Programming-Model Centric Debugging for OpenMP

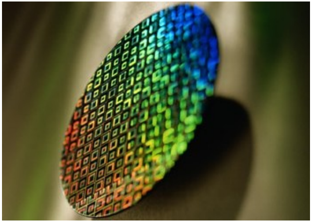
Kevin Pouget
Jean-François Méhaut, Miguel Santana

Université Grenoble Alpes / LIG, STMicroelectronics, France
Nano2017-DEMA project

DEMA Workshop, Grenoble
December 12th, 2016

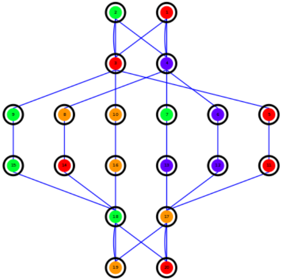
Today's parallel computing

- Multicore processors everywhere
  - HPC systems,
  - laptop and desktop computers,
  - embedded systems ...
- High-level programming environments
- Efficient verification & validation tools



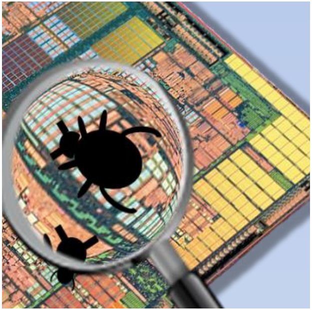
Today's parallel computing

- Multicore processors everywhere
  - HPC systems,
  - laptop and desktop computers,
  - embedded systems ...
- High-level programming environments
  - tasks with data-dependencies,
  - fork-join parallelism
  - ⟹ OpenMP
- Efficient verification & validation tools



Today's parallel computing

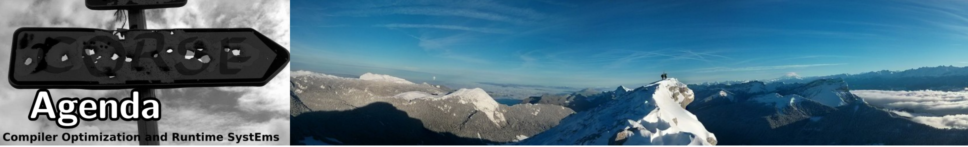
- Multicore processors everywhere
  - HPC systems,
  - laptop and desktop computers,
  - embedded systems ...
- High-level programming environments
  - tasks with data-dependencies,
  - fork-join parallelism
  - $\Longrightarrow$ OpenMP
- Efficient verification & validation tools
  - our research effort!



Agenda

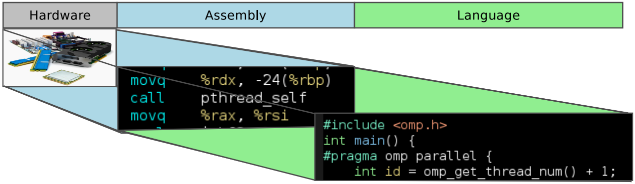
Compiler Optimization and Runtime SystEms

1. Research Context
2. Programming Model Centric Debugging
3. DEMA Year 1: Model-Centric Debugging for OpenMP
4. DEMA Year 2: Interactive Performance Profiling and Debugging



Agenda

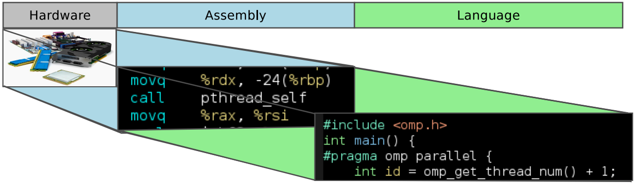
Verification and Validation: Debugging



Source-Level Interactive Debugging (e.g. GDB)

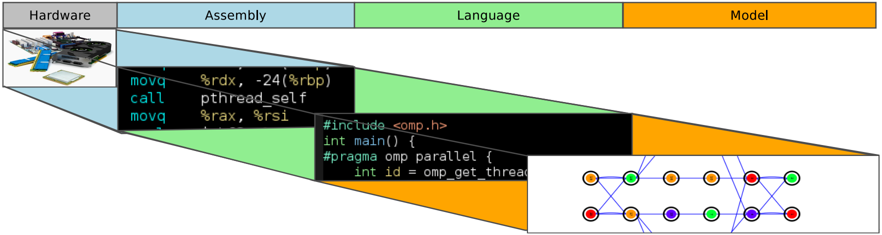
- Developers mental representation VS. actual execution
- Understand the different steps of the execution

Verification and Validation: Debugging



Source-level interactive debuggers operate at language-level.

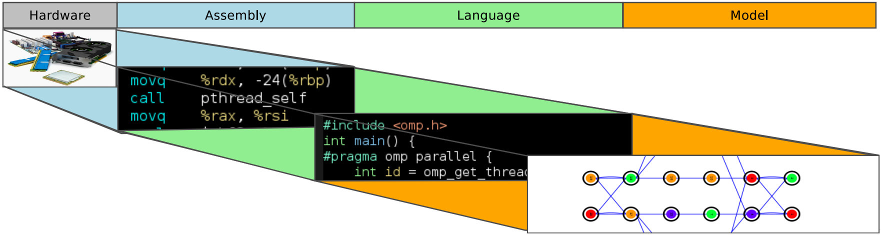
Verification and Validation: Debugging



Source-level interactive debuggers operate at language-level.

What about programming models?

Verification and Validation: Debugging



Source-level interactive debuggers operate at language-level.
What about programming models?

They have **no knowledge** about high-level **programming environments**!

Agenda

Compiler Optimization and Runtime SystEms

1 Research Context

2 Programming Model Centric Debugging

3 DEMA Year 1: Model-Centric Debugging for OpenMP

4 DEMA Year 2: Interactive Performance Profiling and Debugging

Programming Model Centric Debugging

Objective

Provide developers with means to
better understand the state of the high-level applications
and control more easily their execution,
suitable for various models and environments.

Programming Model Centric Debugging

Idea: **Integrate programming model concepts in interactive debugging**

1. Provide a Structural Representation
   - Draw application architecture diagrams
   - Represent the relationship between the entities
2. Monitor Dynamic Behaviors
   - Monitor the collaboration between the tasks
   - Detect communication, synchronization events
3. Interact with the Abstract Machine
   - Control the execution of the entities
   - Support interactions with *real* machine

1. Provide a Structural Representation
   - Draw application architecture diagrams
   - Represent the relationship between the entities
2. Monitor Dynamic Behaviors
   - Monitor the collaboration between the tasks
   - Detect communication, synchronization events
3. Interact with the Abstract Machine
   - Control the execution of the entities
   - Support interactions with *real* machine

Programming Model Centric Debugging

1. Provide a Structural Representation
   - Draw application architecture diagrams
   - Represent the relationship between the entities
2. Monitor Dynamic Behaviors
   - Monitor the collaboration between the tasks
   - Detect communication, synchronization events
3. Interact with the Abstract Machine
   - Control the execution of the entities
   - Support interactions with *real* machine

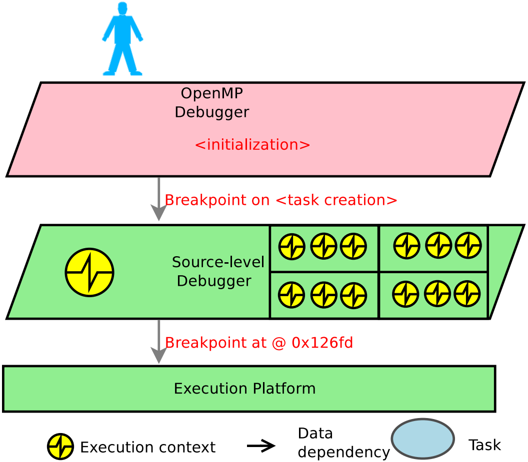
Programming Model Centric Debugging

⇒ Detect and interpret the execution events of the runtime framework

Programming Model Centric Debugging

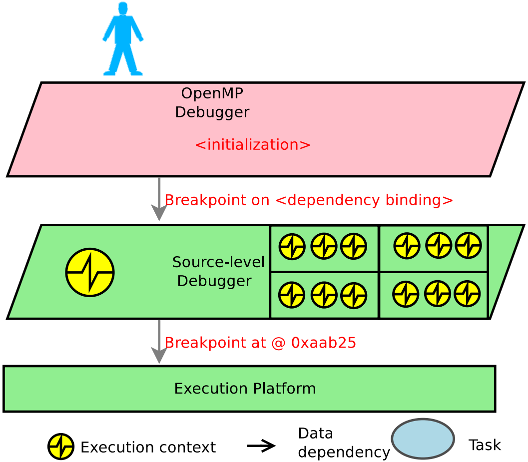
Compiler Optimization and Runtime SystEms

