OAK RIDGE National Laboratory

CAAR Porting Experience on Summit: LS-Dalton

Ashleigh Barnes

Summit Training Workshop

ORNL, Feb 13th, 2019

ORNL is managed by UT-Battelle LLC for the US Department of Energy

U.S. DEPARTMENT OF ENERGY

OAK RIDGE National Laboratory

LS-Dalton: Overview

LS-Dalton

LS-Dalton (Linear Scaling Dalton) is open-source scientific software for **electronic structure calculations**. LS-Dalton is developed at Aarhus and Oslo Universities as well as at ORNL. Most parts of LS-Dalton employ linear scaling and massively parallel implementations, which makes it suitable for calculations on **large molecular systems**, in particular when the calculations are carried out on large super computer architectures. In particular **Divide-Expand-Consolidate (DEC)** scheme allows for **linear-scaling Coupled Cluster Methods**.

Key Features

Divide-Expand-Consolidate(DEC) models:

- DEC-MP2 energy, density and gradient, unrestricted energy
- DEC-RI-MP2 energy and gradient, Laplace-Transformed RI-MP2
- DEC-CCSD energy and gradient, unrestricted energy
- DEC-CCSD(T) energy

Computational details

- Languages: Fortran90, Fortran2003
- Runtime: MPI/OpenMP/OpenACC

Developers (CAAR TEAM)

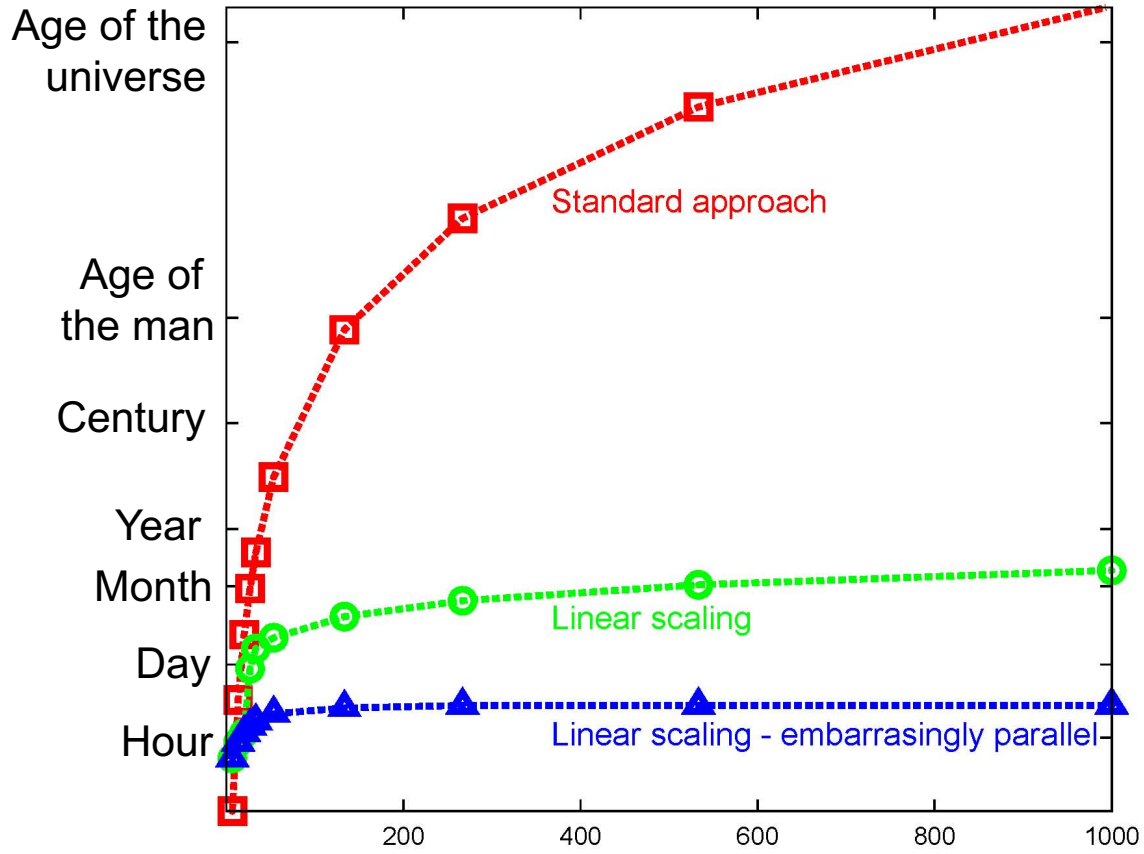
Dmytro Bykov
Ashleigh Barnes
Dmitry I. Liakh
Poul Jorgensen
Thomas Kjaergaard
Patrick Ettenhuber
Janus Eriksen
Kasper Kristensen
Pablo Baudin
Philip Pawlowski
Yang Ming Wang

Full list of developers

http://daltonprogram.org

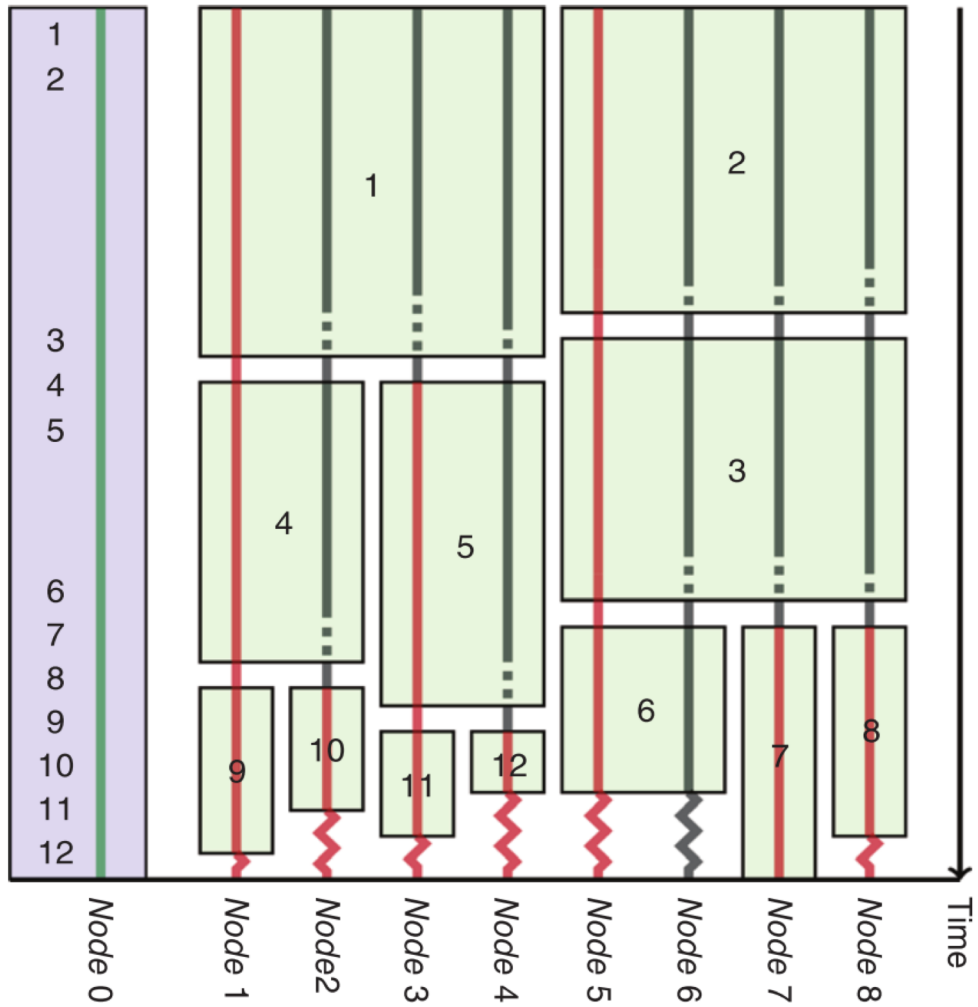
Achieving Linear Scaling

- Canonical coupled cluster (CC) methods are limited by poor scaling with system size:
  - MP2: N^5
  - CCSD: N^6
  - CCSD(T): N^7



