Accelerating Spark MLlib and DataFrame with Vector Processor “SX-Aurora TSUBASA”

Takeo Hosomi
Takuya Araki, Ph.D.

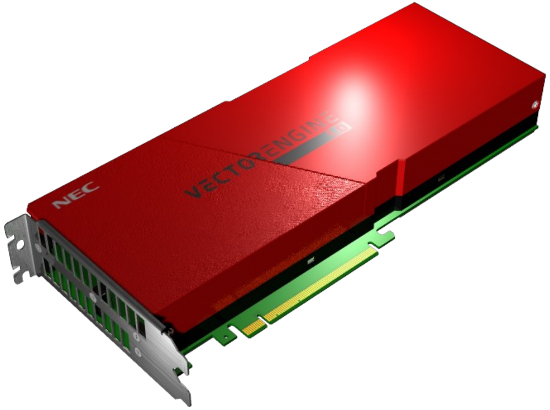
Data Science Research Laboratories
NEC Corporation

Dr. Erich Focht
Senior Manager R&D
NEC Deutschland GmbH

Summary

NEC released new vector processor SX-Aurora TSUBASA

- Different characteristics than GPGPU:
  - Larger memory and higher memory bandwidth
  - Compatible with standard programming languages



Vector processor evolved from HPC

- Optimized for unified Big Data analytics
- Especially suitable for statistical ML

Packaged with machine learning middleware in C++/MPI

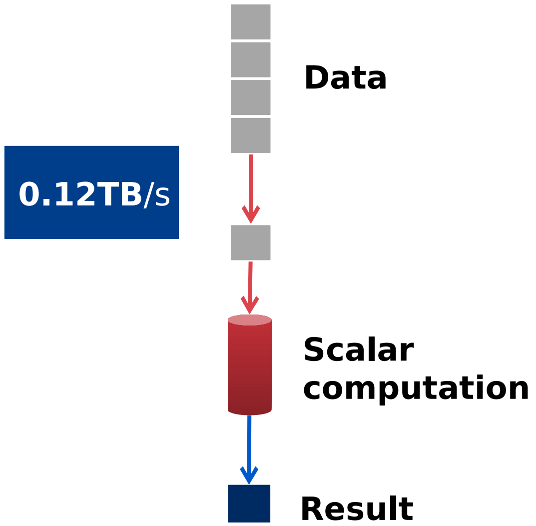
- Distributed and vectorized implementation
- Adapts Apache Spark APIs
- ~100x faster than Spark on x86

What is a Vector Processor ?

Processes many elements with one instruction, which is supported by large memory bandwidth

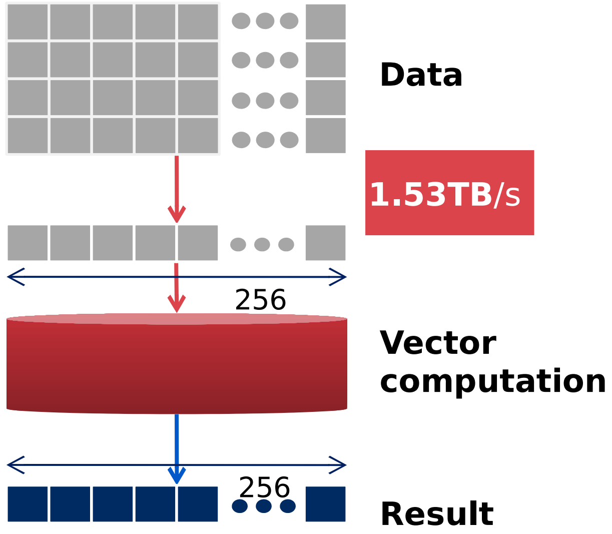
Scalar processor

Unit of computation is small
Suitable for web server, etc.



Vector processor

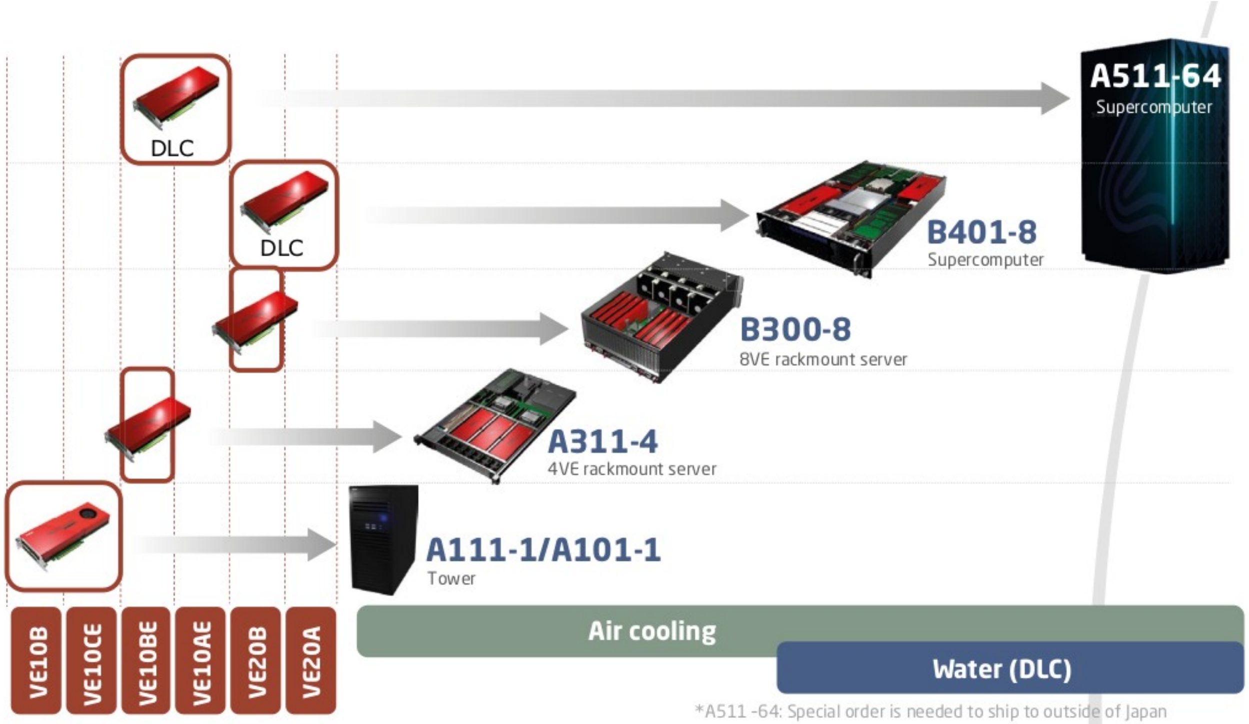
Computes many elements at once
Suitable for simulation, AI, Big Data, etc.



