# Otter: Tracing & Visualising OpenMP Programs as DAGs with LLVM OMPT

Adam Tuft

adam.s.tuft@durham.ac.uk

# OpenMP Tasks

Task = self-contained unit of work + data

Create → Schedule → Do

- Creation: `task[loop]`
- Synchronisation: `taskgroup`, `taskwait`, `depend(...)`

```
int fibonacci(int n) {
        int i, j;
        if (n<2) return n;
        #pragma omp task shared(i) firstprivate
            (n)
               i = f(n-1);
        #pragma omp task shared(j) firstprivate
            (n)
               j = f(n-2);
        #pragma omp taskwait
        return i+j;
}
```

# The Problem

Want a "big picture" view of task-based code → **how?**

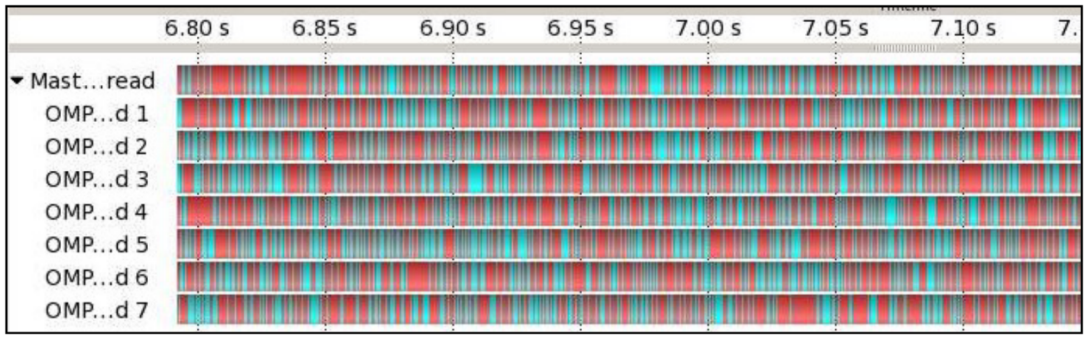
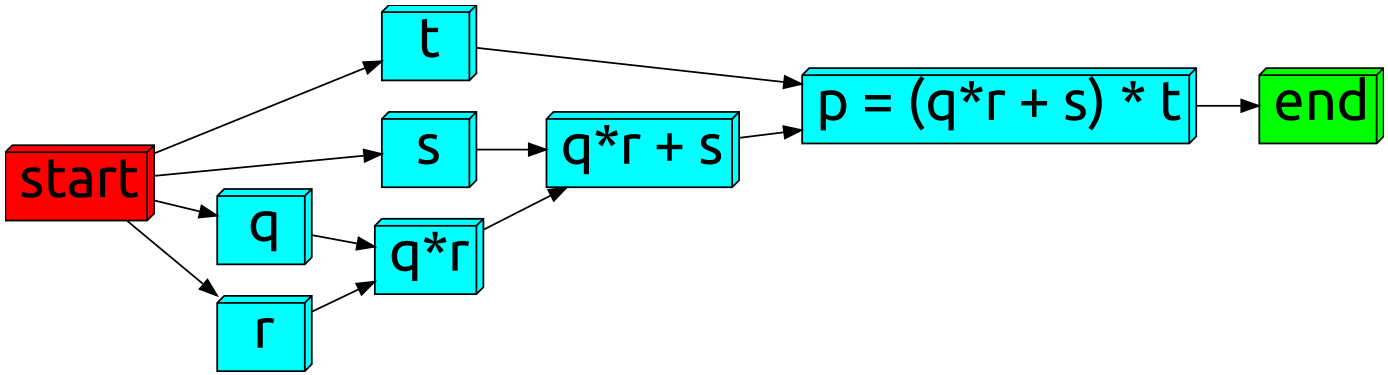



Figure 1: Performance analysis of a task-based Sudoku solver visualised in Vampir. Reproduced from [4].



