Adaptive Computational Fluid Dynamics with Charm++

J. Bakosi, A. Pandare, R. Bird, J. Waltz,
W. Li, H. Luo
E. Bohm, L. Kale, E. Mikida, E. Ramos

[1]Los Alamos National Laboratory

[2]North Carolina State University

[3]Charmworks, Inc.

Managed by Triad National Security, LLC for the U.S. Department of Energy's NNSA

We are writing a Charm++ app from scratch: Quinoa

- Native Charm++, C++17
- Asynchronous, distributed-memory parallel
- Production code: 90% test coverage
- Open source, BSD3: `https://quinoacomputing.org`

Goal: Multiphysics at large scales with automatic load balancing, currently:

- Multiple finite element flow solvers
- Single-, and multi-material hydrodynamics
- Complex engineering geometries
- Adaptive mesh refinement
- Polynomial-order adaptation
- Load balancing with Charm++

Choice of runtime system in 2014: Charm++

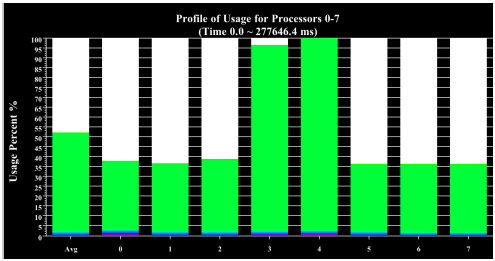
Adaptive numerical methods

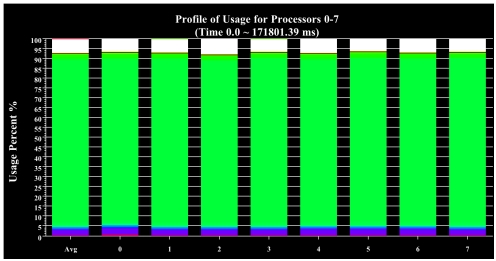
- Concentrate computational power in regions where accuracy is desired
- *A priori* unknown load-imbalance → obvious need for Charm++

Causes of load-imbalance

- Multiple ways of solution-adaptation, e.g., mesh refinement
- Different-cost equations of state for different materials
- Different physics active at different regions
- ...

LB example: Polynomial-order-adaptive finite elements





CPU utilization *without* and *with* LB on a laptop. Mild load imbalance, improvement: 2×.

Fluid equations

- System of PDEs: conservation of mass, momentum, energy + source (e.g., reactions)

$$\frac{\partial \mathbf{U}}{\partial t} + \frac{\partial \mathbf{F}_k}{\partial x_k} = \mathbf{S}$$

- Compute discrete solution in cells of a mesh

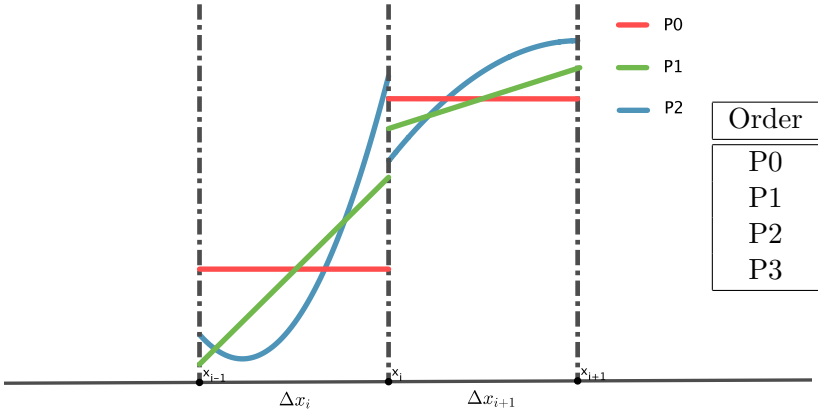
$$\int_\Omega \frac{\partial \mathbf{U}}{\partial t} W_j \mathrm{d}\Omega + \int_\Gamma \mathbf{F}_k(U) n_k W_j \mathrm{d}\Gamma - \int_\Omega \mathbf{F}_k(U) \frac{\partial W_j}{\partial x_k} \mathrm{d}\Omega = \int_\Omega \mathbf{S}(U) W_j \mathrm{d}\Omega$$

