

Parallelizing the calculation of π on the GPU with OpenMP/OpenACC

Goals:

- Build & run an OpenMP code on the GPU (for problem size $N=900000000$)
- Build & run an OpenACC code on the GPU (for problem size $N=900000000$)

Parallelware Trainer

Project Explorer

Code Editor

Version Manager

The screenshot displays the Parallelware Trainer IDE interface. On the left, the Project Explorer shows a project named 'pi' with files 'Makefile', 'pi.c', and 'README.md'. The main area contains two code editors. The left editor shows the source code for 'pi.c', which includes a main function and a 'getClock' helper function. The right editor shows the original source code for comparison. Below the code editors is the Output Console, which displays the results of a parallelization analysis. The console output includes messages such as 'Parallelizing...', 'Parallel scalar reduction pattern identified for variable 'sum'', and 'Analysis completed: 0 opportunities found'. At the bottom of the IDE, there are icons for Build output, Execution output, and Parallelware output, along with a status indicator 'Analysis completed'.

```
pi.c
35 out_result = 4.0 / N * sum;
36
37 // =====
38 double time_finish = getClock();
39
40 // Prints an execution report
41 printf("time (s)= %.6f\n", time_finish - time_start);
42 printf("result\t= %.8f\n", out_result);
43 const double realPiValue = 3.141592653589793238;
44 printf("error\t= %.1e\n", fabs(out_result - realPiValue));
45
46 return 0;
47 }
48
49 double getClock() {
50 #ifdef _OPENMP
51 #include <omp.h>
52 return omp_get_wtime();
53 #elif __linux__ || __APPLE__
54 #include <time.h>
55 struct timespec ts;
56 clock_gettime(CLOCK_MONOTONIC, &ts);
57 return ts.tv_sec + ts.tv_nsec / 1.0e9;
58 #else
59 #include <time.h>
60 return (double)clock() / CLOCKS_PER_SEC;
61 #endif
62 }
63
64
```

```
Original 1
32 out_result = 4.0 / N * sum;
33
34 // =====
35 double time_finish = getClock();
36
37 // Prints an execution report
38 printf("time (s)= %.6f\n", time_finish - time_start);
39 printf("result\t= %.8f\n", out_result);
40 const double realPiValue = 3.141592653589793238;
41 printf("error\t= %.1e\n", fabs(out_result - realPiValue));
42
43 return 0;
44 }
45
46 double getClock() {
47 #ifdef _OPENMP
48 #include <omp.h>
49 return omp_get_wtime();
50 #elif __linux__ || __APPLE__
51 #include <time.h>
52 struct timespec ts;
53 clock_gettime(CLOCK_MONOTONIC, &ts);
54 return ts.tv_sec + ts.tv_nsec / 1.0e9;
55 #else
56 #include <time.h>
57 return (double)clock() / CLOCKS_PER_SEC;
58 #endif
59 }
60
```

```
[12:05:56] Parallelizing...
pi.c line 27: Parallel scalar_reduction pattern identified for variable 'sum' with associative, commutative operator '+'
pi.c line 27: Available parallelization strategies for variable 'sum'
pi.c line 27: #1 OpenMP scalar_reduction (+ implemented)
pi.c line 27: #2 OpenMP atomic_access
pi.c line 27: #3 OpenMP explicit privatization
pi.c line 27: Loop parallelized with multithreading using OpenMP directive 'for'
pi.c line 27: Parallel_region defined by OpenMP directive 'parallel'
pi.c line 27: Make sure there is no aliasing among arguments in 'main': argc, argv
[12:05:56] Parallelization completed successfully
[12:05:57] Analysis completed: 0 opportunities found
```



Output Consoles

Run Parallelware Trainer on CORI

Step 1: Log in to CORI enabling X11 forwarding

```
$ ssh -X <your_login>@cori.nersc.gov
```

Step 2: Copy the sample codes to be used during the course

```
$ cp -a /global/common/cori/software/pwtrainer-1.2.0_linux-x64/examples.zip ~
```

Step 3: Prepare the environment by loading the appropriate modules

```
$ module purge && module load esslurm cuda/10.0 gcc/8.1.1-openacc-gcc-8-branch-20190215 pgi/19.4 pwtrainer
```