# Otter

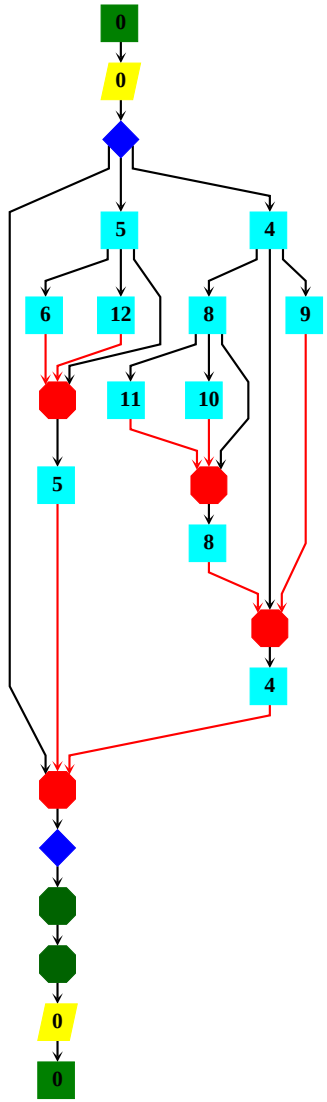
## A tool for visualising task creation and synchronisation

$$\text{OMPT [1]} + \text{OTF2 [2]} \rightarrow \textbf{Otter [6]}$$

- Visualise OpenMP task creation and synchronisation constructs
- Non-invasive
- Observe nested tasks & nested parallelism
- Callback-based (LLVM OMPT)
- OTF2 instrumentation layer

# Example: `fibonacci(5)`

```
int fibonacci(int n) {
        int i, j;
        if (n<2) return n;
        #pragma omp task shared(i)
             firstprivate(n)
                i = f(n-1);
        #pragma omp task shared(j)
             firstprivate(n)
                j = f(n-2);
        #pragma omp taskwait
        return i+j;
}
```



# Example: 2D Euler Equations Solver

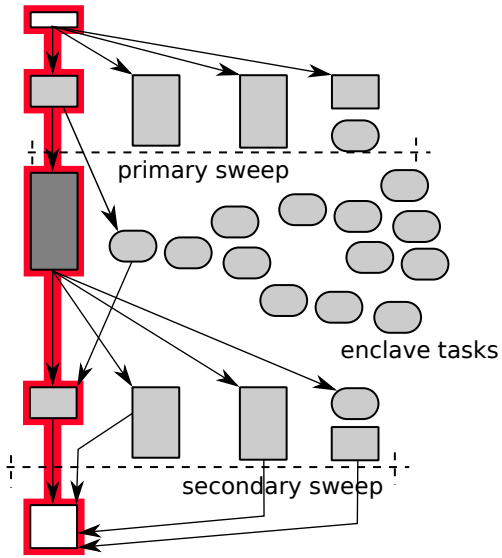



Figure 2: A single timestep in the 2D Euler solver, spawning background tasks for greater concurrency. Reproduced from [5].

