



USER MANUAL FOR V1.3



www.appentra.com

1. Introduction	5
1.1 How does Parallelware Trainer work?	5
1.2 How can Parallelware Trainer help me?	5
1.3 List of key features	5
1.4 Software requirements	6
1.4.1 Linux	6
1.4.2 MacOS	7
1.4.3 Windows	7
1.5 Contacting the authors	7
1.6 Licensing	7
1.7 Copyright and disclaimer notices	8
1.8 Using this documentation	8
2 Getting Started	8
2.1 Obtaining Parallelware Trainer	9
2.2 Installing and Launching Parallelware Trainer	9
2.2.1 Installation using the installer application	9
2.2.2 Manual installation	9
2.2.2.1 Linux	10
2.2.2.2 MacOS	10
2.2.2.3 Windows	10
2.3 Recommended software and environment setup	11
2.3.1 Compiler requirements	11
2.3.2 Recommended additional tools and setup	11
2.3.2.1 Linux	11
2.3.2.2 MacOS	11
2.3.2.3 Windows	12
2.4 Activate Parallelware Trainer	13

2.5 Start using Parallelware Trainer	13
2.5.2 Open an example project	14
2.5.3 Compiling and running code	15
3. Using Parallelware Trainer	15
3.1 The Parallelware Trainer interface	16
3.2 Parallelware Trainer Projects	18
3.2.1 What is a project?	18
3.2.1 Project configuration	18
3.2.1.1 Advanced project configuration	20
3.3 Building your project	21
3.4 Running your project	22
3.5 Adding parallelism to your code	23
3.5.1 Identifying opportunities for parallelism	23
3.5.2 Failed opportunities	24
3.5.3 Adding parallelism	24
3.6 Managing multiple versions of your code	27
3.6.1 Restoring a version	28
3.6.2 Creating a new version	28
3.6.3 Exporting versions	29
3.7 Fixing software defects in your code	29
3.8 Files stored in your local drive	30
3.8.1 Linux	30
3.8.2 MacOS	31
3.8.3 Windows	31
4 Advanced Optimization using Parallelware Trainer	31
4.1 Fine-tuning the parallelization	31
5 FAQ & Troubleshooting	33
5.1 Problems analyzing, building or running	33
5.1.1 Analysis fails upon opening a new file	33

5.1.2 Non-source code file reports failed analysis	33
5.1.3 Compiler or build tool is not found when I build or run	33
5.1.4 Example projects fail to build	34
5.2 Windows specific	34
5.2.1 MinGW / Cygwin tools fail to run	34
5.2.2 Visual C++ reports invalid OpenMP pragmas or clauses	34
5.3 Other	35
5.3.1 How do I change the number of threads in OpenMP?	35
5.3.2 Source file versions have disappeared after renaming or moving it	35
6 Glossary	35

1. Introduction

Parallelware Trainer is an interactive training tool designed to provide support for parallelising and optimising scientific code. Parallelware Trainer is an interactive, real-time editor with graphical user interface (GUI) features to facilitate the learning, usage, and implementation of parallel programming.

1.1 How does Parallelware Trainer work?

Parallelware Trainer uses the Parallelware technology to provide assistance in parallelising software. Unlike other parallelization tools, which use a classical dependence based analysis to identify areas of code for parallelization, the underlying Parallelware technology uses a pattern based analysis. By identifying common code patterns Parallelware is able to detect optimal strategies to improve your code performance.

1.2 How can Parallelware Trainer help me?

The Parallelware Trainer tool is an integrated development environment (IDE) that provides users with targeted advice on sections of code that can be successfully parallelized, along with the ability to test the performance improvements of particular options.

By quickly and efficiently identifying areas for possible parallelism this tool is not only useful for those learning about parallel programming, but also to aid more experienced developers as the Trainer increases the speed of understanding the software and testing possible parallelism methods.

Furthermore, Parallelware Trainer looks for defects in your code as well as potential issues related to concurrency and parallelism, reporting recommendations on how to fix them right from the integrated code editor.

1.3 List of key features

- Support for C and C++ source-code.
- Assisted code parallelization using OpenMP & OpenACC.

- Transparent, local/remote, execution and benchmarking.
- Detailed report of the parallelism discovered in the code.
- Report of code defects and recommendations related to concurrency and parallelism.
- Support for multiple compiler suites.
- *Fortran support: coming soon.*

1.4 Software requirements

Parallelware Trainer is compatible with Linux, MacOS and Windows operating systems for x64 architectures. The Linux version is also available for Power processors, specifically for the *ppc64le* architecture.

Although not strictly needed to use Parallelware Trainer, you most likely will need support software such as a C compiler and OpenMP libraries. The examples shipped with Parallelware Trainer require such software and other such as *make*. Please read the ['Recommended software'](#) section for further details.

1.4.1 Linux

The software should work as provided without requiring any additional software.

Only the C++ Runtime Library (*libstdc++*) version 6.0.20 or newer and *X11* are required. Both come pre-installed in every Linux desktop distributions so no additional steps should be required.

The following Linux distributions have been tested by the Parallelware Team, both on x64 and *ppc64le* architectures:

- Ubuntu Desktop 18.04 and 16.04
- Ubuntu Desktop 14.04 (requires installing gcc 4.9)

If you get the following errors when starting Parallelware Trainer:

```
/usr/lib/x86_64-linux-gnu/libstdc++.so.6: version `CXXABI_1.3.8' not found
/usr/lib/x86_64-linux-gnu/libstdc++.so.6: version `GLIBCXX_3.4.20' not found
```

It means that your *libstdc++* version is lower than 6.0.20 so please update it. You may do so by installing gcc 4.9 or newer.

1.4.2 MacOS

Operating system requirements: MacOS X High Sierra (version 10.13) or later.

You can check your MacOS version by opening the 'About this Mac' information from the Apple menu or from a terminal by running the command:

```
$ system_profiler SPSoftwareDataType
```

1.4.3 Windows

Operating system requirements: Windows 7, 8 or 10.

Software requirements:

- [Microsoft Visual C++ Redistributable Package](#) 2017 or newer (note that this may already be installed in your system since it is commonly used by many applications).

1.5 Contacting the authors

Parallelware Trainer is developed by Appentra Solutions S.L..

We are very interested in, and grateful for, any user comments and reports of program bugs or inaccuracies. If you have any suggestions, bug reports, or general comments about Parallelware Trainer, please send them to us using the details below:

Email: info@appentra.com

Mail: Appentra Solutions S. L.,
Centro de Investigación TIC.
Campus de Elviña S/N.
CP 15071.
A Coruña. Spain.

Telephone: +34 881015556

1.6 Licensing

Parallelware Trainer is available for a 15 day free trial period. To obtain a free trial license please email info@appentra.com.

Parallelware Trainer is available via an annual subscription. To obtain a full license or learn more about subscription options please email info@appentra.com.

1.7 Copyright and disclaimer notices

Parallelware Trainer is distributed under an *End User License Agreement (EULA)* and uses third party code software. Full details of the EULA and Third Party code licenses are distributed with the software.

1.8 Using this documentation

This documentation is designed to cover all aspects of using the Parallelware Trainer tool. This documentation is not designed to provide a tutorial on OpenMP, OpenACC or usage of accelerator hardware, instead we encourage the use of the examples that are part of the tool for full understanding and tutorials that are available on the Appentra [website](#).

We provide several useful features throughout the documentation including:

- **Getting started:** a quick guide to installing and start using the tool.
- **Usage of Parallelware Trainer:** the complete user manual.
- **Advanced features:** the documentation uses **[Advanced]** to indicate an advanced feature where more detailed knowledge of parallelization methods is necessary to ensure optimal and correctly functioning code.
- **Frequently Asked Questions:** answers to common questions regarding issues encountered when using Parallelware Trainer.
- **Glossary of terms:** at the end of this documentation there is a glossary of terms, including those used by Parallelware Trainer and by the various standards such as OpenMP and OpenACC.

2 Getting Started

This section provides a quick walk-through of the steps required to install, activate and start working with Parallelware Trainer.

2.1 Obtaining Parallelware Trainer

Parallelware Trainer is available under license only.