⇒ Detect and interpret the execution events of the runtime framework



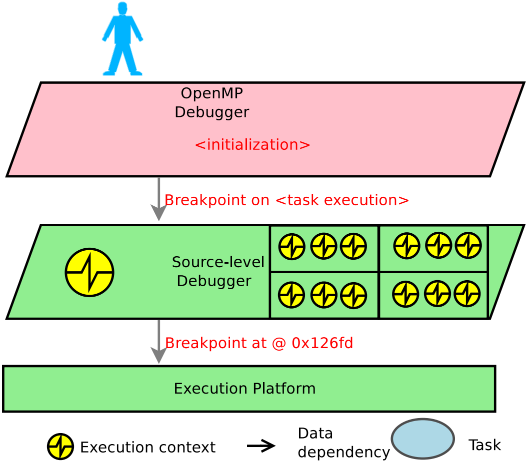
Programming Model Centric Debugging

⇒ Detect and interpret the execution events of the runtime framework

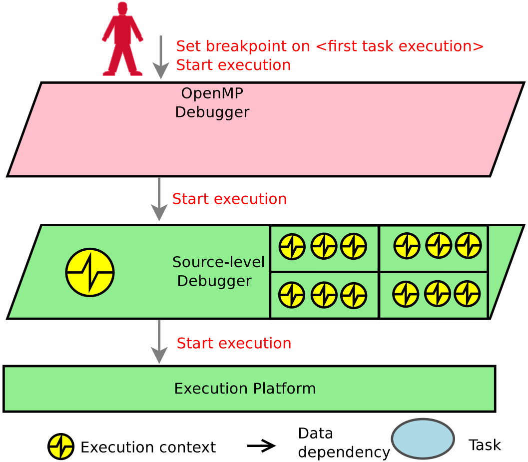


Programming Model Centric Debugging

⇒ Detect and interpret the execution events of the runtime framework

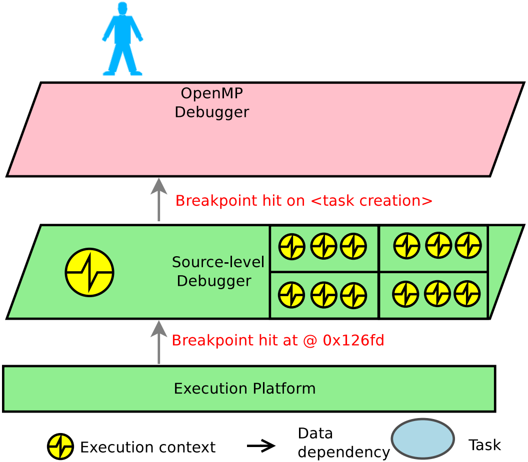


⇒ Detect and interpret the execution events of the runtime framework

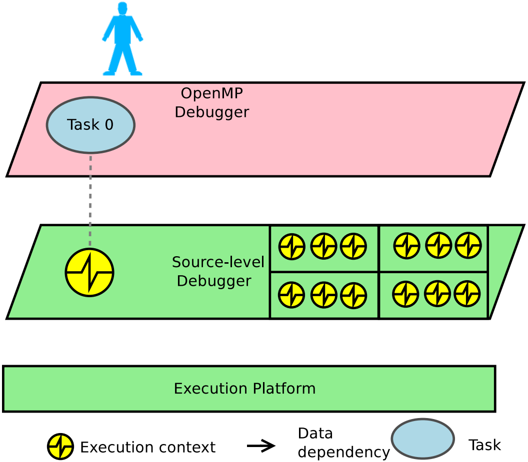


Programming Model Centric Debugging

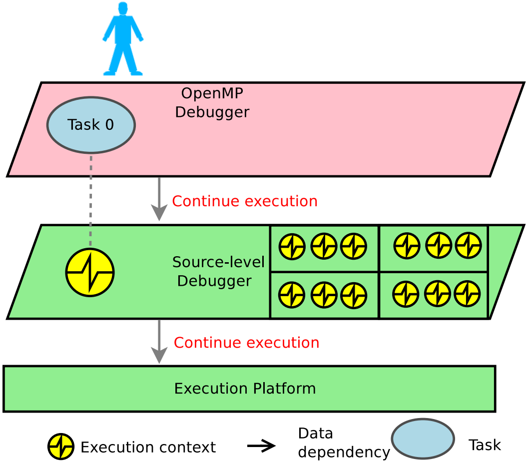
⇒ Detect and interpret the execution events of the runtime framework



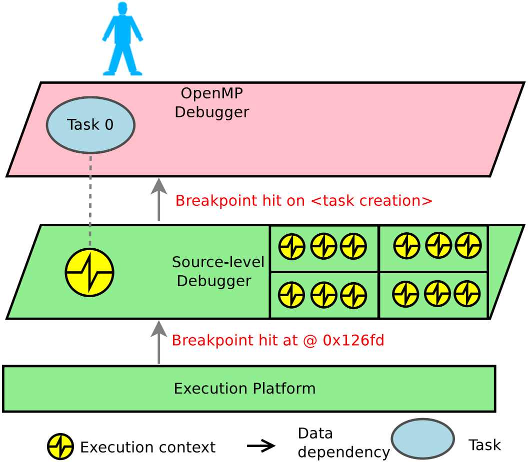
⇒ Detect and interpret the execution events of the runtime framework



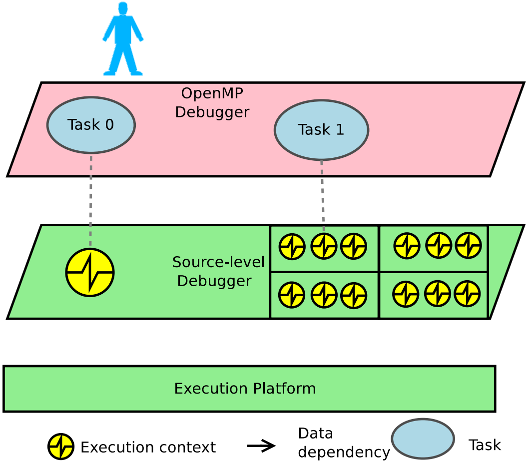
⇒ Detect and interpret the execution events of the runtime framework



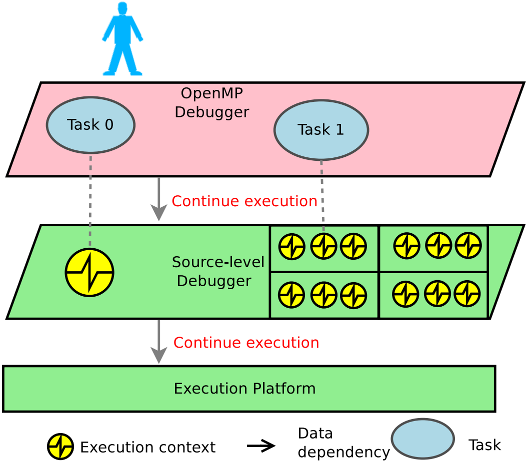
⇒ Detect and interpret the execution events of the runtime framework



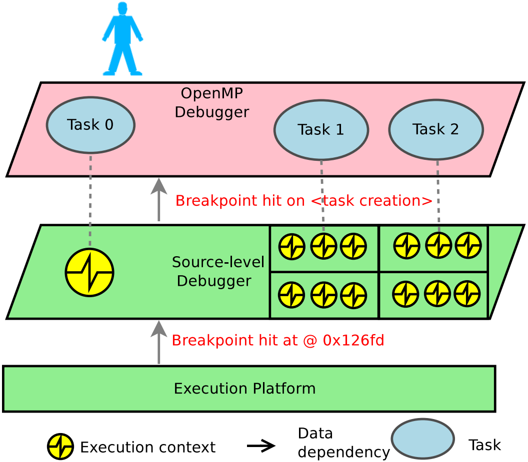
⇒ Detect and interpret the execution events of the runtime framework



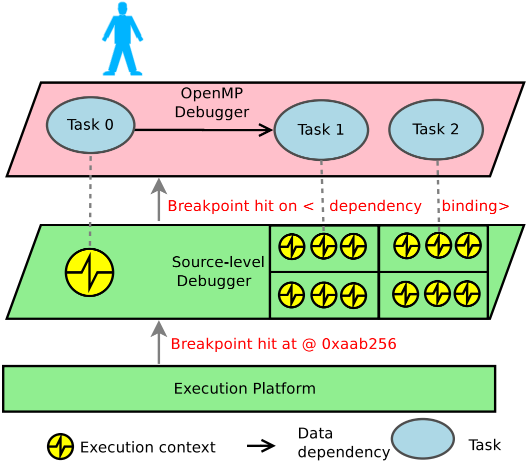
⇒ Detect and interpret the execution events of the runtime framework



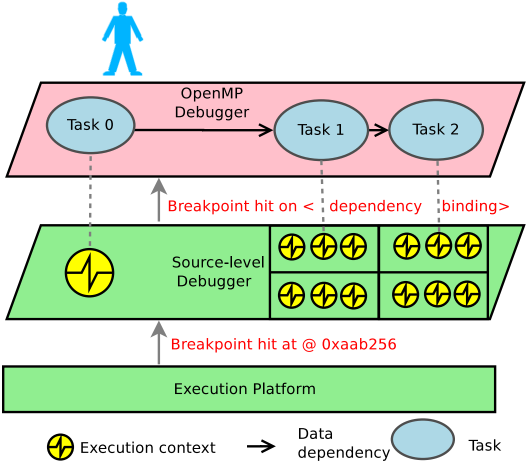
⇒ Detect and interpret the execution events of the runtime framework



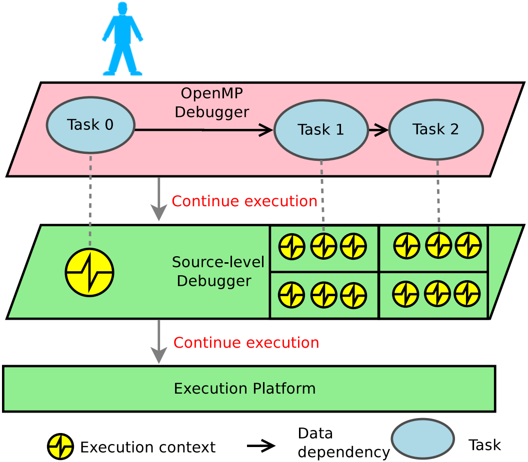
⇒ Detect and interpret the execution events of the runtime framework