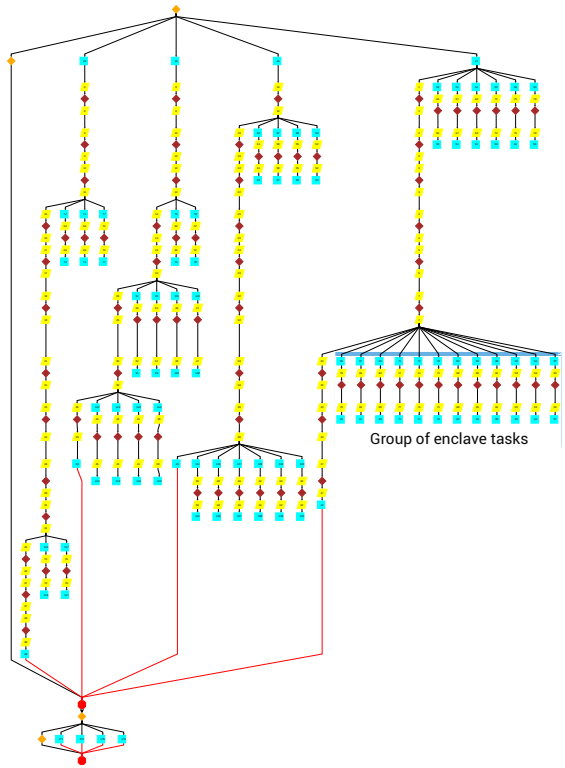
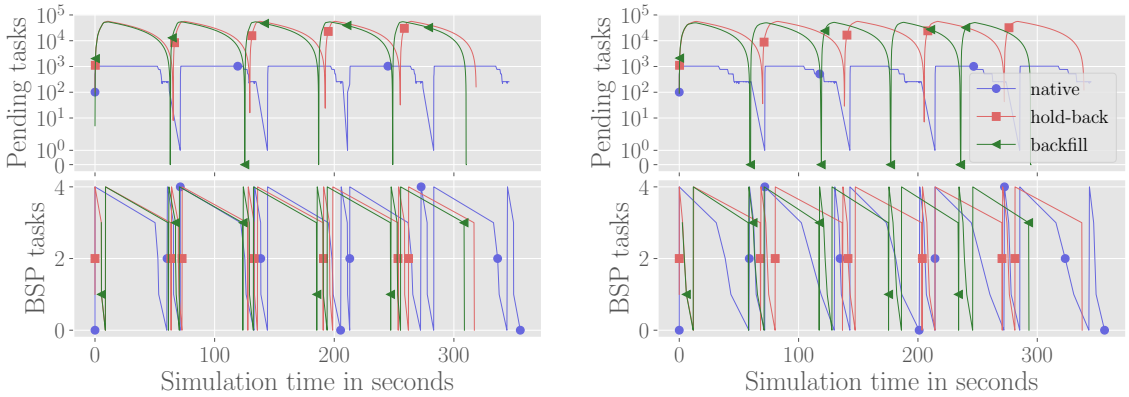



Figure 3: A single timestep observed by Otter. Each task contains a domain traversal and background "enclave" tasks.

# Tasking Inefficiencies Observed in LLVM RTL




(a) Well-balanced setup (b) Ill-balanced setup

Figure 4: Analysis of five time-steps with `OMP_NUM_THREADS=4`. Reproduced from [5].

- Number of queued tasks is capped at c. 1000 (4 threads).
- Pending tasks are processed immediately (unavailable for backfill).
- Need to be able to inform runtime to hold tasks back.

# Future Work

- API for data-driven taskification
- Measure work per task
- Support `depend` clause
- Detect critical path
- Relate to source
- Target tasks
- Device tracing
- Other tasking runtimes

# Conclusion

- Can visualise task creation and synchronisation
- Provide data-driven insight into task-based code
- Unobstructed by particular task scheduling arrangement
- Need outstanding OMPT callbacks in LLVM

# Acknowledgements


- Prof. Tobias Weinzierl
- Dr. Holger Schulz

# Acknowledgements



The ExCALIBUR programme is supported by the UKRI Strategic Priorities Fund.
The programme is led by the Met Office and the Engineering and Physical Sciences Research Council (EPSRC) along with the Public Sector Research Establishment, the UK Atomic Energy Authority (UKAEA) and UK Research and Innovation (UKRI) research councils, including the Natural Environment Research Council (NERC), the Medical Research Council (MRC) and the Science and Technologies Facilities Council (STFC).

# Thank You!

# References I


[1] A. E. Eichenberger et al. "OMPT: An OpenMP Tools Application Programming Interface for Performance Analysis". In: *OpenMP in the Era of Low Power Devices and Accelerators*. Ed. by A. P. Rendell, B. M. Chapman, and M. S. Müller. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 171–185. ISBN: 978-3-642-40698-0. DOI: 10.1007/978-3-642-40698-0_13.

[2] D. Eschweiler et al. "Open Trace Format 2: The Next Generation of Scalable Trace Formats and Support Libraries". In: vol. 22. Jan. 2012, pp. 481–490. ISBN: 9781614990406. DOI: 10.3233/978-1-61499-041-3-481.

