NnmfPack

2.2

Generated by Doxygen 1.8.15

Contents

1 File Index **1**

1.1 File List 1

2 File Documentation **3**

2.1 bdiv_cpu.c File Reference 3

2.1.1 Detailed Description 4

2.1.2 Function Documentation 5

2.1.2.1 dbdiv_cpu() 5

2.1.2.2 dbdivg_cpu() 6

2.1.2.3 dbdivone_cpu() 7

2.1.2.4 derrorbd0_x86() 8

2.1.2.5 derrorbd1_x86() 8

2.1.2.6 derrorbd_x86() 9

2.1.2.7 derrorbdg_x86() 9

2.1.2.8 dkernelH_x86() 10

2.1.2.9 dkernelW_x86() 10

2.1.2.10 dupdate1H_x86() 11

2.1.2.11 dupdate1W_x86() 11

2.1.2.12 dupdate2H_x86() 11

2.1.2.13 dupdate2W_x86() 12

2.1.2.14 sbdiv_cpu() 12

2.1.2.15 sbdivg_cpu() 12

2.1.2.16 sbdivone_cpu() 13

2.1.2.17 serrorbd0_x86() 14

2.1.2.18 serrorbd1_x86() . . . 14
2.1.2.19 serrorbd_x86() . . . 15
2.1.2.20 serrorbdg_x86() . . . 15
2.1.2.21 skernelH_x86() . . . 16
2.1.2.22 skernelW_x86() . . . 16
2.1.2.23 supdate1H_x86() . . . 17
2.1.2.24 supdate1W_x86() . . . 17
2.1.2.25 supdate2H_x86() . . . 18
2.1.2.26 supdate2W_x86() . . . 18
2.2 bdiv_cpu.h File Reference . . . 19
2.2.1 Detailed Description . . . 20
2.2.2 Function Documentation . . . 20
2.2.2.1 dbdiv_cpu() . . . 21
2.2.2.2 dbdivg_cpu() . . . 21
2.2.2.3 dbdivone_cpu() . . . 23
2.2.2.4 derrorbd0_x86() . . . 24
2.2.2.5 derrorbd1_x86() . . . 24
2.2.2.6 derrorbd_x86() . . . 24
2.2.2.7 derrorbdg_x86() . . . 25
2.2.2.8 dkernelH_x86() . . . 25
2.2.2.9 dkernelW_x86() . . . 26
2.2.2.10 dupdate1H_x86() . . . 26
2.2.2.11 dupdate1W_x86() . . . 27
2.2.2.12 dupdate2H_x86() . . . 27
2.2.2.13 dupdate2W_x86() . . . 27
2.2.2.14 sbdiv_cpu() . . . 28
2.2.2.15 sbdivg_cpu() . . . 28
2.2.2.16 sbdivone_cpu() . . . 29
2.2.2.17 serrorbd0_x86() . . . 30
2.2.2.18 serrorbd1_x86() . . . 30

2.2.2.19 serrorbd_x86() . . . 30
2.2.2.20 serrorbdg_x86() . . . 31
2.2.2.21 skernelH_x86() . . . 31
2.2.2.22 skernelW_x86() . . . 32
2.2.2.23 supdate1H_x86() . . . 32
2.2.2.24 supdate1W_x86() . . . 33
2.2.2.25 supdate2H_x86() . . . 33
2.2.2.26 supdate2W_x86() . . . 34
2.3 bdiv_cuda.cu File Reference . . . 34
2.4 bdiv_cuda.h File Reference . . . 34
2.4.1 Detailed Description . . . 35
2.4.2 Function Documentation . . . 36
2.4.2.1 dbdiv_cuda() . . . 36
2.4.2.2 dbdivg_cuda() . . . 36
2.4.2.3 dbdivone_cuda() . . . 36
2.4.2.4 dkernelH_cuda() . . . 37
2.4.2.5 dkernelW_cuda() . . . 37
2.4.2.6 dupdate1H_cuda() . . . 37
2.4.2.7 dupdate1W_cuda() . . . 37
2.4.2.8 dupdate2H_cuda() . . . 38
2.4.2.9 dupdate2W_cuda() . . . 38
2.4.2.10 sbdiv_cuda() . . . 38
2.4.2.11 sbdivg_cuda() . . . 38
2.4.2.12 sbdivone_cuda() . . . 39
2.4.2.13 skernelH_cuda() . . . 39
2.4.2.14 skernelW_cuda() . . . 39
2.4.2.15 supdate1H_cuda() . . . 39
2.4.2.16 supdate1W_cuda() . . . 40
2.4.2.17 supdate2H_cuda() . . . 40
2.4.2.18 supdate2W_cuda() . . . 40