New Vector Processor System “SX-Aurora TSUBASA”

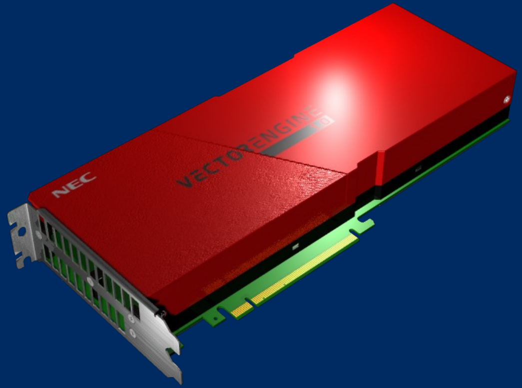
SX-ACE

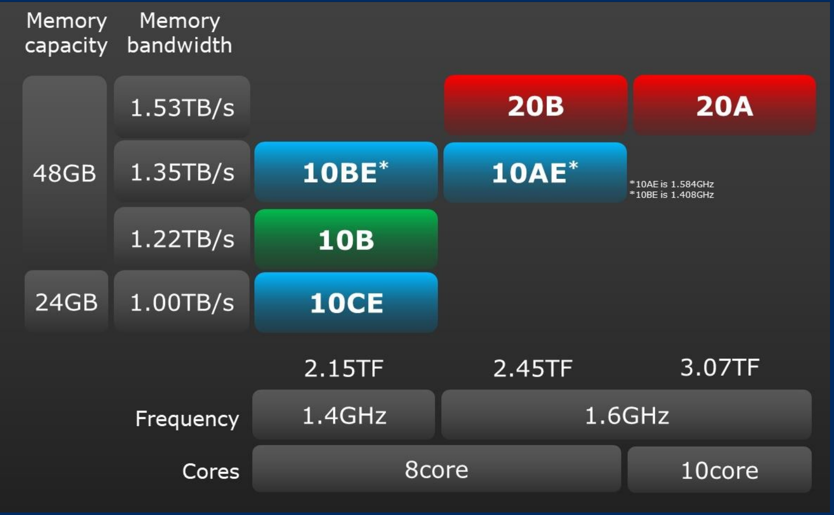
(Supercomputer)



Downsized super computer:
Can be used as an accelerator for Big Data and AI

On-card Vector Processor (Vector Engine)

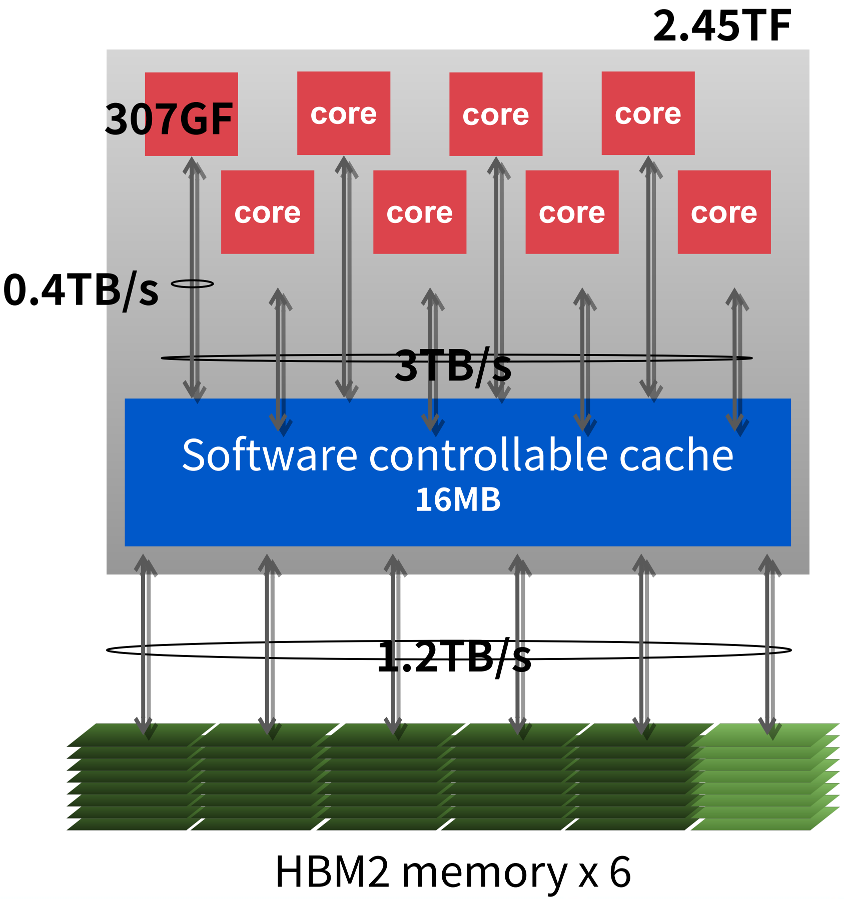




- NEC-designed vector processor
- PCIe card implementation
- 8-10 cores / processor
- 6.14TF performance (single precision)
- 1.53TB/s memory bandwidth, 48GB memory
- Standard programing interface (C/C++/Fortran)