[3] OpenMP Architecture Review Board. *OpenMP Application Program Interface Version 5.0*. Nov. 2018. URL: https://www.openmp.org/wp-content/uploads/OpenMP-API-Specification-5.0.pdf.

# References II

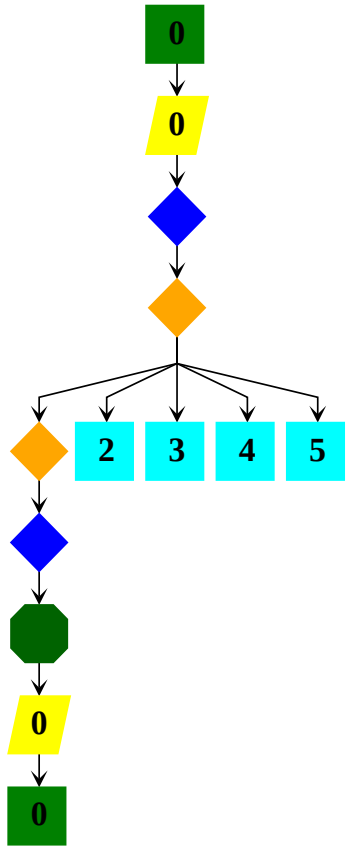

[4] D. Schmidl et al. "Suitability of Performance Tools for OpenMP Task-Parallel Programs". In: *Tools for High Performance Computing 2013*. Ed. by A. Knüpfer et al. Cham: Springer International Publishing, 2014, pp. 25–37. ISBN: 978-3-319-08144-1. DOI: 10.1007/978-3-319-08144-1_3.

[5] H. Schulz et al. "Task Inefficiency Patterns For A Wave Equation Solver". In: *arXiv preprint arXiv:2105.12739* (2021). URL: https://arxiv.org/abs/2105.12739.

[6] A. Tuft. *Otter v0.1*. Version v0.1-submit. Aug. 2021. DOI: 10.5281/zenodo.5259058. URL: https://doi.org/10.5281/zenodo.5259058.

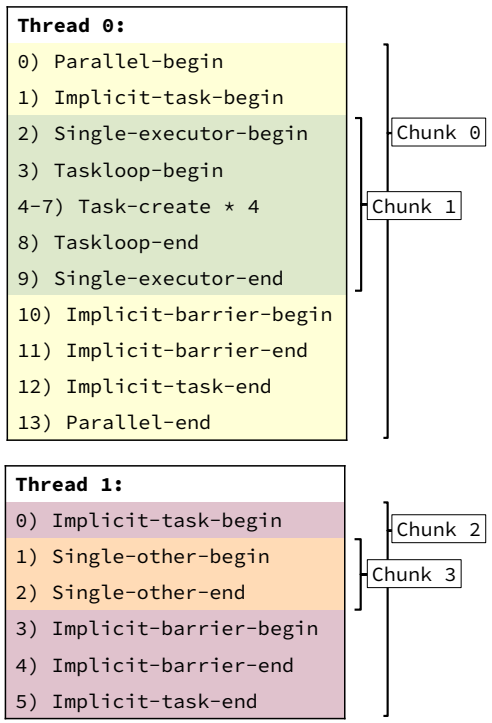
# Example: Single Taskloop

```
#pragma omp parallel num_threads(2)
{
        #pragma omp single nowait
        #pragma omp taskloop nogroup
        for (int j=0; j<4; j++)
        {
                // do work
        }
}
```

