At first, we looked into the architecture of our machine and found out the following configuration in which we ran our code.

```
bob@lenovo-legion-5-15IMH05:~$ lscpu
Architecture:                    x86_64
CPU op-mode(s):                  32-bit, 64-bit
Byte Order:                      Little Endian
Address sizes:                   39 bits physical, 48 bits virtual
CPU(s):                          8
On-line CPU(s) list:             0-7
Thread(s) per core:              2
Core(s) per socket:              4
Socket(s):                       1
NUMA node(s):                    1
Vendor ID:                       GenuineIntel
CPU family:                      6
Model:                           165
Model name:                      Intel(R) Core(TM) i5-10300H CPU @ 2.50GHz
```

As we can see there are 4 cores in our system. So our machine can handle 4 processing jobs parallely. Hence, in general, if we want complete parallelization in the matrix multiplication, the following equation has to be satisfied,

$$r1 \ * \ c2 \ <= \ number\ of\ cores$$

Since, each core can run only one process at a time, the above needs to hold.

```
bob@lenovo-legion-5-15IMH05:~$ nproc
8
```

The maximum user processes (*nproc*) limit on Linux counts the number of threads within all processes that can exist for a given user

However, we should note one thing that the output of the above also shows that we have 8 CPUs. It might seem conflicting that we have 4 cores but 8 CPUs. So although the actual number of physical processing units is 4, the software shows we have 8 logical processing units. It is due to the hyperthreading that is done by modern architectures. So , the operating system distributes multiple tasks across multiple threads which are ready to be executed in a core.

Now, we have to take into account what happens when all the entries in the final matrix do not get computed in parallel. So, since we are creating a child process from the parent for each matrix entry, we would effectively create $r_1 * c_2$ processes in the long run. We checked the maximum number of default user processes allowed in our system.

```
bob@lenovo-legion-5-15IMH05:~$ ulimit -a
core file size          (blocks, -c) 0
data seg size           (kbytes, -d) unlimited
scheduling priority             (-e) 0
file size               (blocks, -f) unlimited
pending signals                 (-i) 63379
max locked memory       (kbytes, -l) 65536
max memory size         (kbytes, -m) unlimited
open files                      (-n) 1024
pipe size            (512 bytes, -p) 8
POSIX message queues     (bytes, -q) 819200
real-time priority              (-r) 0
stack size              (kbytes, -s) 8192
cpu time               (seconds, -t) unlimited
max user processes              (-u) 63379
virtual memory          (kbytes, -v) unlimited
file locks                      (-x) unlimited
bob@lenovo-legion-5-15IMH05:~$
```

As we can see, by default there is a restriction on the number of processes that can be created by the user. However, this is not fixed. We can vary it as a root user.

> */proc/sys/kernel/pid_max* (since Linux 2.5.34)
> This file specifies the value at which PIDs wrap around (i.e., the value in this file is one greater than the maximum PID). PIDs greater than this value are not allocated; thus, the value in this file also acts as a system-wide limit on the total number of processes and threads. The default value for this file, 32768, results in the same range of PIDs as on earlier kernels. On 32-bit platforms, 32768 is the maximum value for *pid_max*. On 64-bit systems, *pid_max* can be set to any value up to 2^22 (**PID_MAX_LIMIT**, approximately 4 million).

```
bob@lenovo-legion-5-15IMH05:~/Desktop/notebook$ cat /proc/sys/kernel/pid_max
4194304
```

The proc files for kernel indicate the maximum limit for process ids, under the name *pid_max*. This value as shown on the manual page for *proc* is set to 2^22 ~ 4 million for 64-bit systems. Therefore, we may be able to increase the user limit for processes to this limit in theory. Since we spawn a child for each r1 * c2 elements of the result matrix, this value is bound by 2^22 in 64-bit systems in UNIX.
Again, the *ulimit* (or the number of processes allowed for a single user) may be quite less than this and thus some superuser changes may have to be made to unlock this limit.