Programming OpenMP

(Using) OpenMP Compilers & Exercises

Christian Terboven

RWTH AACHEN UNIVERSITY

Using OpenMP compilers

Production Compilers w/ OpenMP Support

- GCC
- clang/LLVM
- Intel Classic and Next-gen Compilers
- AOCC, AOMP, ROCmCC
- IBM XL
- … and many more

- See https://www.openmp.org/resources/openmp-compilers-tools/ for a list

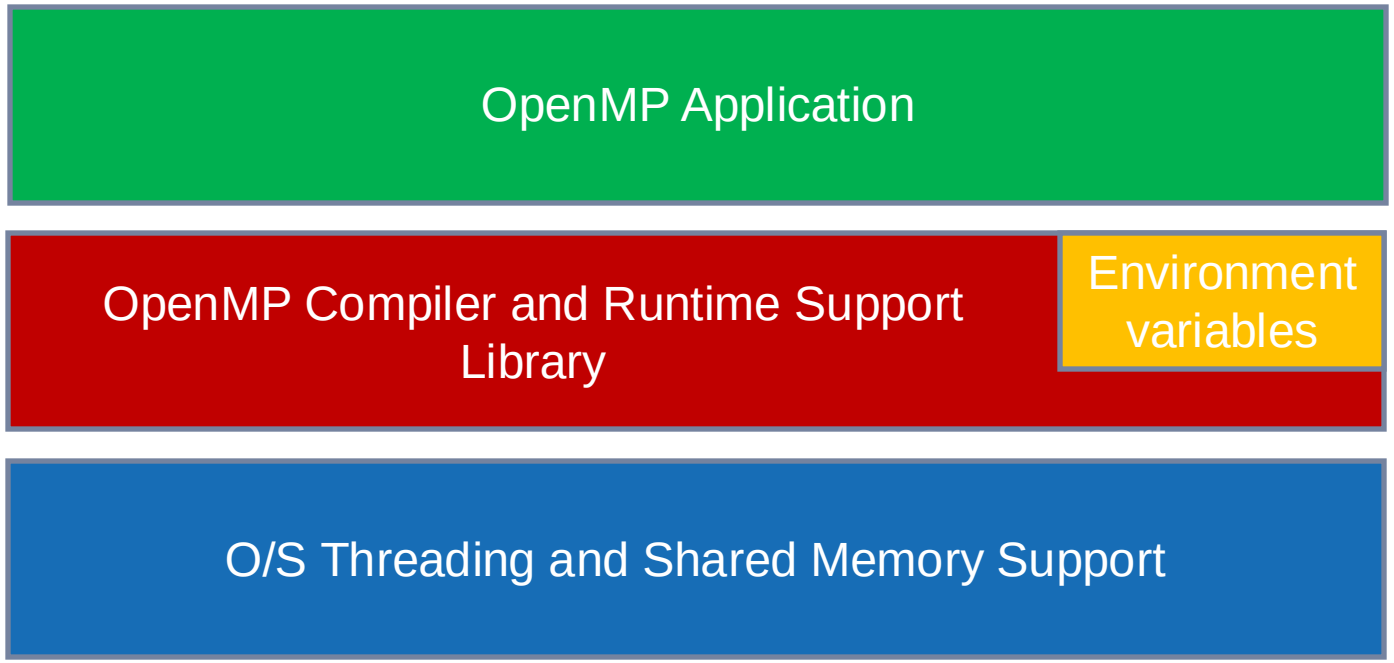
Compiling OpenMP

- Enable OpenMP via the compiler's command-line switches
  - → GCC: `-fopenmp`
  - → clang: `-fopenmp`
  - → Intel: `-fopenmp` or `–qopenmp` (classic) or `–fiopenmp` (next-gen)
  - → AOCC, AOCL, ROCmCC: `-fopenmp`
  - → IBM XL: -qsmp=omp
- Switches have to be passed to both compiler and linker:

```
$ gcc [...] -fopenmp -o matmul.o -c matmul.c
$ gcc [...] -fopenmp -o matmul matmul.o
$./matmul 1024
Sum of matrix (serial):   134217728.000000, wall time 0.413975, speed-up 1.00
Sum of matrix (parallel): 134217728.000000, wall time 0.092162, speed-up 4.49
```