Please register for a 15 day free trial on the [Appentra Website](#) or email the Parallelware Team for information on obtaining a full license. Once you have completed the registration you will be sent an email. If you don't receive this email please [contact the Parallelware Team](#).

2.2 Installing and Launching Parallelware Trainer

Parallelware is distributed as a fully compiled binary file and other associated files including some example projects.

Two installation options are provided for all three supported operating systems: an installer application and a compressed file.

2.2.1 Installation using the installer application

To use the installer, launch the installer wizard which will guide you through the process of installing Parallelware Trainer.

The guided installation includes the following three steps:

1. Choose the directory where the package should be installed.
2. Choose the components to install.
3. Agree to the user license..

Once installation has successfully completed, locate the `pwtrainer` executable and launch the application as usual. By default, the installer will use the subfolder `Appentra/Parallelware Trainer` within your user directory.

To uninstall Parallelware Trainer, launch the `maintenancetool` executable found in the installation directory. Note that on MacOS this tool is available inside 'pwtrainer.app'. This can be found by right-clicking on the `pwtrainer.app` in Finder and selecting 'Show package contents', or by navigating to the `pwtrainer.app` folder in a terminal.

2.2.2 Manual installation

To install Parallelware Trainer manually, download the compressed file containing the Parallelware Trainer package.

2.2.2.1 Linux

To install and use Parallelware Trainer:

1. Move the downloaded tar.gz file to your desired installation directory.
2. Unpack the archive file and navigate to the unpacked directory:
`$ tar xzf pwtrainer-x.y.z_linux-x64.tar.gz`
3. Navigate to the unpacked directory:
`$ cd pwtrainer-x.y.z`
4. Launch the Parallelware Trainer tool:
`$./pwtrainer`

2.2.2.2 MacOS

Parallelware Trainer for MacOS is distributed as a disk image instead of a tar.gz or zip file. This can be immediately opened for use. However, to launch from a terminal:

1. Mount the disk image:
`$ hdiutil attach pwtrainer-x.y.z_macos-x64.dmg`
2. Navigate to the mounted file in /Volumes:
`$ cd /Volumes/pwtrainer`
3. Launch Parallelware Trainer:
`$ open Parallelware\ Trainer.app`

Or alternatively:

```
$ Parallelware\ Trainer.app/Contents/pwtrainer
```

2.2.2.3 Windows

To install and use Parallelware Trainer:

1. Right-click `pwtrainer-x.y.z_win-x64.zip`, select “Extract All” and follow the steps to choose the destination folder
2. To open Parallelware Trainer open the `pwtrainer-x.y.z_win-x64` folder and double-click `pwtrainer`.

2.3 Recommended software and environment setup

2.3.1 Compiler requirements

The Parallelware technology provides very advanced code analysis capabilities but it does not include compiler suites or common scientific libraries. Thus, if you wish to build and run your project from Parallelware Trainer you will need a compiler as well any required libraries.

To build code including OpenMP and OpenACC functionality from Parallelware Trainer a compiler that supports OpenMP 3.0 or higher (although some features require OpenMP 4.0 or higher) and OpenACC 2.0 or higher is required. Note that it is possible to change compilers easily within Parallelware Trainer. Therefore, you do not need one compiler that supports all functionality unless you intend to use all functionality within one piece of code (not recommended). You should check your specific compiler version support for [OpenMP](#) and [OpenACC](#).

2.3.2 Recommended additional tools and setup

To facilitate your workflow we recommend using a tool to facilitate building your code, such as *make*. Examples that are shipped with Parallelware Trainer use *make*, and provide example Makefiles that you can use. Note that using *make* or any other compilation/build tool is optional, as Parallelware Trainer enables the use of any command-line arguments for building code. In the rest of this section we outline suggested compiler, *make* and OpenMP installation methods available on the different operating systems supported by Parallelware Trainer. Please note that this is not an exhaustive list and alternative setups should still work provided they meet the requirements set-out above.

2.3.2.1 Linux

Use the package manager provided by your Linux distribution (e.g. Ubuntu's *apt* or CentOS's *yum*) to install the official packages for *make* and the latest versions of *gcc* or *clang* compilers.

2.3.2.2 MacOS

Xcode is the official solution for Apple development. Other options include community maintained software packages and building the tools:

- Xcode command line tools provide a set of tools including the Clang compiler, *make* and *git*. GNU's *gcc* and *g++* are also provided but they are mere aliases to *clang* and *clang++*. Since Clang follows the *gcc* command line options syntax, you may use them in a similar fashion but you should be aware that you are not really using *gcc*.
- [brew](#) is an unofficial package manager through which you can install *gcc* and *make* among many other software.

- [MacPorts](#) is an easy to use system for compiling, installing, and managing open source software. You can use it to download, build and install *gcc* and *make* among many others. Note that building *gcc* will take a significant amount of time.

Note that the default *gcc* compilers on Mac do *not* support OpenMP. Check available distributions using your chosen package management tool to identify a suitable *gcc* or alternative compiler that supports OpenMP.

2.3.2.3 Windows

Visual C++, GCC (either through MinGW or Cygwin) and Clang are the main C/C++ compiler options for Windows systems.

- Visual C++ is the subset of Microsoft Visual Studio devoted to C and C++ development. For Build Tools for Visual Studio 2017 and 2019, the proper workload is *Microsoft.VisualStudio.Workload.VCTools*. *nmake* is provided as an alternative to *make*. One important limitation of the Visual C++ compiler is that its support of OpenMP is restricted to version 2.0. Therefore, we do not recommend using Visual C++.
- [MinGW](#) is a Windows port of GNU tools such as *gcc* and *make*. We recommend installing the MSYS ("Minimal SYSTEM"), a Bourne Shell command line interpreter system, from which all the GNU tools can be accessed.
- [Cygwin](#), similarly to MinGW is a collection of GNU and Open Source tools which provide functionality similar to a Linux distribution on Windows. However, unlike MinGW, it also introduces a POSIX compatibility layer emulating many Unix features. It also provides a Unix-like terminal with access to all the provided tools.
- [Clang](#) is the C and C++ compiler shipped with the [LLVM Compiler Infrastructure](#). Unlike the previous options, it doesn't offer extra tooling but it is a good option in case you are only concerned with a C/C++ compiler. The regular Windows LLVM installer offers Clang with OpenMP support and compatibility with both GCC and Visual C++.

Note that the installed tools need to be accessible through your *PATH* environment variable in order for Parallelware Trainer to be able to invoke them. Some terminals or the Visual Studio developer command prompt modify this and other variables in order to set up the environment for development. These settings will be inherited by Parallelware Trainer when launched from those terminals or command prompts. If you would rather permanently include them in your *PATH* you need to add the directories where the binary executables are located by going to *System > Advanced system settings > System Variables*.

It is also likely that any C/C++ code requires header files provided by a compiler. On Windows this requires the additional step of informing Parallelware Trainer as to the location of the relevant files. This is done as part of setting up the '[Project configuration](#)'. For instance, for a MinGW 6.3.0 installation on *C:\MinGW*, the following Analysis compiler flags would be specified to provide the location of the header files:

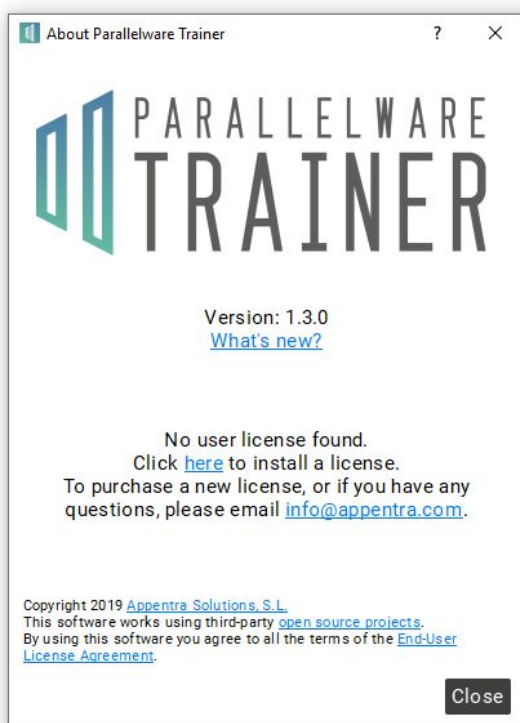
```
-IC:\MinGW\include -IC:\MinGW\lib\gcc\mingw32\6.3.0\include
```

2.4 Activate Parallelware Trainer

In order to use Parallelware Trainer a valid license must be provided. You can obtain your trial license from the trial page at <https://www.appentra.com/products/parallelware-trainer/trial>.

