Turbulent combustion simulations with adaptive mesh refinement on parallel computers

Ulrich Bielert

Eighth International Conference on Numerical Combustion
Amelia Island, Florida

- Introduction to the combustion code COM3D
- Implementation of AMR on parallel architecture
- First benchmarking results
The combustion code COM3D

- Flow equations: Navier-Stokes
- Turbulence model: k-ε or RNG-k-ε
- Combustion model: extended eddy-break-up model

- Fully 3D
- structured grid with cubic cells
- internal obstacles
- explicit solver

- Implemented on (with simple domain decomposition)
  - Cray J90
  - IBM SP2
  - Linux Cluster
  - Siemens Fujitsu VPP

- Models are validated against experiments
- extended experience with this code
Typical applications (validation)

- FZK 12m tube
  \(H_2\)-steam-air-mixtures
  \(H_2\)-air-mixtures with additional inert gases

- RUT tunnel
  \(H_2\)-steam-air-mixtures
Typical applications

- Reactor-Containment
  complex geometry with internal venting
  large size ($\approx 70000 \, m^3$)
  inhomogeneous mixtures

- Road tunnel with hydrogen car
  dimension: $120m \times 9m \times 4.5m$
  inhomogeneous mixtures
Adaptive Mesh Refinement (AMR)

- **Purpose:**
  - increase spatial resolution
  - focus computational effort in 'interesting' areas

- **Structure of time step in AMR**

  level 0

  level 1

  level 2

  level 3

  time step
  initial data
  boundary data
  solution update

- **Consequences:**
  Computational effort is spend in small refined areas.
  Domain decomposition is not suitable.
  Load must be balanced on each refinement level.
**Domain decomposition**

- Domain decomposition without AMR
  
  ![Diagram without AMR]
  
  Load is balanced.

- Domain decomposition with AMR
  
  ![Diagram with AMR]
  
  Load is concentrated in only few processes.

- Domain decomposition on each refinement level
  
  ![Diagram with refinement levels]
  
  Load is balanced.
General approach

- Identification of separate tasks:
  - Solution of transport equations on different grids
  - Grid refinement (selection of cells, interpolation,...)
  - Communication between processes
  - I/O, statistics,...
- Use of independent data structures (objects):
  - Well defined communication interface between objects
  - Independent structures on different refinement levels
- MPMD program structure to separate different tasks
  Manager - worker concept
- Take credit from available packages:
  HAMR, Boxlib, KeLP,...
- Adopt (optimize) for structured grids
- Portability: MPI, STL,...
Patch - main data structure

- Patchinfo
  Information about patch: size, owner, communication,...
- Fine grid
  Field data of this patch
- Coarse grid
  Field data on grid of parent patch
- Boundary data
  Additional data from parent patch (on coarse grid)
Manager worker concept

one manager - but many workers

- Worker process
  - manipulates a local list of its own patches
  - field data is only available at process that owns the patch
  - manipulates its data according to requests from manager (e.g. advance solution, refine grid, I/O, ...)
  - sends requests for new patches to manager

- Manager process
  - owns a list with information about all patches
  - has (almost) no knowledge about field data.
  - optimizes distribution of patches to workers
  - selects workers for newly created patches
  - user interface
  - interface to graphics server
Communication

- Communication between manager and worker
  - Manager broadcasts a message to start an action.
  - Manager collects replies from workers that action was performed.

- Communication between workers during a time step.
  For each patch the worker owns it
  - receives boundary data for next time step from parent patch
  - sends updated solution to parent patch
  - exchanges boundary data with peers (multiple times)
  - sends boundary data for next time step to children
  - receives updated solution from children
  Implemented as ‘Mover’ similar to KeLP.

- Communication to create new patches (on each level)
  - Worker send requests to manager
  - Manager distributes new patch distribution to all workers concerned (parents and children)
  - Workers exchange initial data
Benchmark 1: Variation of patch size

- Calculation on IBM SP2 (Power2)
- Manager + 1 worker
- 1 Patch, no refinement
- 10 time steps
- Parallel environment POE

<table>
<thead>
<tr>
<th>Patch size</th>
<th>10x10x10</th>
<th>20x20x20</th>
<th>30x30x30</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inner cells</td>
<td>1000</td>
<td>8000</td>
<td>27000</td>
</tr>
<tr>
<td>Total cells</td>
<td>2744</td>
<td>13824</td>
<td>39304</td>
</tr>
<tr>
<td>CPU time [sec]</td>
<td>54.59</td>
<td>432.31</td>
<td>1384.85</td>
</tr>
<tr>
<td>CPU Manager [sec]</td>
<td>27.31</td>
<td>216.13</td>
<td>692.42</td>
</tr>
<tr>
<td>CPU Worker [sec]</td>
<td>27.28</td>
<td>216.18</td>
<td>692.43</td>
</tr>
<tr>
<td>Manager User</td>
<td>2.6 %</td>
<td>0.6 %</td>
<td>0.009 %</td>
</tr>
<tr>
<td>Manager MPI</td>
<td>97.4 %</td>
<td>99.4 %</td>
<td>99.91 %</td>
</tr>
<tr>
<td>Worker User</td>
<td>86.4 %</td>
<td>97.6 %</td>
<td>99.5 %</td>
</tr>
<tr>
<td>Worker MPI</td>
<td>13.6 %</td>
<td>2.4 %</td>
<td>0.5 %</td>
</tr>
</tbody>
</table>

- Manager spends most time waiting for messages.
- Patches smaller than 20x20x20 have to much communication
- Large Variation in execution times for individual routines
**Benchmark 2: Variation of patch distribution**

- 8 patches with 20x20x20 inner cells
- 100 time steps
- output every 10 time steps
- Batch execution under LoadLeveler
- 8 different configurations

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Benchmark 2: Runtime measurements

Test with 8 patches 20x20x20 on SP2 under LoadLeveler

- Times were measured with system calls to 'times' and 'gettimeofday'
- Execution time of whole program was measured
- Execution time for single operations (e.g. one time step) was measured
Benchmark 2: Correlation of execution times for one time step

Test with 8 patches 20x20x20 on SP2 under LoadLeveler
Benchmark 2: Execution times as function of time

Test with 8 patches 20x20x20 on SP2 under LoadLeveler
Benchmark 2: Corrected correlation of execution times for one time step

Test with 8 patches 20x20x20 on SP2 under LoadLeveler
Linear correlation coefficient of user time
Linear correlation coefficient of wall clock time
Summary

- Data and program structures have been developed to implement AMR into an existing parallel combustion code.
- Adaptive grid refinement is not yet implemented.
- Calculations with static grid arrangements are under way.
- Load balancing algorithms are not yet implemented.
- Program structure allows for optimization algorithms in load balancing.
- Benchmark calculations give hints to achievable balance.