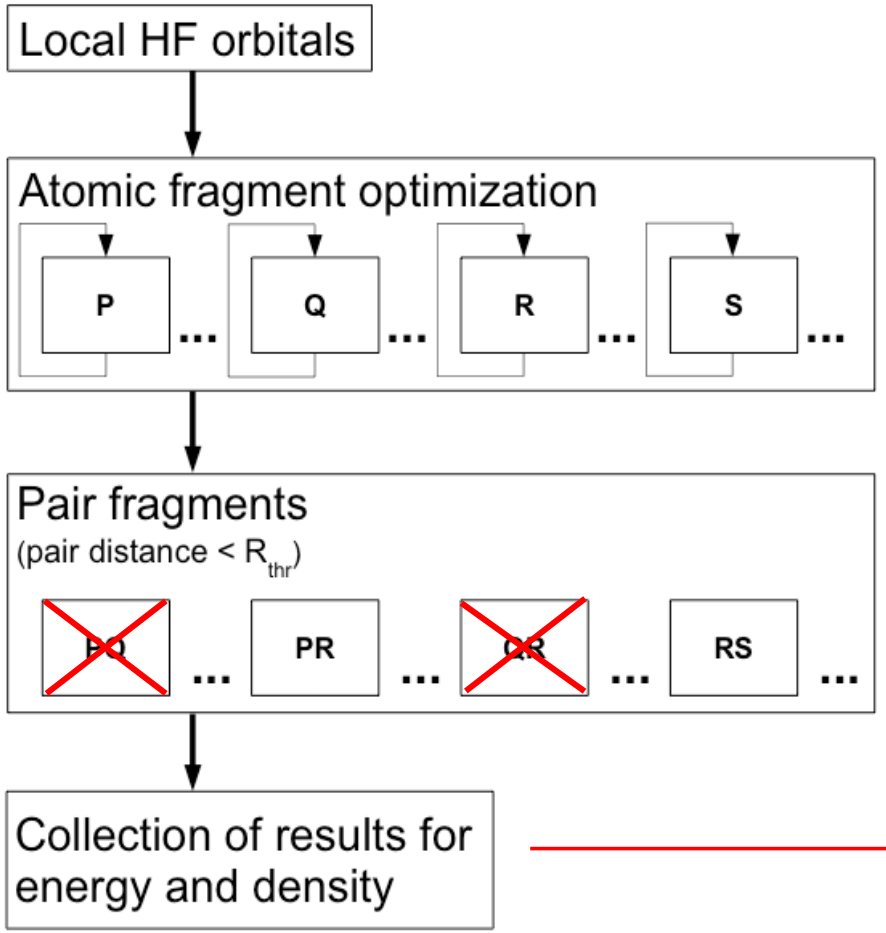
Achieving Linear Scaling

- Divide-Expand-Consolidate (DEC) scheme
  - Local HF orbitals lead to natural fragmentation of the system
  - Correlation energy evaluated for each fragment independently
  - Multiple levels of parallelism



Coarse Grained: Fragments calculated independently
Medium Grained: Each fragment calculation distributed over multiple nodes **(MPI)**
Fine grained: Thread-level parallelism within each node **(OpenMP, OpenACC)**

DEC Scheme



Performed at MP2 (RI-MP2) level

P = Occupied orbital space

$[P]$ = Virtual orbital space

$$E_P = \sum_{\substack{i\in P, j\in P \\ ab\in[P]}} \left(t_{ij}^{ab} + t_i^a t_j^b\right)\left(2g_{iajb} - g_{ibja}\right)$$

$$\Delta E_{PQ} = \sum_{\substack{i\in P, j\in Q \\ ab\in[P]\cup[Q]}} \left(t_{ij}^{ab} + t_i^a t_j^b\right)\left(2g_{iajb} - g_{ibja}\right) + \quad P \leftrightarrow Q \text{ term}$$

$$E_{\text{corr}} = \sum_{P}^{N_{\text{frag}}} \left[E_P + \sum_{Q<P}^{N_{\text{frag}}} \Delta E_{PQ}\right]$$

OAK RIDGE National Laboratory

Porting experience on Summit: Strategy

Porting Strategy

Targeting high-level directive-based porting to GPU: **OpenACC** and **GPU-optimized libraries**.

- GPU offloading of FLOP-intensive parts of **RI-MP2** module using **openACC** and **cuBLAS**.
  - Including Laplace-transformed RI-MP2 (more efficient implementation)
- Increased dependence on parallel distributed memory tensors via **ScaTeLib** library
  - Allow for easy porting of tensor contractions to GPU via **TAL-SH** library

GPU-porting of RI-MP2 Module

Focus attention on rate-determining step:
Construction of integrals g_{aibj} and amplitudes t_{ij}^{ab}

Step	Algorithmic step	Cost	Storage
	Loop B Loop J		
1	$\tilde{t}_{I(J)}^{A(B)} = \sum_{\alpha} C_{AI}^{\alpha} C_{BJ}^{\alpha}$	$N_{aux,AOS} O_{AOS}^2 V_{AOS}^2$	$O_{AOS} V_{AOS}$
2	$t_{I(J)}^{A(B)} = \tilde{t}_{I(J)}^{A(B)} (\epsilon_I + \epsilon_J - \epsilon_A - \epsilon_B)^{-1}$	$O_{AOS}^2 V_{AOS}^2$	$O_{AOS} V_{AOS}$
3	$t_{i(J)}^{A(B)} = \sum_{I} U_{iI} t_{I(J)}^{A(B)}$	$O_{AOS}^2 V_{AOS}^2 O_{EOS}$	$V_{AOS} O_{EOS}$
4	$t_{iJ}^{a(B)} = \sum_{A} U_{aA} t_{i(J)}^{A(B)}$	$O_{AOS} V_{AOS}^3 O_{EOS}$	$O_{AOS} V_{AOS} O_{EOS}$
	End Loop J		
5	$t_{ij}^{aB} = \sum_{J} U_{jJ} t_{iJ}^{a(B)}$	$O_{AOS} V_{AOS}^2 O_{EOS}^2$	$V_{AOS}^2 O_{EOS}^2$
	End Loop B		
6	$t_{ij}^{ab} = \sum_{B} U_{bB} t_{ij}^{aB}$	$V_{AOS}^3 O_{EOS}^2$	$V_{AOS}^2 O_{EOS}^2$
7	$C_{Ai}^{\alpha} = \sum_{I} U_{iI} C_{AI}^{\alpha}$	$N_{aux,AOS} V_{AOS} O_{AOS} O_{EOS}$	$N_{aux,AOS} V_{AOS} O_{EOS}$
8	$C_{ai}^{\alpha} = \sum_{A} U_{aA} C_{Ai}^{\alpha}$	$N_{aux,AOS} V_{AOS}^2 O_{EOS}$	$N_{aux,AOS} V_{AOS} O_{EOS}$
9	$g_{aibj} = \sum_{\alpha} C_{ai}^{\alpha} C_{bj}^{\alpha}$	$N_{aux,AOS} V_{AOS}^2 O_{EOS}^2$	$V_{AOS}^2 O_{EOS}^2$