Understanding OpenMP compilers

OpenMP implementation

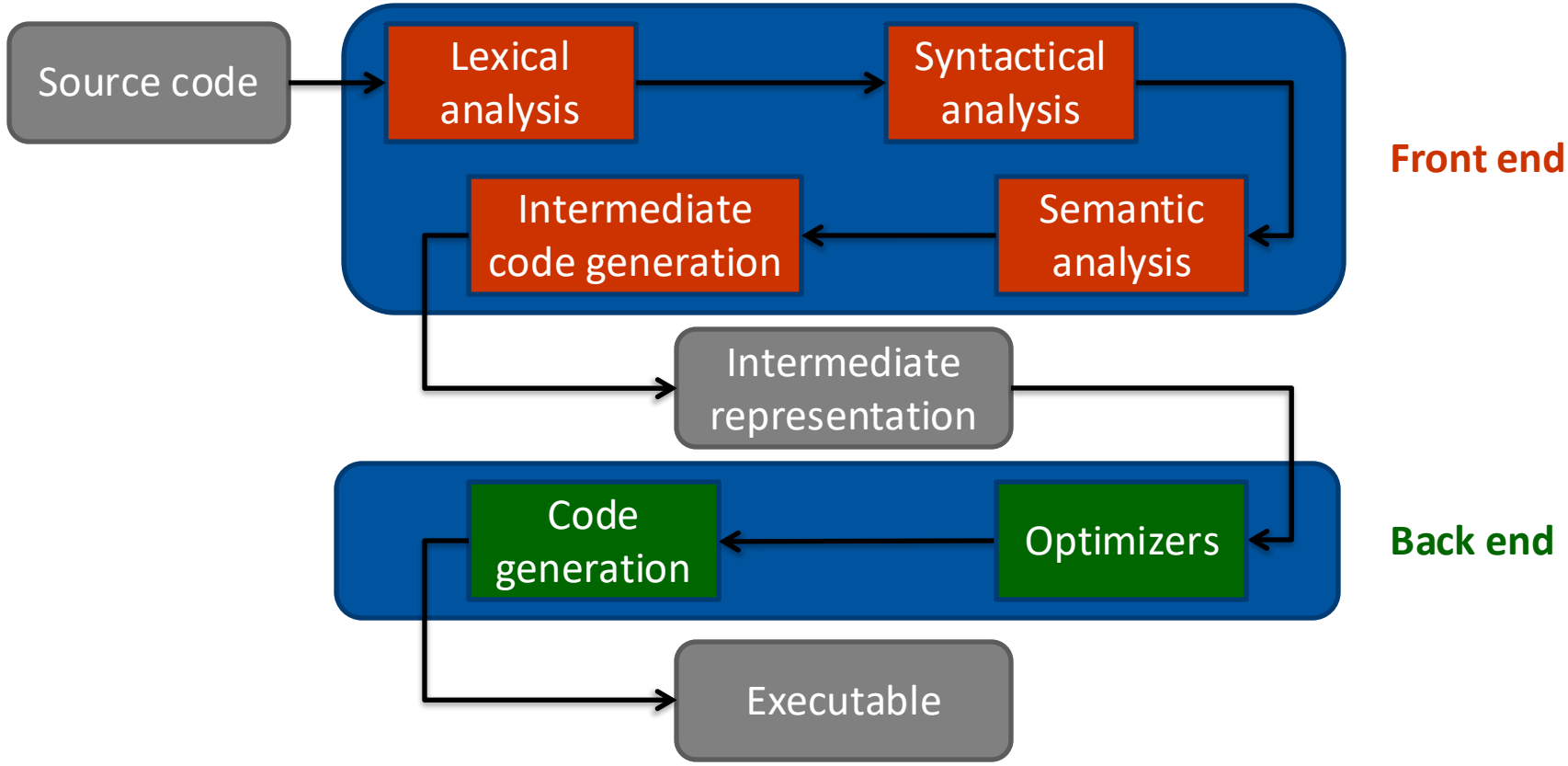


- Before OpenMP: Use of pthreads (POSIX threads) was a common approach for shared memory programming
  - POSIX: Portable Operating System Interface [for Unix]
  - OpenMP runtime manages OpenMP threads
  - OpenMP threads are (commonly) mapped to `pthreads`

