Hewlett Packard
Enterprise

ACCELERATING THE BIG DATA ANALYTICS SUITE

Pierre Carrier HPE, Scott Moe* AMD, Colin Wahl HPE, Alessandro Fanfarillo AMD

May 11, 2023

* now at Microsoft Azure

BIG DATA ANALYTICS SUITE: TABLE OF CONTENT

- Original CPU-only R with pbdMPI BDAS code;
- Concentration of work
- What is the AMD Tensile optimizer?
- Accelerated BDAS: C++/HIP with MPI:
  - ✓ Accelerated K-Means (82% GEMM; 320X)
  - ✓ Accelerated PCA (99% GEMM; 360X)
  - ○ Accelerated SVM (20% GEMM; 120X)
- Conclusions (half-precision, python, modern algorithms)

Same structure as our CUG paper

PRINCIPAL INVESTIGATING VERSUS BENCHMARKING

PI researchers/engineers

Original BDAS
R with pbdMPI

CORAL-2

National Labs'
IT team

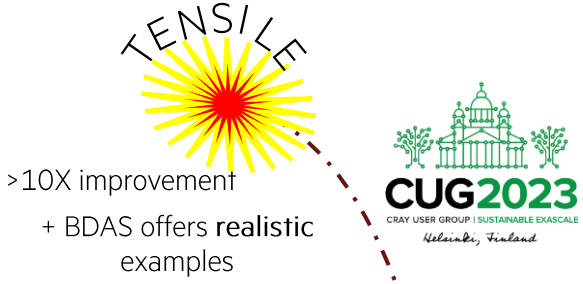
HPE's
IT team

Benchmarking engineering

Improvement factor for Frontier

Optimized BDAS
C++/HIP and MPI

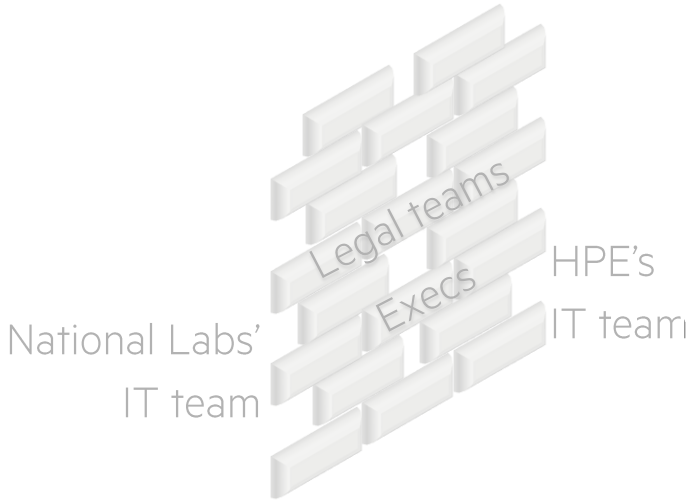
PRINCIPAL INVESTIGATING VERSUS BENCHMARKING



Center of Excellence

PI researchers/engineers

Benchmarking engineering



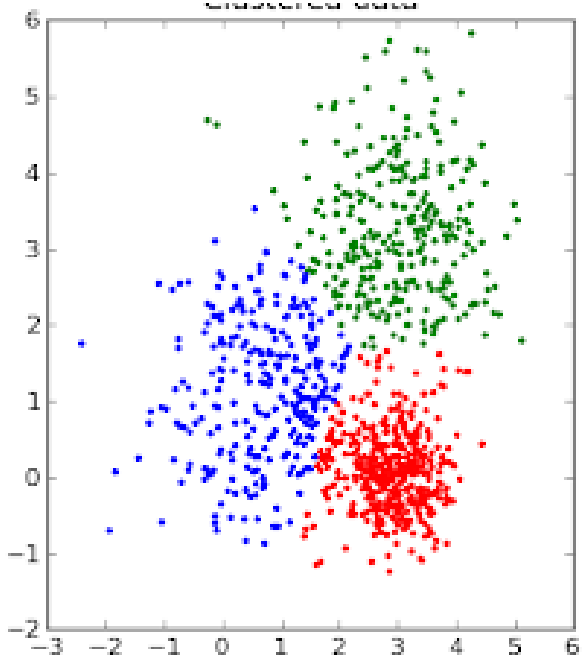
Original BDAS
R with pbdMPI

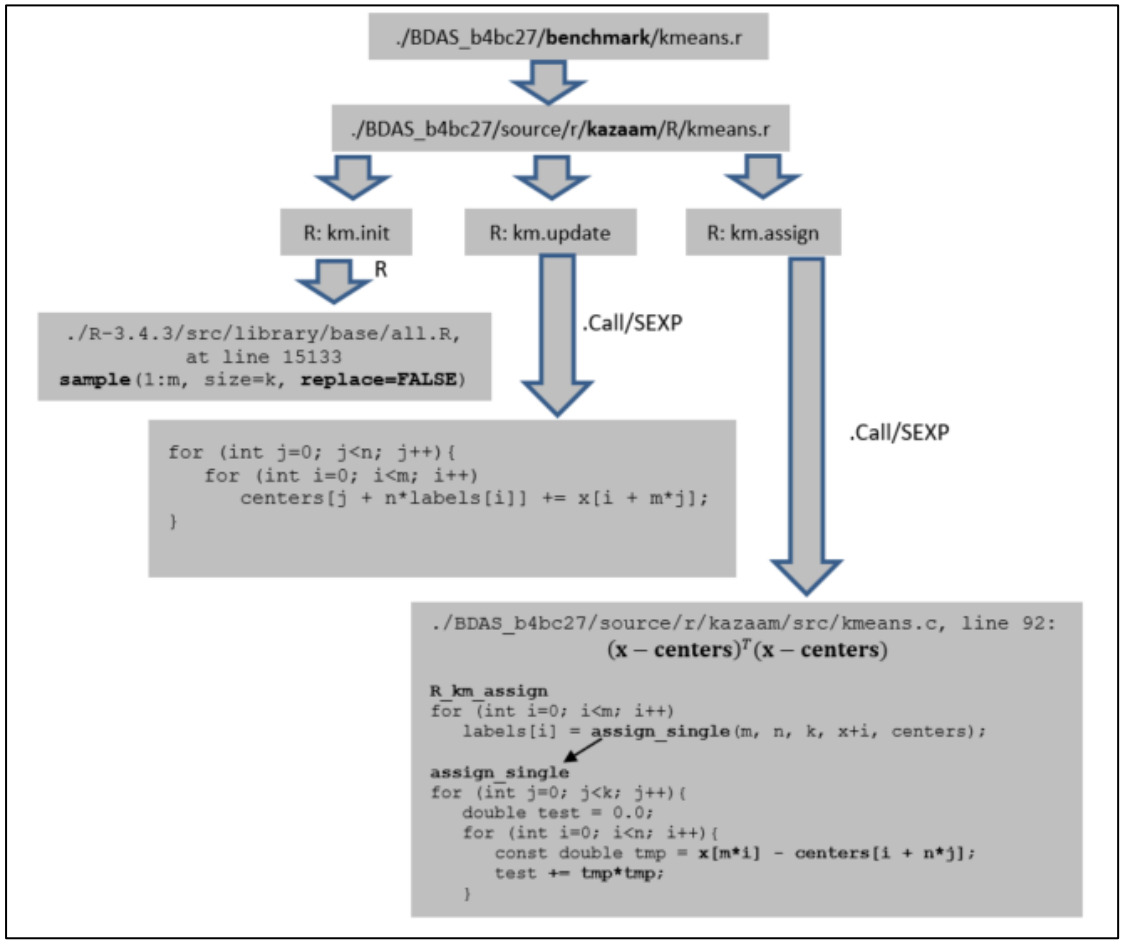
Optimized BDAS
C++/HIP and MPI

WHAT IS BDAS? K-MEANS, PCA, SVM.

K-Means

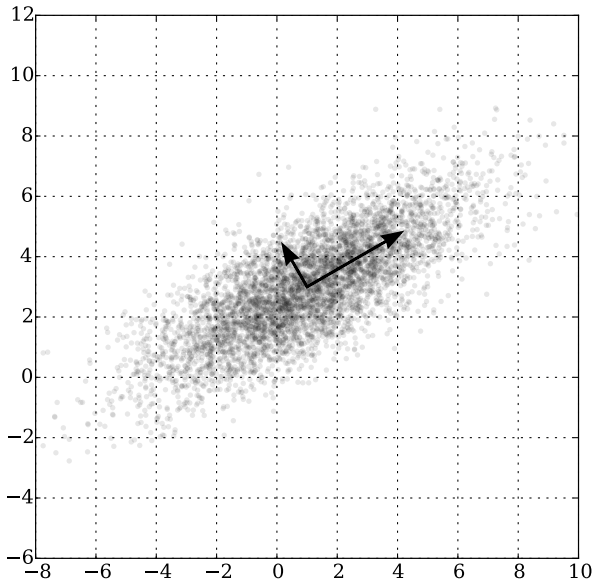
- Partition n data points into k clusters
- Example: Image Segmentation

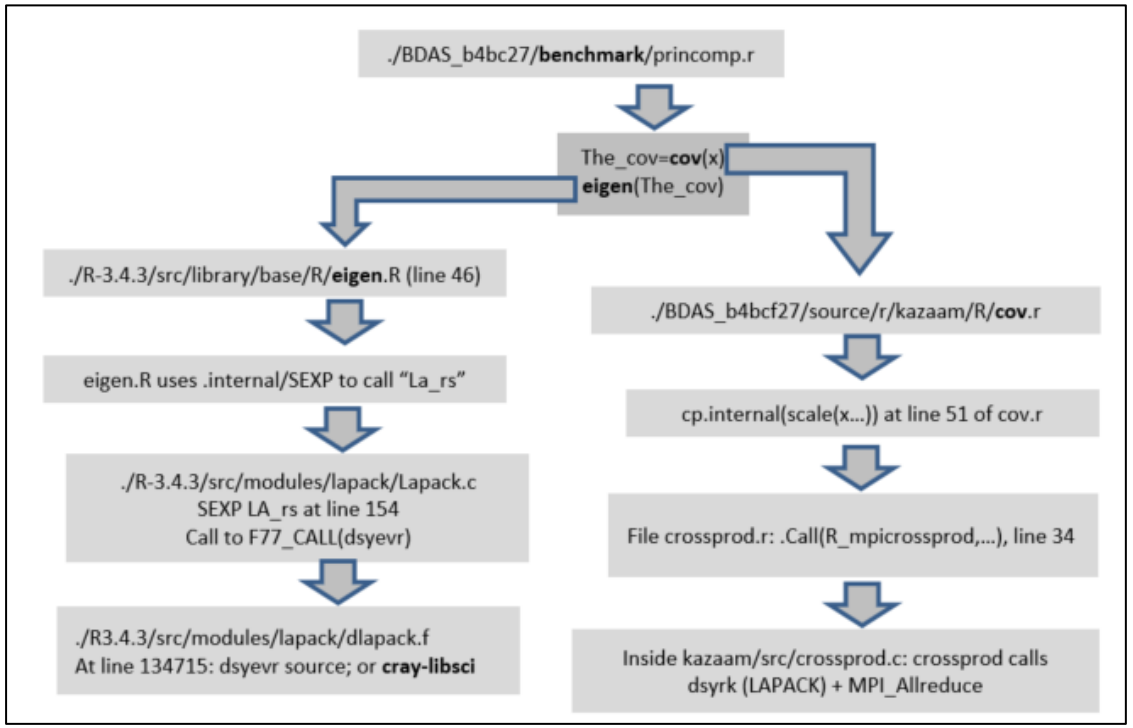




WHAT IS BDAS? K-MEANS, PCA, SVM.

