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)

Manuel Arenaz | Oct-Nov 2020

NERSC

appentra
make code parallel

OpenACC
More Science, Less Programming

OpenMP
Enabling HPC since 1997

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

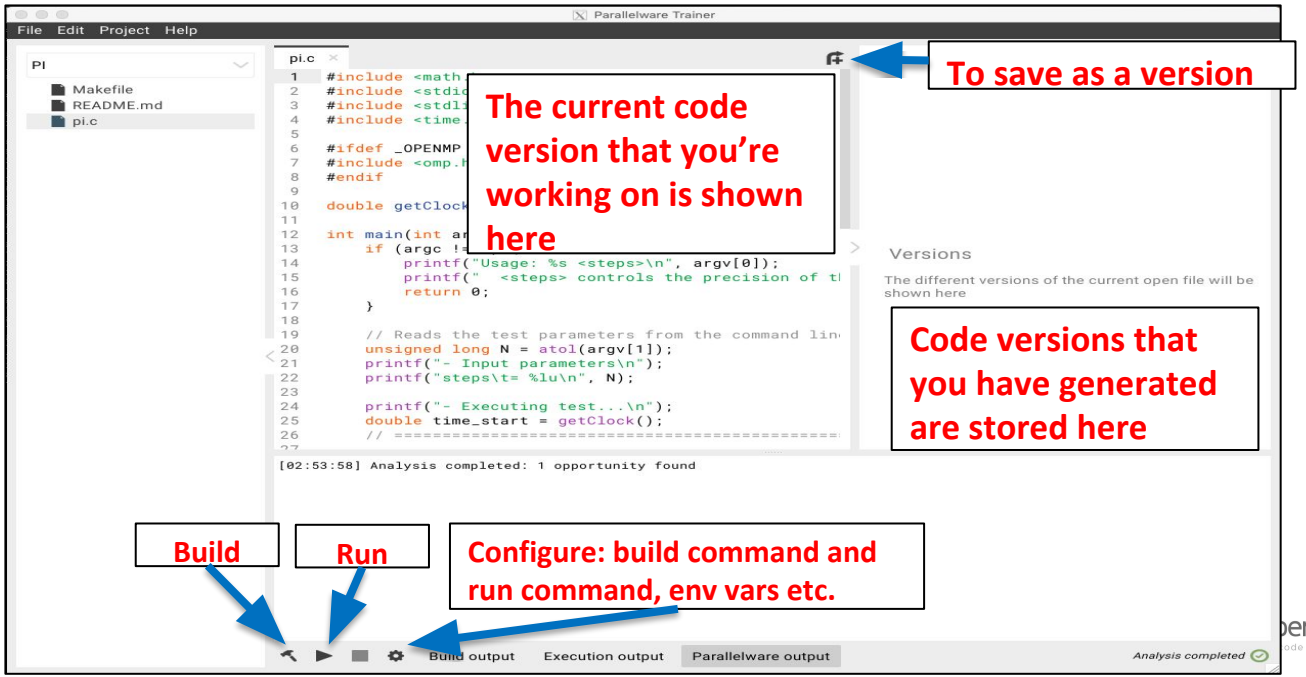
```
$ pwtrainer &
$ pwreport --help
```

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.)

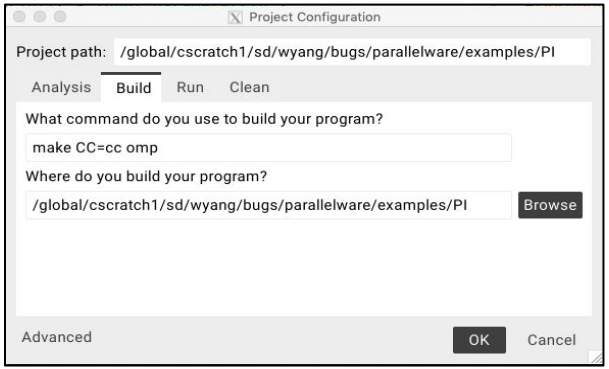
How to use Parallelware Trainer

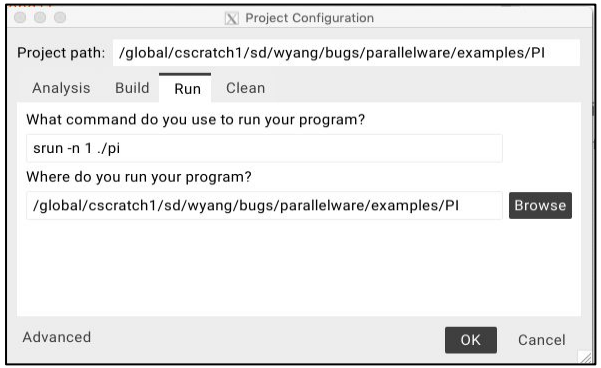
- Start an interactive job and launch *pwtrainer*
- Open a new project (File □ Open Project)
  - and select a directory where you want to run pwtrainer for the codes there
- You will see...