Dgemm → **cuBLAS**

Memory managed with OpenACC data regions

D. Bykov and T. Kjaergaard, *Journal of Computational Chemistry,* **38**, 228–237 (2017)

GPU-porting of RI-MP2 Module

Focus attention on rate-determining step:
Construction of integrals g_{aibj} and amplitudes t_{ij}^{ab}

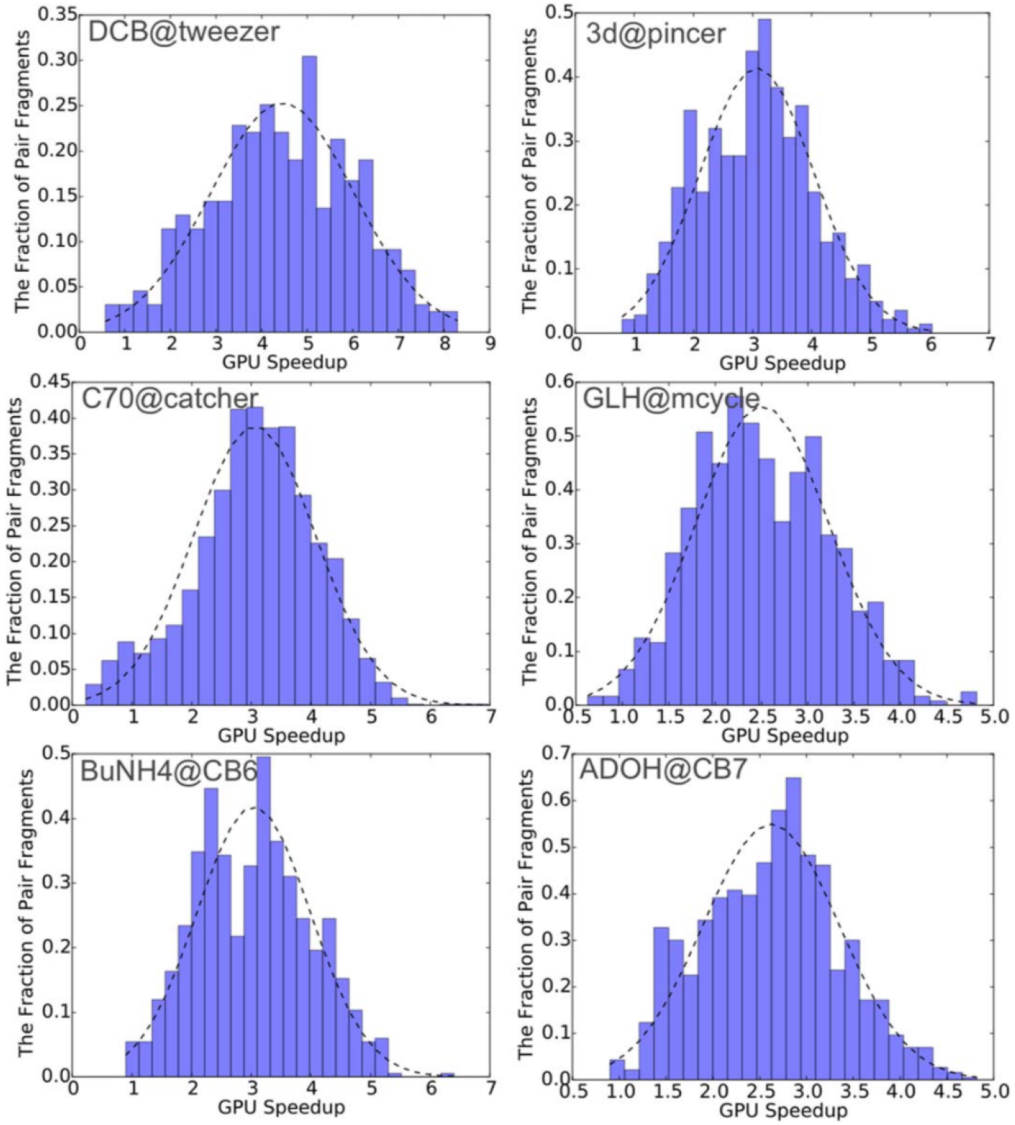
Step	Algorithmic step	Cost	Storage
	Loop B		
	Loop J		
1	$\tilde{t}_{I(J)}^{A(B)} = \sum_{\alpha} C_{AI}^{\alpha} C_{BJ}^{\alpha}$	$N_{aux,AOS} O_{AOS}^2 V_{AOS}^2$	$O_{AOS} V_{AOS}$
2	$t_{I(J)}^{A(B)} = \tilde{t}_{I(J)}^{A(B)} (\epsilon_I + \epsilon_J - \epsilon_A - \epsilon_B)^{-1}$	$O_{AOS}^2 V_{AOS}^2$	$O_{AOS} V_{AOS}$
3	$t_{i(J)}^{A(B)} = \sum_{I} U_{iI} t_{I(J)}^{A(B)}$	$O_{AOS}^2 V_{AOS}^2 O_{EOS}$	$V_{AOS} O_{EOS}$
4	$t_{iJ}^{a(B)} = \sum_{A} U_{aA} t_{i(J)}^{A(B)}$	$O_{AOS} V_{AOS}^3 O_{EOS}$	$O_{AOS} V_{AOS} O_{EOS}$
	End Loop J		
5	$t_{ij}^{aB} = \sum_{J} U_{jJ} t_{iJ}^{a(B)}$	$O_{AOS} V_{AOS}^2 O_{EOS}^2$	$V_{AOS}^2 O_{EOS}^2$
	End Loop B		
6	$t_{ij}^{ab} = \sum_{B} U_{bB} t_{ij}^{aB}$	$V_{AOS}^3 O_{EOS}^2$	$V_{AOS}^2 O_{EOS}^2$
7	$C_{Ai}^{\alpha} = \sum_{I} U_{iI} C_{AI}^{\alpha}$	$N_{aux,AOS} V_{AOS} O_{AOS} O_{EOS}$	$N_{aux,AOS} V_{AOS} O_{EOS}$
8	$C_{ai}^{\alpha} = \sum_{A} U_{aA} C_{Ai}^{\alpha}$	$N_{aux,AOS} V_{AOS}^2 O_{EOS}$	$N_{aux,AOS} V_{AOS} O_{EOS}$
9	$g_{aibj} = \sum_{\alpha} C_{ai}^{\alpha} C_{bj}^{\alpha}$	$N_{aux,AOS} V_{AOS}^2 O_{EOS}^2$	$V_{AOS}^2 O_{EOS}^2$

