An Environment for Interactive Parallel Programming with MPI and OpenMP

Christian Terboven
<terboven@itc.rwth-aachen.de>
<terboven@pm.de>

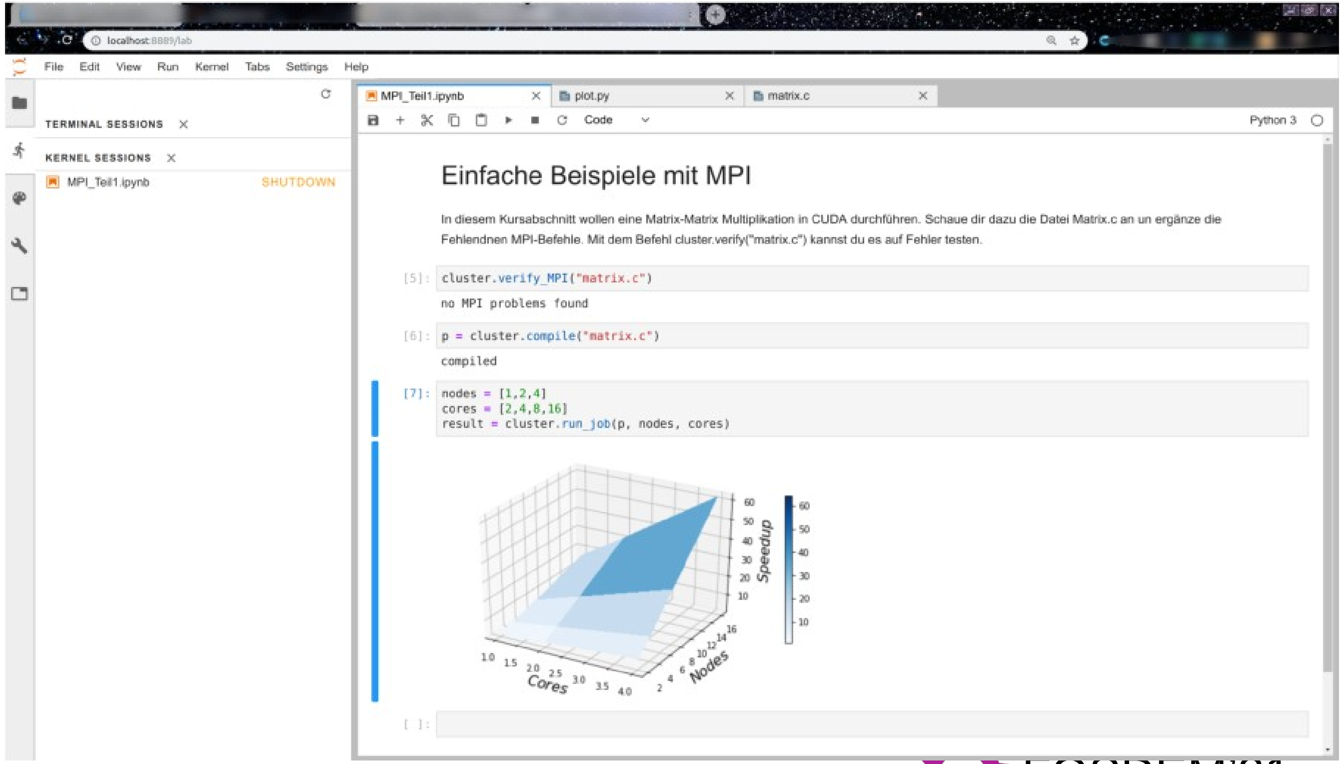
RWTH AACHEN UNIVERSITY

Jonas Hahnfeld
<hahnjo@hahnjo.de>

Motivation and Project Goals

- Increasingly more students and fields of study are interested in parallel programming and data analysis
  - Learning success and motivation increase when effects of parallelism are experienced directly
  - Challenge: technical barriers
    - Access to Linux systems via the command line, using a batch system, using many new dev tools, ...

versus



Interactive C++ with OpenMP and MPI

Interactive C++ and OpenMP / 1

- OpenMP: de-facto standard for shared memory parallelization with threads and tasks
- MPI: de-facto standard for distributed memory message-passing parallelization

```
In [1]: #include <iostream>

In [2]: std::cout << "Test" << std::endl;

        Test

In [3]: #include <omp.h>
        #pragma cling load("/usr/lib/libomp.so")

In [4]: int r = 0;

In [5]: r = 0;
        #pragma omp parallel reduction(+:r)
        {
            r += omp_get_num_threads();
        }
        r

Out[5]: 16
```

- Interactive C++
- Enabling OpenMP
- OpenMP example: 4 threads
  - Reduction on variable `r`

Interactive C++ and OpenMP / 2

- Interactive demonstration of and experiment with key concepts of parallel programming

