Package 'RHPCBenchmark'

January 20, 2025

Title Benchmarks for High-Performance Computing Environments

Version 0.1.0

Author James McCombs [aut, cre]

Maintainer James McCombs <jmccombs@iu.edu>

Description Microbenchmarks for determining the run time performance of aspects of the R programming environment and packages relevant to high-performance computation. The benchmarks are divided into three categories: dense matrix linear algebra kernels, sparse matrix linear algebra kernels, and machine learning functionality.

Depends R (>= 3.3.1), methods

Imports utils, mvtnorm, cluster, Matrix

License Apache License 2.0 | file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 6.0.1

Suggests knitr, rmarkdown

VignetteBuilder knitr

NeedsCompilation no

Repository CRAN

Date/Publication 2017-05-23 17:26:28 UTC

Contents

`CholeskyAllocator` *Allocates and initializes input to the Cholesky factorization dense matrix kernel microbenchmarks*

Description

`CholeskyAllocator` allocates and populates the input to the Cholesky factorization dense matrix kernel for the purposes of conducting a single performance trial with the `CholeskyMicrobenchmark` function. The matrices or vectors corresponding to the `index` parameter must be allocated, populated and returned in the `kernelParameters` list.

Usage

```
CholeskyAllocator(benchmarkParameters, index)
```

Arguments

`benchmarkParameters`	an object of type `DenseMatrixMicrobenchmark` specifying various parameters needed to generate input for the dense matrix kernel.
`index`	an integer index indicating the dimensions of the matrix or vector data to be generated as input for the dense matrix kernel.

Value

a list containing the matrices or vectors to be input for the dense matrix kernel for which a single performance trial is to be conducted.

CholeskyMicrobenchmark *Conducts a single performance trial with the Cholesky factorization dense matrix kernel*

Description

CholeskyMicrobenchmark conducts a single performance trial of the Cholesky factorization dense matrix kernel for the matrix given in the kernelParameters parameter. The function times the single function call chol(kernelParameters$A).

Usage

```
CholeskyMicrobenchmark(benchmarkParameters, kernelParameters)
```

Arguments

benchmarkParameters
: an object of type DenseMatrixMicrobenchmark specifying various parameters for microbenchmarking the dense matrix kernel

kernelParameters
: a list of matrices or vectors to be used as input to the dense matrix kernel

Value

a vector containing the user, system, and elapsed performance timings in that order

Examples

```
## Not run:
# Allocate input to the Cholesky microbenchmark for the first matrix size
# to be tested
microbenchmarks <- GetDenseMatrixDefaultMicrobenchmarks()
kernelParameters <- CholeskyAllocator(microbenchmarks[["cholesky"]], 1)
# Execute the microbenchmark
timings <- CholeskyMicrobenchmark(microbenchmarks[["cholesky"]], kernelParameters)

## End(Not run)
```

ClaraClusteringMicrobenchmark *Conducts a single performance trial with the cluster::clara function*

Description

ClusteringMicrobenchmark conducts a single performance trial of the cluster::clara function with the data given in the kernelParameters parameter.

Usage

```
ClaraClusteringMicrobenchmark(benchmarkParameters, kernelParameters)
```

Arguments

benchmarkParameters
: an object of type ClusteringMicrobenchmark specifying various parameters for microbenchmarking the cluster::clara function

kernelParameters
: a list of data objects to be used as input to the clustering function

Value

a vector containing the user, system, and elapsed performance timings in that order

Examples

```
## Not run:
# Allocate input to the pam clustering microbenchmark
microbenchmarks <- GetClusteringExampleMicrobenchmarks()
kernelParameters <- ClusteringAllocator(microbenchmarks[["clara_cluster_3_3_1000"]])
# Execute the microbenchmark
timings <- ClaraClusteringMicrobenchmark(
   microbenchmarks[["clara_cluster_3_3_1000"]], kernelParameters)

## End(Not run)
```

ClusteringAllocator *Allocates and initializes input to the clustering for machine learning microbenchmarks*

Description

ClusteringAllocator allocates and initializes the data sets that are input to the clustering microbenchmarks for the purposes of conducting a single performance trial with one of the clustering microbenchmark functions.

Usage

```
ClusteringAllocator(benchmarkParameters)
```

Arguments

benchmarkParameters
: an object of type ClusteringMicrobenchmark specifying various parameters needed to generate input for the clustering microbenchmarks.

Value

a list containing the data objects to be input to the clustering microbenchmark

See Also

PamClusteringMicrobenchmark ClaraClusteringMicrobenchmark

ClusteringMicrobenchmark

This class specifies a clustering for machine learning microbenchmark.

Description

This class specifies a clustering for machine learning microbenchmark.

Fields

active a logical indicating whether the microbenchmark is to be executed (TRUE) or not (FALSE).

benchmarkName a character string that is the name of the microbenchmark.

benchmarkDescription a character string describing the microbenchmark.

dataObjectName a character string specifying the name of the data object that is input to the benchmark; the object must be stored in the R data file with the same base name and a .RData extension. Setting the field to NA_character_ indicates that the test data will be dynamically generated by the function given in the allocatorFunction field instead of read from a data file.

numberOfFeatures the number features; this value must match the number of features in the data set given by the field dataObjectName unless the field is populated with NA_character_.

numberOfClusters the number of clusters in the data set; this value must match the number of clusters in the data set given by the field dataObjectName unless the field is populated with NA_character_.

numberOfFeatureVectorsPerCluster the number of feature vectors per cluster; this value must match the number of clusters in the data set given by the field dataObjectName unless the field is populated with NA_character_.

`numberOfTrials` an integer specifying the number of performance trials conducted on the data set to be tested.

`numberOfWarmupTrials` an integer specifying the number of warmup trials to be conducted on the data set.

`allocatorFunction` the function that allocates and initializes input to the benchmark function. The function takes a `ClusteringMicrobenchmark` object. For clustering benchmarks, the allocator function should return a list containing the following items:

featureVectors a matrix, the rows of which are the feature vectors

numberOfFeatures an integer indicating the number of features

numberOfFeatureVectors an integer indicating the number of feature vectors

numberOfClusters an integer indicating the number of clusters in the data set

`benchmarkFunction` the benchmark function which executes the functionality to be timed. The function takes a `SparseMatrixMicrobenchmark` and a list of kernel parameters returned by the allocator function.

`ComputeAverageTime` *Computes the average of a vector of performance trial times*

Description

`ComputeAverageTime` computes the average of a vector of performance trial times. The average is computed only over the first `numberOfSuccessfulTrials` elements of the `times` vector.

Usage

```
ComputeAverageTime(numberOfSuccessfulTrials, times)
```

Arguments

`numberOfSuccessfulTrials`	the number of successful performance trials to be averaged over
`times`	a vector of wall clock times for the performance trials

`ComputeStandardDeviation`
Computes the standard deviation of a vector of performance trial times

---

Description

`ComputeStandardDeviation` computes the standard deviation of a vector of performance trial times. The standard deviation is computed only over the first `numberOfSuccessfulTrials` elements of the `times` vector.

Usage

```
ComputeStandardDeviation(numberOfSuccessfulTrials, times)
```

Arguments

numberOfSuccessfulTrials	the number of successful performance trials over which the standard deviation will be computed
times	a vector of wall clock times for the performance trials

---

CrossprodAllocator *Allocates and populates input to the matrix cross product dense matrix kernel microbenchmarks*

Description

CrossprodAllocator allocates and populates the input to the matrix cross product dense matrix kernel for the purposes of conducting a single performance trial with the CrossprodMicrobenchmark function. The matrices or vectors corresponding to the index parameter must be allocated, initialized and returned in the kernelParameters list.

Usage

```
CrossprodAllocator(benchmarkParameters, index)
```

Arguments

benchmarkParameters	an object of type DenseMatrixMicrobenchmark specifying various parameters needed to generate input for the dense matrix kernel.
index	an integer index indicating the dimensions of the matrix or vector data to be generated as input for the dense matrix kernel.

CrossprodMicrobenchmark *Conducts a single performance trial with the matrix cross product dense matrix kernel*

Description

CrossprodMicrobenchmark conducts a single performance trial of the matrix cross product dense matrix kernel for the matrix given in the kernelParameters parameter. The function times the single function call crossprod(kernelParameters$A).

Usage

```
CrossprodMicrobenchmark(benchmarkParameters, kernelParameters)
```

Arguments

benchmarkParameters	an object of type DenseMatrixMicrobenchmark specifying various parameters for microbenchmarking the dense matrix kernel
kernelParameters	a list of matrices or vectors to be used as input to the dense matrix kernel

Examples

```
## Not run:
# Allocate input to the matrix cross product microbenchmark for the first
# matrix size to be tested
microbenchmarks <- GetDenseMatrixDefaultMicrobenchmarks()
kernelParameters <- CrossprodAllocator(microbenchmarks[["crossprod"]], 1)
# Execute the microbenchmark
timings <- CrossprodMicrobenchmark(microbenchmarks[["crossprod"]], kernelParameters)

## End(Not run)
```

DeformtransAllocator *Allocates and populates input to the matrix deformation and transpose dense matrix kernel microbenchmarks*

Description

DeformtransAllocator allocates and populates the input to the matrix deformation and transpose dense matrix kernel for the purposes of conducting a single performance trial with the DeformtransMicrobenchmark function. The matrices or vectors corresponding to the index parameter must be allocated, initialized and returned in the kernelParameters list.

Usage

```
DeformtransAllocator(benchmarkParameters, index)
```

Arguments

benchmarkParameters	an object of type DenseMatrixMicrobenchmark specifying various parameters needed to generate input for the dense matrix kernel.
index	an integer index indicating the dimensions of the matrix or vector data to be generated as input for the dense matrix kernel.

DeformtransMicrobenchmark

Conducts a single performance trial with the matrix deformation and transpose dense matrix kernel

---

Description

DeformtransMicrobenchmark conducts a single performance trial of the matrix deformation and transpose dense matrix kernel for the matrix given in the kernelParameters parameter. The function times the transposition of the input matrix, resizing of the input matrix, and transposition of the resized matrix.

Usage

```
DeformtransMicrobenchmark(benchmarkParameters, kernelParameters)
```

Arguments

benchmarkParameters
: an object of type DenseMatrixMicrobenchmark specifying various parameters for microbenchmarking the dense matrix kernel

kernelParameters
: a list of matrices or vectors to be used as input to the dense matrix kernel

Examples

```
## Not run:
# Allocate input to the matrix deformation and transpose microbenchmark for
# the first matrix size to be tested
microbenchmarks <- GetDenseMatrixDefaultMicrobenchmarks()
kernelParameters <- DeformtransAllocator(microbenchmarks[["deformtrans"]], 1)
# Execute the microbenchmark
timings <- DeformtransMicrobenchmark(microbenchmarks[["deformtrans"]], kernelParameters)

## End(Not run)
```

DenseMatrixMicrobenchmark

This class specifies a dense matrix microbenchmark.

Description

This class specifies a dense matrix microbenchmark.

Fields

`active` a logical indicating whether the microbenchmark is to be executed (TRUE) or not (FALSE).

`benchmarkName` a character string that is the name of the microbenchmark.

`benchmarkDescription` a character string describing the microbenchmark.