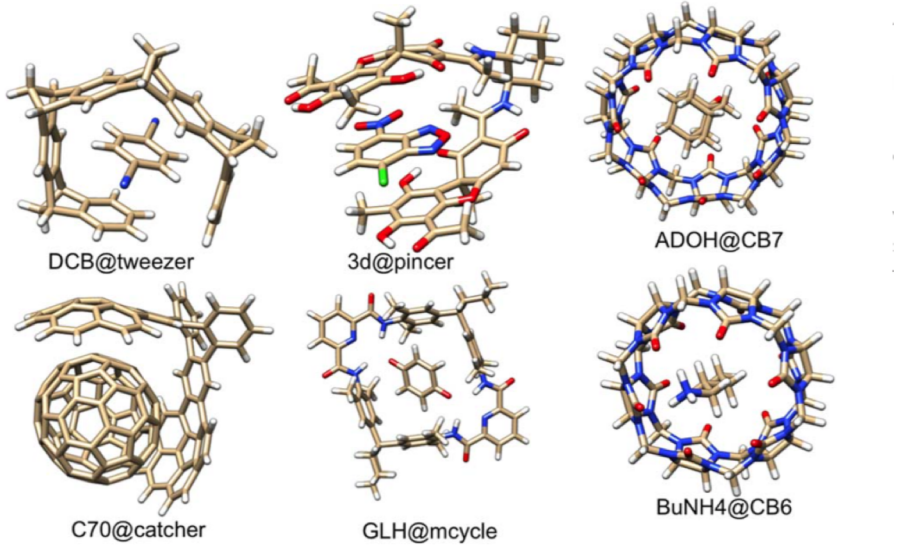
Memory adaptive scheme:

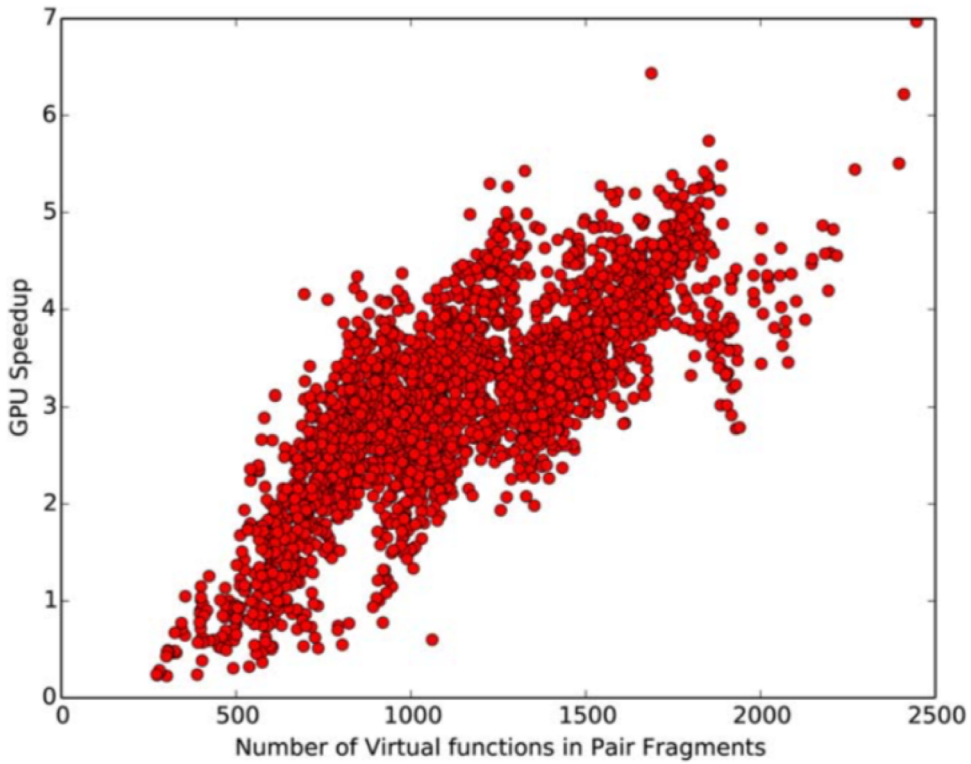
- **Tiny fragments**: Limited by data transfer time. Full algorithm performed on CPU.
- **Small fragments**: Not limited by data transfer or memory. All steps performed on GPU.
- **Large fragments**: Memory limited. Steps 1-5, 7-9 on GPU, step 6 on CPU.
- **Huge fragments**: Smallest intermediates can't fit in GPU memory. Use host cuBLAS calls, library responsible for moving data to and from GPU as needed.

D. Bykov and T. Kjaergaard, *Journal of Computational Chemistry,* **38**, 228–237 (2017)

GPU-porting of RI-MP2 Module







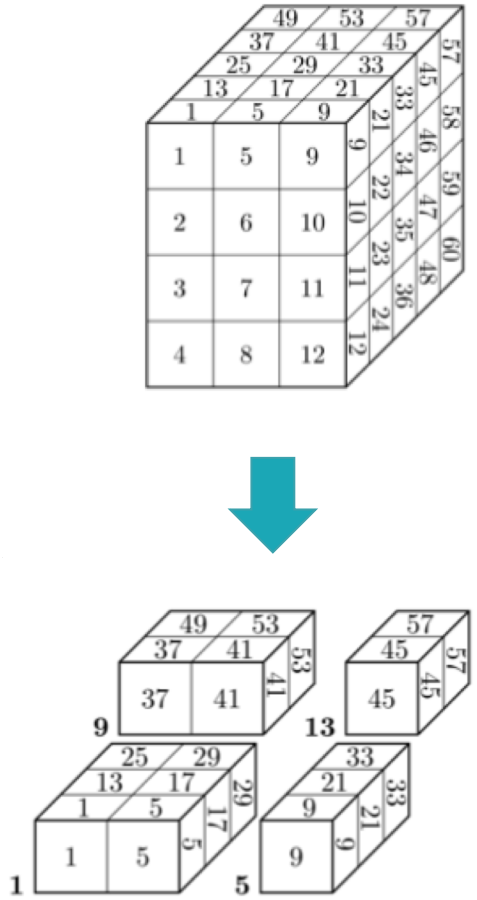
D. Bykov and T. Kjaergaard, *Journal of Computational Chemistry,* **38**, 228–237 (2017)

ScaTelib integration with TAL-SH library

- CC calculations require evaluation of tensor contractions. Efficient handling of these contractions is key to good performance of these modules.
- Currently utilizing ScaTeLib: a Scalable Tensor Library in order to distribute tensor contractions across multiple ranks via tiling.
- E.g.:

