GIANT
INNOVATION CAMPUS

Programming-Model Centric Debugging for OpenMP

Nano2017/Dema Project Meeting

June 24th, 2015
Université Paris Jussieu

informatics mathematics
Inria

Introduction

Compiler Optimization and Runtime SystEms

OpenMP and GDB

- Hard to control the step-by-step execution
- Hard to understand the current state of the different threads

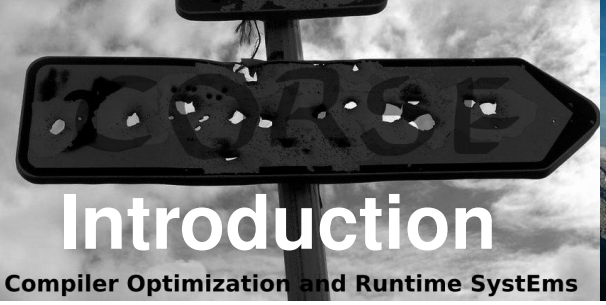
$\Rightarrow$ No high-level vision of the application by GDB

OpenMP and GDB

- Hard to control the step-by-step execution
- Hard to understand the current state of the different threads

$\Rightarrow$ No high-level vision of the application by GDB

```
Id Target Id         Frame
4  Thread 0x..7700 in do_spin () from libgomp.so
3  Thread 0x..8700 in do_spin () from libgomp.so
2  Thread 0x..9700 in GOMP_barrier () from libgomp.so
1  Thread 0x..a780 main._omp_fn.0 () at parallel-demo.c:15
```



Introduction

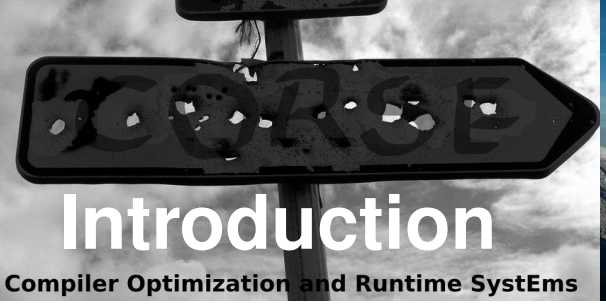
Compiler Optimization and Runtime SystEms

OpenMP and GDB

- Hard to control the step-by-step execution
- Hard to understand the current state of the different threads

$\Rightarrow$ No high-level vision of the application by GDB

```
Thread 1:
#0  main._omp_fn.0 ()                 at parallel-demo.c:15
#1  0x00bcaf in GOMP_parallel () from libgomp.so
#2  0x0009cb in main ()               at parallel-demo.c:6
```

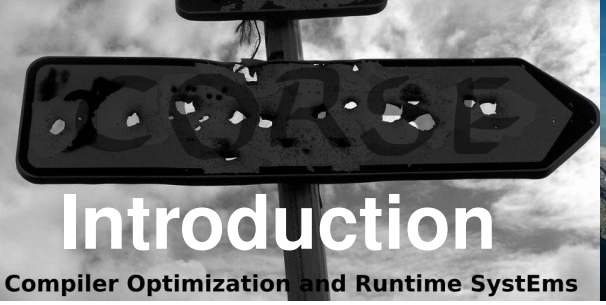


OpenMP and GDB

- Hard to control the step-by-step execution
- Hard to understand the current state of the different threads

$\Rightarrow$ No high-level vision of the application by GDB

```
Thread 2:
#0  0x011cf9 in GOMP_barrier ()          from libgomp.so
#1  0x400a15 in main._omp_fn.0 ()        at parallel-demo.c:11
#2  0x00f45e in gomp_thread_start ()     from libgomp.so
#3  0x80761a in start_thread ()          from libpthread.so
#4  0x106bdd in clone ()                 from libc.so
```



OpenMP and GDB

⇒ No high-level vision of the application by GDB

```
(gdb) list
17  /* <-- current thread is here --> */
18  #pragma omp critical
19  {
20     printf("@%d Inside critical zone\n", id);
21  }

(gdb) next
@4 Inside critical zone
@2 Inside critical zone
20  printf("@\%d Inside critical zone\n", id);
(gdb) # I wanted to be the first :'(
```

Introduction

Compiler Optimization and Runtime SystEms

OpenMP and GDB

$\Rightarrow$ No high-level vision of the application by GDB

The problem is ...

If you can't control it, you can't study it.

If you can't understand it, you can't debug it!

Introduction

Compiler Optimization and Runtime SystEms

What can we do against that?

$\Longrightarrow$ upgrade to mcGDB !

but that requires a bit of work, so let's study what can be done first.

