

Running Parallel Programming Data-Acquisition Scripts from a Windows Powershell



Oregon State
University
Mike Bailey

mjb@cs.oregonstate.edu



This work is licensed under a [Creative Commons
Attribution-NonCommercial-NoDerivatives 4.0
International License](#)



Oregon State
University
Computer Graphics

Here's How our Sample Code Sets NUMT and NUMTRIES

The sample code uses defined constants and global arrays, like this:

```
#ifndef NUMT  
#define NUMT 2  
#endif  
  
#ifndef NUMTRIALS  
#define NUMTRIALS 10000  
#endif  
  
float Heights[NUMTRIALS];
```

which you can then set from the command line in a script, like this:

```
#!/bin/bash  
# number of threads:  
for t in 1 2 4 6 8  
do  
    echo NUMT = $t  
    # number of trials:  
    for n in 1 10 100 1000 10000 100000 1000000  
    do  
        g++ -DNUMTRIALS=$n -DNUMT=$t prog.cpp -o prog -fopenmp  
        ./prog  
    done  
done  
| done
```

If You Want to Use Powershell From Windows, You Need to do it Differently

The sample code uses defined constants and global arrays, like this:

```
#ifndef NUMT  
#define NUMT 2  
#endif  
  
#ifndef NUMTRIALS  
#define NUMTRIALS 10000  
#endif  
  
float Heights[NUMTRIALS];
```

which you can then set from the command line in a script, like this:

```
#!/bin/bash  
# number of threads:  
for t in 1 2 4 6 8  
do  
    echo NUMT = $t  
    # number of trials:  
    for n in 1 10 100 1000 10000 100000  
    do  
        g++ -DNUMT=$t -DNUMTRIALS=$n prog.cpp -o prog -lm -fopenmp  
        ./prog  
    done  
done
```



If You Want to Use Powershell From Windows, We Need to do it Differently

1. Change the NUMT and NUMTRIES to global *int* variables:

```
int NUMT = 2;  
int NUMTRIALS = 10000;
```

2. Change the global arrays to be global pointers:

```
float *Height;
```

argc and argv

When you write in C or C++, your *main* program, which is really a special function, looks like this:

```
int main( int argc, char *argv[ ] )  
{  
    ...  
}
```

These arguments describe what was entered on the command line used to run the program.

The **argc** is the number of arguments (the arg **count**)

The **argv** is a list of argc character strings that were typed (the arg **vector**).

The name of the program counts as the 0th argv (i.e., argv[0])

So, for example, when you type

ls -l

in a shell, the *ls* program sees argc and argv filled like this:

```
argc = 2  
argv[0] = "ls"  
argv[1] = "-l"
```



argc and argv

So, NUMT and NUMTRIALS are global int variables with default values:

```
int NUMT = 2;  
int NUMTRIALS = 10000;
```

You want to set them from the command line, like this:

```
./prog 4 50000
```

Then, *inside your main program*, you would say this:

```
if( argc >= 2 )  
    NUMT = atoi( argv[1] );  
  
if( argc >= 3 )  
    NUMTRIALS = atoi( argv[2] );
```

The if-statements guarantee that nothing bad happens if you forget to type values on the command line.

The **atoi** function converts a string into an integer (“ascii-to-integer”).

If you ever need it, there is also an **atof** function for floating-point.



If You Want to Use Powershell From Windows, We Need to do it Differently

1. Change the NUMT and NUMTRIES to global *int* variables:

```
int NUMT = 2;  
int NUMTRIALS = 10000;
```

2. Change the global arrays to be global pointers:

```
float *Height;
```

3. In the main program, after you have set NUMT and NUMTRIALS using atoi, dynamically allocate the arrays, like this:

```
Height = new float [NUMTRIALS];
```



From then on, you can treat Heights as a normal array. For example:

```
float height = Heights[n];
```

shared() in the *#pragma omp* Line

Also, remember, since NUMTRIALS is a variable, it needs to be declared as *shared* in the *#pragma omp* line:

```
#pragma omp parallel for default(none) shared(NUMTRIALS,Height,Width) reduction(:numHits)
```

NUMT does not need to be declared in this way because it is not used in the for-loop that has the *#pragma omp* in front of it.



