High-Performance Data Analysis with the Helmholtz Analytics Toolkit

Martin Siggel, Debus Charlotte, Alexander Rüttgers, Kai Krajsek, Philipp Knechtges, Markus Götz, Claudia Comito, Björn Hagemeier

German Aerospace Center (DLR)
High-Performance Computing

CSAI, June 14. 2019, University of Jyväskylä
How to perform data analytics on huge datasets?
HeAT!

- HeAT = Helmholtz Analytics Toolkit

- Python framework for parallel, distributed data analytics and machine learning

- Developed within the Helmholtz Analytics Framework Project since 2018

- AIM: Bridge data analytics and high-performance computing

- Open Source licensed, MIT

  [helmholtz-analytics/heat](https://github.com/helmholtz-analytics/heat)
How we started HeAT:
The Helmholtz Analytics Framework (HAF) Project

• Joint project of all 6 Helmholtz centers

• Goal: foster data analytics methods and tools within Helmholtz federation.

• Scope:
  • Development of domain-specific data analysis techniques
  • Co-design between domain scientists and information experts
Motivation: HAF applications

Earth System Modelling

SEVIRI Satellite Images – Near Real Time

Research with Photons

Neuroscience

Aeronautics and Aerodynamics

Structural Biology
Motivation: HAF methods + algorithms

- Clustering
  - k-means, mean shift clustering
- Uncertainty quantification
  - Ensemble methods
- Dimension reduction
  - Autoencoder, reduced order models
- Feature learning
  - Image descriptors, autoencoder
- Data assimilation
  - Kalman filter, 4Dvar, particle filter/smooother
- Classification/Regression
  - Random forest, CNN, SVM
- Modelling
  - Fiber tractography, point processes
- Optimization techniques
  - L-BFGS, simulated annealing
- Hyper-parameter optimization
  - Evidence framework, grid search
- Interpolation
  - Radial basis function, Kriging
- Data mining
  - Frequent item set mining
Greatest Common Denominator?

Machine Learning

= Data

+ Numerical Linear Algebra

https://xkcd.com/1838/
Big Data/Deep Learning Libraries

Big Data
- hadoop
- DISCO
- mahout
- Spark
- Apache Storm
- DASK

Deep Learning
- PyTorch
- TensorFlow
- Chainer
- Keras
- PaddlePaddle
- H2O.ai
- mxnet
- CNTK
- ArrayFire
- dy/net

> High-Performance Data Analysis with the Helmholtz Analytics Toolkit > Martin Siggel > 14.06.2019
**Scope**

- Facilitating applications of HAF in their work
- Bringing HPC and Machine Learning / Data Analytics closer together
- Ease of use

**Design**

- PyTorch
  - Tensor Linear Algebra
  - Automatic Differentiation
  - NumPy-like interface
  - GPU support
- mpi4py
  - Distributed Parallelism (MPI)
- HeAT
  - k-means
  - SVM
  - Deep Learning
  - And more machine learning algorithms
Which framework could be basis for HeAT?

Evaluation criteria

- Feature completeness
- Compute performance ➔ Benchmarks required!
- Ease of development
Which technology stack to use?
Feature completeness

<table>
<thead>
<tr>
<th>Framework</th>
<th>GPU</th>
<th>MPI</th>
<th>AD</th>
<th>LA</th>
<th>nD Tensors</th>
<th>SVD</th>
<th>Dist. tens</th>
</tr>
</thead>
<tbody>
<tr>
<td>PyTorch</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>TensorFlow</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>(X)*</td>
</tr>
<tr>
<td>MXNet</td>
<td>X</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>Arrayfire</td>
<td>X</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>-</td>
</tr>
</tbody>
</table>

- Completeness: PyTorch and TensorFlow
- Ease of implementation and usage: PyTorch and MXNet

*Note: no support of distributed data in TensorFlow in 2018, but today there is first support!
Which technology stack to use? Compute performance

- Implemented 4 benchmark methods in all frameworks (PyTorch, Tensorflow, MXNet, ArrayFire)
  - K-means
  - Self-Organizing Maps (SOM)
  - Artificial Neural Networks (ANN)
  - Support Vector Machines (SVM)
- Example: ResNet Batch Inference (32 Images) on NVIDIA K80 GPU@JURECA
- Similar result for other ML Methods (e.g. k-means)
- Benchmarking is on-going effort:
  - PyTorch seems to be performing best
Distributed tensors
NumPy

<table>
<thead>
<tr>
<th>Runs on</th>
<th>Data structure</th>
<th>Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ND-Tensor</td>
<td>- Elementwise operations</td>
</tr>
<tr>
<td><img src="image" alt="CPU" /></td>
<td><img src="image" alt="Tensor" /></td>
<td>- Slicing</td>
</tr>
<tr>
<td></td>
<td>shape: (4, 3, 2)</td>
<td>- Matrix operations</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Reduction</td>
</tr>
</tbody>
</table>
PyTorch

Runs on

Data structure

ND-Tensor

Operations

- Elementwise operations
- Slicing
- Matrix operations
- Reduction
- Automatic differentiation
HeAT

- Operations
  - Elementwise operations
  - Slicing
  - Matrix operations
  - Reduction
  - Automatic differentiation

Runs on
- CPU
- GPU
- MPI

Data structure
- ND-Tensor

Shape: (4, 3, 2)
Data Distribution

Example:

```python
import heat as ht
# construct a range tensor
>>> range_data = ht.arange(6, split=1)

>>> range_data.mean()
2.5
>>> range_data.argmax()
5
```
What has been done so far?