Principal Component Analysis

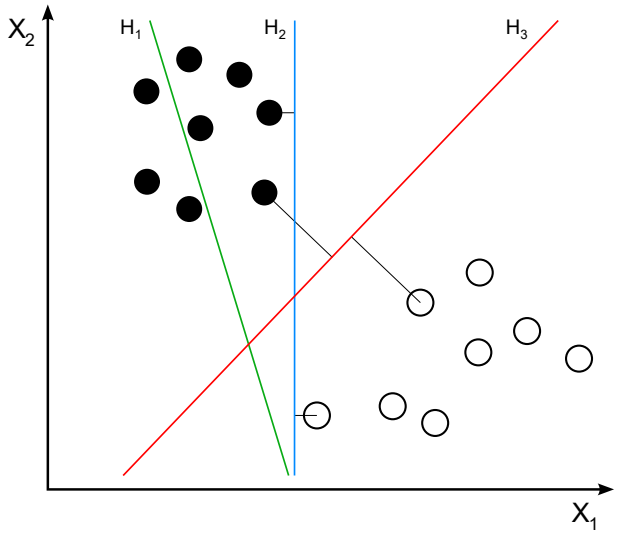




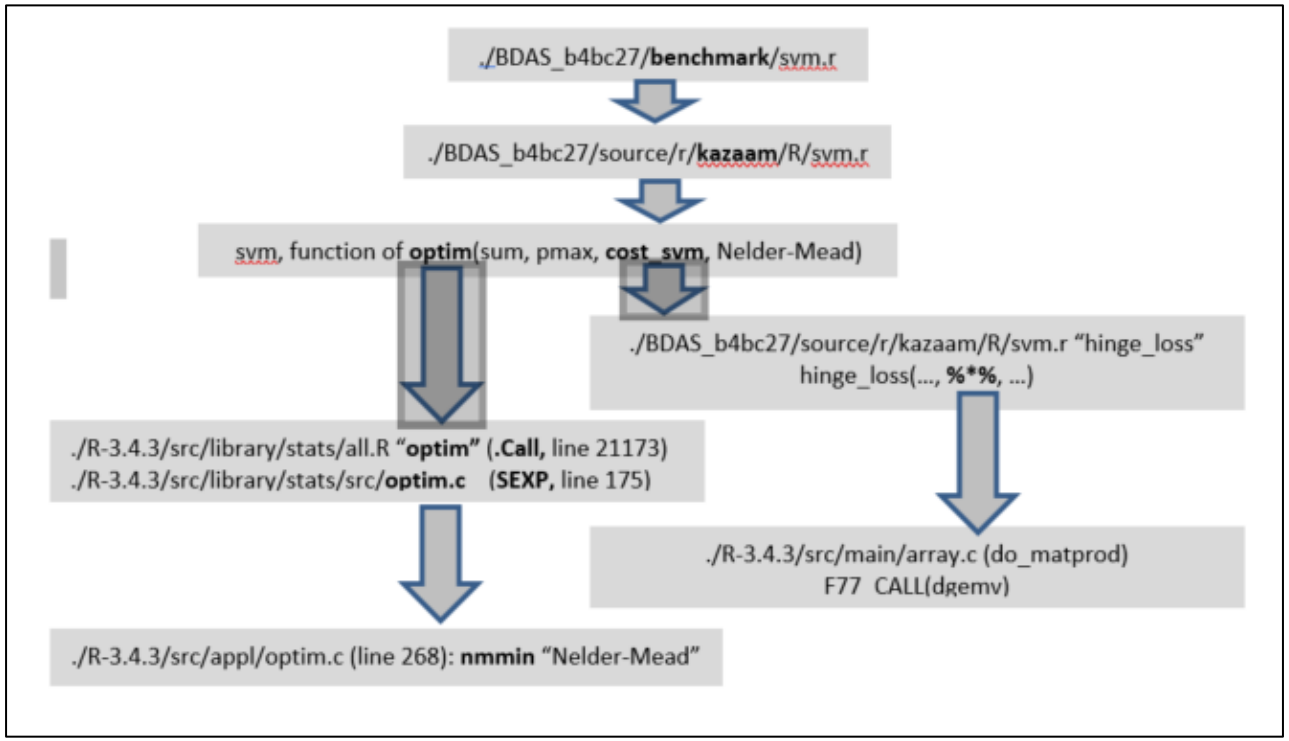
- Reduce dimensionality of data by finding coordinate system with basis sorted by variance
- Example: Data Compression

WHAT IS BDAS? K-MEANS, PCA, SVM.

Support Vector Machine



- Partition data with n dimensional features into two groups separated by an n-1 dimensional hyperplane.
- Example: Handwriting Recognition



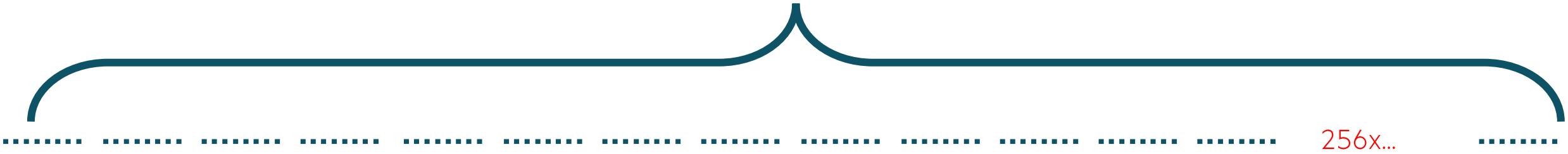
GENERAL OPTIMIZATION SCHEME: CONCENTRATION OF WORK

Reference: 1TB of distributed data

1X the local matrix size

Keeping 1 MPI rank per MI250X GCD

One ensemble: Distribution of local matrices across the network in the R with pbdMPI CPU code



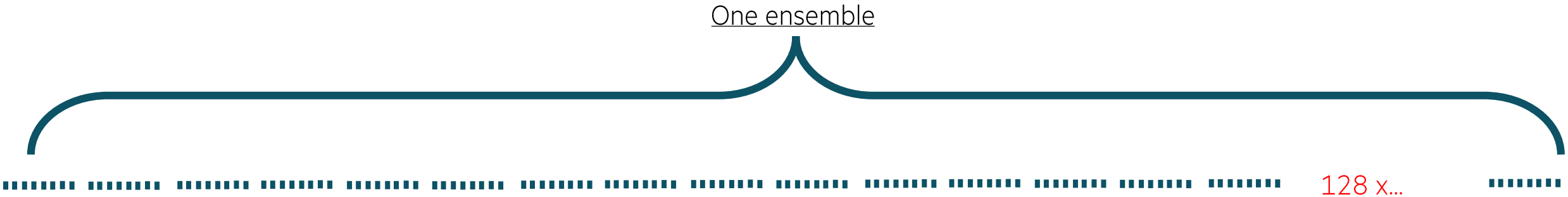
(256 Nodes) 2048 MPI ranks: 250,000 objects X 250 features

GENERAL OPTIMIZATION SCHEME: CONCENTRATION OF WORK

Reference: 1TB of distributed data

2X the local matrix size

Keeping 1 MPI rank per MI250X GCD



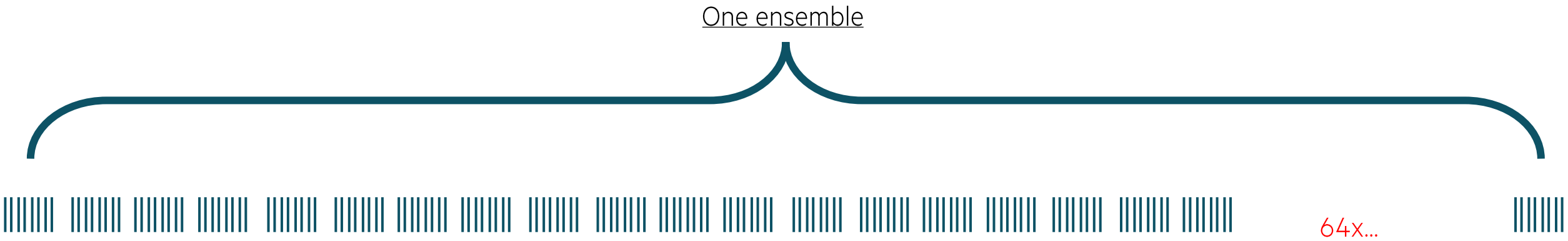
(128 Nodes) 1024 MPI ranks: 500,000 objects X 250 features

