Pacific Northwest
NATIONAL LABORATORY
Proudly Operated by Battelle Since 1965

A Quantum Chemistry Domain-Specific Language for Heterogeneous Clusters

ANTONINO TUMEO, ORESTE VILLA

Collaborators: *Karol Kowalski, Sriram Krishnamoorthy, Wenjing Ma, Simone Secchi*

Outline

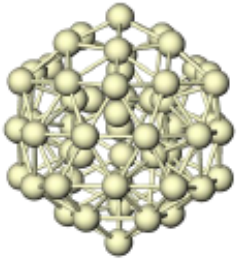
- Introduction
  - NWChem – Tensor Contraction Engine
- Tensor Contraction Generator
  - CUDA optimizations
- Experimental results

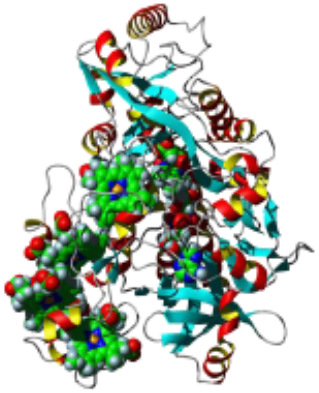
NWChem

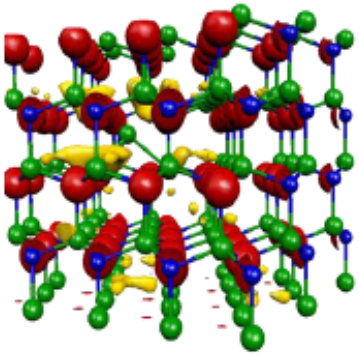
- Scalable computational chemistry tool
  - Treats large scientific computational chemistry problems
  - Exploits workstations, clusters, high performance parallel supercomputers
- Handles
  - Biomolecules, nanostructures, and solid-state
  - From quantum to classical, and all combinations
  - Gaussian basis functions or plane-waves
  - Scaling from one to thousands of processors
  - Properties and relativity
- Actively developed by a consortium of developers
  - maintained by the Environmental Molecular Science Laboratory (EMSL) located at the Pacific Northwest National Laboratory (PNNL)

NWChem TCE – Tensor Contraction Engine

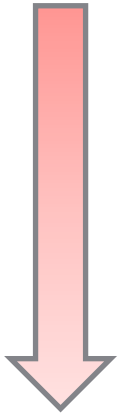
- Quantum Chemistry Coupled Cluster theory is used to understand fundamental optical processes in solar cells, photosynthesis, and other optically active materials.
- ORNL's Jaguar achieve 1.31 PFlops (over 50% of peak) on 225,000 processors during CCSD calculation
  - ***ORNL –PNNL Collaboration Lead Edo Apra*** (Gordon Bell Finalist at SC 2009)



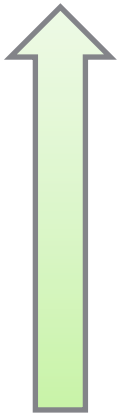




Anatomy of NWChem-TCE



Phase	Complexity
Hartree–Fock	N^4
4-index transformation	N^5
CCSD - Iterative	no^2 - nu^4
CCSD(T) – Not iterative	no^3 - nu^4



no=number of occupied; nu=number of unoccupied orbitals; N=no+nu

- Number of electron (for the closed shell systems N_el=2*no). N basis functions are used (usually defined as a set of atomic orbitals localized on the atoms of your system) N=no+nu
- 4-index transformation transforms 2-electron atomic integrals to 2-electron molecular integrals. The molecular orbitals are obtained in Hartree Fock calculations and are expressed as a linear combinations of atomic orbitals

Coupled cluster

- Coupled Cluster involves the solution of the time independent Schrodinger equation

$$\hat{H}\ \mathrm{e}^{\hat{T}}|\Psi_0\rangle \quad = \quad E\ \mathrm{e}^{\hat{T}}|\Psi_0\rangle$$

$$\hat{T} \quad = \quad \hat{T}_1 + \hat{T}_2 + \hat{T}_3 + \cdots + \hat{T}_N = \sum_p^N \hat{T}_p$$