Once you have downloaded the license file you can install it by launching Parallelware Training and clicking on “Click here to install a license”.

You can update your license at any time through the About dialog (accessible from the Help menu in Linux and Windows or the app menu in MacOS).



2.5 Start using Parallelware Trainer

Once you have activated Parallelware Trainer with the license, you can start using the Parallelware technology to discover new parallelization opportunities in your codes!

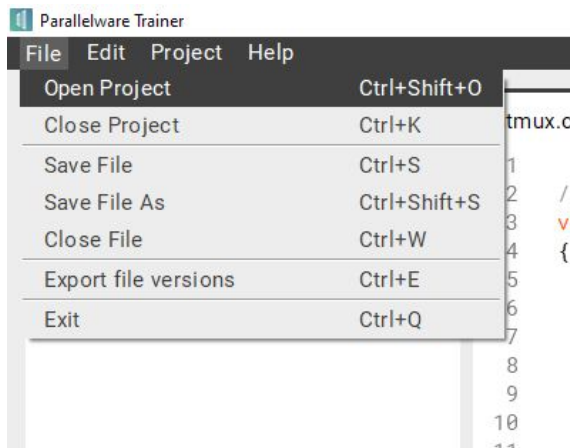
2.5.2 Open an example project

A Parallelware Trainer project refers to a folder that contains a software project. Parallelware Trainer can load any folder, including multiple nested folders, by choosing the topmost level that you wish Parallelware Trainer to access.

When opening a project Parallelware Trainer will search for the `.pwt` hidden directory. This is where Parallelware Trainer will store information about the project, such as source code file versions. When Parallelware Trainer first opens the project, if the directory is not found, a `.pwt` directory will be created.

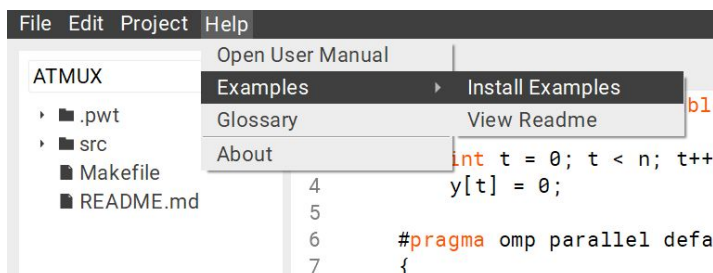
To open project:

1. Select 'Open Project' from the file menu.



2. Using the navigation pane, navigate to the directory containing the code that you wish to use.

Quickstart examples are available to help you get started with Parallelware Trainer. Select *Help > Examples > View Readme* to get an overview of the examples and the quickstart steps. Install them by selecting *Help > Examples > Install Examples* and specifying the destination folder where you want the examples copied to.



A copy of the examples will be created in the folder you specify. This means that you can freely play with their contents, making any modification to the code you see fit. You can install them to as many locations as you want.

3. Select one of the example folders and click 'Open'.

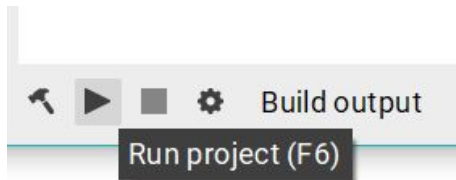
Once the project has loaded you can start using the tool to inspect, develop and parallelize the code.

2.5.3 Compiling and running code

For each project, information about how to compile/build and run the code must be provided. This can be set at any time by opening *Project > Configuration*, or selecting the Configuration button under the console outputs. However, when a build or run action is invoked by the user and this information is missing, Parallelware Trainer will prompt the user for the required information.

To build and run:

1. Click the 'Run' button located in the lower part of the console outputs.



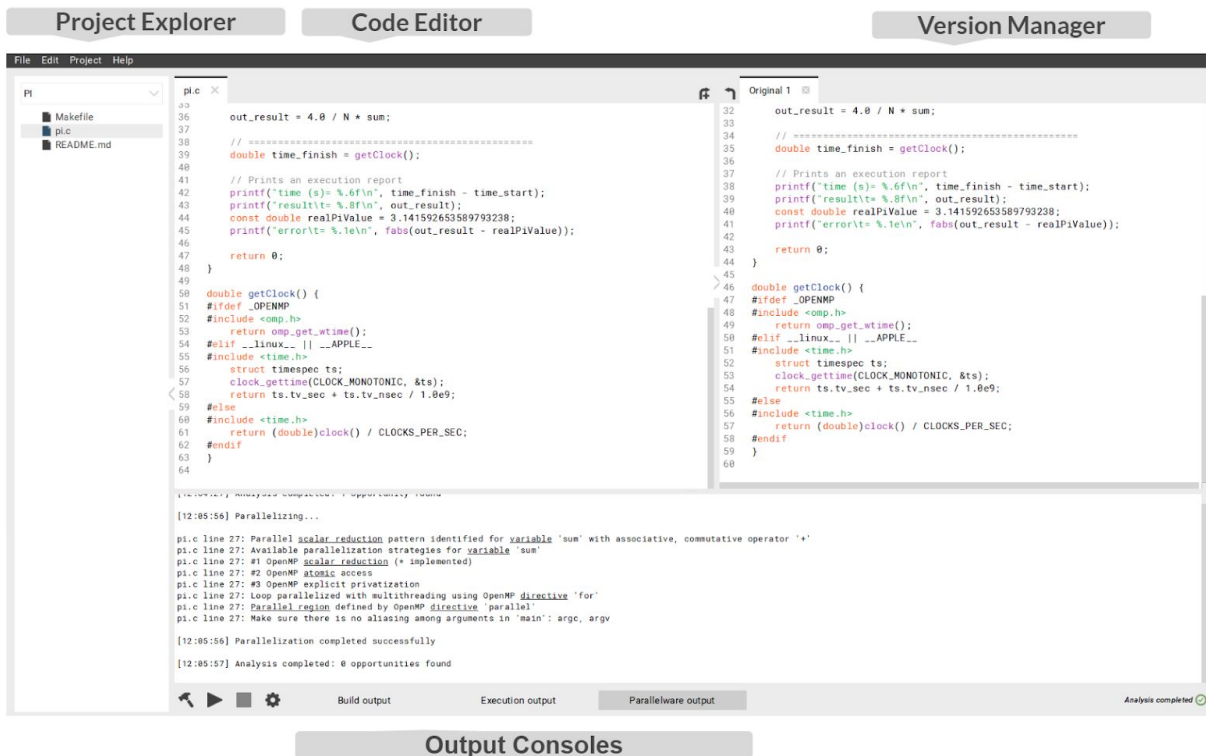
2. If either the build or run settings are missing from the project configuration, Parallelware Trainer will open the configuration dialog. Note that the build and run commands apply to the entire project, not individual source code files.
3. The source code is built and if no problems were found during compilation, the run command will be executed. Both the Build and Execution output consoles will show the output commands as well as the time elapsed during these actions.

3. Using Parallelware Trainer

Parallelware Trainer has been designed to allow you to load any project and start parallelizing immediately. You can also use Parallelware Trainer to compile, run and measure the execution time, allowing you to compare the performance of different parallelizations.

3.1 The Parallelware Trainer interface

The Parallelware Trainer tool is designed to provide all the information you need to quickly and effectively understand where and how to parallelize. The workspace is divided into four main areas:



Project Explorer

- The Project Explorer is the leftmost panel and allows you to view open projects. A project is simply a folder/directory containing your source code.
- With the Project Explorer you can rapidly switch between files and projects that you have open.
- It is where you can open and manage files contained in the selected project.

Code Editor

- The Code Editor panel is the leftmost of the two main code views. In the Code Editor panel you can edit your source code files opened from the project manager.
- You can have multiple files open at the same time and switch between them by selecting the correct tab above the editing panel.

- Close a file by clicking on the 'x' button next to the name in its tab.

