IWOCL 2024

The 12th International Workshop on OpenCL and SYCL

Experience of Porting LAMMPS Application with KOKKOS/SYCL to Aurora

Yasaman Ghadar, Argonne Leadership Computing Facility

Christopher Knight (ANL), Stan Moore (SNL), Daniel Arndt (ORNL)
Acknowledgment: Renzo Bustamante, Mike Brown

APRIL 8-11, 2024 | CHICAGO, USA | IWOCL.ORG

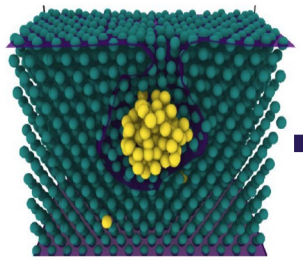
This Work Was Part of EXAALT ECP Project.

- ECP[1] EXAALT project seeks to extend accuracy, length and time scales of material science simulations for fission/fusion reactors using LAMMPS
  - Task management layer to create MD tasks, manage task queues, and store results in databases
  - Long-time, high-accuracy MD simulations with DFTB method
  - Long-time, large-scale MD simulations with machine learned SNAP potential
- Programming models:
  - ParSplice: C++
  - LAMMPS: C/C++, OpenMP, GPU-enabled (Kokkos, CUDA, OpenCL, ROCm)
  - LATTE: F90, OpenMP, GPU-enabled (CUDA)
- EXAALT wants to run millions of small MD replicas (1K to 1M atoms) via ParSplice as fast as possible (not one large simulation with billions of atoms)
- Primary KPP target is MD of nuclear fusion materials that uses the SNAP
  - *(Spectral Neighbor Analysis Potential)* interatomic potential in LAMMPS
- **Performance directly depends on single-node performance for SNAP**

[1]ECP: Exascale Computing Project

LAMMPS and SNAP Potential

- LAMMPS is a classical molecular dynamics code with a focus on materials modeling. It's an acronym for Large-scale Atomic/Molecular Massively Parallel Simulator.
- In the most common version, the trajectories of atoms and molecules are determined by numerically solving Newton's equations of motion for a system of interacting particles, where forces between the particles and their potential energies are often calculated using interatomic potentials or molecular mechanical force fields. (Wiki)
- SNAP: Spectral Neighbor Analysis Potential
  - —Total energy composed as sum of energies of individual atoms.
  - —Potential energy of each atom is sum of weighted bi-spectrum (descriptor) components.



Geometric descriptors of atomic environments

Energy as a function of geometric descriptors

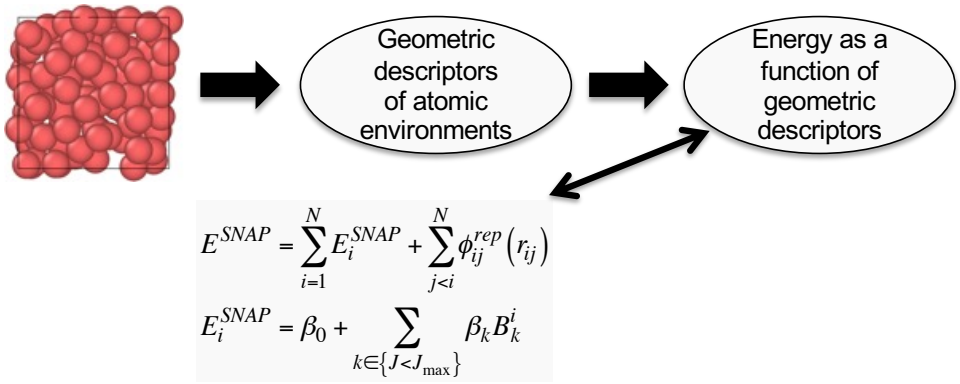
$$E^{SNAP} = \sum_{i=1}^{N} E + \sum_{j<i}^{N} \emptyset_{ij}^{rep}(r_{ij})$$

$$E_i^{SNAP} = \beta_0 + \sum_{k \in \{J<J_{max}\}} \beta_k \beta_k^i$$

SNAP: Spectral Neighbor Analysis Potential

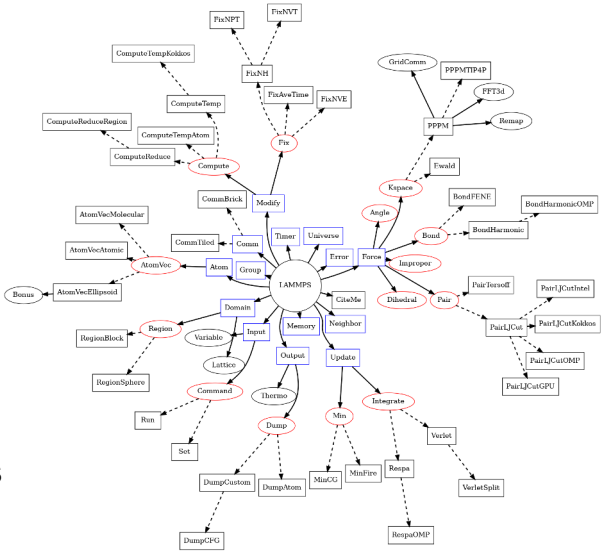
- **SNAP (Spectral Neighbor Analysis Potential):**

  SNAP approach uses Gaussian Approximation Potential neighbor bispectrum, but replaces Gaussian process with **linear regression**.
  - More robust
  - Lower computational cost (training and predicting)
  - Decouples MD speed from training set size
  - Enables large training data sets, more bispectrum coefficients
  - Straightforward sensitivity analysis
  - Fast



