Making OpenMP Easier with Parallelware Tools

Trainer & Analyzer

Manuel Arenaz
manuel.arenaz@appentra.com
Develop, test and benchmark several parallelization strategies in the current increasingly heterogeneous environment.

Automatic data scoping across procedure boundaries in the presence of complex in-memory data layouts
Tell me, I will forget,  
Show me, I may remember,  
Involve me, I will understand.”
Parallelware Trainer (v1.0 Sep 2018)

Technical features:

- Identification of parallelization opportunities.
- Assistance with the introduction of correct OpenMP and OpenACC directives.
- Correct data scoping, including private/shared variables.
- Support for C programming language.
- Use any compiler and any build/compilation tool in Windows, Linux and MacOS.
- Develop, test and benchmark all within the same interface.

Benefits:

- Faster, more effective learning.
- Work on realistic codes rather than toy examples, including your own code.
- Reduced learning curve.
- Parallelize code within minutes.
- Immediate identification of where and how to parallelize.
- Support for multithreading, offloading to GPUs.

https://www.appentra.com/products/parallelware-trainer/
A command-line reporting tool to improve productivity of HPC application developers
Parallelware Analyzer (Beta)

Command line tool leveraging Parallelware technology.

- Help to understand where and how to parallelize in real codes.
- Reports to facilitate understanding the code from different perspectives.
- Batch processing of files/directories of big code bases.
Parallelware Analyzer (Beta)

**DESCRIPTION**

`pwanalyzer` provides different analyses to support code parallelization, reporting detailed information about the code, its data scoping, access patterns, parallelization methods, programming languages and data types.

**SYNOPSIS**

```sh
pwanalyzer <analysis> [options] <source files/directories> [-- <compiler flags>]
```

**ANALYSIS**

- `--overview` analyze programming languages (e.g. C, C++, Fortran) and parallelization methods (e.g. OpenMP, MPI)
- `--code` dumps the source code annotated with parallelization opportunities
- `--functions` summarizes properties of the functions found in the code (e.g. independent, number of parallel loops)
- `--datalayout` provides information about all the built-in and provides information about all the built-in and
- `--datascoping` performs data scoping analysis

**OPTIONS**

- `--noext` skip external functions
- `--brief` do not print the verbose explanations about the meaning of all the fields in the tables
- `--exclude <files/dirs>` skip the specified files and folders
- `--simd | -multi` (functions, code) identify either innermost loops SIMD or outermost loops multithreading parallelization opportunities
- `--name <name/regex>` (functions, datalayout) filter by name (accepts a regular expression)
- `--openmp multi|offload` (datascoping) show OpenMP data scoping using the specified paradigm
- `--openacc offload` (datascoping) show OpenACC data scoping using the specified paradigm
- `--function <name/regex>` (datascoping) filter by function name (accepts a regular expression)
- `--loop <name/regex>` (data scoping) filter by loop name with syntax `<file>::<function>::<line>::<column>` (accepts a regular expression)
- `-- <compiler flags>` gcc/clang-compliant flags required to build source code (must be last if present)

**EXAMPLES**

```sh
$ pwanalyzer --overview src/
$ pwanalyzer --functions --multi main.c
$ pwanalyzer --code --multi main.c -- -I/usr/include/mpi -DCUSTOM_PARAM_ON
$ pwanalyzer --datascoping --openacc offload --function 'compute.*' LULESHmk/
```
Parallelware Analyzer (Beta)

Automatic data scoping across procedure boundaries in codes using complex in-memory data layouts

```
./pwanalyzer --datascoping --openmp multi --openacc offload --function compute_energy_for_node ~/SC18/pwtrainer/samples/LULESHmk
6 total files found
0 user-excluded files
4 unknown type files
2 source code files
2 supported
1 analyzable
1 non-analyzable
0 unsupported

[1/1] /home/jnovo/SC18/pwtrainer/samples/LULESHmk/src/lulesh_mk.c SUCCESS

<table>
<thead>
<tr>
<th>Loop</th>
<th>Variable Kind</th>
<th>Read/Write</th>
<th>Temporary Pattern</th>
<th>OpenMP</th>
<th>OpenACC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>lulesh_mk.c:compute_energy_for_node:32:5 r</td>
<td>scalar</td>
<td>rw</td>
<td>x</td>
<td>sparse reduction shared/atomic(TMP) copyout/atomic(TMP)</td>
</tr>
<tr>
<td></td>
<td>lulesh_mk.c:compute_energy_for_node:32:5 t</td>
<td>scalar</td>
<td>rw</td>
<td>x</td>
<td>sparse reduction shared/atomic(TMP) copyout/atomic(TMP)</td>
</tr>
<tr>
<td></td>
<td>lulesh_mk.c:compute_energy_for_node:32:5 TMP</td>
<td>array</td>
<td>wo</td>
<td>x</td>
<td>sparse reduction shared/atomic(TMP) copyout/atomic(TMP)</td>
</tr>
<tr>
<td></td>
<td>lulesh_mk.c:compute_energy_for_node:32:5 pl</td>
<td>scalar</td>
<td>rw</td>
<td>x</td>
<td>sparse reduction shared/atomic(TMP) copyout/atomic(TMP)</td>
</tr>
<tr>
<td></td>
<td>lulesh_mk.c:compute_energy_for_node:32:5 tnp</td>
<td>scalar</td>
<td>rw</td>
<td>x</td>
<td>sparse reduction shared/atomic(TMP) copyout/atomic(TMP)</td>
</tr>
<tr>
<td></td>
<td>lulesh_mk.c:compute_energy_for_node:34:8 pl</td>
<td>scalar</td>
<td>ro</td>
<td>x</td>
<td>sparse reduction shared/atomic(TMP) copyout/atomic(TMP)</td>
</tr>
<tr>
<td></td>
<td>lulesh_mk.c:compute_energy_for_node:34:8 t</td>
<td>scalar</td>
<td>rw</td>
<td>x</td>
<td>sparse reduction shared/atomic(TMP) copyout/atomic(TMP)</td>
</tr>
<tr>
<td></td>
<td>lulesh_mk.c:compute_energy_for_node:34:8 TMP</td>
<td>array</td>
<td>wo</td>
<td>x</td>
<td>sparse reduction shared/atomic(TMP) copyout/atomic(TMP)</td>
</tr>
<tr>
<td></td>
<td>lulesh_mk.c:compute_energy_for_node:34:8 tnp</td>
<td>scalar</td>
<td>rw</td>
<td>x</td>
<td>sparse reduction shared/atomic(TMP) copyout/atomic(TMP)</td>
</tr>
</tbody>
</table>

Loop : loop name following the syntax <file>::<function>::<line>::<column>
Variable : name of the variable
Kind : variable datatype kind (scalar, pointer, array, dynarray, derived, other)
Read/Write : whether the variable is read-only ("ro"), write-only ("wo"), read-write ("rw") within the loop
Temporary : specifies whether the variable is internal to the loop (i.e. in C/C++ it is declared within the loop body)
Pattern : parallel pattern (read-only, forall, scalar reduction, sparse reduction)
OpenMP : OpenMP scoping (valid values: shared, private, reduction; special value: shared/atomic when atomic directive is also required)
OpenACC : OpenACC scoping (valid values: copy, copyin, copyout; special values: copy/atomic, copyout/atomic when atomic directive is also required)

1 files successfully analyzed and 0 failures in 124 ms
```
Making OpenMP Easier with Parallelware Tools

Trainer & Analyzer

Manuel Arenaz
manuel.arenaz@appentra.com