PERFORMANCE REVIEW – MPI

Using 4 nodes, MPI can calculate a matrix multiplication for 2 200x200 sized matrices in an average of 35ms on my machine. Unfortunately, this is the highest amount my machine can handle, so I can't test with larger data sizes.

With just 1 node, the MPI implementation can solve the same matrix calculation in an average of 75ms. A marked improvement can be seen when using 4 separate processes to calculate the matrix multiplication problem.

Finally, using 50 nodes yields an average execution time of 170ms. This clearly shows that a significant overhead is produced with the large number of nodes in order to execute a relatively small portion of the total work on each node, which counteracts the desired effects that we want from splitting the computation across multiple processes.

PERFORMANCE REVIEW – MPI + OpenMP

Using 4 nodes and one OpenMP thread on each node, it yields very similar results to using 4 nodes on the pure MPI implementation, as expected. If we increase the number of threads on each node to 2 we start to see some small improvement with an average execution time of 26ms.

Using 50 threads and 4 nodes we start to see some decay in the efficiency, with an average execution time of 40ms.

Finally, with 50 nodes each utilising 50 threads we see some very poor performance due to thread creation and join overheads, as well as creation and synchronization of 50 separate processes, with an average execution time of 202ms.

PERFORMANCE REVIEW – MPI + OpenCL

The OpenCL implementation is exceptionally slow, likely due to how much setup, initialisation, and memory allocation is required within each node compared with the low amount of work to do since my machine can't handle large arrays. The results very from 1000ms for 1 nodes to ~2000ms for 4 nodes and above 4000ms for 10+ nodes.

It is clear to see here that because of the lengthy setup and initialisation process, and the increasingly smaller amount of work to do as the problem is split across more nodes, that the OpenCL implementation becomes very slow. When the problem to solve is larger (e.g. with 1 node – master) the problem is of a sufficient size to be able to solve faster, yet still takes a lot of time during the setup and initialisation phase, resulting in the slower times compared with the OpenMP and pure MPI implementations, which require significantly less setup and initialisation resources.

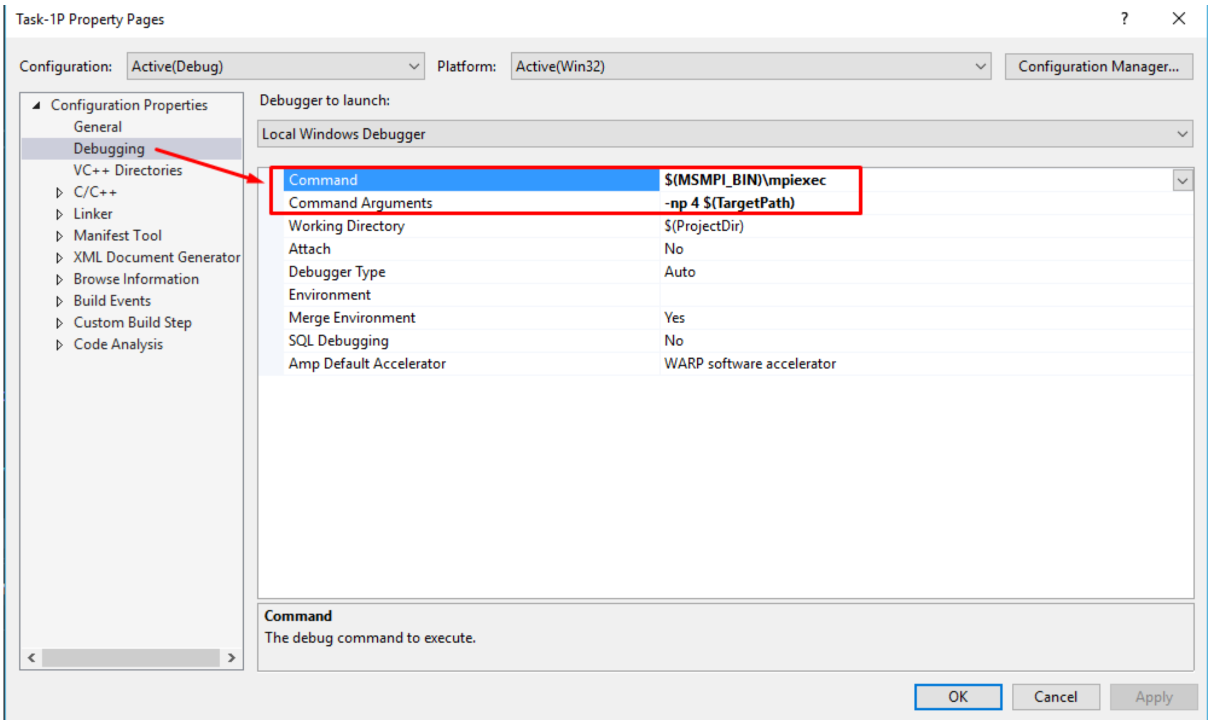
For much larger problem sizes the benefits of computing on GPU cores can be realised using OpenCL. The large number of cores means the problem can be split and run in parallel many more times than what can be achieved by a regular processor, and if the problem subsets are large enough such that the overheads are negligible then the real advantages of OpenCL and GPU processing would be seen.

