

Parallelizing MATrix MULtiplication on the GPU with OpenMP/OpenACC



Goals:

- Build & run an OpenMP code on the GPU (for problem size $N=5000$)
- Build & run an OpenACC code on the GPU (for problem size $N=5000$)
 - Increase parallelism by collapsing loops
 - Choose the better memory layout for optimized memory transfers to the GPU

Parallelware Trainer

Project Explorer

Code Editor

Version Manager

The screenshot displays the Parallelware Trainer IDE interface. On the left is the Project Explorer showing 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 that calculates the sum of 1 to N and prints the result and execution time. It also defines a 'getClock()' function to measure time. The right editor shows the 'Original 1' version of the same code. Below the code editors is the Output Console, which displays the results of a parallelization analysis. The analysis identifies several optimization opportunities, such as scalar reduction and loop parallelization, and reports that the parallelization completed successfully.

```
pi.c
36 out_result = 4.0 / N * sum;
37
38 // =====
39 double time_finish = getClock();
40
41 // Prints an execution report
42 printf("time (s)= %.6f\n", time_finish - time_start);
43 printf("result\t= %.8f\n", out_result);
44 const double realPiValue = 3.141592653589793238;
45 printf("error\t= %.1e\n", fabs(out_result - realPiValue));
46
47 return 0;
48 }
49
50 double getClock() {
51 #ifdef _OPENMP
52 #include <omp.h>
53 return omp_get_wtime();
54 #elif __linux__ || __APPLE__
55 #include <time.h>
56 struct timespec ts;
57 clock_gettime(CLOCK_MONOTONIC, &ts);
58 return ts.tv_sec + ts.tv_nsec / 1.0e9;
59 #else
60 #include <time.h>
61 return (double)clock() / CLOCKS_PER_SEC;
62 #endif
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_cle7/software/pwtrainer/Workshop-Oct19-examples.zip ~
```

Step 3: Prepare the environment by loading the appropriate modules

```
$ module purge && module load esslurm cuda gcc/8.1.1-openacc-gcc-8-branch-20190215 pgi/19.9 pwtrainer
```

Step 4: Open an interactive session in a GPU node
(*)

```
$ salloc -t 60 -N 1 -c 8 -C gpu --gres=gpu:1 --mem=32GB --reservation=trainer -A nintern
```

(*) For CPU nodes, use --reservation=trainer_knl for the training

(*) Some people might need to use -A nstaff or -A m1759 instead of -A nintern

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

```
$ pwtrainer &
```



Memory layout versions

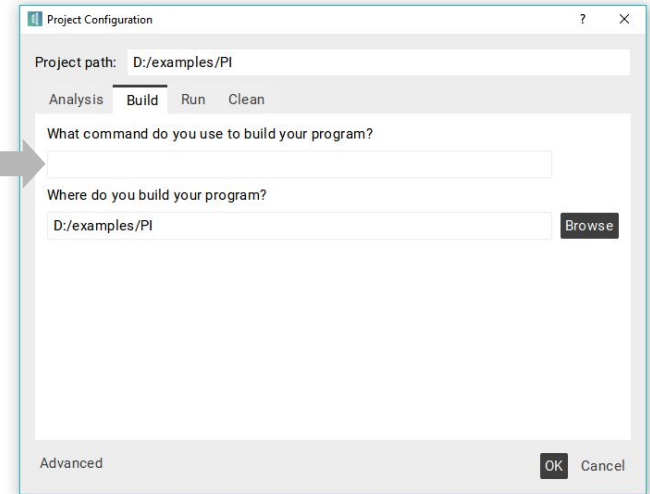
Three MATMUL versions differing in how 2D matrices are implemented using dynamic memory

Dynamic memory is preferred because it uses the **heap**,
unlike static memory which may consume the limited **stack** memory!

Version	Data structure	Memory layout	Array syntax
matmul_v1_dynarray2d_noncont.c	Dynamic array of pointers to dynamic 1D arrays (double**)	NON contiguous	A[i][j]
matmul_v2_dynarray2d.c	Dynamic array of pointers to locations of a dynamic 1D array containing the linearized data (double**)	Contiguous	A[i][j]
matmul_v3_dynarray1d.c	Dynamic 1D array containing the linearized data (double*)	Contiguous	A[i * cols + j]



Build in Parallelware Trainer

		
GCC	<pre>gcc -fopenmp -foffload=nvptx-none="-Ofast -lm -misa=sm_35" -o matmul matmul_v3_dynarray1d.c</pre>	<pre>gcc -fopenacc -foffload=nvptx-none="-Ofast -lm -misa=sm_35"-D_OPENMP -o matmul matmul_v3_dynarray1d.c</pre>
PGI	<pre>pgcc -mp -ta=tesla -Minfo=accel -o matmul matmul_v3_dynarray1d.c</pre>	<pre>pgcc -acc -ta=tesla -Minfo=accel -D_OPENMP -o matmul matmul_v3_dynarray1d.c</pre>

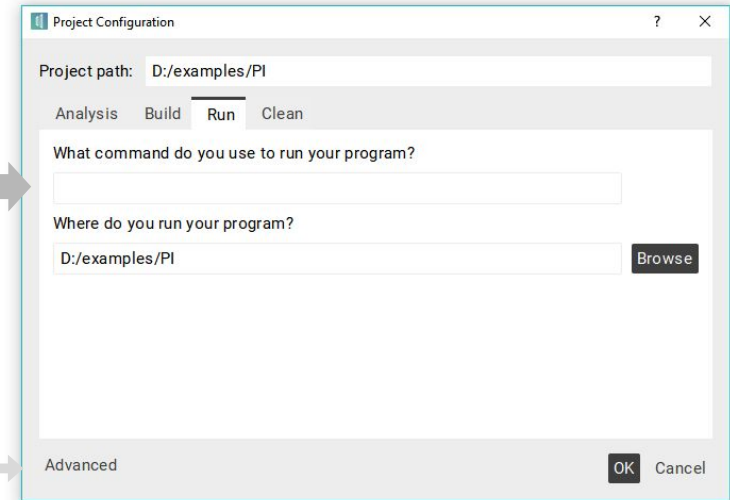


N.B. For Macs, check that your compiler supports OpenMP/OpenACC, and update with the appropriate compiler (e.g. gcc-mp-7)

Run in Parallelware Trainer

	<pre>srunc env OMP_NUM_THREADS=4 ./matmul 5000</pre>
	<pre>srunc env ACC_DEVICE_TYPE=nvidia ./matmul 5000</pre>

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



Decomposition in parallel patterns

Code file "matmul_v3_dynarray1d.c"		Pattern			
Function	Line	Forall	Scalar reduction	Sparse reduction	Convergence loop
<u>matmul()</u>	22	C			
	32	C			
main()					