Version Manager

- The Version Manager is a code viewing panel located to the right of the Code Editor. It appears similar to the Code Editor but does not allow for code changes. The Version Manager is designed to retain different versions of a given source code file, allowing you to restore and rename the versions to facilitate comparison of different parallelization approaches.
- Whenever your source code is modified automatically by Parallelware Trainer the tool saves the previous contents to a version file.
- In this panel you can select through the different versions, rename, export or delete any of them.
- To manually create a new version click the button on the top-right most part of the Code Editor.

Output consoles

- At the bottom of the tool is an output console. There are three different tabs that provide messages related to different actions.
- Build output: displays the output of executing the build instruction.
- Execution output: displays the output of running the code and will include standard output from code and execution time information.
- Parallelware output: displays the output from the Parallelware Technology. Output will be provided here whenever code is analyzed for parallelization opportunities, information on the code inserted and when errors are found that means that analysis is not possible.
- Additional actions:
 - Clear the console: right click and select 'Clear' from the pop-up menu.
 - Copy console output: right click and select 'Copy' from the pop-up menu.
 - Build: select the 'build' button below the output console or in the menu select *Project > Build*.
 - Run: select the 'run' or 'play' button below the output console or in the menu select *Project > Run*. Note that the code is always rebuild whenever the code is run.
 - Stop: interrupt a build or run command by selecting the 'stop' button below the output console or in the menu select *Project > Stop*.

- Edit the project configuration: select the 'configuration' button below the output console to update the information on building, executing, cleaning compilation output files and analysis information. Alternatively, select *Project > Configuration* in the menu.

3.2 Parallelware Trainer Projects

3.2.1 What is a project?

Opening any folder with Parallelware Trainer opens a project. To ensure correct parallelization, compilation and running of code we suggest that the folder selected is the topmost level that contains all the files needed for compiling and running the software to be considered.

Note that in Parallelware Trainer you simply open a folder from your drive and start working. You don't have to explicitly create a project which would require you to name it and give different sorts of information. All of the contents of the folder will be available through the project explorer.

It is important to note that only one build and one run command can be used for a project in Parallelware Trainer. Any valid terminal command is acceptable, including *'make'*, standard calls to compilers and even using custom scripts.

Technically, a Parallelware Trainer Project is a folder containing a *.pwt* subfolder generated by Parallelware Trainer. This subfolder is created by Parallelwar Trainer whenever a new project is loaded that does not already contain one. It is used to store project-specific parameters such as build and run commands and source file versions.

The information contained in the *.pwt* folder is portable (e.g., does not include any absolute path). Therefore it is possible to move the entire software project folder and still retain the information on versions maintained by Parallelware Trainer.

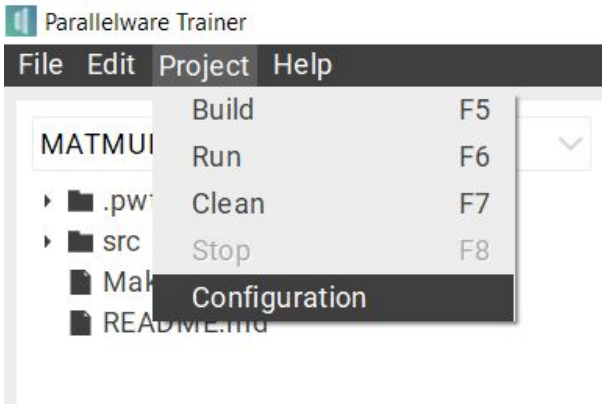
However, Parallelware Trainer relies on the relative path of files within the project for their versioning. If you rename or move files within your file structure, the version manager won't have access to its versions. To regain access to versions it is possible to manually edit the paths of filenames within the *versions* subfolder of the *.pwt* directory.

3.2.1 Project configuration

To configure the build, run, clean or analysis configuration for a project:

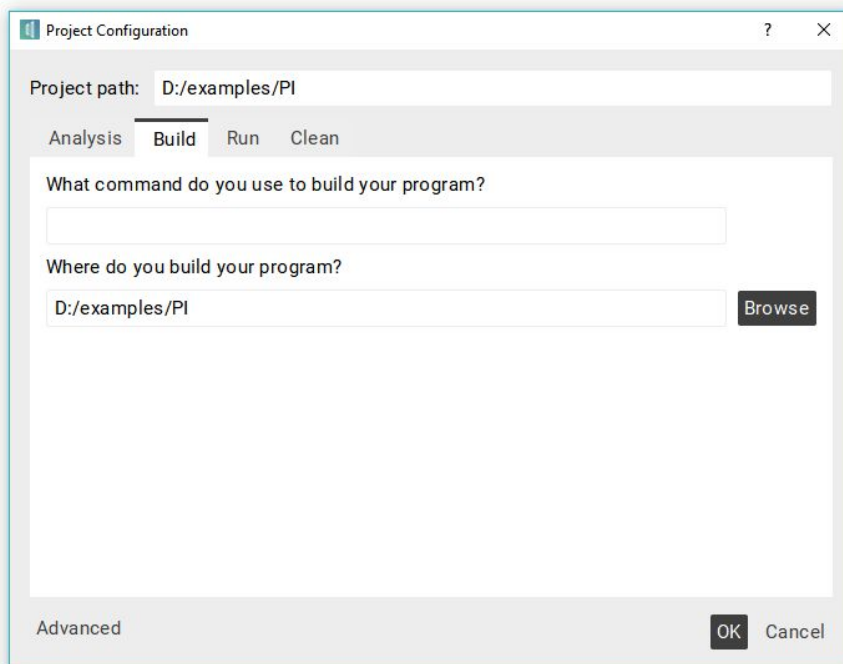
- Ensure you have the required project loaded in the Project Explorer, and open at least one of the files in the Code Editor.

- Open the project configuration dialog via the *Project > Configuration* menu or through the project configuration button located at the bottom of the output console.



Project configuration options are divided into four tabs:

1. Analysis
2. Build
3. Run
4. Clean



The Build, Run, and Clean tabs contain the commands to build, run, and clean the project, respectively. In addition, you should specify the folder where each of these commands is run. Parallelware Trainer will automatically populate this with the topmost folder of the project.

For example, if you build your code using a Makefile you are likely to provide information similar to:

- Build: enter *'make'* or *'make -f <myMakefileName>'*
Where do you build your program: the location that you would navigate to in order to clean files produced during compilation.
- Run: enter your usual run command, e.g. *./myexecutable <arg1>*
Where do you run your program: the location you would navigate to in order to run the command you specify.
- Clean: enter *'make clean'*
Where do you clean your program: the location that you would navigate to in order to clean files produced during compilation.

The first tab, Analysis, is used to provide information required to assist with Parallelware's analysis of the code. If you use compiler flags during execution Parallelware needs to know these flags to perform the analysis. Oftentimes, include directories flags must be supplied.

For example, if you use make or a script to build your code, standard output will provide the compilation command (e.g. *gcc -fopenmp -lm* etc). Please copy these flags and enter them into the compiler flags textbox.

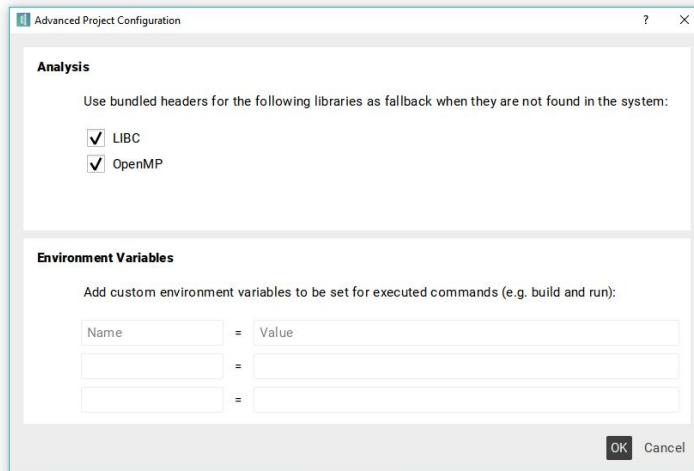
Be aware that your build system (e.g. make) may abstract some details such as include directories compiler flags that might be required for the analysis to work. For Parallelware to work it needs to know these details. Therefore, you need to specify them in the Analysis tab. Think of it this way: if it can't compile it can't be analysed!

