LLNL-CONF-863721-DRAFT

Appendix: Artifact Description

T. Oppelstrup, S. Moore

April 29, 2024

International Conference for High Performance Computing, Networking, Storage, and Analysis
Atlanta, GA, United States
November 17, 2024 through November 22, 2024

Appendix: Artifact Description

Artifact Description (AD)

I. Overview of Contributions and Artifacts

A. Paper's Main Contributions

- C_1 Novel mapping and dataflow EAM MD algorithm for atomistic simulations on the Cerebras Wafer-Scale Engine (WSE).
- C_2 Reference SOTA performance measurements of EAM MD on GPUs and CPUs.
- C_3 Observed performance improved by orders of magintude.

B. Computational Artifacts

- A_1 CPU reference run descriptions (in this document).
- A_2 GPU reference run descriptions (in this document).
- A_3 WSE run description. The WSE code in the Tungsten dataflow programming language is undergoing a legal review for release.

Artifact ID	Contributions Supported	Related Paper Elements
A_1, A_2	C_2	Table III Figure 2
A_3	C_1, C_3	Tables II-III Figure 2
A_3	C_1	Figure 8

II. Artifact Identification

A. Computational Artifact A_1

Relation To Contributions

This artifact is the timing data of the CPU data points in Figure 2. The best performance points for the CPU systems are also displayed in Table III.

Expected Results

These computational experiments demonstrate the performance of the LAMMPS simulation code on CPUs. The resulting timings are used to compare to corresponding timing data for the WSE CS-2 code. These CPU results are significantly slower than the corresponding simulations on the WSE CS-2 chip, and puts its speed in context to leading conventional hardware and algorithms.

Expected Reproduction Time (in Minutes)

The expected computational time for a single benchmark experiment using dual-socket nodes with Intel Xeon E5-2695v4 CPUs, is up to 7 minutes using 8 nodes. The runtime is inversely proportional to the node-count up to around 50 nodes.

Artifact Setup (incl. Inputs)

Hardware: The experiments were conducted using the Quartz cluster at LLNL. Each node has 2 Xeon E5-2695v4 18C 2.1GHz CPUs, 128GBytes of DRAM. The nodes are interconnected with Intel Omni-Path network. Quartz is further documented here:

https://www.top500.org/system/178971/.

To reproduce all data points in Figure 2, 400 nodes are required.

Software: These experiments used the LAMMPS software, available for public download from lammps.org. The LAMMPS version was "Large-scale Atomic/Molecular Massively Parallel Simulator - 8 Feb 2023", and was retrieved from git with the specific commit hash tag of

`730e5d2e64106f3e5357fd739b44c7eec19c7d2a`

of the `develop` branch. The following LAMMPS packages were enabled:

MANYBODY, MEAM, MOLECULE, RIGID.

We used the MPI communication library "MPI v3.1: MVAPICH2 Version: 2.3.7 MVAPICH2 Release date: Wed March 02 22:00:00 EST 2022". The operating system version was Linux "Red Hat Enterprise Linux 8.9 (Ootpa)" 4.18.0-513.18.1.2toss.t4.x86_64 x86_64, and the SLURM scheduler was used for job submitting and launching jobs.

Datasets / Inputs: Scripts for generating initial conditions, and for running the timing benchmarks, as well as all necessary input files to LAMMPS, are in the git repository

`github.com/xxx/yyy.git`,

in the `cpu_benchmarks` sub-directory.

Installation and Deployment: To compile LAMMPS we used the standard MPI makefile `Makefile.mpi` included with the source code, which had the compiler and linker flags set to `-g -O3 -std=c++11`.

Compiler: Intel Classic C++ 20.21.6 / Intel(R) C++ g++ 10.3.1 mode with OpenMP not enabled C++ standard: C++11

Artifact Execution

An example command to perform the Cu benchmark on 8 nodes is:

```
nnodes=8
srun -N$nnodes -n$(($nnodes*36)) \
  lmp_mpi_quartz -in in.bench \
  -var element Cu -var pbc 0 \
  -var nx 172 -var ny 192 -var nz 6 \
  -log bench.log
```

To generate all initial conditions and to run all benchmarks, follow the procedure below:

1) Clone the git repository:
   `github.com/xxx/yyy.git`.
2) Go into the sub-directory `cpu_benchmarks`.
3) Edit the variable `lmp` in the beginning of `make_max_cfgs.sh` (Sec. III-A) and

`scale_bench.sh` (Sec. IV-A) to point to the LAMMPS executable to use.