`dimensionParameters` an integer vector specifying the dimension parameters the microbenchmark uses to define the matrix dimensions to be tested with.

`numberOfTrials` an integer vector specifying the number of performance trials conducted for each matrix to be tested. Must be the same length as `dimensionParameters`.

`numberOfWarmupTrials` an integer vector specifying the number of warmup trials to be performed for each matrix to be tested.

`allocatorFunction` the function that allocates and initializes input to the benchmark function. The function takes a `DenseMatrixMicrobenchmark` object and an integer index indicating which matrix dimension parameter from `dimensionParameters` should be used to generate the matrix.

`benchmarkFunction` the benchmark function which executes the functionality to be timed. The function takes a `DenseMatrixMicrobenchmark` and a list of kernel parameters returned by the allocator function.

`DeterminantAllocator` *Allocates and populates input to the matrix determinant dense matrix kernel microbenchmarks*

Description

`DeterminantAllocator` allocates and populates the input to the matrix determinant dense matrix kernel for the purposes of conducting a single performance trial with the `DeterminantMicrobenchmark` function. The matrices or vectors corresponding to the `index` parameter must be allocated, initialized and returned in the `kernelParameters` list.

Usage

```
DeterminantAllocator(benchmarkParameters, index)
```

Arguments

`benchmarkParameters`
: an object of type `DenseMatrixMicrobenchmark` specifying various parameters needed to generate input for the dense matrix kernel.

`index`
: an integer index indicating the dimensions of the matrix or vector data to be generated as input for the dense matrix kernel.

DeterminantMicrobenchmark *Conducts a single performance trial with the matrix determinant dense matrix kernel*

Description

DeterminantMicrobenchmark conducts a single performance trial of the dense matrix determinant dense matrix kernel for the matrix given in the kernelParameters parameter. The function times the single function call determinant(kernelParameters$A).

Usage

```
DeterminantMicrobenchmark(benchmarkParameters, kernelParameters)
```

Arguments

benchmarkParameters
: an object of type DenseMatrixMicrobenchmark specifying various parameters for microbenchmarking the dense matrix kernel

kernelParameters
: a list of matrices or vectors to be used as input to the dense matrix kernel

Examples

```
## Not run:
# Allocate input to the matrix determinant microbenchmark for the first
# matrix size to be tested
microbenchmarks <- GetDenseMatrixDefaultMicrobenchmarks()
kernelParameters <- DeterminantAllocator(microbenchmarks[["determinant"]], 1)
# Execute the microbenchmark
timings <- DeterminantMicrobenchmark(microbenchmarks[["determinant"]], kernelParameters)

## End(Not run)
```

EigenAllocator *Allocates and populates input to the matrix eigendecomposition kernel microbenchmarks*

Description

EigenAllocator allocates and populates the input to the matrix eigendecomposition dense matrix kernel for the purposes of conducting a single performance trial with the EigenMicrobenchmark function. The matrices or vectors corresponding to the index parameter must be allocated, initialized and returned in the kernelParameters list.

Usage

```
EigenAllocator(benchmarkParameters, index)
```

Arguments

benchmarkParameters	an object of type DenseMatrixMicrobenchmark specifying various parameters needed to generate input for the dense matrix kernel.
index	an integer index indicating the dimensions of the matrix or vector data to be generated as input for the dense matrix kernel.

EigenMicrobenchmark *Conducts a single performance trial with the matrix eigendecomposition dense matrix kernel*

Description

EigenMicrobenchmark conducts a single performance trial of the matrix eigendecomposition dense matrix kernel for the matrix given in the kernelParameters parameter. The function times the single function call eigen(kernelParameters$A, symmetric=FALSE, only.values=FALSE).

Usage

```
EigenMicrobenchmark(benchmarkParameters, kernelParameters)
```

Arguments

benchmarkParameters	an object of type DenseMatrixMicrobenchmark specifying various parameters for microbenchmarking the dense matrix kernel
kernelParameters	a list of matrices or vectors to be used as input to the dense matrix kernel

Examples

```
## Not run:
# Allocate input to the matrix eigendecomposition microbenchmark for the
# first matrix size to be tested
microbenchmarks <- GetDenseMatrixDefaultMicrobenchmarks()
kernelParameters <- EigenAllocator(microbenchmarks[["eigen"]], 1)
# Execute the microbenchmark
timings <- EigenMicrobenchmark(microbenchmarks[["eigen"]], kernelParameters)

## End(Not run)
```

GenerateClusterData *Generates clusters from multivariate normal distributions*

Description

GenerateClusterData generates clusters of feature vectors drawn from multivariate normal (MVN) distributions. The mean values of the normal distribution corresponding to the first cluster is always at the origin. The remaining clusters are generated from MVN distributions with mean values at v_i and -v_i where v_i is the i-th unit vector. The clusters are generated in the following order by mean value of the MVN for each cluster: origin, v_1, -v_1, v_2, -v_2, v_3, -v_3,..., v_(numberOfClusters-1)/2, -v_(numberOfClusters-1)/2 (if numberOfClusters is odd) origin, v_1, -v_1, v_2, -v_2, v_3, -v_3,..., v_(numberOfClusters-1)/2 (if numberOfClusters is even).

Usage

```
GenerateClusterData(numberOfFeatures, numberOfVectorsPerCluster,
  numberOfClusters = 2 * numberOfFeatures + 1)
```

Arguments

numberOfFeatures
: the number of features, the dimension of the feature space

numberOfVectorsPerCluster
: the number of vectors to randomly generate for each cluster

numberOfClusters
: the number of clusters to be generated. The value of this parameter must be in the interval [1,2*numberOfFeatures+1]

Value

a list containing a matrix of feature vectors featureVectors as rows of feature vectors, number of features numberOfFeatures, number of feature vectors numberOfFeatureVectors, and number of clusters numberOfClusters.

GetClusteringDefaultMicrobenchmarks
Initializes the list of default clustering microbenchmarks

Description

GetClusteringDefaultMicrobenchmarks defines the default clustering microbenchmarks to be executed by the RunMachineLearningBenchmark function. The current clustering microbenchmarks are:

1. pam_cluster_3_7_2500N=3, seven clusters with 2500 vectors per cluster, using pam function

2. pam_cluster_3_7_5000N=3, seven clusters with 5000 vectors per cluster, using pam function
3. pam_cluster_3_7_5715N=3, seven clusters with 5715 vectors per cluster, using pam function
4. pam_cluster_16_33_1213N=16, 33 clusters with 1213 vectors per cluster, using pam function
5. pam_cluster_64_33_1213N=64, 33 clusters with 1213 vectors per cluster, using pam function
6. pam_cluster_16_7_2858N=16, seven clusters with 2858 vectors per cluster, using pam function
7. pam_cluster_32_7_2858N=32, seven clusters with 2858 vectors per cluster, using pam function
8. pam_cluster_64_7_5715,N=64, seven clusters with 5715 vectors per cluster, using pam function
9. clara_cluster_64_33_1213N=64, 33 clusters with 1213 vectors per cluster, using clara function
10. clara_cluster_1000_99_1000N=1000, 99 clusters with 1000 vectors per cluster, using clara function

The pam and pam microbenchmarks test those clustering functions. The pam function applies a quadratic time algorithm to partition around medoids (pam); the clara function is a linear time approximation to the partitioning around medoids algorithm. See the documentation for the ClusteringMicrobenchmark class for more details.

Usage

```
GetClusteringDefaultMicrobenchmarks()
```

Value

a list of ClusteringMicrobenchmark objects defining the microbenchmarks to be executed. The microbenchmarks appear in the order listed in the function description and are assigned the names enumerated in the description.

See Also

ClusteringMicrobenchmark pam pam

Other machine learning default microbenchmarks: GetClusteringExampleMicrobenchmarks

---

GetClusteringExampleMicrobenchmarks
Initializes the list of example clustering microbenchmarks

Description

GetClusteringExampleMicrobenchmarks defines the example clustering microbenchmarks to be executed by the RunMachineLearningBenchmark function. The example are chosen so that they can run in a few minutes or less.

1. pam_cluster_3_7_2500N=3, seven clusters with 2500 vectors per cluster
2. clara_cluster_64_33_1213N=64, 33 clusters with 1213 vectors per cluster

See the documentation for the ClusteringMicrobenchmark class for more details.

Usage

```
GetClusteringExampleMicrobenchmarks()
```

Value

a list of ClusteringMicrobenchmark objects defining the microbenchmarks to be executed. Microbenchmarks for the pam and clara functions from the cluster package are provided.

See Also

Other machine learning default microbenchmarks: GetClusteringDefaultMicrobenchmarks

---

GetConfigurableEnvParameter

Retrieves the value of an environment variable referenced by another environment variable

---

Description

GetConfigurableEnvParameters returns the value of the environment variable referenced by the argument configurableVariable which is also an environment variable

Usage

```
GetConfigurableEnvParameter(configurableVariable)
```

Arguments

configurableVariable	a string parameter containing the name of an environment variable which itself references another environment variable

Details

This function takes the argument configurableVariable which contains the name of an environment variable whose value is an environment variable referencing a value to be returned by this function.

Value

the value of the environment variable referenced by the environment variable specified in the `configurableVariable` parameter

---

`GetDenseMatrixDefaultMicrobenchmarks`

Initializes the list of default dense matrix microbenchmarks

Description

`GetDenseMatrixDefaultMicrobenchmarks` defines the default dense matrix microbenchmarks to be executed by the `RunDenseMatrixBenchmark` function. The current microbenchmarks are Cholesky factorization, matrix cross product, matrix determinant, eigendecomposition, linear solve with multiple right hand sides, least squares fit, matrix deformation and transpose, matrix-matrix multiplication, matrix-vector multiplication, QR decomposition, and singular value decomposition. See the documentation for the `DenseMatrixMicrobenchmark` class for more details.

Usage

```
GetDenseMatrixDefaultMicrobenchmarks()
```

Value

a list of `DenseMatrixMicrobenchmark` objects defining the microbenchmarks to be executed. The microbenchmarks appear in the order listed in the function description and are assigned the following names: cholesky, crossprod, determinant, eigen, solve, lsfit, deformtrans, transpose, matmat, matvec, qr, and svd.

See Also

`DenseMatrixMicrobenchmark`

`GetDenseMatrixExampleMicrobenchmarks`

Initializes the list of example dense matrix microbenchmarks

Description

`GetDenseMatrixExampleMicrobenchmarks` defines example dense matrix microbenchmarks to be executed by the examples section of the `RunDenseMatrixBenchmark` function. The examples are chosen so that they can run in a few minutes or less.

Usage

```
GetDenseMatrixExampleMicrobenchmarks()
```

Value

a list of DenseMatrixMicrobenchmark objects defining the microbenchmarks to be executed. Microbenchmarks for Cholesky factorization and matrix cross product are provided.

GetNumberOfThreads *Retrieves the number of threads from the environment*

Description

GetNumberOfThreads retrieves from the environment the number of threads kernels are intended to be executed with

Usage

```
GetNumberOfThreads()
```

Details

This function retrieves the number of threads kernels are intended to be microbenchmarked with. The number of threads is assumed to be stored in an environment variable which this function retrieves.

