Project 2 Parallel Programming Skills

Victoria Routon

**Identifying the components on the raspberry PI B+.**

The components of the Raspberry PI B+ include single board computer, quad-core multicore CPI, 1 GB RAM, power, Display, CPU/RAM, HDMI, ethernet, USB and camera.

**How many cores does the Raspberry PI B+ CPU have?**

Raspberry PI B+ is a quad-core processor, which means that it has four independent units called cores.

**List three main differences between x86 (CISC) and ARM Raspberry PI (RISC). Justify your answer and use your own words.**

The size, complexity as well as the execution of the instruction set are the main differences between x86 and ARM architecture. ARM uses RISC processor which usually involves simplified instructions (limited to less than 100) that are operated only on the registers, while x86 uses much more complex instructions to access the memory with fewer registers than ARM. ARM’s fewer instruction also speeds up the execution time by reducing the clock’s cycle. With fewer instruction comes quicker execution time. Shorter instructions, however, limits the programmers who must always think about efficiency of their code. Intel on the other hand, gives the programmer free will when it comes to the instructions.

**What is the difference between sequential and parallel computation and identify the practical significance of each?**

The main difference between the serial (sequential) instruction and parallel is its execution. In serial computing, the instructions are executed one by one on a single processor, while parallel computing can execute instructions simultaneously on different processors.

**Identify the basic form of data and task parallelism in computational problems.**

Data parallelism is a category where the same computation is applied to multiple data items, where parallelism is proportional to the input size. Task parallelism is a category where parallelism is organized around the functions to be performed rather than around the data. It's especially useful when the context in which the data is evaluated matters.

**Explain the differences between processes and threads.**

Process is an executing application of a running program where processes don't have a shared memory. Thread on the other hand is a process that allows a single executable/process to be broken down into much smaller, independent parts. Threads, unlike the processes share the memory with the process they belong to.

**What is OpenMP and what is OpenMP pragmas?**

OpenMP is an interface that enables multi-platform shared memory programming. OpenMP pragmas is a compiler directive that allows the compiler to generate code. OpenMP uses multithreading, which helps the programmer write a simple code with less errors.

**What applications benefit from multi-core (list four)?**

Some of the applications benefits of multi-core include database and Web servers, compilers and multimedia applications.

**Why Multicore? (why not single core? List four)**

It's difficult to make single-core clock frequencies higher, where higher frequencies mean quicker program runtime. Multicore is also widely used because many current applications are multithreads. Nowadays, many programmers shift more towards more parallelism in computer architecture, which is not possible on a single core. Lastly but not least, deeply pipelined circuits are difficult to design.

**b) Parallel programming**

I begin by typing the given code using nano editor. By typing “#pragma omp parallel” on line 10 I’m specifying that everything inside curly brackets should run simultaneously. Therefore, every piece of code before line 10 as well as the lines 15-17 from should run sequentially (Figure 1).



Figure 1 – spmd2 original code

After compiling the code, I was able to execute the program. By typing different command-line arguments I choose how many threads I want to print. However, I quickly notice that the code has an error. Each thread should have a unique ID that is associate with different thread but the output of my code contains repeated numbers (same threads ID) (Figure 2).

This is to be expected because of the variables I declared are outside of the curly brackets. The processor of Raspberry PI has multiple cores, however they share one memory. Therefore, if I want each thread to keep track of its own unique ID, the variables need to be declared inside curly brackets that are run in parallel.