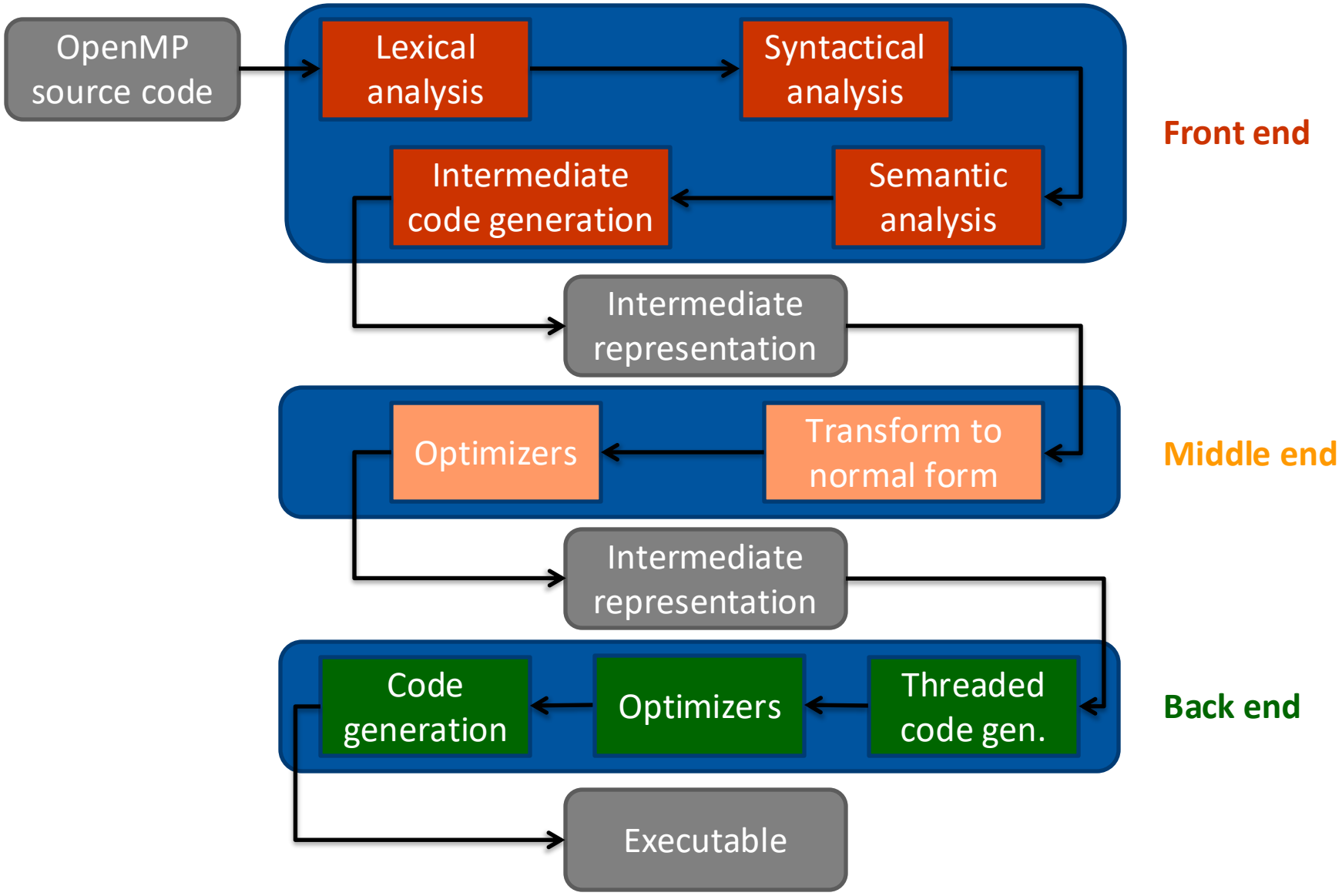
Runtime support library

- Runtime Support Library (C and C++):
  - Manages OpenMP threads, task, scheduling, etc
  - If OpenMP is enabled during compilation, the preprocessor symbol `_OPENMP` is defined
  - To use the OpenMP runtime library, the header `omp.h` has to be included
  - `omp_set_num_threads(int)`: The specified number of threads will be used for the parallel region encountered next
  - `int omp_get_num_threads`: Returns the number of threads in the current team
  - `int omp_get_thread_num()`: Returns the number of the calling thread in the team, the Master has always the id 0
  - Additional functions are available, e.g. to provide locking functionality.