Bartok, Csanyi et al., *Phys. Rev. Lett, 2010*

Status of LAMMPS on Aurora

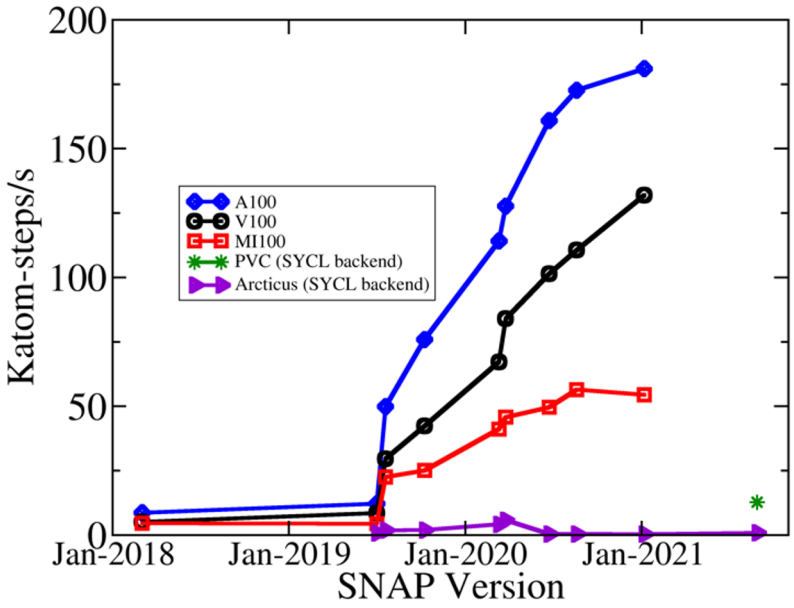


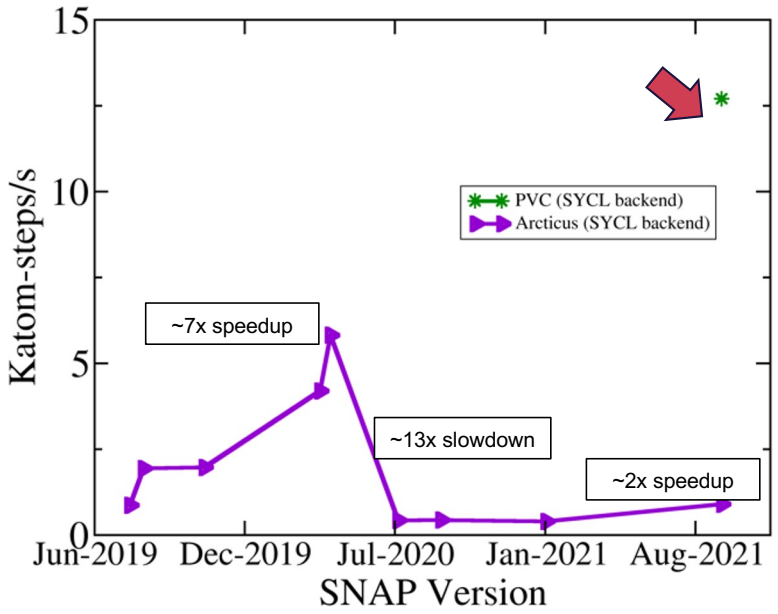
- LAMMPS supports GPU acceleration via two separate packages
  - KOKKOS
    - Kokkos via multiple backends; SYCL is primary on Aurora
    - Likely to use 1 MPI rank per PVC tile
  - GPU
    - Abstraction for multiple backends (CUDA, HIP, OpenCL); OpenCL is
    - Primarily pair calculation offloaded with everything else on host
    - Likely to use 8-16 MPI ranks per PVC tile

- LAMMPS can be built with ~90 optional packages
  - KOKKOS and GPU support for subset
  - Able to build without issues (except for needing to add MKL 1D-FFT support for KSPACE; WIP)

- To install LAMMPS on Aurora
  - $ git clone https://github.com/lammps/lammps.git
  - $ cd lammps/src
  - $ make aurora_kokkos -j 32

LAMMPS/SNAP Was Actively Optimized on NVIDIA & AMD

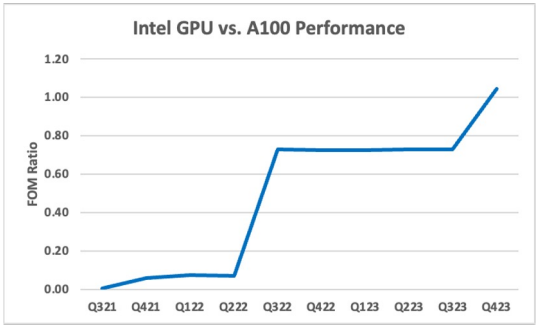




~25x speedup on V100 over 3 years

Data obtained by Stan Moore and Rahul Gayatri

SNAP Figure of Merit on Aurora

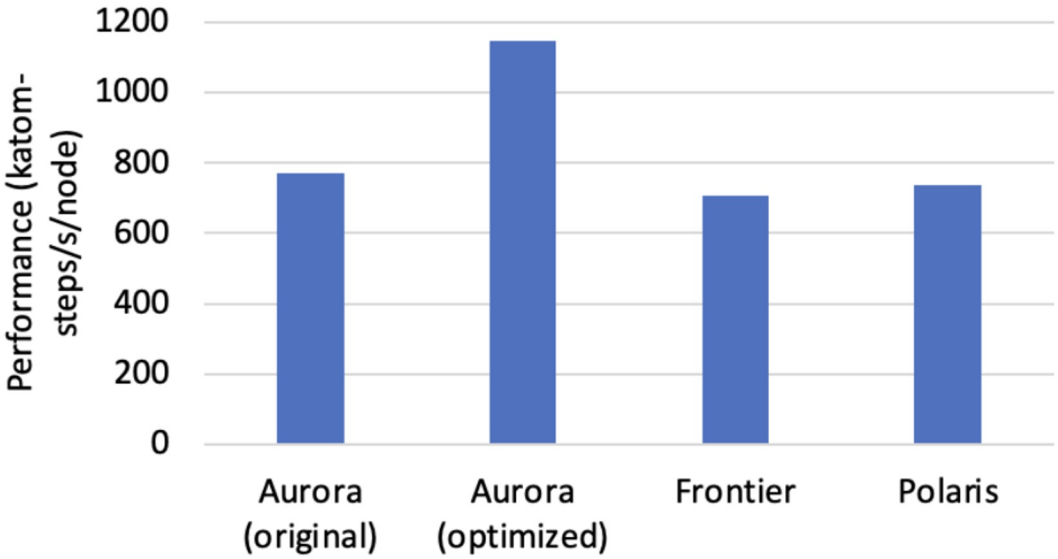


FOM= (# Steps * Atoms) / Time to Solution

Sapphire CPU 56 MPI Ranks FOM	Polaris 1 A100 GPU FOM	Aurora 1 GPU / 2 Tiles Q323 FOM	Aurora 1 GPU / 2 Tiles Q324 FOM
33,340	**190,809**	140,077	**199,240**