Value

the number of threads retrieved from the environment

GetSparseCholeskyDefaultMicrobenchmarks *Initializes the list of default sparse Cholesky factorization microbenchmarks*

Description

GetSparseCholeskyDefaultMicrobenchmarks defines the default sparse Cholesky factorization microbenchmarks to be executed by the RunSparseMatrixBenchmark function. The current sparse Cholesky factorization microbenchmarks cover a variety of matrices of different dimensions and number of non-zero values. They are as follows:

1. cholesky_ct20stif – Boeing structural matrix with 2600295 nonzeros
2. cholesky_Andrews – computer vision matrix with 760154
3. cholesky_G3_circuit – AMD circuit simulation matrix with 7660826 nonzeros

See the documentation for the SparseMatrixMicrobenchmark class for more details.

Usage

```
GetSparseCholeskyDefaultMicrobenchmarks()
```

Value

a list of SparseMatrixMicrobenchmark objects defining the microbenchmarks to be executed. The microbenchmarks appear in the order listed in the function description and are assigned the names enumerated in the description.

See Also

SparseMatrixMicrobenchmark

Other sparse matrix default microbenchmarks: GetSparseCholeskyExampleMicrobenchmarks, GetSparseLuDefaultMicrobenchmarks, GetSparseMatrixVectorDefaultMicrobenchmarks, GetSparseMatrixVectorE GetSparseQrDefaultMicrobenchmarks

---

GetSparseCholeskyExampleMicrobenchmarks

Initializes the list of example sparse Cholesky factorization microbenchmarks

---

Description

GetSparseCholeskyExampleMicrobenchmarks defines the example sparse Cholesky factorization microbenchmarks to be executed by the RunSparseMatrixBenchmark function. The current sparse Cholesky factorization microbenchmarks cover the following matrices:

1. cholesky_ct20stif – Boeing structural matrix with 2600295 nonzeros

See the documentation for the SparseMatrixMicrobenchmark class for more details.

Usage

```
GetSparseCholeskyExampleMicrobenchmarks()
```

Value

a list of SparseMatrixMicrobenchmark objects defining the microbenchmarks to be executed. The microbenchmark for the Cholesky factorization of the ct20stif matrix.

See Also

Other sparse matrix default microbenchmarks: GetSparseCholeskyDefaultMicrobenchmarks, GetSparseLuDefaultMicrobenchmarks, GetSparseMatrixVectorDefaultMicrobenchmarks, GetSparseMatrixVectorE GetSparseQrDefaultMicrobenchmarks

---

GetSparseLuDefaultMicrobenchmarks
Initializes the list of default sparse LU factorization microbenchmarks

---

Description

GetSparseLuDefaultMicrobenchmarks defines the default sparse LU factorization microbenchmarks to be executed by the RunSparseMatrixBenchmark function. The current sparse LU factorization microbenchmarks cover a variety of matrices of different dimensions and number of non-zero values. They are as follows:

1. lu_circuit5M_dc – Freescale DC circuit simulation matrix 2600295 nonzeros
2. lu_stomach – 3D electro-physical model matrix with 3021648 nonzeros
3. lu_torso3 – 3D electro-physical model matrix with 4429042 nonzeros

See the documentation for the SparseMatrixMicrobenchmark class for more details.

Usage

```
GetSparseLuDefaultMicrobenchmarks()
```

Value

a list of SparseMatrixMicrobenchmark objects defining the microbenchmarks to be executed. The microbenchmarks appear in the order listed in the function description and are assigned the names enumerated in the description.

See Also

SparseMatrixMicrobenchmark

Other sparse matrix default microbenchmarks: GetSparseCholeskyDefaultMicrobenchmarks, GetSparseCholeskyExampleMicrobenchmarks, GetSparseMatrixVectorDefaultMicrobenchmarks, GetSparseMatrixVectorExampleMicrobenchmarks, GetSparseQrDefaultMicrobenchmarks

---

GetSparseMatrixVectorDefaultMicrobenchmarks
Initializes the list of default sparse matrix-vector microbenchmarks

---

Description

GetSparseMatrixVectorDefaultMicrobenchmarks defines the default sparse matrix-vector microbenchmarks to be executed by the RunSparseMatrixBenchmark function. The current sparse matrix-vector microbenchmarks cover a variety of matrices of different dimensions and number of non-zero values. They are as follows:

1. matvec_laplacian7pt_100 – 100x100x100 7-point Laplacian operator
2. matvec_laplacian7pt_200 – 200x200x200 7-point Laplacian operator
3. matvec_ca2010 – DIMACS10/ca2010 710145x710145 undirected graph matrix

See the documentation for the SparseMatrixMicrobenchmark class for more details.

Usage

```
GetSparseMatrixVectorDefaultMicrobenchmarks()
```

Value

a list of SparseMatrixMicrobenchmark objects defining the microbenchmarks to be executed. The microbenchmarks appear in the order listed in the function description and are assigned the names enumerate in the description.

See Also

SparseMatrixMicrobenchmark

Other sparse matrix default microbenchmarks: GetSparseCholeskyDefaultMicrobenchmarks, GetSparseCholeskyExampleMicrobenchmarks, GetSparseLuDefaultMicrobenchmarks, GetSparseMatrixVectorExamp
GetSparseQrDefaultMicrobenchmarks

GetSparseMatrixVectorExampleMicrobenchmarks

Initializes the list of example sparse matrix-vector microbenchmarks

---

Description

GetSparseMatrixVectorExampleMicrobenchmarks defines example sparse matrix-vector microbenchmarks to be executed by the RunSparseMatrixBenchmark function. The example matrix-vector microbenchmarks cover the following matrices:

1. matvec_laplacian7pt_100 – 100x100x100 7-point Laplacian operator

See the documentation for the SparseMatrixMicrobenchmark class for more details.

Usage

```
GetSparseMatrixVectorExampleMicrobenchmarks()
```

Value

a list of SparseMatrixMicrobenchmark objects defining the microbenchmarks to be executed. The microbenchmark for matrix-vector operations with the Laplacian operator laplacian7pt_100 is returned.

See Also

Other sparse matrix default microbenchmarks: GetSparseCholeskyDefaultMicrobenchmarks, GetSparseCholeskyExampleMicrobenchmarks, GetSparseLuDefaultMicrobenchmarks, GetSparseMatrixVectorDefau GetSparseQrDefaultMicrobenchmarks

---

GetSparseQrDefaultMicrobenchmarks

Initializes the list of default sparse QR factorization microbenchmarks

---

Description

GetSparseQrDefaultMicrobenchmarks defines the default sparse QR factorization microbenchmarks to be executed by the RunSparseMatrixBenchmark function. The current sparse QR factorization microbenchmarks cover a variety of matrices of different dimensions and number of non-zero values. They are as follows:

1. Maragal_6 – rank deficient least squares matrix of 537694 nonzeros
2. landmark – least squares matrix of 1146848

See the documentation for the SparseMatrixMicrobenchmark class for more details.

Usage

```
GetSparseQrDefaultMicrobenchmarks()
```

Value

a list of SparseMatrixMicrobenchmark objects defining the microbenchmarks to be executed. The microbenchmarks appear in the order listed in the function description and are assigned the names enumerated in the description.

See Also

SparseMatrixMicrobenchmark

Other sparse matrix default microbenchmarks: GetSparseCholeskyDefaultMicrobenchmarks, GetSparseCholeskyExampleMicrobenchmarks, GetSparseLuDefaultMicrobenchmarks, GetSparseMatrixVectorDefau GetSparseMatrixVectorExampleMicrobenchmarks

---

`LsfitAllocator` *Allocates and populates input to the matrix least squares fit dense matrix kernel microbenchmarks*

Description

`LsfitAllocator` allocates and populates the input to the matrix least squares fit dense matrix kernel for the purposes of conducting matrices or vectors corresponding to the `index` parameter must be allocated, initialized and returned in the `kernelParameters` list.

Usage

```
LsfitAllocator(benchmarkParameters, index)
```

Arguments

`benchmarkParameters`	an object of type `DenseMatrixMicrobenchmark` specifying various parameters needed to generate input for the dense matrix kernel.
`index`	an integer index indicating the dimensions of the matrix or vector data to be generated as input for the dense matrix kernel.

`LsfitMicrobenchmark` *Conducts a single performance trial with the matrix least squares fit dense matrix kernel*

Description

`LsfitMicrobenchmark` conducts a single performance trial of the matrix least squares fit dense matrix kernel for the matrix given in the `kernelParameters` parameter. The function times the single function call `lsfit(kernelParameters$A, kernelParameters$b, intercept=FALSE)`.

Usage

```
LsfitMicrobenchmark(benchmarkParameters, kernelParameters)
```

Arguments

`benchmarkParameters`	an object of type `DenseMatrixMicrobenchmark` specifying various parameters for microbenchmarking the dense matrix kernel
`kernelParameters`	a list of matrices or vectors to be used as input to the dense matrix kernel

Examples

```
## Not run:
# Allocate input to the least-squares fit microbenchmark for the
# first matrix size to be tested
microbenchmarks <- GetDenseMatrixDefaultMicrobenchmarks()
kernelParameters <- LsfitAllocator(microbenchmarks[["lsfit"]], 1)
# Execute the microbenchmark
timings <- LsfitMicrobenchmark(microbenchmarks[["lsfit"]], kernelParameters)

## End(Not run)
```

`MatmatAllocator` *Allocates and populates input to the matrix-matrix multiplication dense matrix kernel microbenchmarks*

Description

`MatmatAllocator` allocates and populates the input to the matrix-matrix multiplication dense matrix kernel for the purposes of conducting a single performance trial with the `MatmatMicrobenchmark` function. The matrices or vectors corresponding to the `index` parameter must be allocated, initialized and returned in the `kernelParameters` list.

Usage

```
MatmatAllocator(benchmarkParameters, index)
```

Arguments

`benchmarkParameters`	an object of type `DenseMatrixMicrobenchmark` specifying various parameters needed to generate input for the dense matrix kernel.
`index`	an integer index indicating the dimensions of the matrix or vector data to be generated as input for the dense matrix kernel.

`MatmatMicrobenchmark` *Conducts a single performance trial with the matrix-matrix multiplication dense matrix kernel*

Description

`MatmatMicrobenchmark` conducts a single performance trial of the matrix-matrix multiplication dense matrix kernel for the matrix given in the `kernelParameters` parameter. The function times the single function call `kernelParameters$A %*% kernelParameters$B`.

Usage

```
MatmatMicrobenchmark(benchmarkParameters, kernelParameters)
```

Arguments

benchmarkParameters	an object of type DenseMatrixMicrobenchmark specifying various parameters for microbenchmarking the dense matrix kernel
kernelParameters	a list of matrices or vectors to be used as input to the dense matrix kernel

Examples

```
## Not run:
# Allocate input to the matrix-matrix multiplication microbenchmark for the
# first matrix size to be tested
microbenchmarks <- GetDenseMatrixDefaultMicrobenchmarks()
kernelParameters <- MatmatAllocator(microbenchmarks[["matmat"]], 1)
# Execute the microbenchmark
timings <- MatmatMicrobenchmark(microbenchmarks[["matmat"]], kernelParameters)

## End(Not run)
```

MatvecAllocator *Allocates and populates input to the matrix-vector multiplication dense matrix kernel microbenchmarks*

Description

MatvecAllocator allocates and populates the input to the matrix-vector multiplication dense matrix kernel for the purposes of conducting a single performance trial with the MatvecMicrobenchmark function. The matrices or vectors corresponding to the index parameter must be allocated, initialized and returned in the kernelParameters list.