- The core technology has been identified
- Implementation of a distributed parallel tensor core framework
- NumPy-compatible core functionality
- Some linear algebra routines
- Parallel data I/O via HDF 5 and NETCDF
- A first implementation of the k-means algorithm is available
Example: k-means

- Find k data clusters
- Minimization of
  \[ \arg\ min_c \sum_{i=1}^{k} \sum_{x \in C_i} ||x - \mu_i||^2 \]
- NP-hard problem, many local minima!
- Basic k-means algorithm (heuristic):
  1. Choose k initial centroids \( \mu_1 \ldots \mu_k \)
  2. For each point \( x \) calculate Euclidean distance to all centroids
  3. Assign each point to its closest centroid
  4. Estimate new centroid as mean of points
  5. Go to 2. until convergence
Example: k-means

2. For each point calculate distance to centroids
3. Assign point to closest centroid

```python
>>> data.shape
(18, 2, 1)

>>> centroids.shape
(1, 2, 2)
```

**Numpy vs. HeAT**

```python
>>> distances = ((data - centroids) ** 2).sum(axis=1, keepdims=True)
>>> matching_centroids = np.expand_dims(distances.argmin(axis=2), axis=2)
```

```python
>>> distances = ((data - centroids) ** 2).sum(axis=1)
>>> matching_centroids = distances.argmin(axis=2)
```

```python
>>> matching_centroids.shape
(18, 1, 1)
```
Example: k-means

Numpy vs. HeAT

4. Select data points that are assigned to the current cluster

```python
>>> matching_centroids.shape
(18, 1, 1)
```

```python
>>> for i in range(self.n_clusters):
    selection = (matching_centroids == i).astype(np.int64)
```

```python
>>> for i in range(self.n_clusters):
    selection = (matching_centroids == i).astype(ht.int64)
```

```python
>>> selection.shape
(18, 1, 1)
```
Example: k-means

Numpy vs. HeAT

4. Compute **new centroid positions** by averaging

```python
>>> matching_centroids.shape
(18, 1, 1)
```

```python
>>> data.shape
(18, 2, 1)
```

**NumPy**

```python
>>> for i in range(self.n_clusters):
...     new_centroids[:, :, i:i+1] = ((data*selection).sum(axis=0, keepdims=True) /
...     selection.sum(axis=0).clip(1.0, sys.maxsize))
```

**HeAT**

```python
>>> for i in range(self.n_clusters):
...     new_centroids[:, :, i:i+1] = ((data*selection).sum(axis=0) /
...     selection.sum(axis=0).clip(1.0, sys.maxsize))
```

```python
>>> new_centroids.shape
(1, 2, 2)
```
A real world example:
Rocket engine combustion analysis

- **Goal**: Cost reduction of rocket engines, be competitive with e.g. Space-X

**Traditional rocket engine:**

- 2 Pumps transporting fluid fuel and oxidizer at very high pressure and flow
- **Advantages**
  - Burning rate can be controlled precisely
- **Disadvantages**
  - Pumps are mechanically very complex
  - Expensive
A real world example:
Rocket engine combustion analysis

- **Goal**: Cost reduction of rocket engines, be competitive with e.g. Space-X

**Solid propellant rocket engine**

- Fuel and oxidizer are mixed in solid form
- Advantage
  - Cheap
- Disadvantage
  - Burning rate can not be varied during flight
A real world example: Rocket engine combustion analysis

- **Goal:** Cost reduction of rocket engines, be competitive with e.g. Space-X

**Hybrid rocket engine**

- Pressurized fluid oxidizer
- Solid fuel
- A valve controls, how much oxidizer gets into the combustion chamber
- Advantages
  - Cheap
  - Controllable
A real world example: Rocket engine combustion analysis

• **Goal:** Finding a good design for a hybrid rocket engine

• Hundreds of experiments
• Each experiment 3s video data, ~30000 images/ 8 GB data
• Clustering analysis of combustion experiments
• Identification of different burning phases
• Challenges:
  • Number of clusters unknown a priori
  • High memory consumption and computation demand

➢ **Use HeAT’s k-means for distributed clustering**
  • Each image is a sample in a high-dimensional space
A real world example:
Resulting Clusters, k = 7
Time-dependency of centroids

- Centroid 1
  - $t = 0.5s$
- Centroid 5
  - $t = 1.5s$
- Centroid 6
  - $t = 3.2s$
A real world example:
Results, $k = 7$, Cluster assignment
A real world example: Computational Performance

- Hybrid shared memory + distributed memory setting
- CPU only
- Variation of 1 … 16 MPI total ranks
- Variation of 1 … 3 local threads per process
- Strong scaling analysis: How does the computing time reduce with number of ranks?
- First results look promising, testing on larger systems + GPU necessary
Future Developments

• Completion of neural deep network support, including convolutions and automatic differentiation

• Support for sparse matrices

• In kernel methods (e.g. SVMs), linear system has to be solved with distance matrix

\[ K = \begin{pmatrix}
    k_{11} & k_{12} & \cdots \\
    k_{21} & k_{22} & \cdots \\
    \vdots & \vdots & \ddots
\end{pmatrix} \quad k_{ij} = \exp\left(-\gamma \|x_i - x_j\|^2\right) \]

• The \( k_{ij} \) never become zero, but can be arbitrarily close to \( O(n^2) \)

• Could one not partially approximate the matrix with low-rank matrices?  \( \rightarrow \) Hierarchical Matrices

• Tensor decompositions to reduce computational complexity

Figure taken from Steffen Börm’s lecture notes „Numerical Methods for Non-Local Operators“
Acknowledgments

This work is supported by the Helmholtz Association Initiative and Networking Fund under project number ZT-I-0003.

Thanks for listening!

Contact

Dr. Martin Siggel  
Martin.Siggel@dlr.de
Dr. Charlotte Debus
Dr. Philipp Knechtges

https://github.com/helmholtz-analytics

Scan me
Thanks for listening.

Questions?