- First time SNAP on Aurora PVC GPU (2 tiles) measured as faster than A100: 1.04x
  - Now competitive with A100 and MI-250x, and still more opportunity
- Workload running 2K particles per GPU or tile (e.g. 4K particles per PVC GPU)
  - ~10% FOM increase on PVC with 32K particles per tile
  - ~6% FOM increase on A100 with 32K particles per tile
- Today's CPU FOM used a new addition to INTEL package (~3x faster)

Single-Node SNAP Performance

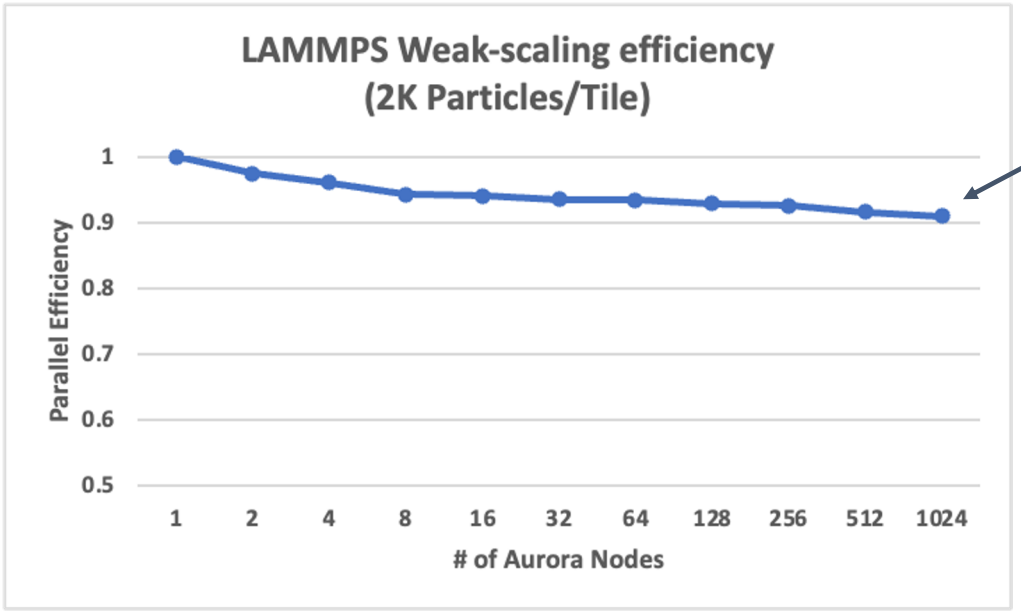


- Workload is 2K particles per MPI Rank and GPU/GCD/tile on 2 nodes
  - —24 replicas on Aurora
    i.e. 12 tiles * 2 nodes
  - —16 replicas on Frontier
  - —8 replicas on Polaris

- Aurora (original) refers to source code available from LAMMPS Github repository and same code was run on Frontier and Polaris
  - —PVC optimizations & tuning had minor impact on A100 performance*

