

# Parallelware Training Series

## Motif-guided Parallelization of ZPIC with OpenMP and OpenACC

Run Parallelware Trainer and Analyzer on CORI at NERSC  
(Additional materials for NERSC users)

# Run Parallelware tools on CORI (an example with GCC)

## Step 1: Log in to CORI using [NoMachine / NX](#)

Alternative you may use SSH with X11 forwarding (although it will be much slower):

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

It is recommended to use other compilers with your application for performance reasons.

- For OpenMP on GPU, use “LLVM”
- For OpenACC on GPU, use “HPCSDK”

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

```
$ cd $SCRATCH
```

```
$ cp -r /global/cfs/cdirs/training/2020/parallelware .
```

Notice the dot at the end

## Step 3: Prepare the environment by loading the appropriate modules

```
$ module load cgpu cuda gcc/8.1.1-openacc-gcc-8-branch-20190215 pwtrainer pwanalyzer
```

## Step 4: Open an interactive session in a GPU node

(\*)

```
$ salloc -C gpu -N 1 -t 60 -c 4 -G 1 -A m3502 --reservation=pwtools1_gpu -q shared
```

(\*) Outside of trainings, for GPU use: `salloc -C gpu -N 1 -t 60 -c 4 -G 1 -A m3502 -q interactive`

(\*) For CPU nodes, do not “module load cgpu”, and use: `snv-gnu`

(\*) Outside of trainings for CPU, do not “module load cgpu”, use: `salloc -C knl -N 1 -t 60 -q interactive`