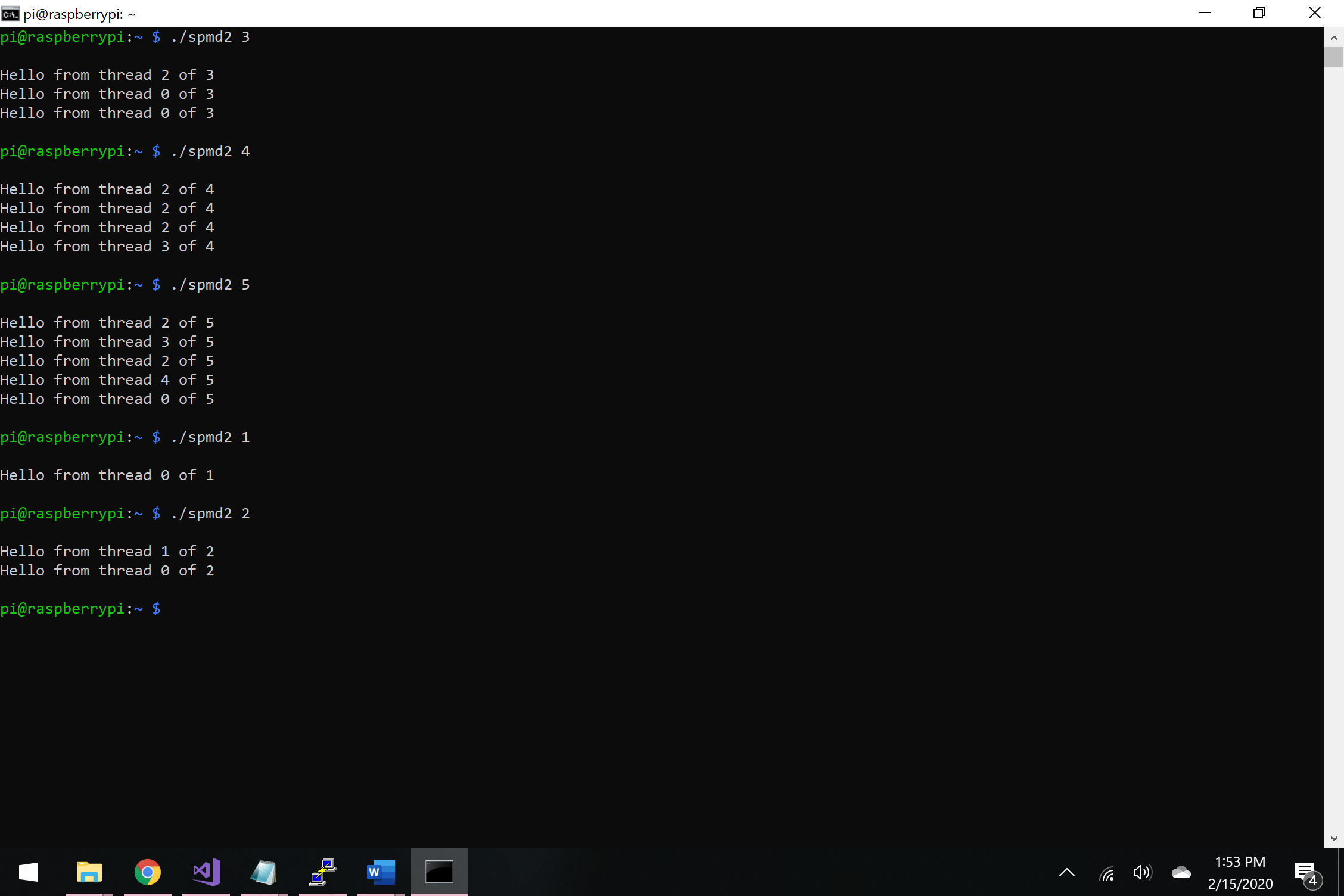


Figure 2 – running spm2 original code

To fix this issue, I commented out the variables that I declared before and declared them inside the curly brackets that are run in parallel instead (line 12 and 13) (Figure 3).