Outline

Compiler Optimization and Runtime SystEms

Outline

Compiler Optimization and Runtime SystEms

Current-State Visualization

Compiler Optimization and Runtime SystEms

Current-state Visualization

- already introduced last time
- worked on better integration inside mcGDB

Current-state Visualization

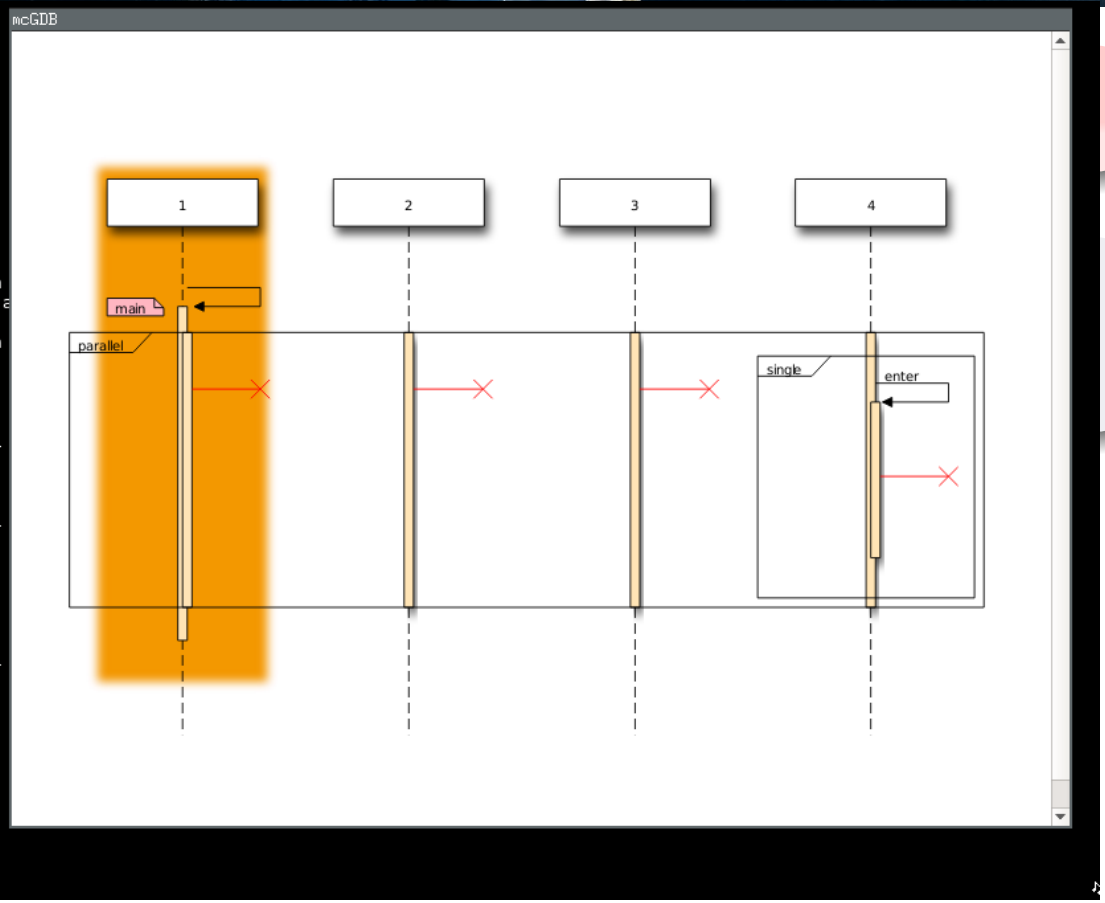
- already introduced last time
- worked on better integration inside mcGDB

```
(gdb) gui start
(gdb) gui show
(gdb) gui control
(gdb) gui quit
```

Current-State Visualization

Compiler Optimization and Runtime SystEms

```
(parallel-demo.c:11 gdb)
$ gui start
Please run this command to connect the GUI:
python2 /home/kevin/travail/Python/mcgdb/toolbox/graphdisplay.py  &
(parallel-demo.c:11 gdb)
$ info workers
> Worker #1: ParallelJob #1
  Worker #2: ParallelJob #1
  Worker #3: ParallelJob #1
  Worker #4: ParallelJob #1 > SingleJob #1
(parallel-demo.c:11 gdb)
$ info threads
  Id   Target Id         Frame
  4    Thread 0x7ffff6dc4700 (LWP 18209) "Worker #4" 0x0000000000400a08 in
  3    Thread 0x7ffff75c5700 (LWP 18208) "Worker #3" GOMP_single_start () a
  2    Thread 0x7ffff7dc6700 (LWP 18207) "Worker #2" omp_get_thread_num ()
* 1    Thread 0x7ffff7dc7780 (LWP 18197) "Worker #1" 0x0000000000400a08 in
(parallel-demo.c:11 gdb)
$ thread apply all where

Thread 4 (Thread 0x7ffff6dc4700 (LWP 18209)):
#0  0x0000000000400a08 in ParallelJob #1::main<0> () at parallel-demo.c:11

Thread 3 (Thread 0x7ffff75c5700 (LWP 18208)):
#0  #pragma omp single_start ()
#1  0x0000000000400a08 in ParallelJob #1::main<0> () at parallel-demo.c:11

Thread 2 (Thread 0x7ffff7dc6700 (LWP 18207)):
#0  omp_get_thread_num ()
#1  0x00000000004009e9 in ParallelJob #1::main<0> () at parallel-demo.c:9

Thread 1 (Thread 0x7ffff7dc7780 (LWP 18197)):
#0  0x0000000000400a08 in ParallelJob #1::main<0> () at parallel-demo.c:11
#2  #pragma omp parallel ()
#4  0x00000000004009cb in main () at parallel-demo.c:6
(parallel-demo.c:11 gdb)
$
```



Current-state Visualization

- already introduced last time
- worked on better integration inside mcGDB

```
(gdb) gui start
(gdb) gui show
(gdb) gui control
(gdb) gui quit
```

Current-state Visualization

- already introduced last time
- worked on better integration inside mcGDB

(gdb) gui start

→ Qt-window popup controlled with Javascript

Current-state Visualization

- already introduced last time
- worked on better integration inside mcGDB

(gdb) gui start

→ Qt-window popup controlled with Javascript

→ `Please run this command to connect the GUI: python2 .../mcgdb/toolbox/graphdisplay.py`

- GDB is a complex process and can freeze after the fork...