Usage

```
MatvecAllocator(benchmarkParameters, index)
```

Arguments

benchmarkParameters	an object of type DenseMatrixMicrobenchmark specifying various parameters needed to generate input for the dense matrix kernel.
index	an integer index indicating the dimensions of the matrix or vector data to be generated as input for the dense matrix kernel.

MatvecMicrobenchmark *Conducts a single performance trial with the matrix-vector multiplication dense matrix kernel*

Description

MatvecMicrobenchmark conducts a single performance trial of the matrix-vector multiplication dense matrix kernel for the matrix given in the kernelParameters parameter. The function times the single function call kernelParameters$A %*% kernelParameters$b.

Usage

```
MatvecMicrobenchmark(benchmarkParameters, kernelParameters)
```

Arguments

benchmarkParameters
: an object of type DenseMatrixMicrobenchmark specifying various parameters for microbenchmarking the dense matrix kernel

kernelParameters
: a list of matrices or vectors to be used as input to the dense matrix kernel

Examples

```
## Not run:
# Allocate input to the matrix-vector multiplication microbenchmark for the
# first matrix size to be tested
microbenchmarks <- GetDenseMatrixDefaultMicrobenchmarks()
kernelParameters <- MatvecAllocator(microbenchmarks[["matvec"]], 1)
# Execute the microbenchmark
timings <- MatvecMicrobenchmark(microbenchmarks[["matvec"]], kernelParameters)

## End(Not run)
```

MicrobenchmarkClusteringKernel *Performs microbenchmarking of a clustering for machine learning kernel*

Description

MicrobenchmarkClusteringKernel performs microbenchmarking of a clustering for machine learning kernel for a given data set

Usage

```
MicrobenchmarkClusteringKernel(benchmarkParameters, numberOfThreads,
  resultsDirectory, runIdentifier)
```

Arguments

benchmarkParameters	an object of type ClusteringMicrobenchmark specifying the data set to be read in or generated and the number of performance trials to perform with the data set.
numberOfThreads	the number of threads the microbenchmark is being performed with. The value is for informational purposes only and does not effect the number threads the kernel is executed with.
resultsDirectory	a character string specifying the directory where all of the CSV performance results files will be saved
runIdentifier	a character string specifying the suffix to be appended to the base of the file name of the output CSV format files

Details

This function performs microbenchmarking of a clustering for machine learning kernel for a given data set and a given number of threads. The kernel to be performance tested and other parameters specifying how the kernel is to be benchmarked are given in the input object benchmarkParameters which is an instance of the class ClusteringMicrobenchmark. The performance results are averaged over the number of performance trials and written to a CSV file. The results of the individual performance trials are retained in a data frame that is returned upon completion of the microbenchmark. The kernel can be executed with multiple threads if the kernel supports multithreading. See ClusteringMicrobenchmark for more details on the benchmarking parameters.

Value

a dataframe containing the performance trial times for the given kernel and data set being tested, that is the raw performance data before averaging. The columns of the data frame are the following:

BenchmarkName The name of the microbenchmark

NumberOfFeatures The number of features in each feature vector

NumberOfFeatureVectors The number of features in the data set

NumberOfClusters The number of clusters in the data set

UserTime The amount of time spent in user-mode code within the microbenchmarked code

SystemTime The amount of time spent in the kernel within the process

WallClockTime The total time spent to complete the performance trial

DateStarted The date and time the performance trial was commenced

DateFinished The date and time the performance trial ended

`MicrobenchmarkDenseMatrixKernel` *Performs microbenchmarking of a dense matrix linear algebra kernel*

Description

`MicrobenchmarkDenseMatrixKernel` performs microbenchmarking of a dense matrix linear algebra kernel for several matrix dimensions

Usage

```
MicrobenchmarkDenseMatrixKernel(benchmarkParameters, numberOfThreads,
  resultsDirectory, runIdentifier)
```

Arguments

`benchmarkParameters`	an object of type `DenseMatrixMicrobenchmark` specifying the matrix dimensions of matrices to be tested and the number of performance trials to perform for each matrix dimension.
`numberOfThreads`	the number of threads the microbenchmark is being performed with. The value is for informational purposes only and does not effect the number threads the kernel is executed with.
`resultsDirectory`	a character string specifying the directory where all of the CSV performance results files will be saved
`runIdentifier`	a character string specifying the suffix to be appended to the base of the file name of the output CSV format files

Details

This function performs microbenchmarking of a dense matrix linear algebra kernel for several matrix dimensions and a given number of threads. The kernel to be performance tested, the matrix dimensions to be tested, and other parameters specifying how the kernel is to be benchmarked are given in the input object `benchmarkParameters` which is an instance of the class `DenseMatrixMicrobenchmark`. For each matrix dimension to be tested, the run time performance of the kernel is averaged over multiple performance trials, and the averages are written to a CSV file. The results of the individual performance trials are retained in a data frame that is returned upon completion of the microbenchmark. The kernel can also be executed with multiple threads if the kernel supports multithreading. See `DenseMatrixMicrobenchmark` for more details on the benchmarking parameters.

Value

a dataframe containing the performance trial times for each matrix tested, that is the raw performance data before averaging. The columns of the data frame are the following:

BenchmarkName The name of the microbenchmark

DimensionParameter The dimension parameters the microbenchmark uses to define the matrix dimensions to be tested with

UserTime The amount of time spent in user-mode code within the microbenchmarked code

SystemTime The amount of time spent in the kernel within the process

WallClockTime The total time spent to complete the performance trial

DateStarted The date and time the performance trial was commenced

DateFinished The date and time the performance trial ended

`MicrobenchmarkSparseMatrixKernel`
Performs microbenchmarking of a sparse matrix linear algebra kernel

---

Description

`MicrobenchmarkSparseMatrixKernel` performs microbenchmarking of a sparse matrix linear algebra kernel for several matrix dimensions

Usage

```
MicrobenchmarkSparseMatrixKernel(benchmarkParameters, numberOfThreads,
  resultsDirectory, runIdentifier)
```

Arguments

`benchmarkParameters`
: an object of type `SparseMatrixMicrobenchmark` specifying the matrix dimensions of matrices to be tested and the number of performance trials to perform for each matrix dimension.

`numberOfThreads`
: the number of threads the microbenchmark is being performed with. The value is for informational purposes only and does not effect the number threads the kernel is executed with.

`resultsDirectory`
: a character string specifying the directory where all of the CSV performance results files will be saved

`runIdentifier`
: a character string specifying the suffix to be appended to the base of the file name of the output CSV format files

Details

This function performs microbenchmarking of a sparse matrix linear algebra kernel for several matrix dimensions and a given number of threads. The kernel to be performance tested, the matrix dimensions to be tested, and other parameters specifying how the kernel is to be benchmarked are given in the input object `benchmarkParameters` which is an instance of the class `SparseMatrixMicrobenchmark`. For each matrix dimension to be tested, the run time performance of the kernel is averaged over multiple runs. The kernel can also be executed with multiple threads if the kernel supports multithreading. See `SparseMatrixMicrobenchmark` for more details on the benchmarking parameters.

Value

a dataframe containing the performance trial times for each matrix tested, that is the raw performance data before averaging. The columns of the data frame are the following:

BenchmarkName The name of the microbenchmark

NumberOfRows An integer specifying the expected number of rows in the input sparse matrix

NumberOfColumns An integer specifying the expected number of columns in the input sparse matrix

UserTime The amount of time spent in user-mode code within the microbenchmarked code

SystemTime The amount of time spent in the kernel within the process

WallClockTime The total time spent to complete the performance trial

DateStarted The date and time the performance trial was commenced

DateFinished The date and time the performance trial ended

`PamClusteringMicrobenchmark` *Conducts a single performance trial with the cluster::pam function*

Description

`ClusteringMicrobenchmark` conducts a single performance trial of the cluster::pam function with the data given in the `kernelParameters` parameter.

Usage

```
PamClusteringMicrobenchmark(benchmarkParameters, kernelParameters)
```

Arguments

`benchmarkParameters`
: an object of type `ClusteringMicrobenchmark` specifying various parameters for microbenchmarking the cluster::pam function

`kernelParameters`
: a list of data objects to be used as input to the clustering function

Value

a vector containing the user, system, and elapsed performance timings in that order

Examples

```
## Not run:
# Allocate input to the pam clustering microbenchmark
microbenchmarks <- GetClusteringExampleMicrobenchmarks()
kernelParameters <- ClusteringAllocator(microbenchmarks[["pam_cluster_3_3_1000"]])
# Execute the microbenchmark
timings <- PamClusteringMicrobenchmark(
   microbenchmarks[["pam_cluster_3_3_1000"]], kernelParameters)

## End(Not run)
```

PerformClusteringMicrobenchmarking

Performs microbenchmarking of machine learning functions specified by an input list

---

Description

`PerformClusteringMicrobenchmarking` performs microbenchmarking of machine learning functionality specified by the input list of `ClusteringMicrobenchmark` objects. Objects with the `active` flag set to TRUE indicate that the corresponding microbenchmark will be performed; FALSE indicates that the microbenchmark will be skipped.

Usage

```
PerformClusteringMicrobenchmarking(microbenchmarks, microbenchmarkingFunction,
  numberOfThreads, runIdentifier, resultsDirectory)
```

Arguments

`microbenchmarks`	a list of `ClusteringMicrobenchmark` objects defining the machine learning microbenchmarks to be executed as part of the machine learning benchmark.
`microbenchmarkingFunction`	a function that performs the run time performance trials, computes the summary performance statistics, and writes the performance results to standard out,
`numberOfThreads`	the number of threads the microbenchmarks are intended to be executed with; the value is for display purposes only as the number of threads used is assumed to be controlled through environment variables
`runIdentifier`	a character string specifying the suffix to be appended to the base of the file name of the output CSV format files

`resultsDirectory`
: a character string specifying the directory where all of the CSV performance results files will be saved

Value

a data frame containing the benchmark name, user, system, and elapsed (wall clock) times of each performance trial each microbenchmark

---

PerformSparseMatrixKernelMicrobenchmarking

Performs microbenchmarking of sparse matrix kernels specified by an input list

---

Description

`PerformSparseMatrixKernelMicrobenchmarking` performs microbenchmarking of sparse matrix kernels specified by the input list of `SparseMatrixMicrobenchmark` objects. Objects with the `active` flag set to TRUE indicate that the corresponding microbenchmark will be performed; FALSE indicates that the microbenchmark will be skipped. If the `matrixObjectName` field of an input `SparseMatrixMicrobenchmark` object is set to `NA_character_`, then the sparse matrix is assumed to be dynamically generated by the allocator function specified in the `allocatorFunction` field. If the `matrixObjectName` field is specified, then the sparse matrix object is expected to be found in an `.RData` file with base file name the same as the value of `matrixObjectName`, and located in the either an attached R data package or a directory named `data` in the current working directory. See the the `data` package for more details.