- CCSD(T) consists in calculating the Cluster operator T up to the third contribution
  - Coupled Cluster Single Double (with Triple correction)
- $T_1 = 0(n^{1 * 2 + 1})$; $T_2 = 0(n^{2 * 2 + 1})$; $T_3 = 0(n^{3 * 2 + 1})$

Tensor Contractions


```
R[a,b,c,i,j,k] -= T[l,k,c,b]*V[l,a,i,j]
```

- Tensor contractions are generalized multidimensional matrix multiplication operations that widely occur in quantum chemistry.
- The tensor contraction involves computing tensor R from tensors T and V.
- The index ***l*** is common to both input tensors, corresponding to the common dimension in matrix-matrix multiplication, and is referred to as the contracted index.

- CCSD(T) consists of 18 such contraction expressions - N^7
- CCSD has 100 types but they are N^5 (types refers to different index permutations) – we can't do it by hand!!

TCG - Tensor Contraction Generator

- Builds on NWChem TCE
  - Developed in Python
  - Tiling: the whole spinorobital domain is partitioned into tiles which contain several spinorbitals of the same spatial- and spin-symmetry.
  - Domain Specific Language: takes in input the mathematical description of the TCs
  - Limited hints for enhancing code generation
- Generates code for:
  - CUDA, OpenCL, OpenMP, OpenACC
  - NWChem TCE is targeted to Fortran
- Optimizes for the different targets
- Currently, mostly focused on CUDA

Basic CUDA kernel

- Memory management
  - Reuses memory allocated for previous kernel calls
- Kernel Arguments
  - common strides offsets pre-computed by the host
- Encoding Thread-Block specific arguments
  - Using CUDA block dimensions to map index dimensions
- Computation in a Thread-Block
  - One thread computes one element in the output array.
  - The threads in each thread block co-operate in moving data between the GPU memory and shared memory
  - Default thread Block size 16x16 = 256 Threads

