Load Balancing and Data Migration in a Hybrid Computational Fluid Dynamics Application

Esteban Meneses
Patrick Pisciuneri

Center for Simulation and Modeling (SaM)
University of Pittsburgh
High Performance Computing
Computer Science
Scientific Computing
Center for Simulation and Modeling (SaM)

HPC researchers/consultants

521 users

8,040 cores

91% utilization in 2014
IPLMCFD

- A massively parallel solver for turbulent reactive flows.
- LES via filtered density function (FDF).
IPLMCFD uses a graph partitioning library (METIS) to redistribute work.

Requires to split execution between calls to repartition cells.
## Reasons for Load Imbalance in CFD

### Approaches:
- Task-parallel
- Zoltan
- Charm++

### Adaptive Mesh Refinement

**Traditional**

![Adaptive Mesh Refinement Traditional](image1)


<table>
<thead>
<tr>
<th>Traditional</th>
<th>IPLMCFD</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Adaptive Mesh Refinement Traditional" /></td>
<td><img src="image2" alt="Chemical Reaction" /></td>
</tr>
</tbody>
</table>

**Chemical Reaction**
Agenda

- IPLMCFD: A Hybrid Computational Fluid Dynamics Application
- Zoltan Library
- PaSR Benchmark
- Zoltan vs Charm++ Comparison
Hybrid CFD Application

- IPLMCFD: Irregularly Portioned Lagrangian Monte Carlo Finite Difference.
- Domain divided into cells, the atomic distribution unit.
- Ensemble of cells:
  - Same number of FD points.
  - Same number of MC particles.
## Computational Fluid Dynamics

**Load Balancing in a CFD Application**

<table>
<thead>
<tr>
<th># Grids</th>
<th># Particles</th>
<th># Species</th>
<th>Required Memory GBs</th>
<th>GFLOP per iteration</th>
<th># Iterations</th>
<th>Serial Run-time (1 GFLOP/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$10^6$</td>
<td>$6 \times 10^6$</td>
<td>9</td>
<td>1.69</td>
<td>29.5</td>
<td>60,000</td>
<td>20.5 days</td>
</tr>
<tr>
<td>$10^6$</td>
<td>$6 \times 10^6$</td>
<td>19</td>
<td>2.48</td>
<td>90.7</td>
<td>60,000</td>
<td>63 days</td>
</tr>
<tr>
<td>$5 \times 10^6$</td>
<td>$50 \times 10^6$</td>
<td>19</td>
<td>24.0</td>
<td>544.7</td>
<td>220,000</td>
<td>3.8 years</td>
</tr>
</tbody>
</table>
Load Balancing in a CFD Application

### Code Structure

- **Iplmcfd**: C++, 10,101 LOC
- **Ipfd**: C++, 3,091 LOC
- **Iplmc**: C++, 3,091 LOC
- **Metis**: Fortran/C, 3,091 LOC
- **TVMet**: Fortran/C, 3,091 LOC
- **Chemkin**: Fortran/C, 3,091 LOC
- **ODE Pack**: Fortran/C, 3,091 LOC

**Interface**
IPLMCFD

- A scalable algorithm for hybrid Eulerian/Lagrangian solvers.

- Goals:
  - Balance the computational load among processors through weighted graph partitioning.
  - To minimize the number of adjacent elements assigned to different processors (minimize the edge-cut).

- Irregularly shaped decompositions:
  - Disadvantages:
    - Nontrivial communication patterns
    - Increased communication cost.
  - Advantage (major):
    - Evenly distributed load among partitions.