PROGRAM COMPILATION AND RUN EVIDENCE

Given that I wrote and compiled the above programs in Windows Visual Studio 2017 with custom configurations, it is likely they will not work on the marker's machine or provided Ubuntu VM. I have included below the relevant Project Properties configurations as well as links to the MS-MPI source and OpenCL for AMD source, and some screenshots of the program outputs for small matrix sizes.

MS-MPI Source: https://www.microsoft.com/en-us/download/details.aspx?id=57467

OpenCL for AMD Source: https://github.com/GPUOpen-LibrariesAndSDKs/OCL-SDK/releases

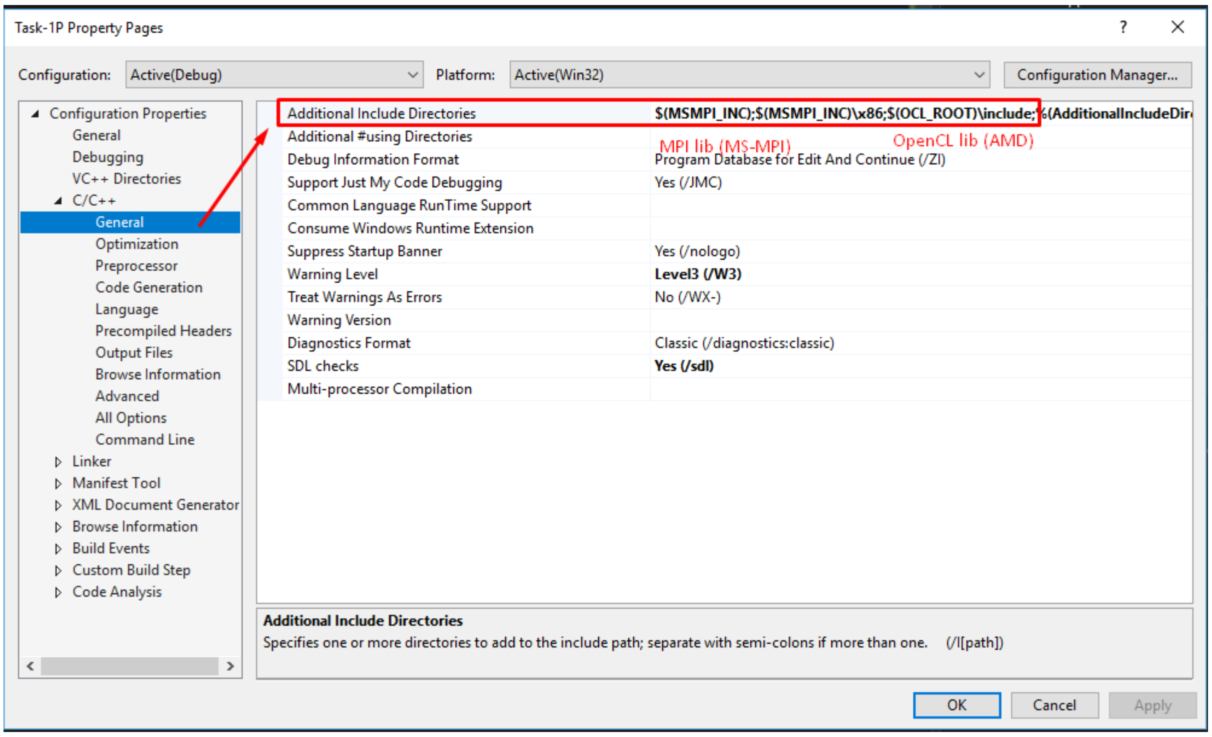


Debugger to launch: Local Windows Debugger

Command	$(MSMPI_BIN)\mpiexec
Command Arguments	-np 4 $(TargetPath)
Working Directory	$(ProjectDir)
Attach	No
Debugger Type	Auto
Environment	
Merge Environment	Yes
SQL Debugging	No
Amp Default Accelerator	WARP software accelerator

Command
The debug command to execute.

Figure 1 - Command line arguments (similar to mpirun -np 4 mpi_1 command in Ubuntu VM)



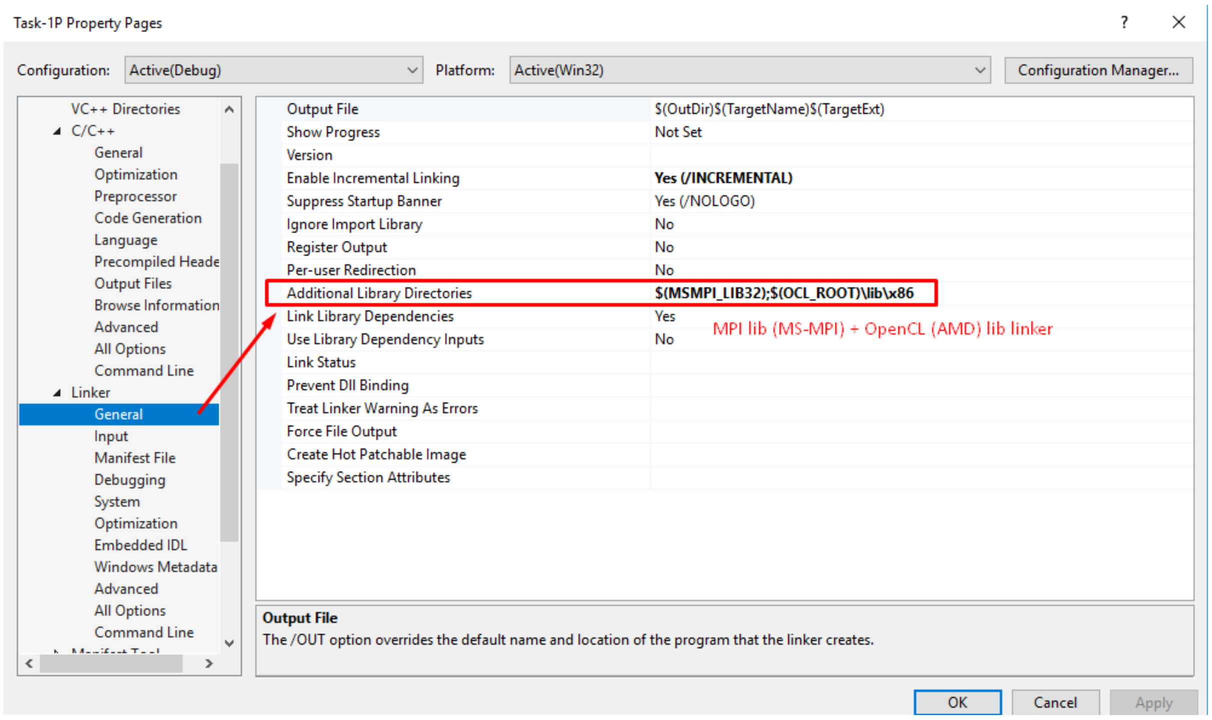
Configuration: Active(Debug) Platform: Active(Win32)