Processor Specifications

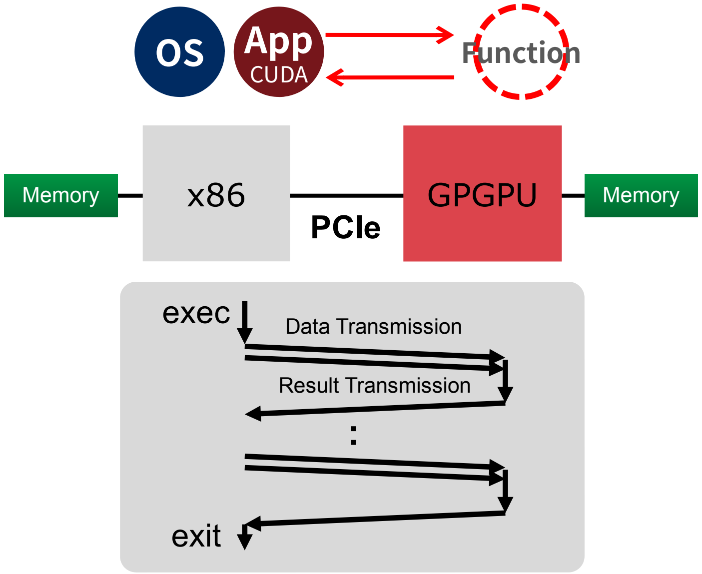
VE1.0 Specification	
vector length	256 words (16k bits)
cores/CPU	8
frequency	1.6GHz
core performance	307GF(DP) 614GF(SP)
CPU performance	2.45TF(DP) 4.91TF(SP)
cache capacity	16MB shared
Memory bandwidth	1.2TB/s
Memory capacity	48GB



GPGPU and Vector Engine Execution Models

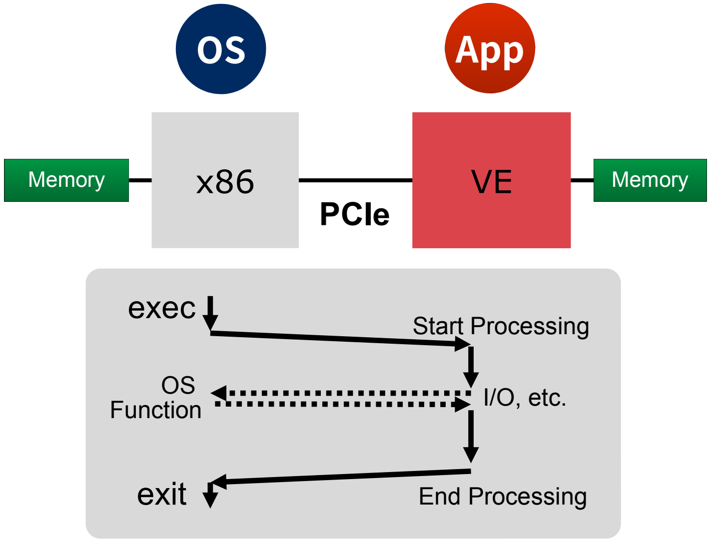
GPGPU: Offloading Model

Parts of App. are executed on GPGPU



Vector Engine: Native Model

Whole App. is executed on VE



Advantage of Native Model

- ✓ Can reduce the data transfer between x86 and Vector Engine

Usability

Programing Environment

```
$ vi sample.c
$ ncc sample.c
```

Vector Cross Compiler

automatic vectorization, automatic parallelization

OS:	RedHat Linux, Cent OS
Fortran:	F2003, F2008(partially)
C:	C11
C++:	C++14
OpenMP:	OpenMP4.5
MPI:	MPI3.1
LLVM-VE	w/ intrinsics, RV, OMP Tgt, experimental