GENERAL OPTIMIZATION SCHEME: CONCENTRATION OF WORK

Reference: 1TB of distributed data

4X the local matrix size

Keeping 1 MPI rank per MI250X GCD



(64 Nodes) 512 MPI ranks: 1,000,000 objects X 250 features

GENERAL OPTIMIZATION SCHEME: CONCENTRATION OF WORK

Reference: 1TB of distributed data

8X the local matrix size

Keeping 1 MPI rank per MI250X GCD

One ensemble

(32 Nodes) 256 MPI ranks: 2,000,000 objects X 250 features

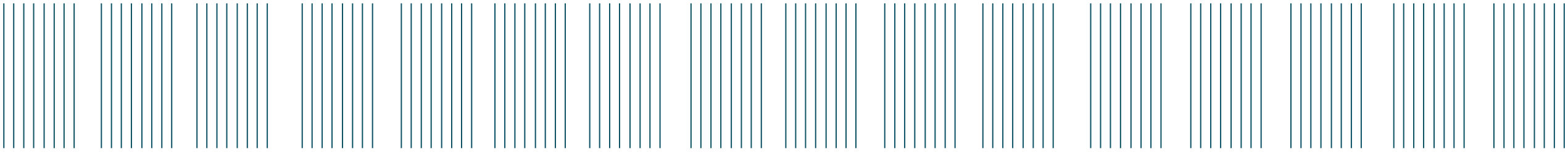
GENERAL OPTIMIZATION SCHEME: CONCENTRATION OF WORK

Reference: 1TB of distributed data

16X the local matrix size

One ensemble

Keeping 1 MPI rank per MI250X GCD

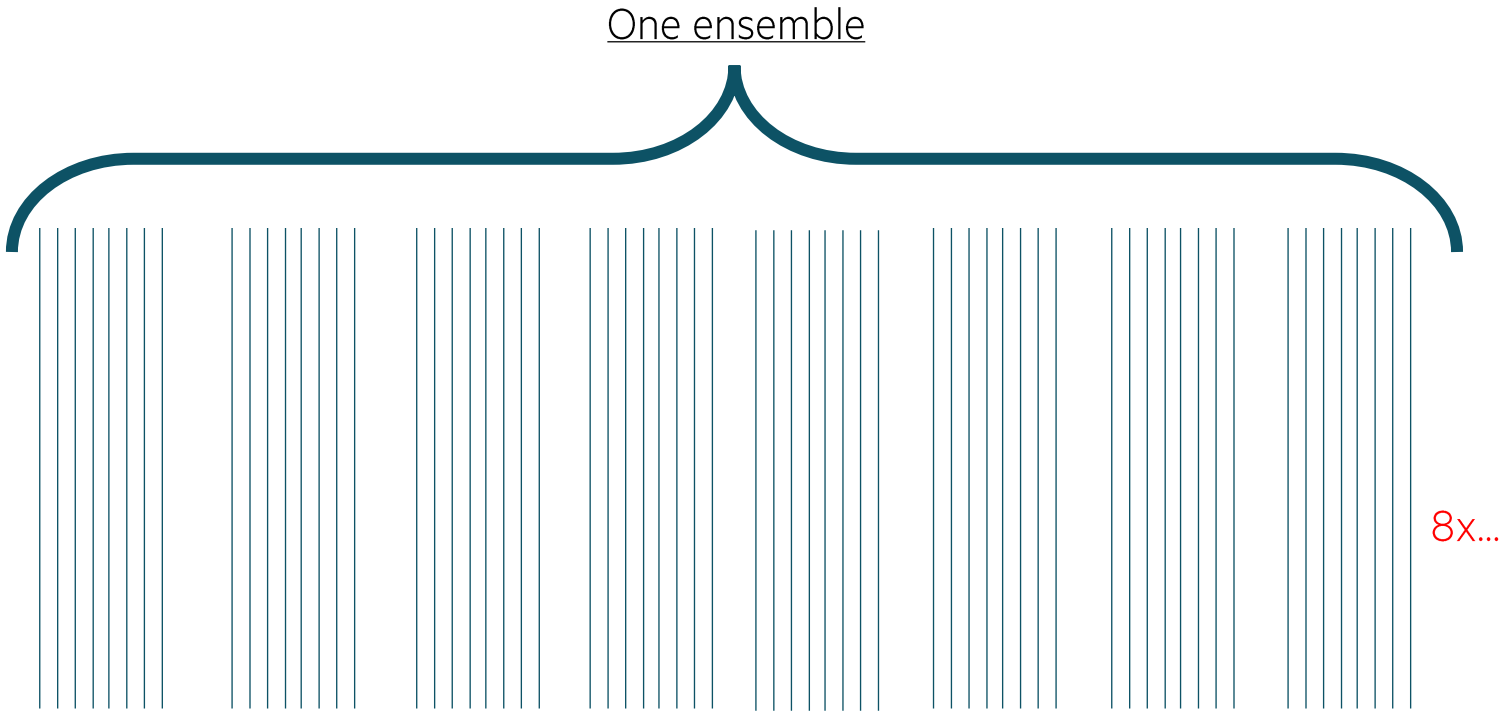


16x...

(16 Nodes) 128 MPI ranks: 4,000,000 objects X 250 features

GENERAL OPTIMIZATION SCHEME: CONCENTRATION OF WORK

Reference: 1TB of distributed data

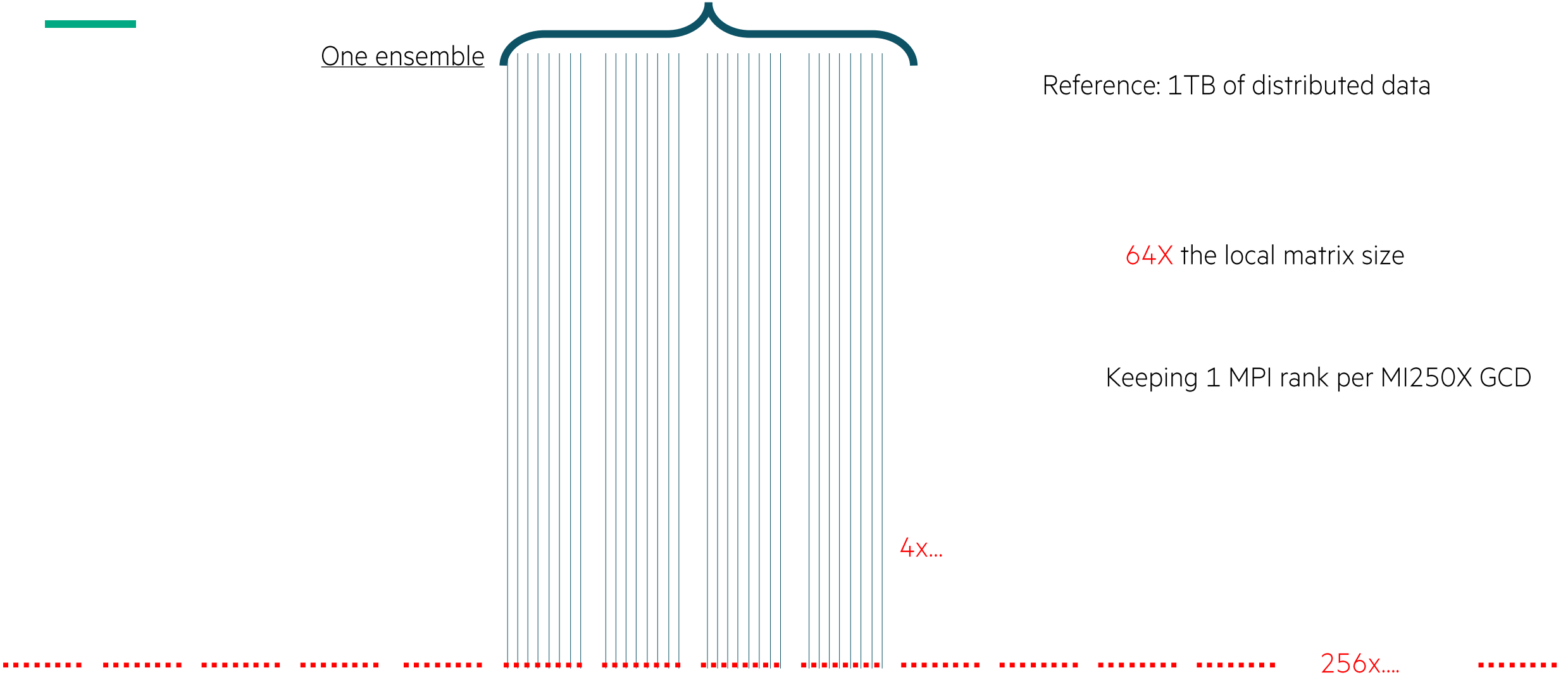


32X the local matrix size

Keeping 1 MPI rank per MI250X GCD

(8 Nodes) 64 MPI ranks: 8,000,000 objects X 250 features

GENERAL OPTIMIZATION SCHEME: CONCENTRATION OF WORK



Reference: 1TB of distributed data

64X the local matrix size

Keeping 1 MPI rank per MI250X GCD

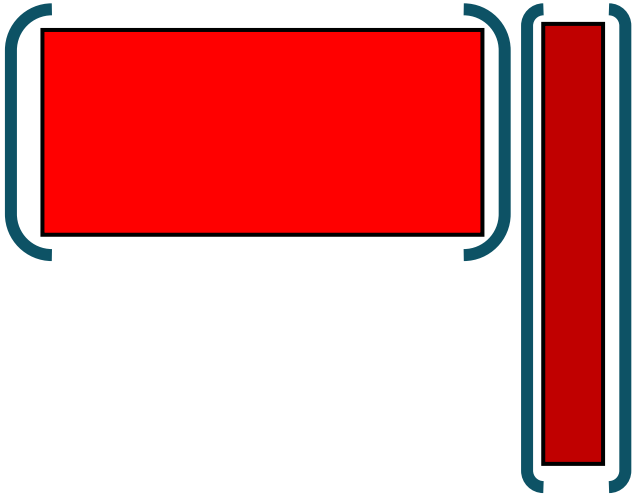
(4 GPU Nodes) 32 MPI ranks: **16,000,000 objects X 250 features**

(256 nodes) 2048 MPI ranks: 250,000 objects X 250 features

WHAT IS AMD TENSILE?

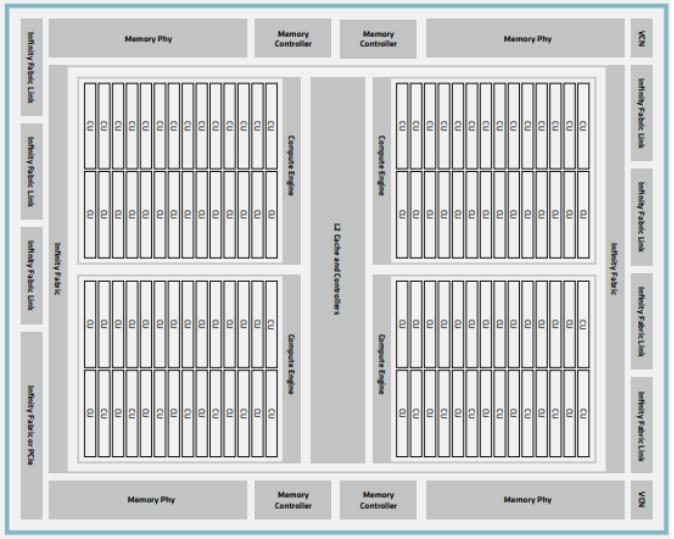
An “Origami” problem: How to “fold” a matrix product onto a GCD, to maximize performance?