Windows Powershell

Windows comes with a shell program called *Powershell*. It might not be as familiar to most of us as some of the Linux shells are (bash, csh), but it can still be used to run multiple combinations of your program parameters in one shot.

There are a number of ways to get Powershell running. Either:

- Click on the Microsoft icon. Then scroll down to **Windows Powershell** and run **Windows Powershell**.
- *Shift right-click* in the directory you want to work in and select **Open Powershell Window**.
- Hold down the Windows key and hit the ‘x’ key, then select **Windows Powershell**.



The resulting window should look like this:

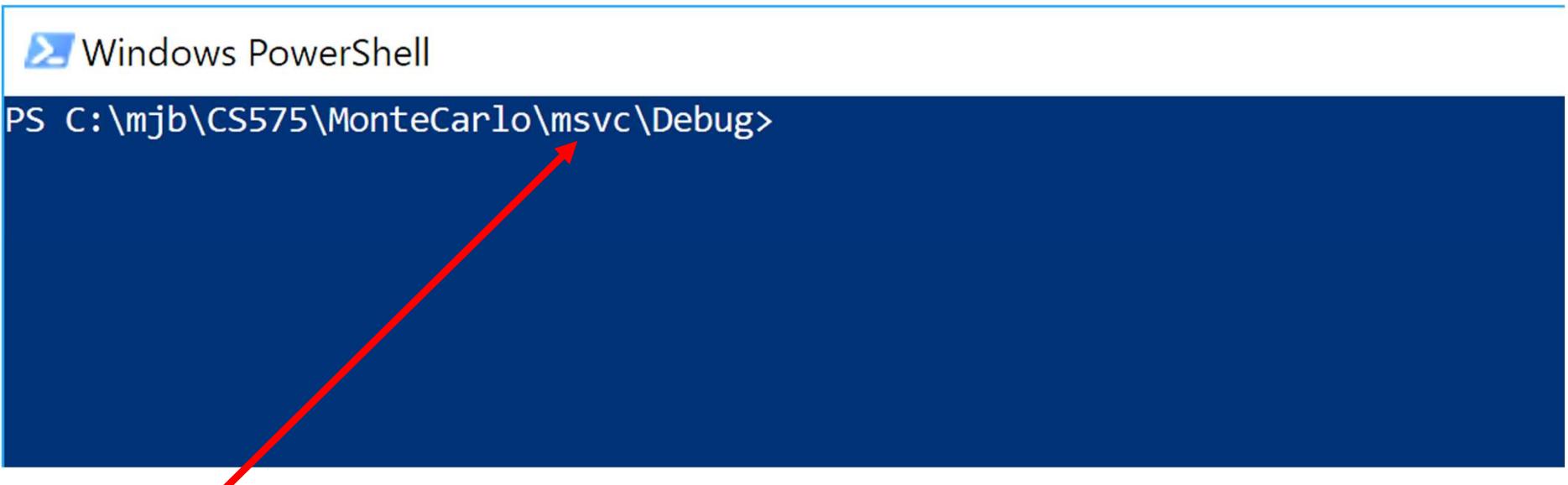
```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

PS C:\Users\Mike Bailey>
```

Change Directory to Where Your .exe File Lives

Then:

1. `cd` (change directory) to your home directory.
2. Then `cd` to the folder with your project
3. Then `cd` to the folder with your executable (*.exe)
(for Visual Studio, this is usually a folder named **Debug**)



```
Windows PowerShell

PS C:\mjb\CS575\MonteCarlo\msvc\Debug>
```

The prompt will always tell you where you are in the file system.