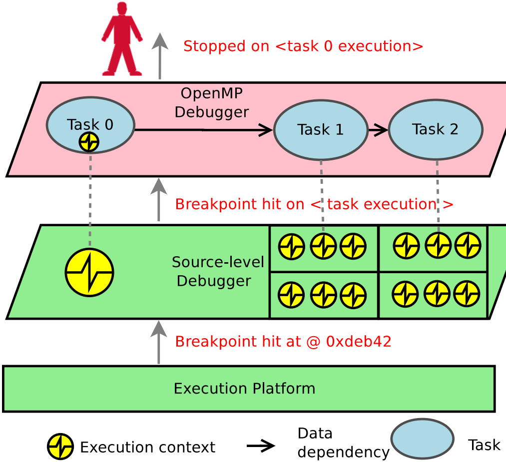
⇒ Detect and interpret the execution events of the runtime framework



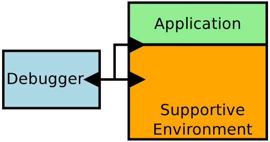
⇒ Detect and interpret the execution events of the runtime framework



⇒ Detect and interpret the execution events of the runtime framework

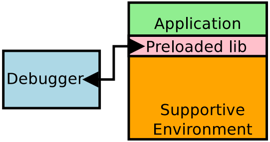


Information Capture Strategies



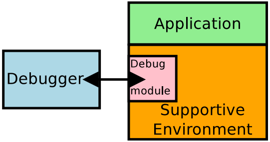
	Breakpoints and Debug Information		
Capturable Info.	**High**		
Execution Overhead	Significant		
Cooperation between Debugger and Env.	**None**		
Portability	Low		

Information Capture Strategies

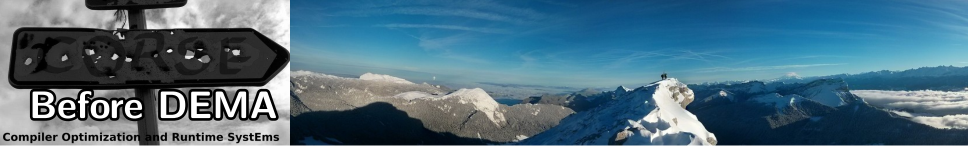


	Breakpoints and Debug Information	**Preloaded Library**
Capturable Info.	**High**	Limited to API
Execution Overhead	Significant	Limited
Cooperation between Debugger and Env.	**None**	Low
Portability	Low	**Very Good**

Information Capture Strategies



	Breakpoints and Debug Information	Preloaded Library	Specialized Debug Module
Capturable Info.	**High**	Limited to API	**Full**
Execution Overhead	Significant	Limited	Limited
Cooperation between Debugger and Env.	**None**	Low	Strong
Portability	Low	**Very Good**	Vendor Specific



Model-Centric Debugging Before DEMA

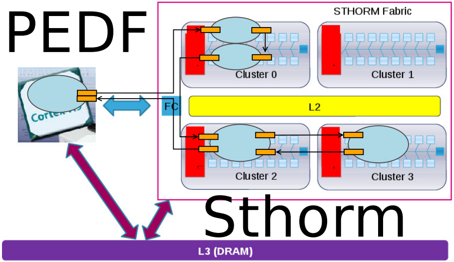
- components (STHORM NPM)
- dataflow (STHORM PEDF)
- kernel-based programming (GPU/STHORM OpenCL)

Model-Centric Debugging Before DEMA

- components (STHORM NPM)
- **dataflow (STHORM PEDF)**
- **kernel-based programming (GPU/STHORM OpenCL)**

Dataflow Debugging

for ST/CEA MPSoC STHROM



H.264 MPEG-4/AVC

logo by bullboykennels

Illustration 1: understanding a deadlock situation

Dataflow Debugging: Deadlock Detection

(gdb) info threads

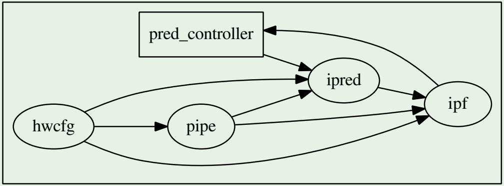
```
 Id   Target Id          Frame
  1   Thread 0xf7e77b  0xf7ffd430 in __kernel_vsyscall ()
* 2   Thread 0xf7e797  operator= (val=..., this=0xa0a1330)
```

Dataflow Debugging: Deadlock Detection

(gdb) info threads

```
 Id   Target Id         Frame
  1   Thread 0xf7e77b  0xf7ffd430 in __kernel_vsyscall ()
* 2   Thread 0xf7e797  operator= (val=..., this=0xa0a1330)
```

(mcgdb) info graph

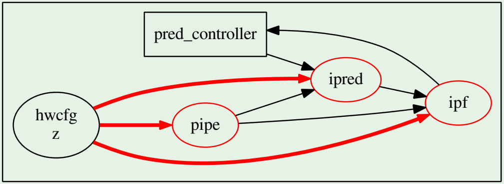


Dataflow Debugging: Deadlock Detection

(gdb) info threads

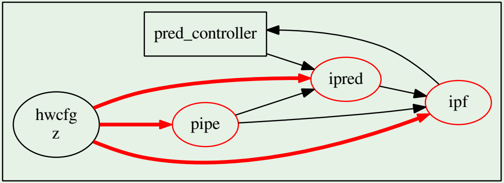
```
 Id   Target Id         Frame
  1   Thread 0xf7e77b  0xf7ffd430 in __kernel_vsyscall ()
* 2   Thread 0xf7e797  operator= (val=..., this=0xa0a1330)
```

(mcgdb) info graph +state



Dataflow Debugging: Deadlock Detection

(mcgdb) info graph +state

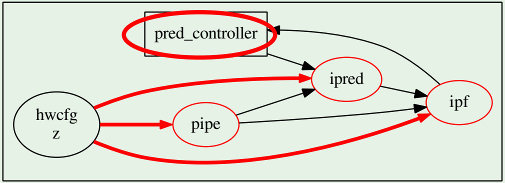


(mcgdb) info actors +state

```
#0 Controller 'pred_controller':
    Blocked, waiting for step completion
#1/2/3 Actors 'pipe/ipref/ipf':
    Blocked, reading from #4 'hwcfg'
#4 Actor 'hwcfg':
    Asleep, Step completed
```

Dataflow Debugging: Deadlock Detection

(mcgdb) info graph +state

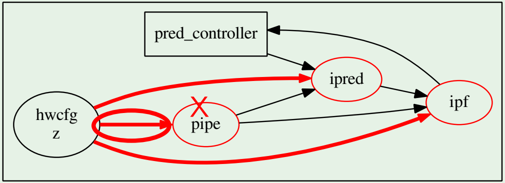


(gdb) thread apply all where

```
Thread 1 (Thread 0xf7e77b):
#0  0xf7ffd430 in __kernel_vsyscall ()
#1  0xf7fcd18c in pthread_cond_wait@ ()
#2  0x0809748f in wait_for_step_completion(struct... *)
#3  0x0809596e in pred_controller_work_function()
#4  0x08095cbc in entry(int, char**) ()
```

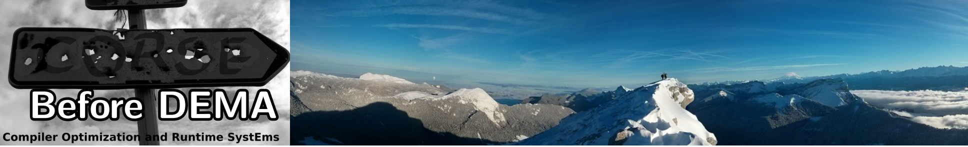
Dataflow Debugging: Deadlock Detection

(mcgdb) info graph +state



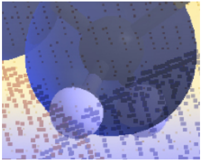
(gdb) thread apply all where

```
Thread 2 (Thread 0xf7e797):
#0 operator= (val=..., this=0xa0a1330)
#1 pipeRead (data=0) at pipeFilter.c:154 ↙
                 154   Smb = pedf.io.hwcfgSmb[count];
#2 0x0804da63 in PipeFilter_work_function () at pipe.c:361
#3 0x080a4132 in PedfBaseFilter::controller (this=0xa0d18)
```



OpenCL debugging

OpenCL



BigDFT

OpenCL (and Cuda)

- Running on STHORM,
  but primarily used with GPU
- Host-side debugging only

Density functional theory solver

- High performance computing
- Hybrid CPU/GPU
- MPI + OpenCL (C/Fortran)

Illustration 2: Why execution visualization is needed

Before DEMA: How execution visualization can help

Let's consider an example ...