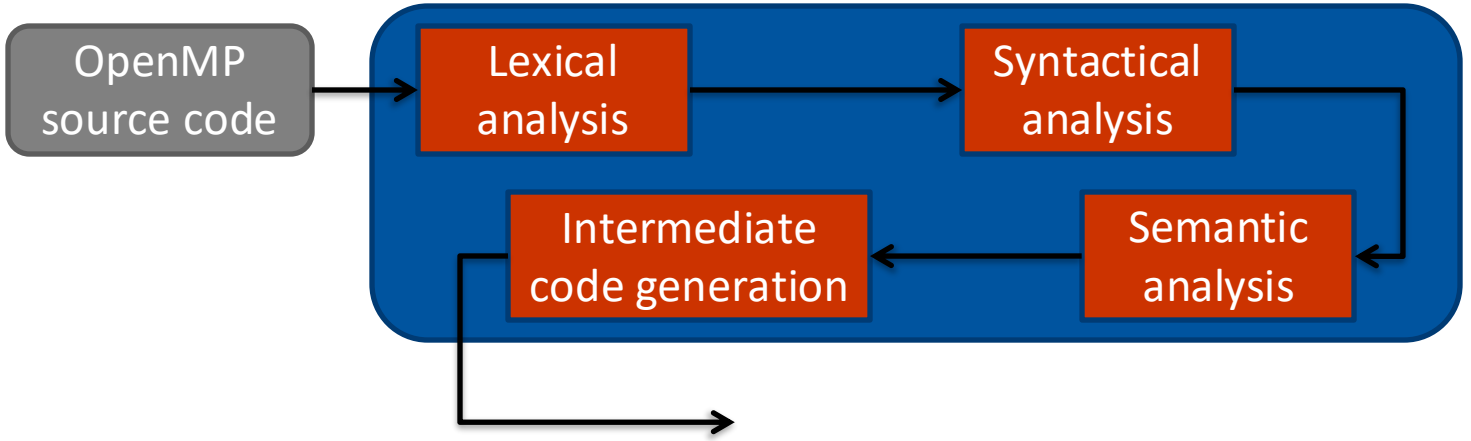
Compiler architecture



OpenMP compiler architecture

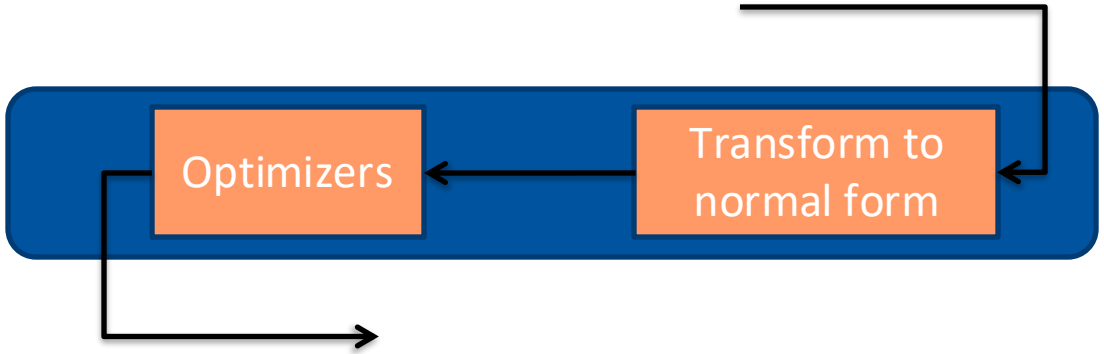


OpenMP compiler front end



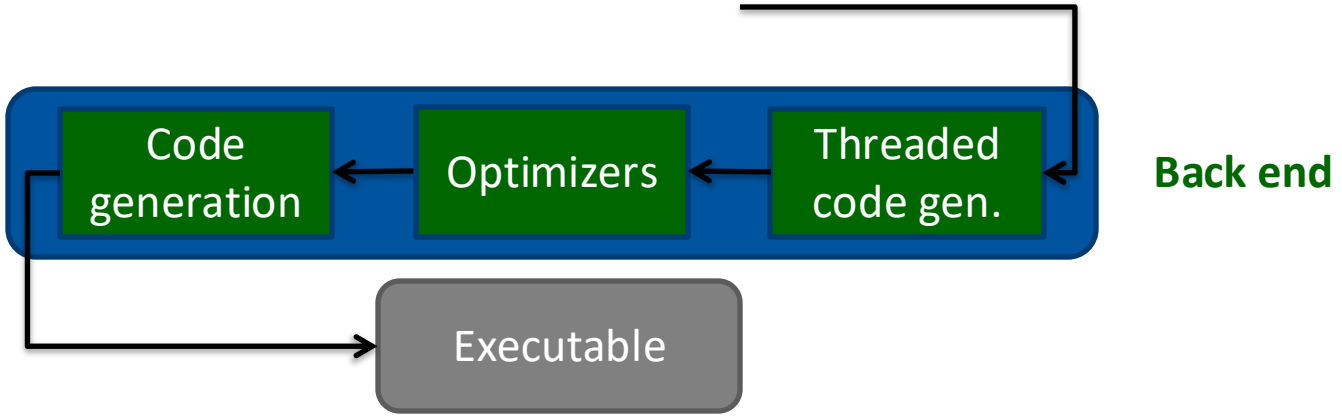
- Parse and analyze OpenMP directives (and code written in base language)
- Perform syntax and semantics checks, e.g.
  - correct usage of pragmas (e.g. loop after worksharing, s. next slides)
  - correct usage and placement of clauses
  - correct nesting of constructs
  - properties of loop counters

OpenMP compiler middle end



- Transform OpenMP into normal form
  - Easier code generation in back end
    - Everything is explicit after this phase
    - Less OpenMP constructs to provide code gen patterns for
  - OpenMP-style optimizations
    - E.g. remove duplicate barriers
    - Merge/split parallel regions
    - Anything that the user cannot detect ☺

OpenMP compiler back end



- Create multi-threaded code
  - Create OpenMP functions by function splicing / outlining
  - Replace some OpenMP constructs with runtime calls (e.g. barriers, critical → see later)
  - Replace some OpenMP constructs with specialized low-level code (e.g. prepare for atomic instructions)
  - Create boiler plate code to invoke parallel regions

Exercises

Exercises: Overview

Exercise no.	Exercise name	OpenMP Topic	Day / Order (proposal)
1	Hello World	Getting started	Start with this
2	Pi	Worksharing, Scoping	First day
3	Jacobi	Worksharing, Scoping, NUMA	First day
4	Fibonacci	Tasking	Second day
5	Work-Distribution	Worksharing	First day, review on Second day
6	Min/Max	Worksharing, Reduction	First day
7	QuickSort	Tasking	Second day
8	Primes	Correctness w/ Tool	Only if you are interested in this topic
9	Mandelbrot	Correctness by Hand	Only if you are interested in this topic