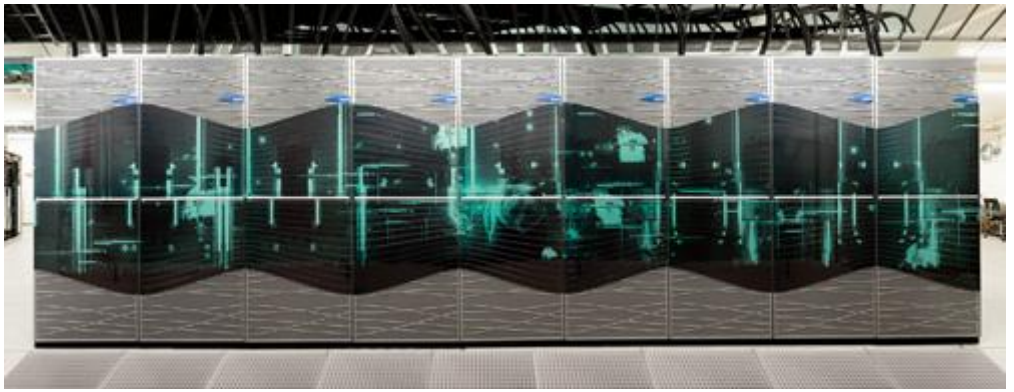
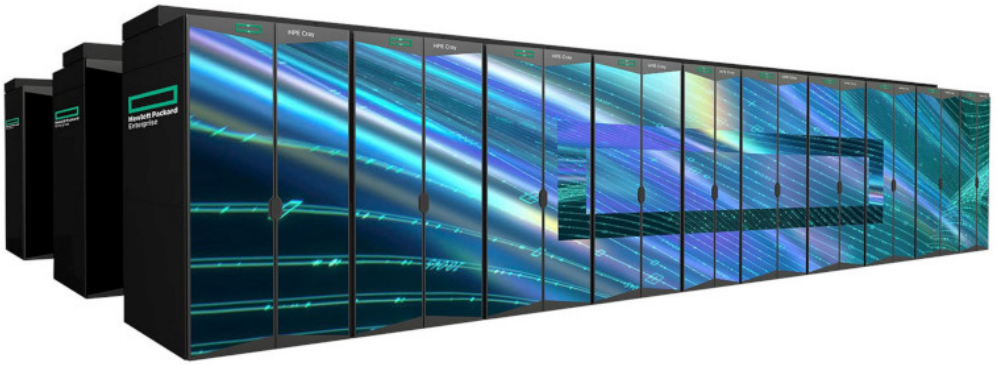
The complexity of modern High-Performance-Computing infrastructure - challenges and opportunities relevant for fusion research.

- Modern HPC (High Performance Computers) have extremly high compute capacity
- It is not easy to utilize it in practice
- There is a high demand for efficiently written software
- Pflop = A thousand trillion floating point operations per second

Mahti – Peak: 7.5 Pflops



LUMI – Peak: > 550 Pflops



What I do

- CSC – IT center for science: this is my permanent work place, and it is the Finnish unversity/State supe-computer center. My job is to support computational physics
- ACH – advanced compute hub at Helsinki University: 50-70% of my time to support development of fusion codes.
- I am a physicist, but have not worked with fusion, but have all my career worked with developing scientific code and helping scientists use HPCs.

Modern HPC infrastructure (CPUs)

- Main idea: large numbers of compute cores, with fast data transfer between them.
- Hierarchical structure of the compute cores, with some (~4) sharing e.g. level 3 (L3) cache, while most have no common memory. This is rather typical for modern CPUs.
- This hardware structure makes creating efficient software challenging.
- Hybrid codes: MPI combined with OpenMP
- Data structures and memory traffic is extremly important – naively structured code can easily render a code 100 times slower than an optimal code.
- Programming languages for HPC: C, C++, Fortran

Typical code problems

1) The code does not run in parallel – all computations on one core. Fix: parallelize.
2) Codes creates, reads and writes to/from disc constantly throughout the computation cerating lots of files: fix: reduce!, parallel-IO
3) Code reads large chunks of data from main memory and only uses a small portion and then flushes out the data and reads a new large chunk: fix: read only what is needed in an ordered fashion, do **all** computations with that data before flushing
4) Unbalanced computations: some cores have very little, while a few have very much computations, and the code has barriers where all wait for the slow ones.

GPUs

- Graphics Processing Unit – even higher level of parallelism: puts even more demand on software development.
- Programming languages: Cuda, HIP, OpenMP, (OpenACC, OpenCL)
- Works for practically any algorithm, but efficient enough to be useful only for a rather small subset of algorithms. Completely hopeless compute platform for many algorithms.
- Key: lots and lots of independent computations using input data in an ordered fashion. Think: compute y=f(x), for a very large catalogue of x:s. E.g. trajectories of many non-interacting charged particles in a constant magnetic field, or compare a given gene sequence to a large gene bank, or the time dynamics on a grid, where dynamics in each cell depend only on a small number of fixed neighbor-cells (or need to update neighbors very rarely). Compare to graphics: increase light on screen -> all pixels on the screen times 2.

ACH work - EIRENE

- The EIRENE code
  - Neutral gas transport Monte Carlo code
- Parallelization
  - Code has OpenMP with a race condition problem - atomics
- Data management
  - Delegated to the CSC EUDAT management

ACH work – MIGRANe

- Dust–wall and dust–plasma interaction
- Parallelized with MPI
- Optimized computations and cache usage

ACH work – DREAM

- Runaway electrons
- Can be run in linear and non-linear mode
- Non-linear is a Newton-Raphson type algorithm – do not converge
- New solver developed
- Implementtion of an AI-algorithm

Opportunities and challenges

Opportunities:

- Enormous compute capacity in modern HPC

Challenges:

- Extremely challenging to construct software that can reach the full potential of the hardware