Current-State Visualization

Compiler Optimization and Runtime SystEms

Current-state Visualization

- already introduced last time
- worked on better integration inside mcGDB

(gdb) gui control

- allows interactivity (= control of GDB) in the GUI
  - GDB is not thread-safe ⇒ CLI + GUI in a thread == segfault

Current-State Visualization

Compiler Optimization and Runtime SystEms

Current-state Visualization

- already introduced last time
- worked on better integration inside mcGDB

(gdb) gui control

- allows interactivity (= control of GDB) in the GUI
  - GDB is not thread-safe ⇒ CLI + GUI in a thread == segfault
- switch threads by clicking on the boxes
- stack-trace on mouse hover (soon)

Visualization

Current-state Visualization

- already introduced last time
- worked on better integration inside mcGDB

Misc...

- auto-refresh on prompt display
- remote connection to the GUI (tcp/ip, via from Python stdlib `SyncManager`)

* illustrations in the following section

Current-State Information

Compiler Optimization and Runtime SystEms

(gdb) info workers

```
> Worker #1: ParallelJob #1 > CriticalJob #1
  Worker #2: ParallelJob #1 > Barrier #1
  Worker #3: ParallelJob #1
  Worker #4: ParallelJob #1 > Barrier #1
```

Current-State Information

Compiler Optimization and Runtime SystEms

(gdb) where

```
#0  #pragma   omp critical_start ()
#1  0x0400a1a in  ParallelJob #1::main<0> () at parallel-demo.
#3  #pragma   omp parallel ()
#5  0x4009cb  in  main () at parallel-demo.c:6
```

Current-State Information

Compiler Optimization and Runtime SystEms

(gdb) where

```
#0  #pragma   omp critical_start ()
#1  0x0400a1a in  ParallelJob #1::main<0> () at parallel-demo.
#3  #pragma   omp parallel ()
#5  0x4009cb  in  main () at parallel-demo.c:6
```

(gdb) where no-filter

```
#0 GOMP_critical_start ()          at libgomp/critical.c:36
#1 0x0400a1a in main._omp_fn.0 () at parallel-demo.c:18
#2 0x7df94dc in GOMP_parallel_tramp () at omp_preload.c:125
#3 0x7bb4caf in GOMP_parallel ()  at libgomp/parallel.c:168
#4 0x7df953c in GOMP_parallel ()  at omp_preload.c:136
#5 0x04009cb in main ()           at parallel-demo.c:6
```

Outline

Compiler Optimization and Runtime SystEms

Implementation Challenges

Compiler Optimization and Runtime SystEms

Controling Parallel Threads is Hard ...

... and I never did it before !

- Dataflow, components, etc. are not SPMD/SIMD!

Controling Parallel Threads is Hard ...

... and I never did it before !

- Dataflow, components, etc. are not SPMD/SIMD!
- GDB/Python is bad at switch-and-continuing threads:
  e.g., to stop after a barrier:
  - set a BP on barrier function
  - continue until (all the threads -1) hit the barrier
  - when the last thread arrives:
    - activate scheduler-locking (= run only one thread at a time)
    - for all the threads:
      switch to the thread
      continue until the end of the barrier function
- should work in theory, but too hacky in practice.

---

a

Implementation Challenges

Compiler Optimization and Runtime SystEms

Controling Parallel Threads is Hard ...

... and I never did it before !

- Dataflow, components, etc. are not SPMD/SIMD!
- GDB/Python is bad at switch-and-continuing threads:
  e.g., to stop after a barrier:
  - set a BP on barrier function
  - continue until (all the threads -1) hit the barrier
  - when the last thread arrives:
    - activate scheduler-locking (= run only one thread at a time)
    - for all the threads:
      switch to the thread ← forbidden ↓ :-([a]
      continue until the end of the barrier function
- should work in theory, but too hacky in practice.

[a] *Thou shalt not alter any data within gdb or the inferior* (gdbdoc 23,2,2,20)

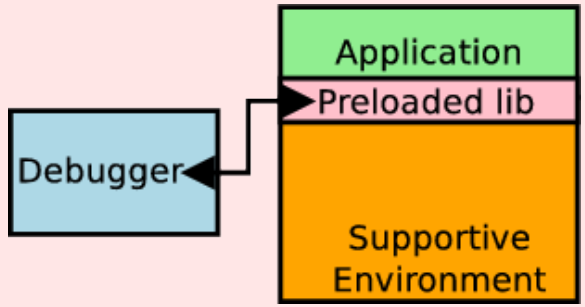
Implementation Challenges

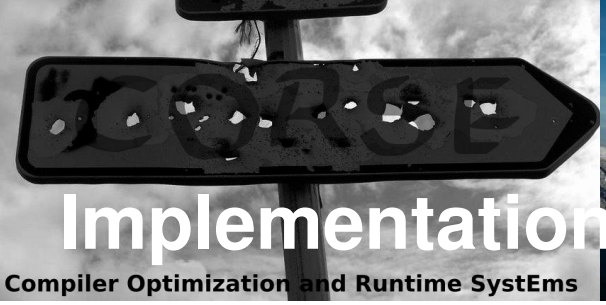
Controling Parallel Threads is Hard ...

