Ensuring Correctness for Exascale and Beyond

Breakout session
DoE X-Stack PI meeting 4/7/2016

Aggravated problems at Exascale

Goals

- Need to maximize parallelism and minimize data transfers while being accurate and correct
- Need to reuse code and components to promote rapid development

Approaches:

- Non-determinism (e.g. 2.5D communication-avoiding algorithms exploit reduction parallelism)
- Approximations (e.g. fast multipole methods [2], -ffast-math)
- Modular design: multiple runtimes (e.g. MPI, OpenMP, GasNet) and languages (e.g. Fortran, C)

Tradeoffs:

- **Inaccuracies** from non-determinism ("non-reproducibility")

Detect and Eliminate Inaccuracies and Inefficiencies

Approaches:

- Theorem proving (e.g. coq)
    - Semi-automated
    - Full correctness
- Static analysis, abstract-interpretation, symbolic execution, model-checking
    - False positives
    - Partial correctness guarantees
- Dynamic analysis

Accuracy & reproducibility: Some approaches

Why we need reproducibility and accuracy

- Debuggability
- Contractual obligations

Determine worst-case accuracy statically (e.g., Coq [4])

- Guarantees, conservative bounds. No runtime overhead. Not fully automatic.

Maximize accuracy by dynamically adapting computations (e.g., Herbie [5], In situ UQ)

- Better average accuracy. May improve worst-case. Runtime overhead.

Minimize precision while guaranteeing good-enough accuracy (e.g. Precimonious)

Questions

Which **specific** properties of **scientific codes** do need verification and analysis?

Are there **tools** to address these challenges, or some of them ?

Can we use dynamic analysis to optimize performance and to avoid bugs?

- Can we build a easy-to-use tool to perform dynamic analysis and optimization?

Error **prevention** vs. **detection**

- Mirrors detection and recommendation of optimization opportunities

What kind of guarantees can we get ?

- Strength, coverage in code **types**, **volume** (locs) and **diversity** (languages)

References

1. Solomonik, Demmel - Communication-optimal 2.5D matrix multiply and LU factorization algorithms. http://www.eecs.berkeley.edu/Pubs/TechRpts/2011/EECS-2011-72.pdf, 2011.

2. Beatson, Greengard - A short course on Fast Multipole Methods. https://web.njit.edu/~jiang/math614/beatson-greengard.pdf

3. Leroy and team - CompCert http://compcert.inria.fr, 2009.

4. Ramananandro et al- A unified Coq framework for verifying C programs with floating-point computations. CPP 2016.

5. Pavel Panchekha et al- Herbie: Automatically Improving Floating-Point Accuracy. http://herbie.uwplse.org/

6. Demmel et al - ReproBLAS: Reproducible BLAS. http://bebop.cs.berkeley.edu/reproblas/

7. Rubio-Gonzalez, Sen, Demmel, Iancu et al.- Precimonious: Precision tuning of floating point programs