

e-Seminar #29

OpenMP in the Exascale Era



Presenter:
Dr. Mark Bull
(EPCC, University of Edinburgh)

18 January 2023

**The e-Seminar will start
at 2pm CET / 1pm GMT**



Moderator:
Tim Weaving
(University College London)



This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 823712



<https://insilicoworld.slack.com/archives/C0151M02TA4>

The e-Seminar series is run
in collaboration with:



e-Seminar #29

OpenMP in the Exascale Era



Presenter:
Dr. Mark Bull
(EPCC, University of Edinburgh)

18 January 2023

Welcome!



Moderator:
Tim Weaving
(University College London)



This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 823712



<https://insilicoworld.slack.com/archives/C0151M02TA4>

The e-Seminar series is run
in collaboration with:



Overview

- What is OpenMP?
- Why is OpenMP relevant for exascale computing?
- Hybrid MPI and OpenMP programs
- OpenMP for GPUs
- OpenMP tasks for irregular parallelism
- Why use OpenMP (and not some other API)?



What is OpenMP?

- OpenMP is an API designed for programming shared memory parallel computers.
- OpenMP is a set of extensions to Fortran, C and C++
- The extensions consist mostly of compiler directives
 - also runtime library routines and environment variables
- Open standard administered by the OpenMP Architecture Review Board
 - 30+ members from industry, academia, government labs
 - <https://www.openmp.org/>
 - started in the mid-1990s

Example

```
#pragma omp parallel for
for (int i=0; i<n; i++){
    for (int j=0; j<m; j++){
        b[i][j] += a[i][j] * c[j];
    }
}
```

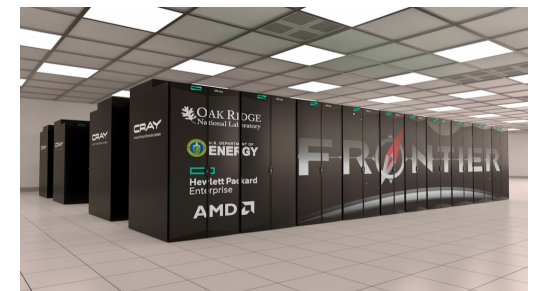
OpenMP is much more than just parallelizing loops, but this is the most commonly used feature!

OpenMP and Exascale

- Large scale HPC systems are not shared-memory!
 - each node has its own memory
- Usually program distributed memory systems using a message-passing library
 - almost always MPI!
- OpenMP can be used to
 - exploit multicore nodes more efficiently
 - program GPUs with a high-level vendor-neutral API

Exascale systems

- More powerful systems achieved by:
 - more cores per node
 - adding GPUs
 - not so much by adding more nodes
- UK national systems
 - ARCHER (2013-2021): 4920 nodes, 24 cores per node
 - ARCHER2 (2021-): 5860 nodes, 128 cores per node
- Frontier (first exascale system)
 - 9472 nodes, 64 cores + 4 GPUs per node



Hybrid MPI + OpenMP

- Hybrid MPI + OpenMP applications are becoming increasingly common on HPC systems
- Can both reduce memory usage and/or improve scalability
- Semantics are straightforward
 - especially if there are no MPI calls inside OpenMP parallel regions
- But...we need to choose how many OpenMP threads to run per MPI process
 - performance is a complex trade-off between multiple sources of overhead
 - difficult to identify any optimization opportunities

Potential advantages of MPI + OpenMP

- Reducing memory usage
 - fewer copies of replicated data structures
 - less data in halo regions
- Exploiting additional levels of parallelism
 - easier to do this by adding OpenMP than trying to do it in pure MPI
- Reducing computation
 - some MPI codes replicate parts of the computation
- Reducing load imbalance
 - easier and cheaper to balance load between threads than between processes
- Reducing communication costs
 - don't communicate unused data
 - fewer ranks in collectives
 - fewer (but maybe larger) point-to-point messages