Usage

```
PerformSparseMatrixKernelMicrobenchmarking(microbenchmarks, numberOfThreads,
  runIdentifier, resultsDirectory)
```

Arguments

`microbenchmarks`
: a list of `SparseMatrixMicrobenchmark` objects defining the sparse matrix microbenchmarks to be executed as part of the sparse matrix benchmark.

`numberOfThreads`
: the number of threads the microbenchmarks are intended to be executed with; the value is for display purposes only as the number of threads used is assumed to be controlled through environment variables

`runIdentifier`
: a character string specifying the suffix to be appended to the base of the file name of the output CSV format files

`resultsDirectory`
: a character string specifying the directory where all of the CSV performance results files will be saved

Value

a data frame containing the benchmark name, user, system, and elapsed (wall clock) times of each performance trial for each microbenchmark

See Also

data

PrintClusteringMicrobenchmarkResults

Prints results of a clustering for machine learning microbenchmark

---

Description

PrintClusteringMicrobenchmarkResults prints performance results for a clustering for machine learning microbenchmark to standard output in a format that is easily human readable

Usage

```
PrintClusteringMicrobenchmarkResults(benchmarkName, numberOfThreads,
  numberOfFeatures, numberOfFeatureVectors, numberOfClusters,
  numberOfSuccessfulTrials, trialTimes, averageWallClockTimes,
  standardDeviations)
```

Arguments

benchmarkName	character string specifying the name of the microbenchmark
numberOfThreads	the number of threads all of the performance trials were conducted with
numberOfFeatures	the number of features, i.e. the dimension of the feature vector
numberOfFeatureVectors	the number of feature vectors in the data set
numberOfClusters	the number of clusters in the data set
numberOfSuccessfulTrials	an integer vector specifying the number of performance trials that were successfully performed for each data set
trialTimes	a real matrix with each column containing the run times of all of the successful performance trials associated with a particular data set. The number of valid entries in each column are specified by the entries in the numberOfSuccessfulTrials vector
averageWallClockTimes	a vector of average wall clock times computed for each matrix tested during the performance trials
standardDeviations	a vector of standard deviations of the wall clock times obtained for each matrix tested during the performance trials

Details

This function prints the performance results obtained by a clustering for machine learning microbenchmark. Summary run time performance statistics for each clustering data set tested are computed and printed. The summary statistics include the minimum, maximum, average, and standard deviation of the wall clock times obtained by the performance trials with respect to each data tested.

`PrintDenseMatrixMicrobenchmarkResults`
Prints results of a dense matrix microbenchmark

Description

`PrintDenseMatrixMicrobenchmarkResults` prints performance results for a dense matrix microbenchmark to standard output in a format that is easily human readable

Usage

```
PrintDenseMatrixMicrobenchmarkResults(benchmarkName, numberOfThreads,
  dimensionParameters, numberOfSuccessfulTrials, trialTimes,
  averageWallClockTimes, standardDeviations)
```

Arguments

`benchmarkName`	character string specifying the name of the microbenchmark
`numberOfThreads`	the number of threads all of the performance trials were conducted with
`dimensionParameters`	an integer vector specifying the dimension parameters the microbenchmark uses to define the matrix dimensions to be tested with; length is assumed to be greater than zero
`numberOfSuccessfulTrials`	an integer vector specifying the number of performance trials that were successfully performed for each matrix tested
`trialTimes`	a real matrix with each column containing the run times of all of the successful performance trials associated with a particular matrix. The number of valid entries in each column are specified by the entries in the `numberOfSuccessfulTrials` vector
`averageWallClockTimes`	a vector of average wall clock times computed for each matrix tested during the performance trials
`standardDeviations`	a vector of standard deviations of the wall clock times obtained for each matrix tested during the performance trials

Details

This function prints the performance results obtained by a dense matrix microbenchmark for matrices of various dimensions. The results include summary statistics for each matrix tested. The summary statistics include the minimum, maximum, average, and standard deviation of the wall clock times obtained by the performance trials with respect to each matrix tested.

PrintSparseMatrixMicrobenchmarkResults

Prints results of a sparse matrix microbenchmark

Description

`PrintSparseMatrixMicrobenchmarkResults` prints performance results for a sparse matrix microbenchmark to standard output in a format that is easily human readable

Usage

```
PrintSparseMatrixMicrobenchmarkResults(benchmarkName, numberOfThreads,
  numberOfRows, numberOfColumns, numberOfNonzeros, numberOfSuccessfulTrials,
  trialTimes, averageWallClockTimes, standardDeviations)
```

Arguments

`benchmarkName`	character string specifying the name of the microbenchmark
`numberOfThreads`	the number of threads all of the performance trials were conducted with
`numberOfRows`	the number of expected rows in the matrix; assumed to be greater than zero
`numberOfColumns`	the number of expected columns in the matrix; assumed to be greater than zero
`numberOfNonzeros`	the number of non-zero elements in the matrix
`numberOfSuccessfulTrials`	an integer vector specifying the number of performance trials that were successfully performed for each matrix tested
`trialTimes`	a real matrix with each column containing the run times of all of the successful performance trials associated with a particular matrix. The number of valid entries in each column are specified by the entries in the `numberOfSuccessfulTrials` vector
`averageWallClockTimes`	a vector of average wall clock times computed for each matrix tested during the performance trials
`standardDeviations`	a vector of standard deviations of the wall clock times obtained for each matrix tested during the performance trials

Details

This function prints the run time performance results obtained by a sparse matrix microbenchmark for matrices of various dimensions. The summary statistics include the minimum, maximum, average, and standard deviation of the wall clock times obtained by the performance trials with respect to each matrix tested.

`QrAllocator` *Allocates and populates input to the QR factorization dense matrix kernel microbenchmarks*

Description

`QrAllocator` allocates and populates the input to the QR factorization dense matrix kernel for the purposes of conducting a single performance trial with the `QrMicrobenchmark` function. The matrices or vectors corresponding to the `index` parameter must be allocated, initialized and returned in the `kernelParameters` list.

Usage

```
QrAllocator(benchmarkParameters, index)
```

Arguments

`benchmarkParameters`
: an object of type `DenseMatrixMicrobenchmark` specifying various parameters needed to generate input for the dense matrix kernel.

`index`
: an integer index indicating the dimensions of the matrix or vector data to be generated as input for the dense matrix kernel.

`QrMicrobenchmark` *Conducts a single performance trial with the QR factorization dense matrix kernel*

Description

`QrMicrobenchmark` conducts a single performance trial of the QR factorization dense matrix kernel for the matrix given in the `kernelParameters` parameter. The function times the single function call `qr(kernelParameters$A, LAPACK=TRUE)`.

Usage

```
QrMicrobenchmark(benchmarkParameters, kernelParameters)
```

Arguments

benchmarkParameters
: an object of type DenseMatrixMicrobenchmark specifying various parameters for microbenchmarking the dense matrix kernel

kernelParameters
: a list of matrices or vectors to be used as input to the dense matrix kernel

Examples

```
## Not run:
# Allocate input to the QR decomposition microbenchmark for the
# first matrix size to be tested
microbenchmarks <- GetDenseMatrixDefaultMicrobenchmarks()
kernelParameters <- QrAllocator(microbenchmarks[["qr"]], 1)
# Execute the microbenchmark
timings <- QrMicrobenchmark(microbenchmarks[["qr"]], kernelParameters)

## End(Not run)
```

RHPCBenchmark — *RHPCBenchmark: A package for performance testing intrinsic R functionality and established packages relevant to high-performance computing*

Description

The benchmarks are divided into three categories: dense matrix linear algebra kernels, sparse matrix linear algebra kernels, and machine learning functionality. All of the dense linear algebra kernels are implemented around BLAS or LAPACK interfaces. The sparse linear algebra kernels are members of the R Matrix library. The machine learning benchmarks currently only cover variants of K-means functionality for clustering using the `cluster` package. The dense matrix linear algebra kernels, sparse matrix linear algebra kernels, and machine learning functions that are benchmarked are all part of the R interpreter's intrinsic functionality or packages included the with the R programming environment standard distributions from CRAN.

Details

For fast performance of the dense matrix kernels, it is crucial to link the R programming environment with optimized BLAS and LAPACK libraries. It is also important to have substantial amounts of memory (16GB minimum) to run most of the microbenchmarks. If any of the microbenchmarks fails to run in a timely manner or fails due to memory constraints, the matrix sizes and number of performance trials per matrix can be adjusted. See the documentation for top-level benchmark functions and the microbenchmark definition classes listed below for information on how to configure the individual microbenchmarks.

Top-level benchmark functions

RunDenseMatrixBenchmark Executes the dense matrix microbenchmarks

RunSparseMatrixBenchmark Executes the sparse matrix microbenchmarks

RunMachineLearningBenchmark Executes the machine learning microbenchmarks

Microbenchmark definition classes

DenseMatrixMicrobenchmark Specifies a dense matrix microbenchmark

SparseMatrixMicrobenchmark Specifies a sparse matrix microbenchmark

ClusteringMicrobenchmark Specifies a clustering for machine learning microbenchmark

---

RunDenseMatrixBenchmark

Runs all of the dense matrix microbenchmarks

Description

RunDenseMatrixBenchmark runs all of the microbenchmarks for performance testing the dense matrix linear algebra kernels

Usage

```
RunDenseMatrixBenchmark(runIdentifier, resultsDirectory,
  microbenchmarks = GetDenseMatrixDefaultMicrobenchmarks())
```

Arguments

runIdentifier	a character string specifying the suffix to be appended to the base of the file name of the output CSV format files
resultsDirectory	a character string specifying the directory where all of the CSV performance results files will be saved
microbenchmarks	a list of DenseMatrixMicrobenchmark objects defining the microbenchmarks to execute as part of the dense matrix benchmark. Default values are provided by the function GetDenseMatrixDefaultMicrobenchmarks.

Details

This function runs all of the dense matrix microbenchmarks defined in the microbenchmarks input list for which the active field is set to TRUE. For each microbenchmark, it attempts to create a separate output file in CSV format containing the performance results for each matrix tested by the microbenchmark. The names of the output files follow the format benchmarkName_runIdentifier.csv, where benchmarkName is specified in the DenseMatrixMicrobenchmark object of each microbenchmark, and runIdentifier is an input parameter to this function. If the file already exists, the results will be appended to the existing file. The microbenchmarks input list contains instances of

the DenseMatrixMicrobenchmark class defining each microbenchmark. The default microbenchmarks are generated by the function GetDenseMatrixDefaultMicrobenchmarks. If the linear algebra kernels are multithreaded, by linking to multithreaded BLAS or LAPACK libraries for example, then the number of threads must be retrievable from an environment variable which is set before execution of the R programming environment. The name of the environment variable specifying the number of threads must be provided in the R HPC benchmark environment variable R_BENCH_NUM_THREADS_VARIABLE. This function will retrieve the number of threads through R_BENCH_NUM_THREADS_VARIABLE so that the number of threads can be printed to the results files and recorded in data frames for reporting purposes. This function utilizes the number of threads only for reporting purposes and is not used by the benchmark to effect the actual number of threads utilized by the kernels, as that is assumed to be controlled by the numerical library. An error exception will be thrown if the environment variable R_BENCH_NUM_THREADS_VARIABLE and the variable it is set to are not both set.