Math:



Hardware:

GCD

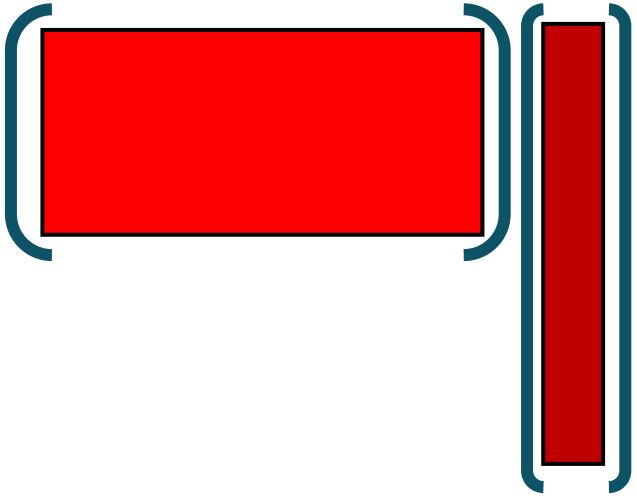


https://www.amd.com/system/files/documents/amd-cdna2-white-paper.pdf

WHAT IS AMD TENSILE?

An “Origami” problem: How to “fold” a matrix product onto a GCD, to maximize performance?

Math:

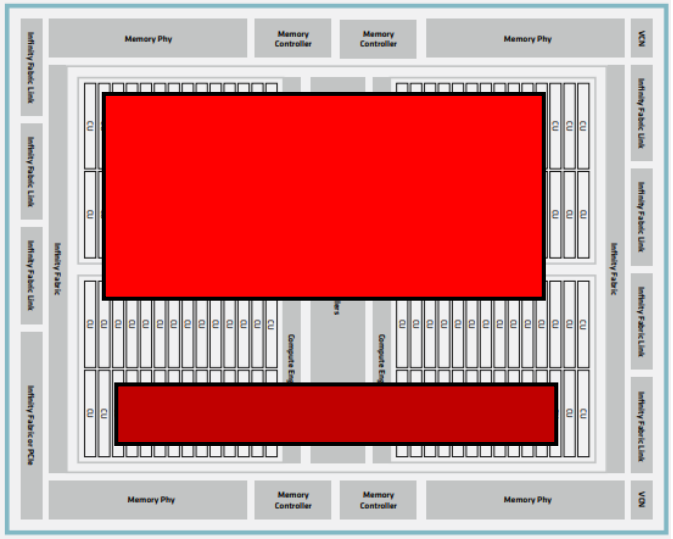


hibblasDgemm(arguments…) NO OPTIMIZATION

Won’t directly “fit”; inefficient

Hardware:

GCD



https://www.amd.com/system/files/documents/amd-cdna2-white-paper.pdf

WHAT IS AMD TENSILE?

An “Origami” problem: How to “fold” a matrix product onto a GCD, to maximize performance?

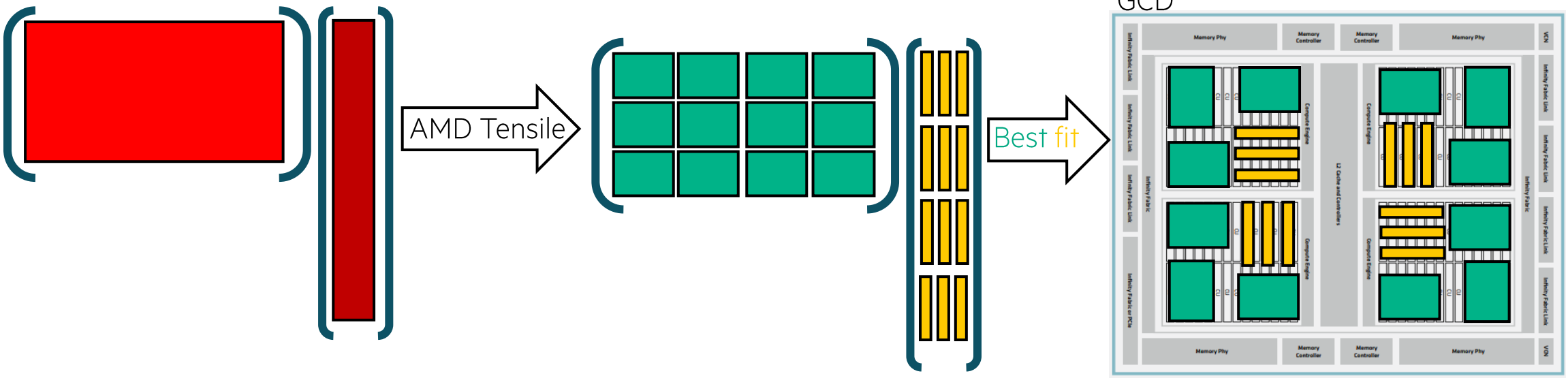
Math:

Software:

librocm: hipblasDgemm(arguments…)

Hardware:

GCD



https://www.amd.com/system/files/documents/amd-cdna2-white-paper.pdf

Tensile finds the optimum tiling, among “all” possible solutions from the parameter space of the GCD.

WHAT IS AMD TENSILE?

https://github.com/ROCmSoftwarePlatform/Tensile

The goal of the Tensile tuning process is to build libraries containing gpu kernels which provide optimized performance for any problem specification within a given problem domain.

1 Tensile and rocBLAS

rocBLAS is a BLAS implementation which is included as part of AMD's ROCm project offerings which optimizes the BLAS routines for AMD's GPUs. The gpu kernels which implement the rocBLAS blas3 gemm routines are optimized using Tensile, which is a utility that benchmarks the performance of selected kernels that are generated from "tuning parameters" and selected for inclusion in the final library based on the results. This library is included as part of the rocBLAS package and loaded at run-time during initialization. For a given gemm call, the problem parameters are passed to the Tensile client which selects the kernel with the best known performance for the problem.

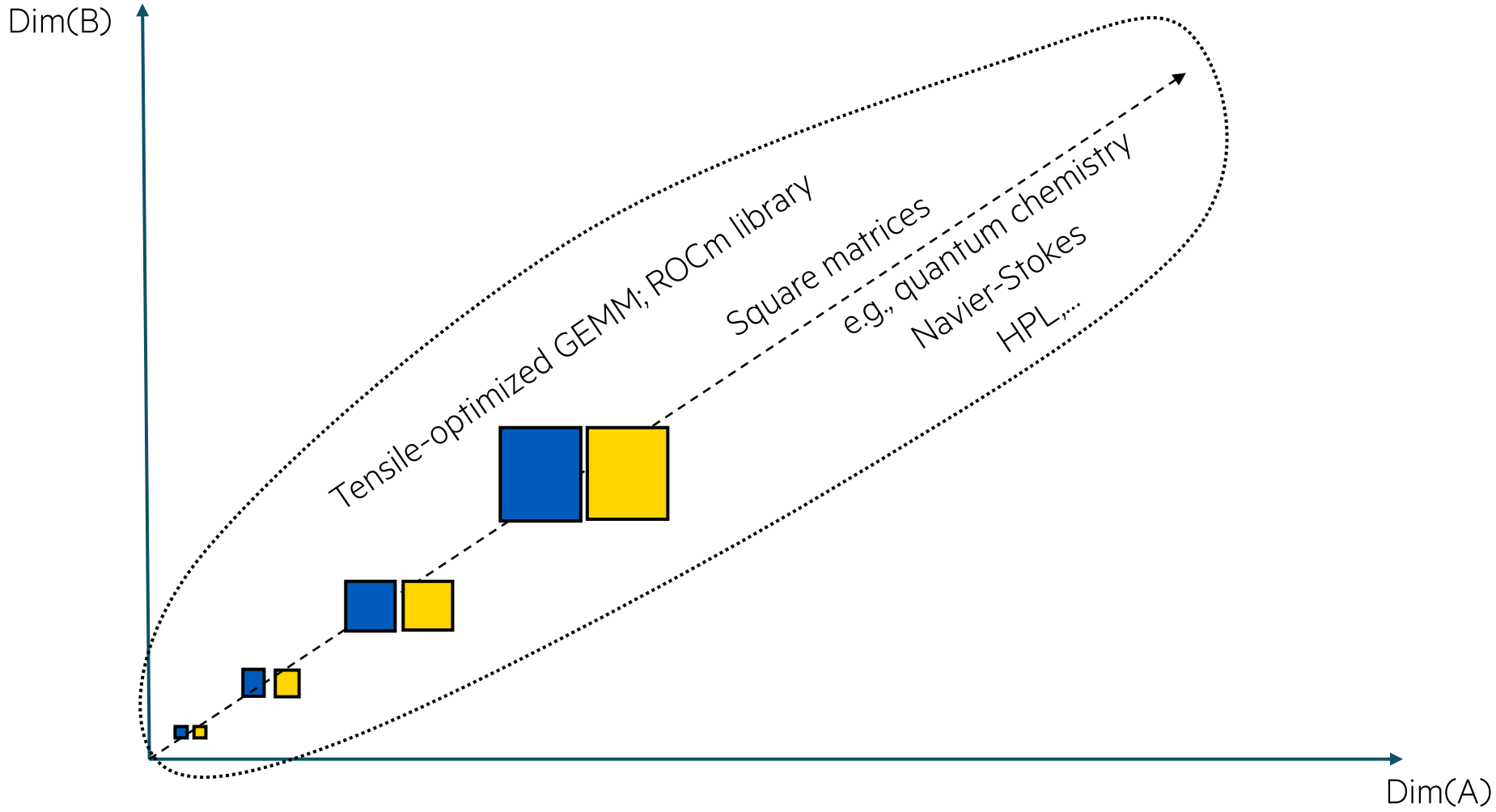
$$\textbf{Cijk\_Ailk\_Bljk} \implies C = A * B$$
$$\textbf{Cijk\_Ailk\_Bjlk} \implies C = A * B^T$$
$$\textbf{Cijk\_Alik\_Bljk} \implies C = A^T * B$$
$$\textbf{Cijk\_Alik\_Bjlk} \implies C = A^T * B^T$$

Considering the full set of parameters involved, it is not practical to build every set of kernels as part of any investigation; this would exhaust compute resources. The target number of kernels we consider optimal for investigation is between 10,000 and 20,000. The tuning workflow may take many iterations of the tuning to reach optimal performance. Each iteration will perform a complete run of the Tensile tuning and evaluation of the results. If some of the problem sizes under investigation do not result in optimal performance then, for the sizes that fail, modifications of the tuning specifications are made to search alternative solutions in the kernel space. The tuning is then rerun using the updated parameter set. This type of iteration is continued until the desired optimization is reached.