In [19]:
```
#pragma omp parallel
#pragma omp single
{
    std::cout << "Hello from single" << std::endl;

    int a;
    #pragma omp task depend(out: a)
    {
        std::cout << "Hello from task 1" << std::endl;
    }
    #pragma omp task depend(in: a)
    {
        std::cout << "Hello from task 2" << std::endl;
    }
    #pragma omp taskwait
    std::cout << "Hello after taskwait" << std::endl;
}
```

```
Hello from single
Hello from task 1
Hello from task 2
Hello after taskwait
```

- OpenMP example: tasking
  - Two tasks
  - Ordered via `depend` clauses
  - Synchronized via `taskwait`

Interactive MPI

```
In [1]: #include <stdio.h>
        #include <mpi.h>
```

- Enabling MPI

```
In [2]: void exchange_info() {
            int rank, size;
            MPI_Comm_rank(MPI_COMM_WORLD, &rank);

            int other = 1 - rank;
            int data = rank;
            MPI_Send(&data, 1, MPI_INT, other, 0, MPI_COMM_WORLD);
            MPI_Recv(&data, 1, MPI_INT, other, 0, MPI_COMM_WORLD, M
        PI_STATUS_IGNORE);

            printf("rank %d received %d from rank %d\n", rank, dat
        a, other);
        }
```

```
In [3]: %%executable mpi.x -- -lmpi
        MPI_Init(NULL, NULL);

        exchange_info();

        MPI_Finalize();

        Writing executable to mpi.x
```

- Building an MPI program

```
In [4]: !mpiexec -np 2 ./mpi.x

        rank 0 received 1 from rank 1
        rank 1 received 0 from rank 0
```

- Executing an MPI program

Interactive Correctness Checking

- MUST / ThreadSanitizer: Correctness Checking of MPI / multi-threaded parallel programs
- Did you notice the problem in the code on the previous slide? It could potentially deadlock!

In [5]:
```
!mustrun -np 2 ./mpi.x
```

```
[MUST] MUST configuration ... centralized checks with fall-
back application crash handling (very slow)
[MUST] Information: overwritting old intermediate data in d
irectory "/rwthfs/rz/cluster/home/jh366276/IkapP/must_tem
p"!
[MUST] Generating P^nMPI configuration ... success
[MUST] Search for linked P^nMPI ... not found ... using LD_
PRELOAD to load P^nMPI ... success
[MUST] Executing application:
rank 0 received 1 from rank 1
rank 1 received 0 from rank 0
============MUST===============
ERROR: MUST detected a deadlock, detailed information is av
ailable in the MUST output file. You should either investig
ate details with a debugger or abort, the operation of MUST
will stop from now.
===============================
[MUST] Execution finished, inspect "/rwthfs/rz/cluster/home
/jh366276/IkapP/MUST_Output.html"!
```

After execution, see results in MUST_Output.html (./MUST_Output.html)

- Executing an MPI program with MUST
- Explanation of possible error
- Detailed report w/ graphical explanation

Implementation

Cling & xeus-cling: C++ in Jupyter

- Cling: interactive C++ interpreter
  - Developed by CERN & Fermilab for the ROOT data analysis framework
  - Uses Clang to parse Abstract Syntax Tree (AST)
  - Invokes LLVM's just-in-time (JIT) compilation & execution

- xeus-cling: C++ kernel for Jupyter
  - Sends code from notebook cells to Cling
  - Cling parses code and hands to JIT
  - Result is transferred back to user

More information: https://root.cern/cling/ (logo: CC BY 4.0)
https://xeus-cling.readthedocs.io/

Enabling OpenMP

- Clang supports OpenMP via -fopenmp
  - Add to arguments in kernel.json
  - Install LLVM OpenMP runtime next to Cling / xeus-cling

- Load OpenMP runtime in the notebook:
  - #pragma cling load("libomp.so")
  - #include <omp.h>

```
In [4]: int r = 0;

In [5]: r = 0;
        #pragma omp parallel reduction(+:r)
        {
            r += omp_get_num_threads();
        }
        r

Out[5]: 16
```

More information: https://openmp.llvm.org/

Support for [f]printf in xeus-cling / 1

- Muscle memory for many HPC developers:
  - Use [f]printf instead of C++ streams (std::cout, std::cerr)
  - But output not redirected with xeus-cling (jupyter-xeus/xeus-cling#112)

```
In [1]: printf("Hello FOSDEM!");
```

- Redirection worked fine with std::cout and std::cerr
  - Reason: Can replace buffer of C++ streams

Support for [f]printf in xeus-cling / 2

- Proposed solution: jupyter-xeus/xeus-cling#315
  - Inject custom implementation of [f]printf
  - Pass formatted string to std::cout, std::cerr, redirected to user
  - Merged 76 minutes after putting up the PR (issue had been open for more than 2 years)

```
In [1]: printf("Hello FOSDEM!");

        Hello FOSDEM!
```

Support for MPI Programs – inside JIT?