- Configuration Properties
  - General
  - Debugging
  - VC++ Directories
  - C/C++
    - General
    - Optimization
    - Preprocessor
    - Code Generation
    - Language
    - Precompiled Headers
    - Output Files
    - Browse Information
    - Advanced
    - All Options
    - Command Line
  - Linker
  - Manifest Tool
  - XML Document Generator
  - Browse Information
  - Build Events
  - Custom Build Step
  - Code Analysis

Additional Include Directories	**$(MSMPI_INC);$(MSMPI_INC)\x86;$(OCL_ROOT)\include;%(AdditionalIncludeDir**
Additional #using Directories	
Debug Information Format	Program Database for Edit And Continue (/ZI)
Support Just My Code Debugging	Yes (/JMC)
Common Language RunTime Support	
Consume Windows Runtime Extension	
Suppress Startup Banner	Yes (/nologo)
Warning Level	**Level3 (/W3)**
Treat Warnings As Errors	No (/WX-)
Warning Version	
Diagnostics Format	Classic (/diagnostics:classic)
SDL checks	**Yes (/sdl)**
Multi-processor Compilation	

Additional Include Directories
Specifies one or more directories to add to the include path; separate with semi-colons if more than one. (/I[path])

Figure 2 - C++ Includes for MS-MPI and OpenCL - lets you #include <CL/cl.h> and <mpi.h>



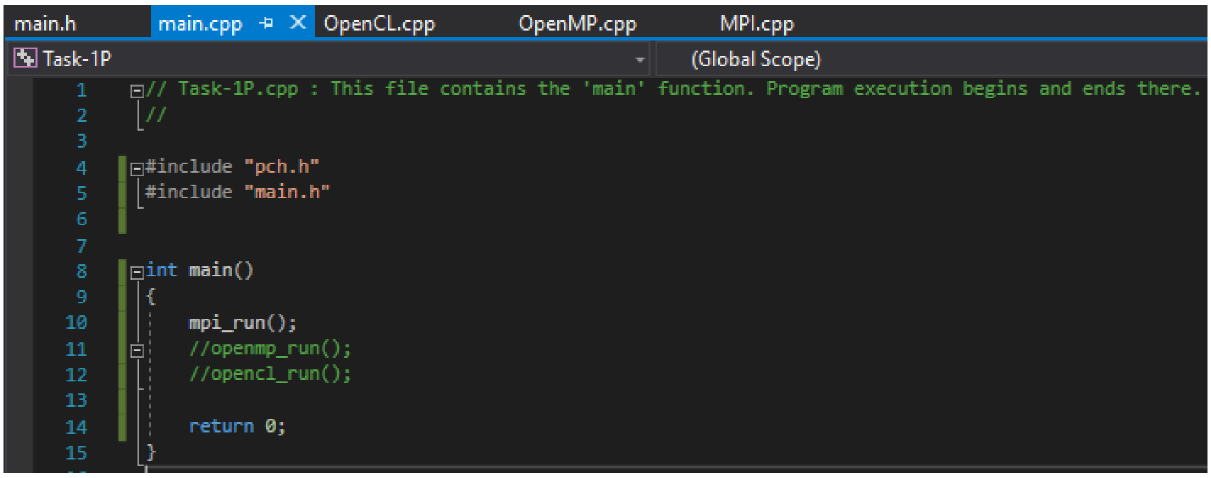
Configuration: Active(Debug) Platform: Active(Win32)

- VC++ Directories
- C/C++
  - General
  - Optimization
  - Preprocessor
  - Code Generation
  - Language
  - Precompiled Heade
  - Output Files
  - Browse Information
  - Advanced
  - All Options
  - Command Line
- Linker
  - General
  - Input
  - Manifest File
  - Debugging
  - System
  - Optimization
  - Embedded IDL
  - Windows Metadata
  - Advanced
  - All Options
  - Command Line

Output File	$(OutDir)$(TargetName)$(TargetExt)
Show Progress	Not Set
Version	
Enable Incremental Linking	**Yes (/INCREMENTAL)**
Suppress Startup Banner	Yes (/NOLOGO)
Ignore Import Library	No
Register Output	No
Per-user Redirection	No
Additional Library Directories	**$(MSMPI_LIB32);$(OCL_ROOT)\lib\x86**
Link Library Dependencies	Yes
Use Library Dependency Inputs	No
Link Status	
Prevent Dll Binding	
Treat Linker Warning As Errors	
Force File Output	
Create Hot Patchable Image	
Specify Section Attributes	

Output File
The /OUT option overrides the default name and location of the program that the linker creates.