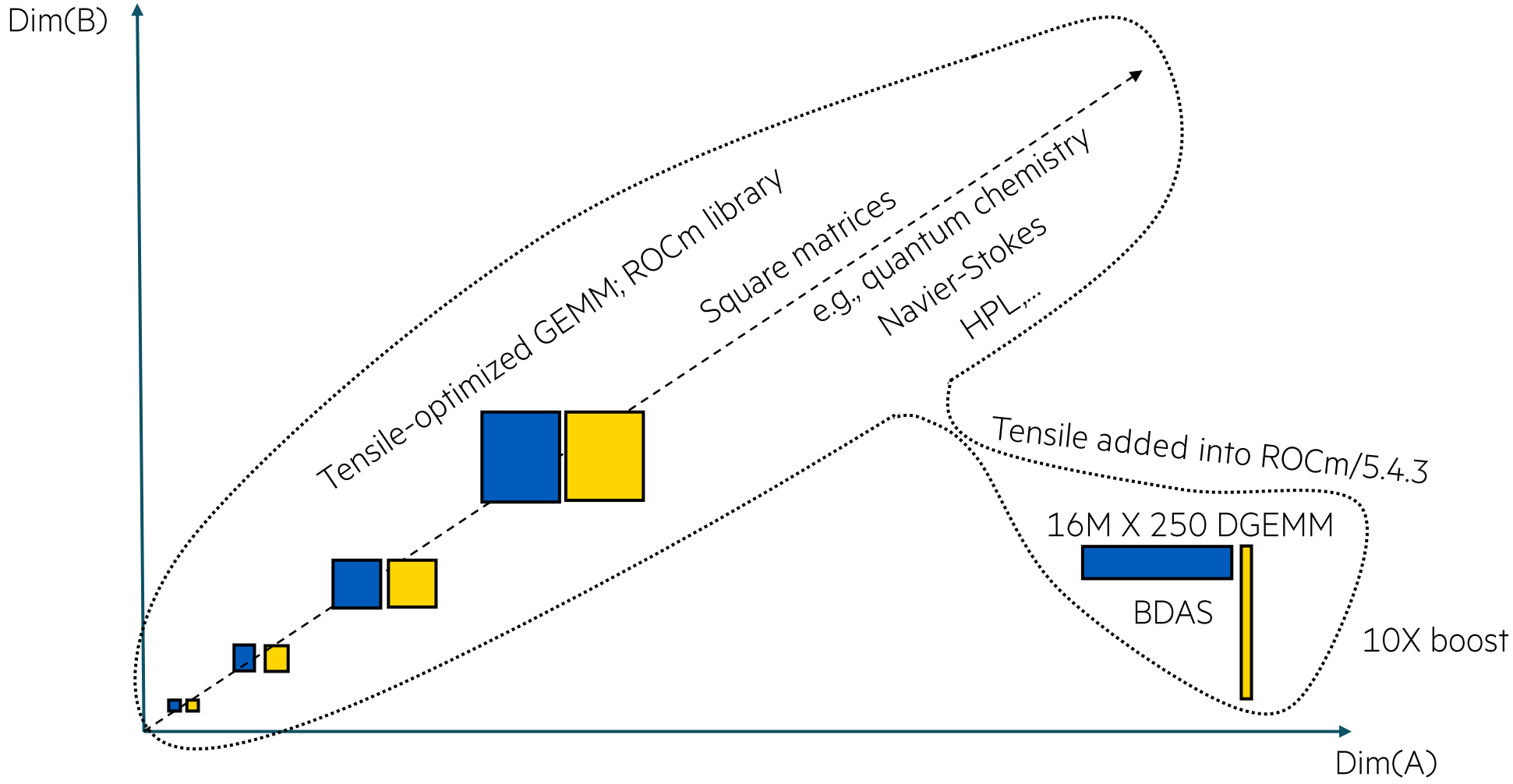
More info: https://github.com/ROCmSoftwarePlatform/Tensile/blob/develop/tuning_docs/tensile_tuning.tex

Email Alessandro or myself if you need to utilize Tensile… (See CUG paper)

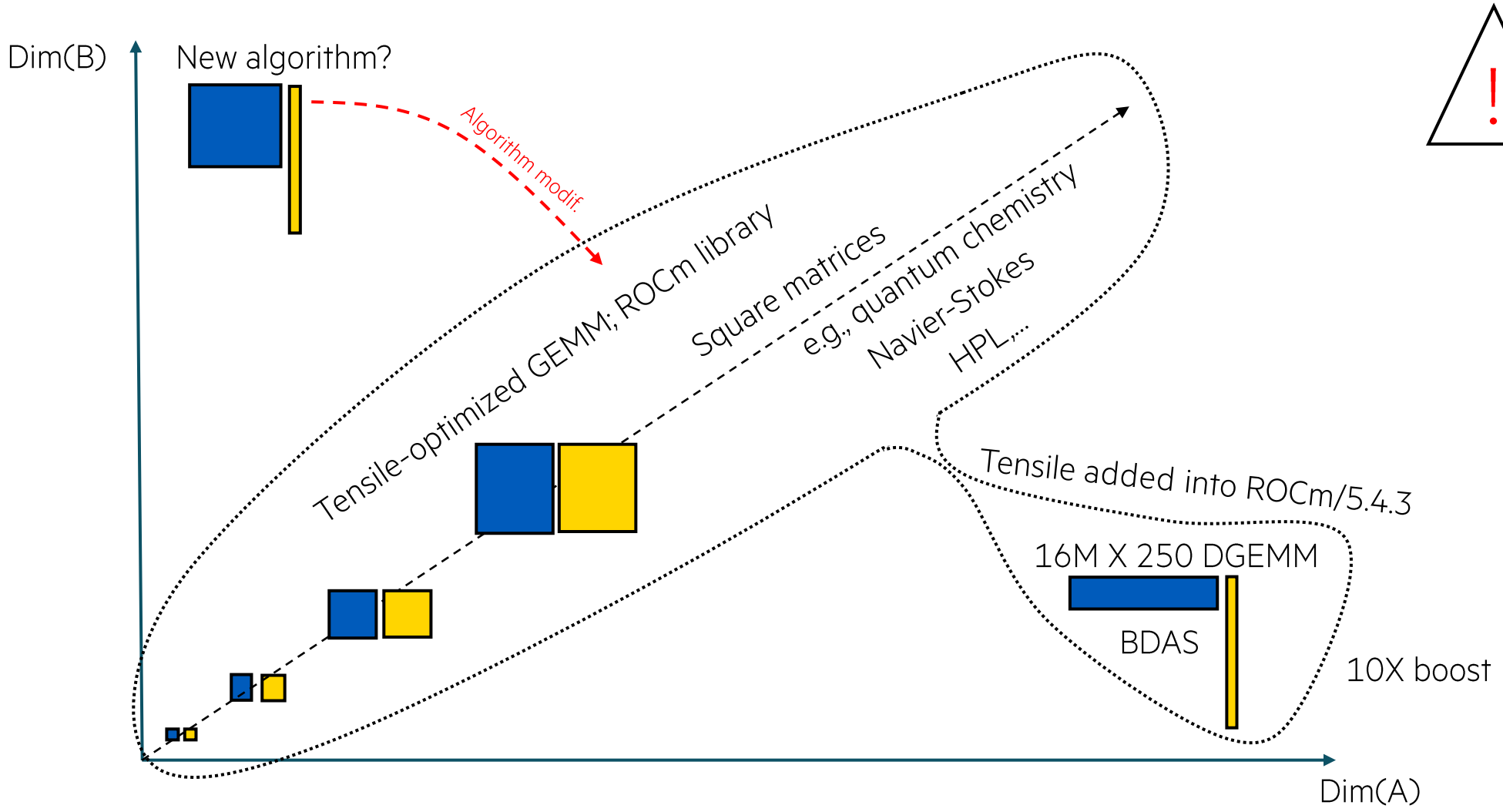
WHAT IS AMD TENSILE?



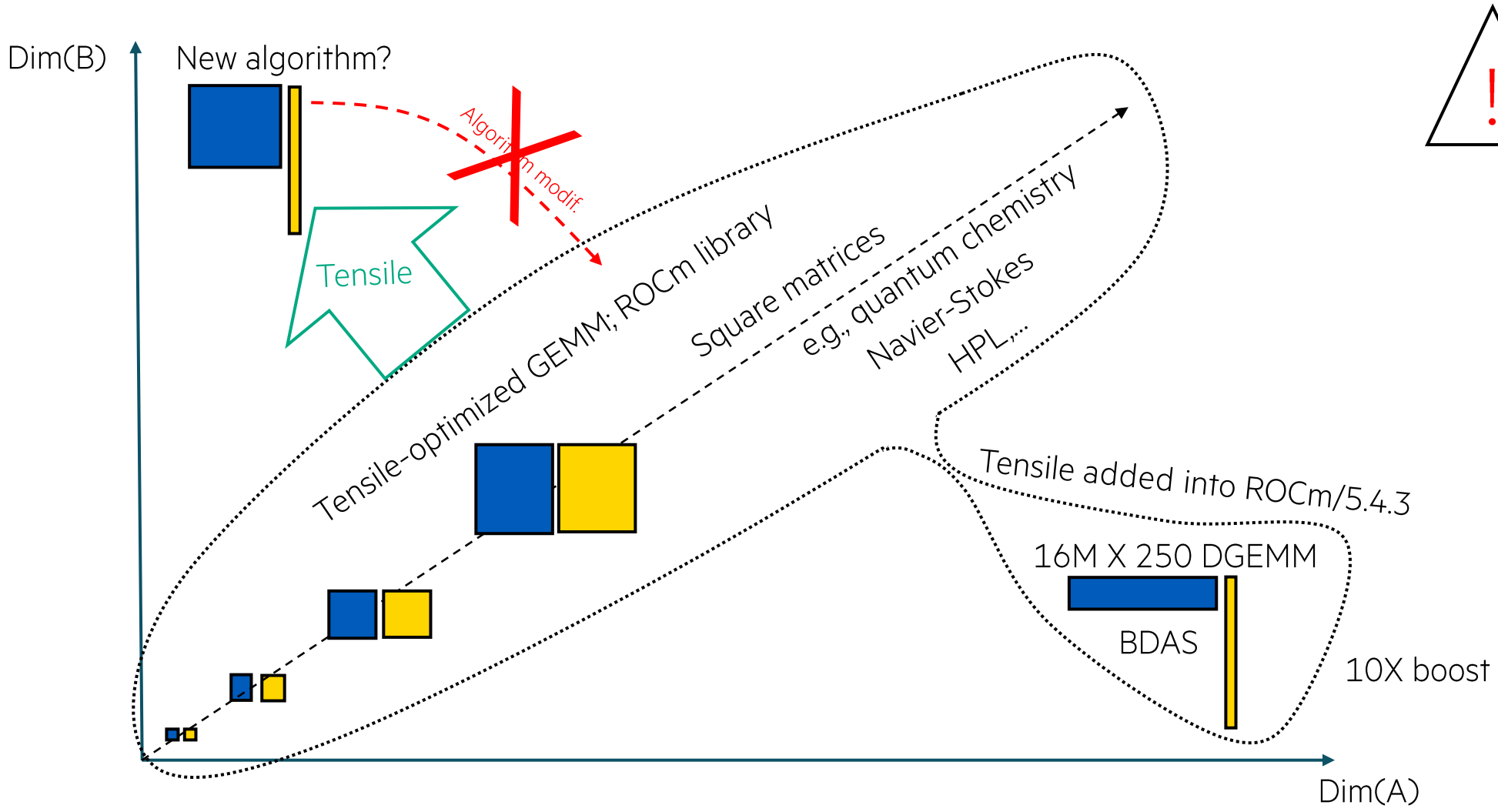
WHAT IS AMD TENSILE?