Performance pitfalls

- Most hybrid applications are written (for simplicity) in master-only style – all MPI calls are outside of OpenMP parallel regions
 - OpenMP threads are necessarily idle during MPI communications
 - cache misses occur if master thread communicates data written/read by other threads
- Implicit point-to-point synchronisation via messages may be replaced by (more expensive) barriers.
 - loose thread-to-thread synchronisation is hard to do in OpenMP
- In a pure MPI code, the intra-node messages will often be naturally overlapped with inter-node messages
 - harder to overlap inter-thread communication with inter-node messages
- OpenMP can suffer from false sharing and NUMA effects
 - MPI naturally avoids these

Typical trends

If we keep the total number of cores fixed and increase the number of threads per MPI process:

- Total time spent in MPI reduces 😊
- Load imbalance between processes reduces 😊
- Amount of computation may reduce 😊
- Thread idle time increases 😞
- Thread synchronization time increases (mostly barriers) 😞
- Load imbalance between threads may increase 😞
- Use of memory system may get less efficient (false sharing, NUMA effects) 😞

Experimental setup

Hardware

- HPE Cray EX system
- 2 x AMD EPYC 7742, 2.25 GHz, 64-core chips per node (128 cores/node)
- 4 NUMA regions per socket (1 NUMA region per 16 cores)
- 16MB L3 cache per 4 cores
- HPE Slingshot network

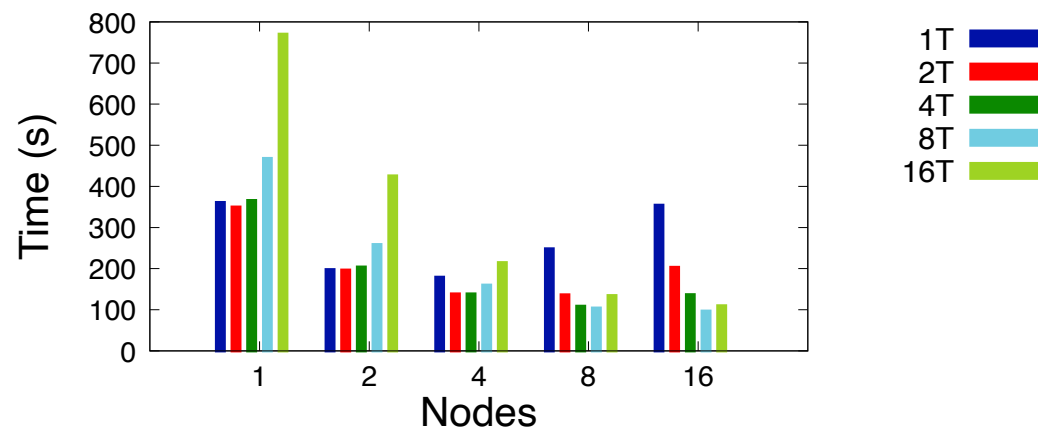
Software

- MPI: HPE Cray MPICH2
- OpenMP: GNU compilers
- Profiling tool: Scalasca

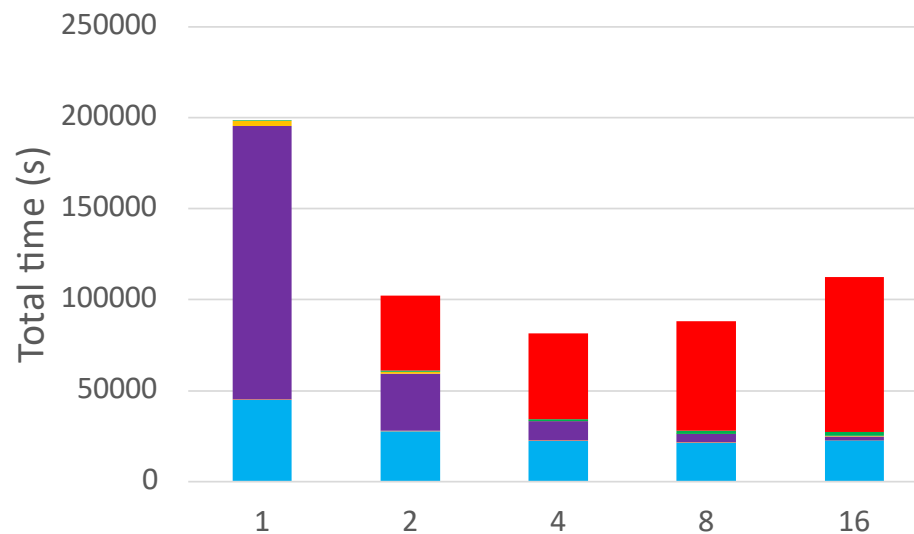


Case study- CASTEP

- CASTEP - density functional theory software package for electronic structure calculations using plane waves
 - Version 20.11 - GCC version 10.2, Intel MKL 19, Cray-mpich 8.1.4, Cray-fftw 3.3.8.11
 - Al3x3 benchmark



