Code Development: Challenges in developing the chemistry codes of tomorrow

Patrick Ettenhuber

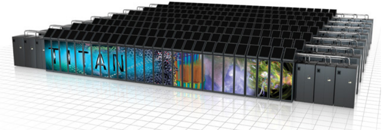
Aarhus University

July 21, 2014

Overview

Motivation

- Conventional CCSD(T) calculations scale as N^7
- Memory requirements scale as N^4
- Heavy parallelism on modern super computers (Apra et *al.*[1], 20 Water molecules)



[1] E. Apra et al. "Liquid Water: Obtaining the right answer for the right reasons". In: *SC09, submission for Gordon Bell prize* (2009).

Motivation

- Conventional CCSD(T) calculations scale as N^7

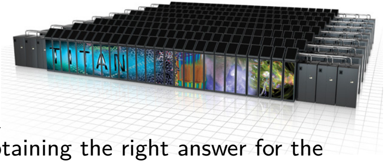
$$2\text{h} \xrightarrow{\times 2} 256\text{h}$$

- Memory requirements scale as N^4

$$460\text{GB} \xrightarrow{\times 2} 7\text{TB}$$

- Heavy parallelism on modern super computers (Apra et *al.*[1], 20 Water molecules)

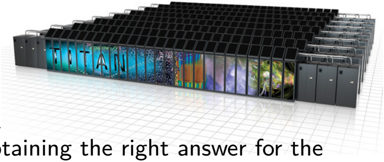
- . . . does not solve the fundamental problem



[1] E. Apra et al. “Liquid Water: Obtaining the right answer for the right reasons”. In: *SC09, submission for Gordon Bell prize* (2009).

Motivation

- Conventional CCSD(T) calculations scale as N^7
  $$2\text{h} \xrightarrow{\times 2} 256\text{h}$$
- Memory requirements scale as N^4
  $$460\text{GB} \xrightarrow{\times 2} 7\text{TB}$$
- Heavy parallelism on modern super computers (Apra et *al.*[1], 20 Water molecules)
- ...does not solve the fundamental problem → DEC
  - fragmentize problem
  - independent fragment calculations
    ⇒ Distribution of memory and work



[1] E. Apra et al. “Liquid Water: Obtaining the right answer for the right reasons”. In: *SC09, submission for Gordon Bell prize* (2009).

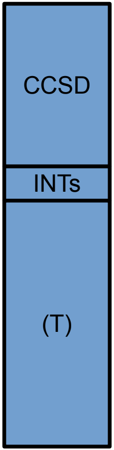
Motivation II

- DEC constitutes a linear-scaling and massively parallel framework
  Advantage and Problem: Rigorous error control

Motivation II

- DEC constitutes a linear-scaling and massively parallel framework
  Advantage and Problem: Rigorous error control
- Fragment size *a priori* unknown
- Size grows with requested precision
- Computational complexity scales as traditional methods with N_{frag}^7
  $\Rightarrow$ Parallelization of fragment calculations necessary

Motivation: CCSD(T)