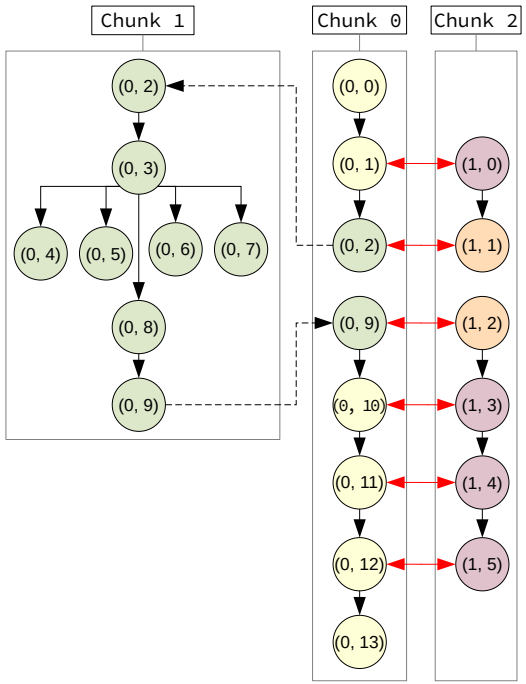


# Post-processing Algorithm

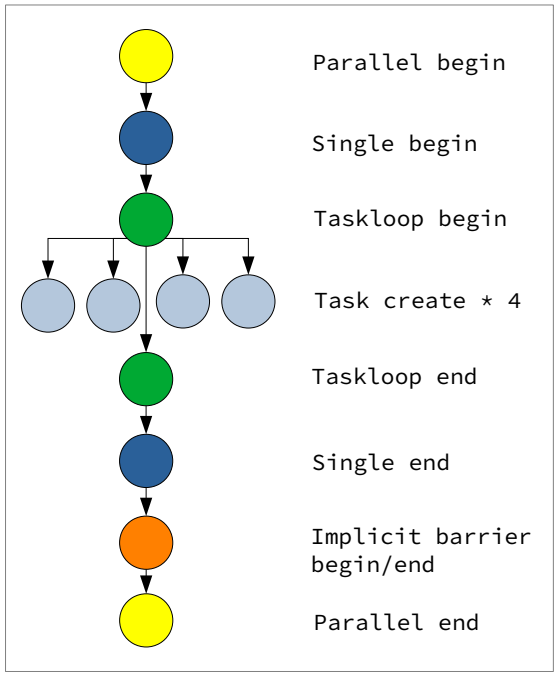
a)Events are grouped in (possibly nested) chunks corresponding to a thread's view of a parallel, single, master or task region.




b)Chunks generate sub-graphs of the final graph. Labels indicate the thread-event pair represented. Dashed black lines show chunk borders and red lines connect nodes representing the same OpenMP construct as viewed by different threads.




c)The disjoint union of the sub-graphs in (b) produces the final graph with equivalent nodes and chunk borders collapsed, and nodes coloured by the type of OpenMP construct they represent.



# Allowed values of ompt_set_callback

Table 4.2 of [3]:

| Return code abbreviation | N | S/P | A | Return code abbreviation | N | S/P | A |
|---|---|---|---|---|---|---|---|
| ompt_callback_thread_begin | | | Y | ompt_callback_mutex_released | Y | Y | Y |
| ompt_callback_thread_end | | | Y | ompt_callback_dependences | Y | Y | Y |
| ompt_callback_parallel_begin | | | Y | ompt_callback_task_dependence | Y | Y | Y |
| ompt_callback_parallel_end | | | Y | ompt_callback_work | Y | Y | Y |
| ompt_callback_task_create | | | Y | ompt_callback_master | Y | Y | Y |
| ompt_callback_task_schedule | | | Y | ompt_callback_target_map | Y | Y | Y |
| ompt_callback_implicit_task | | | Y | ompt_callback_sync_region | Y | Y | Y |
| ompt_callback_target | | | Y | ompt_callback_reduction | Y | Y | Y |
| ompt_callback_target_data_op | | | Y | ompt_callback_lock_init | Y | Y | Y |
| ompt_callback_target_submit | | | Y | ompt_callback_lock_destroy | Y | Y | Y |
| ompt_callback_control_tool | | | Y | ompt_callback_mutex_acquire | Y | Y | Y |
| ompt_callback_device_initialize | | | Y | ompt_callback_mutex_acquired | Y | Y | Y |
| ompt_callback_device_finalize | | | Y | ompt_callback_nest_lock | Y | Y | Y |
| ompt_callback_device_load | | | Y | ompt_callback_flush | Y | Y | Y |
| ompt_callback_device_unload | | | Y | ompt_callback_cancel | Y | Y | Y |
| ompt_callback_sync_region_wait | Y | Y | Y | ompt_callback_dispatch | Y | Y | Y |