Figure 3 - Linker directories for MS-MPI and OpenCL libraries



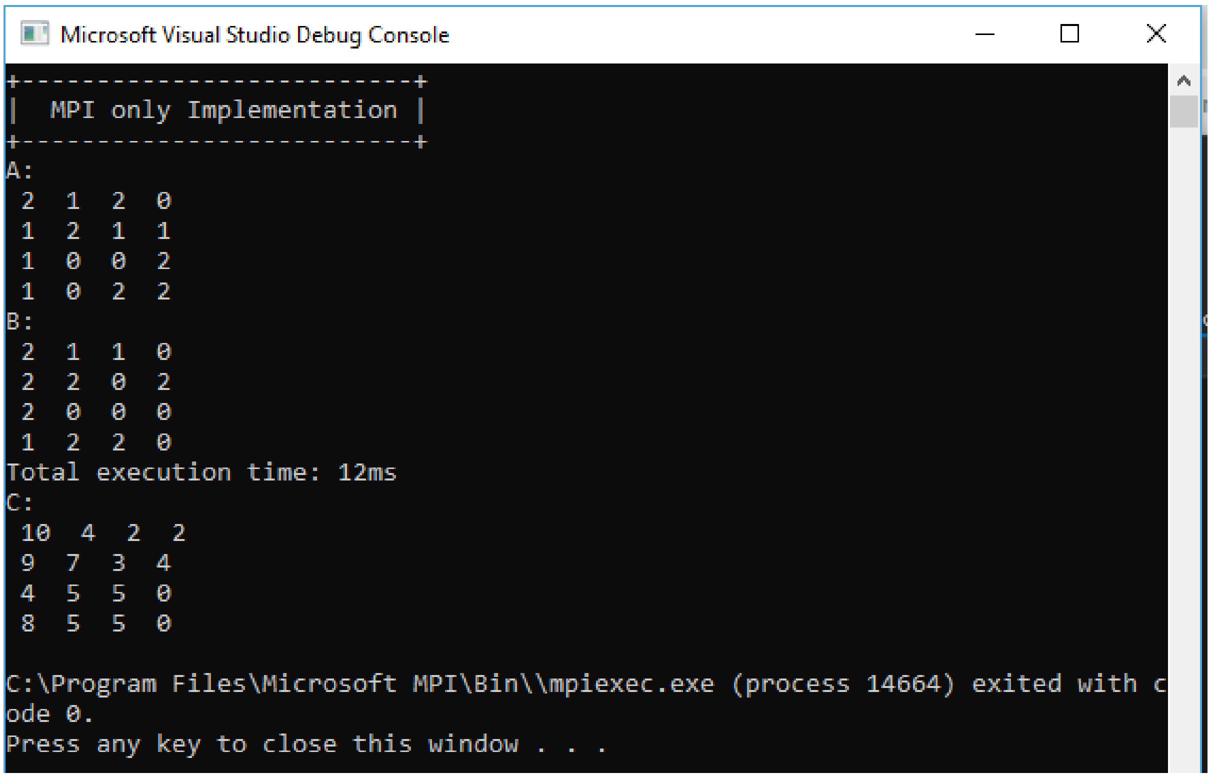
```cpp
// Task-1P.cpp : This file contains the 'main' function. Program execution begins and ends there.
//

#include "pch.h"
#include "main.h"


int main()
{
    mpi_run();
    //openmp_run();
    //opencl_run();

    return 0;
}
```