N_{frag}^{6}

N_{frag}^{5}

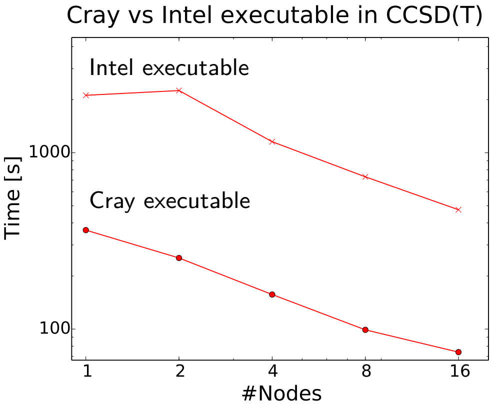
N_{frag}^{7}

Compiler choice

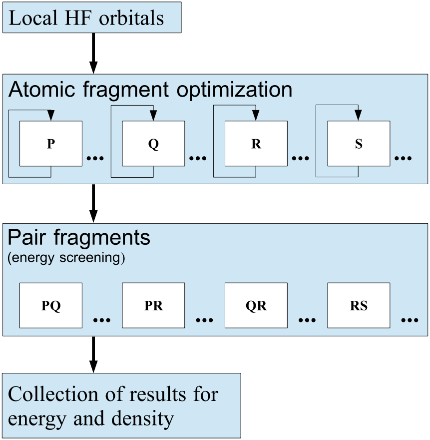
- Test of Intel vs. Cray
- Cray preferred because it is OpenACC enabled

Compiler choice

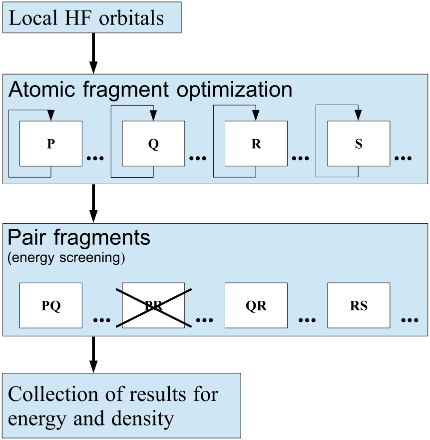
- Test of Intel vs. Cray
- Cray preferred because it is OpenACC enabled
- For the investigated systems, speedup of 5-8x