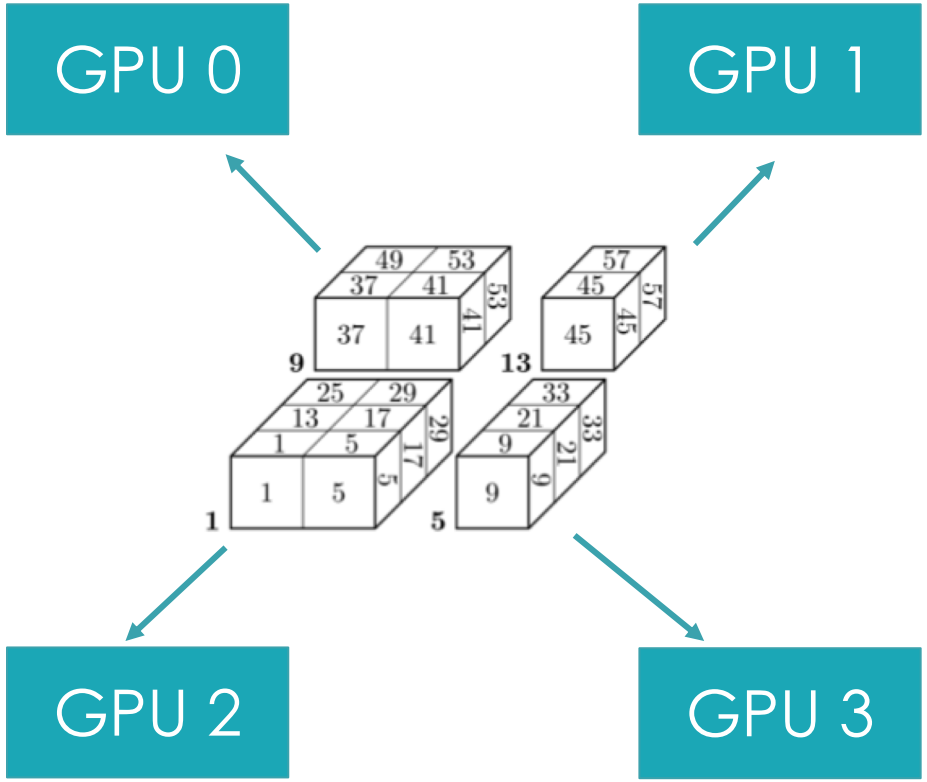
$$C_{ijkl} = \alpha \sum_{a,b} A_{ajlb} B_{kbai} + \beta C_{ijkl}$$

  - A and B must be sorted appropriately: $A_{ajlb} \Rightarrow A_{jlab}$, $B_{kbai} \Rightarrow B_{abki}$
  - Call dgemm
  - Returns contribution to local C tile as C_{jlki}. Reorder $\Rightarrow C_{ijkl}$ and add to local tile.

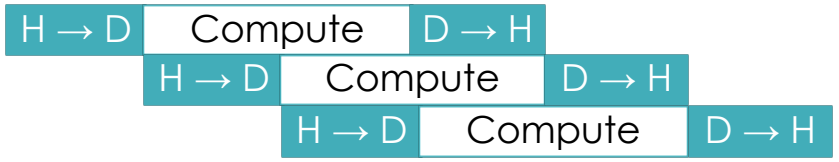


ScaTelib integration with TAL-SH library

- TAL-SH: Tensor Algebra Library for Shared-memory systems. Integrated as backend for ScaTeLib.



- Handles sorting, no explicit reorder calls necessary in ScaTeLib
- Asynchronous task scheduling
- Tasks are pipelined to overlap computation and data transfer
  - 2 active tasks per GPU at any time



- Basic implementation provides ~10x speedup on Summit
  - 1 node: jsrun -n 6 -r 6 -a 1 -g 1 -c 7 -brs
  - CPU version uses ESSL

OAK RIDGE National Laboratory

Porting experience on Summit: Lessons Learned

Compiling and Debugging

Initial compilation and test runs

- Initial efforts on SummitDev
- Focused initially on PGI compilers due to need for extensive OpenACC support
- We were able to compile with GPU support using PGI on Summit with little trouble (once deprecated compiler flags were removed from cmake)
  - Test runs experienced hangs or immediate crashes with "invalid free()" errors
  - Crash was found to be caused by calling acc_init before mpi_init.
  - Hangs were caused by overlapping OpenACC and OpenMP regions in one subroutine. Solution was to remove OpenMP regions.

Challenges

- Wrong answers
- Random hangs with large node counts

Challenges

- Wrong answers
  - Possibly a compiler issue, but was never isolated. This was fixed with next compiler and software stack update (PGI/18.3).
- Random hangs with large node counts
  - Only when using multiple threads
  - Attached gdb debugger to each process – all but 3 were waiting in MPI collectives. Remaining 3 were stuck in an OMP CRITICAL region
  - Replacing OMP CRITICAL with OMP ATOMIC wherever possible fixed the hanging problem
- Currently experiencing problems with GPU builds using PGI versions later than 18.3
  - Cublas handle corruption or segmentation faults at MPI calls