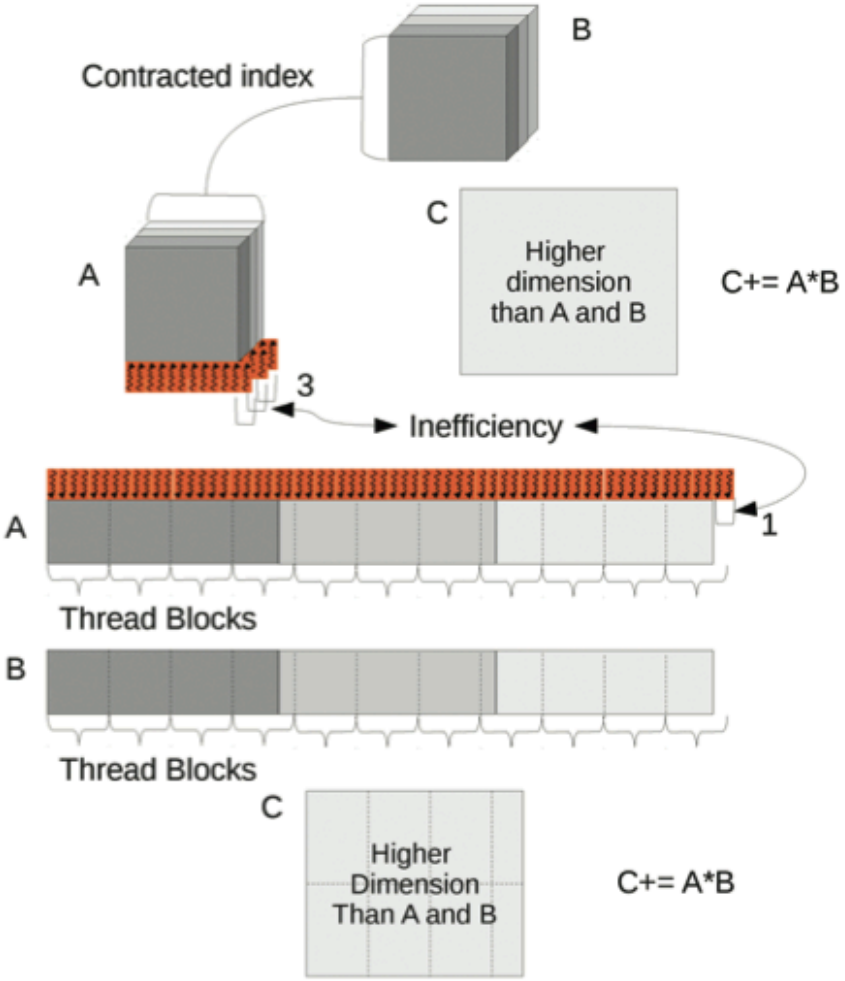
GPU code generation challenges

- Large tensors + memory requirements of the application, results in each dimension being relatively small
  - interferes with achieving good locality in SM
  - incurs high index computation overhead (modulo operations)
  - poor device thread block utilization on GPUs
- In general, tensor contraction is harder to efficiently compute than square matrix-matrix multiplication often employed in benchmark studies
- Problem sizes are not known until runtime
- CCSD(T) has 18 contraction types

CUDA optimizations

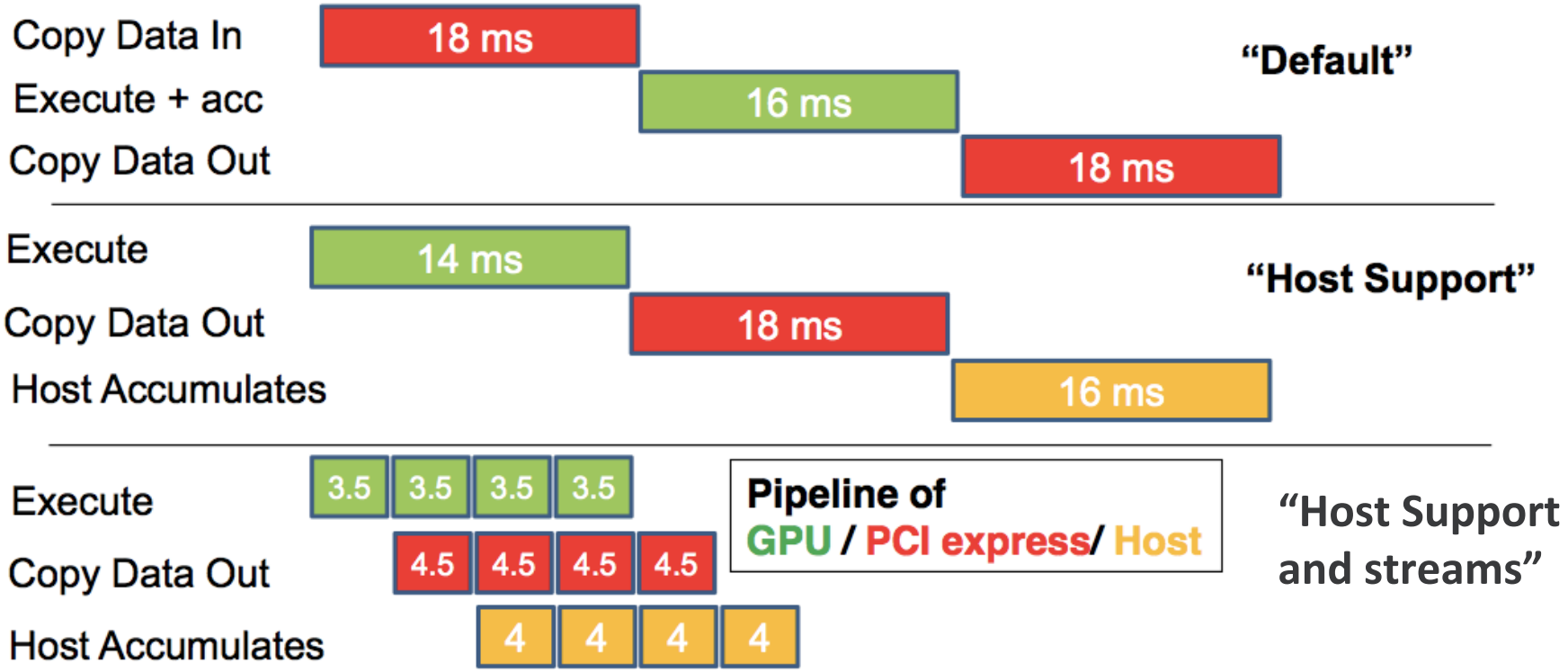
- Index Combining (General optimization)
  - When a pair of indices occur in the same order in every occurrence, they can be replaced by one index whose size is their product
    - reduction of index calculations
- Dimension Flattening (increase utilization in specific tile sizes)
  - Baseline approach works well only when index dimensions are multiple of the thread block configuration. We flatten the loops and we recreate them of the “right” size, with some index operations
    - increase of index operations but better utilization
- Pipelining on the outer dimensions (Streaming) with Host Support for accumulation
  - Most effective optimization to hide PCI express transfers