CASTEP – profiling



No. of OpenMP threads per MPI process

■ Computation ■ MPI Point-to-point
■ MPI Collectives ■ MPI Wait at barrier
■ OpenMP fork+barrier ■ Idle threads

Threads give big savings in compute and comms times (mostly in FFT library)

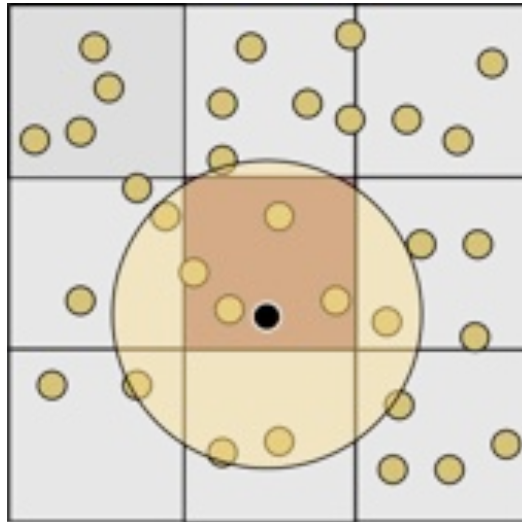
But....

FFT library comms are single threaded, so idle time outweighs savings above 8 threads per process

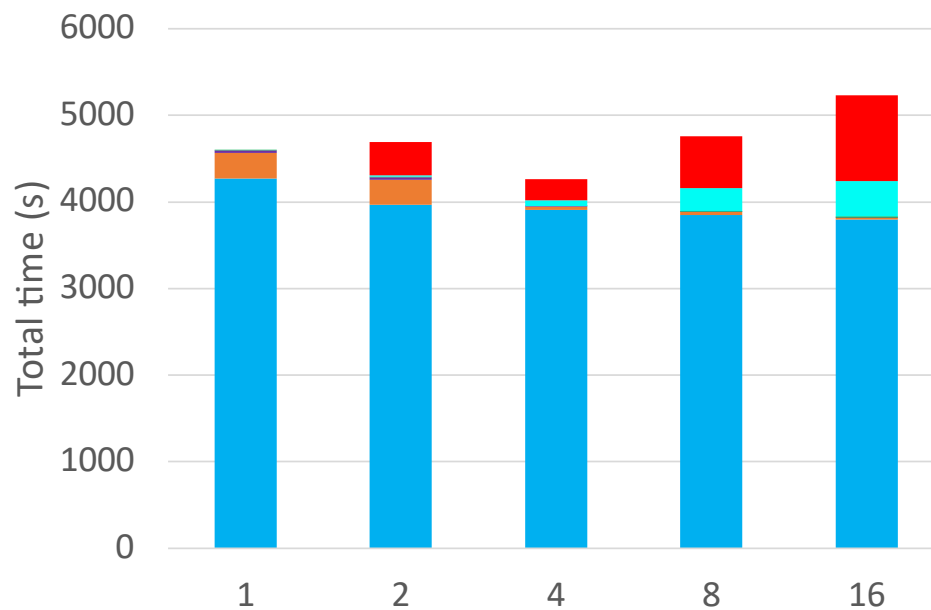
8 nodes (= 1024 cores)

Case study- CoMD

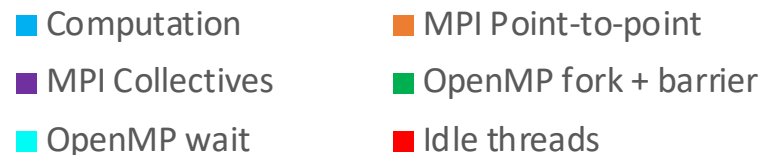
- CoMD Proxy Application – classical molecular dynamics using link cells
 - GCC version 8.3, Cray-mpich 8.1.4
 - 128^3 atoms