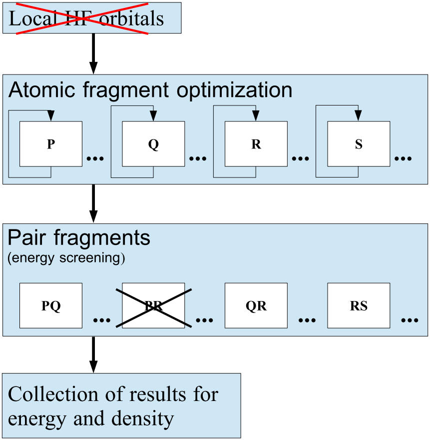
DEC calculation overview



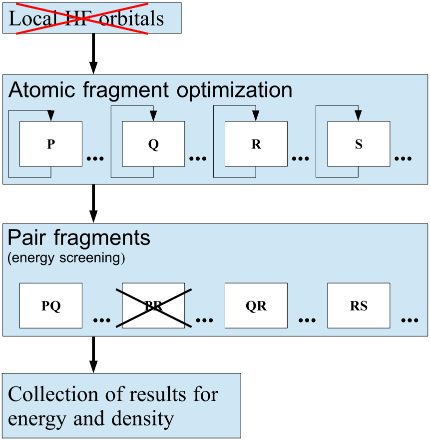
DEC calculation overview



DEC calculation overview



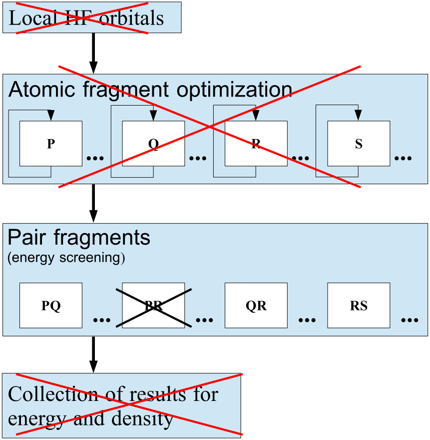
DEC calculation overview



n calculations
ca. 10%

const ∗ *n*
ca. 90%

DEC calculation overview

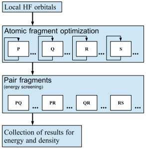


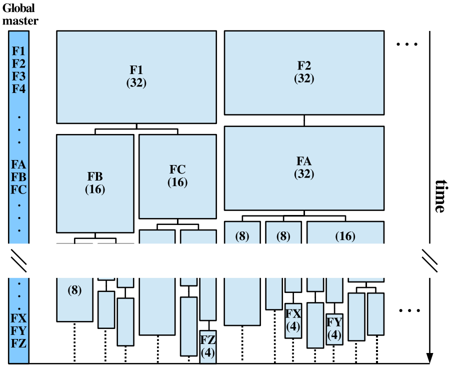
n calculations
ca. 10%

const ∗ *n*
ca. 90%

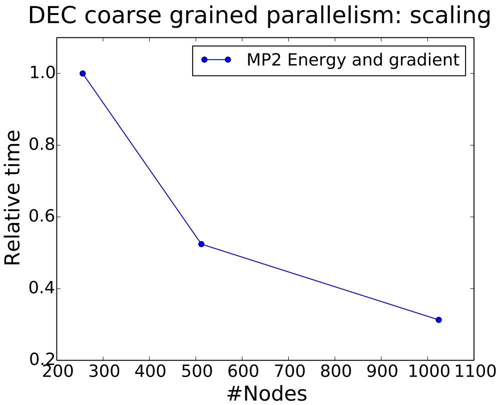
Coarse-grained parallelism

- DEC has 3 levels of parallelism
- Coarse granularity is given by DEC-splitting

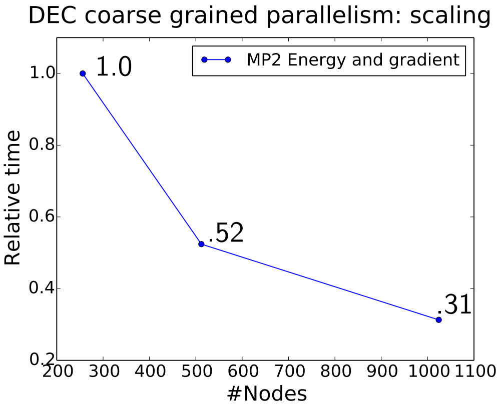




Coarse-grained parallelism: Scaling

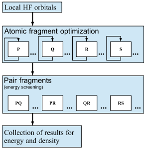


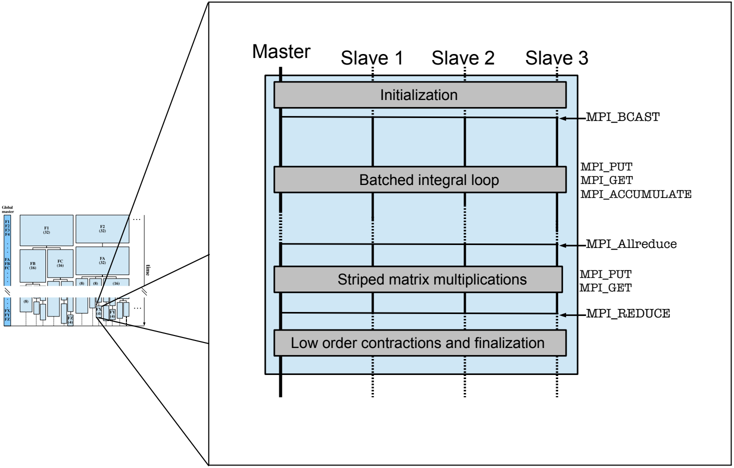
Coarse-grained parallelism: Scaling



Medium-grained parallelism

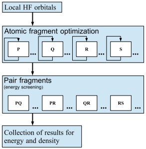
- Heavy work N_{frag}^7
- Heavy memory requirements N_{frag}^4