... and I never did it before !

`libmcgdb_ldpreload_gomp.so` to the rescue!

- dynamically inserted btw app. and lib.
- transparent (mostly < gdb 7.8)
- tied to library implementation/ABI :-(





Implementation Challenges

Compiler Optimization and Runtime SystEms

Controling Parallel Threads is Hard ...

... and I never did it before !

`libmcgdb_ldpreload_gomp.so` to the rescue!

```
void GOMP_barrier (void) {
  real_GOMP_barrier();
  mcgdb_thread_can_run(&mcgdb_can_pass_barrier);
}
```

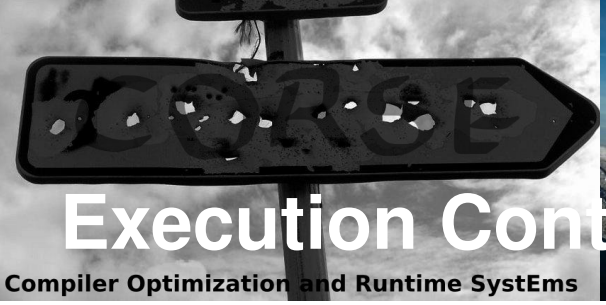
```
(gdb) set mcgdb_can_pass_barrier = 0
// wait for everybody
(gdb) set mcgdb_can_pass_barrier = 1
(gdb) thread apply all finish #(twice)
```

Execution Control

Compiler Optimization and Runtime SystEms

Controlling the Execution Flows

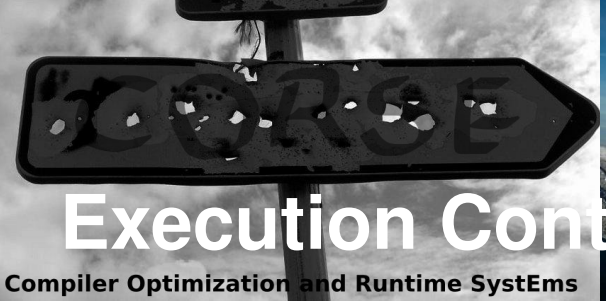
- Navigating intuitively in the execution
- Deterministic (predictable) step-by-step



Execution Control: General Commands

(gdb) omp start

Continues the execution until the beginning of the first parallel zone.



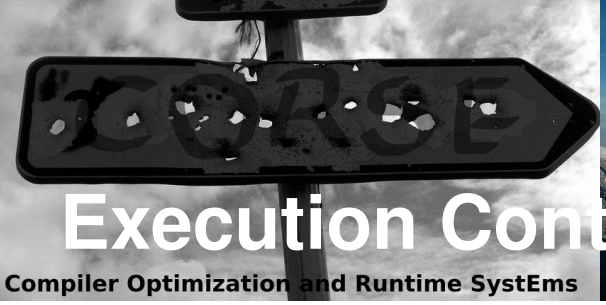
Execution Control: General Commands

(gdb) omp start

Continues the execution until the beginning of the first parallel zone.

(gdb) omp next <zone>

Continues the execution until the next OpenMP <zone>.
(`zone ∈ {single, critical, task, sections, barrier, master}`)



Execution Control: General Commands

(gdb) omp start

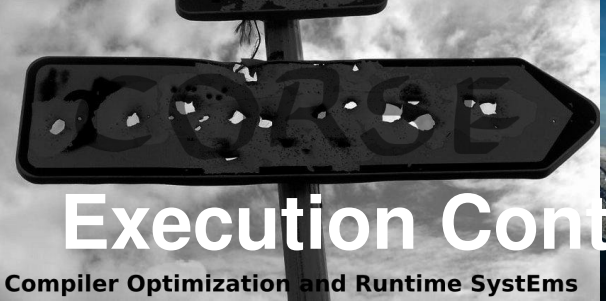
Continues the execution until the beginning of the first parallel zone.

(gdb) omp next <zone>

Continues the execution until the next OpenMP <zone>.
(`zone ∈ {single, critical, task, sections, barrier, master}`)

(gdb) omp step

Continues the exec. until one thread starts working on the current zone.



Execution Control: General Commands

(gdb) omp start

Continues the execution until the beginning of the first parallel zone.

(gdb) omp next <zone>

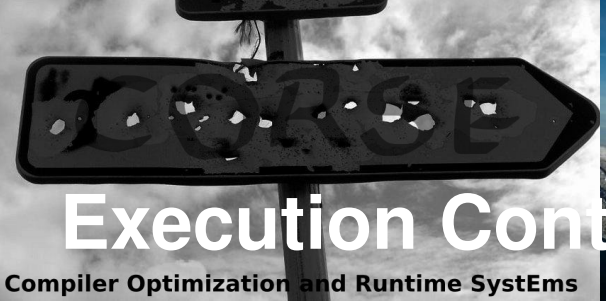
Continues the execution until the next OpenMP <zone>.
(`zone ∈ {single, critical, task, sections, barrier, master}`)

(gdb) omp step

Continues the exec. until one thread starts working on the current zone.