WHAT IS AMD TENSILE?



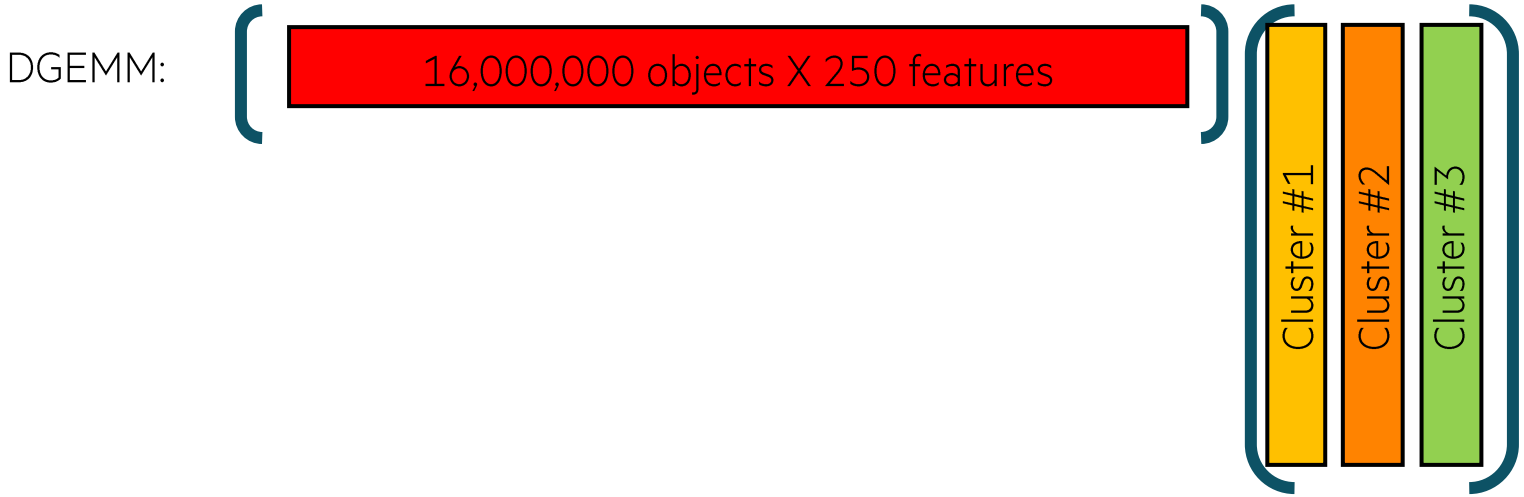
WHAT IS AMD TENSILE?



Email Alessandro or myself if you want to **utilize Tensile**... (See CUG paper)

ACCELERATED K-MEANS (82% GEMMS)

NP-harness (consequence): 2 different initial conditions may converge to different solutions.



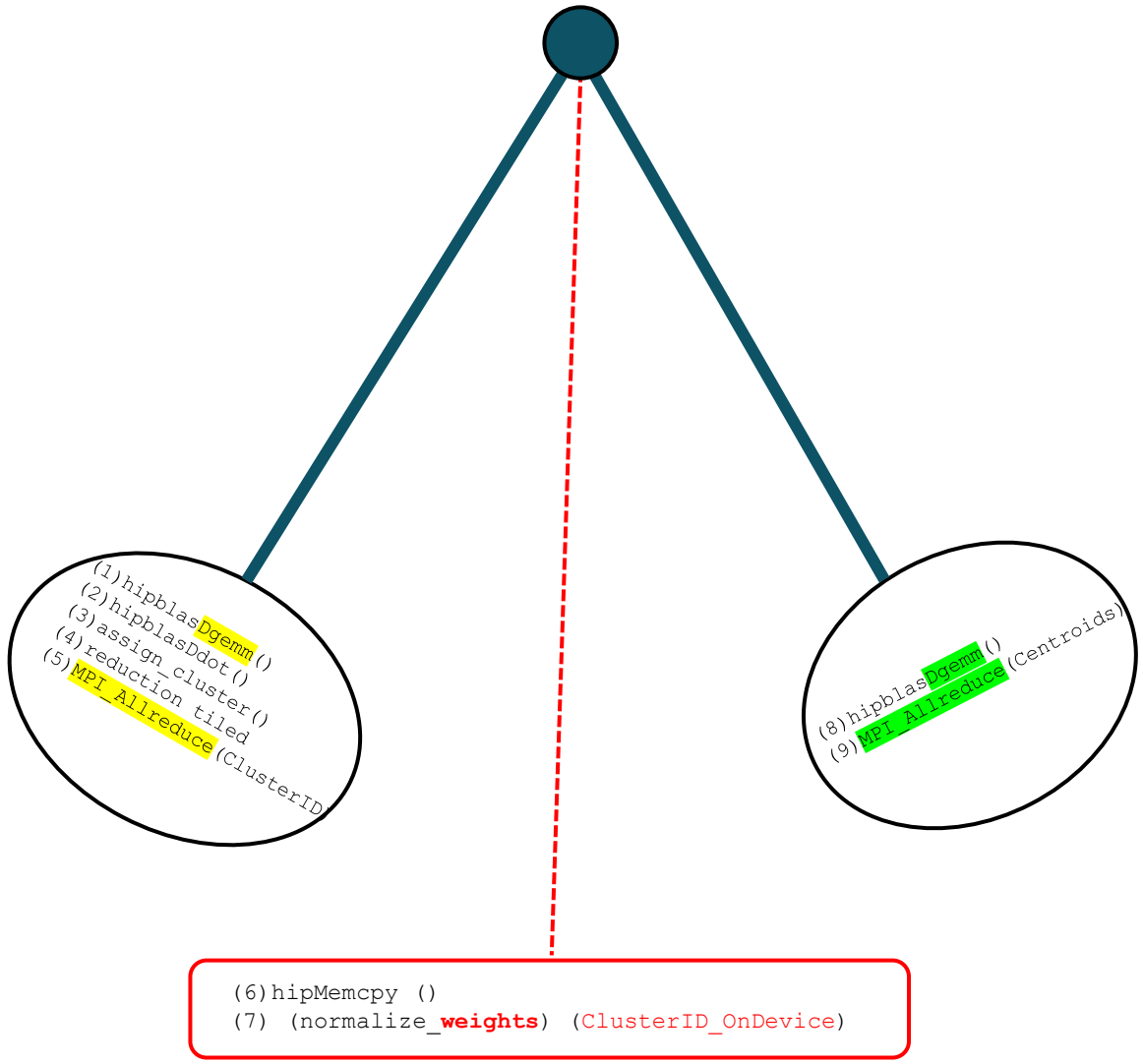
We are stacking 3 problems at the same time

ACCELERATED K-MEANS (82% GEMMS)

Algorithm:

pendulum swing between 2 pairs of GEMM+MPI_Reduce,

until the weights and ID don't change



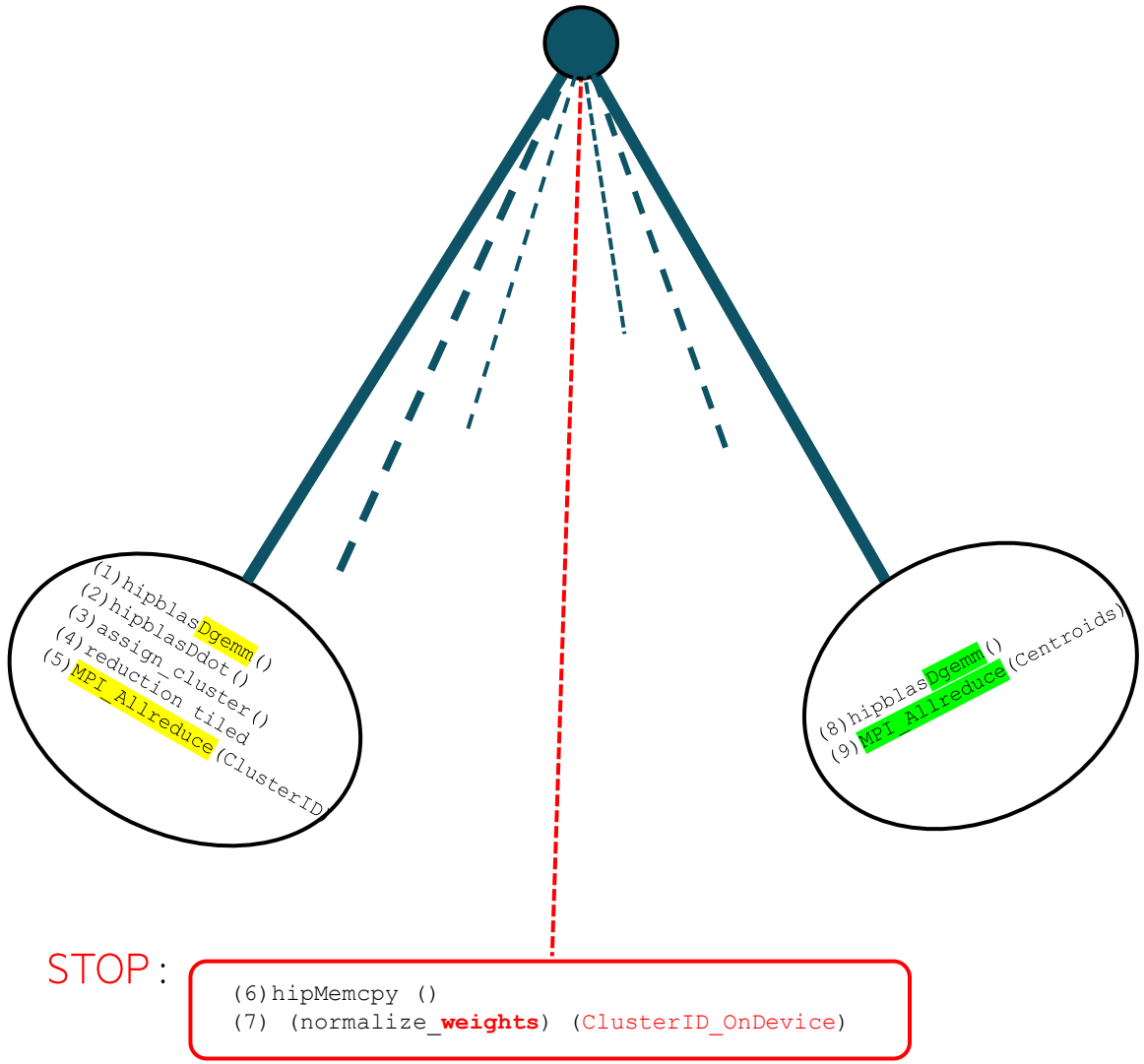
ACCELERATED K-MEANS (82% GEMMS)