Execution Environment

```
$ ve_exec ./a.out
```

execution

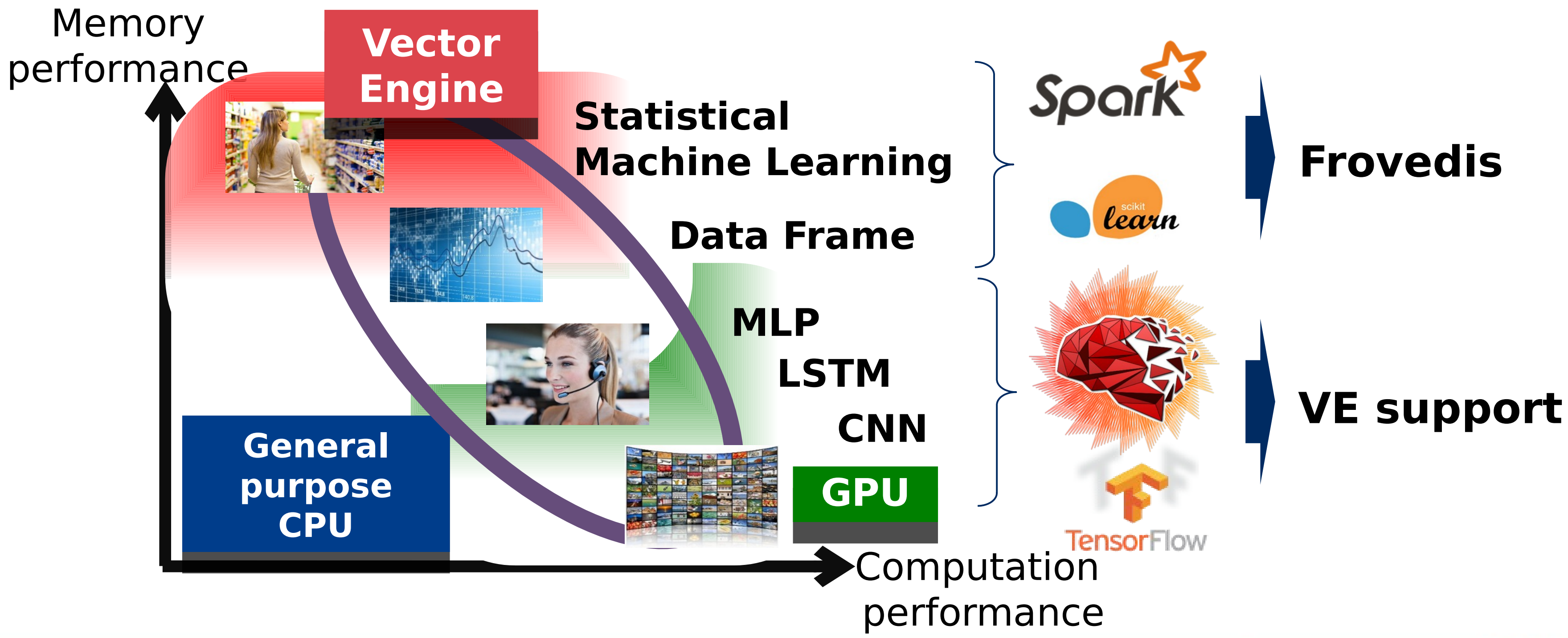
Why Vector Engine?

Can accelerate memory intensive workloads

- ✓ High memory bandwidth and large memory capacity
- ✓ Supports native execution model
- ✓ Standard programing model
- ✓ Scale to multiple vector processors
  - Direct data transfer among multiple vector processors through PCIe and InfiniBand

AI/ML on SX-Aurora TSUBASA

- AI/ML that requires memory performance can be well accelerated
- Provide frameworks for easy utilization



Frovedis:

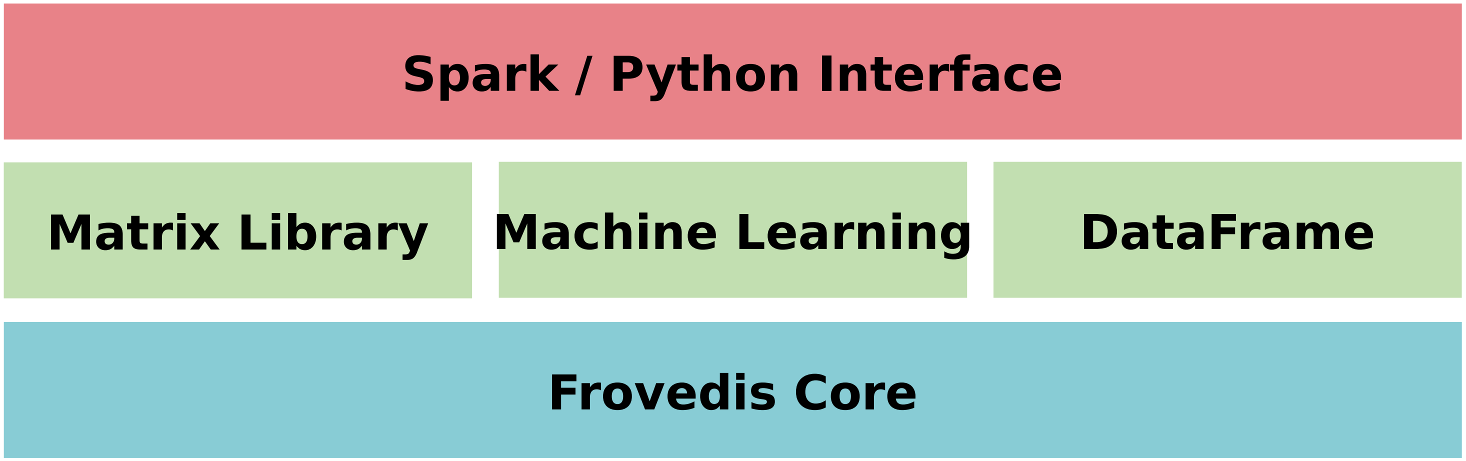
Framework of vectorized and distributed data analytics

\Orchestrating a brighter world

Frovedis: FRamework Of VEctorized and DIStributed data analytics

- C++ framework similar to Spark
  - Supports Spark/Python interface
- MPI is used for high performance communication
- Optimized for SX-Aurora TSUBASA (also works on x86)

Open Source! github.com/frovedis



Frovedis Core

- Provides Spark core-like functionalities (e.g. map, reduce)
  - Internally uses MPI to implement distributed processing
  - Inherently supports multiple cards/servers
- Users need not be aware of MPI to write distributed processing code
  - Write functions in C++
  - Provide functions to the framework to run them in parallel
- Example: double each element of distributed variable

```
int two_times(int i) {return i * 2;}
int main(...) {
  ...
  dvector<int> r = d1.map(two_times);
}
```

distributed variable

run “two_times” in parallel

Complete Sample Program (1/2)