N = ompt_set_never, S = ompt_set_sometimes, P = ompt_set_sometimes_paired, A = ompt_set_always

# Interpretation

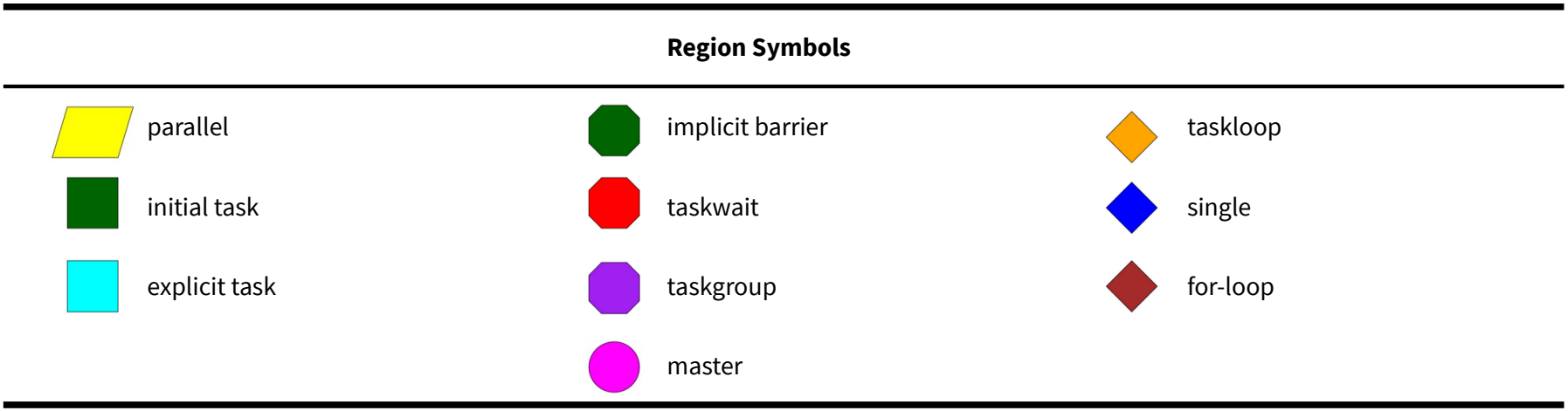
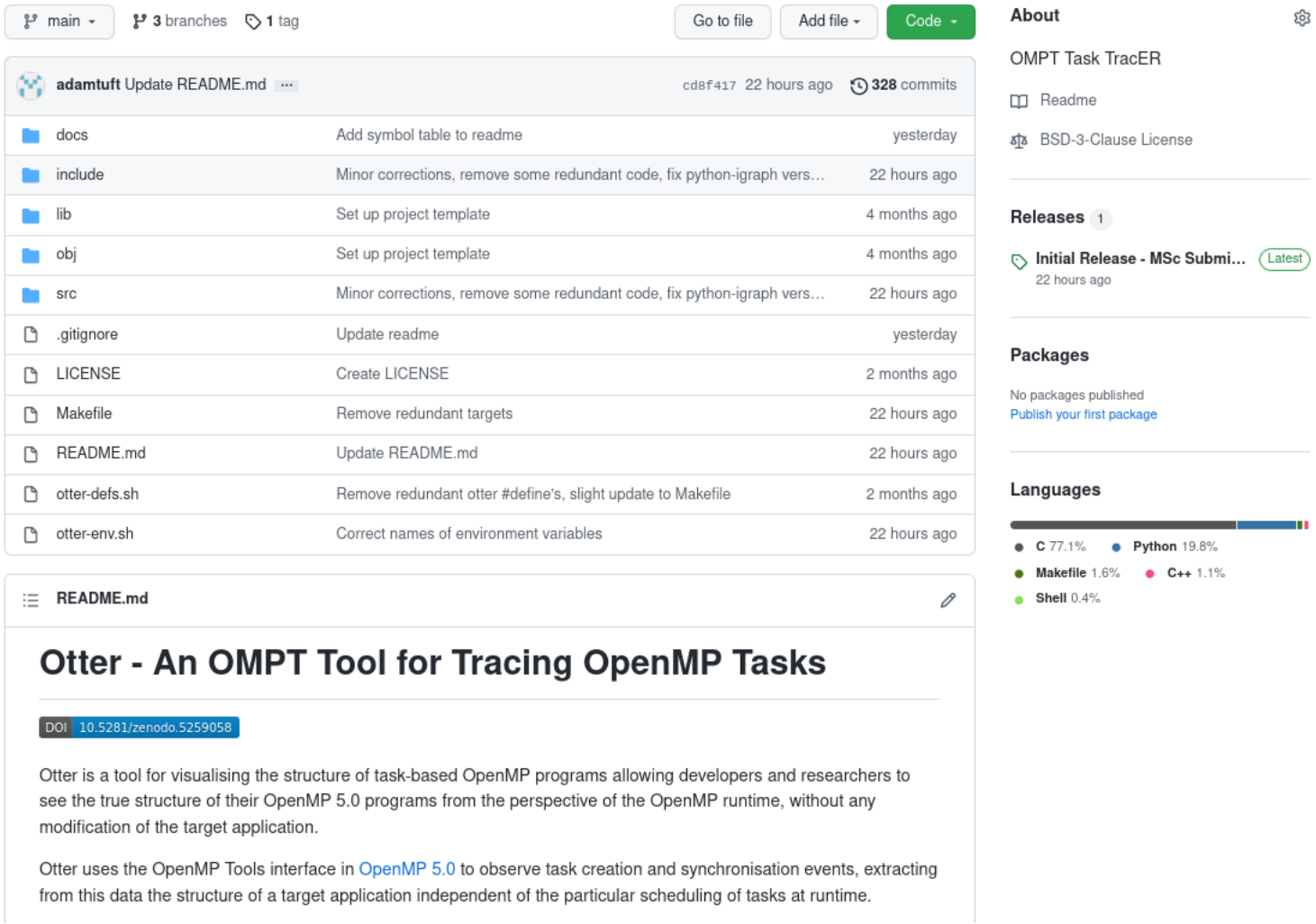