2.4.2.19 vdkernelH_cuda() . . . 40
2.4.2.20 vdkernelW_cuda() . . . 41
2.4.2.21 vdupdate1H_cuda() . . . 41
2.4.2.22 vdupdate1W_cuda() . . . 41
2.4.2.23 vdupdate2H_cuda() . . . 41
2.4.2.24 vdupdate2W_cuda() . . . 41
2.4.2.25 vskernelH_cuda() . . . 42
2.4.2.26 vskernelW_cuda() . . . 42
2.4.2.27 vsupdate1H_cuda() . . . 42
2.4.2.28 vsupdate1W_cuda() . . . 42
2.4.2.29 vsupdate2H_cuda() . . . 42
2.4.2.30 vsupdate2W_cuda() . . . 43
2.5 bdiv_mic.c File Reference . . . 43
2.5.1 Detailed Description . . . 43
2.5.2 Function Documentation . . . 44
2.5.2.1 dbdiv_mic() . . . 44
2.5.2.2 dbdivg_mic() . . . 44
2.5.2.3 dbdivone_mic() . . . 46
2.5.2.4 sbdiv_mic() . . . 47
2.5.2.5 sbdivg_mic() . . . 47
2.5.2.6 sbdivone_mic() . . . 48
2.6 bdiv_mic.h File Reference . . . 48
2.6.1 Detailed Description . . . 49
2.6.2 Function Documentation . . . 49
2.6.2.1 dbdiv_mic() . . . 50
2.6.2.2 dbdivg_mic() . . . 50
2.6.2.3 dbdivone_mic() . . . 52
2.6.2.4 dmlsa_mic() . . . 53
2.6.2.5 sbdiv_mic() . . . 54
2.6.2.6 sbdivg_mic() . . . 54

2.6.2.7 sbdivone_mic() . . . 55
2.6.2.8 smlsa_mic() . . . 55
2.7 config.h File Reference . . . 56
2.8 dbdiv1_cpu.c File Reference . . . 56
2.8.1 Detailed Description . . . 56
2.8.2 Function Documentation . . . 57
2.8.2.1 main() . . . 57
2.9 dbdiv1_cuda.cu File Reference . . . 57
2.10 dbdiv1_mic.c File Reference . . . 57
2.10.1 Detailed Description . . . 57
2.10.2 Function Documentation . . . 57
2.10.2.1 main() . . . 58
2.11 dbdiv2_cpu.c File Reference . . . 58
2.11.1 Detailed Description . . . 58
2.11.2 Function Documentation . . . 58
2.11.2.1 main() . . . 58
2.12 dbdiv2_cuda.cu File Reference . . . 59
2.13 dbdiv2_mic.c File Reference . . . 59
2.13.1 Detailed Description . . . 59
2.13.2 Function Documentation . . . 59
2.13.2.1 main() . . . 59
2.14 dgenerate.c File Reference . . . 59
2.14.1 Detailed Description . . . 60
2.14.2 Function Documentation . . . 60
2.14.2.1 main() . . . 60
2.15 mex_bdivCpu.c File Reference . . . 60
2.15.1 Detailed Description . . . 61
2.15.2 Function Documentation . . . 61
2.15.2.1 mexFunction() . . . 61
2.16 mex_bdivCuda.cu File Reference . . . 61