(gdb) omp all_out

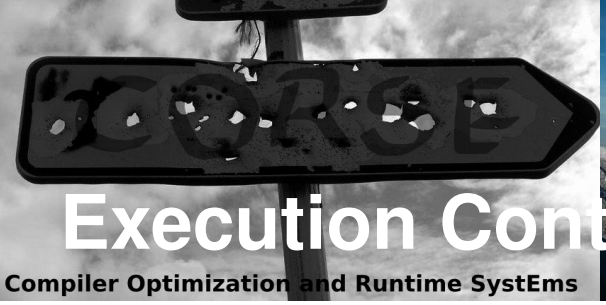
Continues the exec. until all the threads are right after of the current zone.



Execution Control: Zone-Specific Commands (Sections)

(gdb) omp sections new

Catchpoint on the beginning of section zones.



Execution Control: Zone-Specific Commands (Sections)

(gdb) omp sections new

Catchpoint on the beginning of section zones.

(gdb) omp sections step-by-step

Catchpoint on sections' execution.
Activates GDB's scheduler-locking for the zone.

Execu ... ections)

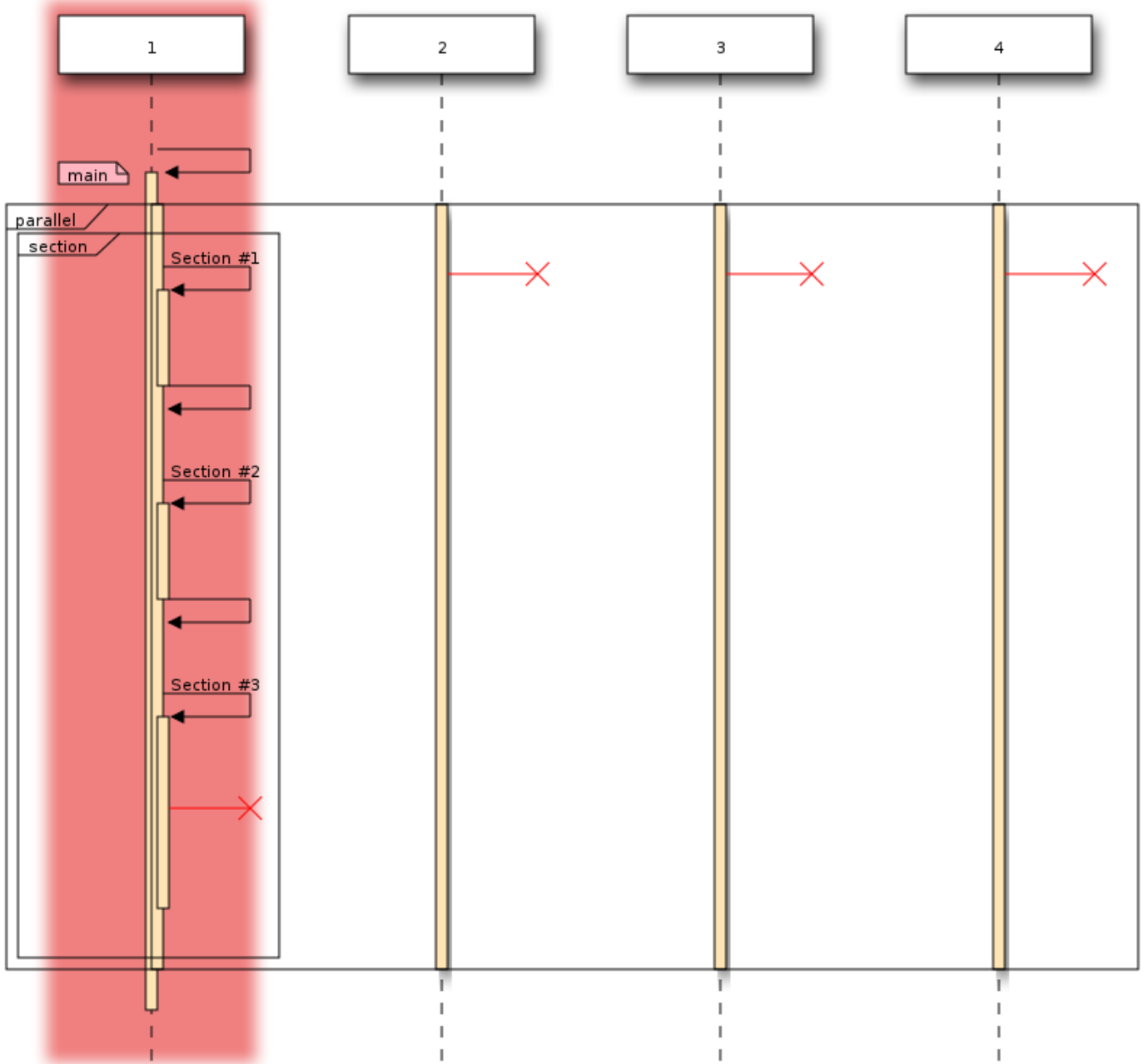
(gdb) c

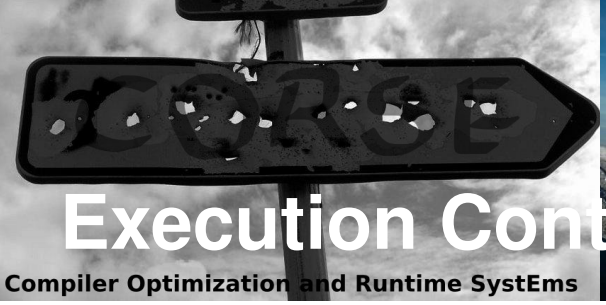
Catchpo

(gdb) c

Catchpo

Activate





Execution Control: Zone-Specific Commands (Sections)