Dimension flattening



Pipelined execution

- We can avoid the O(N^6) copy IN and just have the copy OUT and then accumulate on the host
- We can create “streams” of kernels using one loop dimension and asynchronously copy out partial data



Fermi optimizations (I)

- Fermi significantly improves computation rate with respect to memory bandwidth
  - Execution is bounded by data transfers
- Intermediates are kept in memory, across tensor contractions
  - It is necessary to transfer the inputs into the GPU
  - Intermediate is computed and translated into a scalar contribution, which is transferred back to the host memory

- Fermi includes a wider register file and larger shared memory
- Register tiling: each thread contributes to 16 output elements
  - contributions are stored in 16 registers and finally written back to GPU memory

Fermi optimizations (II)

- Register tiling results in each thread performing more computation
  - Enables scalar optimizations across calculations for the 16 output elements
- Each write to the GPU memory is enclosed in a boundary check to ensure that a thread has a valid contribution to make
  - When tile executed is smaller than the thread block size
  - Condition checks are coalesced

Hybrid execution – task based framework

- A CPU core drives a GPU
  - Accumulation is done on the CPU core
- Nodes have more CPU cores than GPUs
- We exploit a task based framework
  - Exploits the Global Arrays toolkit (PGAS)
  - Load balance at the level of the node
  - Load balance at the level of the cluster
- Free cores execute a sequential version of the TC
  - We expect to be able to use OpenMP (or OpenCL for CPUs) version in the next few months

Experimental setup

- 60 nodes cluster named “Barracuda” hosted at PNNL/EMSL
- Each node of “Barracuda” consists of two quad-core Intel Xeon X5560 CPUs with 8MB L2 cache running at 2.80GHz.
- A Tesla S1070 box, which has 4 Tesla 10 GPUs, is shared between two compute nodes, resulting in two GPUs being available per SMP node (tot 128 GPUs)
- Each SMP node is interconnected in the cluster with an Infiniband QDR card.