Running an Executable

So, if you have `cd`'ed to where your executable (.exe) file lives, you can run it from the command line like this:

```
Windows PowerShell
PS C:\mjb\CS575\MonteCarlo\msvc\Debug> ls          Ask to see the files in this folder
Directory: C:\mjb\CS575\MonteCarlo\msvc\Debug

Mode                LastWriteTime      Length Name
----                -----          -
-a---        4/13/2020 8:28 PM       47164 MonteCarlo.exe   Yup, here it is
-a---        4/13/2020 8:28 PM      403468 MonteCarlo.ilk
-a---        4/13/2020 8:28 PM      626688 MonteCarlo.pdb

PS C:\mjb\CS575\MonteCarlo\msvc\Debug> ./MonteCarlo.exe Execute it
2 threads ;    50000 trials ; megatrials/sec = 57.42
PS C:\mjb\CS575\MonteCarlo\msvc\Debug> -
```

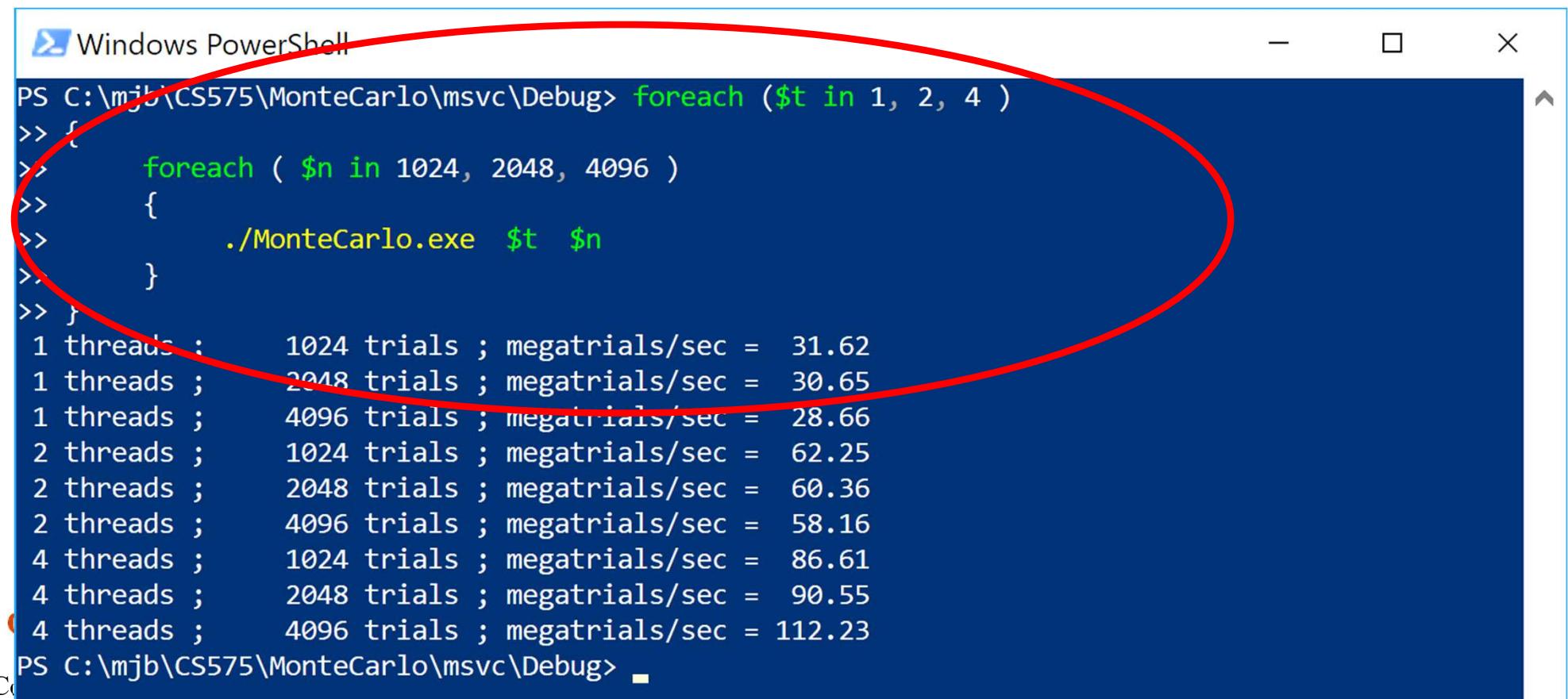


Running a Loop

But, here's the cool part. Type:

```
foreach ( $t in 1, 2, 4 )
{