4) Edit the `srun` commands in `make_max_cfgs.sh` and `scale_bench.sh` to be the correct job running/job submission commands for your machine.
5) Adjust the `nnodes` loop in `scale_bench.sh` to run over the desired node counts.
6) Run or source the `make_max_cfgs.sh` script to produce initial conditions. Using 8 nodes (as listed on the `srun` line in the script), this process will run three jobs which each should complete in less than two minutes. The resulting configuration files can be found in `{Cu,Ta,W}/cfgs`.
7) Run or source the `scale_bench.sh` script to perform the benchmark simulations. It runs one job per element per node count. On 8 nodes such a job takes up to 7 minutes. The runtime is inversely proportional to the node-count up to around 50 nodes.
8) The runtime for the simulations can be extracted by the command:

```
grep Loop nbench.log.*
```

B. Computational Artifact A_2

Relation To Contributions

This artifact is the timing data of the GPU data points in Figure 2. The best performance points for the GPU systems are also displayed in Table III.

Expected Results

These computational experiments demonstrate the performance of the LAMMPS simulation code on GPUs. The resulting timings are used to compare to corresponding timing data for the WSE CS-2 code. These GPU results are significantly slower than the corresponding simulations on the WSE CS-2 chip, and puts its speed in context to leading conventional hardware and algorithms.

Expected Reproduction Time (in Minutes)

The expected computational time on Frontier for a single benchmark data point is less than 2 minutes in all cases. All runs combined should take less than 30 minutes.

Artifact Setup (incl. Inputs)

Hardware: The experiments were conducted using the OLCF Frontier exascale supercomputer. Each Frontier node consists of 8 AMD Instinct MI250X GPU compute dies (GCDs) attached to an AMD Optimized 3rd Generation EPYC 64C 2GHz CPU. A total of 9,408 nodes are connected together by HPE's Slingshot-11 network. Frontier is further documented here: https://www.top500.org/system/180047/.

To reproduce all GPU data points in Figure 2, 32 nodes are required.

Software: The LAMMPS version was "Large-scale Atomic/Molecular Massively Parallel Simulator - 7 Feb 2023", and was retrieved from git with the specific commit hash tag of `edcbd2e7618518e6bb1a5d843081474d7176872d` of the `develop` branch.

Datasets / Inputs: Scripts for running the timing benchmarks, as well as all necessary input files to LAMMPS, are in the git repository

`github.com/xxx/yyy.git`,

in the `gpu_benchmarks` sub-directory.

Installation and Deployment: The following modules were loaded on top of the default modules: `cray-mpich/8.1.27`, `rocm/5.7.1`.

LAMMPS was compiled using `Makefile.frontier_kokkos` included with the source code. The following LAMMPS packages were enabled:

KOKKOS, MANYBODY.

The following environment variables were set:

- `MPICH_GPU_SUPPORT_ENABLED=1`
- `MPICH_OFI_NIC_POLICY=NUMA`
- `LD_LIBRARY_PATH=`
  `${CRAY_LD_LIBRARY_PATH}:${LD_LIBRARY_PATH}`

The SLURM scheduler was used for job submitting and launching jobs.

Artifact Execution

An example command to perform the Cu benchmark on 1 GCD is:

```
srun -u -N1 -n1 -c1 \
  --cpu-bind=map_cpu:50 \
  --gpus=1 ./lmp_frontier_kokkos \
  -in in.bench -k on g 1 -sf kk \
  -pk kokkos neigh full newton off \
  -var element Cu \
  -var nx 172 -var ny 192 \
  -var nz 6 -var pbc 0 \
  -log bench.log
```

An example command to perform the Cu benchmark on 8 nodes is:

```
srun -u -N 8 --ntasks-per-node=8 \
  --cpus-per-task=6 --gpus-per-task=1 \
  --gpu-bind=closest \
  ./lmp_frontier_kokkos \
  -in in.bench -k on g 1 -sf kk \
  -pk kokkos neigh full newton off \
  -var element Cu \
  -var nx 172 -var ny 192 \
  -var nz 6 -var pbc 0 \
  -log bench.log
```