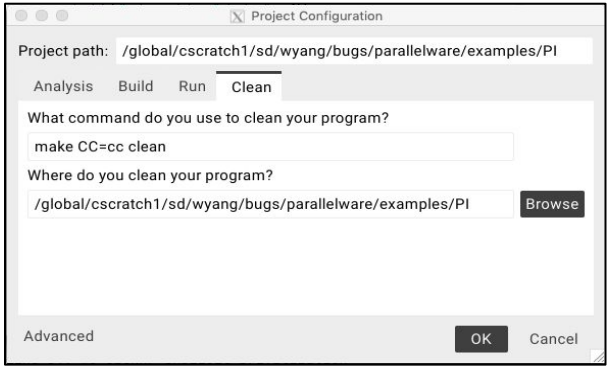


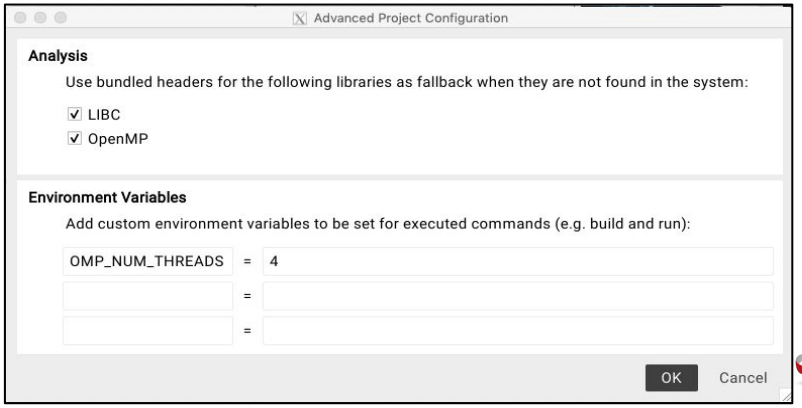
How to use Parallelware Trainer (cont'd)

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



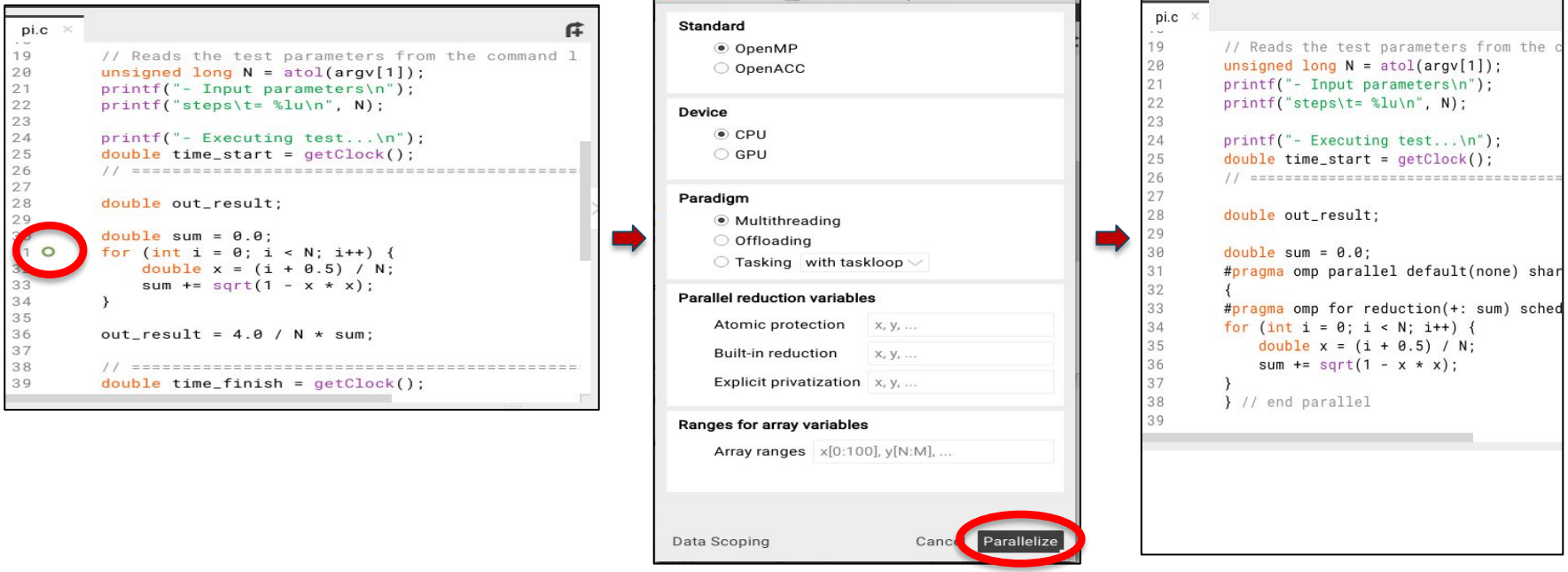






How to use Parallelware Trainer (cont'd)

- 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



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

NERSC

appentra
make code parallel

Introduction to Parallelware tools: Ensuring parallel programming best practices

Exercises