Quinoa: Adaptive Computational Fluid Dynamics

J. Bakosi, R. Bird, C. Junghans, R. Pavel, J. Waltz  
Los Alamos National Laboratory

F. Gonzalez  
University of Illinois Urbana-Champaign

B. Rogers  
University of Tennessee

April 18, 2017

Goal: hardware-adaptive large-scale multiphysics

- Fluid dynamics, turbulence, particle transport, chemistry, plasma physics of non-ideal multiple mixing materials
- Automatic dynamic computational load redistribution for real-world problems
- Preserving the domain scientist’s sanity

Agenda:

- Philosophy
- Infrastructure
- Two tools: particle solver, unstructured-grid PDE solver
- Future plan

https://github.com/quinoacomputing/quinoa

LA-UR-17-22931
Philosophy
- Partition everything
- Be asynchronous everywhere
- Automate everything
- Remember that everything fails

Strategy
- Most physics codes start with capability then software engineering is an afterthought
- We start with a state-of-the-art production code then put in physics
- From scratch: not based on existing code
- C++11 & Charm++ (fully asynchronous, distributed-memory parallel)

Funding & history
- Started as a hobby project in 2013 (weekends and nights)
- First funding: Oct 2016

Work in progress
Infrastructure

- 46K lines of code
- 20+ third-party libraries, 3 compilers
- Unit-, and regression tests
- Open source: https://github.com/quinoacomputing/quinoa
- Continuous integration (build & test matrix) with Travis & TeamCity
- Continuous quantified *test code* coverage with Gcov & CodeCov.io
- Continuous quantified *documentation* coverage with CodeCov.io
- Continuous static analysis with CppCheck & SonarQube
- Continuous deployment (of binary releases) to DockerHub

Ported to Linux, Mac, Cray (LANL, NERSC), Blue Gene/Q (ANL)
Current tools

1. **walker** – Random walker for stochastic differential equations
2. **inciter** – Partial differential equations solver on 3D unstructured grids
3. **rngtest** – Random number generator test suite
4. **unittest** – Unit test suite
5. **meshconv** – Mesh file converter
Quinoa::Walker

- **Particle solver**
- **Numerical integrator for stochastic differential equations**
- Used to analyze and design the evolution of fluctuating variables and their statistics
- Used in production for the design of statistical moment approximations required for modeling mixing materials in turbulence

**Future plan:** Predict the probability density function in turbulent flows

\[
\frac{\partial}{\partial t} F(Y, t) = -\sum_{\alpha=1}^{N-1} \frac{\partial}{\partial Y_\alpha} \left[ A_\alpha(Y, t) F(Y, t) \right] + \frac{1}{2} \sum_{\alpha=1}^{N-1} \sum_{\beta=1}^{N-1} \frac{\partial^2}{\partial Y_\alpha \partial Y_\beta} \left[ B_{\alpha\beta}(Y, t) F(Y, t) \right]
\]

\[
dY_\alpha(t) = A_\alpha(Y, t) dt + \sum_{\beta=1}^{N} b_{\alpha\beta}(Y, t) dW_\beta(t), \quad \alpha = 1, \ldots, N, \quad B_{\alpha\beta} = b_{\alpha\gamma} b_{\gamma\beta}
\]
Walker SDAG for each PE

AdvP – advance particles
OrdM – estimate ordinary moments
CenM – estimate central moments, e.g., $\langle y - \langle Y \rangle \rangle^2$
OutS – output statistical moments
EvT – evaluate time step
OrdP – estimate ordinary PDFs
CenP – estimate central PDFs, e.g., $F(y - \langle Y \rangle)$
OutP – output PDFs
NoSt – no stats, nor PDFs

src/Walker/distributor.ci
Walker weak scaling with up to $3 \times 10^9$ particles

![Graph showing Walker weak scaling with up to $3 \times 10^9$ particles.](image)
**Goal:** Predict the probability density function in turbulent flows

**Why:** Because it requires less approximations

**How:** Integrate a large particle ensemble governed by stochastic differential equations

- The ensemble represents the fluid itself
- Statistics and the discrete PDF extracted from the ensemble in cells

![Diagram with particles and PDF curves](image.png)
Quinoa::Inciter

- PDE solver for 3D unstructured (tet-only) grids
- Native Charm++ code using MPI-only libs: *hypre*, *Zoltan2*
- Simple Navier-Stokes solver for compressible flows
- Finite elements
- Flux-corrected transport
- Asynchronous linear system assembly
- File/PE I/O
- Current work: adaptive mesh refinement, V&V
- **Future plan:** use AMR to explore scalability with large load-imbalances
Flux-corrected transport

- Used when stuff (e.g., energy) moves from A to B (i.e., all the time)
- Godunov theorem: No linear scheme of order greater than one will yield monotonic (wiggle-free) numerical solutions.
- A solution: Use a nonlinear scheme
- Combine a low-order (guaranteed to be monotonic) with a high-order (more accurate) scheme in a nonlinear fashion

![Graphs]

- exact
- low-order
- high-order
- FCT
Matrix assembly

Matrix distributed across PEs (Charm++ group)

L1, L2, ... – LinSysMerger Charm++ group elements
- interact with MPI–only linear system solver lib
- do not migrate

C1, C2, ... – Carrier worker Charm++ array elements
- perform heavy–lifting of physics
- migrate (not yet but will)
Inciter SDAG for each PE

ChRow – chares contribute their global row IDs
ChBC – chares contribute their BC node IDs
RowComplete – all groups have finished their row IDs
Init – chares initialize
dt – chares compute their next \( \Delta t \)
Aux – Low order solution
Solve – Call hypre to solve linear system
Asm* – Assemble RHS/LHS/UNK
Hypre* – Convert RHS/LHS/UNK to hypre data structure
Compressible Navier-Stokes, 794M
(setup, 100 time steps, no I/O)

Wall clock time, sec

Number of CPU cores (36/node)

~50Kel/PE
Quinoa::Inciter future plan

▷ **Now:** Distributed-memory-parallel asynchronous AMR
▷ **Next:** Explore scalability with large load-imbalances (migration)
▷ **Future:**
  ▶ Asynchronous I/O
  ▶ Explore various threading and SIMD abstractions
  ▶ Explore CERN’s ROOT framework for data storage, statistical analysis, and visualization
  ▶ Fault tolerance

Figure 2: Allowable refinement (left) and derefinement (right) patterns

Acknowledgments

TPLs: Charm++, Parsing Expression Grammar Template Library, C++ Template Unit Test Framework, Boost, Cartesian product, PStreams, HDF5, NetCDF, Trilinos: SEACAS, Zoltan2, Hypre, RNGSSE2, TestU01, PugiXML, BLAS, LAPACK, Adaptive Entropy Coding library, libc++, libstdc++, MUSL libc, OpenMPI, Intel Math Kernel Library, H5Part, Random123

Compilers: Clang, GCC, Intel

Tools: Git, CMake, Doxygen, Ninja, Gold, Gcov, Lcov, NumDiff