Image borrowed from Stan Moore and ECP CoPA FY24Q1 report

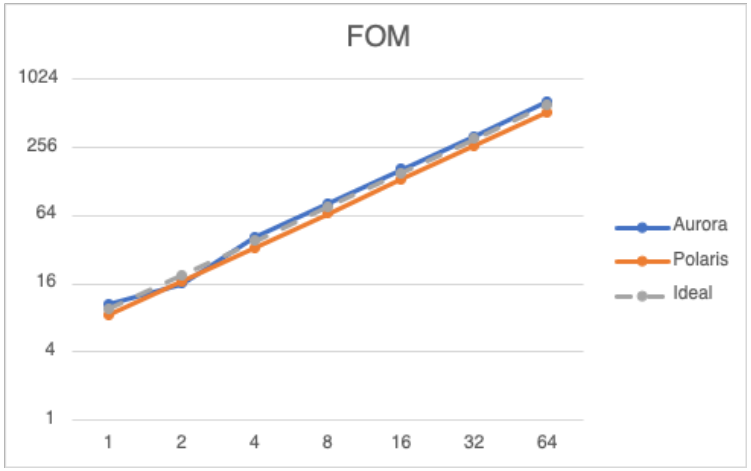
Current State of SNAP on Aurora

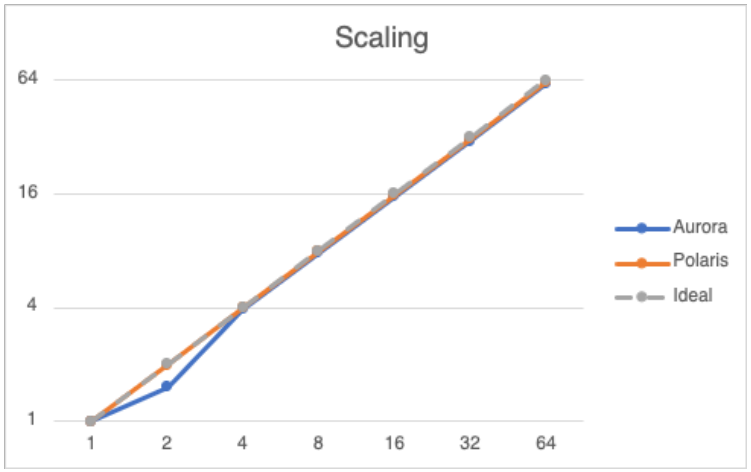


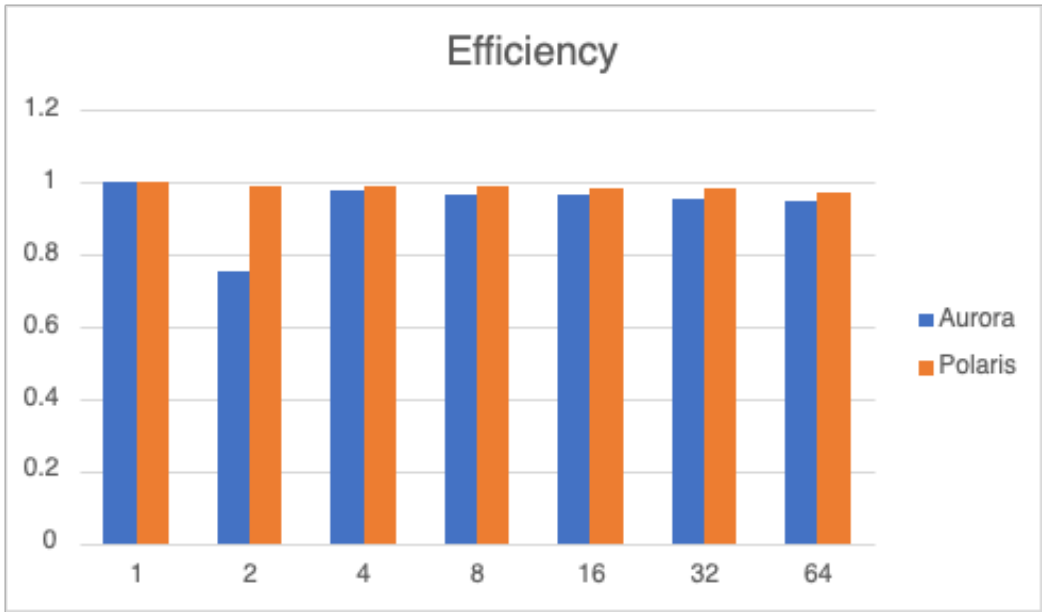
Good sign considering 2K/tile is a small workload, but SNAP is much more expensive than other models in LAMMPS.

- Success up to 1024 nodes on Aurora

SNAP Performance is Independent of the System Size!







2J8 system with 16,000 particles per tile (32,000 per GPU)

SNAP Bottleneck Kernels

- Using variety of tools such as iprof, VTUNE, NVIDIA NSIGHT, … 3 bottleneck kernels were identified on LAMMPS/SNAP
- **Compute Yi**
  - The Clebsch-Gordon products for each atom are calculated
- **Compute Fused DeiDrj**
  - The force vector for each (atom, neighbor) pair is computed
- **Compute Ui**
  - The compute_U routine calculates the expansion coefficients for each (atom, neighbor) pair

Time to Solution for Bottleneck Kernels

2023

Kernel Name (timings in milliseconds/call)	A100	PVC Orig - SG16	PVC Orig - SG32	PVC June - SG32
TagPairSNAP**ComputeYi**	9.00	15.77	14.6	12.38
TagPairSNAP**ComputeUi**Small	0.52	1.33	1.61	1.61
TagPairSNAP**ComputeFusedDeidrj**<0>	1.05	3.14	4.63	4.49
TagPairSNAPComputeFusedDeidrj<1>	1.06	3.07	4.45	4.46
TagPairSNAPComputeFusedDeidrj<2>	1.06	2.88	4.45	4.46
TagPairSNAP**ComputeZi**	8.93	14.29	13.43	11.60
TagPairSNAPComputeYiWithZlist	1.31	2.96	2.02	1.88

- SG refers to sub_group size and it is also related to Vector length in LAMMPS
- PVC Supports two types of sub_group size (think SIMD) 16 and 32
- Kernels were 1.5-4x slower on 1-tile PVC compared to A100

Time to Solution for Bottleneck Kernels

Kernel Name (timings in milliseconds/call)	A100	PVC Orig - SG16	PVC Orig - SG32
TagPairSNAPComputeYi*	9.00	15.77	14.6
TagPairSNAPComputeUiSmall*	0.52	1.33	1.61
TagPairSNAPComputeFusedDeidrj<0>*	1.05	3.14	4.63
TagPairSNAPComputeFusedDeidrj<1>*	1.06	3.07	4.45
TagPairSNAPComputeFusedDeidrj<2>*	1.06	2.88	4.45
TagPairSNAPComputeZi	8.93	14.29	13.43
TagPairSNAPComputeYiWithZlist	1.31	2.96	2.02

*called every step

- Kernels were 1.5-4x slower on 1-tile PVC compared to A100
  - ComputeYi was priority as 60-70% of runtime
  - ComputeZi is essentially ComputeYi plus energy calculation

ComputeYi (really evaluate_zi)

