**Setting Up VSCode for MPI Programming**

The Article is by [Li Nguyen](https://medium.com/@li.nguyen_15905?source=post_page-----b6665da6b4ad--------------------------------)

Visual Studio Code or VSCode for short is becoming more and more popular among developers, as it’s a light-weight editor which is customizable and extendable to the user’s needs. It has built in versioning, plug-ins for different languages and allows the developer to do debugging.

Whether you’re working with MPI at work or on your private project or just need to work with it in the scope of your parallel computing class at university, here’s a quick tutorial on how to setup VSCode for MPI Programming on MacOS. I have set this up for Catalina 10.15.4 and VSCode Version 1.46.0.

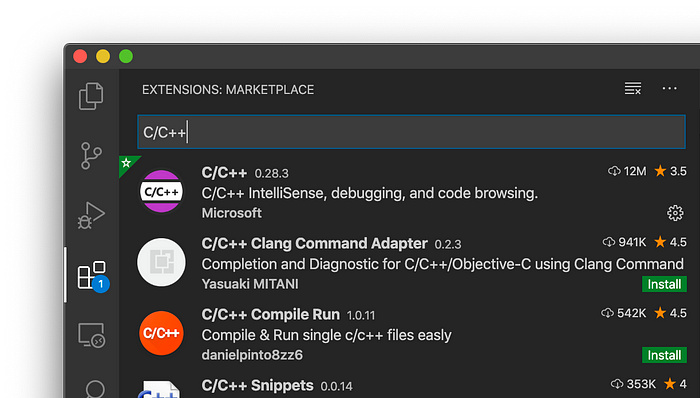
**1. Install Visual Studio Code on macOS**

If you don’t have VSCode on your system yet, download it from their [website](https://code.visualstudio.com/docs?dv=osx).

Later, we want to open VSCode from the terminal. To be able to do that open the command palette by pressing ⇧⌘P and type ‘shell command’. From the dropdown menu choose **Shell Command: Install ‘code’ command in PATH.**

**2. Install the C/C++ Extension for VSCode.**

To do this you go to the extensions icon in the icons bar on the left and search for C/C++. Then click on “Install”.



**3. Install OpenMPI**

Download the current stable release of OpenMPI from their [website](https://www.open-mpi.org/software/ompi/v2.0/) (I chose the tar.gz-file) and unpack by double click. Change into the unpacked directory named ‘openmpi-2.0.x’ and run the following command.

cd openmpi-2.0.x/  
./configure --prefix=$HOME/opt/usr/local

So let’s take a closer look at what we’re doing here. When installing software onto UNIX computers, this generally involves three steps:

1. Configuring the makefile
2. Compiling the code according to the makefile and
3. Installing it to your system

The first step is what we’re doing using the configure script and it tailors the software to the system you’re installing it to. Usually, when installing software with make install the files belonging to the installation may be placed in different folders. To make sure that the installation is located in one folder, you can specify this using --prefix. So we’re making sure everything’s installed in $HOME/opt/usr/local .

Now, we’ll proceed with the second and third step of the installation. To compile and install OpenMPI run:

> make all  
> make install

If you then run $HOME/opt/usr/local/bin/mpirun --version you should see the output below:

> $HOME/opt/usr/local/bin/mpirun --versionmpirun (Open MPI) 2.0.2  
Report bugs to http://www.open-mpi.org/community/help/

However, typing in $HOME/opt/usr/local/bin/mpirun every time you want to run an MPI program is kinda tidious, so let’s include that into out PATH environment variable to make life easier. Currently, on Catalina you can add that to your .zshrc file. In your home directory open the .zshrc file using vi .zshrc and adjust your path to contain the binary $HOME/opt/usr/local/bin/ . To currently make OpenMPI 2.0.x releases work, a quick workaround is needed, that is setting the TMPDIR environment variable to a shorter dirctory name, here I chose /tmp.

Finally, “activate” your zshrc with . ~/.zshrc . In case the zshrc file is not run every time you open your terminal you can edit those under Terminal’s Preferences → Profiles → Shell under Startup (see below). Tick the box Run Command and enter . ~/.zshrc .

export PATH="$HOME/opt/usr/local/bin:$PATH"  
export TMPDIR="/tmp"



**4. Create your MPI directory**

Let’s create a working directory for OpenMPI, change into that directory and open VSCode with code . . The dot stands for your current working directory.

mkdir helloworld  
cd helloworld  
code .