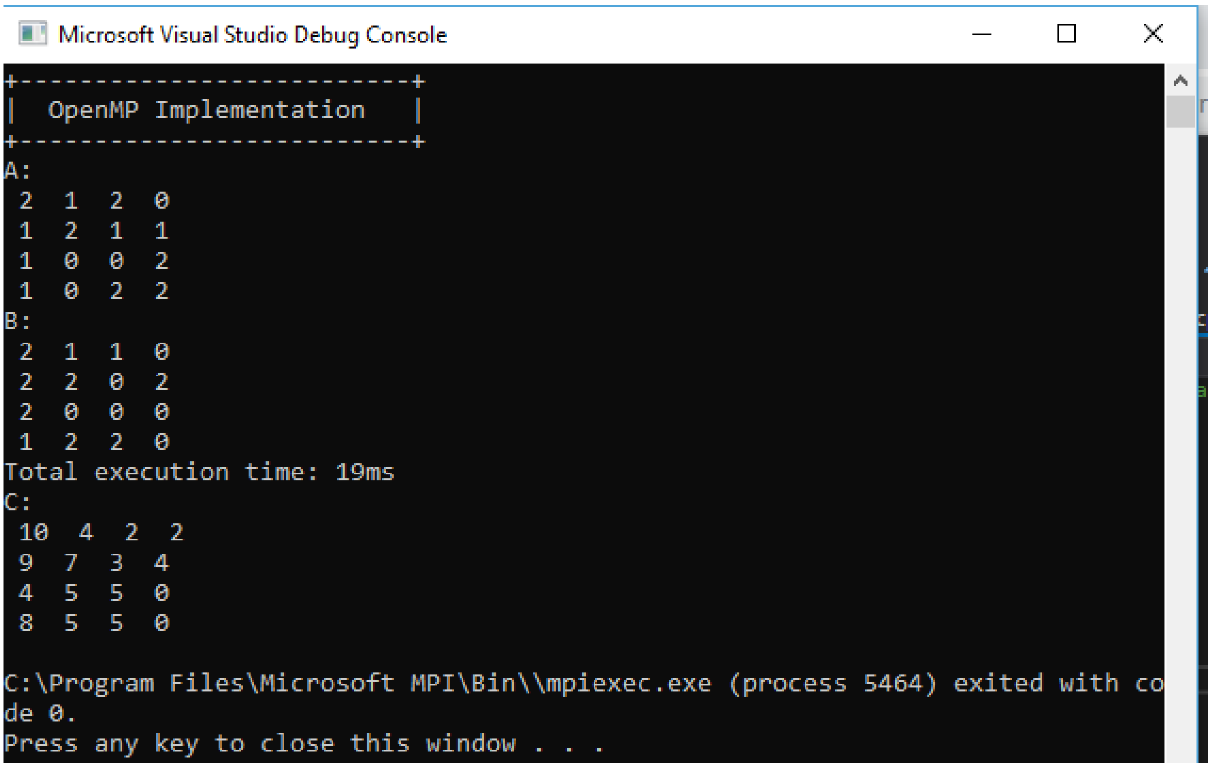
Figure 4 - Program structure - each .cpp file contains the respective _run function that is called in main.cpp, and main.h contains the declarations.



```
+--------------------------+
|  MPI only Implementation |
+--------------------------+
A:
 2  1  2  0
 1  2  1  1
 1  0  0  2
 1  0  2  2
B:
 2  1  1  0
 2  2  0  2
 2  0  0  0
 1  2  2  0
Total execution time: 12ms
C:
 10  4  2  2
 9  7  3  4
 4  5  5  0
 8  5  5  0

C:\Program Files\Microsoft MPI\Bin\\mpiexec.exe (process 14664) exited with c
ode 0.
Press any key to close this window . . .
```