Algorithm:

pendulum swing between 2 pairs of GEMM+MPI_Reduce,

until the weights and ID don't change

```
for (int outer=0; outer < FIXEDPOINT; outer++) {
```



STOP :

```
(6)hipMemcpy ()
(7) (normalize_weights) (ClusterID_OnDevice)
```

```
}
```

ACCELERATED K-MEANS (82% GEMMS)

One TB of data is **40X faster** than the CPU R code.
320X faster (including 8X concentration factor)

Tensile easily provides an order of magnitude of that improvement.

```
for (int outer=0; outer < FIXEDPOINT; outer++) {
```

Rocprof…

Cijk_Alik_Bljk (32x16x32): 50%

(1)hipblasDgemm()
(2)hipblasDdot()
(3)assign_cluster()
(4)reduction tiled
(5)MPI_Allreduce(ClusterID)

Cijk_Ailk_Bljk (1024x32x8): 30%

(8)hipblasDgemm()
(9)MPI_Allreduce(Centroids)

STOP:

```
(6)hipMemcpy ()
(7) (normalize_weights) (ClusterID_OnDevice)
```

```
}
```

ACCELERATED PCA (99% GEMM)

```
...
if (rank == 0) {
   eig.init_syevd(Variance_OnHost, NFeatures);
}

hipblasDgemm(handle, HIPBLAS_OP_T, HIPBLAS_OP_N, NFeatures, NFeatures, MObjects_rank, &alpha, DataMatrix_OnDevice, MObjects_rank, DataMatrix_OnDevice, MObjects_rank,
                                                                                          &beta, Variance_OnDevice, NFeatures);

MPI_Reduce(&Variance_OnDevice[0], &Variance_OnHost[0], NFeatures*NFeatures, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);

if (rank == 0) {
   eig.compute_eigenvalues(Variance_OnHost);
}
```

The MPI_Reduce takes the variance on the device and sends it on host of rank 0, only through the MPI arguments

ACCELERATED PCA (99% GEMM)

```
...
if (rank == 0) {
   eig.init_syevd(Variance_OnHost, NFeatures);
}

hipblasDgemm(handle, HIPBLAS_OP_T, HIPBLAS_OP_N, NFeatures, NFeatures, MObjects_rank, &alpha, DataMatrix_OnDevice, MObjects_rank, DataMatrix_OnDevice, MObjects_rank,
                                                                                          &beta, Variance_OnDevice, NFeatures);

MPI_Reduce(&Variance_OnDevice[0], &Variance_OnHost[0], NFeatures*NFeatures, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);

if (rank == 0) {
   eig.compute_eigenvalues(Variance_OnHost);
}
```

The MPI_Reduce takes the variance on the device and sends it on host of rank 0, only through the MPI arguments

Profiles show that the code uses **99% DGEMM**: Cijk_Alik_Bljk (128x256x16)
One ensemble 1TB of data is 45X faster than the CPU R code.
Or 360X faster, including the 8X concentration factor.

Tensile easily provides an order of magnitude of that improvement.

ACCELERATED PCA (99% GEMM)

```
...
if (rank == 0) {
   eig.init_syevd(Variance_OnHost, NFeatures);
}

hipblasDgemm(handle, HIPBLAS_OP_T, HIPBLAS_OP_N, NFeatures, NFeatures, MObjects_rank, &alpha, DataMatrix_OnDevice, MObjects_rank, DataMatrix_OnDevice, MObjects_rank,
                                                                                            &beta, Variance_OnDevice, NFeatures);

MPI_Reduce(&Variance_OnDevice[0], &Variance_OnHost[0], NFeatures*NFeatures, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);

if (rank == 0) {
   eig.compute_eigenvalues(Variance_OnHost);
}
```

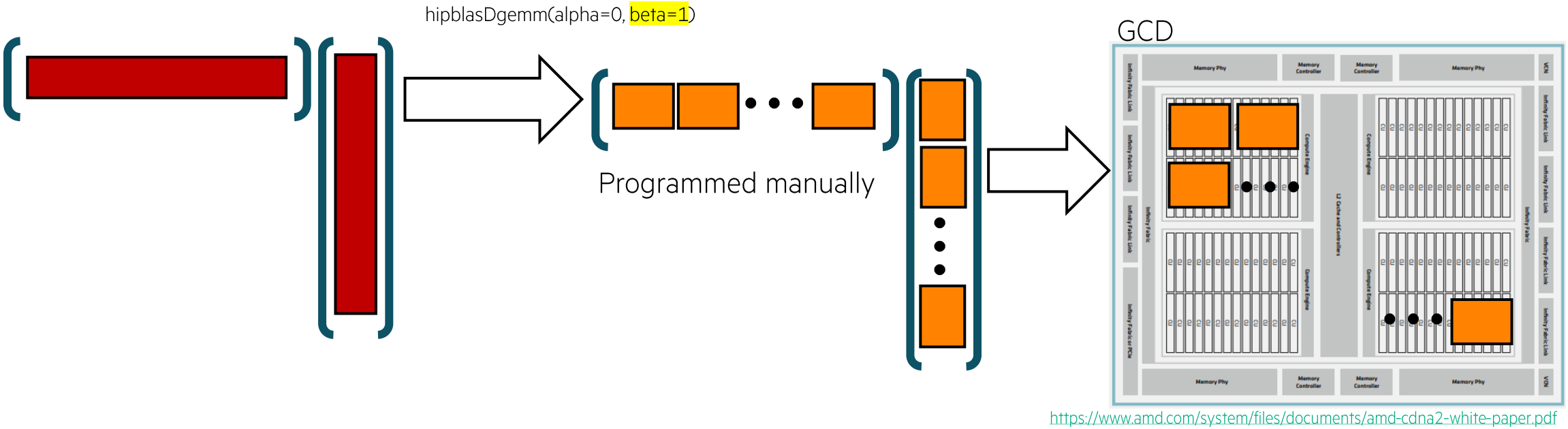
The MPI_Reduce takes the variance on the device and sends it on host of rank 0, only through the MPI arguments

We've also considered two other implementations (that we eventually discarded):

- scaLAPACK PDSYEVD+PDGEMM
  - unnecessary communication,
  - no PDGEMM in HIP/ROCm
- Created a series of local "squarer" matrices managed with dot products of matrices **(next slide…)**
  - Let Tensile find the best optimization

ACCELERATED PCA (99% GEMM)

Created a series of local “squarer” matrices managed with dot products of matrices:



https://www.amd.com/system/files/documents/amd-cdna2-white-paper.pdf

This is ONE of the Tensile parameter space test (among more than 10,000)

It is better to let Tensile find the best optimization

ACCELERATED SVM (MATRIX VECTOR VERSION)

Nelder-Mead:

```
Initialize: hinge_loss (1)
Loop hinge_loss (2) DGEMM-like
if (Highest > Lowest && Lowest > tolerance)
     hinge_loss (3)
     if (Reflection < Lowest)
          hinge_loss (4)
     else (Reflection >= Lowest)
          hinge_loss (5)
endif
```

Method	Percentage	DGEMM
		0
gemvn_kernel	99.16	
reduction_hip_tiled	0.74	
temp_accumulate	0.07	
rocblas_reduction_strided	0.015	

120X faster than R code

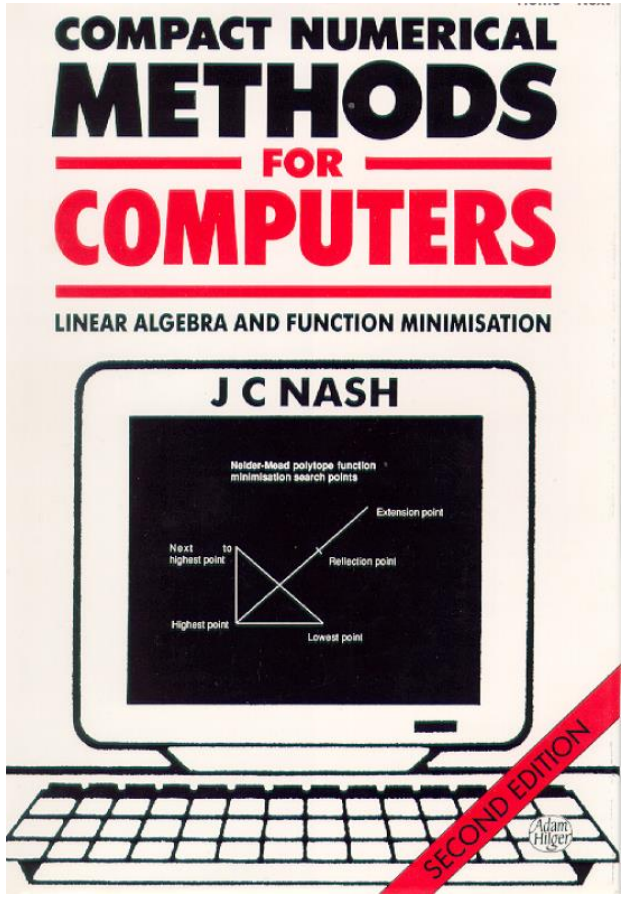
Matrix-vector:

Non-blocking Communication On CPU

```
double hinge_loss(double * Weights, int hipDeviceRank)
{
   hipblasDgemv(handle_global, HIPBLAS_OP_N, MObjects, NFeatures, &alpha, DataMatrix_Device, MObjects, Weights, 1, &beta, ComputedSpecies_Device, 1);

   double out = reduction(ComputedSpecies_Device, Species_Device, MObjects, hipDeviceRank);

   MPI_Request request;
   MPI_Iallreduce(MPI_IN_PLACE, &out, 1, MPI_DOUBLE, MPI_SUM, MPI_COMM_WORLD, &request);

   double norm = 0.0;
   hipblasDnrm2(handle, NFeatures, Weights, 1, &norm);
   MPI_Wait(&request, MPI_STATUS_IGNORE);

   out = 1.0/regularizationParameter*(out + 0.5*norm); // L2 Regularization

 return out;
}
```

ACCELERATED SVM (20% GEMM)

Nelder-Mead schematic on the front page!