- Scatter a vector; double each element; then gather

```
#include <frovedis.hpp>
using namespace frovedis;

int two_times(int i) {return i*2;}

int main(int argc, char* argv[]) {
  use_frovedis use(argc, argv);

  std::vector<int> v = {1,2,3,4,5,6,7,8};
  dvector<int> d1 = make_dvector_scatter(v);
  dvector<int> d2 = d1.map(two_times);
  std::vector<int> r = d2.gather();
}
```

initialization

scatter to create dvector

gather to std::vector

- Do not have to be aware of MPI (SPMD programming style)
  - Looks more like a sequential program

Complete Sample Program (2/2)

Works as an MPI program

```
#include <frovedis.hpp>
using namespace frovedis;

int two_times(int i) {return i*2;}

int main(int argc, char* argv[]) {
  use_frovedis use(argc, argv);

  std::vector<int> v = {1,2,3,4,5,6,7,8};
  dvector<int> d1 = make_dvector_scatter(v);
  dvector<int> d2 = d1.map(two_times);
  std::vector<int> r = d2.gather();
}
```

MPI_Init is called in the constructor, then branch:
- **rank 0: execute the below statements**
- **rank 1-N: wait for RPC request from rank 0**

in the destructor of "use", MPI_Finalize is called and send RPC request to rank 1-N to stop the program

rank 0 sends RPC request to rank 1-N to do the work

Matrix Library

- Implemented using Frovedis core and existing MPI libraries[*]

  [*] ScaLAPACK/PBLAS, LAPACK/BLAS, Parallel ARPACK

- Supports dense and sparse matrix of various formats
  - Dense: row-major, column-major, block-cyclic
  - Sparse: CRS, CCS, ELL, JDS, JDS/CRS Hybrid (for better vectorization)
- Provides basic matrix operations and linear algebra
  - Dense: matrix multiply, solve, transpose, etc.
  - Sparse: matrix-vector multiply (SpMV), transpose, etc.

Example

```
blockcyclic_matrix<double> A = X * Y; // mat mul
gesv(A, b);  // solve Ax = b
```

Machine Learning Library

Implemented with Frovedis Core and Matrix Library

- ✓ Supports both dense and sparse data
- ✓ Sparse data support is important in large scale machine learning

Supported algorithms:

- Linear model
  - Logistic Regression
  - Multinominal Logistic Regression
  - Linear Regression
  - Linear SVM
- ALS
- K-means
- Preprocessing
  - SVD, PCA
- Word2vec
- Factorization Machines
- Decision Tree
- Naïve Bayes
- Graph algorithms
  - Shortest Path, PageRank, Connected Components
- Frequent Pattern Mining
- Spectral Clustering
- Hierarchical Clustering
- Latent Dirichlet Allocation
- Deep Learning (MLP, CNN)
- Random Forest
- Gradient Boosting Decision Tree

We will support more!

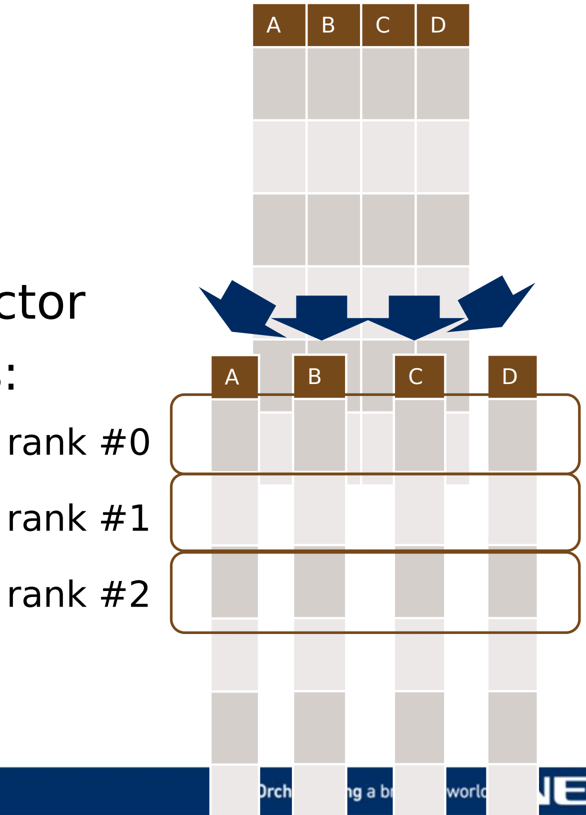
DataFrame

Supports similar interface as Spark DataFrame

- Select, Filter, Sort, Join, Group by/Aggregate
- (SQL interface is not supported yet)

Implemented as distributed column store

- Each column is represented as distributed vector
- Each operation only scans argument columns: other columns are created when necessary (late materialization)