    foreach ( $n in 1024, 2048, 4096)
    {
        ./MonteCarlo.exe $t $n
    }
}
```

followed by Enter:

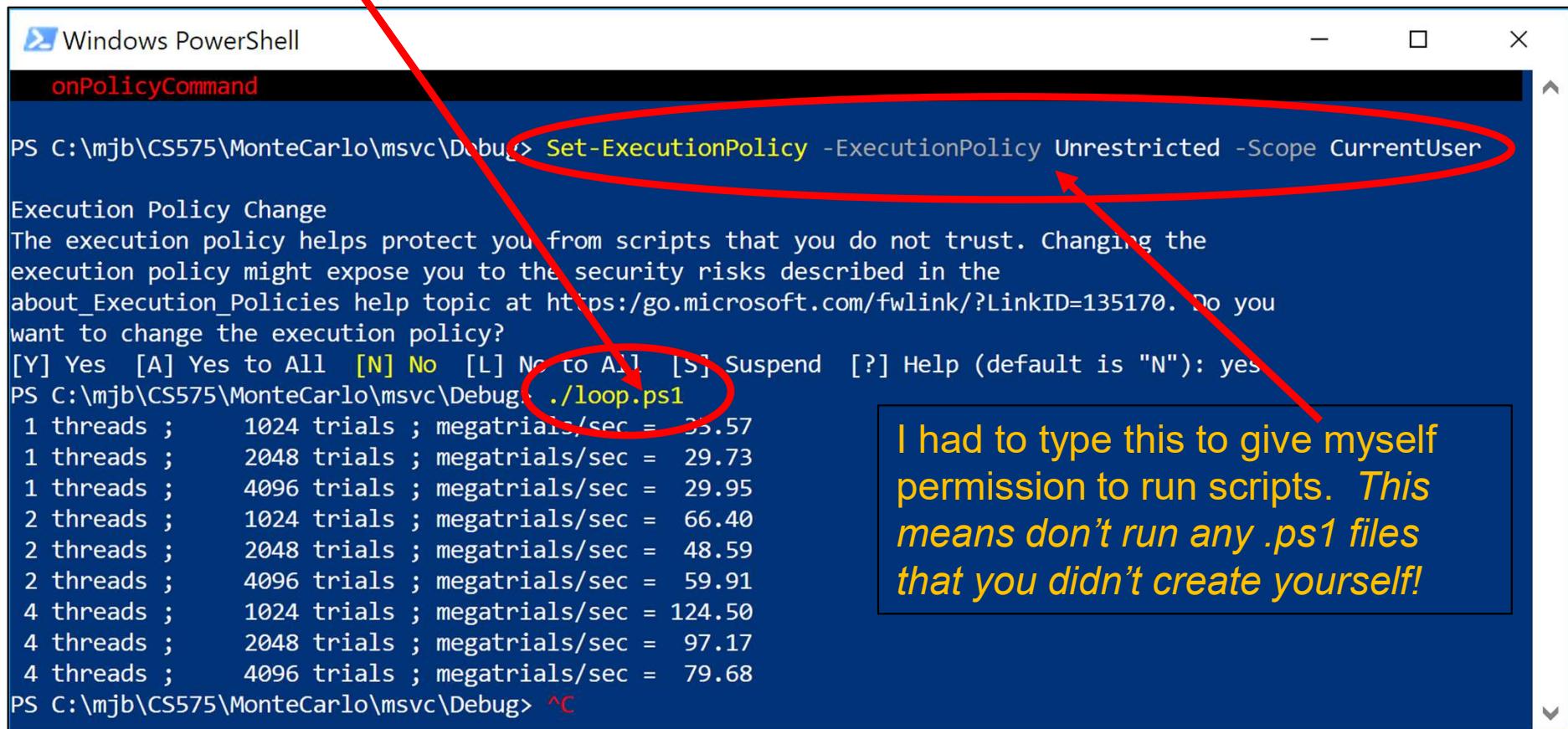


```
Windows PowerShell
PS C:\mjb\CS575\MonteCarlo\msvc\Debug> foreach ($t in 1, 2, 4 )
>> {
>>     foreach ( $n in 1024, 2048, 4096 )
>>     {
>>         ./MonteCarlo.exe $t $n
>>     }
>> }
1 threads ;      1024 trials ; megatrials/sec = 31.62
1 threads ;      2048 trials ; megatrials/sec = 30.65
1 threads ;      4096 trials ; megatrials/sec = 28.66
2 threads ;      1024 trials ; megatrials/sec = 62.25
2 threads ;      2048 trials ; megatrials/sec = 60.36
2 threads ;      4096 trials ; megatrials/sec = 58.16
4 threads ;      1024 trials ; megatrials/sec = 86.61
4 threads ;      2048 trials ; megatrials/sec = 90.55
4 threads ;      4096 trials ; megatrials/sec = 112.23
PS C:\mjb\CS575\MonteCarlo\msvc\Debug>
```