CoMD – profiling



No. of OpenMP threads per MPI process



8 nodes (= 1024 cores)

Threads give some savings in compute time

Load imbalance across processes is reduced

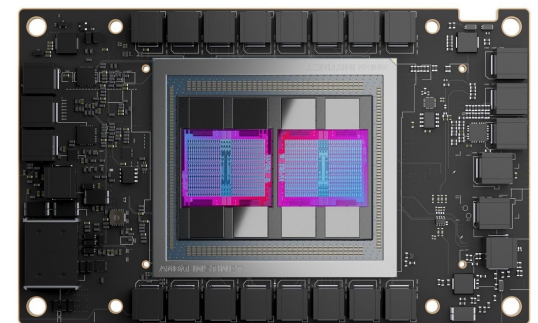
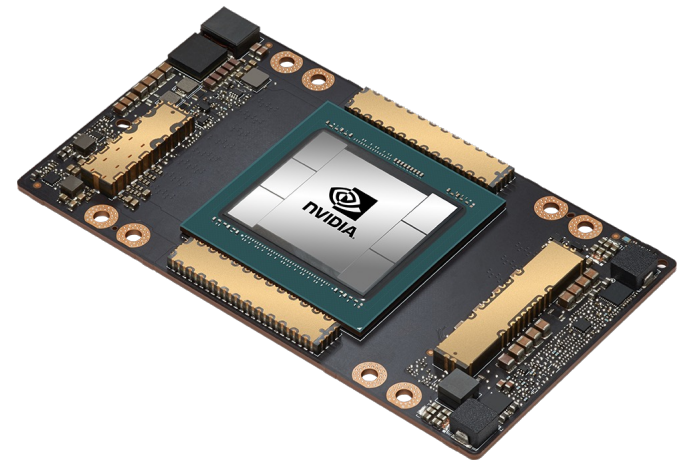
But..

