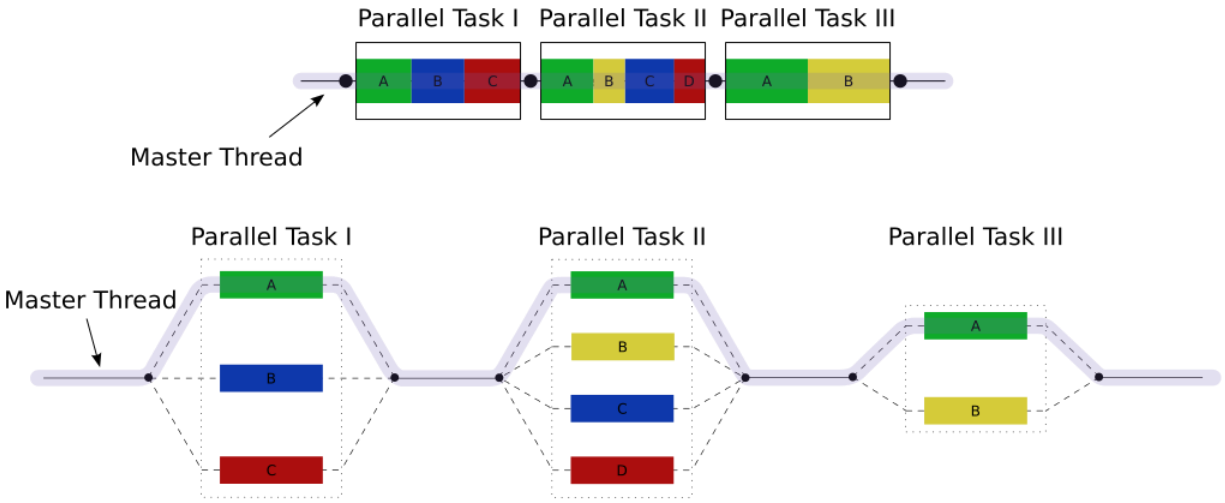
Laborator 1
OpenMP și Visual Studio C++

Ce este OpenMP?

OpenMP este o bibliotecă de tip API (application programming interface) pentru programarea paralelă a sistemele multiprocesor cu memorie partajată. Se poate utiliza în limbajele C, C++ și Fortran și are suport pentru majoritatea sistemelor de operare (Windows, Linux, Unix, macOS) și mediilor de dezvoltare (gcc, Visual C++, Oracle Developer Studio, IBM XL C/C++, Intel C/C++).

Utilizează un model de tip fire multiple de execuție în care un fir de execuție principal (master thread) generează mai multe fire de execuție paralele secundare (slave parallel tasks) – model fork / join. Acest model permite într-un mod simplu accelerarea execuției pe un sistem de calcul paralel cu memorie partajată.



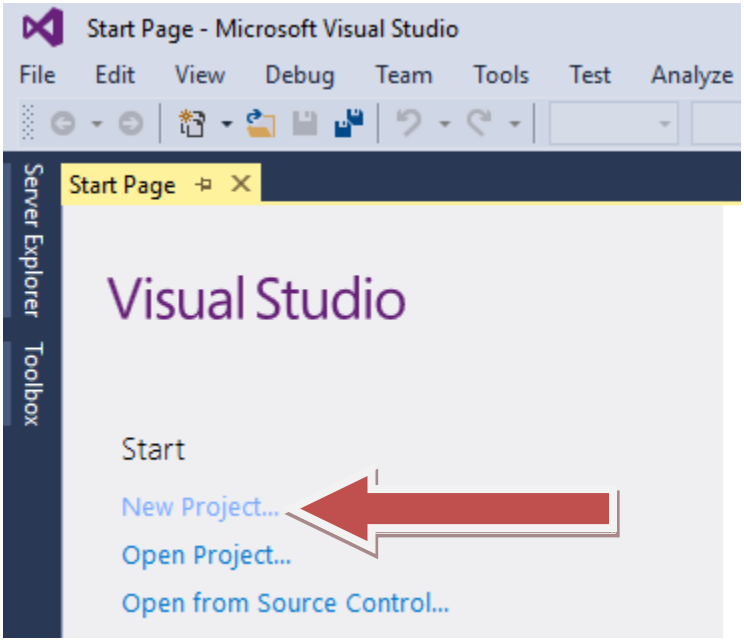
Utilizarea OpenMP în Visual Studio C++

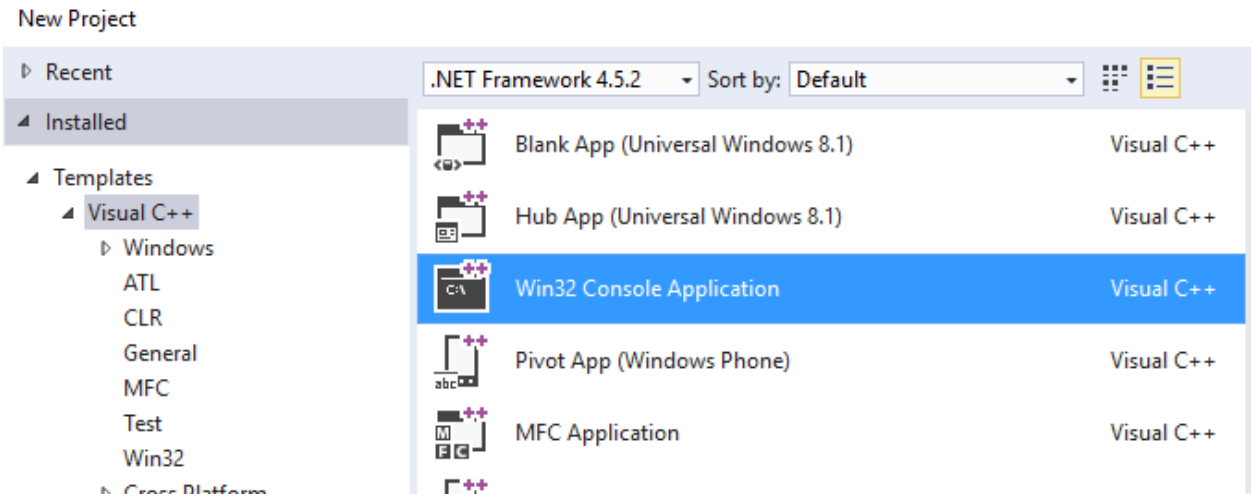
Mediul de dezvoltare Visual Studio include suport pentru OpenMP 2.0:

http://www.openmp.org/wp-content/uploads/cspec20.pdf

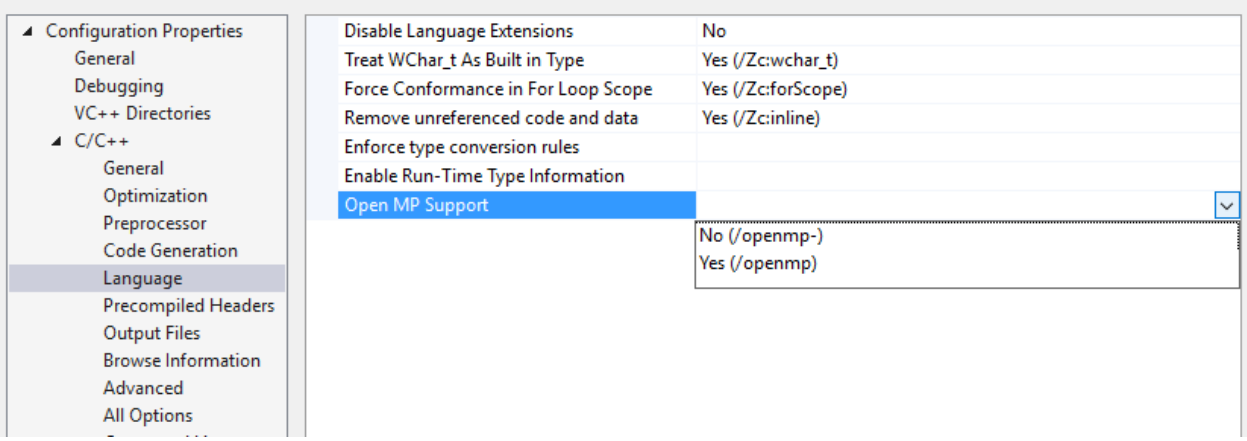
Pentru a scrie un program C++ în Visual Studio care să utilizeze OpenMP se vor efectua următorii pași:

- ✓ Se va crea un nou proiect de tip Visual C++ / Win32 Console Application





- ✓ Se va configura suportul OpenMP pentru proiect (click dreapta pe proiect / Properties / C/C++ / Language / Open MP Support / Yes (/openmp))



- ✓ În fișierul principal al proiectului (nume_proiect.cpp) se va introduce următorul cod

```
#include "stdafx.h"
#include "omp.h"

int main()
{
#pragma omp parallel
	{
		int ID = omp_get_thread_num();
		printf(" hello(%d) ", ID);
		printf(" world(%d) \n", ID);
	}
getchar();
return 0;
}
```

- ✓ Dacă configurația proiectului este corectă rezultatul execuției proiectului va arăta de genul

```
hello(0)  hello(2)  world(2)
hello(3)  hello(1)  world(1)
world(0)
world(3)
```

Includerea bibliotecii (*omp.h*) și a directivei specifice (*#pragma omp parallel*) a făcut ca execuția programului să implice pornirea mai multor fire de execuție (numărul lor depinde de numărul de procesoare / nuclee din sistem și a resurselor sistem disponibile – încărcare procesor,

memorie). Exemplul precedent a fost rulat pe un microprocesor Intel Core i5 cu patru nuclee. Dacă dorim să controlăm numărul de fire de execuție putem introduce instrucțiunea

```
omp_set_num_threads(N);
```

înainte de directiva *#pragma omp*. *N* va indica numărul de fire de execuție impuse spre execuție, de exemplu pentru N=16 execuția va genera:

```
hello(5)  hello(8)  world(8)
hello(0)  world(0)
hello(4)  world(4)
hello(9)  world(9)
hello(3)  world(3)
hello(10)  world(10)
hello(2)  world(2)
hello(11)  world(11)
hello(1)  world(1)
hello(12)  world(12)
hello(7)  world(7)
hello(13)  world(13)
world(5)
hello(6)  world(6)
hello(14)  world(14)
hello(15)  world(15)
```

Funcția *omp_get_thread_num()*, utilizată în program, returnează ID-ul firului de execuție de unde este invocată și ne permite să observăm execuția distinctă a firelor de execuție. Se observă că execuția paralelă nu este ordonată, rezultatul execuției (afișarea) nu este în ordine ci este aleatoriu dictat de accesul concurent la consolă (fiecare execuție va arăta altfel). Temă de gândire: de ce apar un număr diferit de elemente (cuvinte hello / world) pe linii?