C code

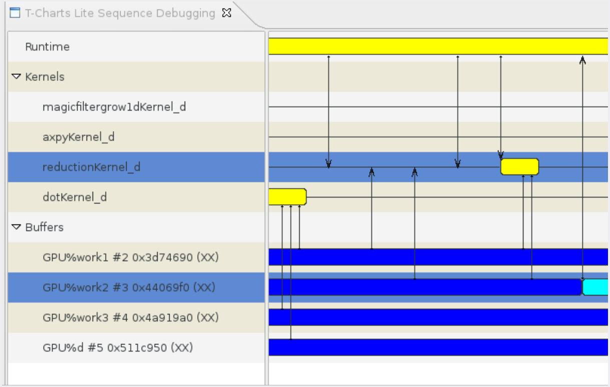
```
reductionKernel (int n, double *in, double *out){...}
checkStatus(int *ptr, char *msg) { if(ptr == 0) exit(-1);}

void main()  {
 double *in = malloc(...) ; checkStatus(in,"in failed");
 double *out = malloc(...); checkStatus(out,"out failed");

 initialize(in);
 reductionKernel(N, in, out);
 // free ...
}
```

Before DEMA: How execution visualization can help

OpenCL equivalent:

```
/* Instantiate the runtime.  */
command_queue = clCreateCommandQueue((*context)->context, aDevices[0], 0, &ciErrNum);
kerns->reduction_kernel_d=clCreateKernel(reductionProgram, "reductionKernel_d",&ciErrNum);
oclErrorCheck(ciErrNum,"Failed to create kernel!");
/* Allocate the buffers on the GPU.   */
*buff_ptr = clCreateBuffer((*context)->context,  CL_MEM_READ_ONLY, *size, NULL, &ciErrNum);
oclErrorCheck(ciErrNum,"Failed to create read buffer!");
/* Push the initial values to the GPU memory.  */
cl_int ciErrNum = clEnqueueWriteBuffer((*command_queue)->command_queue,  *buffer, CL_TRUE, 0, *size, p...
oclErrorCheck(ciErrNum,"Failed to enqueue write buffer!");
/* Set the kernel parameters.  */
clSetKernelArg(kernel, i++,sizeof(*ndat), (void*)ndat); clSetKernelArg(kernel, i++,sizeof(*in), (void*...
clSetKernelArg(kernel, i++,sizeof(*out), (void*)out);   clSetKernelArg(kernel, i++,sizeof(cl_dbl)*blk_...
/* Trigger the kernel execution.  */
ciErrNum = clEnqueueNDRangeKernel(command_queue->command_queue, kernel, 1, NULL, globalWorkSz, localWo...
oclErrorCheck(errNum,"Failed to enqueue reduction kernel!");
/* Get the result back.  */
cl_int ciErrNum = clEnqueueReadBuffer((*command_queue)->command_queue, *input, CL_TRUE, 0, sizeof(cl_d...
oclErrorCheck(ciErrNum,"Failed to enqueue read buffer!");
/* Then release the memory ... */
```

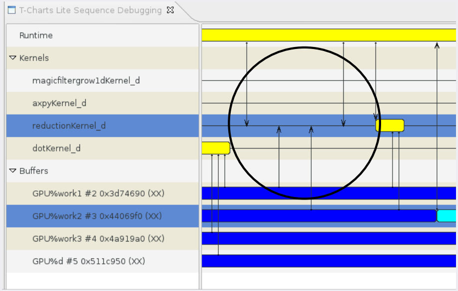
(mcgdb) print_flow (an Eclipse visualization engine)



Update on user request / automatically on exec. stops, step-by-step, ...

(mcgdb) print_flow

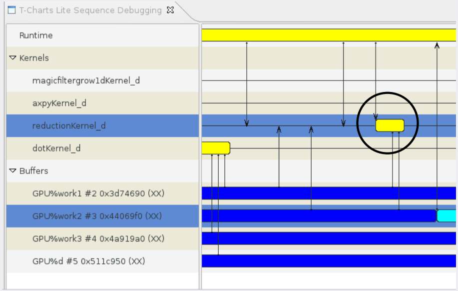
(an Eclipse visualization engine)



- Set the kernel arguments.
  - 2 scalars
  - 2 GPU buffers

```
clSetKernelArg(kernel, i++, sizeof(*ndat),(void*)ndat);
clSetKernelArg(kernel, i++, sizeof(*in), (void*)in);
clSetKernelArg(kernel, i++, sizeof(*out), (void*)out);
clSetKernelArg(kernel, i++, sizeof(*sz), (void*)sz);
```

(mcgdb) print_flow (an Eclipse visualization engine)

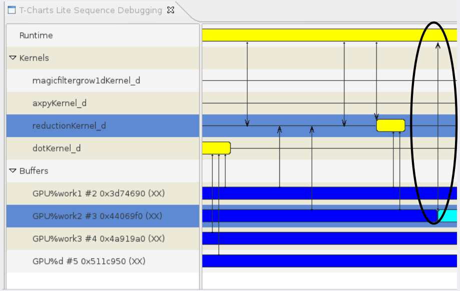


- Set the kernel arguments.
  - 2 scalars
  - 2 GPU buffers
- Trigger the kernel execution
  - 2 buffers involved

```
ciErrNum = clEnqueueNDRangeKernel(command_queue->command_q,
                                  kernel, 1, NULL, globalWorkSz,
                                  localWorkSize, 0, NULL, NULL);
```

Programming Model Centric Debugging: (before Dema) D

(mcgdb) print_flow (an Eclipse visualization engine)



- Set the kernel arguments.
  - 2 scalars
  - 2 GPU buffers
- Trigger the kernel execution
  - 2 buffers involved
- Retrieve the result
  - buffer content is saved

```
cl_int ciErrNum = clEnqueueReadBuffer(
                    (*command_queue)->command_queue,
                    *input, CL_TRUE, 0, sizeof(cl_double),
                    out, 0, NULL, NULL);
```

Agenda

Compiler Optimization and Runtime SystEms

1 Research Context

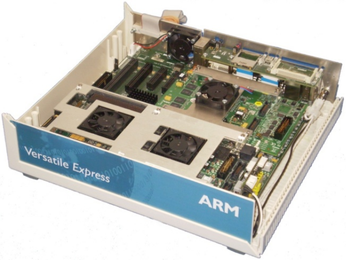
2 Programming Model Centric Debugging

3 DEMA Year 1: Model-Centric Debugging for OpenMP

4 DEMA Year 2: Interactive Performance Profiling and Debugging

Debugging Embedded and Multicore Applications

ARM Juno



- asymmetric arch.
- ARM big.LITLE + Mali GPU

OpenMP Parallel Programming

- Fork/join multithreading
- Tasks with dependencies
- GNU Gomp, Intel OpenMP, ...

mcGDB debugger

- Python extension of GDB
- Support for dataflow, components, ...
- Developed in partnership with ST

control the execution of the entities

1. start
2. omp start
3. omp step
4. omp next barrier
5. omp critical next
6. omp critical next
7. omp critical next
8. omp critical next

```
int main() {
  ①// beginning of main function
  #pragma omp parallel {
    // beginning of parallel region

    #pragma omp single {
      // execute single
    }//implicit barrier

    #pragma omp critical {
       // execute critical
    }
```

control the execution of the entities

1. start
2. omp start
3. omp step
4. omp next barrier
5. omp critical next
6. omp critical next
7. omp critical next
8. omp critical next

```
int main() {
  // beginning of main function
  #pragma omp parallel {
    ①②③④// beginning of parallel region

    #pragma omp single {
      // execute single
    }//implicit barrier

    #pragma omp critical {
       // execute critical
    }
```

control the execution of the entities

1. start
2. omp start
3. omp step
4. omp next barrier
5. omp critical next
6. omp critical next
7. omp critical next
8. omp critical next

```
int main() {
  // beginning of main function
  #pragma omp parallel {
    ②③④// beginning of parallel region

    #pragma omp single {
      ①// execute single
    }//implicit barrier

    #pragma omp critical {
        // execute critical
    }
```

OpenMP: OpenMP Execution Control

control the execution of the entities

1. start
2. omp start
3. omp step
4. omp next barrier
5. omp critical next
6. omp critical next
7. omp critical next
8. omp critical next

```
int main() {
  // beginning of main function
  #pragma omp parallel {
    // beginning of parallel region

    #pragma omp single {
      // execute single
    }①②③④//implicit barrier

    #pragma omp critical {
       // execute critical
    }
```

control the execution of the entities

1. start
2. omp start
3. omp step
4. omp next barrier
5. omp critical next
6. omp critical next
7. omp critical next
8. omp critical next

```
int main() {
  // beginning of main function
  #pragma omp parallel {
    // beginning of parallel region

    #pragma omp single {
      // execute single
    }//implicit barrier

    #pragma omp critical ①③④ {
      ② // execute critical
    }
```

control the execution of the entities

1. start
2. omp start
3. omp step
4. omp next barrier
5. omp critical next
6. omp critical next
7. omp critical next
8. omp critical next

```
int main() {
  // beginning of main function
  #pragma omp parallel {
    // beginning of parallel region

    #pragma omp single {
      // execute single
    }//implicit barrier

    #pragma omp critical ③④ {
      ①// execute critical
    }②
```

OpenMP: OpenMP Execution Control

control the execution of the entities

1. start
2. omp start
3. omp step
4. omp next barrier
5. omp critical next
6. omp critical next
7. omp critical next
8. omp critical next

```
int main() {
  // beginning of main function
  #pragma omp parallel {
    // beginning of parallel region

    #pragma omp single {
      // execute single
    }//implicit barrier

    #pragma omp critical ④ {
       ③// execute critical
    }①②
```

control the execution of the entities

1. start
2. omp start
3. omp step
4. omp next barrier
5. omp critical next
6. omp critical next
7. omp critical next
8. omp critical next

```
int main() {
  // beginning of main function
  #pragma omp parallel {
    // beginning of parallel region

    #pragma omp single {
      // execute single
    }//implicit barrier

    #pragma omp critical {
      ④// execute critical
    }①②③
```

OpenMP: structural representation

... provide a structural representation
... provide details about entity state

1. **fork-join** $\Longrightarrow$ OpenMP sequence diagrams
2. **task-based** $\Longrightarrow$ mcGDB+Temanejo cooperation