- Reduces size of data to access

Spark / Python Interface

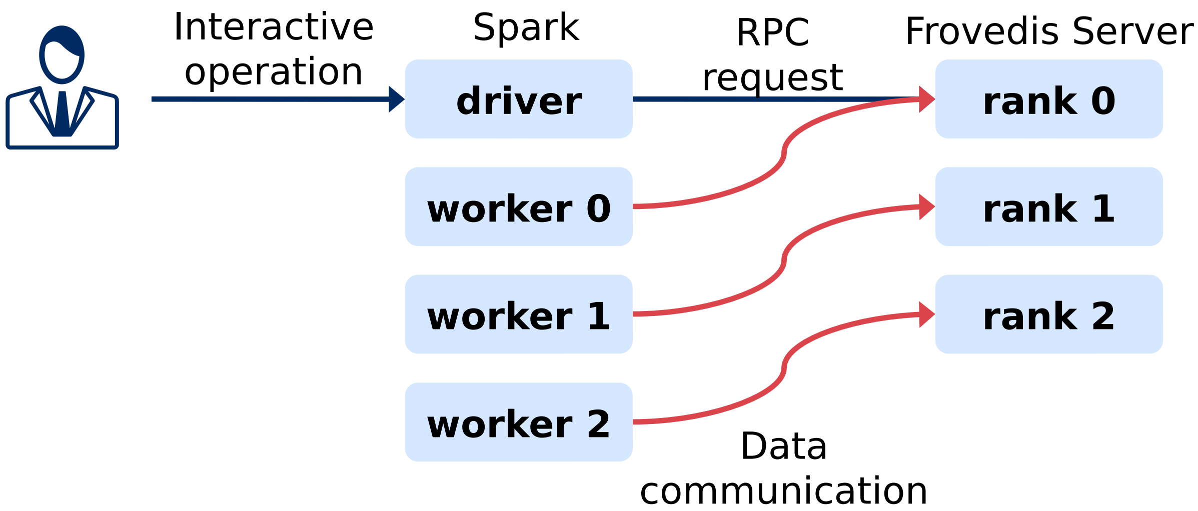
- Writing C++ programs is sometimes tedious, so we created a wrapper interface to Spark
  - Call the framework through the same Spark API
  - Users do not have to be aware of vector hardware

- Implementation: created a server with the functionalities
  - Receives RPC request from Spark and executes ML algorithm, etc.
  - Only pre-built algorithms can be used from Spark

- Other languages can also be supported by this architecture
  - Currently Python is supported (scikit-learn API)

How it works

- Rank 0 of the Frovedis server waits for RPC from driver of Spark
- Data communication is done in parallel
  - All workers/ranks send/receive data in parallel
  - Assuming that the data can fit in the memory of the Frovedis server



Programming Interface

- Provides same interface as the Spark’s MLlib

Original Spark program: logistic regression

```
...
import org.apache.spark.mllib.classification.LogisticRegressionWithSGD
...
val model = LogisticRegressionWithSGD.train(data)
...
```

Change import

```
...
import com.nec.frovedis.mllib.classification.LogisticRegressionWithSGD
...
FrovedisServer.initialize(...)
val model = LogisticRegressionWithSGD.train(data)
FrovedisServer.shut_down()
...
```

Same API (no change)

Start/Stop server

Python (scikit-learn) Interface

Original Python program: logistic regression

```
…
from sklearn.linear_model import LogisticRegression
…
clf = LogisticRegression(…).fit(X, y)
…
```

Change import

```
…
from frovedis.mllib.linear_model import LogisticRegression
…
FrovedisServer.initialize(…)
clf = LogisticRegression(…).fit(X, y)
FrovedisServer.shut_down()
…
```

Same API (no change)

Start/Stop server

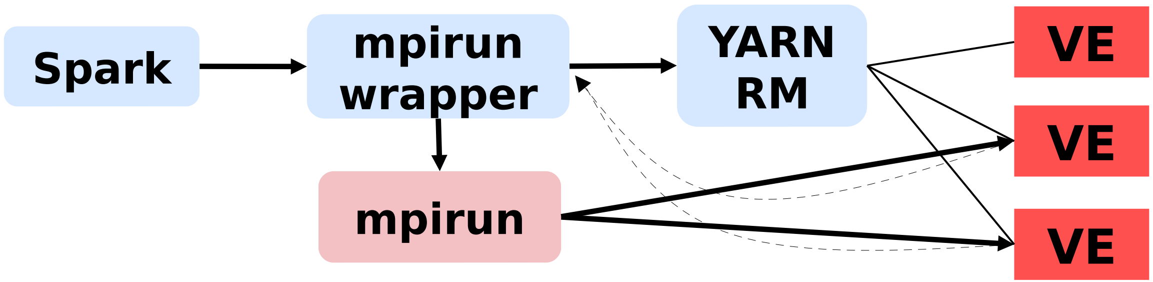
YARN Support

Resource allocation by YARN is also supported

- Implemented in the collaboration with Cloudera (formerly Hortonworks) team

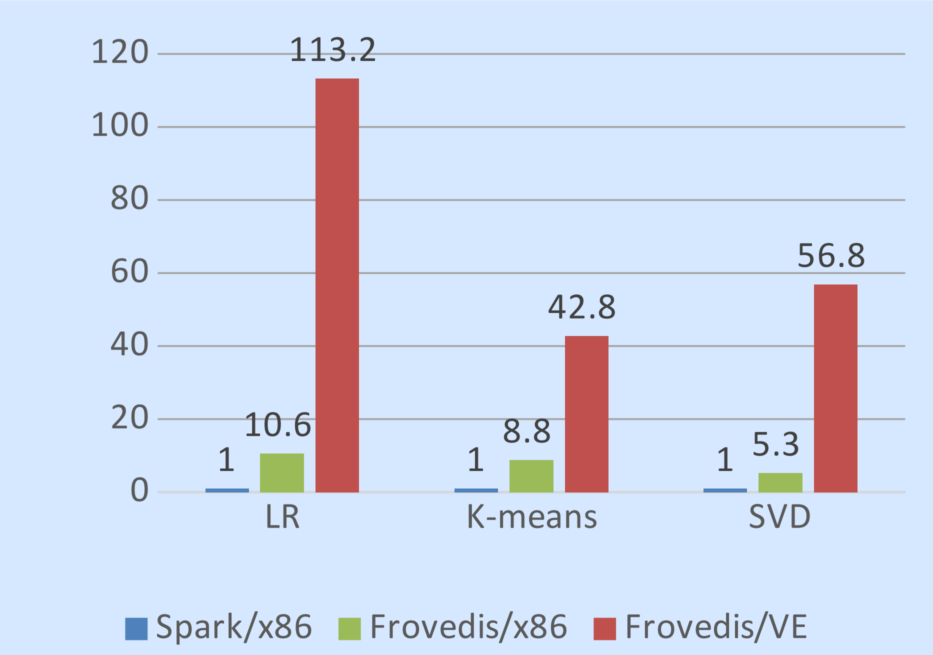
Implementation:

