

Parallelware Analyzer

Quality Assurance & Development Tools for Parallel Code

Parallelware Analyzer is a suite of command-line tools aimed at helping software developers to build better quality parallel software in less time. Designed around the needs of developers, Parallelware Analyzer provides the appropriate tools for the key stages of the parallel development workflow, aiding developers with code analysis that would otherwise be error-prone and completed manually. It can also be easily integrated with DevOps tools to exploit information from build systems such as CMake and benefit from its automatic usage during Continuous Integration.

1. Get a quick high-level overview of the code.

Getting started can be daunting, no matter whether you wrote the code or not.

- ▶ Use an entry-point tool that provides a high-level summary of your code and suggests next steps.

2. Prepare your code for parallelization.

How an algorithm is coded has a great influence on how it can be parallelized.

- ▶ Get recommendations on how to make your code follow the best practices for parallelization.

3. Detect and fix defects related to parallelism.

Data race conditions are very hard to detect and debug.

- ▶ Find data race conditions hidden in your code and obtain information about how to fix them.

4. Identifying opportunities for parallelization.

Your code most likely contains many loops which can be parallelized in different ways.

- ▶ Discover which loops are most suitable for parallelization, how they can be classified into common parallel patterns and which is the best parallelization strategy for them.

5. Parallelize opportunities for several technologies and heterogeneous computing platforms.

Different parallel hardware and technologies to exploit it are available.

- ▶ Create new parallel versions of your code using technologies such as OpenMP and OpenACC and targeting different hardware, including multicore CPUs and GPUs.

Join the Early Access Program at
appentra.com/products/parallelware-analyzer



Contact us and get more information:
www.appentra.com
+34 881 015 556
info@appentra.com

```

▶ /parallelware --datascoping --openmp multi --openacc ofload --function compute_energy_for_node --/SC18/ptrainer-samples/LULESHmk
0 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/jnovotny/SC18/ptrainer-samples/LULESHmk/src/lulesh_mk.c SUCCESS

Loop      Variable Kind      Read/write Temporary Pattern      OpenMP      OpenACC
-----
lulesh_mk.c:compute_energy_for_node:3215 r      scalar      rw
lulesh_mk.c:compute_energy_for_node:3215 r      scalar      rw      x
lulesh_mk.c:compute_energy_for_node:3215 tmp     array      wo      x      sparse reduction shared/atomic(tmp) cpyout/atomic(tmp)
lulesh_mk.c:compute_energy_for_node:3215 pt      scalar      rw      x
lulesh_mk.c:compute_energy_for_node:3215 tmp     scalar      rw      x
lulesh_mk.c:compute_energy_for_node:3216 pt      scalar      ro
lulesh_mk.c:compute_energy_for_node:3418 l      scalar      rw
lulesh_mk.c:compute_energy_for_node:3418 tmp     array      wo      x      forall      shared(tmp)      cpyout(tmp)
lulesh_mk.c:compute_energy_for_node:3418 tmp     scalar      rw

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 (ie, it is for it to be declared within the loop body)
Pattern : parallel pattern (read-only, forall, scalar reduction, sparse reduction)
OpenMP : OpenMP scoping (valid values: shared, private, reduction); special values: shared/atomic when atomic directive is also required)
OpenACC : OpenACC scoping (valid values: copy, cpyin, cpyout); special values: copy/atomic, cpyout/atomic when atomic directive is also required)

1 files successfully analyzed and 0 failures in 124 ms

```