2.17 mex_bdivMic.c File Reference . . . 61
2.17.1 Detailed Description . . . 62
2.17.2 Function Documentation . . . 62
2.17.2.1 mexFunction() . . . 62
2.18 mlsa_cpu.c File Reference . . . 62
2.18.1 Function Documentation . . . 63
2.18.1.1 ddotdiv_x86() . . . 63
2.18.1.2 dmlsa_cpu() . . . 63
2.18.1.3 sdotdiv_x86() . . . 64
2.18.1.4 smlsa_cpu() . . . 65
2.19 mlsa_cpu.h File Reference . . . 65
2.19.1 Detailed Description . . . 66
2.19.2 Function Documentation . . . 66
2.19.2.1 ddotdiv_x86() . . . 66
2.19.2.2 dmlsa_cpu() . . . 67
2.19.2.3 sdotdiv_x86() . . . 68
2.19.2.4 smlsa_cpu() . . . 68
2.20 mlsa_cuda.cu File Reference . . . 69
2.21 mlsa_cuda.h File Reference . . . 69
2.21.1 Detailed Description . . . 69
2.21.2 Function Documentation . . . 70
2.21.2.1 ddotdiv_cuda() . . . 70
2.21.2.2 dmlsa_cuda() . . . 70
2.21.2.3 sdotdiv_cuda() . . . 70
2.21.2.4 smlsa_cuda() . . . 70
2.21.2.5 vddotdiv_cuda() . . . 71
2.21.2.6 vsdotdiv_cuda() . . . 71
2.22 mlsa_mic.c File Reference . . . 71
2.22.1 Detailed Description . . . 71
2.22.2 Function Documentation . . . 72

2.22.2.1 dmlsa_mic() . . . 72
2.22.2.2 smlsa_mic() . . . 73
2.23 mlsa_mic.h File Reference . . . 73
2.23.1 Detailed Description . . . 74
2.23.2 Function Documentation . . . 74
2.23.2.1 dmlsa_mic() . . . 74
2.23.2.2 smlsa_mic() . . . 75
2.24 nnmfpackCpu.h File Reference . . . 76
2.24.1 Detailed Description . . . 76
2.25 nnmfpackCuda.h File Reference . . . 76
2.25.1 Detailed Description . . . 77
2.26 nnmfpackMic.h File Reference . . . 77
2.26.1 Detailed Description . . . 77
2.27 sbdiv1_cpu.c File Reference . . . 77
2.27.1 Detailed Description . . . 78
2.27.2 Function Documentation . . . 78
2.27.2.1 main() . . . 78
2.28 sbdiv1_cuda.cu File Reference . . . 78
2.29 sbdiv1_mic.c File Reference . . . 78
2.29.1 Detailed Description . . . 79
2.29.2 Function Documentation . . . 79
2.29.2.1 main() . . . 79
2.30 sbdiv2_cpu.c File Reference . . . 79
2.30.1 Detailed Description . . . 79
2.30.2 Function Documentation . . . 80
2.30.2.1 main() . . . 80
2.31 sbdiv2_cuda.cu File Reference . . . 80
2.32 sbdiv2_mic.c File Reference . . . 80
2.32.1 Detailed Description . . . 80
2.32.2 Function Documentation . . . 80

2.32.2.1 main() . . . 81
2.33 sgenerate.c File Reference . . . 81
2.33.1 Detailed Description . . . 81
2.33.2 Function Documentation . . . 81
2.33.2.1 main() . . . 81
2.34 utils.c File Reference . . . 82
2.34.1 Detailed Description . . . 82
2.34.2 Function Documentation . . . 82
2.34.2.1 dread_file() . . . 82
2.34.2.2 dwrite_ascii_file() . . . 83
2.34.2.3 dwrite_file() . . . 83
2.34.2.4 read_file_header() . . . 84
2.34.2.5 sread_file() . . . 84
2.34.2.6 swrite_ascii_file() . . . 84
2.34.2.7 swrite_file() . . . 85
2.35 utils.h File Reference . . . 85
2.35.1 Detailed Description . . . 86
2.35.2 Function Documentation . . . 86
2.35.2.1 dread_file() . . . 86
2.35.2.2 dwrite_ascii_file() . . . 87
2.35.2.3 dwrite_file() . . . 87
2.35.2.4 read_file_header() . . . 87
2.35.2.5 sread_file() . . . 88
2.35.2.6 swrite_ascii_file() . . . 88
2.35.2.7 swrite_file() . . . 89
2.36 utils_cuda.cu File Reference . . . 89
2.37 utils_cuda.h File Reference . . . 89
2.37.1 Detailed Description . . . 90
2.37.2 Function Documentation . . . 90
2.37.2.1 ddiv_cuda() . . . 90