Figure 5: Node style encodes the type of OpenMP construct.



main · 3 branches · 1 tag · Go to file · Add file · Code

adamtuft Update README.md · cd8f417 22 hours ago · 328 commits

| | | |
|---|---|---|
| docs | Add symbol table to readme | yesterday |
| include | Minor corrections, remove some redundant code, fix python-igraph vers… | 22 hours ago |
| lib | Set up project template | 4 months ago |
| obj | Set up project template | 4 months ago |
| src | Minor corrections, remove some redundant code, fix python-igraph vers… | 22 hours ago |
| .gitignore | Update readme | yesterday |
| LICENSE | Create LICENSE | 2 months ago |
| Makefile | Remove redundant targets | 22 hours ago |
| README.md | Update README.md | 22 hours ago |
| otter-defs.sh | Remove redundant otter #define's, slight update to Makefile | 2 months ago |
| otter-env.sh | Correct names of environment variables | 22 hours ago |

README.md

# Otter - An OMPT Tool for Tracing OpenMP Tasks

DOI 10.5281/zenodo.5259058

Otter is a tool for visualising the structure of task-based OpenMP programs allowing developers and researchers to see the true structure of their OpenMP 5.0 programs from the perspective of the OpenMP runtime, without any modification of the target application.

Otter uses the OpenMP Tools interface in OpenMP 5.0 to observe task creation and synchronisation events, extracting from this data the structure of a target application independent of the particular scheduling of tasks at runtime.

About

OMPT Task TracER

Readme

BSD-3-Clause License

Releases 1

Initial Release - MSc Submi… Latest

22 hours ago

Packages

No packages published

Publish your first package

Languages

C 77.1% · Python 19.8% · Makefile 1.6% · C++ 1.1% · Shell 0.4%

github.com/adamtuft/otter