3.2.1.1 Advanced project configuration

The 'Advanced' button located in the bottom left corner of the project configuration dialog gives access to the following advanced options:

- Bundled header files. Most C/C++ code use functions from the C standard library, also known as *libc*. In some cases, such as in Windows systems, *libc* header files may not be available which most likely will prevent Parallelware Trainer from successfully analyzing the code. To aid in those cases, Parallelware Trainer bundles [musl libc](#) header files as well the OpenMP 5.0 header file, and will resort to them when they are not found in the system. To change this behavior and prevent Parallelware Trainer from doing so, uncheck the corresponding checkboxes.

- Environment variables. Up to three environment variables can be specified to be set up for build and run commands. This can be used, for instance, to set the `OMP_NUM_THREADS` variable to the desired value for OpenMP applications.



3.3 Building your project

You can build your project by either selecting the *Project > Build* menu option or by clicking on the build button located under the output consoles.

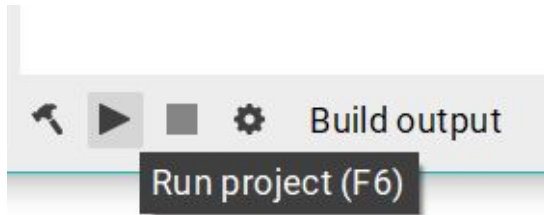
Whenever a project is run, it will automatically build the files to ensure the most recent version is used.

If the build configuration has not previously been specified for a project, selecting build will prompt you for the build configuration information. See ['Project configuration'](#) for information on completing the build configuration.

The *build* command may be any valid compilation command used in a terminal, but must only be one command. Typical commands include direct use of a compiler (e.g. *gcc*), using a build system (e.g. *make*) or a custom script (e.g., *./my-build.sh*).

3.4 Running your project

You can run your project by either selecting the *Project > Run* menu option or by clicking on the run button located under the output consoles. Everytime the code is run Parallelware Trainer will ensure the correct version is being used by building the code first.



If the run/execution configuration has not previously been specified for a project, selecting run command will prompt you for the run (and build) configuration information. See ['Project configuration'](#) for information on completing the run configuration.

The *run* command may be any valid compilation command used in a terminal, but must only be one command. Typical commands include the direct invocation of the built binary (e.g., `./my-binary`) or a script.

All necessary environment variables including the number of OpenMP threads, should be set during the run command. You can specify custom environment variables to be set for build and run commands can be set through the Advanced project configuration dialog (see ['Advanced project configuration'](#) section). Alternatively, you may wish to use a script that sets multiple variables. If you just wish to set just one variable, such as the number of threads, this can be entered in the run command box, e.g. `env OMP_NUM_THREADS=8 ./my-binary`.

N.B. if you are using Parallelware Trainer for remote execution via a queuing system on an external machine, the run command should invoke the usual submission command for the appropriate queue submission system (e.g `qsub ...` or `sbatch`).

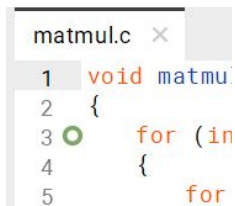
3.5 Adding parallelism to your code

3.5.1 Identifying opportunities for parallelism

To add parallelization to your code you first need to identify areas for parallelism. Parallelware Trainer does this for you by analyzing your source code files to search for parallelization opportunities.

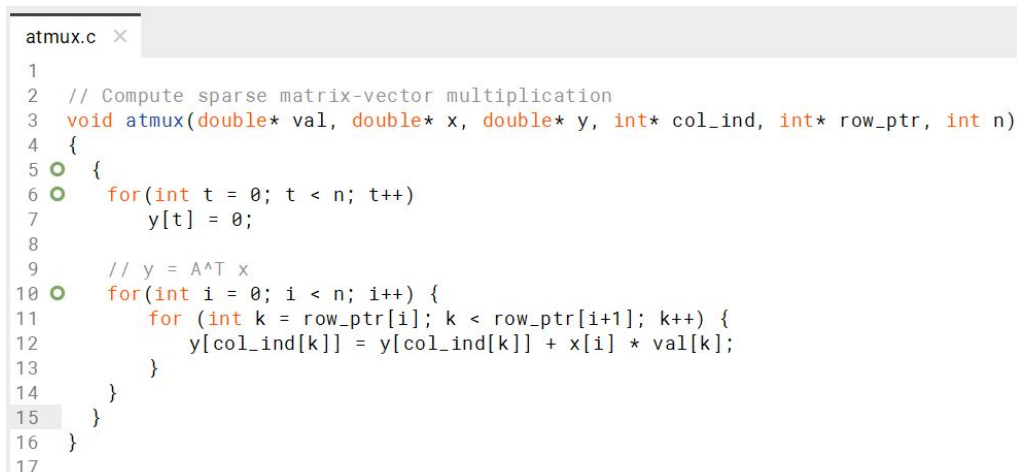
The analysis is performed every time you open a file or you save your changes.

Opportunities are identified through green markers next to the line number:



```
matmul.c x
1 void matmul
2 {
3   for (in
4   {
5     for
```

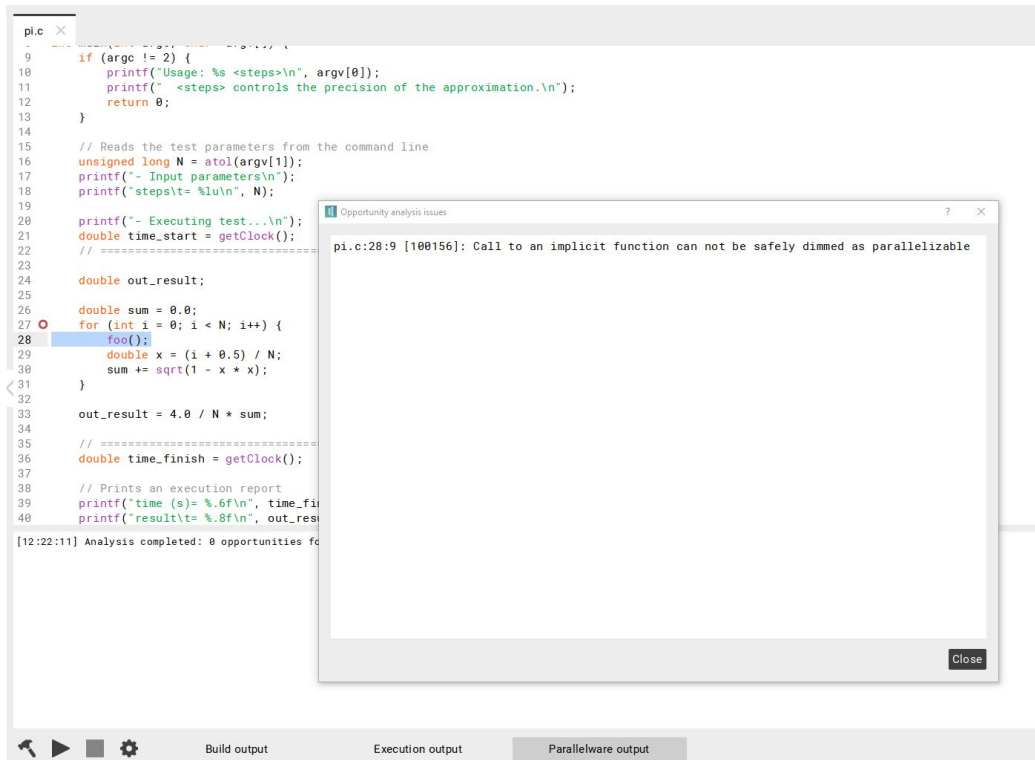
One common optimization when tuning parallelized code is grouping several loops into the same parallel region. To have Parallelware Trainer analyze such a region for opportunities, explicitly enclose it with curly braces. If the region is suitable for parallelization, a clickable green marker will be shown next to the line number of the opening curly brace.



```
atmux.c x
1
2 // Compute sparse matrix-vector multiplication
3 void atmux(double* val, double* x, double* y, int* col_ind, int* row_ptr, int n)
4 {
5   {
6     for(int t = 0; t < n; t++)
7       y[t] = 0;
8
9     // y = A^T x
10    for(int i = 0; i < n; i++) {
11      for (int k = row_ptr[i]; k < row_ptr[i+1]; k++) {
12        y[col_ind[k]] = y[col_ind[k]] + x[i] * val[k];
13      }
14    }
15  }
16 }
17
```

3.5.2 Failed opportunities

Sometimes, the analysis of the source code file succeeds but some specific loop could not be analyzed. For instance, this happens for loops where an external function is called: since the source code of that function is not available, Parallelware can not determine whether the loop is an opportunity for parallelization or not. In such cases, a red marker – instead of a green one – is shown next to the corresponding line number.