2.37.2.2 derror_cuda() . . . 90
2.37.2.3 derrorbd_cuda() . . . 91
2.37.2.4 dlarngenn_cuda() . . . 91
2.37.2.5 dmemset_cuda() . . . 91
2.37.2.6 dsub_cuda() . . . 91
2.37.2.7 sdiv_cuda() . . . 91
2.37.2.8 serror_cuda() . . . 92
2.37.2.9 serrorbd_cuda() . . . 92
2.37.2.10 slarngenn_cuda() . . . 92
2.37.2.11 smemset_cuda() . . . 92
2.37.2.12 ssub_cuda() . . . 92
2.37.2.13 vddiv_cuda() . . . 93
2.37.2.14 vderrorbd0_cuda() . . . 93
2.37.2.15 vderrorbd1_cuda() . . . 93
2.37.2.16 vderrorbdg_cuda() . . . 93
2.37.2.17 vdmemset_cuda() . . . 93
2.37.2.18 vdsub_cuda() . . . 94
2.37.2.19 vsdiv_cuda() . . . 94
2.37.2.20 vserrorbd0_cuda() . . . 94
2.37.2.21 vserrorbd1_cuda() . . . 94
2.37.2.22 vserrorbdg_cuda() . . . 94
2.37.2.23 vsmemset_cuda() . . . 95
2.37.2.24 vssub_cuda() . . . 95
2.38 utils_x86.c File Reference . . . 95
2.38.1 Detailed Description . . . 96
2.38.2 Function Documentation . . . 96
2.38.2.1 ddiv_x86() . . . 96
2.38.2.2 derror_x86() . . . 96
2.38.2.3 dlarngenn_x86() . . . 97
2.38.2.4 dmemset_x86() . . . 97

2.38.2.5 dsub_x86() . . . 98
2.38.2.6 sdiv_x86() . . . 98
2.38.2.7 serror_x86() . . . 98
2.38.2.8 slarngenn_x86() . . . 99
2.38.2.9 smemset_x86() . . . 99
2.38.2.10 ssub_x86() . . . 100
2.39 utils_x86.h File Reference . . . 100
2.39.1 Detailed Description . . . 101
2.39.2 Function Documentation . . . 101
2.39.2.1 ddiv_x86() . . . 101
2.39.2.2 derror_x86() . . . 102
2.39.2.3 dlarngenn_x86() . . . 102
2.39.2.4 dlarnv_() . . . 103
2.39.2.5 dmemset_x86() . . . 103
2.39.2.6 dsub_x86() . . . 103
2.39.2.7 sdiv_x86() . . . 104
2.39.2.8 serror_x86() . . . 104
2.39.2.9 slarngenn_x86() . . . 104
2.39.2.10 slarnv_() . . . 105
2.39.2.11 smemset_x86() . . . 105
2.39.2.12 ssub_x86() . . . 105

Chapter 1

File Index

1.1 File List

Here is a list of all files with brief descriptions:

bdiv_cpu.c
File with functions to calcule NNMF using betadivergence methods for CPUs . . . 3
bdiv_cpu.h
Header file for using the betadivergence cuda functions with CPU . . . 19
bdiv_cuda.cu . . . 34
bdiv_cuda.h
Header file for using the betadivergence cuda functions with GPUs . . . 34
bdiv_mic.c
File with functions to calcule NNMF using betadivergence methods with Intel Xeon Phi (MIC) . . . 43
bdiv_mic.h
Header file for using the betadivergence functions with MIC/MIC . . . 48
config.h . . . 56
dbdiv1_cpu.c
Beta Divergence (bdiv) example of usage. Matrices A, W and H are randomly generated. Double precision is used. Layout 1D column-mayor . . . 56
dbdiv1_cuda.cu . . . 57
dbdiv1_mic.c
Beta Divergence (bdiv) example of usage. Matrices A, W and H are randomly generated. Double precision is used. Layout 1D column-mayor . . . 57
dbdiv2_cpu.c
Beta Divergence (bdiv) example of usage. It reads matrices A, W and H from files and writes updated matrices W and H to files. Double precision is used. Layout 1D column-mayor . . . 58
dbdiv2_cuda.cu . . . 59
dbdiv2_mic.c
Beta Divergence (bdiv) example of usage. It reads matrices A, W and H from files and writes updated matrices W and H to files. Double precision is used. Layout 1D column-mayor . . . 59
dgenerate.c
Auxiliar code to randomly generate matrices. Matrices are written to a binary file. Double precision is used. Layout: 1D Column mayor . . . 59
mex_bdivCpu.c
Mex_bdivCpu is the nnmfpack's CPU-driver (nnmfpack's functions runs on CPU) for using it from from Matlab/Octave. Layout 1D column-mayor . . . 60
mex_bdivCuda.cu . . . 61
mex_bdivMic.c
Mex_bdivMic is the nnmfpack's MIC-driver (nnmfpack's functions runs on Intel Xeon Phi) for using it from from Matlab/Octave. Layout 1D column-mayor . . . 61

mlsa_cpu.c 62
mlsa_cpu.h
File with functions to calcule NNMF using the mlsa algorithm for CPUs 65
mlsa_cuda.cu 69
mlsa_cuda.h
Header file for using the mlsa algorithm using cuda functions with GPUs 69
mlsa_mic.c
File with functions to calcule NNMF using mlsa algorithm with Intel Xeon Phi (MIC) 71
mlsa_mic.h
Header file for using the mlsa algorithm with MIC/MIC 73
nnmfpackCpu.h
General header file for using NnmfPack with CPUs 76
nnmfpackCuda.h
General header file for using NnmfPack with CUDA/GPUs 76
nnmfpackMic.h
General header file for using NnmfPack with Intel Xeon Phi (MIC) 77
sbdiv1_cpu.c
Beta Divergence (bdiv) example of usage. Matrices A, W and H are randomly generated. Simple precision is used. Layout 1D column-mayor 77
sbdiv1_cuda.cu 78
sbdiv1_mic.c
Beta Divergence (bdiv) example of usage. Matrices A, W and H are randomly generated. Simple precision is used. Layout 1D column-mayor 78
sbdiv2_cpu.c
Beta Divergence (bdiv) example of usage. It reads matrices A, W and H from files and writes updated matrices W and H to files. Simple precision is used. Layout 1D column-mayor 79
sbdiv2_cuda.cu 80
sbdiv2_mic.c
Beta Divergence (bdiv) example of usage. It reads matrices A, W and H from files and writes updated matrices W and H to files. Simple precision is used. Layout 1D column-mayor 80
sgenerate.c
Auxiliar code to randomly generate matrices. Matrices are written to a binary file. Simple precision is used. Layout: 1D Column mayor 81
utils.c
Some transversal auxiliar functions. Double and simple precision 82
utils.h
Header file for transversal auxiliar functions. Double and simple precision 85
utils_cuda.cu 89
utils_cuda.h
Header file for using utility modules from CUDA source codes 89
utils_x86.c
Some auxiliar functions. Double and simple precision for CPU and MIC 95
utils_x86.h
Header file for using utility modules from CPU/MIC source codes 100

Chapter 2

File Documentation

2.1 bdiv_cpu.c File Reference

File with functions to calcule NNMF using betadivergence methods for CPUs.

Functions

- int dbdivg_cpu (const int m, const int n, const int k, const double *A, double *W, double *H, const double expo, const int uType, int nIter)

  dbdivg_cpu performs the NNMF using beta-divergence when beta is != 1 and !=2, using double precision.
- int sbdivg_cpu (const int m, const int n, const int k, const float *A, float *W, float *H, const float expo, const int uType, int nIter)

  sbdivg_cpu performs NNMF using betadivergence for general case (beta <> 1 and 2) using simple precision
- int dbdivone_cpu (const int m, const int n, const int k, const double *A, double *W, double *H, const int uType, const int nIter)

  dbdivone_cpu performs NNMF using beta-divergence when beta=1, using double precision
- int sbdivone_cpu (const int m, const int n, const int k, const float *A, float *W, float *H, const int uType, const int nIter)

