PPCES 2020 online - Q&A

Is the number of threads limited to the maximal number of cores or can a thread be seen as work package that will be executed once a core is free?

- The number of threads is not limited to the number of cores. However, in scientific-technical programs, you should not expect a speedup from using more threads than (logical) cores. However, a thread is different from a work package, if I understand you correctly. Your concept of a work package might be what a task in OpenMP is, which will be introduced in the afternoon.
- On a well-suited scientific application each thread should compute (roughly) same volume of work - load balancing. In this case it would be hardly profitable to start more threads that execution units, which are not necessarily the logical cores OS report to see, but (depending on compute load AND hardware layout!), maybe physical cores. If your application is known to be not well load-balanced, you can try to workaround this by starting more threads that execution units - but consider the TASK construct which would deliver the result much cheaper. If threads do each own work and sleeping most of the time (not often in scientific applications), starting way more threads that execution units is common practice (count the threads of Firefox browser for example).

Will OpenMP in Fortran count threads starting from 0 as well?

- Yes, the thread ids are in the range 0 ... n-1 if there are n threads.
- This is independent of the language. And by the way, OpenMP supports mixing C/C++ and Fortran within a single program.

Is it possible to decide number of threads and chunk size based on the size of computation and cache size given by a certain system?

- Yes, you can specify the number of threads and the used chunk size, e.g., based on hardware properties which may be dynamically computed at runtime

Is the master construct also available in OpenMPI?

- No, OpenMPI is an MPI implementation and not related to OpenMP. We will cover MPI tomorrow.
- OpenMPI ≠ OpenMP: OpenMPI is an MPI implementation intended for parallelization on Distributed Memory Systems while OpenMP is a standard for parallelization on Shared Memory Systems

Can OpenMP also be used for parallelization in a network of PCs or is it just limited to a local machine?

- OpenMP is a programming model for shared memory and is used on "single" machines. But there are large machines (so called "SMP nodes", also available in CLAIX) with several processors that have shared memory and for which you can use OpenMP. For parallelization of a network of PCs (which have separated memories and communicate data via network), you use MPI which we will discuss tomorrow. Often, you also combine OpenMP (parallel usage of cores / processors in a local machine) with MPI (parallel usage of multiple machines / clusters), this is called "hybrid programming".

- There are (mostly commercial) solutions for running OpenMP (and shared memory in general) on clusters of computers (= distributed memory), ScaleMP to name one. Due to the fact that the different nodes are connected via some network, which is typically *significantly* slower than inter-node communication, the NUMA-ratio of such virtual shared memory systems is very high in comparison to "real" nodes. This lead to significant issues on speedup and scalability of many real world applications.

I noticed that the run time is sometimes twice as high when I change the number of threads and run it for the first time. Where does this effect come from?

- Might have different reason. First thing to keep in mind is that you work on a frontend, where up 100 users are logged in at the same time. Many doing OpenMP exercises at the same time ;-). All these users and user's processes compete for the same resources which can slow down your execution. For productive runs and performance analysis you should use a batch system to ensure that resources are explicitly available to your application

Can two different communicators have the same ranks?

- Yes (was answered in the live session in details).
- You can find more information on custom communicators in our 2019 event https://doc.itc.rwth-aachen.de/display/VE/PPCES+2019 (see slides about advanced MPI)

What's your advice regarding MPI_Send? Should I use MPI_Ssend for prototyping to spot deadlocks and later MPI_Bsend for increased performance?

- That is an interesting question. That sounds like a good plan. However it also depends on the size of the application and whether you want to always search and replace terms. In general I would always use the regular Send and Isend functions in a production environment as there are some heuristics within each implementation when it makes sense to use one or the other protocol. Further, it should be noted that there are also some correctness checking tools like MUST that are able to find deadlocks in applications.
- When you use MPI_Bsend, you explicitly need to make sure that the allocated buffer, passed to MPI_Buffer_attach is not exceeded. But using MPI_Ssend for development and than switch to MPI_Send for production makes sense
- > and whether you want to always search and replace terms. -- Instead of "search and replace" of MPI_?send versions, you can use an own MyMPI_Send function, which use some define (to be set to a default value at one place) in order to switch from MPI_Ssend and MPI_Isend and ...) just by one single-byte switch at compile time, or even make it a run-time option to be changed on the fly.
- Maybe you can build yourself a little function with a preprocessor if / else clause which is reacting to debug and release flag.

```
#ifdef DEBUG
#define MPI_Send MPI_Ssend
#endif
```

- Maybe `#ifdef NDEBUG` would be a better choice since assert is disabled via the same mechanism. Thus `NDEBUG` is automatically defined my CMake when using CMAKE_BUILD_TYPE Release

If I use MPI_Send how do I afterwards check if MPI_Ssend or MPI_Bsend was used by the MPI library?

- Why do you want to know which method is actually used? They both share the same semantics that you can reuse the send buffer as soon as the MPI_Send call returns. Thus, from a usage point of view it does not matter
- The MPI vendors usually provide many implementations for the MPI routines (especially true for many-2-many communication with many choices for communication graph), and choose one by heuristics. That is why you sometimes observe scalability/speed artefacts. You typically can force one implementation by some options (vendor specific). All this is highly complicated, definitely not basic optimization stuff... For example, see https://arxiv.org/pdf/1707.09965.pdf (just googled example of research matter on different choices for a implementation of a given MPI function). Another example (cf. run time parameters which can be tweaked) for MPICH: https://eprints.lancs.ac.uk/id/eprint/133821/4/1570538287.pdf

Sometimes it does matter with regards to deadlocks and performance. I guess I'll need a profiler to see which Send is used?

- If you depend on the type of sending, which is done when calling MPI_Send, then I would consider that a bug in your program. Do not depend on implementation dependent behavior!
- Regarding performance, you are right, you would need to use a special performance tool for that as explained live by Marc-André. The implementation dependent "Eager Limit" can also be consulted for this information

What Tools do you have for debugging a MPI Process? What IDEs do you suggest for the development of OpenMP/MPI Programs?

- There are multiple tools for debugging serial, OpenMP, MPI and hybrid applications. On our cluster we do have DDT which can be loaded by module load forge that I recommend using. But there are also different other tools like TotalView (also installed on our cluster). However, based on experience you might also need to do some printf debugging at a certain point in time :)
- IMO, the IDE is not specific to OpenMP / MPI as both are just another library which is used by your application. Thus to get code completion, the setup does not really differ from any other library - OpenMP pragmas are a little different but I am not aware of any IDE providing auto-completion / checking for that. As a good starting point, VSCode works fine. It is general-purpose, and with the recent development of Remote Development Extensions, one can easily develop code on the cluster without going through an X-Server / FastX
- Concerning IDEs: Everyone has his or her own priorities. Depends whether you need a lot of autocompleting or not. Some use only a Vim or other regular editor. I personally use a combination of VS Code and Vim to develop but compile and run on the cluster only.
- Debugging of a single MPI process is possible with regular (any) debugger. Debugging of N MPI ranks is possible by N serial debuggers but this is usually not what you like. There are some (commercial) debuggers able to understand MPI and connect to all (or subset of) MPI ranks, to be named: TotalView, ARM DDT (part of ARM Forge). Using that debugger you can switch from one rank to other very easily, and they offer some special features like "circle detection" in Totalview (= deadlock detection). However, for MPI applications at size (> 16 ranks or so...) you typically do not even want to start debugging as you would be overwhelmed and not necessarily see the original issue. It is better to use special correctness checking tools first like MUST.