(gdb) omp sections new

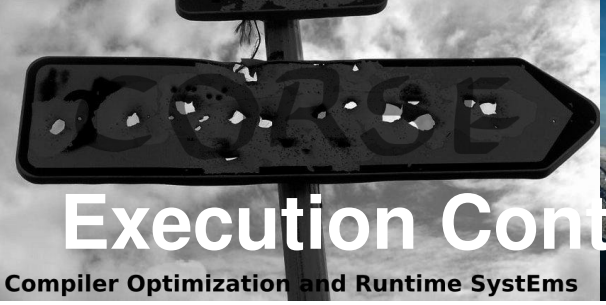
Catchpoint on the beginning of section zones.

(gdb) omp sections step-by-step

Catchpoint on sections' execution.
Activates GDB's scheduler-locking for the zone.

(gdb) omp sections finish

Continues the execution until the end of the section zone.



Execution Control: Zone-Specific Commands (Critical)

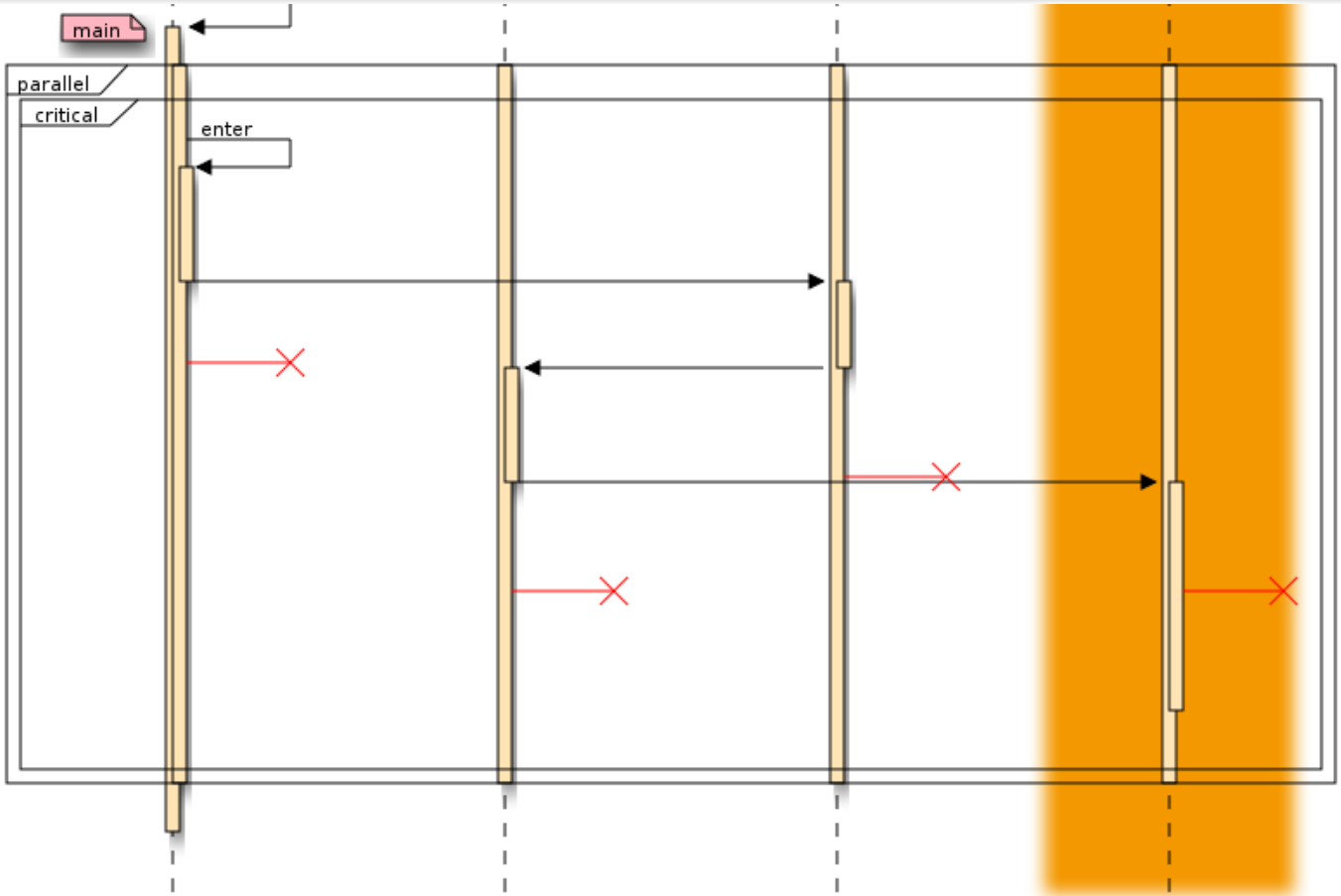
(gdb) omp critical next

Continues the execution until the next thread enters the critical zone.

Execution Control: Zone-Specific Commands (Critical)

(gdb) omp critical next

Continues the execution until the next thread enters the critical zone.