OpenMP: structural representation

... provide a structural representation
... provide details about entity state

1. **fork-join** $\Longrightarrow$ OpenMP sequence diagrams

2. **task-based** $\Longrightarrow$ mcGDB+Temanejo cooperation

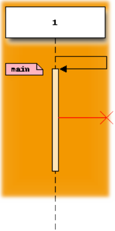
OpenMP: structural representation

... provide a structural representation
... provide details about entity state

1. **fork-join** $\Longrightarrow$ OpenMP sequence diagrams
2. **task-based** $\Longrightarrow$ mcGDB+Temanejo cooperation

OpenMP: OpenMP Sequence Diagram

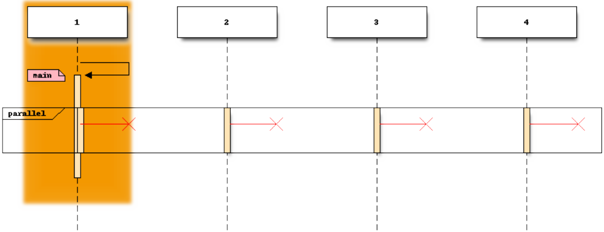
1. start
2. omp start
3. omp step
4. omp next barrier
5. thread 2
6. omp critical next
7. omp critical next
8. omp critical next



OpenMP: OpenMP Sequence Diagram

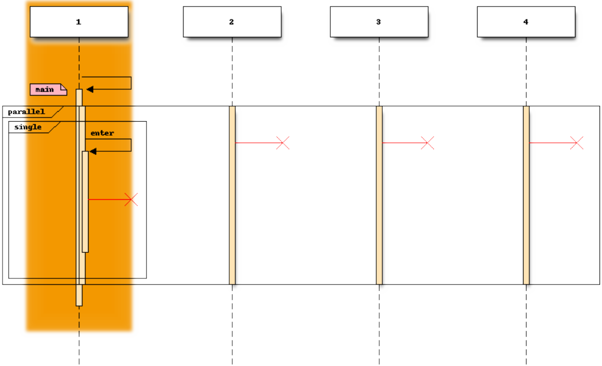
Compiler Optimization and Runtime SystEms

1. start
2. omp start
3. omp step
4. omp next barrier
5. thread 2
6. omp critical next
7. omp critical next
8. omp critical next



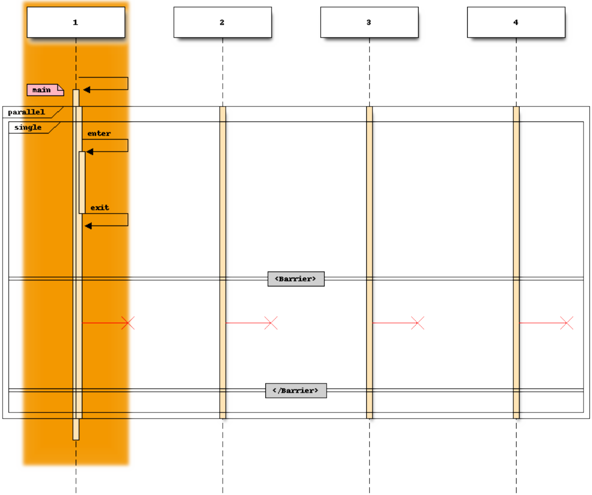
OpenMP: OpenMP Sequence Diagram

1. start
2. omp start
3. omp step
4. omp next barrier
5. thread 2
6. omp critical next
7. omp critical next
8. omp critical next



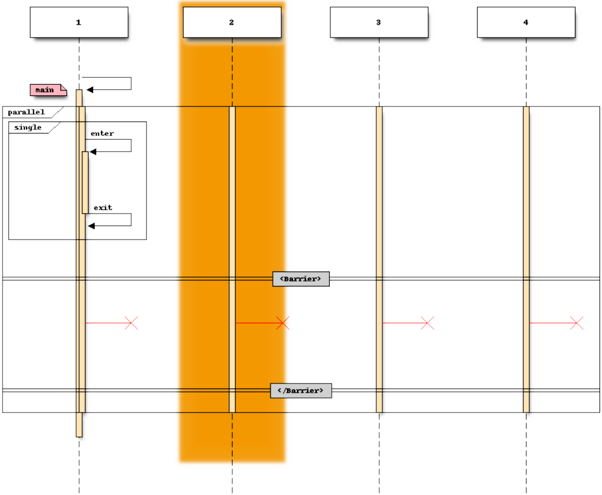
OpenMP: OpenMP Sequence Diagram

1. start
2. omp start
3. omp step
4. omp next barrier
5. thread 2
6. omp critical next
7. omp critical next
8. omp critical next

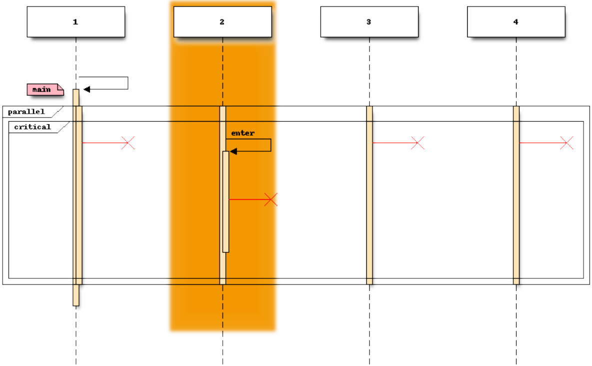


OpenMP: OpenMP Sequence Diagram

1. start
2. omp start
3. omp step
4. omp next barrier
5. thread 2
6. omp critical next
7. omp critical next
8. omp critical next

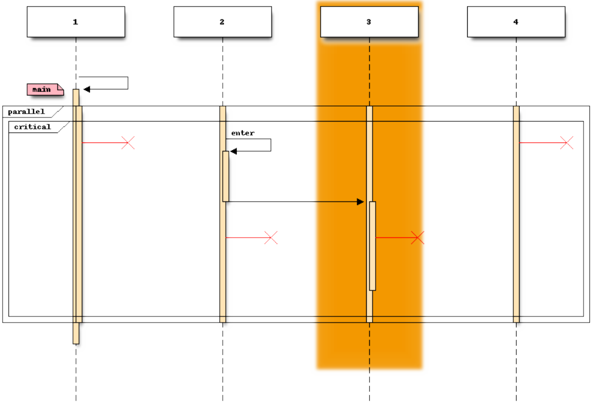


1. start
2. omp start
3. omp step
4. omp next barrier
5. thread 2
6. omp critical next
7. omp critical next
8. omp critical next



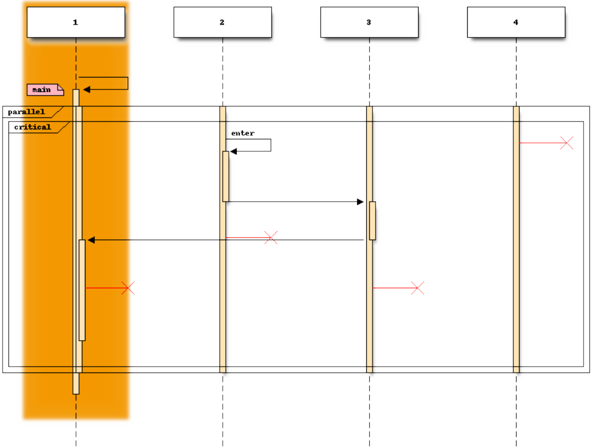
OpenMP: OpenMP Sequence Diagram

1. start
2. omp start
3. omp step
4. omp next barrier
5. thread 2
6. omp critical next
7. omp critical next
8. omp critical next



OpenMP: OpenMP Sequence Diagram

1. start
2. omp start
3. omp step
4. omp next barrier
5. thread 2
6. omp critical next
7. omp critical next
8. omp critical next



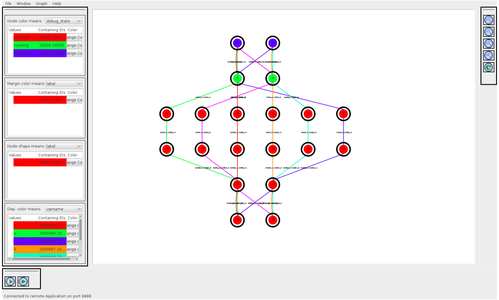
OpenMP: structural representation

... provide a structural representation
... provide details about entity state

1. **fork-join** $\Longrightarrow$ OpenMP sequence diagrams
2. **task-based** $\Longrightarrow$ mcGDB+Temanejo cooperation

(HLRS Stuttgart) Temanejo ...

✓ is a great visualization tool for task debugging,
✗ and does not support source-level debugging.



Task-Graph Visualization

Compiler Optimization and Runtime SystEms

(HLRS Stuttgart) Temanejo ...

- ✓ is a great visualization tool for task debugging,
- ✗ and does not support source-level debugging.

GDB/mcGDB ...

- ✗ has no visualization engine,
- ✓ but provides source debugging at language (gdb) and model level.

Task-Graph Visualization

(HLRS Stuttgart) Temanejo ...

- ✓ is a great visualization tool for task debugging,
- ✗ and does not support source-level debugging.

GDB/mcGDB ...

- ✗ has no visualization engine,
- ✓ but provides source debugging at language (gdb) and model level.

So let's combine them!

mcGDB – Temanejo cooperation:

Temanejo

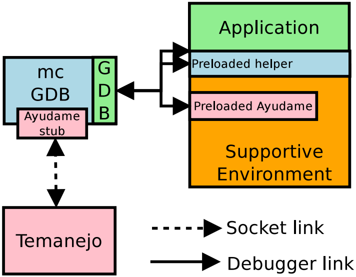
- task graph visualization
- simple model-level execution control.
- sequence diagram visualization.

mcGDB

- task graph and exec. events capture,
- advanced model-level exec. control.