2nd call to hinge loss: Pascal Code (or think of it as pseudocode):

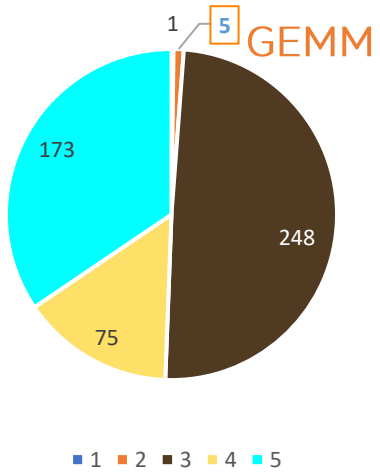
```
...
-- STEP 10
if calcvert then
begin
for j := 1 to nl do
   begin
   if j<>L then
   begin
   for i := 1 to n do Bvec[i] := P[i,j];
   f:= hinge_loss(n,Bvec,Workdata,notcomp);
   ...
```

A DGEMv inside a loop = DGEMM

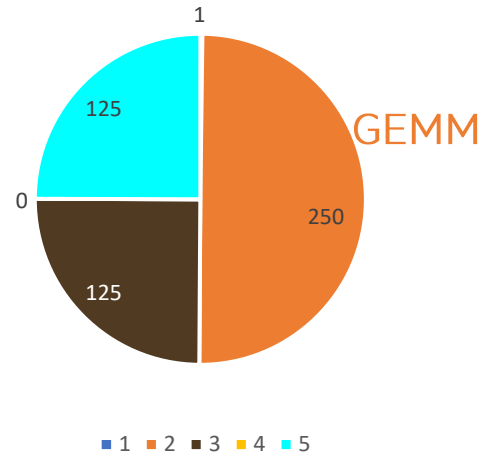
ACCELERATED SVM (20% GEMM)

```
Initialize: hinge_loss (1)
Loop hinge_loss (2) DGEMM-like
if (Highest > Lowest && Lowest > tolerance)
     hinge_loss (3)
     if (Reflection < Lowest)
          hinge_loss (4)
     else (Reflection >= Lowest)
          hinge_loss (5)
endif
```

Iris dataset



Large dataset (random data)



Inconvenient:
Data_Matrix must be placed inside Nelder-Mead

```
void NelderMead_GEMM(const int  NFeatures,
                int             MObjects_rank,
                realtype         *  DataMatrix_Device,
                realtype         *  ComputedSpecies_Device,
                realtype         *  Species_Device,
                realtype *      Weights_Device,
                realtype *      hipWeights,      /* output (hipWeights) from minmeth */
                realtype &      Fmin,    /* b^A^Xminimumb^A^Y function value */
                int &           fail,    /* true if method has failed */
                realtype        abstol,
                realtype        intol,
                int &           fncount,
                int             kcount,
                int             hipDeviceRank,
                hipblasHandle_t handle,
                realtype        MPI_HingeLoss(realtype * Weights, realtype out),
                realtype reduction(realtype * a_d,realtype * y_d,const int ni, int hipDeviceRank),
                realtype    hinge_loss ( realtype x[], int hipDeviceRank)
               ) {
```

CONCLUSIONS

- The accel-BDAS code is easily 2 orders of magnitudes faster than the original CPU-only version.
- Tensile optimization is responsible for one order of magnitude of that improvement.
- Matrices of special shapes require extra optimization. ROCm already includes optimized GEMM operations for standard (square) matrices.
- More recent algorithms: the accel-BDAS can easily be expanded to other algorithms: K-Means++, SVD, non-linear SVM.
- Half-precision [#include <hip/hip_fp16.h>; __half; __half2] formats should provide significant performance improvement in future AMD accelerators. Accel-BDAS already treats both single and double precision data.
- Once HIP-python is available, integration of python/cython into BDAS will be relatively straightforward.
- Once HIP-Python is available, python-based communication scheme, such as multiprocessing; dragon can easily be integrated into accel-BDAS, because MPI communication is kept minimal.
- The optimized BDAS source has been provided to Oak Ridge National Laboratory.
  - Contact me Pierre.Carrier@hpe.com for further information on how to obtain the optimized BDAS source.

THANK YOU!

pierre.carrier@hpe.com [on BDAS and/or Tensile]

alessandro.fanfarillo@amd.com [on Tensile]

CONFIDENTIAL DISCLOSURE AGREEMENT

- The information contained in this presentation is proprietary to Hewlett Packard enterprise and is offered in confidence, subject to the terms and conditions of a binding Confidential Disclosure Agreement (CDA)
- HPE requires customers and partners to have signed a CDA in order to view this training
- The information contained in this training is HPE confidential
- This presentation is NOT to be used as a 'leave behind' for customers and information may only be shared verbally with HPE external customers under NDA
- This presentation may be shared with Partners under NDA in hard-copy or electronic format for internal training purposes only
- Do not remove any classification labels, warnings or disclaimers on any slide or modify this presentation to change the classification level
- Do not remove this slide from the presentation
- HPE does not warrant or represent that it will introduce any product to which the information relates
- The information contained herein is subject to change without notice
- HPE makes no warranties regarding the accuracy of this information
- The only warranties for HPE products and services are set forth in the express warranty statements accompanying such products and services
- Nothing herein should be construed as constituting an additional warranty
- HPE shall not be liable for technical or editorial errors or omissions contained herein
- Strict adherence to the HPE Standards of Business Conduct regarding this classification level is critical

TRIPLET OF SYNCHRONIZATION: CPU + NETWORK + GPU

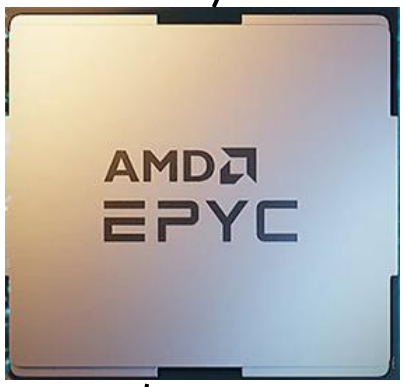
Details in CUG paper.

MPI

https://www.hpe.com/us/en/compute/hpc/slingshot-interconnect.html

GPU-to-GPU

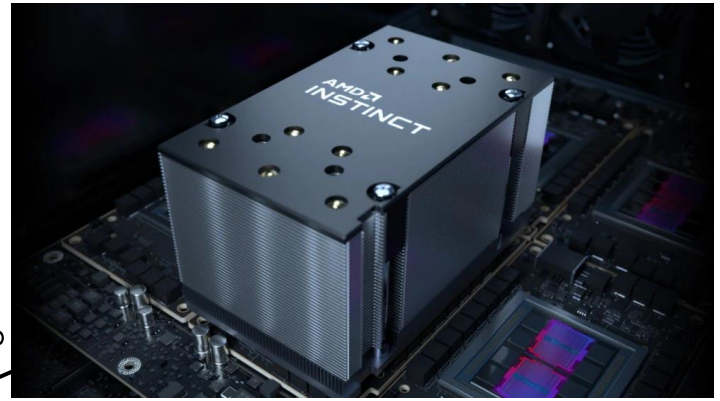
C++



https://www.amd.com/en/events/epyc4

```
export LOCAL_RANK=$SLURM_LOCALID
...
#include "mpi.h"
#include <hip/hip_runtime.h>
#include "hipblas.h"
#include "rocblas.h"
...
int main(int argc, const char** argv){

// MPI Initialization
MPI_Init( NULL, NULL);
int num_ranks, rank;
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &num_ranks);

// HIP + ROCm Initialization
int dev, dev_count;
char* str;
hipError_t hip_result;
if ((str = getenv("LOCAL_RANK")) != NULL) {
   hipGetDeviceCount(&dev_count);
   int local_rank = std::atoi(str);
   dev = (local_rank % dev_count);
}
hip_result = hipInit(0);
if (hip_result != hipSuccess) {return 1;}
hip_result = hipSetDevice(dev);
if (hip_result != hipSuccess) {return 1;}
hipblasHandle_t handle;
hipblasCreate(&handle);
rocblas_initialize();
```

HIP



https://www.amd.com/en/products/server-accelerators/instinct-mi250x

TRIPLET OF SYNCHRONIZATION: CPU + NETWORK + GPU

Details in CUG paper.

Simplest example of triplet of synchronization, in K-Means:

C++ HIP initialize

```
...
double*      ClusterCentroids_OnDevice;

hipMalloc((void**)&ClusterCentroids_OnDevice, NFeatures*KClusters*sizeof(realtype))

...
```

HIP "wavefront" on device

```
hipblasDgemm(handle, HIPBLAS_OP_T, HIPBLAS_OP_N, NFeatures, KClusters, MObjects_rank,&alpha,
             DataMatrix_OnDevice, MObjects_rank,
             ClusterID_OnDevice, MObjects_rank, &beta, ClusterCentroids_OnDevice, NFeatures);
```

Synchronize GPU wavefronts

```
hipDeviceSynchronize();
```

MPI collective

```
MPI_Allreduce(MPI_IN_PLACE, &ClusterCentroids_OnDevice[0], NFeatures*KClusters, MPI_DOUBLE, MPI_SUM, MPI_COMM_WORLD);
```

Synchronize MPI communication (implicit in that case)

```
// The MPI_Barrier() is implicit
...
```

Other synchronization examples in PCA and SVM: details in CUG paper