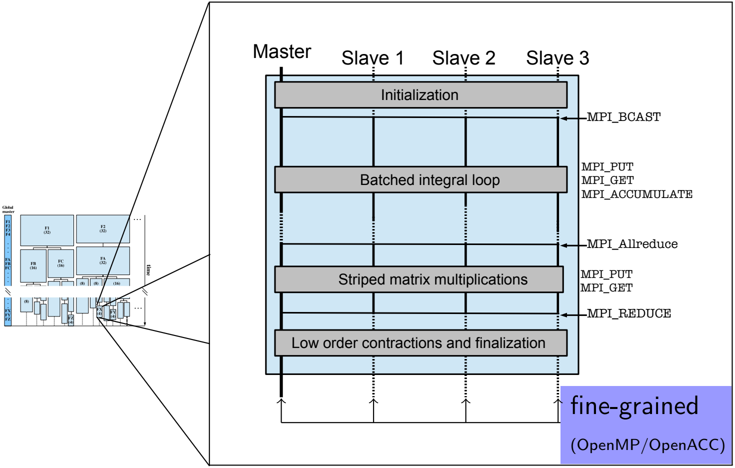




Medium-grained parallelism

- Heavy work N_{frag}^7
- Heavy memory requirements N_{frag}^4



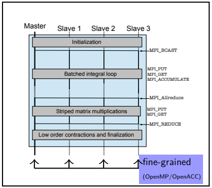


Fine-grained Parallelism

- High memory requirements per fragment
  ⇒ use all memory per node (1ppn)
- test OMP with `aprun -cc 0,2,4,...,1,3,...`

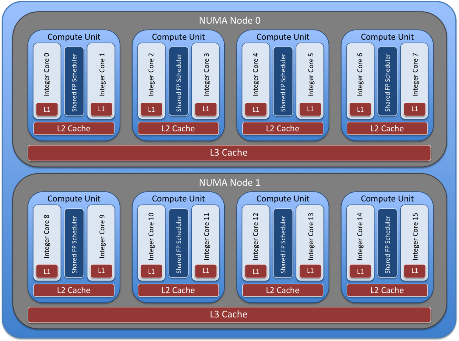
[1] Picture from
https://www.olcf.ornl.gov/wp-content/uploads/2012/11/opteron_6274_cpu.png

Fine-grained Parallelism



- High memory requirements per fragment
  $\Rightarrow$ use all memory per node (1ppn)
- test OMP with `aprun -cc 0,2,4,...,1,3,...`

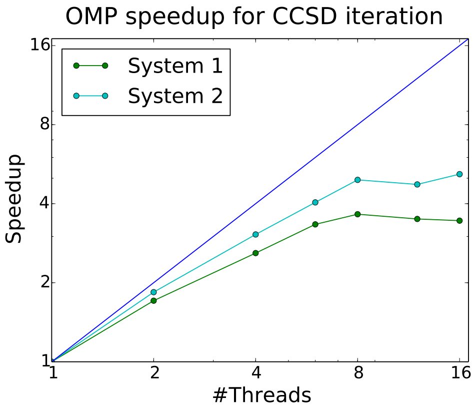
AMD Opteron™ 6274 (Interlagos) CPU



[1] Picture from
https://www.olcf.ornl.gov/wp-content/uploads/2012/11/opteron_6274_cpu.png

Fine-grained Parallelism

Figure : Timing of the algorithm with respect to the threads




```
aprun -n $MPIPROC -N 1 -d 8 -j 1
```

Fine-grained parallelism: OpenMP and OpenACC

- OpenMP is used explicitly for the calculation of integrals
- OpenMP is otherwise provided by threaded libraries

Fine-grained parallelism: OpenMP and OpenACC

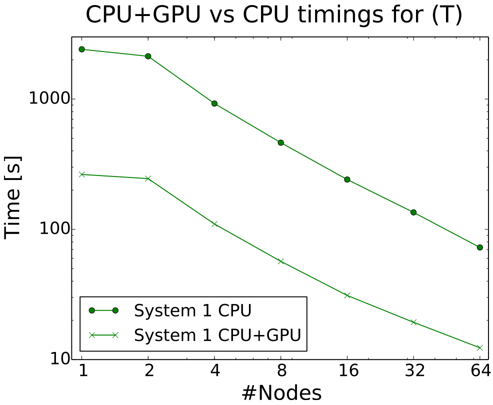
- OpenMP is used explicitly for the calculation of integrals
- OpenMP is otherwise provided by threaded libraries
- OpenACC is used in the (T) correction
- OpenACC implementation of the integral calculation is in progress
- OpenACC implementations for MP2 and CCSD residual are considered