GDB

- language and assembly level execution control, memory inspection.



mcGDB – Temanejo cooperation:

Temanejo

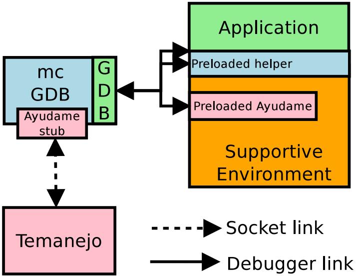
- task graph visualization
- simple model-level execution control.
- sequence diagram visualization.

mcGDB

- task graph and exec. events capture,
- advanced model-level exec. control.

GDB

- language and assembly level execution control, memory inspection.



mcGDB – Temanejo cooperation:

Temanejo

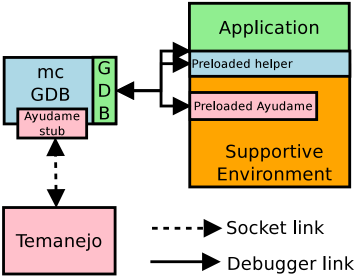
- task graph visualization
- simple model-level execution control.
- sequence diagram visualization.

mcGDB

- task graph and exec. events capture,
- advanced model-level exec. control.

GDB

- language and assembly level execution control, memory inspection.



mcGDB – Temanejo cooperation:

Temanejo

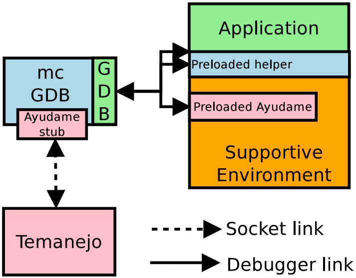
- task graph visualization
- simple model-level execution control.
- sequence diagram visualization.

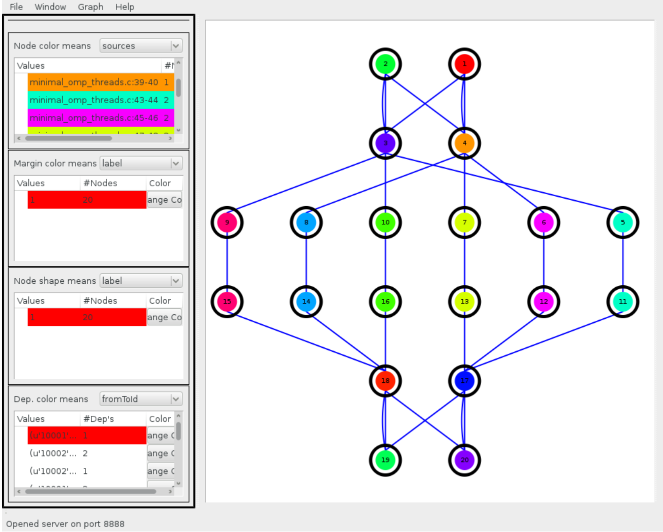
mcGDB

- task graph and exec. events capture,
- advanced model-level exec. control.

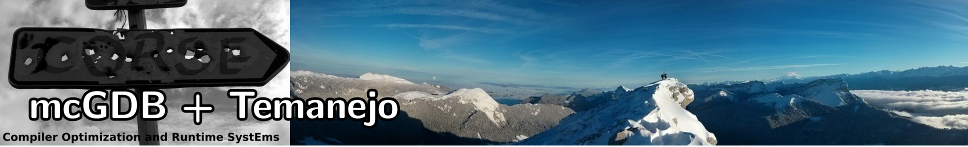
GDB

- language and assembly level execution control, memory inspection.

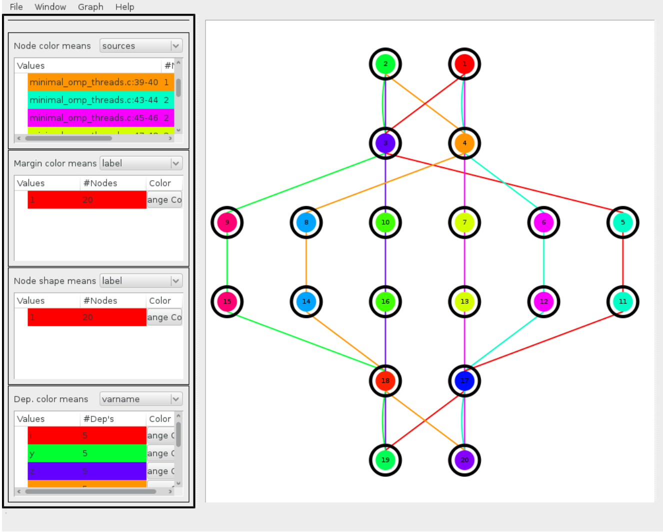




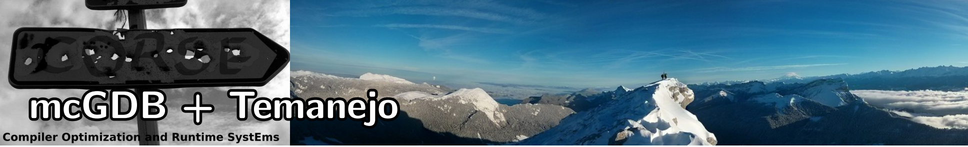
- Node color
  - sources files



mcGDB + Temanejo

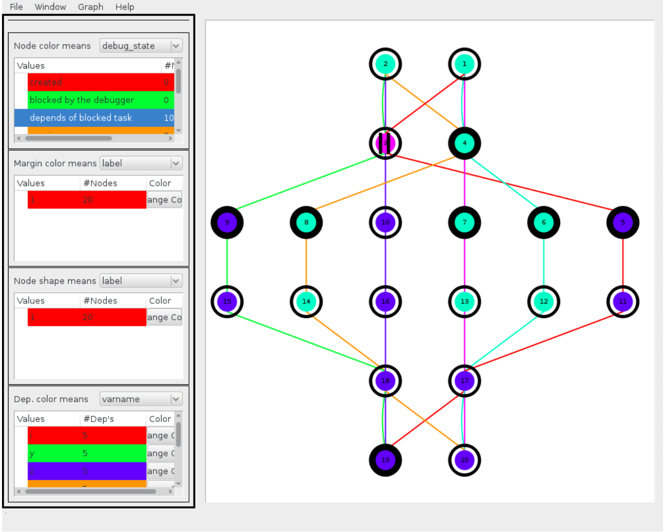


- Node color
  - sources files
- Links color
  - dependencies



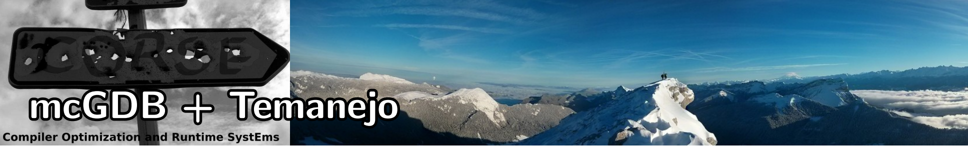
mcGDB + Temanejo

Compiler Optimization and Runtime SystEms



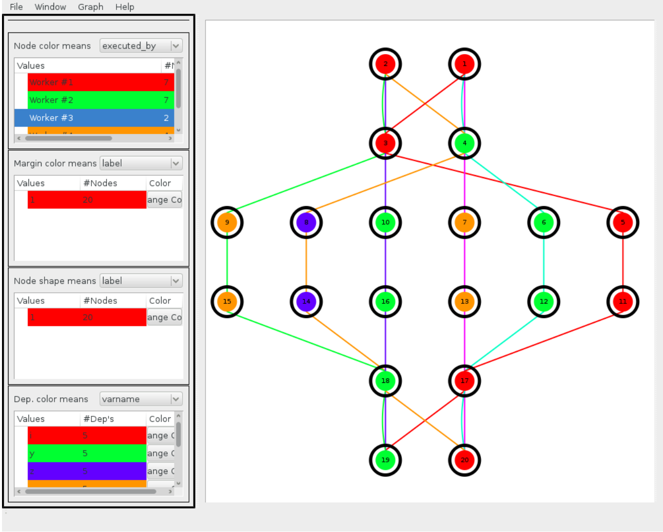
- Node color
  - ~~sources files~~
  - debug state
- Links color
  - dependencies
- Task 3 blocked

blue finished
purple blocked



mcGDB + Temanejo

Compiler Optimization and Runtime SystEms



- Node color
  - ~~sources files~~
  - ~~debug state~~
  - executed by
- Links color
  - dependencies
- Task 3 blocked

  blue finished

  purple blocked
- Exec. finished

Agenda

1. Research Context
2. Programming Model Centric Debugging
3. DEMA Year 1: Model-Centric Debugging for OpenMP
4. DEMA Year 2: Interactive Performance Profiling and Debugging

Performance Debugging Methodology

1. Benchmark the code
2. Locate the time-expensive areas
3. Estimate their (in)efficiency: how is the time spent? can it be reduced?
4. Optimize the code accordingly
5. Go back to step 1.

Performance Debugging Methodology

1. Benchmark the code
2. Locate the time-expensive areas
3. Estimate their (in)efficiency: how is the time spent? can it be reduced?
4. Optimize the code accordingly
5. Go back to step 1.

Profiling tools

- gprof
- perf stat,
- PAPI
- trace-based analyzers (aftermath)

Performance Debugging Methodology

1. Benchmark the code
2. Locate the time-expensive areas
3. Estimate their (in)efficiency: how is the time spent? can it be reduced?
4. Optimize the code accordingly
5. Go back to step 1.