  sbdivone_cpu performs NNMF using betadivergence when beta=1 using simple precision
- int dbdiv_cpu (const int m, const int n, const int k, const double *A, double *W, double *H, const double beta, const int uType, const int nIter)

  dbdiv_cpu is a wrapper that calls the adequate function to performs NNMF using betadivergence using double precision with CPU
- int sbdiv_cpu (const int m, const int n, const int k, const float *A, float *W, float *H, const float beta, const int uType, const int nIter)
- void dkernelH_x86 (const int m, const int n, const double *L, const double *A, double *__restrict__ R, const double expo)
- void skernelH_x86 (const int m, const int n, const float *L, const float *A, float *__restrict__ R, const float expo)

  *This function computes simple precision R(i)=(L(i)^expo)*A[i] and R(i+m*n)=L[i]*(L(i)^expo) Note "expo" is a real number expo < 0 or expo > 0.*
- void dkernelW_x86 (const int m, const int n, const double *L, const double *A, double *__restrict__ R, const double expo)

  *This function computes double precision R(pos)=L(i)^expo)*A(i) and R(pos+m)=L(i)*(L(i)^expo) Note expo is a real number expo < 0 or expo > 0.*
- void skernelW_x86 (const int m, const int n, const float *L, const float *A, float *__restrict__ R, const float expo)

This function computes simple precision R(pos)=L(i)^ expo)∗A(i) and R(pos+m)=L(i)∗(L(i)^ expo) Note expo is a real number expo < 0 or expo > 0.

- void dupdate1H_x86 (const int n, const double ∗X, double ∗__restrict__ H)

  This function computes double precision H(i)=H(i)∗B(i)/C(i) where matrices B and C are stored in the same buffer (called X). All B 1st and after C.

- void supdate1H_x86 (const int n, const float ∗X, float ∗__restrict__ H)

  This function computes simple precision H(i)=H(i)∗B(i)/C(i) where matrices B and C are stored in the same buffer (called X). All B 1st and after C.

- void dupdate1W_x86 (const int m, const int n, const double ∗X, double ∗__restrict__ W)
- void supdate1W_x86 (const int m, const int n, const float ∗X, float ∗__restrict__ W)

  This function computes double precision W[i]=W[i]∗D[i]/E[i] where matrices D and E are stored in the same buffer (called X) according beta-divergence general case (see dbdivg_x86(...))

- void dupdate2H_x86 (const int m, const int n, const double ∗y, const double ∗B, double ∗__restrict__ H)

  This function computes double precision H(i)=H(i)∗(B(i)/y(j))

- void supdate2H_x86 (const int m, const int n, const float ∗y, const float ∗B, float ∗__restrict__ H)

  This function performs the simple H(i)=H(i)∗(B(i)/y(j))

- void dupdate2W_x86 (const int m, const int n, const double ∗y, const double ∗B, double ∗__restrict__ W)

  This function performs double precision W(i)=W(i)∗(B(i)/y(j))

- void supdate2W_x86 (const int m, const int n, const float ∗y, const float ∗B, float ∗__restrict__ W)

  This function computes simple precision W(i)=W(i)∗(B(i)/y(j))

- void derrorbd0_x86 (const int n, const double ∗x, double ∗__restrict__ y)

  This function performs auxiliar double precision operations when error is computed using betadivergence error formula and beta = 0.

- void serrorbd0_x86 (const int n, const float ∗x, float ∗__restrict__ y)

  This function performs auxiliar simple precision operations when error is computed using betadivergence error formula and beta = 0.

- void derrorbd1_x86 (const int n, const double ∗x, double ∗__restrict__ y)

  This function performs auxiliar double precision operations when error is computed using betadivergence error formula and beta = 1.

- void serrorbd1_x86 (const int n, const float ∗x, float ∗__restrict__ y)

  This function performs auxiliar simple precision operations when error is computed using betadivergence error formula and beta = 1.

