) = x;
		*(data +i + 1) = gsl_sf_bessel_J0(x);
	}

	if ((fp = fopen(argv[1], "wb")) == NULL) {
		fprintf(stderr, " %s: nu pot crea fisierul de date %s. Ies ... \n", argv[0], argv[1] )return MY_FILE_ERR;
	}

	for (i=0; i < 2*NMAX; i+=2) {
		fwrite(data+i, sizeof(double), 2, fp);
	}

	fclose(fp);

	if ((fp = fopen(argv[1], "rb")) == NULL) {
			fprintf(stderr,"%s: Nu pot deschide fisierul de date %s. Ies ...\n",argv[0],argv[1]); 3

		return MY_FILE_ERR;
```

```
    }

    fseek(fp, 12*sizeof(double), SEEK_SET);
    fread(data_set, sizeof(double), 2, fp);
    fclose(fp);
    printf(" Perechea a 7-ea din fisier: x = %lf, y=%lf \n\n", data_set[0], data_set[1]); free(data);
    return MY_SUCCESS;
}
```

El se va salva într-un fișier *CPP*, compilat și executat în linia de comandă. Astfel, pentru complilarea lui - deoarece folosește librăria gsl (Gnu Scientific Linux) va trebui să o includem și pe aceasta în linia de compilare. Considerând faptul ca am salvat fișierul sub denumirea **exc.cpp** vom deschide un terminal (**CTRL+P** - New Integrated Terminal) în consola WSL unde vom rula următoarea comandă:

```
gcc -o exc exc.cpp -lgsl
```

după rularea corectă a acestei instrucțiuni putem să executăm programul nostru cu parametrul ce conține fișierul de ieșire astfel:

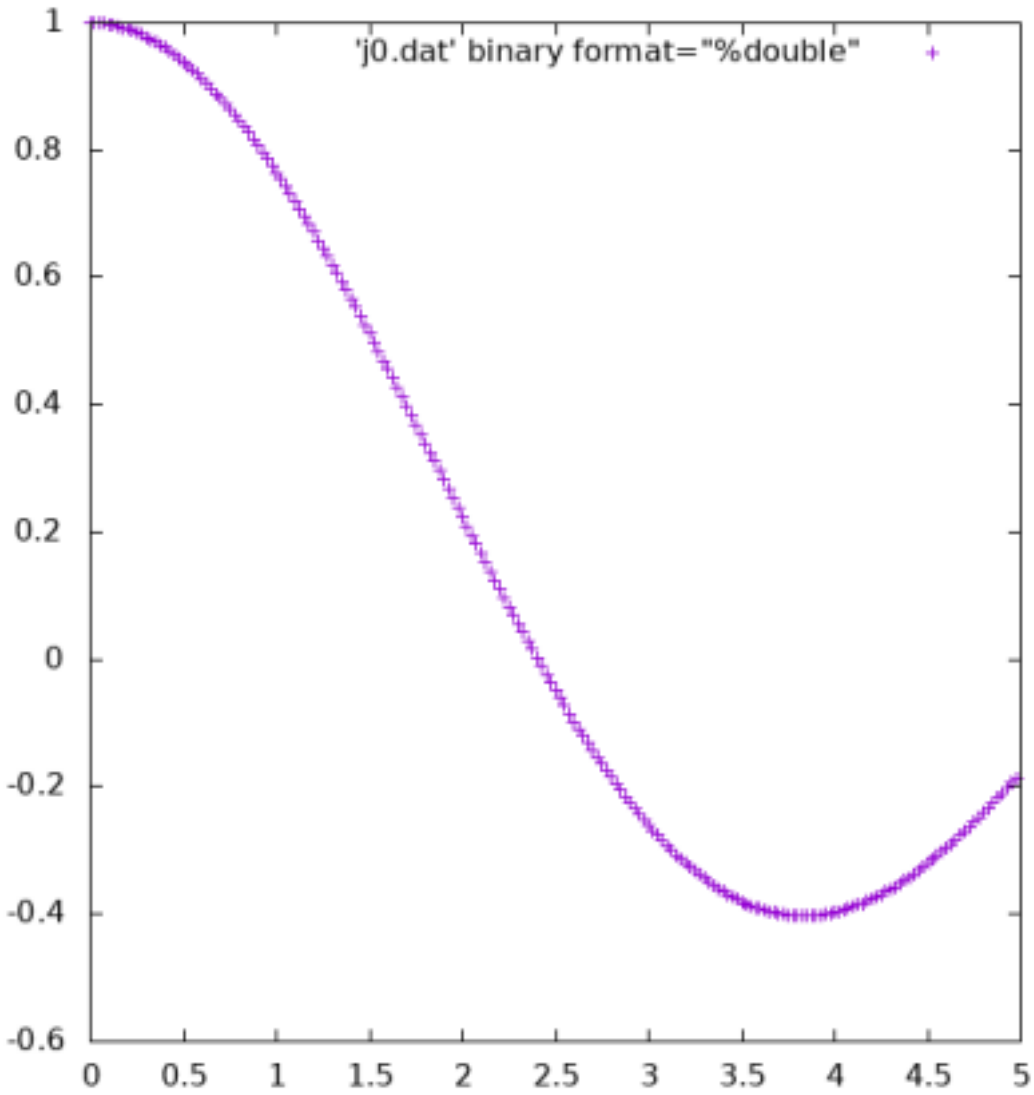
```
./exc j0.dat
```

și el va crea fișierul binar. Pentru trasarea graficului aferent fișierului binar vom folosi următoarea secvență:

```
gnuplot
set terminal png size 600,600
set output 'j0.png'
plot 'j0.dat' binary format='%double'
quit
```

după care, în VSCode va apare fișierul **j0.png** pe care îl deschideți pentru a verifica conținutul.

```
[2]: from IPython.display import display, Image
     display(Image(filename="j0.png", height=600, width=600))
```



Până în acest moment am verificat integrarea compilatorului C/C++ din mediul WSL cu VSCode. În cele ce urmează vom testa funcționalitatea mediului MPI.

1.2 Verificarea integrării cu MPI

Pentru că în acest semestru ne vom ocupa de calcul paralel distribui iar distribuțiile WSL alocă implict un singur procesor va fi nevoie să alocăm mașinii virtuale un număr mai mare de core-uri. Acest lucru se realizează implicit în versiunea 2 a WSL.

Vom rula următoarul program:

MPI: Hello world

```
#include <stdlib.h>
#include <mpi.h>
```



```
#include <stdio.h>
```

```
int main (int argc, char **argv)
{
// numarul de procese din comunicator
int numprocs;
// rangul procesului
int myrank;

// Initializare mpi
MPI_Init (&argc, $argv);
MPI_Comm_size (MPI_COMM_WORLD, $numprocs);
MPI_Comm_rank (MPI_COMM_WORLD, $myrank);

//Program
printf( "Eu sunt nodul %i \n", myrank);

//Finalizare MPI
MPI_Finalize();

return 0;
}
```

Pentru compilarea programului folosind utilitarul specific MPI va trebui să intrăm în terminal și să rulăm următoarea comandă în folderul în care am creat fișierul:

mpicc -o LI Li.cpp

după care putem rula programul nostru folosind comanda:

mpirun -n 2 ~/ASP/L1/L1

Atenție, trebuie folosită calea absolută către fișierul executabil. Variind valorile lui n puteți executa programul pe mai multe core-uri în funcție de numărul de core-uri disponibile pe sistemul Dvs.

1.3 Bibliografie

Materialul suport pentru laboratorul de Arhitectura Sistemelor Paralele

https://www.windowscentral.com/how-install-wsl2-windows-10

https://code.visualstudio.com/docs/cpp/config-wsl#_add-a-source-code-file 6