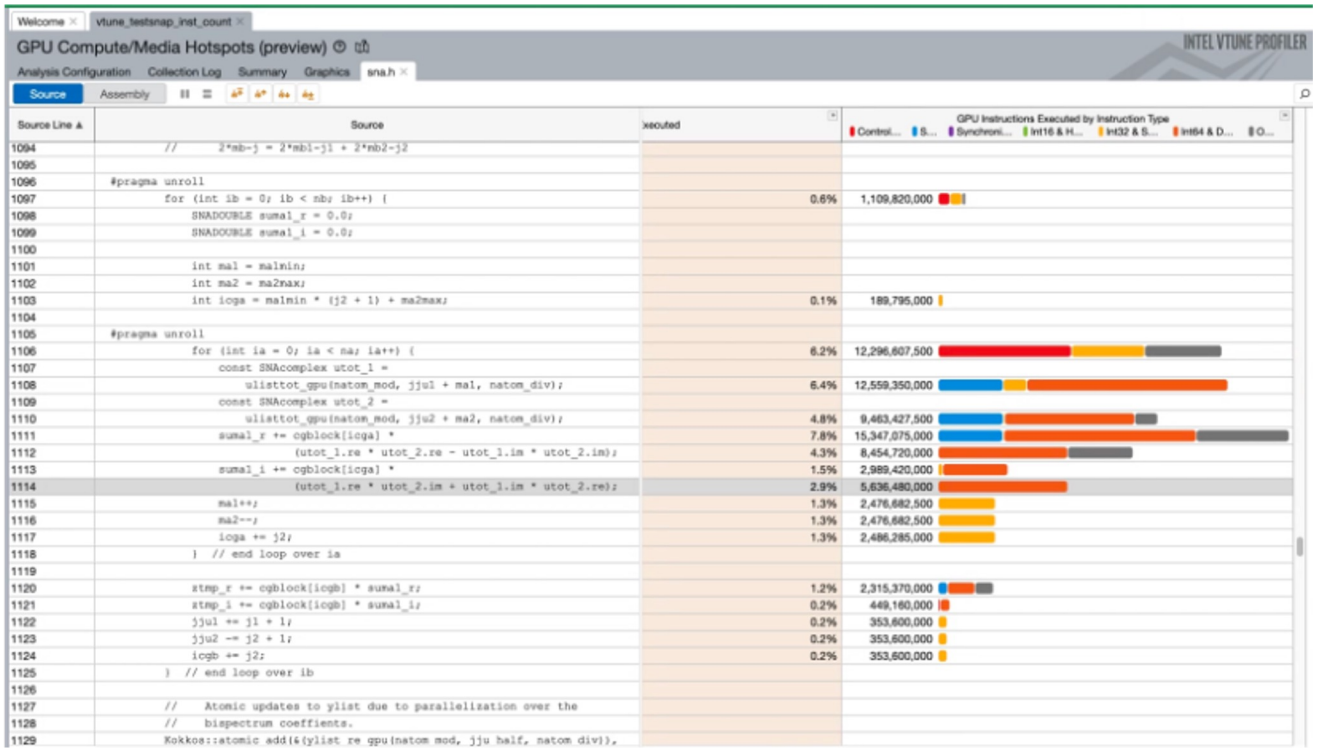
```
#ifdef LMP_KK_DEVICE_COMPILE
#pragma unroll
#endif
for (int ib = 0; ib < nb; ib++) {

  int ma1 = ma1min;
  int ma2 = ma2max;
  int icga = ma1min*(j2+1) + ma2max;

  #ifdef LMP_KK_DEVICE_COMPILE
  #pragma unroll
  #endif
  for (int ia = 0; ia < na; ia++) {
    const complex utot1 = ulisttot_pack(iatom_mod, jju1+ma1, elem1, iatom_div);
    const complex utot2 = ulisttot_pack(iatom_mod, jju2+ma2, elem2, iatom_div);
    const real_type cgcoeff_a = cgblock[icga];
    const real_type cgcoeff_b = cgblock[icgb];
    ztmp.re += cgcoeff_a * cgcoeff_b * (utot1.re * utot2.re - utot1.im * utot2.im);
    ztmp.im += cgcoeff_a * cgcoeff_b * (utot1.re * utot2.im + utot1.im * utot2.re);
    ma1++;
    ma2--;
    icga += j2;
  } // end loop over ia

  jju1 += j1 + 1;
  jju2 -= j2 + 1;
  icgb += j2;
} // end loop over ib
```

4D Kokkos View

Indexing Kokkos views (up to 4 dimensions) in SNAP is expensive on PVC and leads to an integer operation bottleneck.

Fun **with VTune**

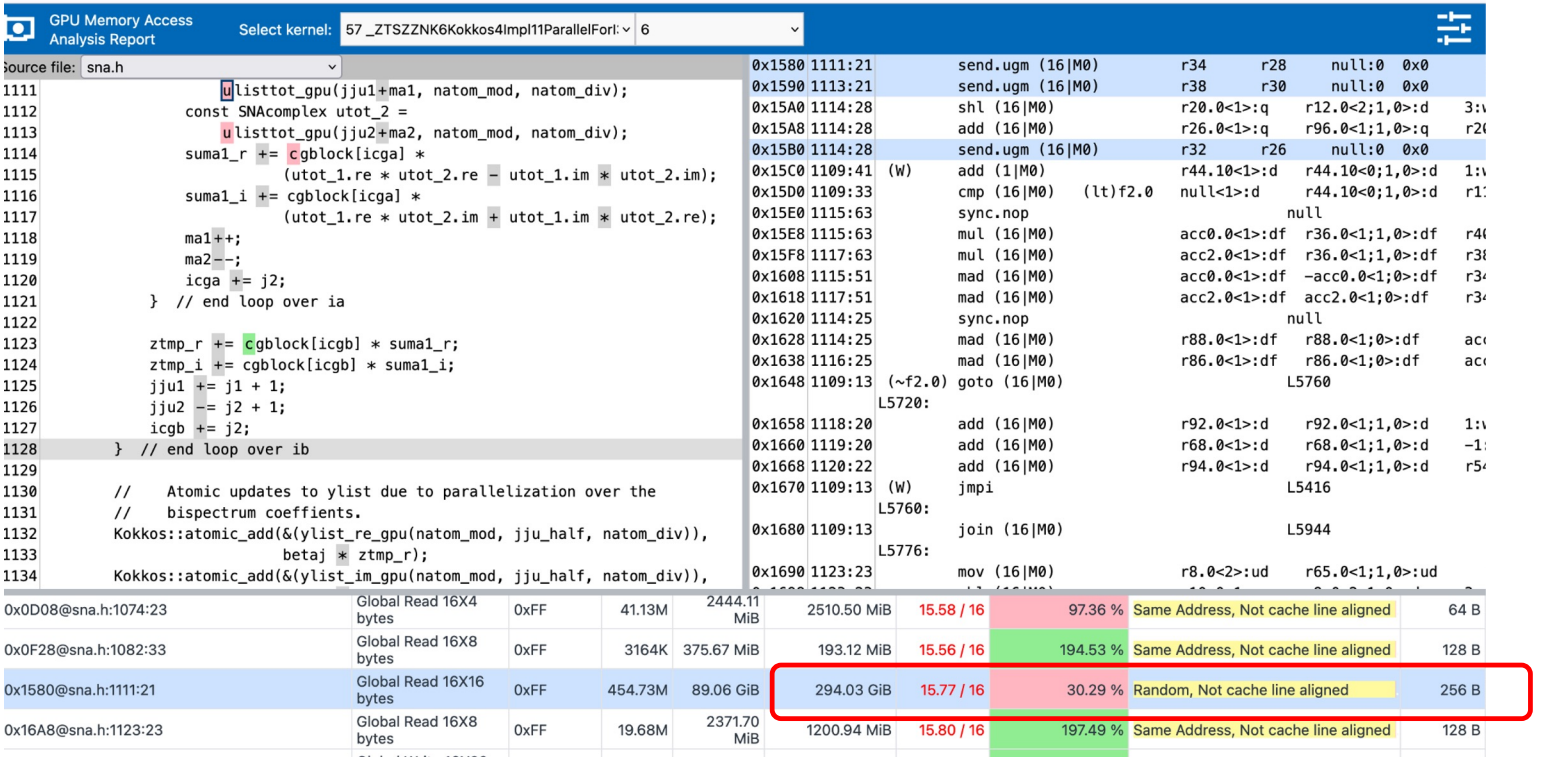
- It was no easy task to do some detailed kernel-level profiling with full LAMMPS or mini-app
  - Xiao Zhu (Intel) and JaeHyuk Kwack (ANL) were very helpful