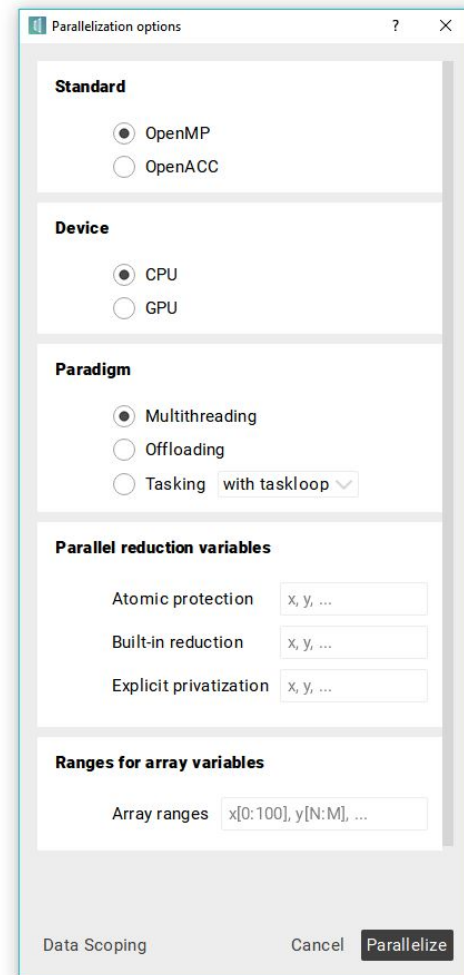
You can click to the marker to open a dialog showing some information regarding the problem encountered during analysis. This information may give you some hints on how to change the code to allow the analysis to succeed.

3.5.3 Adding parallelism


Parallelware Trainer will assist you in parallelizing any of the identified opportunities. It will be able to automatically parallelize most of them. In some cases, you will need to provide additional information that can not be automatically determined by analyzing your code. In a few cases it will determine that your code can not be parallelized without refactoring/restructuring the code.

To introduce parallelism:

1. Click the green marker next to the line of the opportunity you want to parallelize. This opens the parallelization options dialog.
2. Select your desired parallelization standard:
 - OpenMP (default): introduce OpenMP parallelism.
 - OpenACC: introduce OpenACC (for GPGPU accelerators) parallelism.
3. Select the hardware you wish to target with the parallelism:
 - CPU (default): targets CPU hardware (only valid with OpenMP).
 - GPU: targets GPGPU hardware.
4. Select your desired parallelization paradigm:
 - Multithreading (default): will apply loop multithreading parallelization strategies to the region beginning at the green line marker.
 - Offloading: will offload computation from the region beginning at the green line marker to accelerator devices.
 - Tasking: will apply the tasking paradigm to the region beginning at the green line marker. Two options are available:
 - i. “with taskloop”: the whole loop is parallelized using the OpenMP 4.5 *taskloop* worksharing construct.
 - ii. “per iteration”: a task is created per loop iteration and a *taskwait* directive is added after the loop to ensure a synchronization barrier. This requires at least OpenMP 3.0.
5. (Optional) Click the ‘Data Scoping’ button on the bottom left to launch a data scoping analysis of all the variables used within the loop. For each variable, this analysis provides its kind, read/write usage, and whether it is a temporary. It also shows if the variable is subject to a parallel pattern such as a scalar reduction. Furthermore, the



scoping clauses for multithreading with OpenMP and offloading with OpenACC are provided if applicable.


Data Scoping (preview)

Loop at (6,2)

Name	Kind	Usage	Temp	Pattern	OpenMP (multi)	OpenACC (offload)
n	scalar	r	no		shared(n)	copyin(n)
t	scalar	rw	no			
y	pointer	w	no	forall	shared(y)	copyout(y)

Loop at (10,2)

Name	Kind	Usage	Temp	Pattern	OpenMP (multi)	OpenACC (offload)
x	pointer	r	no		shared(x)	copyin(x)
n	scalar	r	no		shared(n)	copyin(n)
row_ptr	pointer	r	no		shared(row_ptr)	copyin(row_ptr)
col_ind	pointer	r	no		shared(col_ind)	copyin(col_ind)
val	pointer	r	no		shared(val)	copyin(val)
i	scalar	rw	no			
k	scalar	rw	yes			
y	pointer	rw	no	sparse reduction	shared/atomic(y)	copy/atomic(y)

Loop at (x,y): variable usages for loop located at line x and column y.
Name: name of the variable.
Kind: variable datatype kind (scalar, pointer, array, dynarray, derived, other).
Usage: the read/write status of the variable: read-only (ro), write-only (wo), read-write (rw).
Temp: variable is temporary, that is, internal to the loop (i.e. declared within the loop in C/C++).
Pattern: the type of parallel pattern a variable is subject to (forall, scalar reduction, sparse reduction).
Note that most variables will not have an associated parallel pattern.
OpenMP (multi): the variable's required datascope clause for use in OpenMP multi-threading.
Valid values: shared, private, reduction.
Special value shared/atomic is specified if the atomic directive is also required.
OpenACC (offload): the variable's required datascope clause for use in OpenACC offloading.
Valid values: copy, copyin, copyout.
Special values copy/atomic or copyout/atomic are specified if the atomic directive is also required.

This is a preview feature, likely to change in the future.

Close

6. (Optional) Parallel reduction variables

- [Advanced] Atomic protection: add the name of the variable to be accessed atomically. Note that this may slow down your parallelism but is sometimes needed to ensure correctness.
- [Advanced] Built-in reduction: target a reduction using the built-in functionality from OpenACC or OpenMP.
- [Advanced] Explicit privatization: make private copies of variables/arrays.

Note that these advanced options can be introduced at any time, but you may first want to use the default parallelization opportunity to identify if any of these options are

valid. The Parallelware output console will provide information on the possible solutions after closing the parallelization dialog.

7. (Optional) [Advanced] Ranges for array variables: comma separated list of array bounds using the syntax `arrayName[startIndex:count]`. For instance, given an array `y` and a variable `N`, you would specify: `y[0:N]`. Note that the syntax specifies the start index and length. Thus, `y[10:100]` corresponds to the range from the 10th element up to the 110th element.
8. Click 'Parallelize'.

Parallelware Trainer will attempt to parallelize the opportunity given your settings. If parallelization is possible, the required pragmas and any necessary code modifications will be inserted. In some cases, one or more actions might be required to ensure that the parallelization is correct (e.g., validate that a condition assumed by Parallelware is true). This information is shown in the Parallelware output console.





Notice that the parallelization options dialog will validate your set of selections to ensure that you don't select any invalid combinations.

The Parallelware output console shows information regarding the parallelization performed by Parallelware Trainer. This includes information on what parallelization strategies have been implemented (e.g. "*Ranking of available parallelization strategies for variable ...*"). The implemented strategy is highlighted with "*(*) selected*". The type of parallelization is indicated e.g.:

- "*Parallel sparse reduction on variable 'y'*": variable `y` is subject to a 'reduction' that is an opportunity for parallelization.
- "*Fully parallel computations on variable 'C'*": all computations involving variable `C` can be parallelized within this code region.

```
[09:35:32] Parallelizing...
D:/pwtrainer-1.0.0-RC1/docs/samples/ATMUX/src/atmux.c:1:1: note: Analyzed function 'atmux'
D:/pwtrainer-1.0.0-RC1/docs/samples/ATMUX/src/atmux.c:6:2: note: Parallel loop
D:/pwtrainer-1.0.0-RC1/docs/samples/ATMUX/src/atmux.c:6:2: note: Ranking of available parallelization strategies for variable 'y'
D:/pwtrainer-1.0.0-RC1/docs/samples/ATMUX/src/atmux.c:6:2: note: #1 Use of pragma <atomic> (*) selected
D:/pwtrainer-1.0.0-RC1/docs/samples/ATMUX/src/atmux.c:6:2: note: #2 Use of explicit privatization
D:/pwtrainer-1.0.0-RC1/docs/samples/ATMUX/src/atmux.c:6:2: note: Parallel sparse reduction on variable 'y'
D:/pwtrainer-1.0.0-RC1/docs/samples/ATMUX/src/atmux.c:6:2: note: Dependencies due to temporary variables do not prevent
parallelization: 'k'
Parallelware: note: Summary of parallelization (Total / Opportunities / Parallelized)
Parallelware: note:   Loops:   3 / 2 / 1
Parallelware: note:   Statements: 11 / 10 / 7

[09:35:32] Parallelization completed successfully
```





 Build output Execution output **Parallelware output**

3.6 Managing multiple versions of your code

Everytime Parallelware Trainer automatically modifies source code, it stores a copy of its previous state so that you never lose your original code. You can also choose to create a version of your code at any time.

