Comp3402 Working with Matrices

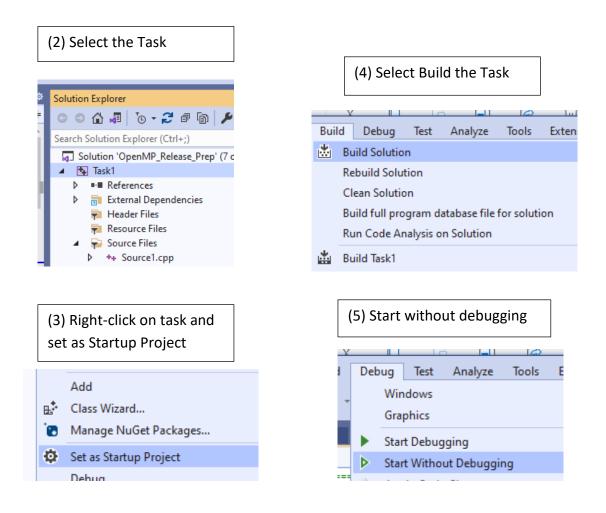
C.B.Price March 2022

Purpose	(i) To investigate basic concepts of the OpenMP (ii) To investigate its application to vector and matrix problems							
Files Required	Visual Studio Solution comprising various tasks							
ILO Contribution	6							
Send to Me								
Assignment Info.								
Homework	Read Chapter 8							

Activities

Workflow. Here you will proceed as follows:

(1) Open up the Visual Studio solution (sln), then follow the following steps.



1 A Simple Matrix – How C stores the values of a matrix in physical memory

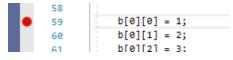
Here we shall be looking at a tiny matrix-vector multiplication and do the computation in sequential mode. Here's the arrays we shall use

$$\begin{bmatrix} ?\\ ?\\ ?\end{bmatrix} = \begin{bmatrix} 1 & 2 & 3\\ 4 & 5 & 6 \end{bmatrix} \cdot \begin{bmatrix} 1\\ 2\\ 3 \end{bmatrix}$$

The result on the left is **a**, the matrix in the middle is **b** and the vector on the right is **c**.

(a) Open up Task 6, inspect the code and run the task. You should get an answer of [14, 32].

(b) Now we are interested in how the matrix **b** is stored in physical memory. So, add a breakpoint at the point where we are writing into the matrix like this



Now call Debug > Start Debugging. Go to the Debug tab (in the running debugger) and select the following

	Deb	ug Test	Analyze	Tools	Extensions	Window	Help	Search (Ctrl+Q)	م		OpenMP_Rele	ase_Prep		
Ŧ		Windows Graphics				+	-	eakpoints ception Settings	Ctrl+Alt+B Ctrl+Alt+E		<u>≣</u> 2≣	। भ भ •	² ∭. _∓	
u		Terminate All Restart Apply Code Changes			F5 Ctrl+Alt+Break Shift+F5		(0) X/	utput AML Binding Failures now Diagnostic Tools	Ctrl+Alt+F2	ar	ar ** arqv)			
	×						XX G	PU Threads			uigvy			
ar C	ර 2				Ctrl+Shift+F5 Alt+F10	t+F5	 Tasks Parallel Stacks Parallel Watch 		Ctrl+Shift+D, K Ctrl+Shift+D, S	,				
te un pr t	\$ \$	Performance Profiler Relaunch Performance Profiler Attach to Process Other Debug Targets		Alt+F2 Shift+Alt+F2 Ctrl+Alt+P		周 Ar [性] Lo	/atch utos pcals	Ctrl+Alt+V, A Ctrl+Alt+V, L	•					
ູວນ	÷• ?• ↔	Step Into Step Over Step Out	o Over		F11 F10 Shift+F11	 Immediate Live Visual Tree Live Property Explorer 		Ctrl+Alt+I						
цe	⇔	QuickWatc	h		Shift+F9			all Stack	Ctrl+Alt+C					
	*	Toggle Brea New Break Delete All B	ooint reakpoints		F9 Ctrl+Shif	► t+F9	I ■ M [®] Pr	Modules Processes	Ctrl+Alt+H Ctrl+Alt+U Ctrl+Alt+Z Ctrl+Shift+Alt+D					
	02	Disable All					М	lemory		۲ e	Memory 1	Ctr	I+Alt+M, 1	

You will get a Window Memory1. At the right set Columns to 12. Now in the Address: place type **&b**. You may need to scroll the Source window down so you can see the assignments of the matrix.

Now hit F11 to step through the code. You should see each element of **b** coming into memory. More important is the 3 elements of the first row come into successive memory locations, then the three elements of the second row of **b** com in. *In other words, a matrix is stored as rows within memory*.

2 How to work with matrices in loops

or

To work on a matrix we need two loops, one looping over the rows and one over the columns. We could do this in two ways. In the first, the outer loop goes over the rows and the inner loop over the columns, and in the second way the outer loop takes the columns and the inner loop the rows. Like this.

```
for (int row = 0; row < nrRows; row++) {
    for (int col = 0; col < nrCols; col++) {
        sum += a[row][col];
    }
}
for (int col = 0; col < nrCols; col++) {
    for (int col = 0; row < nrRows; row++) {
        sum += a[row][col];
    }
}</pre>
```

Let's investigate which of these is more efficient.

(a) Open up **Task 7** where the first variant is coded. You may need to reduce NR_ROWS and/or NR_COLS to get some speed-up. Note down the time it takes

(b) Comment out the above variant and un-comment out the second. Recompile and re-run. What do you find?

(c) Does this make sense knowing that rows are stored in memory? It's all to do about the cache.