Step 4: Open an interactive session in a GPU node

```
$ salloc -t 60 -N 1 -c 16 -C gpu --gres=gpu:2 --mem=32GB
```



Step 5: Run the Parallelware Trainer tool from the GPU node

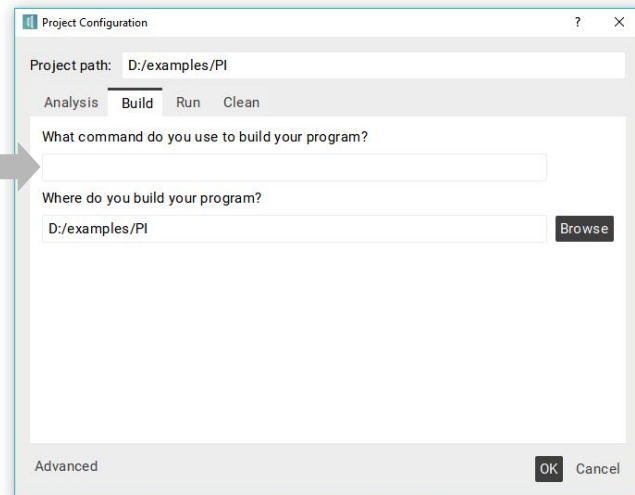
```
$ pwtrainer &
```

Step 6: Activate Parallelware Trainer

Click on 'Click here to install a license' and enter the following in the 'File name' text box:
`/global/common/cori/software/pwtrainer-1.2.0_linux-x64/pwtrainer-Jun19-NERSC.lic`



Build in Parallelware Trainer

		
GCC	<pre>gcc -fopenmp -foffload=nvptx-none="-Ofast -lm -misa=sm_35" -lm -o pi pi.c</pre>	<pre>gcc -fopenacc -foffload=nvptx-none="-Ofast -lm -misa=sm_35" -lm -D_OPENMP -o pi pi.c</pre>
PGI	<pre>pgcc -mp -ta=tesla:cc70,cuda10.0 -Minfo=accel -lm -o pi pi.c</pre>	<pre>pgcc -acc -ta=tesla:cc70,cuda10.0 -Minfo=accel -lm -D_OPENMP -o pi pi.c</pre>

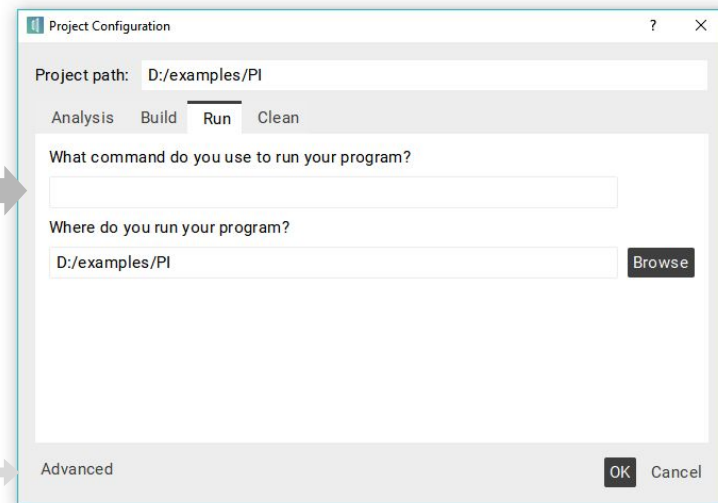


N.B. For Macs, check that your compiler supports OpenMP/OpenACC, and update with the appropriate compiler (e.g. gcc-mp-7)

Run in Parallelware Trainer

	<pre>srun env OMP_NUM_THREADS=4 ./pi 900000000</pre>
	<pre>srun env ACC_DEVICE_TYPE=nvidia ./pi 900000000</pre>

N.B. Instead of using the `env` command, you can add this variable by clicking the `Advanced` button.



Decomposition of PI into Patterns

Code file "pi.c"		Pattern			
Function	Line	Forall	Scalar reduction	Sparse reduction	Convergence loop
main()	29		sum		