Value

a data frame containing the benchmark name, user, system, and elapsed (wall clock) times of each performance trial for each microbenchmark

See Also

GetDenseMatrixDefaultMicrobenchmarks GetDenseMatrixExampleMicrobenchmarks

Examples

```
## Not run:
# Set needed environment variables for multithreading.  Only single threading
# is used in the example.
#
# Note: The environment variables are usually set by the user before starting
#       the R programming environment; they are set here only to facilitate
#       a working example.  See the section on multithreading in the vignette
#       for further details.
Sys.setenv(R_BENCH_NUM_THREADS_VARIABLE="MKL_NUM_THREADS")
Sys.setenv(MKL_NUM_THREADS="1")
#
# Generate example microbechmarks that can be run in a few minutes; see
# the vignette for more involved examples.  The Cholesky factorization and
# matrix crossproduct microbenchmarks are performed in the example code
# below.
#
# Note: These microbenchmarks are different than the microbenchmarks
#       generated by \code{\link{GetDenseMatrixDefaultMicrobenchmarks}}.
#       They are chosen for their short run times and suitability for
#       example code.
exampleMicrobenchmarks <- GetDenseMatrixExampleMicrobenchmarks()
# Set the output directory of the CSV summary results files
resultsDirectory <- "./DenseMatrixExampleOutput"
# Create the output directory
dir.create(resultsDirectory)
# Set an appropriate run identifier
```

```
runIdentifier <- "example"
resultsFrame <- RunDenseMatrixBenchmark(runIdentifier, resultsDirectory,
   microbenchmarks=exampleMicrobenchmarks)

# This example runs all but the matrix transpose microbenchmarks.
exampleMicrobenchmarks[["transpose"]]$active <- FALSE
# Set an appropriate run identifier
runIdentifier <- "no_transpose"
exTransposeResultsFrame <- RunDenseMatrixBenchmark(runIdentifier,
  resultsDirectory, microbenchmarks=exampleMicrobenchmarks)

# This example runs only the matrix-matrix multiplication microbenchmark,
# and it adds a larger matrix to test.
matMatMicrobenchmark <- list()
matMatMicrobenchmark[["matmat"]] <- GetDenseMatrixExampleMicrobenchmarks()[["matmat"]]
matMatMicrobenchmark[["matmat"]]$dimensionParameters <- as.integer(c(1000, 2000))
matMatMicrobenchmark[["matmat"]]$numberOfTrials <- as.integer(c(3, 3))
matMatMicrobenchmark[["matmat"]]$numberOfWarmupTrials <- as.integer(c(1, 1))
# Set an appropriate run identifier
runIdentifier <- "matmat"
matMatResults <- RunDenseMatrixBenchmark(runIdentifier, resultsDirectory,
   microbenchmarks=matMatMicrobenchmark)

## End(Not run)
```

`RunMachineLearningBenchmark` *Runs all of the machine learning microbenchmarks*

Description

`RunMachineLearningBenchmark` runs all of the microbenchmarks for performance testing machine learning functionality

Usage

```
RunMachineLearningBenchmark(runIdentifier, resultsDirectory,
  clusteringMicrobenchmarks = GetClusteringDefaultMicrobenchmarks())
```

Arguments

`runIdentifier`	a character string specifying the suffix to be appended to the base of the file name of the output CSV format files
`resultsDirectory`	a character string specifying the directory where all of the CSV performance results files will be saved

clusteringMicrobenchmarks
: a list of ClusteringMicrobenchmark objects defining the clustering microbenchmarks to execute as part of the machine learning benchmark. Default values are provided by the function GetClusteringDefaultMicrobenchmarks.

Details

This function runs the machine learning microbenchmarks, which are divided into four categories supported by this benchmark, defined in the clusteringMicrobenchmarks input list. For each microbenchmark, it attempts to create a separate output file in CSV format containing the performance results for data set and function tested by the microbenchmark. The names of the output files follow the format benchmarkName_runIdentifier.csv, where benchmarkName is specified in the ClusteringMicrobenchmark object of each microbenchmark and runIdentifier is an input parameter to this function. If the file already exists, the results will be appended to the existing file. Each input list contains instances of the ClusteringMicrobenchmark class defining each microbenchmark. Each microbenchmark object with the active field set to TRUE will be executed. The lists of default microbenchmarks are generated by the function GetClusteringDefaultMicrobenchmarks. Each ClusteringMicrobenchmark specifies an R data file which contains the data object needed by the microbenchmark. The needed R data files should either be given in an attached R package or given in the data subdirectory of the current working directory, and they should have the extension .RData. If the linear algebra kernels are multithreaded, by linking to multithreaded BLAS or LAPACK libraries for example, then the number of threads must be retrievable from an environment variable which is set before execution of the R programming environment. The name of the environment variable specifying the number of threads must be provided in the R HPC benchmark environment variable R_BENCH_NUM_THREADS_VARIABLE. This function will retrieve the number of threads through R_BENCH_NUM_THREADS_VARIABLE so that the number of threads can be printed to the results files and recorded in data frames for reporting purposes. This function utilizes the number of threads only for reporting purposes and is not used by the benchmark to effect the actual number of threads utilized by the kernels, as that is assumed to be controlled by the numerical library. An error exception will be thrown if the environment variable R_BENCH_NUM_THREADS_VARIABLE and the variable it is set to are not both set.

Value

a data frame containing the user, system, and elapsed (wall clock) time of times of each performance trial

See Also

GetClusteringDefaultMicrobenchmarks GetClusteringExampleMicrobenchmarks

Examples

```
## Not run:
# Set needed environment variables for multithreading.  Only single threading
# is used in the example.
#
# Note: The environment variables are usually set by the user before starting
#       the R programming environment; they are set here only to facilitate
#       a working example.  See the section on multithreading in the vignette
```

```
#       for further details.
Sys.setenv(R_BENCH_NUM_THREADS_VARIABLE="MKL_NUM_THREADS")
Sys.setenv(MKL_NUM_THREADS="1")
#
# Generate example microbechmarks that can be run in a few minutes; see
# the vignette for more involved examples. Clustering microbenchmarks
# are defined in the examples.
#
# Note: These microbenchmarks are different than the microbenchmarks
#       generated by \code{\link{GetDenseMatrixDefaultMicrobenchmarks}}.
#       They are chosen for their short run times and suitability for
#       example code.
exampleMicrobenchmarks <- GetClusteringExampleMicrobenchmarks()
# Set the output directory of the CSV summary results files
resultsDirectory <- "./MachineLearningExampleOutput"
# Create the output directory
dir.create(resultsDirectory)
# Set an appropriate run identifier
runIdentifier <- "example"
resultsFrame <- RunMachineLearningBenchmark(runIdentifier, resultsDirectory,
   clusteringMicrobenchmarks=exampleMicrobenchmarks)

# Create a new clustering microbenchmark that tests the clara method from
# the cluster package using a data set with 16 features, 8 clusters, and
# 1000 normally distributed feature vectors per cluster.
claraMicrobenchmark <- list()
claraMicrobenchmark[["clara_cluster_16_8_1000"]] <- methods::new(
   "ClusteringMicrobenchmark",
   active = TRUE,
   benchmarkName = "clara_cluster_16_8_1000",
   benchmarkDescription = "Example of new clara microbenchmark",
   dataObjectName = NA_character_,
   numberOfFeatures = as.integer(16),
   numberOfClusters = as.integer(8),
   numberOfFeatureVectorsPerCluster = as.integer(1000),
   numberOfTrials = as.integer(3),
   numberOfWarmupTrials = as.integer(1),
   allocatorFunction = ClusteringAllocator,
   benchmarkFunction = ClaraClusteringMicrobenchmark
)

# Set an appropriate run identifier
runIdentifier <- "clara_new"
# Run the clara microbenchmark
claraResults <- RunMachineLearningBenchmark(runIdentifier, resultsDirectory,
   clusteringMicrobenchmarks=claraMicrobenchmark)

## End(Not run)
```

RunSparseMatrixBenchmark *Runs all of the sparse matrix microbenchmarks*

Description

RunSparseMatrixBenchmark runs all of the microbenchmarks for performance testing the sparse matrix linear algebra kernels

Usage

```
RunSparseMatrixBenchmark(runIdentifier, resultsDirectory,
  matrixVectorMicrobenchmarks = GetSparseMatrixVectorDefaultMicrobenchmarks(),
  choleskyMicrobenchmarks = GetSparseCholeskyDefaultMicrobenchmarks(),
  luMicrobenchmarks = GetSparseLuDefaultMicrobenchmarks(),
  qrMicrobenchmarks = GetSparseQrDefaultMicrobenchmarks())
```

Arguments

runIdentifier	a character string specifying the suffix to be appended to the base of the file name of the output CSV format files
resultsDirectory	a character string specifying the directory where all of the CSV performance results files will be saved
matrixVectorMicrobenchmarks	a list of SparseMatrixMicrobenchmark objects defining the matrix-vector multiplication microbenchmarks to execute as part of the sparse matrix benchmark. Default values are provided by the function GetSparseMatrixVectorDefaultMicrobenchmarks. If the value is NULL, then all of the matrix-vector multiplication microbenchmarks will be skipped.
choleskyMicrobenchmarks	a list of SparseMatrixMicrobenchmark objects defining the Cholesky factorization microbenchmarks to execute as part of the sparse matrix benchmark. Default values are provided by the function GetSparseCholeskyDefaultMicrobenchmarks. If the value is NULL, then all of the Cholesky factorization microbenchmarks will be skipped.
luMicrobenchmarks	a list of SparseMatrixMicrobenchmark objects defining the LU factorization microbenchmarks to execute as part of the sparse matrix benchmark. Default values are provided by the function GetSparseLuDefaultMicrobenchmarks. If the value is NULL, then all of the LU factorization microbenchmarks will be skipped.
qrMicrobenchmarks	a list of SparseMatrixMicrobenchmark objects defining the QR factorization microbenchmarks to execute as part of the sparse matrix benchmark. Default values are provided by the function GetSparseQrDefaultMicrobenchmarks. If the value is NULL, then all of the QR factorization microbenchmarks will be skipped.

Details

This function runs the sparse matrix microbenchmarks, which are divided into four categories supported by this benchmark, defined in the `matrixVectorMicrobenchmarks`, `choleskyMicrobenchmarks`, `luMicrobenchmarks`, and `qrMicrobenchmarks` input lists For each microbenchmark, it attempts to create a separate output file in CSV format containing the performance results for each matrix tested by the microbenchmark. The names of the output files follow the format `benchmarkName_runIdentifier.csv`, where `benchmarkName` is specified in the `SparseMatrixMicrobenchmark` object of each microbenchmark and `runIdentifier` is an input parameter to this function. If the file, already exists, the results will be appended to the existing file. Each input lists contains instances of the `SparseMatrixMicrobenchmark` class defining each microbenchmark. Each microbenchmark object with the `active` field set to TRUE will be executed. The lists of default microbenchmarks are generated by the functions `GetSparseMatrixVectorDefaultMicrobenchmarks`, `GetSparseCholeskyDefaultMicrobenchmarks`, `GetSparseLuDefaultMicrobenchmarks`, and `GetSparseQrDefaultMicrobenchmarks`. Each `SparseMatrixMicrobenchma` specifies an R data file which contains the sparse matrix object needed by the microbenchmark. The needed R data files should either be given in an attached R package or given in the `data` subdirectory of the current working directory, and they should have the extension `.RData`. If the linear algebra kernels are multithreaded, by linking to multithreaded BLAS or LAPACK libraries for example, then the number of threads must be retrievable from an environment variable which is set before execution of the R programming environment. The name of the environment variable specifying the number of threads must be provided in the R HPC benchmark environment variable R_BENCH_NUM_THREADS_VARIABLE. This function will retrieve the number of threads through R_BENCH_NUM_THREADS_VARIABLE so that the number of threads can be printed to the results files and recorded in data frames for reporting purposes. This function utilizes the number of threads only for reporting purposes and is not used by the benchmark to effect the actual number of threads utilized by the kernels, as that is assumed to be controlled by the numerical library. An error exception will be thrown if the environment variable R_BENCH_NUM_THREADS_VARIABLE and the variable it is set to are not both set.