Profiling tools : not really interactive

- gprof, perf stat, aftermath, . . .
  - profile all or nothing (`perf` can attach/detach)
- PAPI
  - customizable, but from within the code

Performance Debugging Methodology

1. Benchmark the code
2. Locate the time-expensive areas
3. Estimate their (in)efficiency: how is the time spent? can it be reduced?
4. Optimize the code accordingly
5. Go back to step 1.

Source-level debuggers (gdb/mcgdb) have interactivity!

- execute the code step-by-step,
- study the state,
- alter it to test hypotheses on-the-fly

. . . but nothing for performance debugging!

Interactive Performance Debugging

Performance Debugging Methodology

1. Benchmark the code
2. Locate the time-expensive areas
3. Estimate their (in)efficiency: how is the time spent? can it be reduced?
4. Optimize the code accordingly
5. Go back to step 1.

Source-level debuggers (gdb/mcgdb) have interactivity!

- execute the code step-by-step,
- study the state,
- alter it to test hypotheses on-the-fly

...but nothing for performance debugging!

Interactive Performance Debugging

This is an on-going work

1. Interactive profiling
   - What to measure?
   - Where to profile?
2. OpenMP profiling
3. MG benchmark performance bug and mcGDB
   - loop profiling
   - intermediate profiling charts
   - execution control and profiling
   - performance optimization and results

What to measure?

- `/proc/$PID/...` values (mem usage, context switches, ...)
- `gprof` counters
- function/address execution counter (breakpoints involved)
- `perf stat` counters

What to measure?

- /proc/$PID/... values (mem usage, context switches, ...)
- gprof counters
- function/address execution counter (breakpoints involved)
- perf stat counters
  - cache-misses, cache-references
  - instructions
  - cpu-clock, task-clock
  - node-load-misses, node-store-misses

Where to profile?

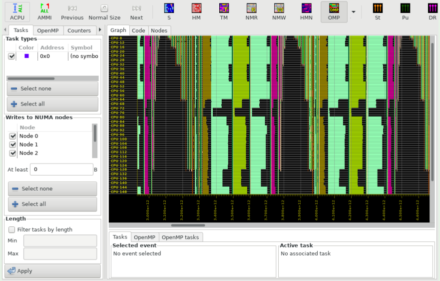
- During the execution:
  - a function execution
  - a region: from line ... to line ... (breakpoints involved)
  - start and stop on user request

- Outside of the normal execution (base on gdb+gcc dynamic compilation)
  - code compiled on-demand and inserted in the process address-space
  - custom function calls,
  - repeat *n* times
  - test different compilation flags, ...

Where to profile?

- During the execution:
  - a function execution
  - a region: from line ... to line ... (breakpoints involved)
  - start and stop on user request
  - what about OpenMP?
- Outside of the normal execution (base on gdb+gcc dynamic compilation)
  - code compiled on-demand and inserted in the process address-space
  - custom function calls,
  - repeat *n* times
  - test different compilation flags, ...

Profiling the whole execution: Aftermath[1] DEMA SP2



Fine-grain Interactive Profiling: mcGDB profiler

- use mcGDB for a fine-grained profiling of loops and tasks
- use mcGDB to trigger the generation of on-going Aftermath traces

[1] http://www.openstream.info/aftermath

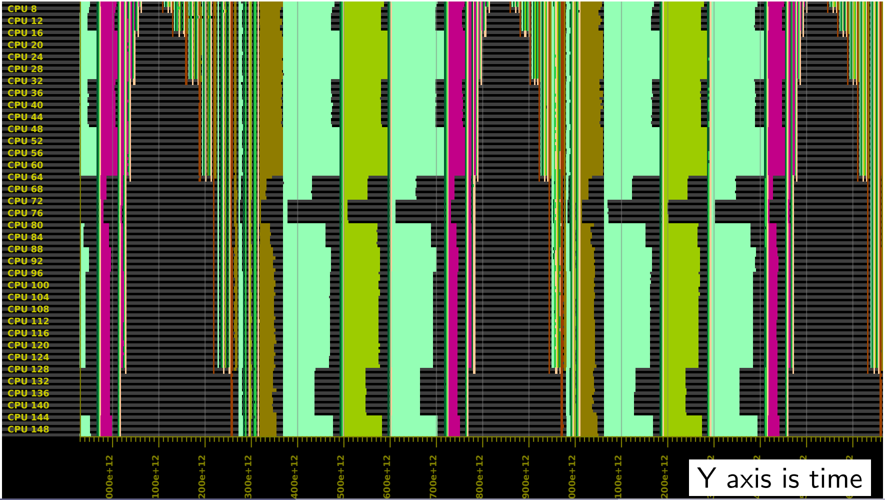
Before going further: mg.C performance bug

- performance bug on idchire (numa arch, 24 nodes, 192 cores)

```
#pragma omp for                      /* mc.c function resid */
  for (i3 = 1; i3 < n3-1; i3++) {
    for (i2 = 1; i2 < n2-1; i2++) {
      for (i1 = 0; i1 < n1; i1++) {
        u1[i1] = u[i3][i2-1][i1] + u[i3][i2+1][i1]
               + u[i3-1][i2][i1] + u[i3+1][i2][i1];
        u2[i1] = u[i3-1][i2-1][i1] + u[i3-1][i2+1][i1]
               + u[i3+1][i2-1][i1] + u[i3+1][i2+1][i1];
      }
      for (i1 = 1; i1 < n1-1; i1++) {
        r[i3][i2][i1] = v[i3][i2][i1] - a[0] * u[i3][i2][i1]
          - a[2] * (u2[i1] + u1[i1-1] + u1[i1+1])
          - a[3] * (u2[i1-1] + u2[i1+1]);
  } } }
```

Before going further: mg.C performance bug

- performance bug on idchire (numa arch, 24 nodes, 192 cores)



Y axis is time

Before going further: mg.C performance bug

- performance bug on idchire (numa arch, 24 nodes, 192 cores)

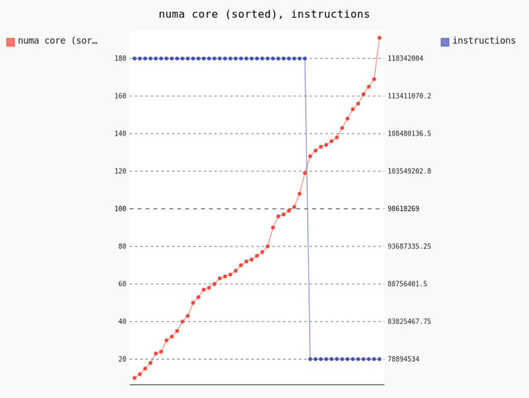
Use mcGDB knowledge for a fine-grained profiling of loops and tasks

- attach/detach `perf stat` when a loop iteration starts/stops
  - force sequentiality for accuracy / feasibility

```
| #23 loop profile
| cache-references:            20,322
| cycles:                  41,501,975
| node-stores:                  2,828
| node-misses:                  2,445
| instructions:            78,896,610
| omp_loop_len:                     1
| omp_loop_start:                 441
| numa node/code:              19/156
|
```

mg.C performance bug: intermediate chart view

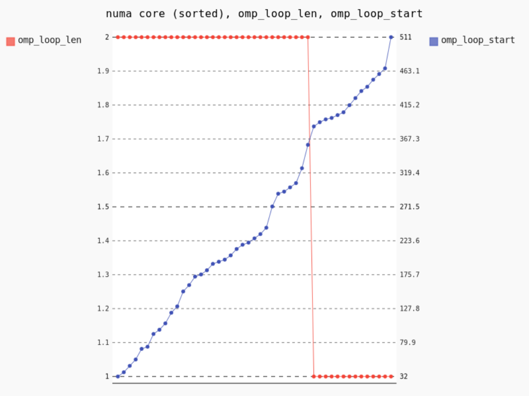
Instructions count sorted by *numa core id*; columns are loop iterations



Two phases (2 then 1 chunk), but the instruction count is constant.

mg.C performance bug: intermediate chart view

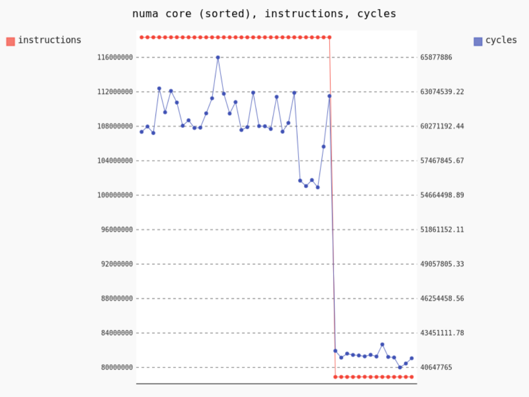
Loop length and *thread 1st loop index* sorted by *numa core id* (hidden)



(confirmation that the instruction count depends on the loop length)

mg.C performance bug: intermediate chart view

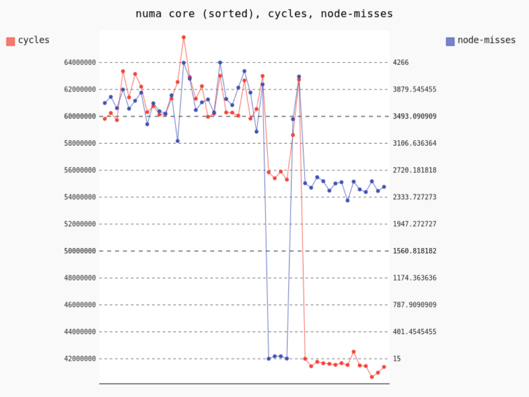
Instructions count and *cycles* sorted by *numa core id* (hidden)