Load imbalance across threads appears

Idle time becomes significant

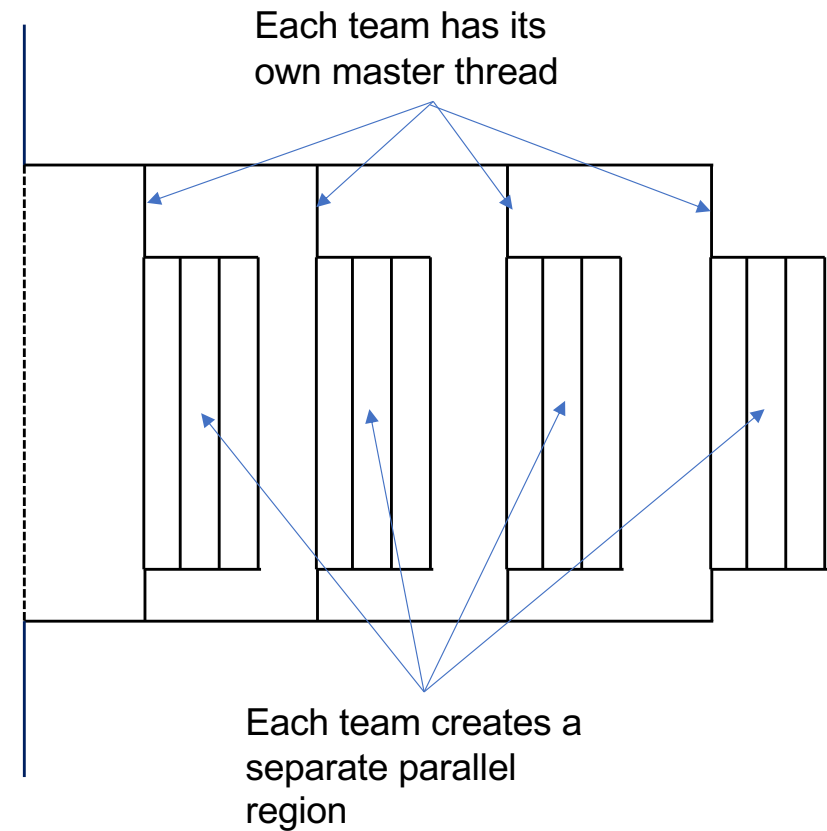
Accelerator support in OpenMP

- Not GPU specific
 - Not many other interesting devices at the moment, however!
- Fully integrated into OpenMP for the CPU
- Introduced in OpenMP 4.0, with significant revisions/extensions in 4.5 and 5.0, 5.1, 5.2.
- Similar to, but not the same as, OpenACC directives.
 - OpenACC is an alternative standard for offloading to GPUs
 - Developed before OpenMP 4.0
- Current, usable implementations of OpenMP for GPUs include: NVIDIA, Cray, IBM, LLVM/clang, gcc, Intel



OpenMP for GPUs

- Higher level interface than CUDA/HiP
 - Compiler/runtime does a lot of the work for you
 - Some loss of performance in some cases
- Includes constructs that map to hierarchical hardware on GPUs (SMs and threads)
- Vendor-neutral API
 - Good for portability and software sustainability
- Better Fortran support



Example

```
#pragma omp target teams loop map(to:a,c) \  
map(tofrom:b) collapse(2)  
for (int i=0; i<n; i++){  
    for (int j=0; j<m; j++){  
        b[i][j] += a[i][j] * c[j];  
    }  
}
```

- Map clauses specify movement of data between CPU and GPU memories.
- Collapse clause will parallelise both loops
- More control is possible if desired...

Performance issues

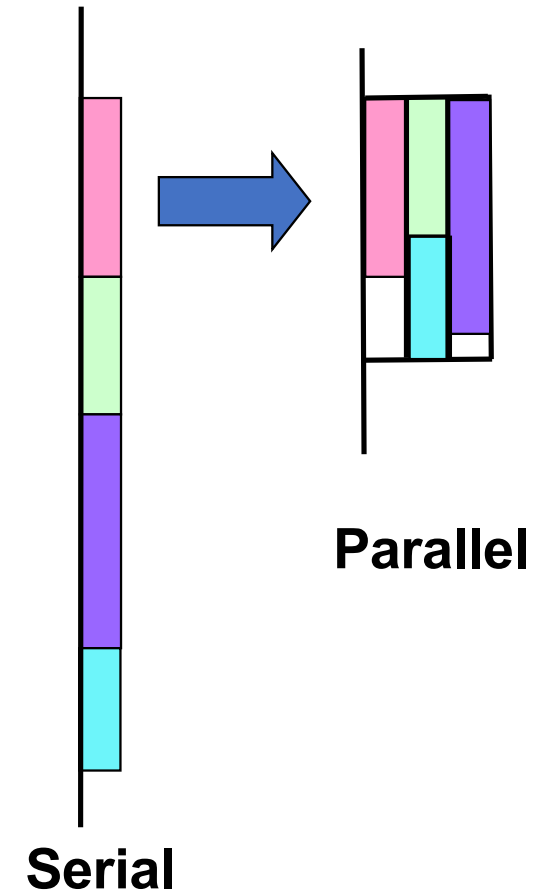
- Transferring data between host and device is expensive
 - May improve in future hardware designs
- Need to minimize this as much as possible
 - Don't transfer anything that's not required
 - Keep data on the device as far as possible (using **target data** regions)
- GPUs need lots of threads to work efficiently
 - Need to expose a lot of parallelism – much more than for the CPU
 - For nested loops can use the **collapse** clause to parallelise two or more loops in the nest

OpenMP tasks

- Most parallelism in scientific applications comes from parallel for/do loops
 - OpenMP handles these very well!
- But there are other cases
 - While loops
 - Recursion
 - Functional parallelism
 - Irregular, nested parallelism

What are OpenMP tasks?

- Tasks are independent units of work
- Tasks are composed of:
 - a block of code to execute
 - data to compute with (per-task private copies)
- Tasks are assigned to threads for execution.
 - programmer has little control over how this is done
- Decouples specification of parallelism from mapping to threads



Power of OpenMP tasks

OpenMP tasks are powerful because:

- They can be nested: a task may itself generate more tasks
 - More flexible than nested loops or nested teams of threads
- We can specify dependencies between tasks, if some tasks need to be executed before others
 - Enables a coarse-grain dataflow programming style
 - Very effective in some cases (e.g. dense linear algebra, PLASMA library)

Tasks for exascale?