To run all benchmarks, follow the procedure below:

1) Clone the git repository
   `github.com/xxx/yyy.git`
2) Go into the sub-directory `gpu_benchmarks`
3) Update the `#SBATCH --account=xxx` command in `run_multi.sh` (sec. V-A) to use your account on Frontier
4) Update the variable `EXE` in the `run_multi.sh` script to point to your LAMMPS executable
5) Submit the `run_multi.sh` to the Frontier queue using the SLURM `sbatch` command. This process will run

several LAMMPS jobs, each of which should complete in less than two minutes.

6) The runtime for the simulations can be extracted by the command:

```
grep "Loop time" bench.log.*
```

C. Computational Artifact A_3

Relation To Contributions

This artifact is the timing data of the WSE data points in Figure 2. The performance points for the system are also displayed in Table III.

Expected Results

These computational experiments demonstrate the performance of the EAM MD implementation on WSE. The performance exceeds LAMMPS on CPU and GPU systems by a few orders of magnitude.

Expected Reproduction Time (in Minutes)

Each run takes less than 5 minutes.

Artifact Setup (incl. Inputs)

Hardware: WSE experiments were conducted on a CS-2 system equipped with 769×1044 processing elements. System clock rate is configurable and was set to 850MHz.

Software: The EAM algorithm was written in the Tungsten programming language and compiled with an internal compiler (April 2024 release).

Cerebras plans to make the compiler accessible to researchers through partnerships with PSC, EPCC, and participation in the NAIRR Pilot program.

Datasets / Inputs: The datasets from the LAMMPS runs are used for the WSE runs.

III. Configuration generation

A. *make_max_cfgs.sh*

```
lmp="./lmp_mpi_quartz"

## This script generates max size configurtaions that fits on the
## CS-2 machine we run on for this project.


# Max Ta,W shapes fitting 'std size CS-2': "186 248 8" "249 248 6"
for shapes in "256 261 6" ; do
    nx=`echo $shapes | awk '{print $1}'`
    ny=`echo $shapes | awk '{print $2}'`
    nz=`echo $shapes | awk '{print $3}'`
    for elem in Ta W ; do
for bc in 0 ; do
    echo "Submitting $nx x $ny x $nz, $elem bc=$bc"
    srun -N8 -n288 -t10 $lmp -in in.eq \
     -var element $elem \
     -var nx $nx -var ny $ny -var nz $nz -var pbc $bc \
     -log xlog.$nx.$ny.$nz.$elem.$bc > xrun.$nx.$ny.$nz.$elem.$bc  &
done
    done
done

for shapes in "174 192 6" ; do
    nx=`echo $shapes | awk '{print $1}'`
    ny=`echo $shapes | awk '{print $2}'`
    nz=`echo $shapes | awk '{print $3}'`
    for elem in Cu ; do
for bc in 0 ; do
    echo "Submitting $nx x $ny x $nz, $elem bc=$bc"
    srun -N8 -n288 -t10 $lmp -in in.eq \
     -var element $elem \
     -var nx $nx -var ny $ny -var nz $nz -var pbc $bc \
     -log xlog.$nx.$ny.$nz.$elem.$bc > xrun.$nx.$ny.$nz.$elem.$bc  &
done
    done
done
```