Relative number of integer operations drew our attention, but again, it was a 4D Kokkos view…

Fun with MAAT

- MAAT is a tool for memory operation on Intel GPU's



What happens if we do the 4D index calculation ourselves?

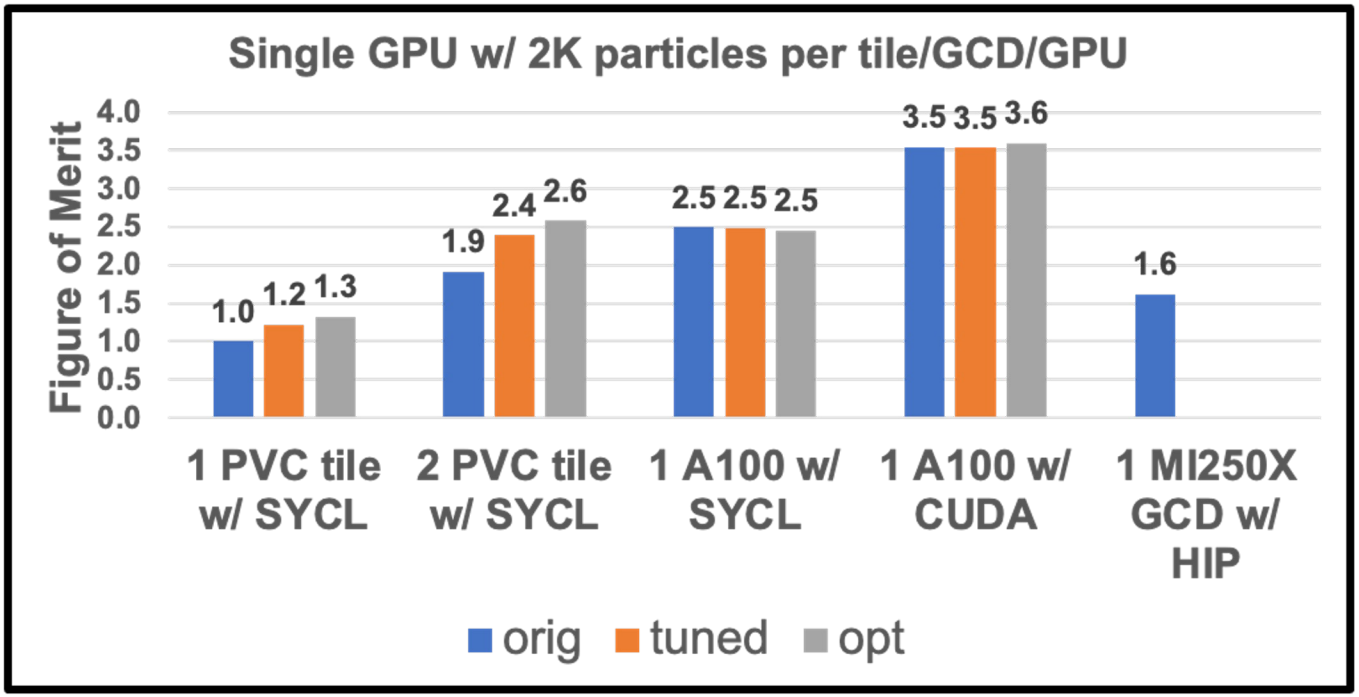
```
#ifndef KOKKOS_ENABLE_SYCL
  #ifdef LMP_KK_DEVICE_COMPILE
  #pragma unroll(8)
  #endif
#endif
  for (int ib = 0; ib < nb; ib++) {

    int ma1 = ma1min;
    int ma2 = ma2max;
    int icga = ma1min*(j2+1) + ma2max;

    // do index calculation ourselves
    const int e0 = ulisttot_pack.extent(0);
    const int e1 = ulisttot_pack.extent(1);
    const int e2 = ulisttot_pack.extent(2);
    const complex * ptr_ulisttot_pack = ulisttot_pack.data();

    const size_t indx_01 = iatom_mod + e0 * (jju1 + e1 * (elem1 + e2 * iatom_div));
    const size_t indx_02 = iatom_mod + e0 * (jju2 + e1 * (elem1 + e2 * iatom_div));

#ifndef KOKKOS_ENABLE_SYCL
    #ifdef LMP_KK_DEVICE_COMPILE
    #pragma unroll(8)
    #endif
#endif
    for (int ia = 0; ia < na; ia++) {
      const complex utot1 = ptr_ulisttot_pack[indx_01 + e0*ma1];
      const complex utot2 = ptr_ulisttot_pack[indx_02 + e0*ma2];
      const real_type cgcoeff_a = cgblock[icga];
      const real_type cgcoeff_b = cgblock[icgb];
      ztmp.re += cgcoeff_a * cgcoeff_b * (utot1.re * utot2.re - utot1.im * utot2.im);
      ztmp.im += cgcoeff_a * cgcoeff_b * (utot1.re * utot2.im + utot1.im * utot2.re);
      ma1++;
      ma2--;
      icga += j2;
    } // end loop over ia

    jju1 += j1 + 1;
    jju2 -= j2 + 1;
    icgb += j2;
  } // end loop over ib
```