- MPI for distributed parallelism
  - Requires starting multiple processes
  - Naive approach: fork()

- Difficult for multiple reasons:
  - Communication with MPI library
    - Process discovery
    - Multiple MPI_Init + MPI_Finalize
    - Keep processes alive across cells?
  - Integration with Cling & Jupyter
    - Interpret new code in all processes?
  - Limited to one machine, no “distributed”

Support for MPI Programs – via Executables!

- Our approach: teach xeus-cling to dump executables
  - Define current cell as main()
  - Query current AST from Cling
  - Use clang::CodeGenerator to generate LLVM IR
  - Use LLVM backend to produce an object file
  - Link object file to an executable
- Implementation: around 200 lines of code interfacing the APIs

In [2]:
```
%%executable mpi.x -- -lmpi
MPI_Init(NULL, NULL);
int rank;
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
printf("Hello from rank %d\n", rank);
MPI_Finalize();
```

```
Writing executable to mpi.x
```

Support for MPI Programs – Launching Executables

- Launch executable with mpiexec: business as usual!

In [2]:
```
%%executable mpi.x -- -lmpi
MPI_Init(NULL, NULL);
int rank;
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
printf("Hello from rank %d\n", rank);
MPI_Finalize();
```

```
Writing executable to mpi.x
```

In [3]:
```
!mpiexec -np 4 ./mpi.x
```

```
Hello from rank 1
Hello from rank 2
Hello from rank 3
Hello from rank 0
```

Interactive Correctness Analysis: ThreadSanitizer

- Instrumentation would be hard in JIT
  - Sanitizer output at process termination
  - Cannot instrument already generated code

- Feasible extension with support for dumping executables
  - Parse flags from the user
  - Enable sanitizer & debug information if requested

```
In [2]: %%executable tsan.x -- -fsanitize=thread
        int a = 0;
        auto t = std::thread([&]() { a++; });
        a++;
        t.join();

        Enabling instrumentation for ThreadSanitizer
        Writing executable to tsan.x

In [3]: !./tsan.x

        ==================
        WARNING: ThreadSanitizer: data race (pid=623)
          Write of size 4 at 0x7ffdc60d157c by main thread:
```

Interactive Correctness Analysis: ThreadSanitizer for OpenMP

- Extension to OpenMP:
  - “Archer” library part of the LLVM OpenMP runtime
  - Developed in collaboration between RWTH Aachen University and DoE labs
  - Makes OpenMP semantics available to ThreadSanitizer

  → Avoid false-positives

In [2]:
```
%%executable tsan-omp.x -- -fopenmp -fsanitize=thread
int r = 0;
#pragma omp parallel num_threads(4) reduction(+:r)
{
    r += omp_get_num_threads();
}
printf("r = %d\n", r);
```

```
Enabling instrumentation for ThreadSanitizer
Writing executable to tsan-omp.x
```

In [3]:
```
!TSAN_OPTIONS='ignore_noninstrumented_modules=1' ./tsan-omp.x
```

```
r = 16
```

Interactive Correctness Analysis: MUST for MPI Programs

- MUST: runtime correctness analysis for MPI
  - Developed by TU Dresden and RWTH Aachen University
  - Checks for common classes of errors

More information: https://itc.rwth-aachen.de/must/

- Usage: replace mpiexec/mpirun by mustrun

```
In [4]: !mustrun -np 4 ./mpi.x

[MUST] MUST configuration ... centralized checks with fall-back application crash handling (very slow)
[MUST] Information: overwritting old intermediate data in directory "/home/jovyan/must_temp"!
[MUST] Weaver ... success
[MUST] Code generation ... success
[MUST] Build file generation ... success
[MUST] Configuring intermediate build ... success
[MUST] Building intermediate sources ... success
[MUST] Installing intermediate modules ... success
[MUST] Generating P^nMPI configuration ... success
[MUST] Search for linked P^nMPI ... not found ... using LD_PRELOAD to load P^nMPI ... success
[MUST] Executing application:
Hello from rank 1
Hello from rank 2
Hello from rank 3
Hello from rank 0
[MUST] Execution finished, inspect "/home/jovyan/MUST_Output.html"!
```

Summary

Agenda

- jupyter-based environment for interactive C/C++ OpenMP and MPI parallel programming
  - Developed with funding from IkapP project
- In use for teaching
  - Kubernetes-based infrastructure at jupyter.rwth-aachen.de (only for members of RWTH)
  - Live illustrations in lectures on
    - High-Performance Computing
    - Parallel and Data-centric Programming
  - Corresponding exercises can be done classically or in the new environment
  - Environment was designed without a pandemic in mind, but proved to be an excellent tool
- All components are open source, all patches got accepted