- For ease of implementation, many applications have more synchronization than is really necessary
 - e.g. barriers at the end of parallel loops
 - also causes load imbalance
- MPI interface supports overlapping of computation and communication
 - most MPI implementations don't actually work very well: library doesn't have access to compute resources to make progress in the background

Tasks everywhere!

- If we could express all the parallelism as tasks with dataflow dependencies, *including the MPI calls*, we could potentially solve both problems!
- Unfortunately, letting MPI calls happen out-of-order causes problems
 - message mismatches, deadlocks
- Need a special MPI library such as TAMPI (developed at BSC) integrated with the tasking runtime.
- Basic idea: when an MPI call blocks, the task containing it is suspended, and only resumes when the message arrives.
 - meanwhile, the thread/core can run other tasks
- Not without problems: debugging and performance analysis is hard!
- For more details see **<https://zenodo.org/record/7524540>**

Why use OpenMP and not some other API?

- OpenMP is a mature (but still evolving standard)!
- OpenMP has a robust standards process and very wide support from vendors/compiler implementers
- OpenMP is not perfect: what are the other options?

Alternatives for CPUs

Posix threads (pthreads)

- Widely available, open standard
- Lower level than OpenMP, lacks support for parallel loops, reductions, tasks etc.
- No standard Fortran interface

C++ threads

- C++ only(!)
- Better integration with latest C++ features than OpenMP
- Also quite low-level
- Some issues with control over threads

TBB

- C++ only
- Intel specific
- Has some higher level features like OpenMP

MPI shared memory

- No interoperability issues
- Limited support for loop parallelism, load balancing tasks

Alternatives for GPUs

CUDA/HiP

- Robust, widely used
- Lower level than OpenMP
- Vendor specific (more-or-less)
- Fortran support is not that great

SYCL (DPC++)

- C++ only
- Open standard
- Not as mature as OpenMP
- Not many implementations

OpenCL

- *Very* low-level
- Open standard
- Performance is not great


OpenACC

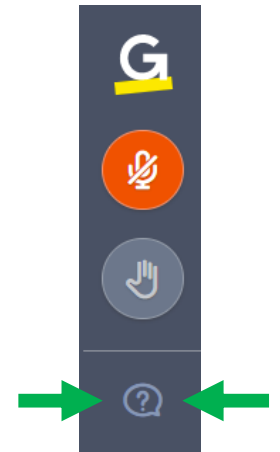
- Similar to OpenMP (directives)
- More mature than OpenMP
- Not many implementations

Summary

- OpenMP is a well-established standard for shared memory and accelerator offload programming
- The design of large-scale systems means it has a role to play for exascale computing
 - both for CPUs and GPUs
- OpenMP tasks are a potentially useful feature – work in progress!

Q&A

To pose a question, please click on the  symbol and send your question via the 'Ask the staff a question' panel



This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 823712



<https://insilicoworld.slack.com/archives/C0151M02TA4>

The e-Seminar series is run in collaboration with:



Thank you for participating!

...don't forget to fill in our feedback questionnaire...

Visit the CompBioMed website (www.compbiomed.eu/training)
for a full recording of this and other e-Seminars,
to download the slides
and to keep updated on our upcoming trainings



This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 823712



<https://insilicoworld.slack.com/archives/C0151M02TA4>

The e-Seminar series is run
in collaboration with:

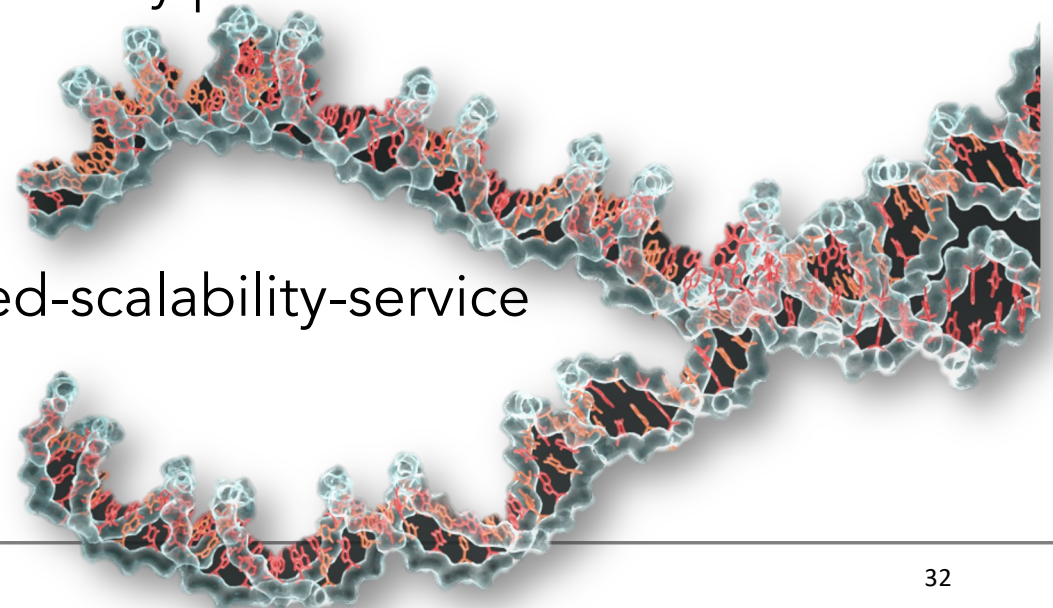


CompBioMed's *Free Scalability Service*



- Improves performance of your biomedicine applications on high performance computers
 - Experts in both biomedical applications and high performance computers
 - Make your biomedicine applications run in parallel
 - Improving the scalability of those already parallelised

- www.compbiomed.eu/compbiomed-scalability-service



- Contact for *Free* Service
 - General technical questions
 - Slack: #scalability channel of *the InSilicoWorld Community of Practice*
 - Email: compbiomed-support@ucl.ac.uk
 - Full service
 - Application Form or light-weight web form
 - Formal collaborative relationship with CompBioMed Centre of Excellence
- Application and Data Security
 - Great care when adapting your applications and managing your data
 - Our Data Policies cover Data Privacy, Data Security and Research Data Management

InSilicoWorld Community of Practice



The first community entirely on *in silico* medicine on Slack

www.insilico.world/community

Expertise

- The community is invitation only: in this way we ensure only interested experts have access

Collaboration

- Join teams and collaboratively work on shared goals, projects, concerns, problems or topics

Safe space

- A pre-competitive space where experts from academia, industry, and regulatory agencies can ask for and exchange advices

More than 500 experts have already joined the community and its channels

InSilicoWorld Members



- **Large Biomedical Companies**

Medtronic, Smith & Nephew, Pfizer, Johnson and Johnson, Innovative Medicine Initiative, CSL Behring, Ambu, RS-Scan, Corwave EN, Zimmer Biomet, Novartis, Bayer, ATOS, Biogen, Agfa, Icon PLC, Amgen, ERT, Exponent, etc.

- **Biomedical SMEs**

Nova Discovery, Lynkeus, Obsidian Biomedical, Quibim, Mediolanum Cardio Research, Voisin Consulting, CRM-Microport, Mimesis srl, H. M. Pharmacon, MCHCE, etc.

- **Independent Software Vendors**

Ansys, In Silico Trials Technologies, 3DS, KIT, ASD Advanced Simulation & Design GmbH, Kuano-AI, Aparito, Chemotargets, Digital Orthopaedics, ExactCure, Materialise, Bio-CFD, Matical, FEOPS, 4RealSim, Exploristics, Synopsis, Virtonomy, Cad-Fem Medical, etc.

- **Regulators and Standardisation Bodies**

FDA, DIN, BSCI China, NICE, Critical Path Institute, ACQUAS, etc.

- **Clinical Research Institutions**

Istituto Ortopedico Rizzoli, Sloan Kettering Cancer Center, Royal College of Surgeons Ireland, Gratz University Hospital, Charite Berlin, Centre Nacional Inestigaciones Oncologicas, Aspirus Health, Universitätsklinikum des Saarlandes, European Society for Paediatric Oncology, etc.