Value

a data frame containing the benchmark name, user, system, and elapsed (wall clock) times of each performance trial for each microbenchmark

See Also

`GetSparseMatrixVectorDefaultMicrobenchmarks` `GetSparseCholeskyDefaultMicrobenchmarks` `GetSparseLuDefaultMicrobenchmarks` `GetSparseQrDefaultMicrobenchmarks` `GetSparseMatrixVectorExampleMicr` `GetSparseCholeskyExampleMicrobenchmarks`

Examples

```
## Not run:
# Set needed environment variables for multithreading.  Only single threading
# is used in the example.
#
# Note: The environment variables are usually set by the user before starting
#       the R programming environment; they are set here only to facilitate
#       a working example.  See the section on multithreading in the vignette
#       for further details.
```

```
Sys.setenv(R_BENCH_NUM_THREADS_VARIABLE="MKL_NUM_THREADS")
Sys.setenv(MKL_NUM_THREADS="1")
#
# Generate example microbenchmarks that can be run in a few minutes; see
# the vignette for more involved examples.  The matvec_laplacian7pt_100
# and cholesky_ct20stif microbenchmarks are defined in the examples.
#
# Note: The example microbenchmarks are different than the microbenchmarks
#       generated by
#       \code{\link{GetSparseMatrixVectorDefaultMicrobenchmarks}},
#       \code{\link{GetSparseCholeskyDefaultMicrobenchmarks}},
#       \code{\link{GetSparseLuDefaultMicrobenchmarks}}, and
#       \code{\link{GetSparseQrDefaultMicrobenchmarks}};
#       they were chosen for their short run times and suitability for
#       example code.
exampleMatrixVectorMicrobenchmarks <- GetSparseMatrixVectorExampleMicrobenchmarks()
exampleCholeskyMicrobenchmarks <- GetSparseCholeskyExampleMicrobenchmarks()
# Set the output directory of the CSV summary results files
resultsDirectory <- "./SparseMatrixExampleOutput"
# Create the output directory
dir.create(resultsDirectory)
# Set an appropriate run identifier
runIdentifier <- "example"
# Run only the matrix-vector and Cholesky factorization microbenchmarks, as
# the others take a long time
resultsFrame <- RunSparseMatrixBenchmark(runIdentifier, resultsDirectory,
   matrixVectorMicrobenchmarks=exampleMatrixVectorMicrobenchmarks,
   choleskyMicrobenchmarks=exampleCholeskyMicrobenchmarks,
   luMicrobenchmarks=NULL,
   qrMicrobenchmarks=NULL)

# This example runs only the Cholesky factorization microbenchmarks.
runIdentifier <- "choleksy_only"
# Run only the sparse Choleksy factorization microbenchmarks
choleskyResults <- RunSparseMatrixBenchmark(runIdentifier, resultsDirectory,
   matrixVectorMicrobenchmarks=NULL, luMicrobenchmarks=NULL,
   qrMicrobenchmarks=NULL)


## End(Not run)
```

`SolveAllocator` *Allocates and populates input to the dense matrix kernel microbenchmark for computing the solution to a system of linear equations with multiple right-hand sides*

Description

`SolveAllocator` allocates and populates the input to the solve kernel for the purposes of conducting a single performance trial with the `QrMicrobenchmark` function. The matrices or vectors corre-

sponding to the index parameter must be allocated, initialized and returned in the kernelParameters list.

Usage

```
SolveAllocator(benchmarkParameters, index)
```

Arguments

benchmarkParameters	an object of type DenseMatrixMicrobenchmark specifying various parameters needed to generate input for the dense matrix kernel.
index	an integer index indicating the dimensions of the matrix or vector data to be generated as input for the dense matrix kernel.

SolveMicrobenchmark *Conducts a single performance trial with the dense matrix kernel for computing the solution to a system of linear equations with multiple right-hand sides*

Description

SolveMicrobenchmark conducts a single performance trial of the solve dense matrix kernel for the matrix given in the kernelParameters parameter. The function times the single function call solve(kernelParameters$A, kernelParameters$B, LAPACK=TRUE).

Usage

```
SolveMicrobenchmark(benchmarkParameters, kernelParameters)
```

Arguments

benchmarkParameters	an object of type DenseMatrixMicrobenchmark specifying various parameters for microbenchmarking the dense matrix kernel
kernelParameters	a list of matrices or vectors to be used as input to the dense matrix kernel

Examples

```
## Not run:
# Allocate input to the linear solve microbenchmark for the
# first matrix size to be tested
microbenchmarks <- GetDenseMatrixDefaultMicrobenchmarks()
kernelParameters <- SolveAllocator(microbenchmarks[["solve"]], 1)
# Execute the microbenchmark
timings <- SolveMicrobenchmark(microbenchmarks[["solve"]], kernelParameters)

## End(Not run)
```

SparseCholeskyAllocator

Allocates and initializes input to the Cholesky factorization sparse matrix kernel microbenchmarks

Description

SparseCholeskyAllocator allocates and initializes the sparse matrix that is input to the sparse matrix kernel for the purposes of conducting a single performance trial with the SparseCholeskyMicrobenchmark function. The matrix is populated and returned in the kernelParameters list.

Usage

```
SparseCholeskyAllocator(benchmarkParameters, index)
```

Arguments

benchmarkParameters	an object of type SparseMatrixMicrobenchmark specifying various parameters needed to generate input for the sparse matrix kernel.
index	an integer index indicating the dimensions of the matrix or vector data to be generated as input for the sparse matrix kernel.

SparseCholeskyMicrobenchmark

Conducts a single performance trial with the Cholesky factorization sparse matrix kernel

Description

SparseMatrixVectorMicrobenchmark conducts a single performance trial of the Cholesky factorization sparse matrix kernel for the matrix given in the kernelParameters parameter. The function times the single function call kernelParameters$A %*% kernelParameters$b.

Usage

```
SparseCholeskyMicrobenchmark(benchmarkParameters, kernelParameters)
```

Arguments

benchmarkParameters	an object of type SparseMatrixMicrobenchmark specifying various parameters for microbenchmarking the sparse matrix kernel
kernelParameters	a list of matrices or vectors to be used as input to the sparse matrix kernel

Value

a vector containing the user, system, and elapsed performance timings in that order

Examples

```
## Not run:
# Allocate input to the Cholesky factorization microbenchmark for the
# ct20stif matrix
microbenchmarks <- GetSparseCholeskyDefaultMicrobenchmarks()
kernelParameters <- SparseCholeskyAllocator(microbenchmarks[["cholesky_ct20stif"]], 1)
# Execute the microbenchmark
timings <- SparseCholeskyMicrobenchmark(
   microbenchmarks[["choleksy_ct20stif"]], kernelParameters)

## End(Not run)
```

SparseLuAllocator *Allocates and initializes input to the LU factorization sparse matrix kernel microbenchmarks*

Description

`SparseLuAllocator` allocates and initializes the sparse matrix that is input to the sparse matrix kernel for the purposes of conducting a single performance trial with the `SparseCholeskyMicrobenchmark` function. The matrix is populated and returned in the `kernelParameters` list.

Usage

```
SparseLuAllocator(benchmarkParameters, index)
```

Arguments

`benchmarkParameters`	an object of type `SparseMatrixMicrobenchmark` specifying various parameters needed to generate input for the sparse matrix kernel.
`index`	an integer index indicating the dimensions of the matrix or vector data to be generated as input for the sparse matrix kernel.

SparseLuMicrobenchmark *Conducts a single performance trial with the LU factorization sparse matrix kernel*

Description

SparseMatrixVectorMicrobenchmark conducts a single performance trial of the Cholesky factorization sparse matrix kernel for the matrix given in the kernelParameters parameter. The function times the single function call kernelParameters$A %*% kernelParameters$b.

Usage

```
SparseLuMicrobenchmark(benchmarkParameters, kernelParameters)
```

Arguments

benchmarkParameters
: an object of type SparseMatrixMicrobenchmark specifying various parameters for microbenchmarking the sparse matrix kernel

kernelParameters
: a list of matrices or vectors to be used as input to the sparse matrix kernel

Examples

```
## Not run:
# Allocate input to the LU factorization microbenchmark for the
# circuit5M_dc matrix
microbenchmarks <- GetSparseLuDefaultMicrobenchmarks()
kernelParameters <- SparseLuAllocator(microbenchmarks[["lu_circuit5M_dc"]], 1)
# Execute the microbenchmark
timings <- SparseLuMicrobenchmark(
   microbenchmarks[["lu_circuit5M_dc"]], kernelParameters)

## End(Not run)
```

SparseMatrixMicrobenchmark *This class specifies a sparse matrix microbenchmark.*

Description

This class specifies a sparse matrix microbenchmark.

Fields

`active` a logical indicating whether the microbenchmark is to be executed (TRUE) or not (FALSE).

`benchmarkName` a character string that is the name of the microbenchmark.

`benchmarkDescription` a character string describing the microbenchmark.

`matrixObjectName` a character string specifying the name of the sparse matrix object that is input to the benchmark; the object must be stored in the R data file with name `matrixObjectName`.RData Setting the field to NA_character_ indicates that the test data will be generated dynamically by the function given in the `allocatorFunction` field instead of read from a data file.

`numberOfRows` an integer specifying the expected number of rows in the input sparse matrix.

`numberOfColumns` an integer specifying the expected number of columns in the input sparse matrix.

`numberOfNonzeros` an integer specifying the expected number of nonzeros in the input sparse matrix.

`numberOfTrials` an integer vector specifying the number of performance trials conducted for each matrix to be tested.

`numberOfWarmupTrials` an integer vector specifying the number of warmup trials to be performed for each matrix to be tested.

`allocatorFunction` the function that allocates and initializes input to the benchmark function. The function takes a `SparseMatrixMicrobenchmark` object and an integer index indicating which matrix parameter from `numberOfRows`, `numberOfColumns`, and `numberOfNonzeros` should be used to generate the matrix.

`benchmarkFunction` the benchmark function which executes the functionality to be timed. The function takes a `SparseMatrixMicrobenchmark` and a list of kernel parameters returned by the allocator function.

`SparseMatrixVectorAllocator`

Allocates and initializes input to the matrix-vector multiplication sparse matrix kernel microbenchmarks

---

Description

`SparseMatrixVectorAllocator` allocates and initializes the sparse matrix and vector that are inputs to the sparse matrix kernel for the purposes of conducting a single performance trial with the `SparseMatrixVectorMicrobenchmark` function. The matrix and vector are populated and returned in the `kernelParameters` list.