B. *in.eq*

```
print "Simulation of ${element} with ${nx}x${ny}x${nz} unit cells"
include ${element}/${element}.inc

units metal
atom_style atomic
atom_modify map hash

lattice ${latttype} $(v_alatt*v_thexp) # Effective lattice constant

region box block 0 ${nx} 0 ${ny} 0 ${nz}

create_box 1 box
create_atoms 1 box
mass 1 ${M}
```

```
if "${pbc}==1" then "jump SELF pbc_section"
if "${pbc}==2" then "jump SELF z_bc_section"
label open_bc_section
  displace_atoms all move 0.25 0.25 0.25
  change_box all boundary s s s
  variable bc string open
  jump SELF bc_section_end

label z_bc_section
  displace_atoms all move 0.25 0.25 0.0
  change_box all boundary s s p
  variable bc string openxy
  jump SELF bc_section_end

label pbc_section
  variable bc string pbc

label bc_section_end


velocity all create $(2*v_T) 87287 loop geom # 2*T to compensate for equipartitioning
velocity all zero linear
velocity all zero angular

if "${alloytype}==1" then "jump SELF alloy_section"
label eam_section
  pair_style eam
  pair_coeff * * ${element}/${potfile}
  jump SELF potential_end
label alloy_section
  pair_style eam/alloy
  pair_coeff * * ${element}/${potfile} ${element}
label potential_end

print "Input file mass = ${M}, after potential init, atom mass = $(mass[1])"

neighbor 0.5 bin
neigh_modify every 1 delay 0 check yes

fix 1 all nve

compute msd all msd
fix avgs all ave/time 1 1000 1000 c_thermo_press c_thermo_temp

thermo_style custom step f_avgs[2] pe etotal f_avgs[1] c_msd[1]

timestep 0.002  ## 2 fs timestep.
thermo 1000
run 20000

write_dump all custom &
  ${element}/cfgs/dump.${element}-crystal.${nx}x${ny}x${nz}.${bc} &
  id type x y z vx vy vz fx fy fz  modify sort id
```

IV. CPU BENCHMARK SCRIPTS

A. *scale_bench.sh*

Job running script to run CPU benchmarks on Quartz.

```
lmp="./lmp_mpi_quartz"

#for nnodes in 400 256 144 128 100 64 36 32 25 16 9 8 6 4 3 2 1 ; do
for nnodes in 8 ; do
    ntasks=$((nnodes*36))
    # Ta, W
    if ( true ) ; then
    #for shapes in "192 261 8" "256 261 6" ; do
    for shapes in "256 261 6" ; do

nx=`echo $shapes | awk '{print $1}'`
ny=`echo $shapes | awk '{print $2}'`
nz=`echo $shapes | awk '{print $3}'`
for elem in Ta W ; do
    echo "element=$elem, nx=$nx, ny=$ny, nz=$nz, nnodes=$nnodes"
    for bc in 0 ; do
srun -N$nnodes -n$ntasks -t199  $lmp -in in.bench \
     -var element $elem \
     -var nx $nx -var ny $ny -var nz $nz -var pbc $bc \
     -log nbench.log.$nx.$ny.$nz.$elem.$bc.p$nnodes \
     > nbench.run.$nx.$ny.$nz.$elem.$bc.p$nnodes  &
    done
done
    done
    fi
    # Cu
    if ( true ) ; then
    for shapes in "174 192 6" ; do
nx=`echo $shapes | awk '{print $1}'`
ny=`echo $shapes | awk '{print $2}'`
nz=`echo $shapes | awk '{print $3}'`

for elem in Cu ; do
    echo "element=$elem nx=$nx, ny=$ny, nz=$nz, nnodes=$nnodes"
    for bc in 0 ; do
srun -N$nnodes -n$ntasks -t199 $lmp -in in.bench \
     -var element $elem \
     -var nx $nx -var ny $ny -var nz $nz -var pbc $bc \
     -log nbench.log.$nx.$ny.$nz.$elem.$bc.p$nnodes \
     > nbench.run.$nx.$ny.$nz.$elem.$bc.p$nnodes  &
    done
done
    done
    fi

    echo "Submitted all jobs for nnodes=$nnodes"
    sleep 5
done
```