Single node performance – single tile

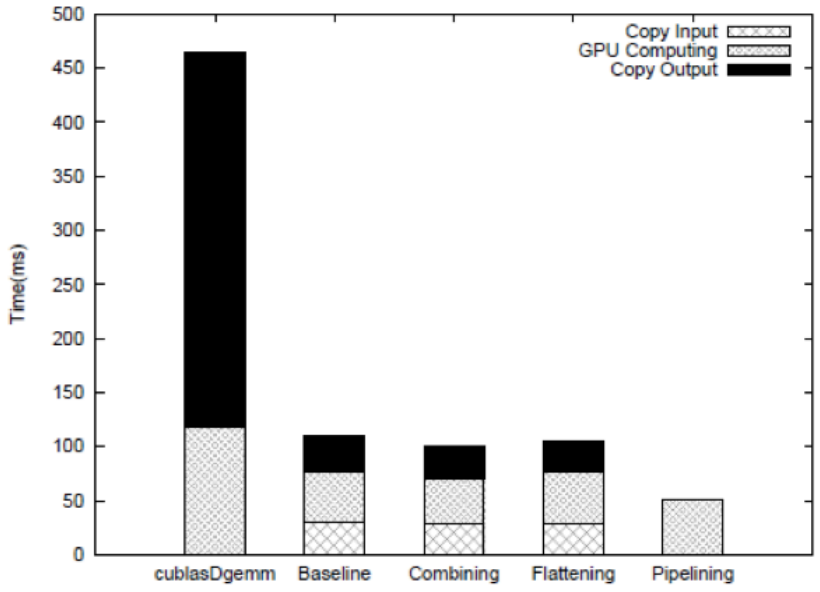


Fig. 1. Sequential benchmark with problem sizes that matches the GPU thread block configuration (ad=bd=cd=id=jd=kd=ld=16)

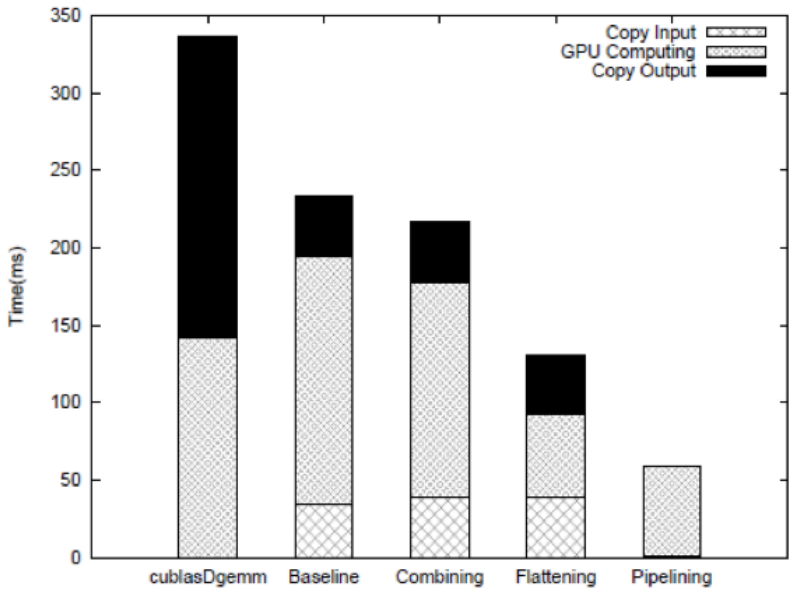


Fig. 2. Sequential benchmark with problem mismatched with GPU thread block configuration (ad=bd=cd=17, id=jd=kd=16)