## Step 5: Run the Parallelware tools from the node

```
$ pwtrainer &
```

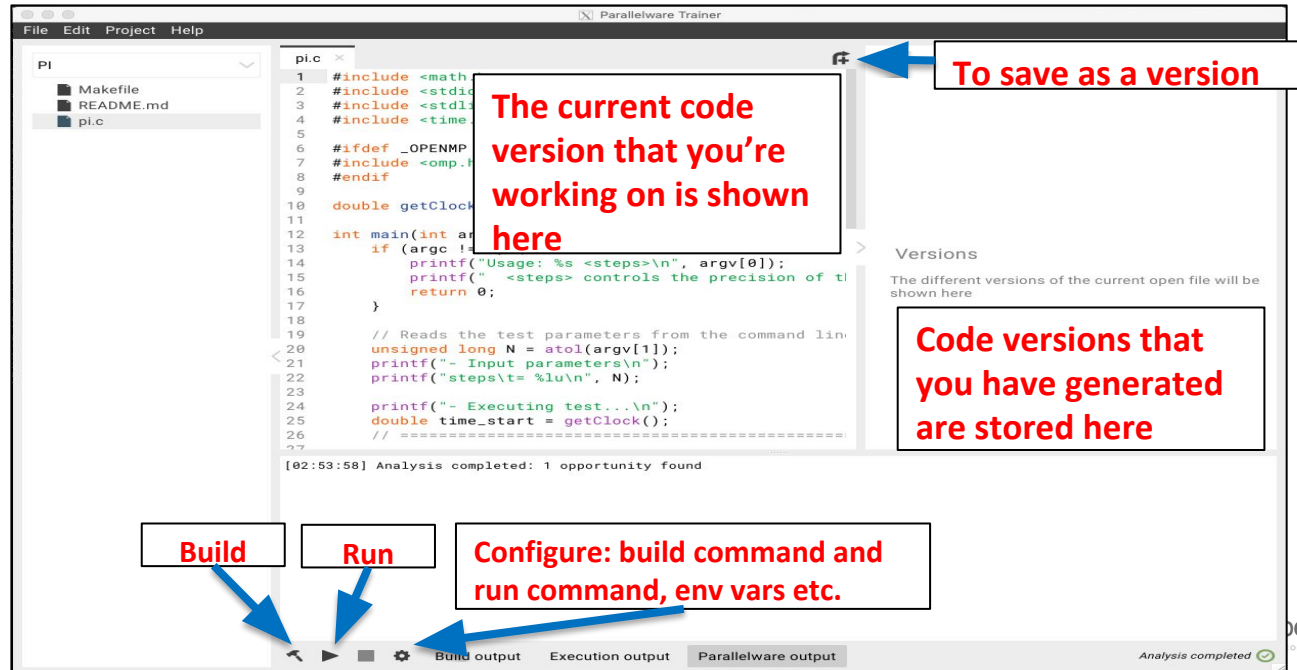
```
$ pwreport --help
```

# Intro to Parallelware Trainer

- **“Trainer” GUI tool developed by Appentra** (<https://www.appentra.com>)
  - OpenMP and OpenACC beginners can learn about quick coding changes they can make in their codes
- **It detects code segments that can be parallelized with OpenMP and OpenACC, and provides necessary directives:**
  - Implementation of reductions provided in 3 flavors (built-in reduction, atomic operation & explicit privatization for the reduction variable)
  - Offloading supported
  - Deferred support for some parallel (sparse) scatter operations
  - Current support limited to C codes – C++ and Fortran support to come
  - Some notable features not being explored (e.g., loop collapsing)
- **Based on static code pattern analysis**
  - Profiling not part of the tool workflow
  - Users need to profile with a profiler of own choice, to identify hot spots or to evaluate the resulting performance
  - Users are expected to work further for optimizations (memory use optimization, chunk scheduling, loop collapsing, etc.)

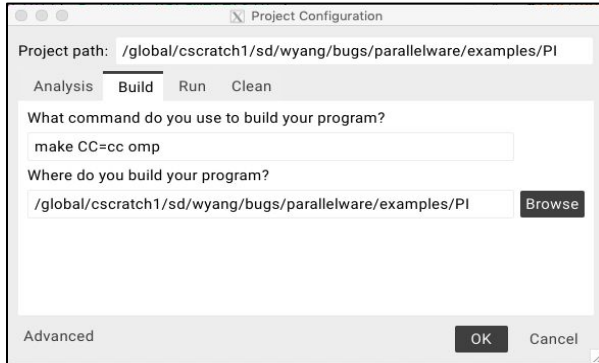
# How to use Parallelware Trainer

- Start an interactive job and launch *pwtrainer*
- Open a new project (File □ Open Project)
  - and select a directory where you want to run pwtrainer for the codes there
- You will see...



# How to use Parallelware Trainer (cont'd)

- Set configuration for build, run and clean, ... and also for analysis by Parallelware.



Project path: /global/cscratch1/sd/wyang/bugs/parallelware/examples/PI

Analysis Build Run Clean

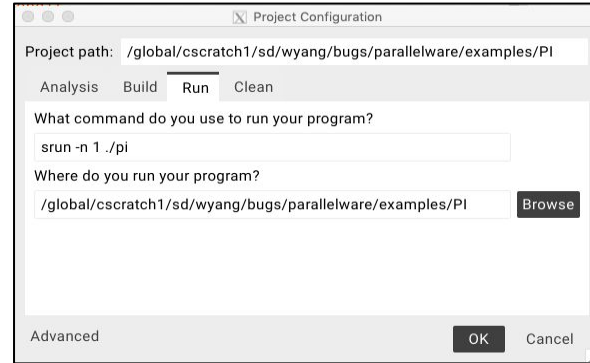
What command do you use to build your program?

make CC=cc omp

Where do you build your program?

/global/cscratch1/sd/wyang/bugs/parallelware/examples/PI Browse

Advanced OK Cancel



Project path: /global/cscratch1/sd/wyang/bugs/parallelware/examples/PI

Analysis Build Run Clean

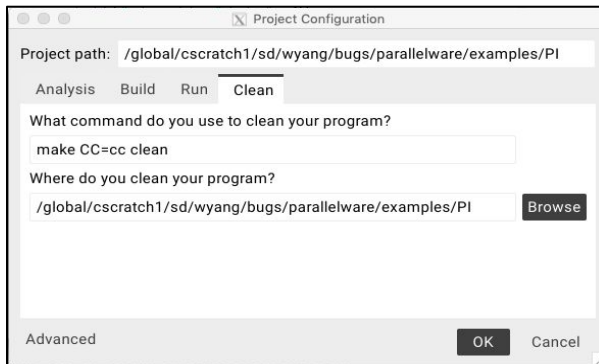
What command do you use to run your program?

srun -n 1 ./pi

Where do you run your program?

