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

Goals:

- Build & run an OpenMP code on the GPU (for problem size N=100000)
- Build & run an OpenACC code on the GPU (for problem size N=100000)
    - Using OpenACC *kernels* directive for automatic parallelization (manually, not supported in Parallelware Trainer)
    - Using OpenACC *loop* directives for finer control
    - Using a single *data* directive enclosing all *loop* ones
    - Using a single *data* directive for the program

Parallelware Trainer

Project Explorer

Code Editor

Version Manager

File Edit Project Help

PI

- Makefile
- pi.c
- README.md

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
```

Build output | Execution output | Parallelware output

Analysis completed

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 &
```

Build in Parallelware Trainer

	OpenMP (Enabling HPC since 1997)	OpenACC (More Science, Less Programming)
GCC	`gcc -fopenmp -foffload=nvptx-none="-Ofast -lm -misa=sm_35" -lm -o heat heat.c`	`gcc -fopenacc -foffload=nvptx-none="-Ofast -lm -misa=sm_35" -lm -D_OPENMP -o heat heat.c`
PGI	`pgcc -mp -ta=tesla -Minfo=accel -lm -o heat heat.c`	`pgcc -acc -ta=tesla -Minfo=accel -lm -D_OPENMP -o heat heat.c`

Project Configuration

Project path: D:/examples/PI

Analysis | Build | Run | Clean

What command do you use to build your program?

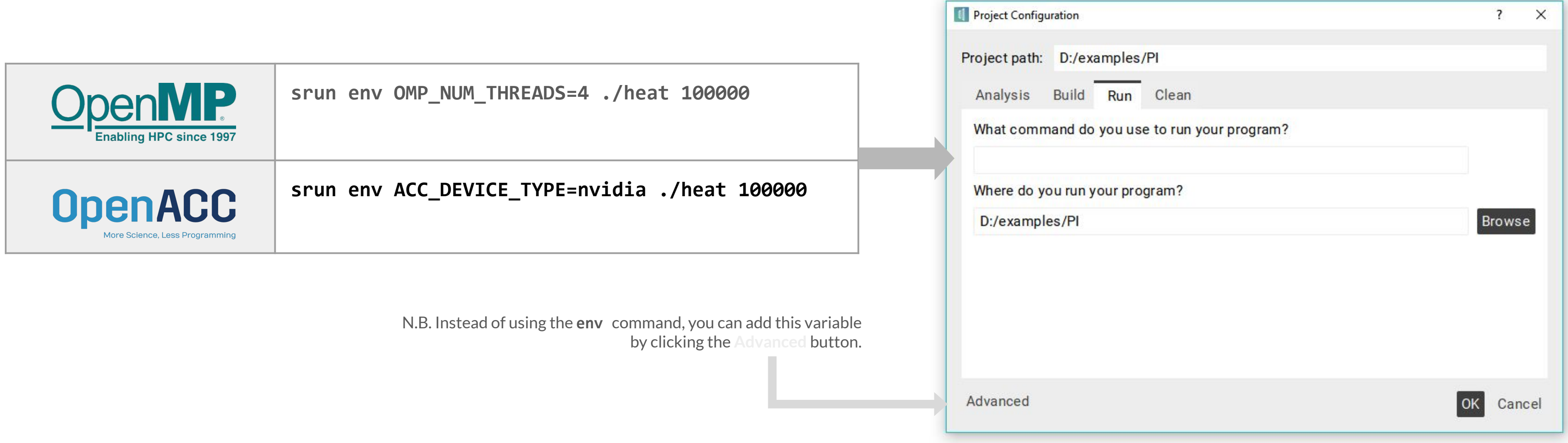
Where do you build your program?

D:/examples/PI [Browse]

Advanced [OK] [Cancel]

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



Decomposition in parallel patterns

Code file "heat.c"		Pattern			
Function	**Line**	**Forall**	**Scalar reduction**	**Sparse reduction**	**Convergence loop**
compute()	25	T			
	28			T	
	33		maxdiff		
	39	uk			
initialize_uk()	49				
initialize_T()	54				
	56				
cfd_heat_diffusion()	73				
main()					
calculate_checksum()	133				