Reproducible Reductions in OpenMP with SPRAY

Jan Hückelheim
jhueckelheim@anl.gov

Johannes Doerfert
doerfert1@llnl.gov

U.S. DEPARTMENT OF ENERGY
Argonne National Laboratory is a U.S. Department of Energy laboratory managed by UChicago Argonne, LLC.

Argonne NATIONAL LABORATORY
75 1946–2021

Flexible reductions with SPRAY

- Originally intended for sparse array reductions like this:

```
#pragma omp parallel for reduction(+:res)
for(int i=0, i<n; i++) {
  res[idx(i)] += ...
}
```

- Default OpenMP: Create copy of `res` on each thread
- Customized behavior with SPRAY: Atomic updates, block-wise lazy privatization…
- See our IPDPS 2021 paper: https://ieeexplore.ieee.org/document/9460492

Flexible reductions with SPRAY

- Originally intended for sparse array reductions like this:

```
res = spray::BlockReduction(res_orig)
#pragma omp parallel for reduction(+:res)
for(int i=0, i<n; i++) {
  res[idx(i)] += ...
}
```

- Default OpenMP: Create copy of `res` on each thread
- Customized behavior with SPRAY: Atomic updates, block-wise lazy privatization…
- See our IPDPS 2021 paper: https://ieeexplore.ieee.org/document/9460492

Reproducibility

- Common reviewer / user question: Are SPRAY reductions reproducible?

- Different kinds of reproducibility:
  - Run-to-run: Same executable, runtime, machine, settings, static schedule
  - Portable: Same code, but different compiler, thread count, dynamic schedule, etc

- OpenMP can be “run-to-run” reproducible depending on setup
- SPRAY offers “portable” reproducibility

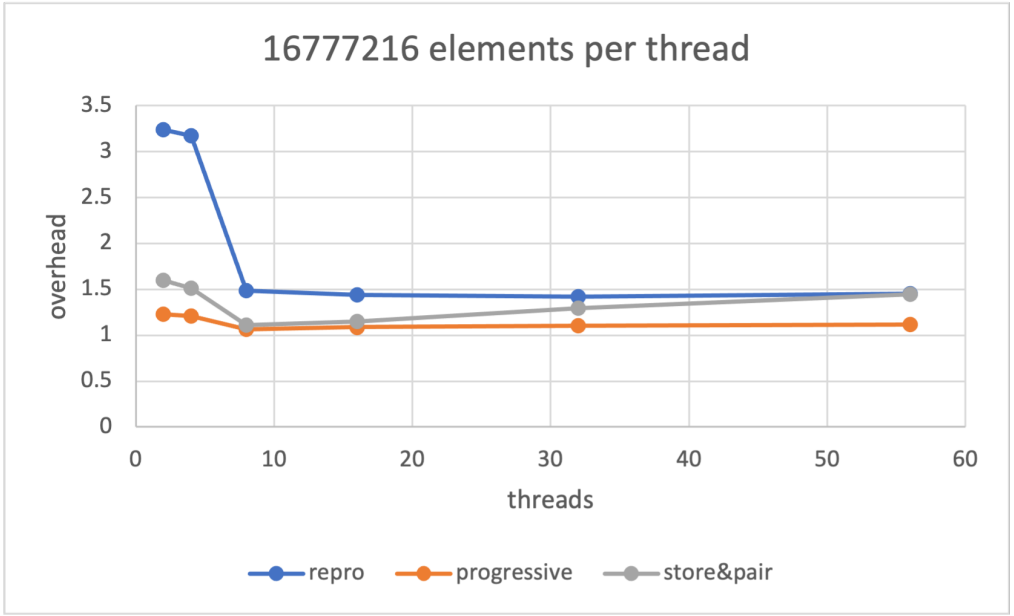
Reproducible reductions with SPRAY

- Reduction variable is wrapped in SPRAY reducer object:

```
res = spray::ReproReduction(res_orig)
#pragma omp parallel for reduction(+:res)
for(int i=0, i<n; i++) {
  res += ...
}
```

- Use BinnedNumbers https://dl.acm.org/doi/10.1145/3389360
- Constant factor overhead in time/memory
- Same result regardless of schedule, thread count, compiler

Overhead of reproducible reductions



- Depending on thread and iteration count, 1.5x or >3x slowdown
- Other approaches (progressive, store&pair) are faster, but require instrumentation or support by compiler / OpenMP run time

Questions?

- Jan Hückelheim: jhueckelheim@anl.gov
- Johannes Doerfert: doerfert1@llnl.gov

Acknowledgements

The work was funded in part by support from the U.S. Department of Energy, Office of Science, under contract DE-AC02-06CH11357. We gratefully acknowledge the computing resources provided and operated by the Joint Laboratory for System Evaluation (JLSE) at Argonne National Laboratory. The work was additionally in part supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of two U.S. Department of Energy organizations (Office of Science and the National Nuclear Security Administration) responsible for the planning and preparation of a capable exascale ecosystem, including software, applications, hardware, advanced system engineering, and early testbed platforms, in support of the nation's exascale computing imperative.