- YARN is modified to support Vector Engine (VE) as resource (like GPU)
- Created a wrapper program of mpirun, which works as YARN client
  - Obtain VE from YARN Resource Manager, and run MPI program on the given VE
- Used the wrapper as the server invocation command
  - Specified in FrovedisServer.initialize(...)



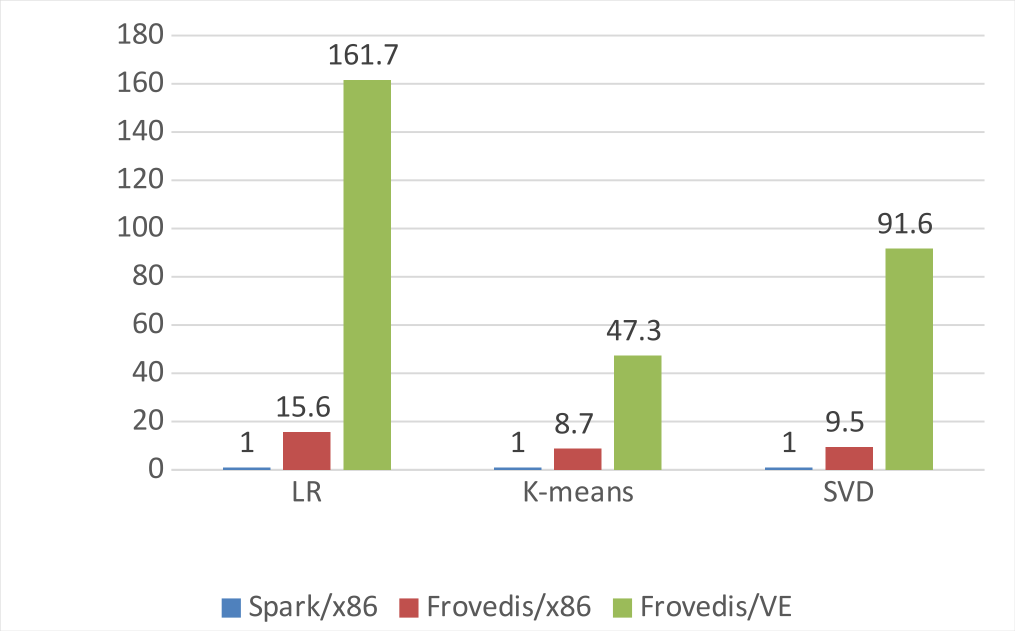
Performance Evaluation: Machine Learning

- Xeon (Gold 6126) 1 socket vs 1x VE10B, with sparse data (w/o I/O)
  - LR uses CTR data provided by Criteo (1/4 of the original, 6GB)
  - K-means and SVD used Wikipedia doc-term matrix (10GB)
  - Spark version: 2.2.1



Performance Evaluation: Machine Learning

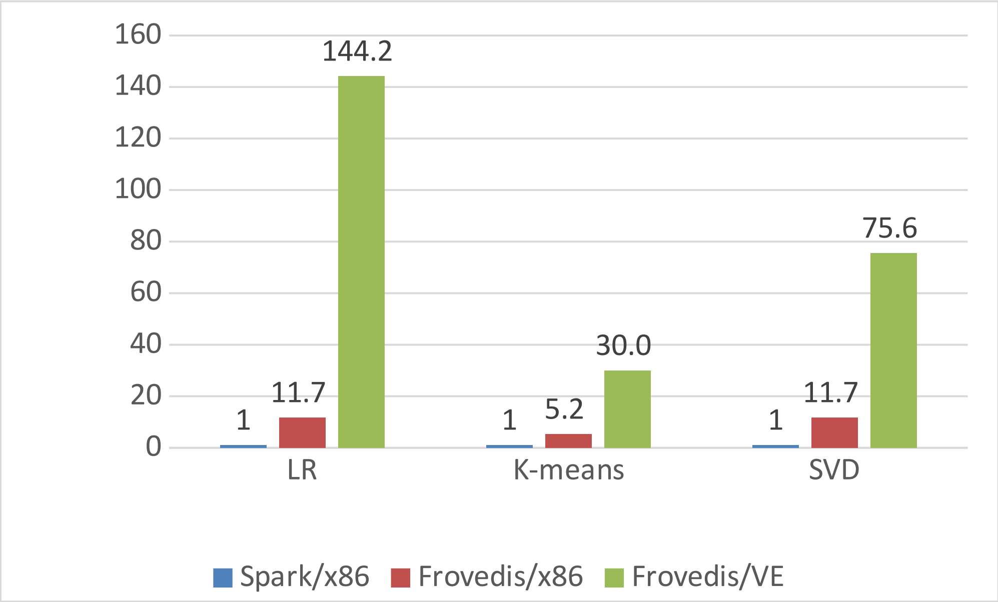
- Xeon (Gold 6226) 1 socket vs 1 VE10BE with sparse data (w/o I/O)
  - LR uses CTR data provided by Criteo (1/4 of the original, 6GB) Spark version 2.2.1
  - K-means and SVD used Wikipedia doc-term matrix (10GB) Spark version 3.0.0