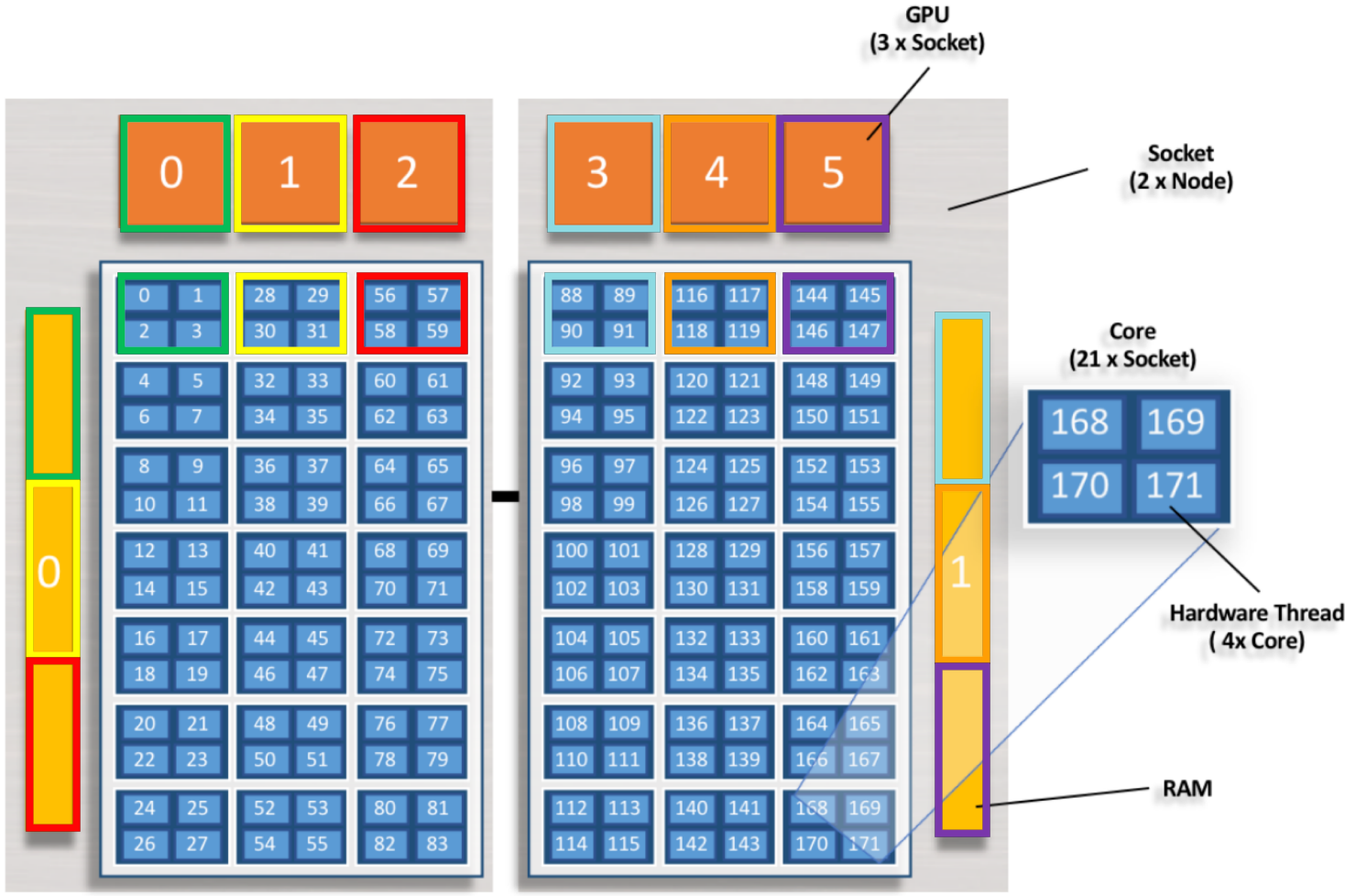
OAK RIDGE National Laboratory

Porting experience on Summit: Lessons Learned

Improving Performance

Performance improvement: optimizing jsrun options

- 6 RS/node: 1 GPU/RS
  - CC calculations are extremely memory intensive. Balance GPU usage and memory/rank by using 1 MPI rank/GPU
  - For our code, SMT1 > SMT2 > SMT4. SMT4 with 28 OMP threads/rank was about 2x as slow as SMT1 with 7 OMP threads per rank.
    - SMT2 with 14 threads was slightly slower than SMT1



Example: jsrun -n 24 -r 6 -a 1 -g 1 -c 7 -brs ./lsdalton.x

Performance improvement: NVProf

Instrument code with nvtx custom ranges (used nvtx_mod.F90 from David Appelhans, IBM: https://github.com/dappelha/gpu-tips.git)

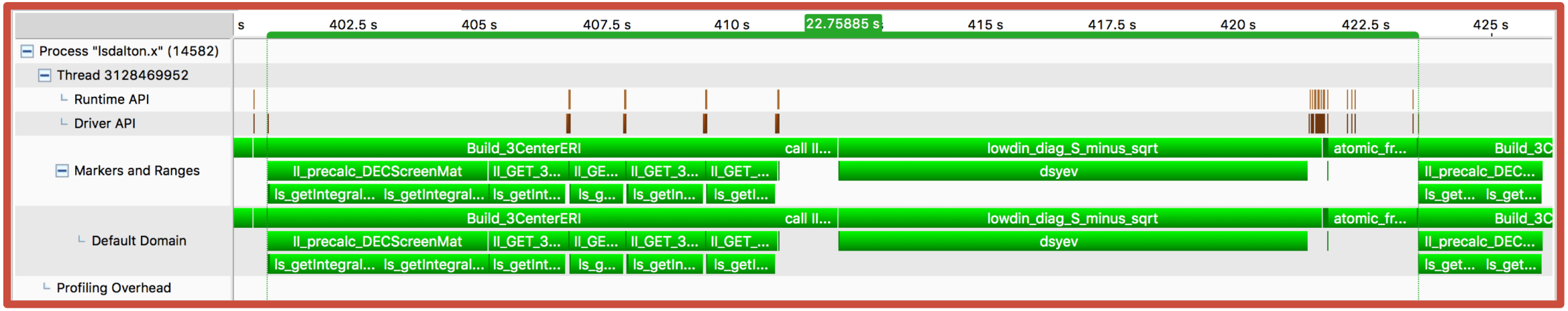
- Where are we wasting time?
- What routines can be easily ported to the GPU (e.g. via libraries)?

Performance improvement: NVProf

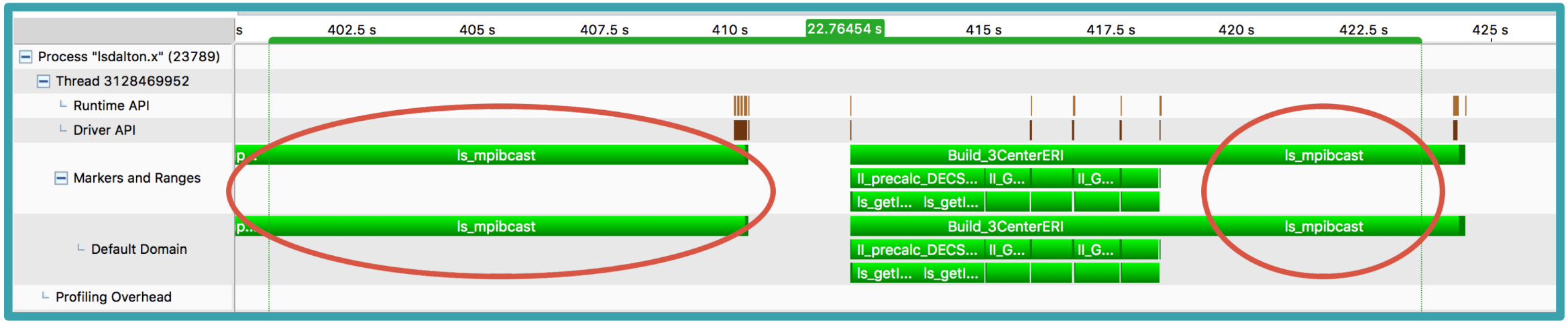
- **Where are we wasting time?**
- What routines can be easily ported to the GPU (e.g. via libraries)?

Performance improvement: NVProf

Master:



Worker:

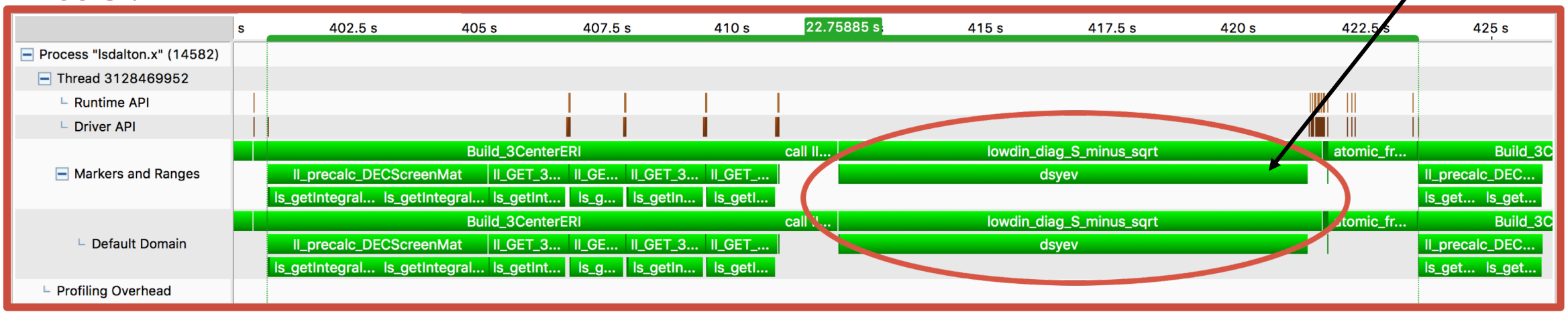


Performance improvement: NVProf

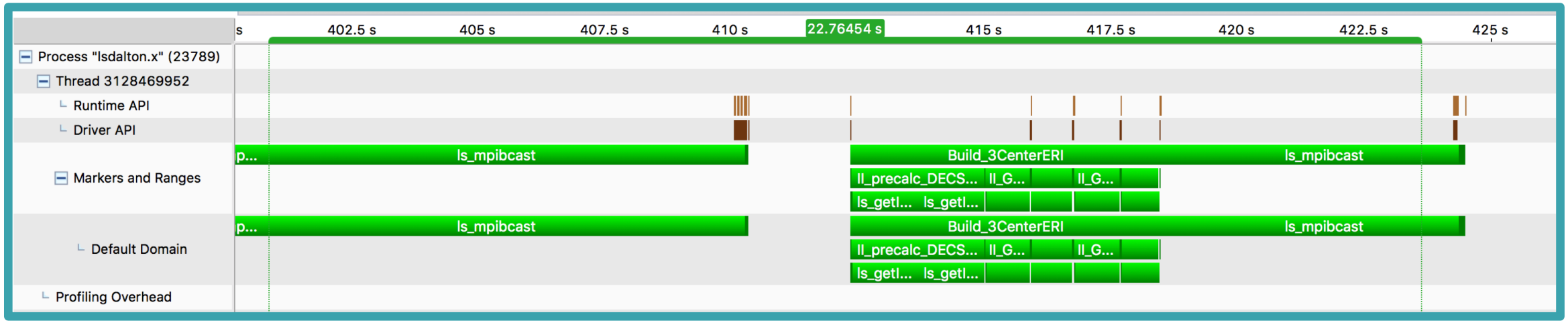
- Where are we wasting time?
- **What routines can be easily ported to the GPU (e.g. via libraries)?**

Performance improvement: NVProf

Master:

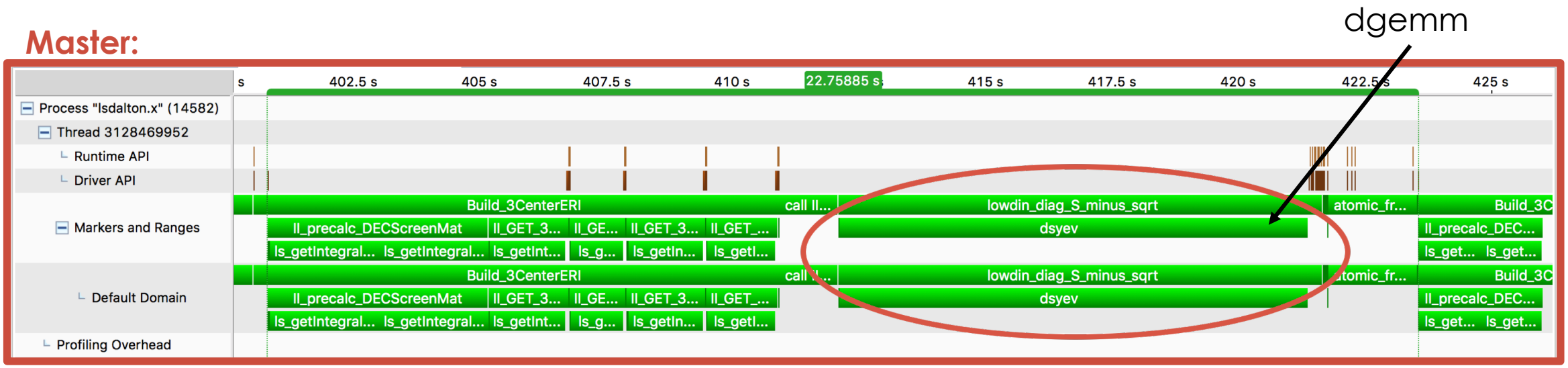


Worker:



Performance improvement: NVProf

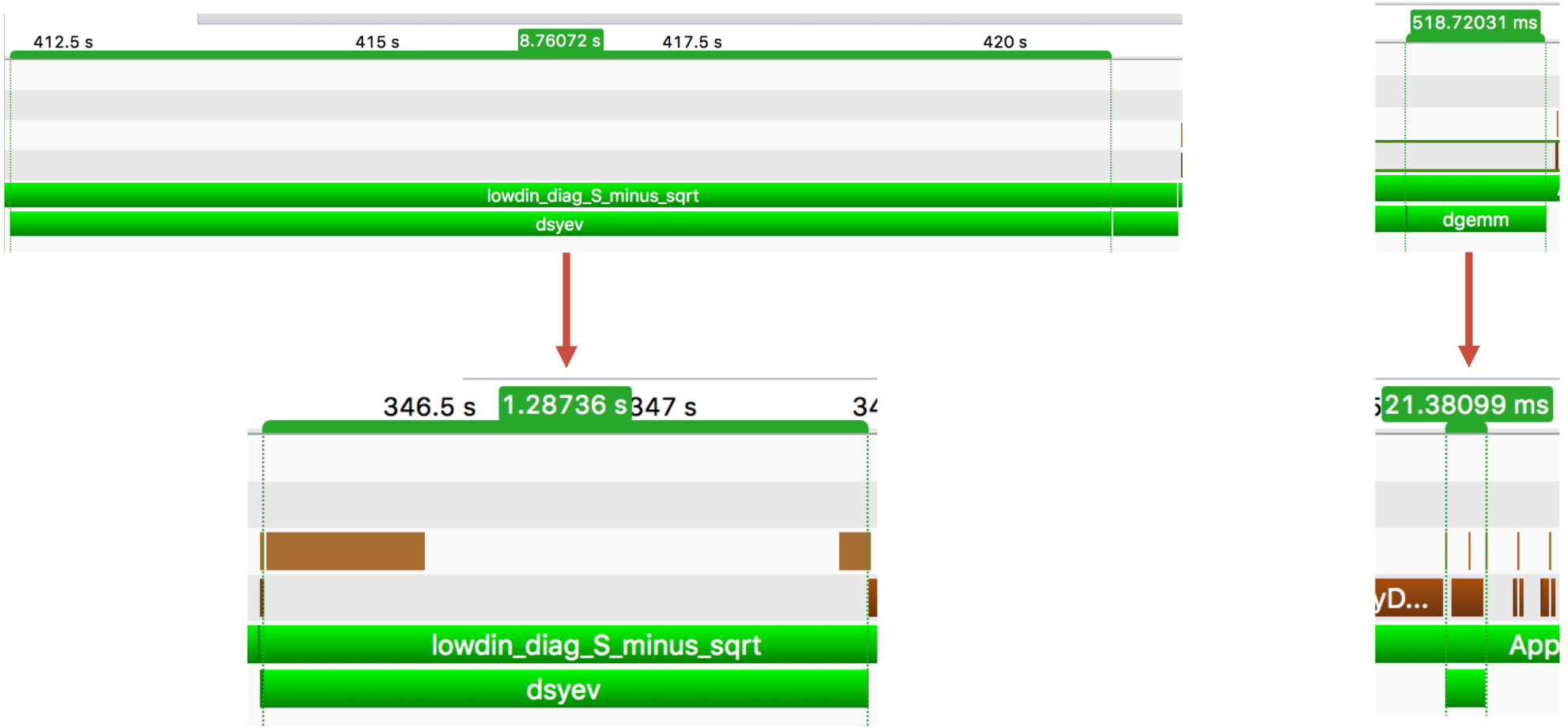
Master:



~35-40% of master rank's time is spent in these dsyev and dgemm calls.