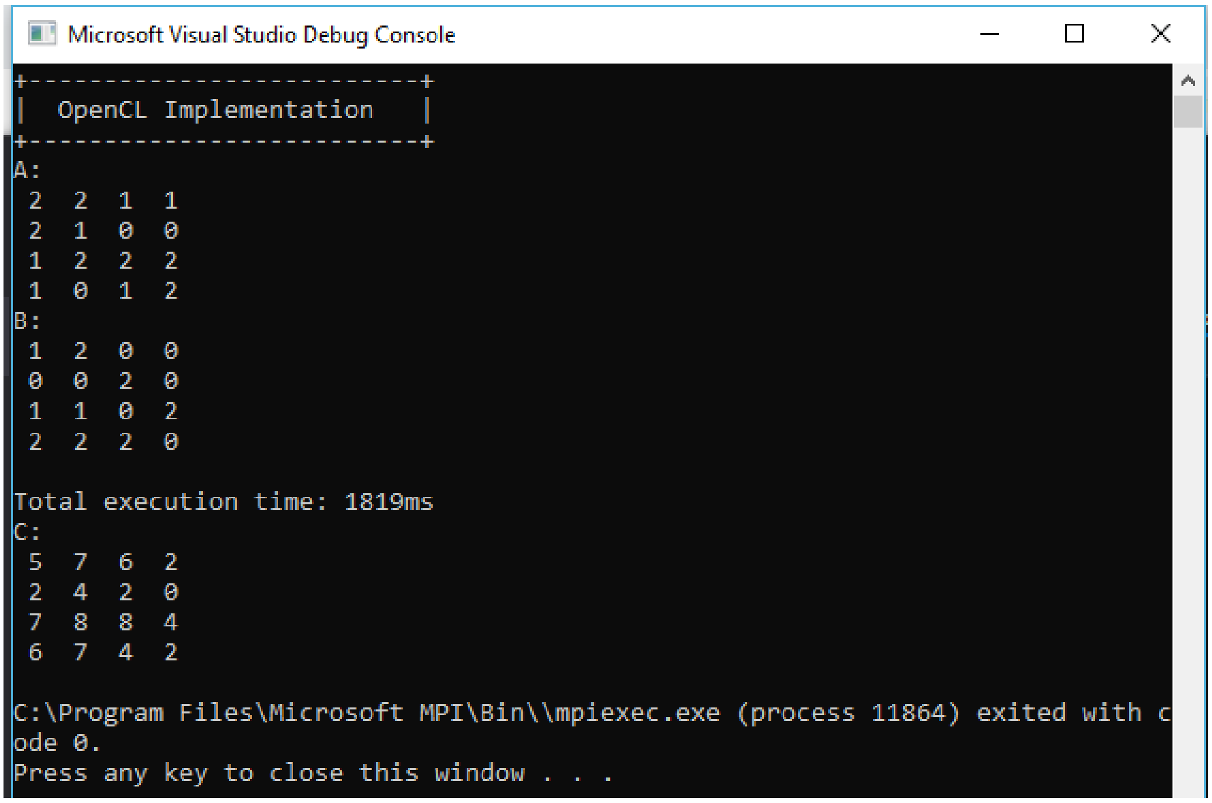
Figure 5 - MPI only implementation output



```
+---------------------------+
|  OpenMP Implementation    |
+---------------------------+
A:
 2  1  2  0
 1  2  1  1
 1  0  0  2
 1  0  2  2
B:
 2  1  1  0
 2  2  0  2
 2  0  0  0
 1  2  2  0
Total execution time: 19ms
C:
 10  4  2  2
 9  7  3  4
 4  5  5  0
 8  5  5  0

C:\Program Files\Microsoft MPI\Bin\\mpiexec.exe (process 5464) exited with co
de 0.
Press any key to close this window . . .
```

Figure 6 - OpenMP + MPI implementation output



```
+---------------------------+
|  OpenCL Implementation    |
+---------------------------+
A:
 2  2  1  1
 2  1  0  0
 1  2  2  2
 1  0  1  2
B:
 1  2  0  0
 0  0  2  0
 1  1  0  2
 2  2  2  0

Total execution time: 1819ms
C:
 5  7  6  2
 2  4  2  0
 7  8  8  4
 6  7  4  2

C:\Program Files\Microsoft MPI\Bin\\mpiexec.exe (process 11864) exited with c
ode 0.
Press any key to close this window . . .
```

Figure 7 - OpenCL + MPI implementation output