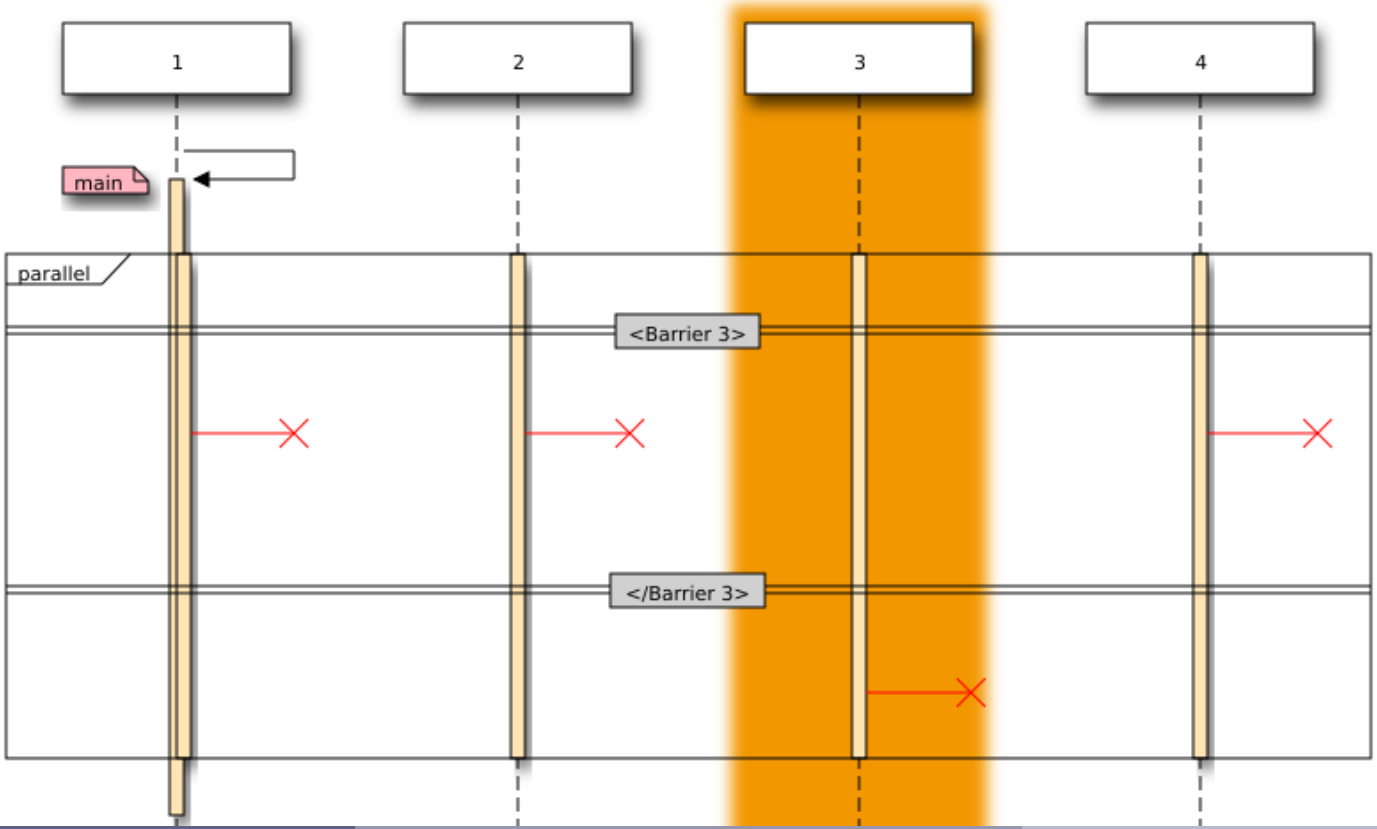


Execution Control: Zone-Specific Commands (Barrier)

(gdb) omp barrier pass

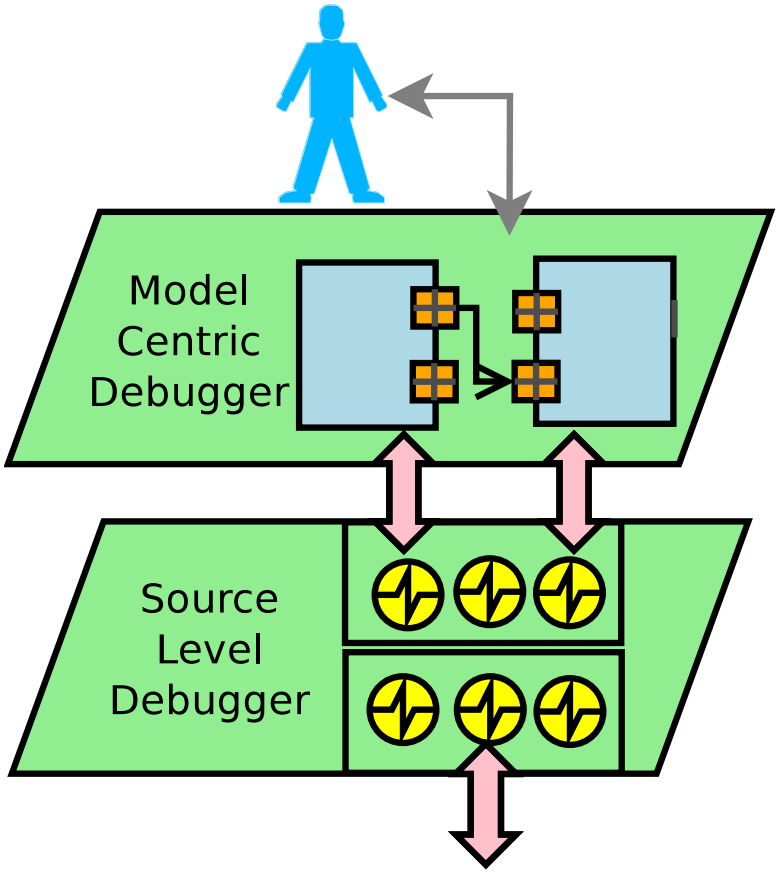
Continues the exec. until all the threads are right after the current barrier.