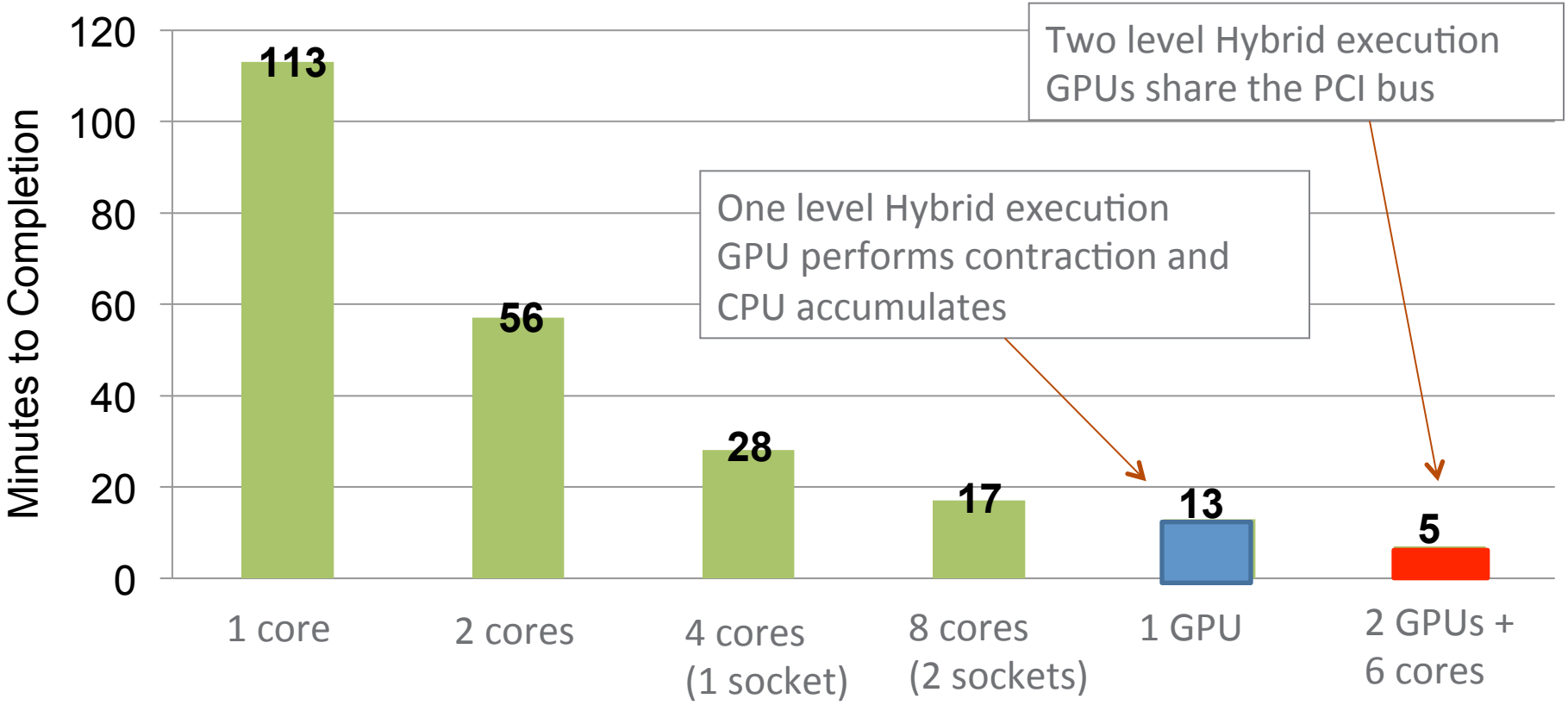
6 dimensions of size 16

3 dimensions of size 16
3 dimensions of size 17

cublasDgemm: (first approach used in the past) In NWChem, each tensor contraction is translated into index permutation and dgemm operations.

Results for 60 nodes GPU cluster (millions of tiles)

Pacific Northwest NATIONAL LABORATORY
Proudly Operated by Battelle Since 1965



Double precision calculations for green fluorescent protein (GFP) with 284 and 476 basis functions. In all calculations core electrons were not correlated.

Under development

- Code generators for other targets
  - OpenMP, OpenCL and OpenACC
  - Mostly working, not yet optimized
- Many parameters to optimize for
  - Trying to distill the most relevant
- We are thinking about auto-tuning mechanisms
  - Generate all the possible code permutations
  - Choose the best code and processing units depending on TC sizes
  - Evolutionary methods for the design space exploration?

Conclusion

- NWChem TCE
  - Domain specific language for generating Tensor Contractions
- New, re-targetable code generator
  - Generates optimized code for CUDA
  - Initial support for: OpenMP, OpenCL, OpenACC
- Task based, cluster wide load balancing
  - Based on Global Arrays (PGAS)
  - Supports Hybrid Execution
  - Independent from NWChem TCE

Thank you for your attention!

Pacific Northwest
NATIONAL LABORATORY
Proudly Operated by Battelle Since 1965

- Questions?