LA-UR-14-28110
Approved for public release; distribution is unlimited.

Title: LANL Summer Internship 2014

Author(s): Grosset, Andre
Junghans, Christoph

Intended for: Predictive Science Academic Alliance Program Meeting

Issued: 2014-10-16

LANL Summer Internship 2014

Pascal Grosset

The Project

- Evaluating Distributed Runtimes in the Context of Adaptive Mesh Refinement

- Setup:
  - Summer co-design school
    - Mentors: Allen McPherson, Ben Bergen, Christoph Junghans
    - 3 Computer Science students + 3 Applied Maths Students
  - Work:
    - Science part: Sod Shock problem
    - CS part: Investigate Runtimes
    - Common: Evaluate how to best represent the AMR time varying data

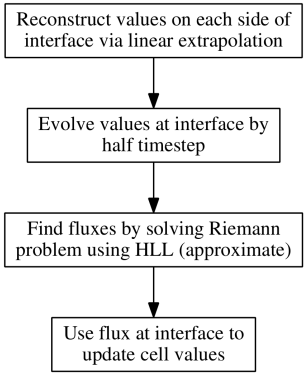
Sod Shock problem

High Pressure | Low Pressure

2D Euler Equation

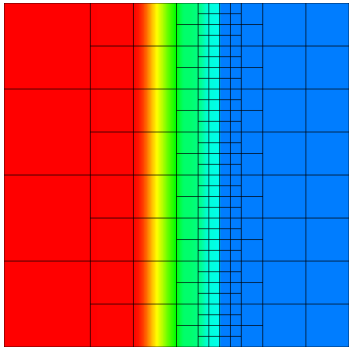
$$\frac{\partial}{\partial t}\begin{bmatrix}\rho\\ \rho u_x\\ \rho u_y\\ E\end{bmatrix}+\frac{\partial}{\partial x}\begin{bmatrix}\rho u_x\\ \rho u_x^2+p\\ \rho u_x u_y\\ (E+p)u_x\end{bmatrix}+\frac{\partial}{\partial y}\begin{bmatrix}\rho u_y\\ \rho u_y u_x\\ \rho u_y^2+p\\ (E+p)u_y\end{bmatrix}=0$$

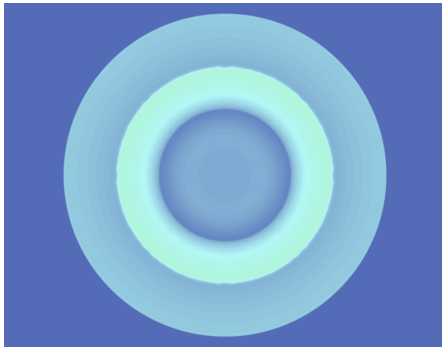
MUSCL-Hancock Scheme



AMR Scheme

- Tile-based AMR
- Only finest resolution stored
- Refinement Strategy
  - Gradient criteria
  - Löhner error estimator





Runtimes

- Why?
  - Simplify programming on distributed systems
    - Abstraction of MPI, threads and memory
  - Has useful features like:
    - Load balancing
    - Fault Tolerance
    - Management of distributed memory
  - Runtimes considered:
    - Charm++ (Urbana Champaign)
    - HPX (Louisiana State University)
    - CnC (Intel)

HPX

- C++ runtime for parallel and distributed systems
- Features:
  - Message driven
  - Asynchronous execution
  - Work stealing task schedulers
  - Global Address Space
- Limitations:
  - HPX is new and rapidly changing (version 0.98)
  - It does not have many features of more established runtimes

Evaluating Distributed Runtimes in the Context of Adaptive Mesh Refinement

Introduction

Hydrodynamics simulations with shock discontinuities represent a variety of applications. Due to the large size of many application problems, it is infeasible to solve the entire problem on a uniform grid as they are both computationally expensive and memory intensive. To combat this issue, adaptive mesh refinement (AMR) is often used in order to limit both computational cost and memory use while achieving the desired accuracy. This project implements hydrodynamics simulations based on the physical model of the Euler equations using a second order Finite Volume Method with AMR.

Additionally, a growing trend in the field of scientific computing is the utilization of system-level runtimes to simultaneously provide a more intuitive means of exploiting the inherent parallelism of algorithms while also simplifying the challenges of load balancing. To this end, we have performed a survey of contemporary runtime systems with a focus on the inherent challenges of adaptive mesh refinement and have implemented our application in Charm++, HPX, and Intel's Concurrent Collections.

Physical Model

- **2D Euler Equations:**
  - Given

$$\mathbf{q} = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ E \end{bmatrix}, \; f = \begin{bmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ (E+p)u \end{bmatrix}, \; g = \begin{bmatrix} \rho v \\ \rho uv \\ \rho v^2 + p \\ (E+p)v \end{bmatrix}$$

  the Euler equations may be expressed in conservative form as

$$\mathbf{q}_t + f(\mathbf{q})_x + g(\mathbf{q})_y = 0.$$

- **Ideal Gas:**
  - The conserved quantities are coupled with pressure and velocity via the equation of state

$$E = \frac{p}{\gamma - 1} + \frac{1}{2}\rho(u^2 + v^2).$$

Numerical Solution

- **Dimensional Splitting:**
  - Solution is computed by splitting into two coupled one-dimensional problems:

$$\mathbf{q}_t + f(\mathbf{q})_x = 0,$$
$$\mathbf{q}_t + g(\mathbf{q})_y = 0.$$

  - Two one-dimensional problems are solved using a finite volume formulation.