(not reflected in the illustration below)



Outline

Compiler Optimization and Runtime SystEms



Interaction

Representation

Capture

- Core codebase: `Capture` ⇒ `Representation`
- `Interaction` is open to improvements
- (and `Capture` is subject to replacement)

Aspect-Based Extensions

Compiler Optimization and Runtime SystEms

```
# representation object, methods called by capture
class SectionJob(aspect.Tracker):
  def __init__(self, parallel_job, worker, count=0):
    self.has_completed = False; self.sections = {}

  def work_on_section(self, worker, section_id):
    if section_id != 0:
      self.sections[worker] = section_id
      worker.work(self, start=True)
    else:
      worker.work(self, stop=True)

  def completed(self):
    self.has_completed = True
```

Aspect-Based Extensions

Compiler Optimization and Runtime SystEms

```
@Tracks(representation.SectionJob)
class SectionJobTracker:              # Sequence Diagram builder
  def __init__(this):
    this.block = Block(this.args.parallel_block, "section")
    this.block.add_node(this.args.node)

  def work_on_section(this):
    this.block.add_node(node)
    if this.args.section_id != 0:
      this.block.enter(node, label="Section #{section_id}")
      this.working.add(node)

  def completed(this):
    this.block.finish()
```


```
@Tracks(representation.SectionJob)
class SectionJobTracker:                    # Next catchpoint helper
  def __init__(this):
    check_nexting("sections")

  def work_on_section(this):
    if (this.args.section_id != 0):
      check_stepping("inside Section #{section_id}"))

  def completed(this):
    check_stepping_out("Section zone")
```

Aspect-Based Extensions

Compiler Optimization and Runtime SystEms

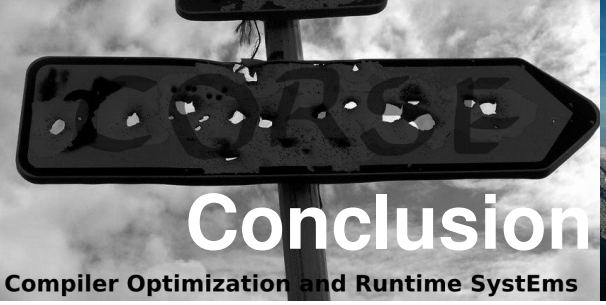
```
@Tracks(representation.SectionJob)
class SectionJobTracker:                          # Graph builder
  def __init__(this, before=False):
    this.job = MultiTask(thread, "sect", "Section")

    inner = this.job.internal_task("inner", shape="point")
    this.exit_task = this.job.internal_task("exit_task")
    inner.happened_after(thread.task)
    this.exit_task.happened_after(inner)

  def work_on_section(this):
    if this.args.section_id != 0:
      thread.moveTo(this.job.internal_task())
    else:
      thread.moveTo(this.exit_task, from_start=False)
```

Outline

Compiler Optimization and Runtime SystEms

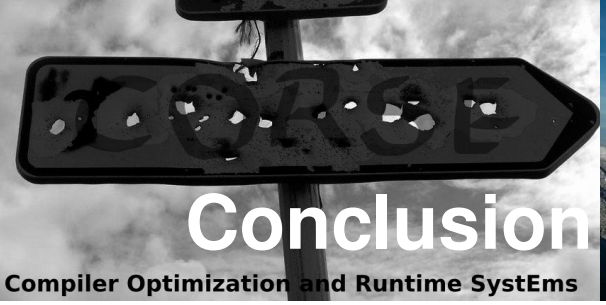
Outline

Compiler Optimization and Runtime SystEms



Conclusion and Future Work

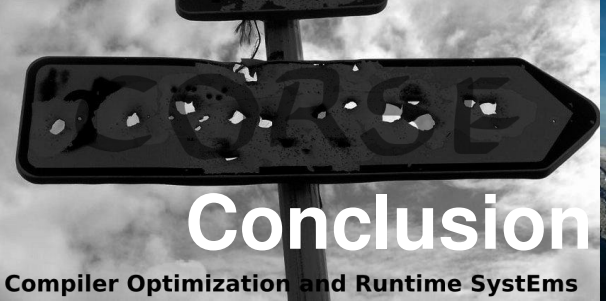
- Everything documented in Dema website (private part)
  `http://dema.gforge.inria.fr/mcgdb/openmp.html`



Conclusion and Future Work

- Everything documented in Dema website (private part)
  `http://dema.gforge.inria.fr/mcgdb/openmp.html`

- Fix order bug with visualization engine
- More extended tests on real OpenMP applications?



Conclusion and Future Work

- Everything documented in Dema website (private part)
  `http://dema.gforge.inria.fr/mcgdb/openmp.html`

- Fix order bug with visualization engine
- More extended tests on real OpenMP applications?

- Continue with OpenMP 4.0 tasks?
- Start working on debugger-controlled profiling?