Running a Loop from a File

You can also use a text editor like *notepad* or *notepad++* and put these lines into a file called, say, **loop.ps1** (ps1 is the Powershell file extension).

Then, you can run this script from Powershell just by typing it:



A screenshot of a Windows PowerShell window titled "Windows PowerShell" with the title bar "onPolicyCommand". The command entered is "Set-ExecutionPolicy -ExecutionPolicy Unrestricted -Scope CurrentUser". A red circle highlights the command. A large red X is drawn over the entire window, indicating that this method is not recommended.

```
PS C:\mjb\CS575\MonteCarlo\msvc\Debug> Set-ExecutionPolicy -ExecutionPolicy Unrestricted -Scope CurrentUser
Execution Policy Change
The execution policy helps protect you from scripts that you do not trust. Changing the
execution policy might expose you to the security risks described in the
about_Execution_Policies help topic at https://go.microsoft.com/fwlink/?LinkID=135170. Do you
want to change the execution policy?
[Y] Yes [A] Yes to All [N] No to All [S] Suspend [?] Help (default is "N"): yes
PS C:\mjb\CS575\MonteCarlo\msvc\Debug> ./loop.ps1
1 threads ; 1024 trials ; megatrials/sec = 35.57
1 threads ; 2048 trials ; megatrials/sec = 29.73
1 threads ; 4096 trials ; megatrials/sec = 29.95
2 threads ; 1024 trials ; megatrials/sec = 66.40
2 threads ; 2048 trials ; megatrials/sec = 48.59
2 threads ; 4096 trials ; megatrials/sec = 59.91
4 threads ; 1024 trials ; megatrials/sec = 124.50
4 threads ; 2048 trials ; megatrials/sec = 97.17
4 threads ; 4096 trials ; megatrials/sec = 79.68
PS C:\mjb\CS575\MonteCarlo\msvc\Debug> ^C
```

I had to type this to give myself permission to run scripts. This means don't run any .ps1 files that you didn't create yourself!

Instead of printing these lines to the screen, you probably want to divert them to a CSV file that can then be imported by Excel.

Diverting Output to an Excel CSV File from Powershell

To divert just the printf's to a file, do this:

```
./loop.ps1 > out.csv
```

or this:

```
./loop.ps1 1> out.csv
```

To divert *both* printf's and fprintf(stderr,...)'s together, do this:

```
./loop.ps1 2>&1> out.csv
```

Use this because the sample code does a lot of printing to stderr