B. *in.bench*

Main LAMMPS input file for benchmark runs on Quartz (CPUs).

```
print "Simulation of ${element} with ${nx}x${ny}x${nz} unit cells"
include ${element}/${element}.inc

processors * * 1 grid twolevel 36 6 6 1

units metal
atom_style atomic
atom_modify map hash

lattice ${latttype} $(v_alatt*v_thexp) # Effective lattice constant
region box block 0 ${nx} 0 ${ny} 0 ${nz}
create_box 1 box

if "${pbc}==1" then "jump SELF pbc_subsection"
if "${pbc}==2" then "jump SELF z_bc_subsection"
label open_bc_subsection
  variable bc string open
  change_box all boundary s s s
  jump SELF bc_subsection_end

label z_bc_subsection
  variable bc string openxy
  change_box all boundary s s p
  jump SELF bc_subsection_end

label pbc_subsection
  variable bc string pbc

label bc_subsection_end


read_dump  ${element}/cfgs/dump.${element}-crystal.${nx}x${ny}x${nz}.${bc} &
  20000  x y z vx vy vz box yes add yes
mass 1 ${M}

if "${alloytype}==1" then "jump SELF alloy_subsection"
label eam_subsection
  pair_style eam
  pair_coeff * * ${element}/${potfile}
  jump SELF potential_end
label alloy_subsection
  pair_style eam/alloy
  pair_coeff * * ${element}/${potfile} ${element}
label potential_end

print "Input file mass = ${M}, after potential init, atom mass = $(mass[1])"

neighbor 0.5 bin
neigh_modify every 1 delay 0 check yes

fix 1 all nve

compute msd all msd

#fix avgs all ave/time 1 1000 1000 c_thermo_press c_thermo_temp
#thermo_style custom step f_avgs[2] pe etotal f_avgs[1] c_msd[1]
```

```
thermo_style custom step temp pe etotal press c_msd[1]

timestep 0.002  ## 2 fs timestep.
thermo 10000
run 100000
#write_dump all custom &
#  dump.x &
#  id type x y z vx vy vz fx fy fz  modify sort id
```

V. CPU BENCHMARK SCRIPTS

A. *run_multi.sh*

Job submission script for running all benchmarks on Frontier.

```
#!/bin/bash
#SBATCH --account=xxx
#SBATCH --nodes=32
#SBATCH --partition=batch
#SBATCH --time=00:30:00
#SBATCH --exclusive
#SBATCH --core-spec=16
#SBATCH --ntasks-per-node=8
#SBATCH --threads-per-core=1
#SBATCH --job-name=benchmarking
#SBATCH --output=lammps_snap.%j.out

module load craype-accel-amd-gfx90a
module load rocm/5.7.1
module load cray-mpich/8.1.27

export LD_LIBRARY_PATH=${CRAY_LD_LIBRARY_PATH}:${LD_LIBRARY_PATH}

#rm log.*

# Enable GPU MPI buffers
export MPICH_GPU_SUPPORT_ENABLED=1

# Make sure the appropriate NIC is chosen per MPI rank.
export MPICH_OFI_NIC_POLICY=NUMA

# Announce and then actually run MY_EXE for each MPI rank
echo "Starting actual srun now: `date '+%y%m%d_%H%M:%S'`"

EXE=./lmp_frontier_kokkos
INPUT=in.bench

for elem in Cu Ta W ; do
  if [ "$elem" == "Cu" ]; then
    nx=174
    ny=192
    nz=6
  else
    nx=256
    ny=261
    nz=6
  fi
  for bc in 0 ; do

    /usr/bin/time -f'%C\nTook %e secs' srun -u -N1 -n1 -c1 --cpu-bind=map_cpu:50 --gpus=1 ${EX
      -k on g 1 -sf kk -pk kokkos neigh full newton off \
      -var element $elem \
      -var nx $nx -var ny $ny -var nz $nz -var pbc $bc \
      -log bench.log.$elem.${nx}x${ny}x${nz}.pbc_$bc.gpus_1

    for NNODES in 1 2 4 8 16 32 ; do
      /usr/bin/time -f'%C\nTook %e secs' srun -u -N ${NNODES} --ntasks-per-node=8 --cpus-per-t
      -k on g 1 -sf kk -pk kokkos neigh full newton off \
      -var element $elem \
      -var nx $nx -var ny $ny -var nz $nz -var pbc $bc \
      -log bench.log.$elem.${nx}x${ny}x${nz}.pbc_$bc.gpus_$((NNODES*8))
    done
  done
done

# All done!
endTS=`date '+%s'`
echo "JobID ${SLURM_JOBID} on ${SLURM_NNODES} nodes ended: `date '+%y%m%d_%H%M:%S'`: $((endTS
```