Opening VSCode via your terminal is important. Remember, you’ve set the PATH variable, so for the changes to be inherited to VSCode you want to start it from the terminal in which you just activated your bash rc.

**5. Create an MPI helloworld program**

To be able to test if compiling and debugging in VSCode works for MPI, let’s create a simple helloworld program for MPI in our helloworld folder and name it helloworld.c and save with ⌘S.

**6. Setting up compiling and debugging environment**

We are going to adjust the following three files which are located in your .vscode folder. If they don’t exist yet, we’ll see later how to create them.

* tasks.json : This file contains your compiler build settings for VSCode.
* launch.json : This file contains the settings for the debugger.
* c\_cpp\_properties.json : This file contains the compiler path and IntelliSense settings.

If you don’t have a tasks.json file yet, create it and paste the following code:

The label gives this compile option a name, so we call it mpicc build active file, since mpicc is the compiler used for MPI programs. The command value leads us to where we’ve installed this compiler. In the arguments you can see that mpicc generates an executable file with the -o flag. The executable file will have the same name as your .c file. The -g flag generated debug information.

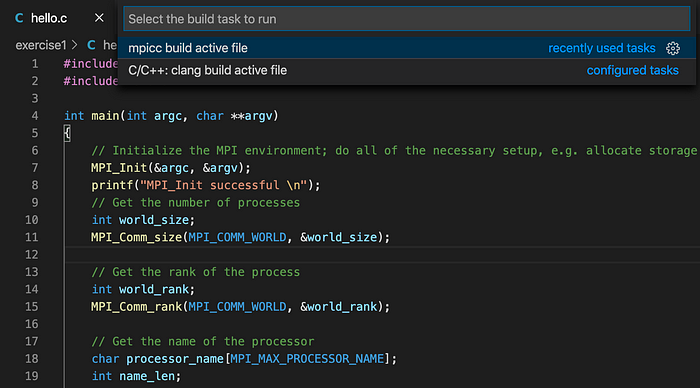
Next, we’re going to create the launch.json file for the debugger settings (if you don’t have that file in the .vscode folder yet. Paste the following code:

Lastly, we create the c\_cpp\_properties.json file (if is does not yet exist). Here, we need to add the include folder to the includePath variable, so that VSCode finds our #include <mpi.h> import. You might want to check your own path to the include folder.

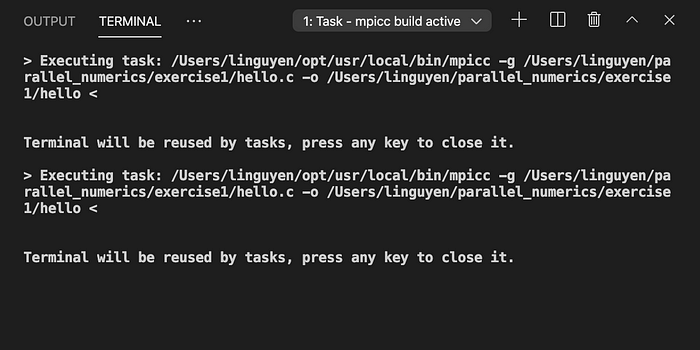
**7. Building and Running our Helloworld Program**

We’re ready to compile and start debugging our little program!

Go back to your helloworld.c file and build by pressing ⇧⌘B or navigate to Terminal → Run Build Task. Choose the mpicc build active file that we configured before:



A successful compilation looks something like this:



Click on the plus symbol to open another terminal. If you want to debug the program you can add a breakpoint somewhere in your helloworld program. To run the program you do:

mpirun -np <number\_of\_processes> helloworld

The -np flag specifies the number of processes. Just try different numbers of processes and see what happens. helloworld is the executable that was generated in the compilation step. When using for example two processes you’d get:

Hello world from processor o29n81p78, rank 0 out of 2 processors  
Hello world from processor o29n81p77, rank 1 out of 2 processors

If you want to start debugging you have to start the program from VSCode by either hitting F5 or Run → Start Debugging in the tab bar.

Thanks for reading!