/global/cscratch1/sd/wyang/bugs/parallelware/examples/PI Browse

Advanced OK Cancel



Project path: /global/cscratch1/sd/wyang/bugs/parallelware/examples/PI

Analysis Build Run Clean

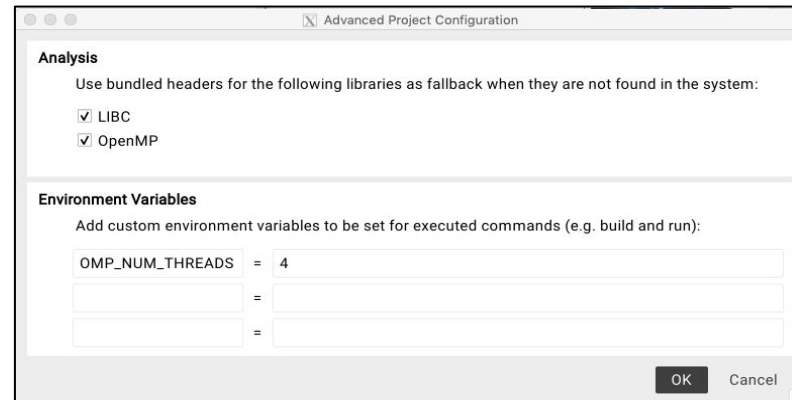
What command do you use to clean your program?

make CC=cc clean

Where do you clean your program?

/global/cscratch1/sd/wyang/bugs/parallelware/examples/PI Browse

Advanced OK Cancel



Advanced Project Configuration

**Analysis**

Use bundled headers for the following libraries as fallback when they are not found in the system:

☒ LIBC

☒ OpenMP

**Environment Variables**

Add custom environment variables to be set for executed commands (e.g. build and run):

OMP\_NUM\_THREADS = 4

=

=

OK Cancel

# How to use Parallelware Trainer (cont'd)

- Parallelize a code section that the tool has identified
  - Green circle icon** in the source pane highlights **"Parallelization Opportunities"**
  - Click the icon and set parallelization option details

```

pi.c x
19 // Reads the test parameters from the command l
20 unsigned long N = atol(argv[1]);
21 printf("- Input parameters\n");
22 printf("steps\t= %lu\n", N);
23
24 printf("- Executing test...\n");
25 double time_start = getClock();
26 // =====
27
28 double out_result;
29
30 double sum = 0.0;
31 for (int i = 0; i < N; i++) {
32     double x = (i + 0.5) / N;
33     sum += sqrt(1 - x * x);
34 }
35
36 out_result = 4.0 / N * sum;
37
38 // =====
39 double time_finish = getClock();
    
```

Parallelization options

**Standard**

☒ OpenMP  
☐ OpenACC

**Device**

☒ CPU  
☐ GPU

**Paradigm**

☒ Multithreading  
☐ Offloading  
☐ Tasking with taskloop

**Parallel reduction variables**

Atomic protection x, y, ...  
Built-in reduction x, y, ...  
Explicit privatization x, y, ...

**Ranges for array variables**

Array ranges x[0:100], y[N:M], ...

Data Scoping Cancel **Parallelize**

```

pi.c x
19 // Reads the test parameters from the c
20 unsigned long N = atol(argv[1]);
21 printf("- Input parameters\n");
22 printf("steps\t= %lu\n", N);
23
24 printf("- Executing test...\n");
25 double time_start = getClock();
26 // =====
27
28 double out_result;
29
30 double sum = 0.0;
31 #pragma omp parallel default(none) shar
32 {
33     #pragma omp for reduction(+: sum) sched
34     for (int i = 0; i < N; i++) {
35         double x = (i + 0.5) / N;
36         sum += sqrt(1 - x * x);
37     }
38 } // end parallel
39
    
```

# How to use Parallelware Trainer (cont'd)

- You can edit in the source pane what the tool suggested
- If you are satisfied, click the build button at the bottom to build
- To run, click the run button
- You can save this version by clicking the up-right arrow at the top right corner of the current code pane
  - This will create a version in the version pane
  - To bring a previous version to the current version, choose the tab for the corresponding version and click the up-left arrow
- You repeat the above processes with different parallelization options (and offload option, too) and implementation options for your needs

# How to use Parallelware Analyzer

- **pwreport** provides a high-level overview of your code: summary of parallelized regions, defects, recommendations and opportunities..
- **pwcheck** looks for **defects** such as race-conditions and issues **recommendations** on best-practices and performs **data-race analysis**.
- **pwloops** provides insight into the **parallel properties of loops** found in the code which constitute opportunities for parallelism.
- **pwdirectives** provides guided generation of parallel code for multicore CPUs and GPUs, with **OpenMP** and **OpenACC**, using multithreading, offloading and tasking.



# Parallelware Training Series

Motif-guided Parallelization of  
ZPIC with OpenMP and  
OpenACC



Introduction to Parallelware tools:  
Ensuring parallel programming best  
practices

Exercises