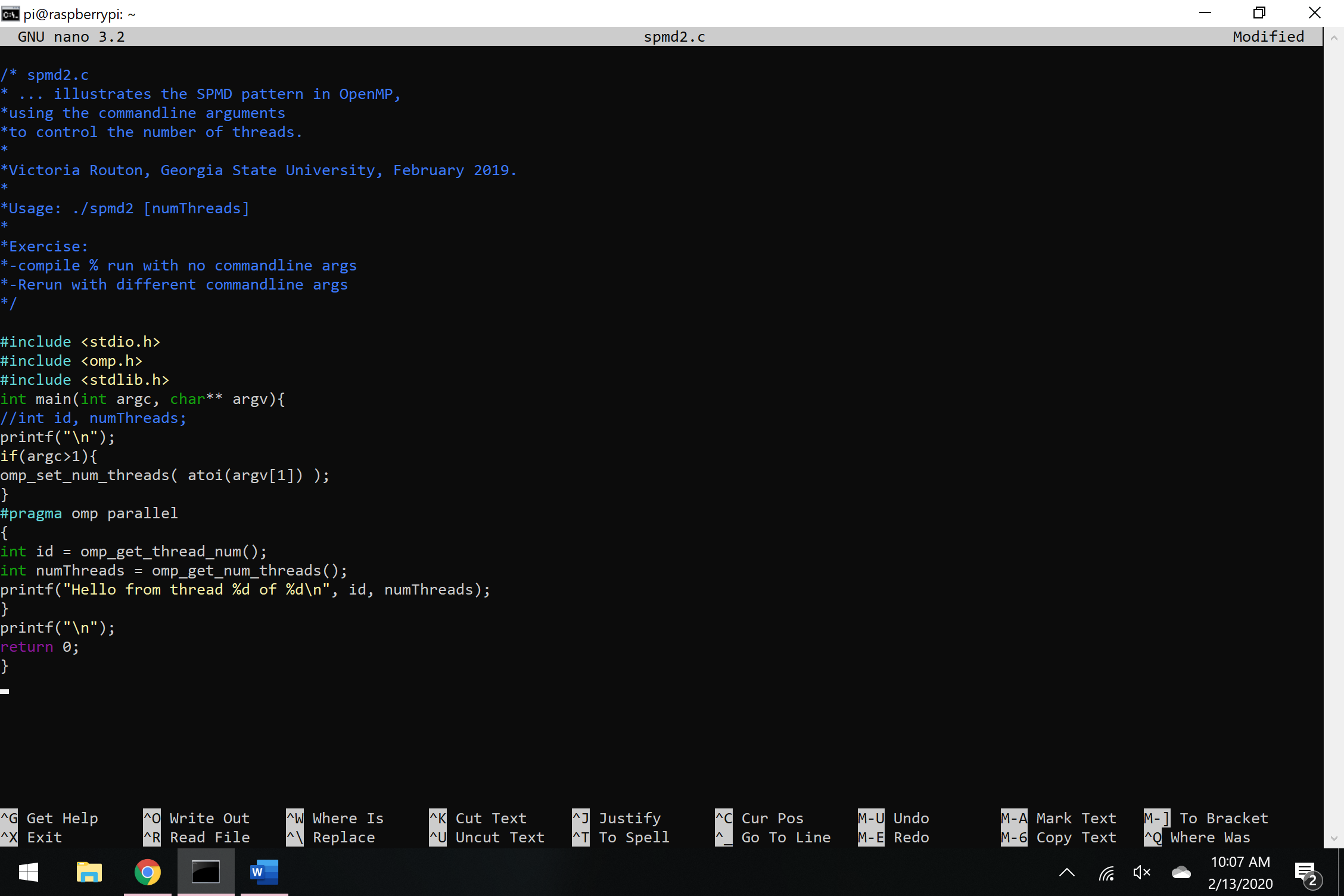


Figure 3 – fixed spmd2 code

After compiling and running the code, each thread keeps track of its own unique ID. This code is not only an excellent example of parallel programming, but it also shows how a simple program can run different parts of the code on different data values stored in memory (Figure 4).