Strong Scaling

- **Geometry:**
  - 2.5 million FD points
  - 20 million MC particles
  - Chemistry: 9 species, 5-step

- **Top:**
  - Unbalanced: 22% efficiency (9K cores)
  - IPLMCFD: 76% efficiency (9K cores)

- **Bottom:**
  - Performance of IPLMCFD improves as the number of MC particles increases
  - IPLMCFD: 84% efficiency at 9k processors for 40M particles

- **Timing:**
  - The average of 10 iterations immediately after load balancing
Simulation of a Premixed Flame

Load Balancing in a CFD Application
Temporal Performance of IPLMCFD

- **Unbalanced:** approx. static performance
- **IPLMCFD:** variable performance
  - Load balancing is performed approx. every 2000 iterations
  - Optimal performance immediately after load balancing
  - Performance degrades in time
- **Potential walltime savings afforded by IPLMCFD for this example:**

\[ T_{\text{Unbalanced}} - T_{\text{IPLMCFD}} = 30 \text{ hours} \]
Cost of Repartitioning

• **Naïve approach:**
  • Immediately before load-balancing checkpoint the entire simulation
  • Restart the simulation with a new decomposition
  • Costly, involves:
    • Writing to shared filesystem
    • Simulation cleanup
    • Simulation startup
    • Reading from shared filesystem
  • Does not scale
  • $O(10^2 – 10^3)$ iterations in cost

• **Optimal approach:**
  • Repartitioning should be handled in memory
  • The new partition is aware of the previous partition, thus minimal data movement and interruption
Zoltan

- “A toolkit of parallel combinatorial algorithms for unstructured and/or adaptive computations”.
- Sandia-OSU collaboration since 2000.
- Part of Trilinos package.
- Zoltan2 project in C++.

Dynamic load balancing
Parallel repartitioning
Data migration tools
Distributed data directories
Unstructured communication
Dynamic memory management
● Zoltan’s callback function interface.

● Methodology:
  ❖ Atomic unit → cell (irregular subdomains).
  ❖ Data registration → number of objects, object weights.
  ❖ Graph management → number of edges, edge weights.
  ❖ Migration → pack/unpack functions.
  ❖ Load balancing → partition, repartition, refinement.
  ❖ Global information → distributed data directory.
Goal: fully exploit Charm++ features.

Methodology:
- Atomic unit $\rightarrow$ subdomain (regular subdomains).
- Containing class $\rightarrow$ 3D chare array.
- Process-based data $\rightarrow$ chare group.
- Communication $\rightarrow$ outermost level.
- Structured control flow $\rightarrow$ Structured Dagger.
- Migration $\rightarrow$ PUP methods.
Partially Stirred Reactor (PaSR)

- **Parameters:**
  - IC: Stoichiometric mixture of methane & air reacted until equilibrium ($T \approx 2230$ K)
  - Simulation duration: $t_{end} = 10 \tau_{res}$

- **Realizability:**
  - Lower bound, no mixing
  - Upper bound, perfectly stirred

![Diagram of partially stirred reactor](image)

![Graph showing temperature over time](image)
Dynamic Load-Balancing

Static Partition

Dynamic Partitioning

Load Balancing in a CFD Application
Strong Scaling

- Parameters:
  - 10,000 particles
  - Chemistry: 9 species, 5-step

- Timings over the entire simulation (Stampede)
  - The Zoltan and Charm++ timings include all overhead associated with repartitioning and data migration

ZOLTAN

Charm++
## Programming Effort

<table>
<thead>
<tr>
<th></th>
<th>Zoltan IPLMCFD</th>
<th>Charm++ IPLMCFD</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Startup</strong></td>
<td>39</td>
<td>0</td>
</tr>
<tr>
<td><strong>Object Graph Management</strong></td>
<td>80</td>
<td>0</td>
</tr>
<tr>
<td><strong>Data Migration</strong></td>
<td>427</td>
<td>61</td>
</tr>
<tr>
<td><strong>Load Balancing</strong></td>
<td>40</td>
<td>3</td>
</tr>
</tbody>
</table>

Measured in lines of code (LOC)
Charm++ Wishlist

- MPI → Charm++ migration guide:
  - Instructions on using Charm++ with build systems.
  - Translating common MPI programming patterns.
  - Dealing with communication operations.
  - Highlighting opportunities for improvement.
- Parallel I/O documentation.
- Accelerator programming documentation.
Conclusions

- Competitive performance between Zoltan and Charm++ for adaptive simulations of turbulent reactive flows.
- Charm++ alleviates programming effort of infrastructure for adaptive computation.

Thank You!

Q&A