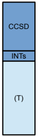
Fine-grained parallelism: OpenMP and OpenACC

- OpenMP is used explicitly for the calculation of integrals
- OpenMP is otherwise provided by threaded libraries
- OpenACC is used in the (T) correction
- OpenACC implementation of the integral calculation is in progress
- OpenACC implementations for MP2 and CCSD residual are considered

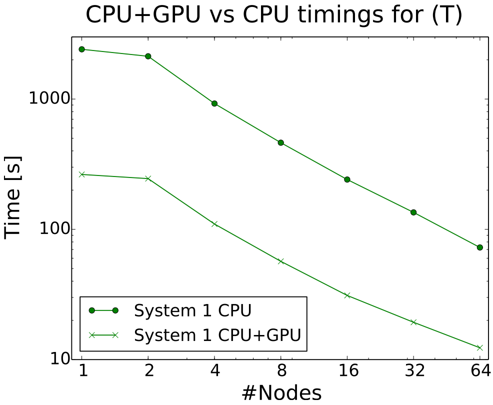
Use of directives for portability, ease of implementation and code maintenance

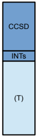
Fine-grained parallelism: Accelerators in (T)





Fine-grained parallelism: Accelerators in (T)





- unstructured data regions
- 10 unique async handles
- asynchronous wait directives to avoid race conditions
- calls to `dgemm_acc_openacc_async` of the `cray-libsci_acc` library

Medium-grained parallelism

Diverse parallelism. Specific for the level of theory. Difficult to implement, but *traditional* parallelization techniques can be used

- Non-linear equation solver
- Residual routines
- perturbative corrections

Medium-grained parallelism

Diverse parallelism. Specific for the level of theory. Difficult to implement, but *traditional* parallelization techniques can be used

- Non-linear equation solver (tasks/memory: $N_{\text{frag}}^4 / N_{\text{frag}}^4$)
- Residual routines (tasks/memory: $N_{\text{frag}}^6 / N_{\text{frag}}^4$)
- perturbative corrections (tasks/memory: $N_{\text{frag}}^7 / N_{\text{frag}}^4$)

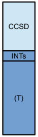
Medium-grained parallelism

Diverse parallelism. Specific for the level of theory. Difficult to implement, but *traditional* parallelization techniques can be used

- Non-linear equation solver (tasks/memory: $N_{\text{frag}}^4/N_{\text{frag}}^4$)
- Residual routines (tasks/memory: $N_{\text{frag}}^6/N_{\text{frag}}^4$)
- perturbative corrections (tasks/memory: $N_{\text{frag}}^7/N_{\text{frag}}^4$)

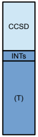
$\Rightarrow$ distribution of tasks (N_{frag}^7) and memory (N_{frag}^4) using one-sided MPI routines in a tiled tensor framework.

Medium-grained parallelism: CCSD



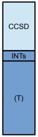
- Use of the conjugate-residual technique with preconditioning

Medium-grained parallelism: CCSD



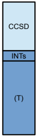
- Use of the conjugate-residual technique with preconditioning
- Requires to save iterative subspace information for fast and stable convergence $\rightarrow$ employ CROP solver, only use last 4 iterations

Medium-grained parallelism: CCSD



- Use of the conjugate-residual technique with preconditioning
- Requires to save iterative subspace information for fast and stable convergence → employ CROP solver, only use last 4 iterations
- Distribute tensors in PDM