- **Finite Volume Method:**
  - Domain decomposed into discrete cells.
  - Average value stored in each cell

$$Q_i^n = \frac{1}{\Delta x}\int_{C_i} q(x, t_n)dx.$$

  - Averages are iterated in time

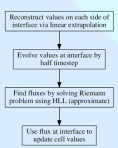
$$Q_i^{n+1} = Q_i^n - \frac{\Delta t}{\Delta x}\left(F_{i+1/2}^n - F_{i-1/2}^n\right),$$

  where

$$F_{i-1/2}^n = \frac{1}{\Delta t}\int_{t_n}^{t_{n+1}} f(q(x_{i-1/2}, t))dt.$$

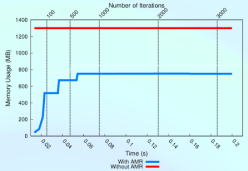
  - The fluxes, $F_{i-1/2}^n$, $F_{i+1/2}^n$, are determined using a Riemann solver.

- **MUSCL-Hancock Scheme:**
  - Predictor-corrector scheme
  - Second-order accurate in space and time



Introduction to Adaptive Mesh Refinement

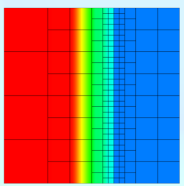
- **What is AMR?**
  - Refine only a small portion of the grid
  - Refine where needed (often near discontinuities) in order to achieve desired accuracy
- **Why AMR?**
  - Memory usage
  - Execution time
- **Considerations with AMR:**
  - Frequently changing work load
  - Flux correction at interfaces with varying refinement level



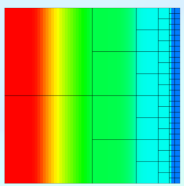
Comparison of memory usage with uniform and adaptive refinement.

AMR Implementation

- **Tile-based AMR:**
  - Distinctly different from "Cell-based AMR" and Berger's "Patch-based AMR"
  - Grid represented as combination of non-overlapping, fixed-size (number of cells) tiles
  - Tiles refined into 4 new tiles
  - Each tile stores "ghost cells"
  - Only lowest level tiles stored
- **Variety of Refinement Strategies:**
  - Gradient criterion
  - Loehner error estimator
- **Simulation of Sod Shock Tube Problem with AMR:**



Density after 160 iterations.



Density after 990 iterations.

Introduction to Runtimes

- **What are Runtimes?**
  - Provide high level abstraction for parallel programming on distributed systems
  - Offer portability between platforms to enhance productivity
  - Mask explicit communication with high level primitives
- **Conventional Scientific Computing Approach: MPI+X Model**
- **Why Runtimes?**
  - Load balancers to handle changing workloads
  - Fault tolerance to increase resiliency
  - Homogeneous interface for application developers
  - Management of distributed memory

Charm++

- **What is Charm++?**
  - A parallel object-oriented programming languaged based on C++
  - Focuses on enhancing programmer produtivity through abstraction of parallel programming
  - Original developed at the University of Illinois in 1993
- **Key Features:**
  - Tasks are fine grain, over-decomposed, asynchronous units of work
  - Communication is message-driven, where messages trigger compute events
  - Designed to be asynchronous, including syntax for structured control flow.
  - Automatic load balancing
  - Automatic work distribution
  - Automatic checkpointing
  - Shown to scale successfully to over 300,000 cores
- **Limitations:**
  - Non-trivial learning curve, including the use of new syntax

High Performance ParalleX (HPX)

- **What is HPX?**
  - C++ runtime system for parallel and distributed applications
  - Aims to overcome common issues in parallel programming such as work starvation, latency, overhead, waiting for contention resolution
- **Key Features:**
  - Global address space provides communication transparency
  - Lightweight control objects instead of barriers
  - Message driven
  - Fully asynchronous execution
  - Work stealing task schedulers
- **Limitations:**
  - HPX is quite new (currently at version 0.98) and rapidly changing
  - Does not yet provide load balancing or fault tolerance

Intel's Concurrent Collections (CnC)

- **What is CnC?**
  - Task-based C++ runtime built on Intel Thread Building Blocks (TBB)
  - Focus on productivity of domain experts
- **Key Features:**
  - Programming model built around specifying data dependencies of tasks
  - Specify parallelism of application in the form of dependencies and constraints
    - A task consumes and produces *Items* during a *Step*
    - Scheduler handles load balancing and exploits parallelism based on availability of *Items*
  - High portability due to separation of algorithm and tuning
  - Designed with composability in mind
- **Limitations:**
  - Write-once memory simplifies memory consistency model but adds additional complexities
  - No integrated fault tolerance

Closing Thoughts and Ongoing Work

- **Runtimes:**
  - Simplify and accelerate the implementation
  - Still working on performance comparison
- **Ongoing Work:**
  - Sequential performance analysis
  - Comparing tile and cell refinement
  - Thermal diffusion
  - Integrating EOSPAC

We acknowledge the support and guidance of our mentors: Allen McPherson and Ben Bergen. We also thank Christoph Junghans for additional advisement. We extend our gratitude to Thomas Masser, Bob Robey, and William Dai for providing insight into adaptive mesh techniques. Finally, we thank Phil Miller for assistance in properly utilizing the Charm++ runtime.




It was also fun outside the lab …

Thank you