Grab raw pointer to data in view

Precompute part of 4D index; trying to keep it simple

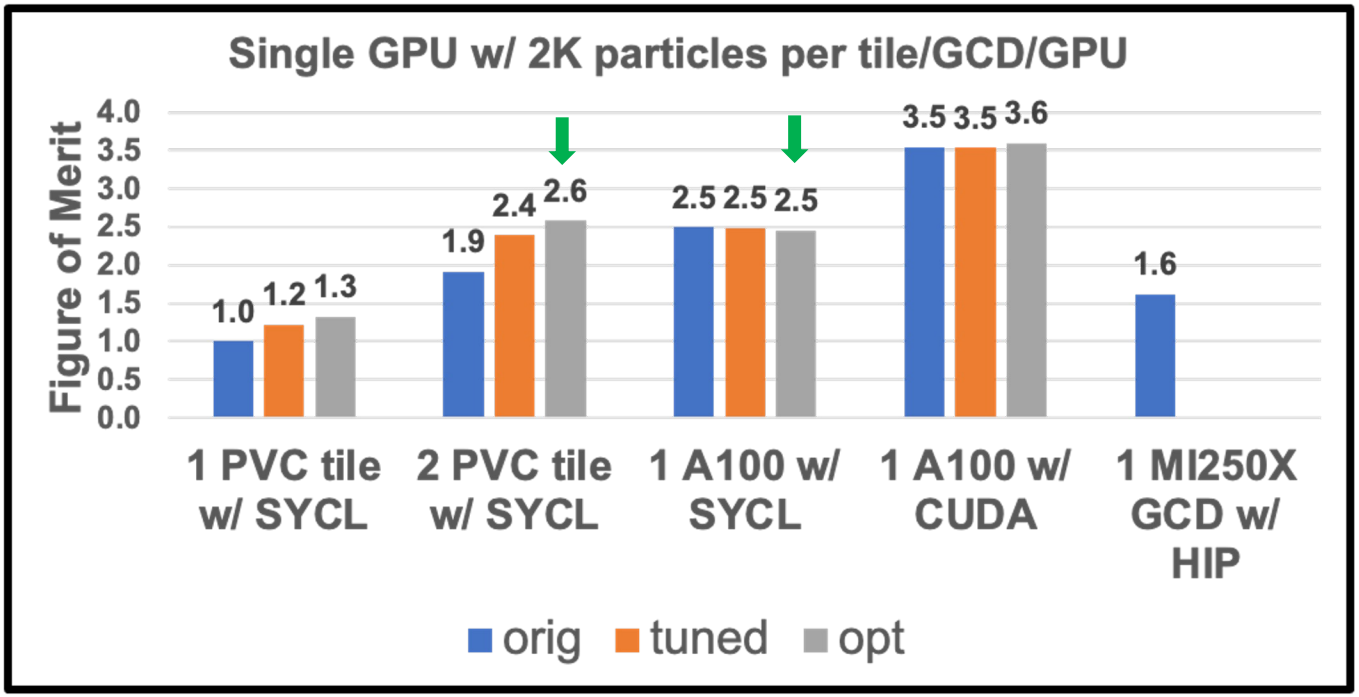
Everything else left as-is

Single GPU Performance for SNAP



- Tuned code adjusted sub-group size and team/tile sizes for various kernels
- The “optimized” index calculation resulted in 30% in overall application runtime

Single GPU Performance for SNAP



- Tuned code adjusted sub-group size and team/tile sizes for various kernels
- The “optimized” index calculation resulted in 30% in overall application runtime

Today's Time to Solution!

Kernel Name (timings in milliseconds/call)	A100	PVC Orig - SG32	PVC Oct - SG32
TagPairSNAPComputeYi	9.00	14.6	9.34
TagPairSNAPComputeUiSmall	0.52	1.61	0.82
TagPairSNAPComputeFusedDeidrj<0>	1.05	4.63	2.05
TagPairSNAPComputeFusedDeidrj<1>	1.06	4.45	2.01
TagPairSNAPComputeFusedDeidrj<2>	1.06	4.45	1.99
TagPairSNAPComputeZi	8.93	13.43	8.87
TagPairSNAPComputeYiWithZlist	1.31	2.02	1.81

- This optimization was done with help of Mike Brown

Path to Today's Optimization!

- In SYCL there are different types of pointers for different address spaces: **global** and **local**
- Some kernels in SNAP use Kokkos level 0 scratch memory (shared memory) which is in the local address space
- However currently Kokkos always returns a pointer to the global address space.
  - This requires the compiler to add additional control flow due to the presence of these generic address space operations, leading to unnecessary overhead.
  - ***As a workaround, Mike manually cast the shared memory pointers from global to local address space***
  - ***Daniel Arndt (ORNL) created an experimental Kokkos interface for using scratch memory inside kernels that allows the user to specify the scratch level at compile-time***

Path to Today's Optimization!