As a first approach, port dgemm and dsyev calls to GPU via cuBLAS and cuSOLVER. See how this affects idle communication time

Performance improvement: NVProf



- For this test case, TTS reduced by >35% by porting just two function calls to GPU-accelerated libraries.

Performance improvement: NVProf

- What about the time we spent waiting in MPI_Bcast?

	Total time (s, all nodes)	Avg. time/node (s)	Avg. time/call (s)
Before porting	4193	127.1	1.257
After porting	1383	41.9	0.415

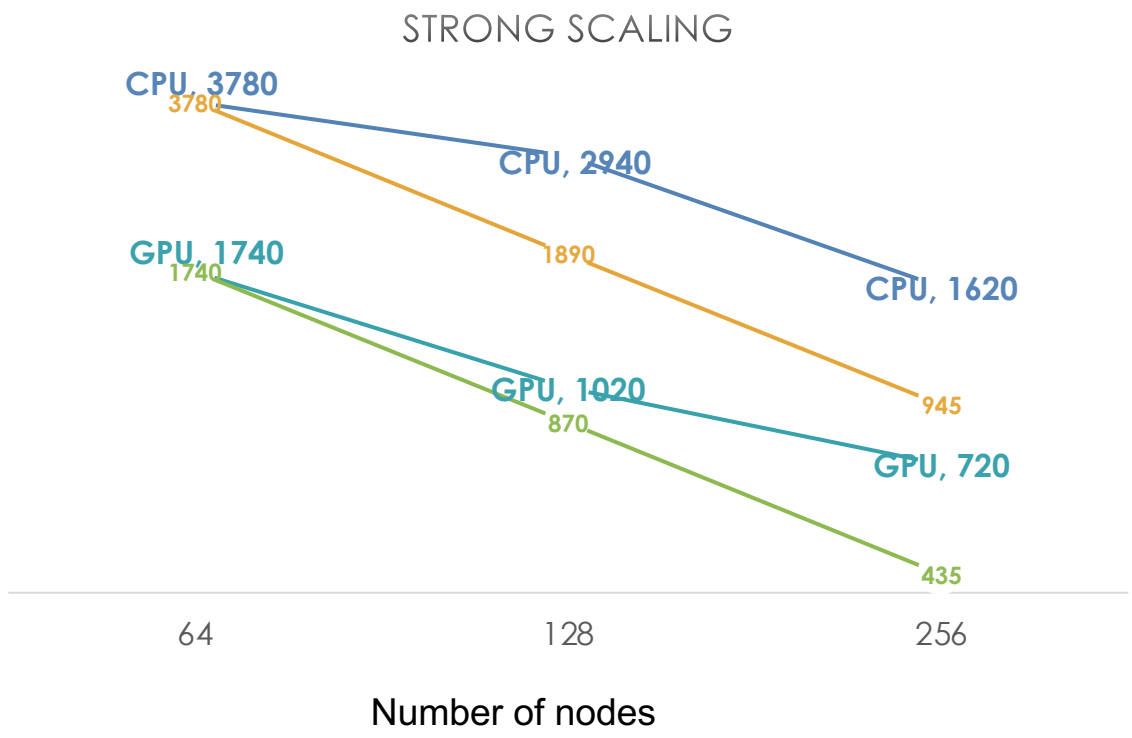
Wait time reduced by 67%!

OAK RIDGE National Laboratory

Porting experience on Summit: Results

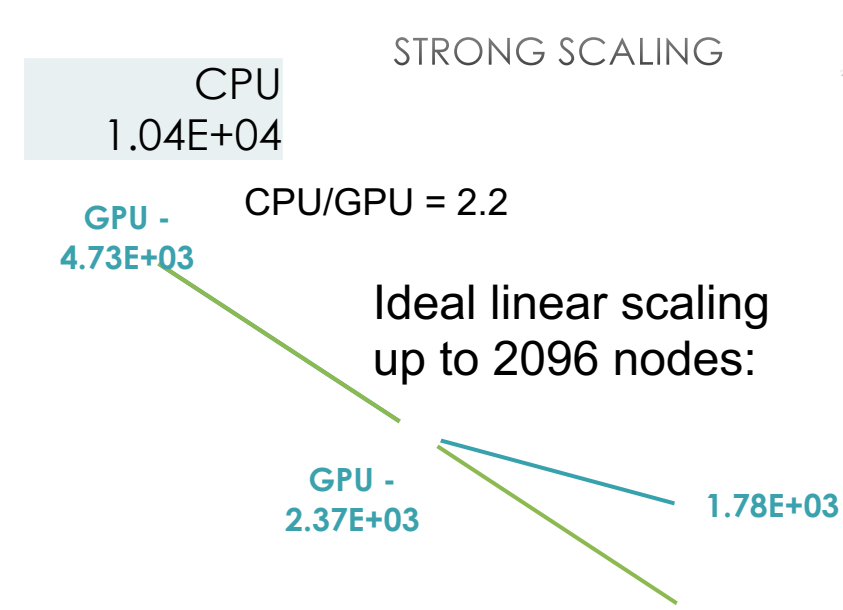
LS-Dalton Benchmark Demonstration: RI-MP2 Calculations

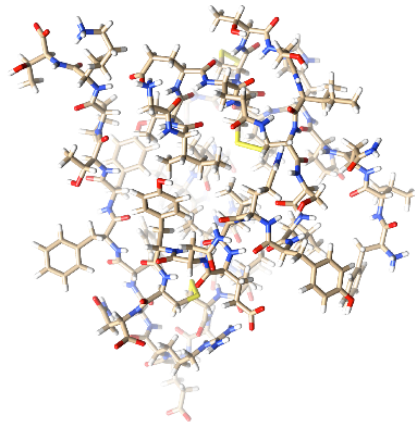
Middle-size Examples ~1500 basis functions
Scaling up to 256 nodes:



GPU speedup between 2.2 and 2.9 – without cuSOLVER

Large-size Example – Insulin: 4433 basis functions
Scaling up to 4096 using GPUs:





Not enough work to demonstrate ideal linear scaling.

Acknowledgements

- Tjerk Straatsma
- LSDalton CAAR team
  - Dmytro Bykov
  - Dmitry Lyakh
- OLCF
  - Verónica G. Melesse Vergara
- Nvidia
  - Steve Abbott
  - Jeff Larkin
- IBM
  - David Appelhans

This research used resources of the Oak Ridge Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC05-00OR22725.