Having quick access to multiple versions of the same code enables quick and efficient comparison and testing of different parallelization strategies.

The version manager occupies the right-most area of the Parallelware Trainer interface. If you don't see it, make sure that it is expanded by clicking on the arrow button (<) to the right of the Code Editor. The available versions are shown as different tabs. You can rename, export or delete them by right-clicking on the name.



Note that you can only ever have *one* version of the code live in the Code Editor. This ensures that during analysis, compilation and execution there is no confusion over which version is being used. The version in the Code Editor is also the version that is currently saved as the 'master' on disk. That is, when you access your software with any other tool, only the live version in the Code Editor will be available.

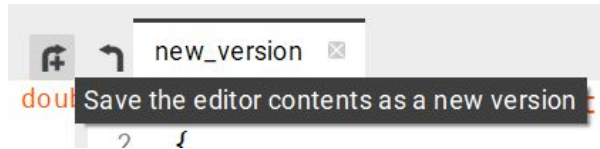
3.6.1 Restoring a version

The left-facing arrow located to the left of the first version allows you to restore the selected version, thus overwriting the source code opened in the editor.



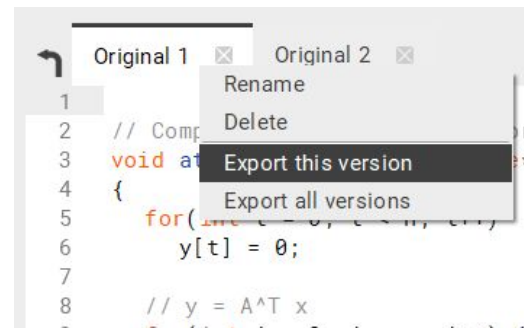
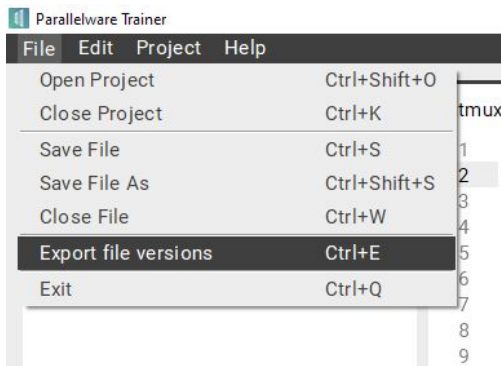
3.6.2 Creating a new version

You can easily create a new version of the code in the editor at any time by clicking the 'new version' button, located on the top-right corner of the Code Editor. It will ask you to provide a name for the new version.



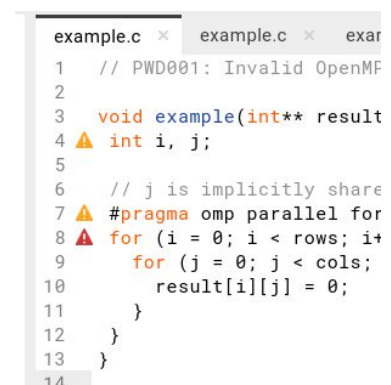
3.6.3 Exporting versions

Often, and especially when experimenting, you will create several versions of your code, each using a different parallelization strategy and options. You will switch back and forth among those versions to compare which one yields a better performance. In case you want to export those versions out of Parallelware Trainer, you can do so by accessing the *File > Export file versions* menu entry. Alternatively you can export a specific version by right-clicking its name in the version manager and selecting *Export this version*.

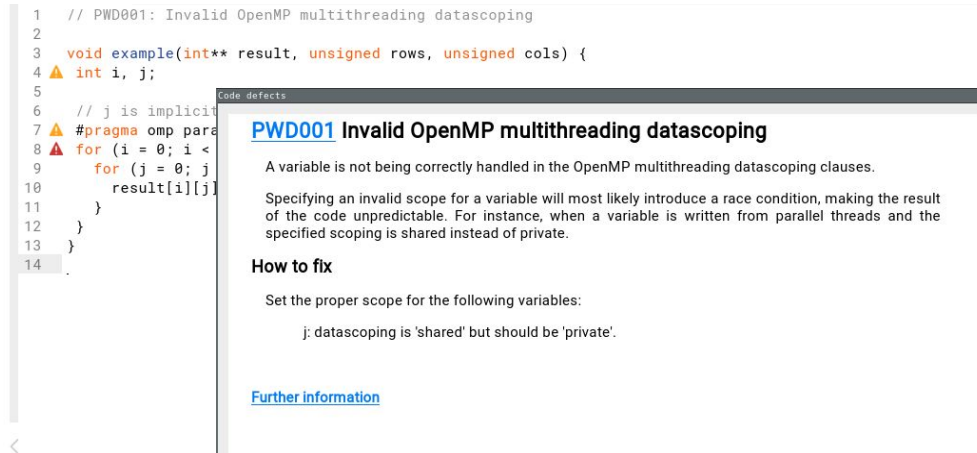


3.7 Fixing software defects in your code

Parallelware Trainer is able to detect software defects regarding concurrency and parallelism. If your code contains any of these defects it may build and run without any problem but hidden bugs and wrong calculations can arise. When a defect is found, a red warning icon is displayed next to the corresponding line number. Yellow warning icons are also displayed to show recommendations about issues in the code that might lead to defects or that do not follow best practices on concurrency and parallelism.



By clicking a warning icon, a window displaying extended information about the defect or recommendation will be shown. The information includes a description and guidance on how to change the code to fix it.



Parallelware Trainer offers code examples showcasing supported defects and recommendations. Install them by selecting *Help > Examples > Install Examples*.

You can find detailed information about the supported defects and recommendations in the knowledge base area of the Appentra website: <https://www.appentra.com/knowledge/checks>.

3.8 Files stored in your local drive

As explained in [‘What is a project?’](#), Parallelware Trainer will create and manage a .pwt subfolder located right within the top level directory that you open as a project through *File > Open Project*. The information contained in this directory is specific to the project and its contents (e.g., source code file versions).

However, just like any other application, Parallelware Trainer also stores global settings, which are shared by all projects, as well as some temporary files. These files are stored inside the user's personal folder but the specific location varies with the operating system.

If you have installed Parallelware Trainer from a compressed archive and you want to totally remove it from your system you should remove these folders yourself. The same applies if you want to reinstall it without keeping any settings from the previous installation.

3.8.1 Linux

The default locations for Parallelware Trainer global settings and temporary files on Linux are:

`~/config/Appentra/pwtrainer`

`~/local/share/Appentra/pwtrainer`

3.8.2 MacOS

The default locations for Parallelware Trainer global settings and temporary files on MacOS are:

`~/Library/Preferences/Appentra/pwtrainer`

`~/Library/Application Support/Appentra/pwtrainer`

3.8.3 Windows

The default location for Parallelware Trainer global settings and temporary files on Windows is:

`%LocalAppData%\Appentra\pwtrainer`

4 Advanced Optimization using Parallelware Trainer

4.1 Fine-tuning the parallelization

With Parallelware Trainer, the parallel performance and eventual speed-up of the parallel version of your code can be fine-tuned by using directives.

We suggest that you create a copy of your code for each change you make so that you can easily compare the resulting performance.