- Some kernels in SNAP run better with a workgroup size of 32, while others are faster with size of 16. Daniel Arndt created experimental code to allow setting workgroup sizes on a per-kernel basis in Kokkos
- One kernel had a high register count leading to register spilling, so Mike added the SYCL "use large grf" kernel property specification in this kernel to increase the size of the general register file (GRF).

```
#include <sycl/ext/intel/experimental/kernel_properties.hpp>
#define SYCL_SPECIFY_HIGH_REG_COUNT() sycl::ext::intel::experimental::set_kernel_prope
rties(sycl::ext::intel::experimental::kernel_properties::use_large_grf);
```

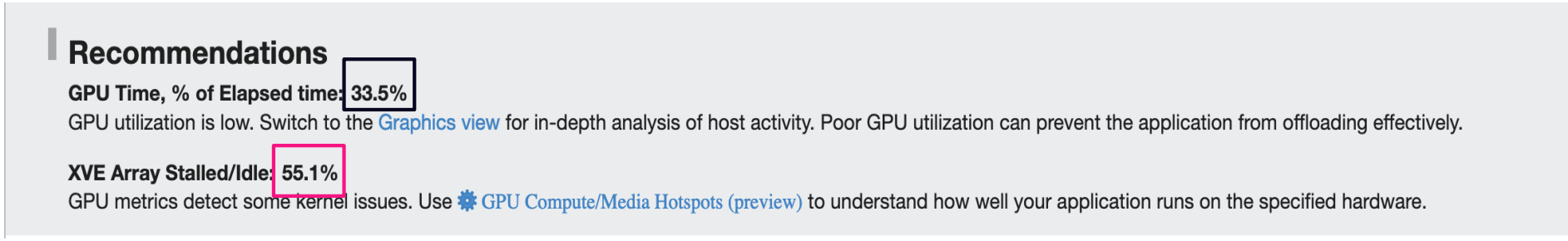
Today's Time to Solution!

Kernel Name (timings in milliseconds/call)	A100	PVC Orig - SG32	PVC Oct - SG32
TagPairSNAPComputeYi	9.00	14.6	9.34
TagPairSNAPComputeUiSmall	0.52	1.61	0.82
TagPairSNAPComputeFusedDeidrj<0>	1.05	4.63	2.05
TagPairSNAPComputeFusedDeidrj<1>	1.06	4.45	2.01
TagPairSNAPComputeFusedDeidrj<2>	1.06	4.45	1.99
TagPairSNAPComputeZi	8.93	13.43	8.87
TagPairSNAPComputeYiWithZlist	1.31	2.02	1.81

- With all optimizations to-date, 2-tile Aurora PVC is 1.04x faster than A100 for this workload
  - MAAT: cacheline utilization for ulisttot_pack increased from 30% to 117%
  - Manually casting shared memory pointer from global to local address space
    - Reduced additional control flow

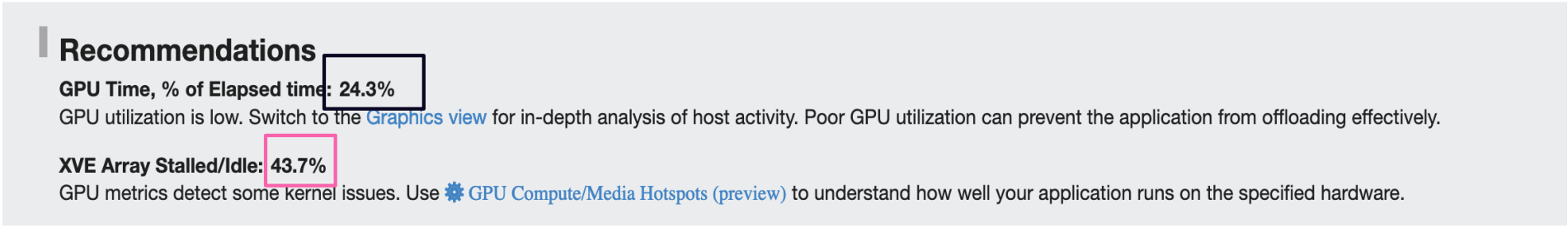
Current VTune Studies with LAMMPS (Opt vs Orig)

LAMMPS Summer 2023



Idle time has decreased significantly

LAMMPS Winter 2024



Summary of Current Optimizations

- Further optimization of 4D index calculation
- Manually casting shared memory pointer from global to local address space
  - Reduced additional control flow due to generic address space operations
  - Experimental Kokkos interface added to specify scratch level at compile-time
    - https://github.com/kokkos/kokkos/pull/5879
    - Same effect without manually casting pointers
- Tuning of workgroup size for different kernels
  - Experimental Kokkos interface to set workgroup size on per-kernel basis
  - https://github.com/kokkos/kokkos/pull/6496
- Fusing of 3 TagPairSNAPComputeFusedDeidrj<> kernels

Future Directions

- Continue pushing on performance optimizations of SNAP model
- Begin concerted effort to understand performance for other common LAMMPS workloads across GPU and KOKKOS packages

- Special thanks to
  - **Renzo Bustmante, Chris Knight, Varsha Madananth, Daniel Arndt, Stan Moore, Mike Brown**
  - **plus many special guest appearances: Xiao Zhu (VTune, MAAT), Xinmin Tian (compiler/runtime), …**

Acknowledgment

This work was done on a pre-production supercomputer with early versions of the Aurora software development kit. This research used resources of the Argonne Leadership Computing Facility, a U.S. Department of Energy (DOE) Office of Science user facility at Argonne National Laboratory and is based on research supported by the U.S. DOE Office of Science-Advanced Scientific Computing Research Program, under Contract No. DE-AC02-06CH11357. SNL is managed and operated by NTESS under DOE NNSA contract DE-NA0003525. This manuscript has been authored by UT-Battelle, LLC, under Grant DE-AC05-00OR22725 with the U.S. Department of Energy (DOE).

Thank you!