Medium-grained parallelism: CCSD



- Use of the conjugate-residual technique with preconditioning
- Requires to save iterative subspace information for fast and stable convergence → employ CROP solver, only use last 4 iterations
- Distribute tensors in PDM
- Only vector-vector operations, easy to perform in PDM

Medium-grained parallelism: Obtaining the residual r_n

- Construction of r_n scales as N_{frag}^6

Medium-grained parallelism: Obtaining the residual r_n

- Construction of r_n scales as N_{frag}^6
- Avoid big intermediates by batching and remote updates (`MPI_ACCUMULATE`)

Medium-grained parallelism: Obtaining the residual r_n

- Construction of r_n scales as N_{frag}^6
- Avoid big intermediates by batching and remote updates (MPI_ACCUMULATE)
- Avoid storing → recalculate

Medium-grained parallelism: Obtaining the residual r_n

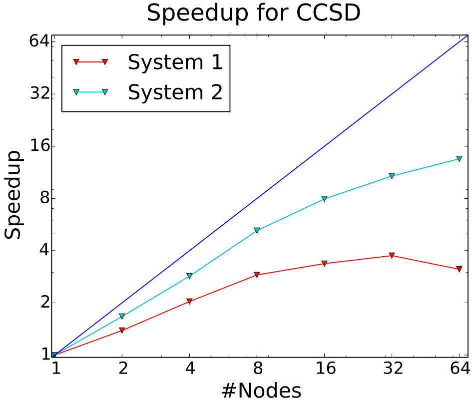
- Construction of r_n scales as N_{frag}^6
- Avoid big intermediates by batching and remote updates (`MPI_ACCUMULATE`)
- Avoid storing $\rightarrow$ recalculate
- Exploit symmetry

Medium-grained parallelism: Obtaining the residual r_n

- Construction of r_n scales as N_{frag}^6
- Avoid big intermediates by batching and remote updates (`MPI_ACCUMULATE`)
- Avoid storing $\rightarrow$ recalculate
- Exploit symmetry
- Several CCSD schemes with different focus on N_{frag}

Medium-grained parallelism: Solving the CCSD problem

- Time in solver is negligible
- Use best scheme for current fragment
- Use DEC parallelism to ensure most efficient distribution of fragment



Medium-grained parallelism: Perturbative methods

- Corresponding to solving an equation in one step

Medium-grained parallelism: Perturbative methods

- Corresponding to solving an equation in one step
- Two methods are currently implemented: MP2, (T)

Medium-grained parallelism: Perturbative methods

- Corresponding to solving an equation in one step
- Two methods are currently implemented: MP2, (T)
- Characterized by calculation of integrals and matrix-matrix multiplications

Medium-grained parallelism: Perturbative methods

- Corresponding to solving an equation in one step
- Two methods are currently implemented: MP2, (T)
- Characterized by calculation of integrals and matrix-matrix multiplications
- Scaling MP2: N_{frag}^{5} and (T): N_{frag}^{7}

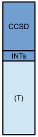
Medium-grained parallelism: Perturbative methods

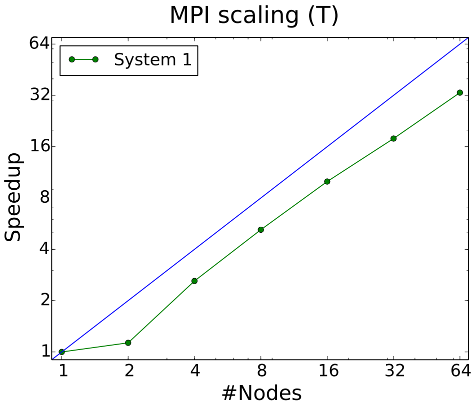
- Corresponding to solving an equation in one step
- Two methods are currently implemented: MP2, (T)
- Characterized by calculation of integrals and matrix-matrix multiplications
- Scaling MP2: N_{frag}^5 and (T): N_{frag}^7
- "Easy" and efficient to port to GPUs
  $\rightarrow$ 5-9x speedup for (T)