Usage

```
SparseMatrixVectorAllocator(benchmarkParameters, index)
```

Arguments

benchmarkParameters
: an object of type SparseMatrixMicrobenchmark specifying various parameters needed to generate input for the sparse matrix kernel.

index
: an integer index indicating the dimensions of the matrix or vector data to be generated as input for the sparse matrix kernel.

Value

a list containing the matrices or vectors to be input for the sparse matrix kernel for which a single performance trial is to be conducted.

SparseMatrixVectorMicrobenchmark

Conducts a single performance trial with the matrix-vector multiplication sparse matrix kernel

---

Description

SparseMatrixVectorMicrobenchmark conducts a single performance trial of the matrix-vector multiplication sparse matrix kernel for the matrix given in the kernelParameters parameter. The function times the single function call kernelParameters$A %*% kernelParameters$b.

Usage

```
SparseMatrixVectorMicrobenchmark(benchmarkParameters, kernelParameters)
```

Arguments

benchmarkParameters
: an object of type SparseMatrixMicrobenchmark specifying various parameters for microbenchmarking the sparse matrix kernel

kernelParameters
: a list of matrices or vectors to be used as input to the sparse matrix kernel

Value

a vector containing the user, system, and elapsed performance timings in that order

Examples

```
## Not run:
# Allocate input to the matrix-vector microbenchmark for the first Laplacian
# matrix
microbenchmarks <- GetSparseMatrixVectorDefaultMicrobenchmarks()
kernelParameters <- SparseMatrixVectorAllocator(
   microbenchmarks[["matvec_laplacian7pt_100"]], 1)
```

```
# Execute the microbenchmark
timings <- SparseMatrixVectorMicrobenchmark(
   microbenchmarks[["matvec_laplacian7pt_100"]], kernelParameters)

## End(Not run)
```

`SparseQrAllocator` *Allocates and initializes input to the QR factorization sparse matrix kernel microbenchmarks*

Description

`SparseQrAllocator` allocates and initializes the sparse matrix that is input to the sparse matrix kernel for the purposes of conducting a single performance trial with the `SparseQrMicrobenchmark` function. The matrix is populated and returned in the `kernelParameters` list.

Usage

```
SparseQrAllocator(benchmarkParameters, index)
```

Arguments

`benchmarkParameters`	an object of type `SparseMatrixMicrobenchmark` specifying various parameters needed to generate input for the sparse matrix kernel.
`index`	an integer index indicating the dimensions of the matrix or vector data to be generated as input for the sparse matrix kernel.

`SparseQrMicrobenchmark` *Conducts a single performance trial with the QR factorization sparse matrix kernel*

Description

`SparseQrVectorMicrobenchmark` conducts a single performance trial of the QR factorization sparse matrix kernel for the matrix given in the `kernelParameters` parameter. The function times the single function call `qr(kernelParameters$A)`.

Usage

```
SparseQrMicrobenchmark(benchmarkParameters, kernelParameters)
```

Arguments

benchmarkParameters
: an object of type SparseMatrixMicrobenchmark specifying various parameters for microbenchmarking the sparse matrix kernel

kernelParameters
: a list of matrices or vectors to be used as input to the sparse matrix kernel

Examples

```
## Not run:
# Allocate input to the QR factorization microbenchmark for the
# Maragal_6 matrix
microbenchmarks <- GetSparseQrDefaultMicrobenchmarks()
kernelParameters <- SparseQrAllocator(microbenchmarks[["qr_Maragal_6"]], 1)
# Execute the microbenchmark
timings <- SparseQrMicrobenchmark(
   microbenchmarks[["qr_Maragal_6"]], kernelParameters)

## End(Not run)
```

SvdAllocator — *Allocates and populates input to the singular value decomposition (SVD) dense matrix kernel microbenchmarks*

Description

SvdAllocator allocates and populates the input to the SVD dense matrix kernel for the purposes of conducting a single performance trial with the SvdMicrobenchmark function. The matrices or vectors corresponding to the index parameter must be allocated, initialized and returned in the kernelParameters list.

Usage

```
SvdAllocator(benchmarkParameters, index)
```

Arguments

benchmarkParameters
: an object of type DenseMatrixMicrobenchmark specifying various parameters needed to generate input for the dense matrix kernel.

index
: an integer index indicating the dimensions of the matrix or vector data to be generated as input for the dense matrix kernel.

SvdMicrobenchmark *Conducts a single performance trial with the singular value decomposition (SVD) dense matrix kernel*

Description

SvdMicrobenchmark conducts a single performance trial of the SVD dense matrix kernel for the matrix given in the kernelParameters parameter. The function times the single function call svd(kernelParameters$A).

Usage

```
SvdMicrobenchmark(benchmarkParameters, kernelParameters)
```

Arguments

benchmarkParameters
: an object of type DenseMatrixMicrobenchmark specifying various parameters for microbenchmarking the dense matrix kernel

kernelParameters
: a list of matrices or vectors to be used as input to the dense matrix kernel

Examples

```
## Not run:
# Allocate input to the singular value decomposition microbenchmark for the
# first matrix size to be tested
microbenchmarks <- GetDenseMatrixDefaultMicrobenchmarks()
kernelParameters <- SvdAllocator(microbenchmarks[["svd"]], 1)
# Execute the microbenchmark
timings <- SvdMicrobenchmark(microbenchmarks[["svd"]], kernelParameters)

## End(Not run)
```

TransposeAllocator *Allocates and populates input to the matrix transpose dense matrix kernel microbenchmarks*

Description

TransposeAllocator allocates and populates the input to the transpose dense matrix kernel for the purposes of conducting a single performance trial with the TransposeMicrobenchmark function. The matrices or vectors corresponding to the index parameter must be allocated, initialized and returned in the kernelParameters list.

Usage

```
TransposeAllocator(benchmarkParameters, index)
```

Arguments

benchmarkParameters	an object of type DenseMatrixMicrobenchmark specifying various parameters needed to generate input for the dense matrix kernel.
index	an integer index indicating the dimensions of the matrix or vector data to be generated as input for the dense matrix kernel.

TransposeMicrobenchmark

Conducts a single performance trial with the matrix transpose dense matrix kernel

---

Description

TransposeMicrobenchmark conducts a single performance trial of the SVD dense matrix kernel for the matrix given in the kernelParameters parameter. The function times the single function call svd(transposeParameters$A).

Usage

```
TransposeMicrobenchmark(benchmarkParameters, kernelParameters)
```

Arguments

benchmarkParameters	an object of type DenseMatrixMicrobenchmark specifying various parameters for microbenchmarking the dense matrix kernel
kernelParameters	a list of matrices or vectors to be used as input to the dense matrix kernel

Examples

```
## Not run:
# Allocate input to the matrix transpose microbenchmark for the
# first matrix size to be tested
microbenchmarks <- GetDenseMatrixDefaultMicrobenchmarks()
kernelParameters <- TransposeAllocator(microbenchmarks[["transpose"]], 1)
# Execute the microbenchmark
timings <- TransposeMicrobenchmark(microbenchmarks[["transpose"]], kernelParameters)

## End(Not run)
```

`WriteClusteringPerformanceResultsCsv` *Appends performance test results of a clustering microbenchmark to a file in CSV format*

Description

`WriteClusteringPerformanceResultsCsv` appends performance results for a clustering for machine learning microbenchmark to a CSV file.

Usage

```
WriteClusteringPerformanceResultsCsv(numberOfThreads, numberOfFeatures,
  numberOfFeatureVectors, numberOfClusters, averageWallClockTime,
  standardDeviation, csvResultsFileName)
```

Arguments

`numberOfThreads`
: the number of threads all of the performance trials were conducted with

`numberOfFeatures`
: the number of features, i.e. the dimension of the feature vector

`numberOfFeatureVectors`
: the number of feature vectors in the data set

`numberOfClusters`
: the number of clusters in the data set

`averageWallClockTime`
: average wall clock time computed for the data set tested during the performance trials

`standardDeviation`
: standard deviation of the wall clock times obtained for the performance trials

`csvResultsFileName`
: the CSV results file the performance result will be appended to

Details

This function appends the performance results obtained by a single clustering for machine learning microbenchmark conducted with a specific data set. If the CSV file does not exist, header information is printed on the first line to describe the subsequent entries. Each entry includes the number of features, number of feature vectors, and number of clusters in the data set. The performance results included in each entry are the average of the wall clock times obtained for the performance trials, the standard deviation of the performance trial wall clock times, and the number of threads the performance trials were conducted with.

`WriteDenseMatrixPerformanceResultsCsv` *Appends dense matrix performance test results to a file in CSV format*

Description

`WriteDenseMatrixPerformanceResultsCsv` appends performance results for a single dense matrix microbenchmark to a CSV file

Usage

```
WriteDenseMatrixPerformanceResultsCsv(numberOfThreads, dimensionParameter,
  averageWallClockTime, standardDeviation, csvResultsFileName)
```

Arguments

`numberOfThreads`
: the number of threads all of the performance trials were conducted with

`dimensionParameter`
: an integer vector specifying the dimension parameter the microbenchmark uses to define the dimension of the test matrix

`averageWallClockTime`
: average wall clock time computed for the matrix tested during the performance trials

`standardDeviation`
: standard deviation of the wall clock times obtained for the performance trials

`csvResultsFileName`
: the CSV results file the performance result will be appended to

Details

This function appends to a CSV file the performance results obtained by a single dense matrix performance microbenchmark conducted for a specific matrix. If the CSV file does not exist, header information is printed on the first line to describe the subsequent entries. Each entry consists of the dimension parameter used to specify the dimensions of the matrix, the average of the wall clock times obtained for the performance trials, the standard deviation of the performance trial wall clock times, and the number of threads the performance trials conducted with.

`WriteSparseMatrixPerformanceResultsCsv` *Appends sparse matrix performance test results to a file in CSV format*

Description

`WriteSparseMatrixPerformanceResultsCsv` appends performance results for a single sparse matrix performance test to a CSV file.

Usage

```
WriteSparseMatrixPerformanceResultsCsv(numberOfThreads, numberOfRows,
  numberOfColumns, numberOfNonzeros, averageWallClockTime, standardDeviation,
  csvResultsFileName)
```

Arguments

`numberOfThreads`	the number of threads all of the performance trials were conducted with
`numberOfRows`	the number of rows in the matrix
`numberOfColumns`	the number of columns in the matrix
`numberOfNonzeros`	the number of non-zero elements in the matrix
`averageWallClockTime`	average wall clock time computed for the matrix tested during the performance trials
`standardDeviation`	standard deviation of the wall clock times obtained for the performance trials
`csvResultsFileName`	the CSV results file the performance result will be appended to

Details

This function appends the performance results obtained by a single sparse matrix microbenchmark conducted for a specific matrix. If the CSV file does not exist, header information is printed on the first line to describe the subsequent entries. Each entry consists of the dimension parameter used to specify the dimensions of the matrix, the average of the wall clock times obtained for the performance trials, the standard deviation of the performance trial wall clock times, and the number of threads the performance trials were conducted with.

Index