Results of Ateles and Musubi Code Analyses

Harald Klimach

harald.klimach@uni-siegen.de

University of Siegen
Our Group

• Simulation Techniques and Scientific Computing at University of Siegen
• Head: Sabine Roller
• Part of the engineering department
• Looking into large scale simulations, mainly in the field of computational fluid dynamics
About our Software (APES)
About our Software

• Started development in 2011 at the German Research School for Simulation Sciences in Aachen
• Set out to overcome scalability limitations we encountered with previous tools
• Development in modern Fortran (mainly F95, but some F2003 features required)
• All in all somewhat more than 100,000 lines of code. Definitely smaller than 200k.
About TreELM

• Basis: octree mesh representation
• Partitioning by space-filling curve
  – Simplistic mesh representation on disk, easy to read on a distributed parallel system
  – Little amount of meta data (number of elements, definition of root node) can be read in by single process and broadcasted
  – All other data can be read independently
• Distributed parallel neighbor identification in locally refined meshes
Mesh Generation (Seeder)

• Creating Octree Meshes
  – With specific data format
  – Allow for special boundary treatment in
    • Lattice-Boltzmann (q-values)
    • Discontinuous Galerkin (penalty terms)

• Serial for up to 2 billion elements
• Parallel version for larger meshes

• Solver internal generation of simple meshes
Post-Processing (Harvester)

• Generating visualization files  
  – (VTK)
• Out of binary dumps by the solver
• Solver writes binary data via MPI-IO to single file in solver-specific format
• Postprocessing separated to allow processing on different machines
• Allows extraction of parts and computation of derived quantities
Configuration (Aotus)

- Quite generic Lua binding
- Utilizes ISO-C interface
- Some abstractions to ease use as configuration for Fortran
- Allows for user-defined functions
- Parameter relations can be explicitly expressed in the configuration
- Read by single process and broadcasted
Musubi

• Lattice-Boltzmann Solver
  – Simple loop operation
  – Explicit scheme for incompressible flows
• Implements various Kernels
• Unstructured neighbor handling (from treelm), sparse matrix approach
• Allows for multi-species simulations
• Allows for locally refined simulations
Ateles

- Discontinuous Galerkin
  - High order (8 and higher)
  - Modal/nodal representations
- For compressible flows
- Also implements other hyperbolic systems, like Maxwell equations
- Cubical elements
- Geometry by penalization
On the Work by POP

• Code analyzed and improved by:
  – José Gracia
  – Christoph Niethammer
  – Stephan Walter
  – Anastasia Shamakina

• Many thanks to those kind colleagues at HLRS
Systems used?

- Ateles was analyzed on the Stuttgart system Hazel Hen

- Musubi was investigated on our local university cluster Horus
  - Tools were installed in cooperation between university computing center and POP staff
  - Was not involved in that as a user
Ateles Reports

- There have been three reports on Ateles
  - The initial performance audit
  - A more detailed performance plan
  - A proof of concept implementing some suggestions
Ateles Setup

- POP asked for a realistic testcase
- The setup I provided them was the 3D Jet with local refinement, sponge and covolume
- This includes basically all relevant features

- Never looked into the performance of this ourselves before
Setup Illustration

Cut in the xy-plane

View from the left (yz-plane)
Bad Strong Scaling ...

![Graph showing speed up vs number of MPI processes for Ateles with 2288 and 16896 elements compared to ideal strong scalability.]
Bad Weak Scaling

![Graph showing weak scaling performance](image)
Reason: Large Load Imbalances
Code Analysis

- Load imbalances due to non-optimized code in covolume:
  - Posofmodgcoeffqtens: Index lookup in the polynomials
    - Had been replaced in other parts of the code but not in covolume -> gone now completely
- Facevalleftbndans and facevalrightbndans: functions just returning 1 or -1, easily inlined
- Tem_isnan: check interval
Lua Functions...

• Flu_* and aot_* functions showed up as contributing factors to the load imbalance
  – Can be overcome by implementing the respective functions as predefined functions in Fortran itself.
Recommendations

• Avoid many small function calls.
  – Kind of done already by Peter

• Reduce the amount of duplicate computation, or specialise code at compile time.
  – Harder to achieve, but probably due to no one looking into the covolume stuff for optimization so far

• Reduce load imbalance by improving static domain decomposition.
Performance Plan

• More detailed look into load imbalances
• Vampir traces
• And Scalasca analysis
Intermediate result

• Imbalances manifest at two points:
  – Allreduce after each timestep (actually controllable with check interval)
  – Waitall after each substep (neighbor exchange)
• Further investigation of the imbalances at the waitall as more severe.
Observation

• Load imbalances mainly due to covolume interpolation at refinement boundaries
  – Note: These especially made use of many small function calls (low efficiency)

• The computational load imbalance clearly is of main interest.
• Nevertheless, also the communication has been investigated.
Communication Matrix

Number of messages

Sender

Master_thread_0
Master_thread_1
Master_thread_2
Master_thread_3
Master_thread_4
Master_thread_5
Master_thread_6
Master_thread_7
Master_thread_8
Master_thread_9
Master_thread_10
Master_thread_11
Master_thread_12
Master_thread_13
Master_thread_14
Master_thread_15
Master_thread_16
Master_thread_17
Master_thread_18
Master_thread_19
Master_thread_20
Master_thread_21
Master_thread_22
Master_thread_23