B. *in.bench*

Main LAMMPS input file for benchmark runs on Forntier (GPUs).

```
print "Simulation of ${element} with ${nx}x${ny}x${nz} unit cells"
include ${element}/${element}.inc

units metal
atom_style atomic

lattice ${latttype} $(v_alatt*v_thexp) # Effective lattice constant
region box block 0 ${nx} 0 ${ny} 0 ${nz}
create_box 1 box

if "${pbc}==1" then "jump SELF pbc_section"
if "${pbc}==2" then "jump SELF z_bc_section"
label open_bc_section
  variable bc string open
  change_box all boundary s s s
  jump SELF bc_section_end

label z_bc_section
  variable bc string openxy
  change_box all boundary s s p
  jump SELF bc_section_end

label pbc_section
  variable bc string pbc

label bc_section_end


read_dump  ${element}/cfgs/dump.${element}-crystal.${nx}x${ny}x${nz}.${bc}.gz &
  20000  x y z vx vy vz box yes add yes
```

```
mass 1 ${M}

if "${alloytype}==1" then "jump SELF alloy_section"
label eam_section
  pair_style eam
  pair_coeff * * ${element}/${potfile}
  jump SELF potential_end
label alloy_section
  pair_style eam/alloy
  pair_coeff * * ${element}/${potfile} ${element}
label potential_end

neighbor 1.0 bin
neigh_modify every 1 delay 0 check yes

fix 1 all nve

thermo_style custom step temp pe etotal press

timestep 0.002  ## 2 fs timestep.
thermo 0
run 8000
```

VI. Material parameters and potentials

The following files define material parameters and describe where to obtain the potential files. The material parameter files are used by the equilibration and benchamrk scripts, for both CPU and GPU runs.

A. *Cu/Cu.inc*

```
#variable element string Cu
variable M equal 63.55

variable latttype string fcc
variable alatt equal 3.615

variable T equal 290.0
variable thexp equal 1.00468

variable alloytype equal 0
variable potfile string Cu_u6.eam
```

B. *Cu/Cu_u6.readme.txt*

```
The Cu_u6.eam file in this directory can also be obtained from any
LAMMPS distribution, in the potentials subdirectory, and from the
OpenKIM potential database:

  https://openkim.org/id/
    EAM_Dynamo_AdamsFoilesWolfer_1989Universal6_Cu__MO_145873824897_000

with a direct link to the compressed file here:

  https://openkim.org/download/
    EAM_Dynamo_AdamsFoilesWolfer_1989Universal6_Cu__MO_145873824897_000.zip
```

C. Ta/Ta.inc

```
#variable element string Ta
variable M equal 180.95

variable latttype string bcc
variable alatt equal 3.3026

variable T equal 290.0
variable thexp equal 0.99888

variable alloytype equal 1
variable potfile string Ta_Li_2003.eam.alloy
```

D. Ta/Ta_Li_2003.eam.alloy.readme.txt

```
The Ta_Li_2003.eam.alloy file in this directory can also be obtained
from the OpenKIM potential database:

  https://openkim.org/id/
    EAM_Dynamo_LiSiegelAdams_2003_Ta__MO_103054252769_005

with a direct link to the compressed file here:

  https://openkim.org/download/
    EAM_Dynamo_LiSiegelAdams_2003_Ta__MO_103054252769_005.zip
```

E. W/W.inc

```
#variable element string W
variable M equal 183.84

variable latttype string bcc
variable alatt equal 3.157

variable T equal 290.0
variable thexp equal 1.003635

variable alloytype equal 1
variable potfile string W_zhou_short.eam.alloy
```

F. W/W_zhou_short.eam.alloy.readme.txt

```
The W_zhou_short.eam.alloy file (*) in this directory can also be
obtained from any LAMMPS distribution, in the potentials subdirectory,
and from the OpenKIM potential database:

  https://openkim.org/id/
    EAM_Dynamo_ZhouWadleyJohnson_2001_W__MO_621445647666_000

with a direct link to the compressed file here:

  https://openkim.org/download/
    EAM_Dynamo_ZhouWadleyJohnson_2001_W__MO_621445647666_000.zip

(*) Modification: Compared to the publicly available references above,
we modified the file in the following ways: We changed the cut-off
radius to rc = 5.524750 A. For the rho(r) and phi(r) we maintained
only every 14'th point of each function table, starting with the
```

```
first, and omitted points for r > rc. For the F(rho) function, we
maintained only every 20'th points, starting with the first. That
means we have 501 points in each of the function tables as opposed to
10001 in the original file.
```

Acknowledgments

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. The document release number is LLNL-CONF-863721.