Opportunities for fine-tuning include:

- Testing different clauses in the inserted directives (e.g. the *schedule* clause in OpenMP)
- Testing the different parallelization strategies. When multiple strategies are possible, the Parallelware output console will list all available strategies. To test the different strategies:
 - a. save the existing parallel version into the Version manager;

- b. restore the original code and reselect the green button;
- c. using the information from the Parallelware console on the available strategies select the appropriate advanced parallelization options.

To quickly compare performance of different strategies:

1. Make sure all versions are saved in the Version Manager with appropriate naming conventions, including the version currently visible in the Code Editor window.
2. For each version:
 - a. Restore the version into the code manager
 - b. Build and run the code
 - c. Record the execution time.
3. To change the number of threads, ensure that environment variable is properly set (see ['Advanced project configuration'](#)) or that the run script or runtime command are updated accordingly.

You are then able to choose the most effective implementation based on the execution time. Note that in some cases peak performance on a particular number of threads may be achieved with a different parallel implementation. It is therefore important to understand the overall performance for all versions with different numbers of threads.

5 FAQ & Troubleshooting

5.1 Problems analyzing, building or running

5.1.1 Analysis fails upon opening a new file

In order for Parallelware to be able to analyze your code, a compiler must be able to correctly interpret the source-code. In many cases a source code file depends on other files such as C header files installed in the system. In other cases certain symbols must be defined through preprocessor directives. In such scenarios you need to supply that information to Parallelware in the same way you would do so for a compiler: through compiler flags.

Open *Project > Configuration* and make sure you specify any required compiler flags in the *Analysis* tab. Read the ['Project configuration'](#) section for further details.

5.1.2 Non-source code file reports failed analysis

Parallelware output console shows “Parallelware: error: unsupported language (<none>) for file '/home/jnovo/dev/pwt-samples-acc/ATMUX/Makefile'” when opening or saving files such as a Makefile. This is merely a notification stating that the file is not analyzable by Parallelware since it is not a source code file of a supported language and thus, can be ignored.

5.1.3 Compiler or build tool is not found when I build or run

Most likely there is a problem with the binary not being accessible from your *PATH* environment variable.

Commands specified in your project *Build* and *Run* settings inherit the same environment variables as the Parallelware Trainer process. In Linux and MacOS it is common to have different environment variables set for shells such as *bash* than for your window manager environment. When you launch a terminal from your window manager, a new shell process is created for you, thus the configuration of that shell is applied. As a result, the environment variables are likely to differ from those available to graphical applications launched directly through the window manager.

You can check environment variables by entering *env* as your build command and building your project. This will print all available environment variables to the build output console of Parallelware Trainer. Try again by using a shell to run the same command, for instance enter

`bash -c env`, build again and check the console again. To check environment variables on Windows you may run `cmd /c set`.

You can ensure that you use your configured shell by invoking your build and run commands through (for instance by prepending `bash -c` to them).

Alternatively, you can launch Parallelware Trainer from your favorite terminal by invoking the `pwtrainer` executable. This way, it will inherit the terminal environment variables.

The same applies to Windows and the command prompt. If you use Visual C++ you most likely will need to launch Parallelware Trainer from the Visual Studio Developer Prompt in order to have the environment set properly.

5.1.4 Example projects fail to build

Ensure that the required software is installed. The shipped examples require a C compiler such as `gcc` that supporting either OpenMP or OpenACC as well as the `make` utility.

Read the ['Recommended software'](#) section for further details.

5.2 Windows specific

5.2.1 MinGW / Cygwin tools fail to run

If you get an error similar to “error while loading shared libraries” or “code execution cannot proceed because XXX was not found” then the Parallelware Trainer execution environment is not properly configured for MinGW or Cygwin. Open the MinGW/Cygwin terminal and launch Parallelware Trainer using the command line as explained in [‘Installing and Launching Parallelware Trainer’](#). This will allow Parallelware Trainer to inherit the appropriate environment setup.

5.2.2 Visual C++ reports invalid OpenMP pragmas or clauses

The Visual C++ `cl` compiler only supports OpenMP 2.0 and most likely your pragmas or clauses require a higher version. Switch to a compiler that uses Open 3.0 or higher, such as `gcc`, `clang` or `pgi`.

5.3 Other

5.3.1 How do I change the number of threads in OpenMP?

You can set the number of desired OpenMP threads by setting the `OMP_NUM_THREADS` environment variable to the proper value. Not doing so results in OpenMP using just one thread in most systems. Refer to your compiler and OpenMP runtime library documentation for details. Read ['Advanced project configuration'](#) to see how to set the variable for commands executed from Parallelware Trainer.

5.3.2 Source file versions have disappeared after renaming or moving it

As explained in ['What is a project?'](#), Parallelware Trainer relies on the relative path of files within the project for their versioning. If you rename or move files within your file structure, the version manager won't have access to its versions. To regain access to versions it is possible to manually edit the paths of filenames within the `versions` subfolder of the `.pwt` directory.

6 Glossary

Accelerator:

A device other than the CPU that can be used to provide additional FLOPs (floating point operations) and therefore improve software performance. To use an accelerator requires specific directives, such as using OpenMP and OpenACC for GPGPUs.

Atomic:

Use OpenMP atomic operations to allow multiple threads to safely update a shared numeric variable. An atomic operation applies only to the single assignment statement that immediately follows it, so atomic operations are useful for code that requires fine-grain synchronization.

Build Console:

The left console in the Parallelware Trainer tool that provides information on the build of code.

Console:

The area of the Parallelware Trainer tool that provides information to the user on current activities. Parallelware Trainer has three consoles: the Build, Execution and Parallelware consoles.

CPU:

The 'Central Processing Unit': where your code is executed unless you use an accelerator.

Directive:

A directive or pragma (from "pragmatic") is a computer programming language construct that specifies how a compiler (or other translator) should process its input. Directives are not part of the grammar of a programming language, and may vary from compiler to compiler. They can be processed by a preprocessor to specify compiler behavior, or function as a form of in-band parameterization.

Execution Console:

The middle console in the Parallelware Trainer tool that provides information on the execution of code.

GPU:

A GPU (Graphics Processing Unit) is an accelerator based on the graphics processing technology that originated in games consoles. GPU-accelerated computing offloads compute-intensive portions of the application to the GPU, while the remainder of the code still runs on the CPU.

GUI:

A Graphical User Interface (GUI) is an interface that uses manipulation of icons, menus and windows by a mouse and keyboard to facilitate human-computer interaction. Parallelware Trainer is a training GUI for assisted parallelism.

IDE:

An Integrated Development Environment (IDE) is a software application that provides comprehensive support to computer programmers for software development. Parallelware Trainer is an IDE that supports parallelization during the software development cycle through the use of the Parallelware technology.

Offload:

'Offloading' the computation work to another device, such as a GPGPU, means to execute the program on a device other than the CPU.

Parallelware:

The technology that provides guided and automated parallelization. This is used by the Parallelware Trainer tool to provide assistance in identifying areas for parallelism in your code.

Parallelware Console:

The console in the Parallelware Trainer tool that provides the detailed parallelization report.

Parallelware Trainer:

An interactive integrated development environment (IDE) that provides assistance in identifying areas for parallelism in your code.

Pragma:

See definition of Directive.

Privatization:

Remove dependencies that occur across different threads in a parallel program by declaring a variable as 'private'. This results in the processor making private copies of a variable shared by multiple threads, hence making each thread capable to operate on its local copy of this variable rather than a shared one.

Project:

A folder containing user files plus a .pwt subfolder created and used by Parallelware Trainer to store its project-specific information. Each project its own build and run commands that must be configured.

Project explorer:

Leftmost area of the Parallelware Trainer user interface, devoted to selecting the active project and exploring its contents.

PW (and pw):

The Parallelware acronym.

Reduction:

Reduction operations are some of the most commonly used parallel patterns. A reduction combines all the elements in a collection (often an array) into one using an operator. To implement a reduction operation using Parallelware Trainer select the parallelization interface and select 'Reduction' then add the variable name that you wish to perform a reduction on.

SIMD:

Single Instruction Multiple Data, support coming soon.

Task:

A paradigm, not yet supported by Parallelware Trainer, where different blocks of code are executed in parallel.

Version manager:

Rightmost area of the Parallelware Trainer user interface, devoted to managing the different versions of a given source code file.