Receiver

Master_thread_0
Master_thread_1
Master_thread_2
Master_thread_3
Master_thread_4
Master_thread_5
Master_thread_6
Master_thread_7
Master_thread_8
Master_thread_9
Master_thread_10
Master_thread_11
Master_thread_12
Master_thread_13
Master_thread_14
Master_thread_15
Master_thread_16
Master_thread_17
Master_thread_18
Master_thread_19
Master_thread_20
Master_thread_21
Master_thread_22
Master_thread_23

APES Code Analyses

19.05.17
## Surprise: 0-sized messages

<table>
<thead>
<tr>
<th>Number of Messages per Message Size</th>
<th>12.0 k</th>
<th>10.5 k</th>
<th>9.0 k</th>
<th>7.5 k</th>
<th>6.0 k</th>
<th>4.5 k</th>
<th>3.0 k</th>
<th>1.5 k</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>12,496 (34.73%)</td>
<td>2,210</td>
<td>1,400</td>
<td>2,070</td>
<td>1,199</td>
<td>1,107</td>
<td>1,066</td>
<td>1,003</td>
<td>880</td>
<td>82</td>
</tr>
</tbody>
</table>

*Note: All values are in bytes.*
Proof of Concept

• Look into serial performance and implementation of some of the suggestions from the Audit

• Improving code vectorization

• Brief evaluation on SX-ACE
Main Factors

• Inlining posofmodgqtens reduced the number of function calls by nearly 97 %
• Reduces the execution time on Hazel Hen to 94 % of the original, not inlined, code version
• Replace divisions by multiplications, further reduction to 72 % of the inlined version.
SX-ACE

• Bad vectorization because sxmpif03 often vectorizes over nScalars
• In the meantime improved for various code paths

• Strong memory-conflicts for orders of powers of two (should be avoided, padding may need to be introduced)!
Musubi

• Testcase was flow around a sphere
  – With involved boundaries (q-values, inflow and outflow)
  – No local refinements
  – Excluded I/O (only reading of mesh in the initialization)

• Run on Horus
  – Intel Xeon X5600 6-core CPU, 2 sockets per node
  – Infiniband Interconnect
Main Focus: Scalability (Strong Scaling)

Figure 1: Execution timeline of an 8-iteration block within the main computational phase for a run with 12 MPI ranks. The timelines of the 12 MPI ranks/cores are stacked vertically; time runs from left to right. The cores are either executing user code (green color) which is considered useful, or code in the MPI runtime system (red color), which is considered non-useful parallelisation overhead. At the end of each iteration, we observe a phase of non-blocking MPI communication. Iterations are grouped in blocks of 8 iterations. These blocks are separated by a blocking MPI collective (vertical black lines) which leads to a synchronisation across all processes.

Figure 2: Strong scaling for 12 to 192 MPI ranks. The green line marks ideal scaling, i.e. scaling efficiency 100%, while the black line corresponds to a scaling efficiency of 80%, which is considered the lowest acceptable value.
Surprisingly Bad Scaling

- Still acceptable, but...
- Expected to see a better scaling, as we usually observe nearly perfect scalability with these kind of testcases

- Turns out, the boundary conditions cause a larger load-imbalance than expected
Machine Seems to Have Bad Days...

Figure 3: MLUPs strong scaling for 12 to 192 ranks and always 12 ranks per node. For more than eight nodes a significant performance degeneration can occur. The green line marks linear scaling between the execution time and the used ranks/nodes. The black line marks the 80% scaling boundary that defines the lower limit to define that an application is scaling.

Figure 4: Constant very low Instructions Per Second count on three out of 16 CPU's in four nodes. The IPS is shown in a colour scale, with blue=0.75 Giga-IPS and red=1.45 Giga-IPS. The grey parts are related to a higher or lower IPS, that is mainly related to MPI functions. The IPS can degenerate by up to a factor of two what corresponds to the degenerated MLUPs for this run compared to the expected performance from other runs.
Observations from the Machine (Horus)

• Significant performance variability
  – Clock speed degeneration, probably due to *ondemand* performance governor, mainly in MPI IO and MPI_Ssend
  – Day to day variation of overall instructions per second
  – Noise on the interconnect
Recommendations for Musubi

• Imbalances by boundaries should be taken care
  – We have dynamic load-balancing now
• Using non-temporal stores
  – Implemented now, improved performance by around 50%
• Avoid indirection
  – Can't do that
• Overlap communication and computation
  – Requires major effort
Feedback

• The audits by POP were very helpful to us and revealed various issues, we weren't even aware of before

• Initial communication could have been a little more active (things like the inlining of small functions could have been easily resolved early on)

• Would be happy to have code modifications on a branch or somewhere
The Reports are Great

- I love the neat and well explained reports
- I can give them to the developers along with lessons learned
- Direct recommendations on Do's and Don'ts for everybody

- Again some earlier communication with preliminary results is even more productive.
Thank you!