Medium-grained parallelism: Perturbative methods

- Corresponding to solving an equation in one step
- Two methods are currently implemented: MP2, (T)
- Characterized by calculation of integrals and matrix-matrix multiplications
- Scaling MP2: N_{frag}^5 and (T): N_{frag}^7
- "Easy" and efficient to port to GPUs
  $\rightarrow$ 5-9x speedup for (T)
- Simple parallelization over many nodes by batching

Medium-grained parallelism: Solving the (T) problem





Medium-grained parallelism: Trouble with MPI one-sided

- One sided MPI calls are handy, communication can be hidden and save memory

Medium-grained parallelism: Trouble with MPI one-sided

- One sided MPI calls are handy, communication can be hidden and save memory

BUT:

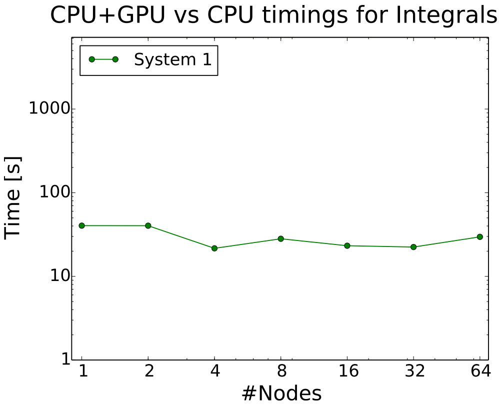
- Used asynchronously terrible, especially `MPI_ACCUMULATE` → partial serialization

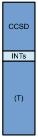
Medium-grained parallelism: Trouble with MPI one-sided

- One sided MPI calls are handy, communication can be hidden and save memory

BUT:

- Used asynchronously terrible, especially `MPI_ACCUMULATE` $\rightarrow$ partial serialization





Medium-grained parallelism: Trouble with MPI one-sided

SOLUTION:

- Asynchronous progress engine (communication threads in MPI)
- Efficient use with CLE5.2 and MPI3 possible

```
aprun -e MPICH_NEMESIS_ASYNC_PROGRESS=1
      -e MPICH_MAX_THREAD_SAFETY=multiple
      -e MPICH_RMA_OVER_DMAPP=1
      -n $MPIPROC -N 1 -d 8
      -cc  0,2,4,...,1,3,...  -r 1
```

Outlook: Specifications of a new machine

- Memory, more per node and per core

Outlook: Specifications of a new machine

- Memory, more per node and per core
- More memory on the accelerators

Outlook: Specifications of a new machine

- Memory, more per node and per core
- More memory on the accelerators
- Either OpenACC or OpenMP-4 compatible use of accelerators

Outlook: Specifications of a new machine

- Memory, more per node and per core
- More memory on the accelerators
- Either OpenACC or OpenMP-4 compatible use of accelerators
- General purpose codes should work, i.e. all standards should be supported

Outlook: Specifications of a new machine

- Memory, more per node and per core
- More memory on the accelerators
- Either OpenACC or OpenMP-4 compatible use of accelerators
- General purpose codes should work, i.e. all standards should be supported
- More support for asynchronicity (MPI/GPU)

Outlook: Specifications of a new machine

- Memory, more per node and per core
- More memory on the accelerators
- Either OpenACC or OpenMP-4 compatible use of accelerators
- General purpose codes should work, i.e. all standards should be supported
- More support for asynchronicity (MPI/GPU)
- Large molecular systems are in the focus of science
  $\rightarrow$ more nodes or more runtime

Acknowledgements

Work supported by:

- US Department of Energy
- Oak Ridge Leadership Computing Facility
- European Research Council
- Danish Council for Independent Research – Natural Sciences
- Aarhus University