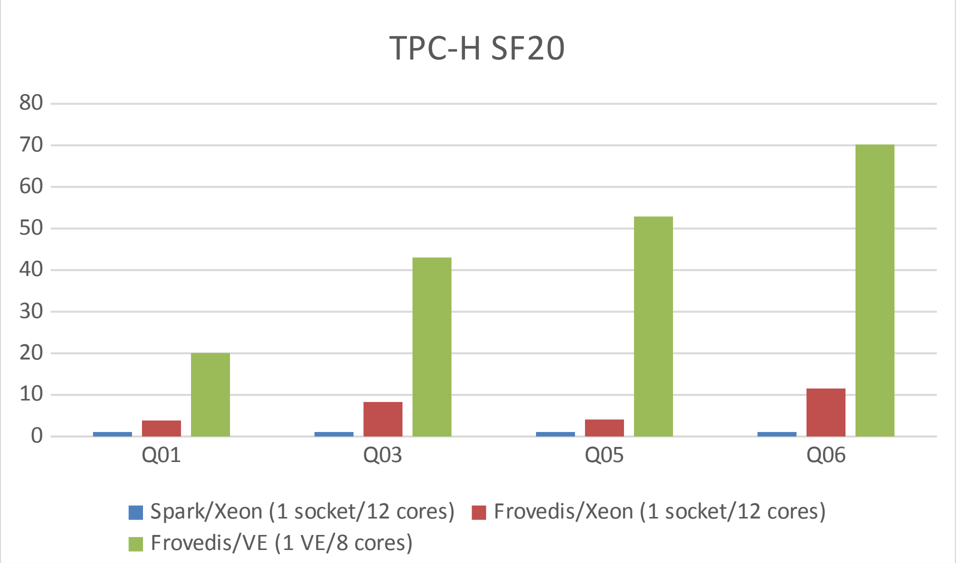
Performance Evaluation: Machine Learning

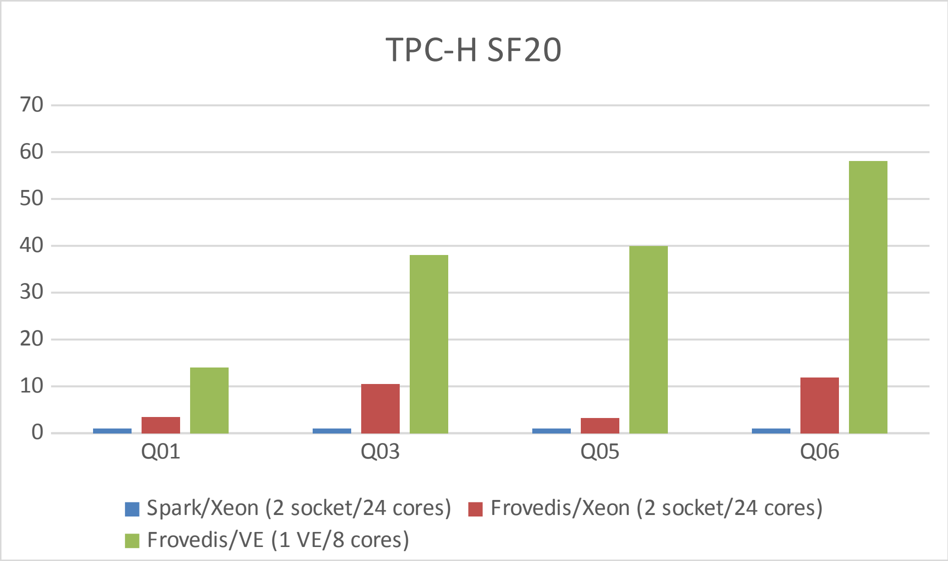
Xeon (Gold 6226) 2 socket
vs 1 VE10BE
with sparse data (w/o I/O)

- LR uses CTR data provided by Criteo (1/4 of the original, 6GB) Spark version 2.2.1
- K-means and SVD used Wikipedia doc-term matrix (10GB) Spark version 3.0.0



Performance Evaluation: DataFrame





XEON Gold 6226, Aurora A311-8 with VE10BE

Evaluated with TPC-H SF-20

- Q1: group by/aggregate
- Q3: filter, join, group by/aggregate
- Q5: filter, join, group by/aggregate (larger join)
- Q6: filter, group by/aggregate

Pointers and Resources

- https://github.com/frovedis/frovedis
  - Top README.md explains how to install
  - Check out Releases
- Start with Tutorials for Python/Spark
  https://github.com/frovedis/frovedis/blob/master/doc/tutorial_python/tutorial_python.pdf
  https://github.com/frovedis/frovedis/blob/master/doc/tutorial_spark/tutorial_spark.pdf
- Continue with Manuals of Python/Spark API
  https://github.com/frovedis/frovedis/blob/master/doc/manual/manual_python.pdf
  https://github.com/frovedis/frovedis/blob/master/doc/manual/manual_spark.pdf
- Performance benchmark and tips to improve performance
  https://github.com/frovedis/benchmark/blob/master/Tips.md

Conclusion

- NEC released new vector processor SX-Aurora TSUBASA that can accelerate data analytics and machine learning applications

- We have developed data analytics middleware Frovedis for SX-Aurora TSUBASA

- We show a 10x to 100x performance improvement on several machine learning and data frame processing

- NEC-X has opened VEDAC lab for accessing SX-Aurora TSUBASA AI platform with Frovedis.

\Orchestrating a brighter world

NEC