With the same instruction count, some cores consume less cycles.

mg.C performance bug: intermediate chart view

Cycles and *node-misses* sorted by *numa core id* (hidden)



Low cycle count → low node misses ⟹ numa memory-location problem

OpenMP Profiling: mcGDB / Aftermath cooperation

Cooperation with Aftermath

- Correlation could have been highlighted with the help of Aftermath:
  - (gdb) aftermath trace dump
  - (gdb) aftermath visu reload
  - (gdb) aftermath trace insert_marker "stopped here"

⟹ preliminary code written this summer

OpenMP Profiling: execution control and inspection

Compiler Optimization and Runtime SystEms

Profiling breakpoint

```
(gdb) profile break if node-misses < 100
```

Loop control

```
(gdb) omp loop break before/after next
```

Numa-aware state inspection

```
(gdb) numa pagemap &r[$omp_loop_start()][0][0]
| Address 0x7fdbc9336380 is located on node N12
(gdb) numa current_node
| Thread #102 is bound to node N12, cpu 100.
```

`https://forge.imag.fr/projects/pagemap` by B. Videau et V. Danjean

OOpenMP Profiling: numa optimizations

```
(gdb) run # on breakpoint after memory alloc
19s + 54s # init and compute time
```

- normal run, launched from shell or GDB

OOpenMP Profiling: numa optimizations

```
(gdb) run # on breakpoint after memory alloc
19s + 54s # init and compute time
```

- normal run, launched from shell or GDB

```
(gdb) numa spread_heap # on breakpoint after memory alloc
20s + 13s
```

- spreads the whole heap (3GB) over the nodes, page by page
  - ⇒ confirmation of numa memory-location problem

OOpenMP Profiling: numa optimizations

Compiler Optimization and Runtime SystEms

```
(gdb) run # on breakpoint after memory alloc
19s + 54s # init and compute time
```

- normal run, launched from shell or GDB

```
(gdb) numa spread_heap # on breakpoint after memory alloc
20s + 13s
```

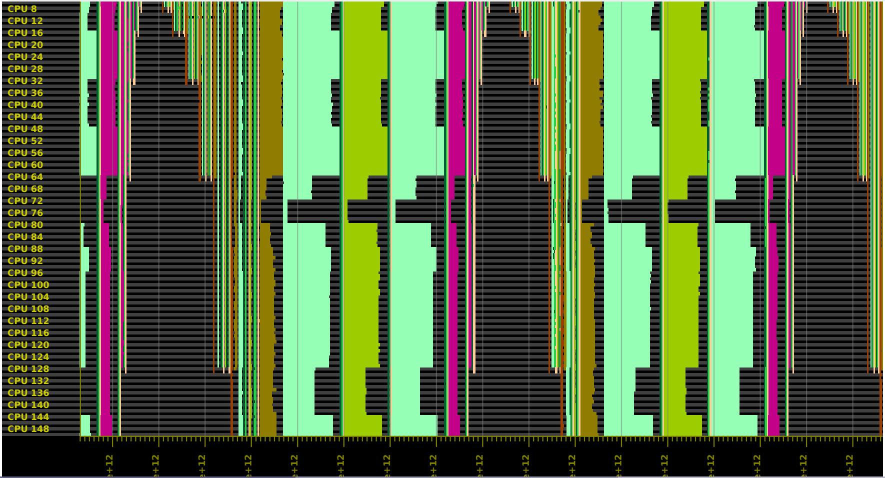
- spreads the whole heap (3GB) over the nodes, page by page
  - ⇒ confirmation of numa memory-location problem

```
(gdb) numa spread_3D_mat r[$i] m3[$i] m2[$i] m1[$i]
34s + 16s # i=9 and m3[$i]=m2[$i]=m1[$i]=514
```

- spread only `r[9]` and `u[9]` 3D matrices
- spread them according to OpenMP static loop distribution
  - ⇒ confirmation of numa memory-location problem

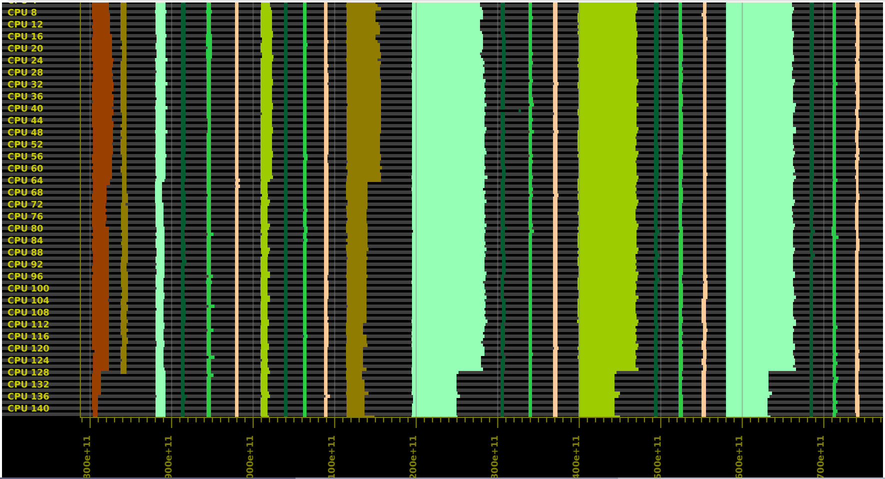
Before going further: mg.C performance bug

Back to Aftermath for comparison … 1/Native execution



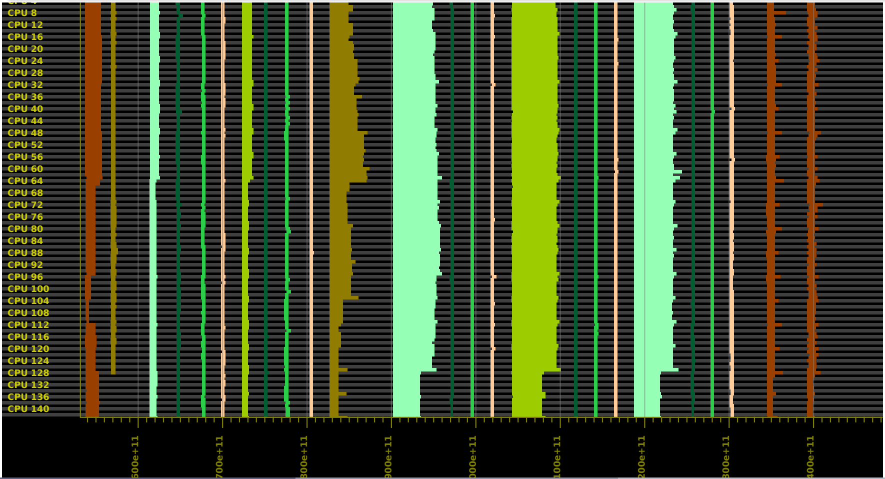
Before going further: mg.C performance bug

Back to Aftermath for comparison ... 2/Heap spread



Before going further: mg.C performance bug

Back to Aftermath for comparison ... 3/Matrix remapped



Conclusion

Compiler Optimization and Runtime SystEms

- Debuggers lack information about
  - programming models
  - runtime libraries
  - architectures
- They could benefit from additional knowledge
  - to provide a better code execution understanding

Our contribution: model-centric interactive debugging and profiling

- mcGDB extends GDB through its Python interface:
  - Extensible framework for model-centric debugging and performance testing and profiling
- mcGDB OpenMP support:
  - Developed for GNU GOMP and Intel OpenMP
  - Better control and understanding of fork-join / task-based execution
  - Fine-grained loop and task performance profiling

Conclusion

- Debuggers lack information about
  - programming models
  - runtime libraries
  - architectures
- They could benefit from additional knowledge
  - to provide a better code execution understanding

Our contribution: model-centric interactive debugging and profiling

- mcGDB extends GDB through its Python interface:
  - Extensible framework for model-centric debugging and performance testing and profiling
- mcGDB OpenMP support:
  - Developed for GNU GOMP and Intel OpenMP
  - Better control and understanding of fork-join / task-based execution
  - Fine-grained loop and task performance profiling

Conclusion

Compiler Optimization and Runtime SystEms

- Debuggers lack information about
  - programming models
  - runtime libraries
  - architectures
- They could benefit from additional knowledge
  - to provide a better code execution understanding

Our contribution: model-centric interactive debugging and profiling

- mcGDB extends GDB through its Python interface:
  - Extensible framework for model-centric debugging and performance testing and profiling
- mcGDB OpenMP support:
  - Developed for GNU GOMP and Intel OpenMP
  - Better control and understanding of fork-join / task-based execution
  - Fine-grained loop and task performance profiling

Conclusion

Compiler Optimization and Runtime SystEms

- Debuggers lack information about
  - programming models
  - runtime libraries
  - architectures
- They could benefit from additional knowledge
  - to provide a better code execution understanding

Our contribution: model-centric interactive debugging and profiling

- mcGDB extends GDB through its Python interface:
  - Extensible framework for model-centric debugging and performance testing and profiling
- mcGDB OpenMP support:
  - Developed for GNU GOMP and Intel OpenMP
  - Better control and understanding of fork-join / task-based execution
  - Fine-grained loop and task performance profiling

Programming-Model Centric Debugging for OpenMP

Kevin Pouget
Jean-François Méhaut, Miguel Santana

Université Grenoble Alpes / LIG, STMicroelectronics, France
Nano2017-DEMA project

DEMA Workshop, Grenoble
December 12^{th}, 2016