- Discrete equations assembled to a large linear system

$$\mathbf{M}_{ij} \frac{\partial \mathbf{u}_i}{\partial t} = \mathbf{R}_j$$

Approximating polynomials in a cell

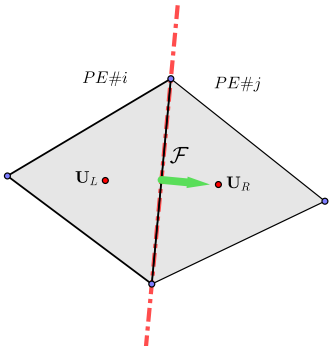
Solution polynomial: $\mathbf{U}_h(x) = \sum_i^{ndof} u_i B_i(x)$



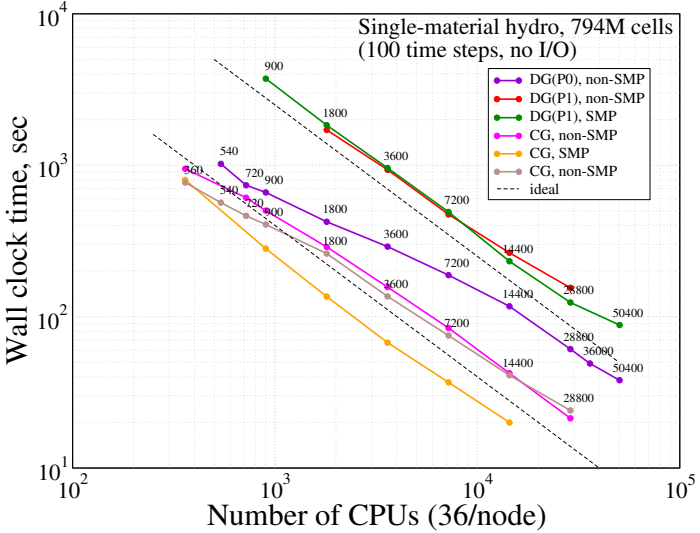
Order	B_i
P0	$[1]$
P1	$[1, \tilde{x}]$
P2	$[1, \tilde{x}, \tilde{x}^2]$
P3	$[1, \tilde{x}, \tilde{x}^2, \tilde{x}^3]$

Parallel communication across mesh/chare partitions

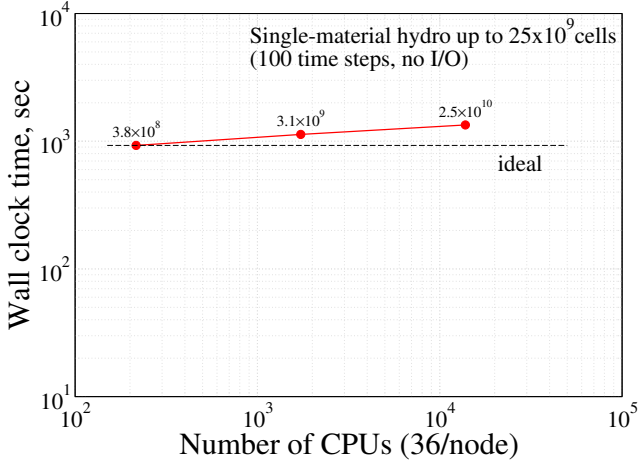
- Solver needs matching left and right states
- Requires (p2p) communication multiple times per time step
- All flow algorithms use a single global sync per time step