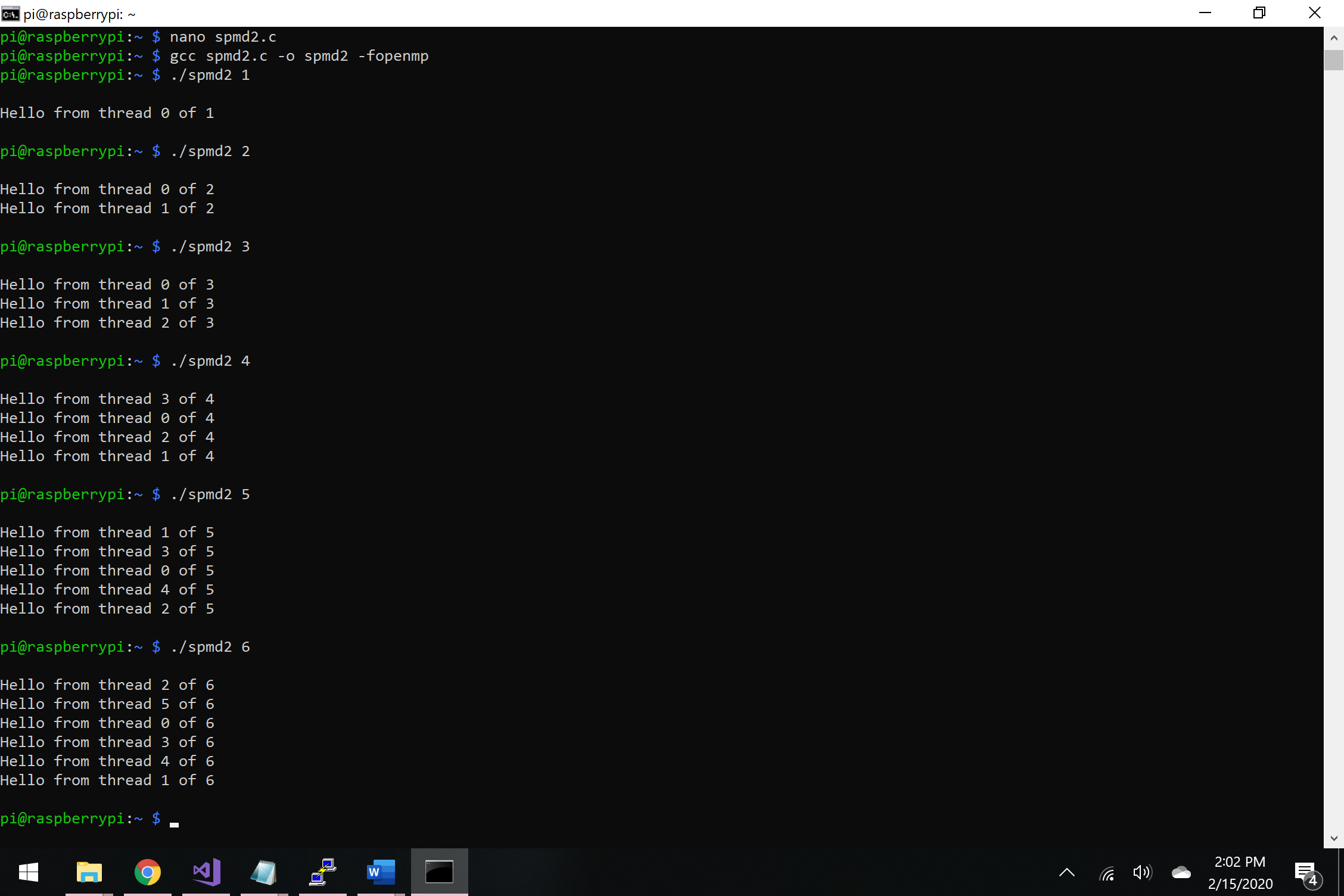


Figure 4 – fixed spmd2 code’s output