Strong scaling



Weak scaling



Advection with adaptive method

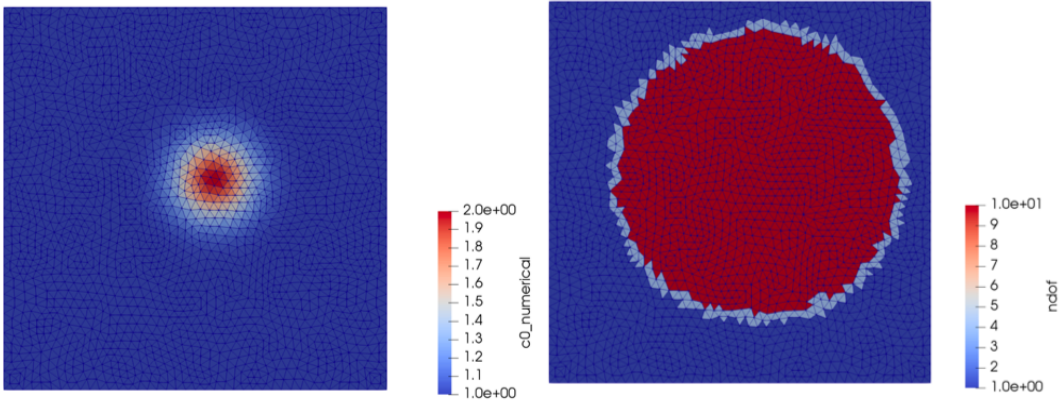


Figure: Fluid density (*left*) and computational cost per cell (*right*) using the adaptive method

Shock-hydrodynamics with adaptive method

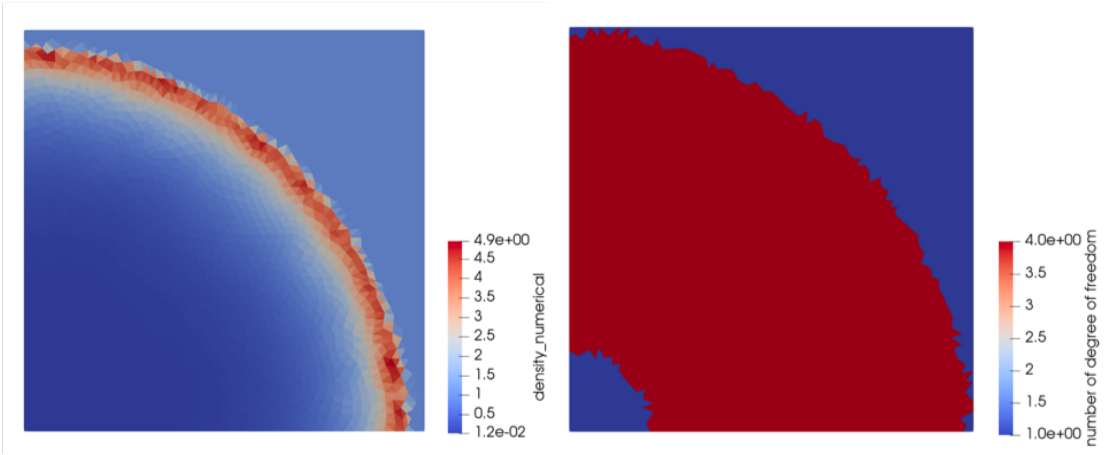


Figure: Fluid density (*left*) and computational cost (*right*) using the adaptive method

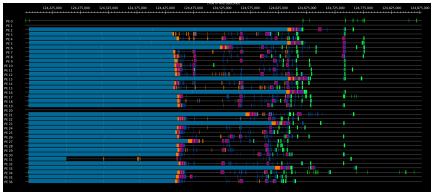
LB with p-adaptation: same numerical errors

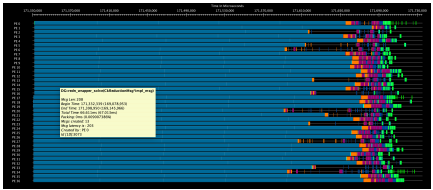
1 CPU	Time
3rd order DG (non-adaptive)	28:31
Adaptive DG	14:36

It is worth adapting!

8 CPUs	Time
3rd order DG (non-adaptive)	4:04
Adaptive DG (no load-balancing)	4:10
Adaptive DG (load-balanced)	2:39

But (in parallel) only if balanced!





Summary

Our ultimate goal is

- to simulate large and complex engineering multiphysics problems
- with a production-quality code that is extensible and maintainable,
- using hardware resources efficiently
- even with *a priori* unknown, heterogeneous, and dynamic load distribution.

Collaborators and contributors are welcome, see `https://quinoacomputing.org`.