- void derrorbdg_x86 (const int n, const double ∗x, double ∗__restrict__ y, const double beta)

  This function performs auxiliar double precision operations when error is computed using betadivergence error formula with (beta != 0) and (beta != 1)

- void serrorbdg_x86 (const int n, const float ∗x, float ∗__restrict__ y, const float beta)

  This function performs auxiliar simple precision operations when error is computed using betadivergence error formula with (beta != 0) and (beta != 1)

- double derrorbd_x86 (const int m, const int n, const int k, const double ∗A, const double ∗W, const double ∗H, const double beta)

  This function returns double precision error when error is computed using betadivergence error formula.

- float serrorbd_x86 (const int m, const int n, const int k, const float ∗A, const float ∗W, const float ∗H, const float beta)

  This function returns simple precision error when error is computed using betadivergence error formula.

2.1.1 Detailed Description

File with functions to calcule NNMF using betadivergence methods for CPUs.

Author

Information Retrieval and Parallel Computing Group (IRPCG)
University of Oviedo, Spain
Interdisciplinary Computation and Communication Group (INCO2)
Universitat Politecnica de Valencia, Spain.
Contact: nnmfpack@gmail.com

Date

04/11/14

2.1.2 Function Documentation

2.1.2.1 dbdiv_cpu()

```
int dbdiv_cpu (
            const int m,
            const int n,
            const int k,
            const double * A,
            double * W,
            double * H,
            const double beta,
            const int uType,
            const int nIter )
```

dbdiv_cpu is a wrapper that calls the adequate function to performs NNMF using betadivergence using double precision with CPU

Parameters

m	(input) Number of rows of matrix A and matrix W
n	(input) Number of columns of matrix A and matrix H
k	(input) Number of columns of matrix W and rows of H
A	(input) Double precision input matrix of (m $*$ n) number of elements stored using 1D column-major
W	(inout) Double precision input/output matrix of (m $*$ k) number of elements stored using 1D column-major
H	(inout) Double precision input/output matrix of (k $*$ n) number of elements stored using 1D column-major
beta	(input) Double precision value. The parameter beta of betadivergence method
uType	(input) It can be UpdateAll (W and H are updated), UpdateW (only H is updated) or UpdateH (only W is updated)
nIter	(input) Number of iterations

Returns

: 0 if all is OK, -1 otherwise

Definition at line 1059 of file bdiv_cpu.c.

2.1.2.2 dbdivg_cpu()

```
int dbdivg_cpu (
              const int m,
              const int n,
              const int k,
              const double * A,
              double * W,
              double * H,
              const double expo,
              const int uType,
              int nIter )
```

dbdivg_cpu performs the NNMF using beta-divergence when beta is != 1 and !=2, using double precision.

The algorithm is
repit nIter times
STEP 1
L=W∗H
L1=L(.^)(beta-2)
L2=L1(.∗)A
L1=L1(.∗)L
B=W'*L2*
C=W'∗L1
H=H(.)B(./)C

STEP 2
L=W∗H
L1=L(.^)(beta-2)
L2=L1(.∗)A
L1=L1(.∗)L
D=L2∗H'
E=L1∗H'
W=W(.∗)D(./)E
end repit
End algorithm

In real live L1 and L2 are (m∗n) matrices used in STEP 1 and 2. For performance reasons only one 1D colum-mayor buffer, named R in next code, of size 2∗m∗n is used In STEP 1, first part of R (m∗n positions) is L2 and the second part is L1. In STEP 2, fisrt column of R (2∗m positions) is composed by the first column of L2 following first column of L1. Second column of R (2∗m positions) is composed by the sencond column of L2 following second column of L1. 3rd column of R ... and so on

In real live B and C are (k∗n) matrices used in STEP 1, and D and E are (m∗k) matrices used in STEP 2. B/C and D/E are independent. However we do not have L1 and L2, we have R, and we can do B=W'∗L2 and C=W'∗L1 (or D=L2∗H' and E=L1∗H') at the same time. For this reason only one matrix is declared to save space. This is matrix M with size 2∗max(m,n)∗k

Parameters

m	(input) Number of rows of matrix A and matrix W

Parameters

n	(input) Number of columns of matrix A and matrix H
k	(input) Number of columns of matrix W and rows of H
A	(input) Double precision input matrix of (m ∗ n) number of elements stored using 1D column-major
W	(inout) Double precision input/output matrix of (m ∗ k) number of elements stored using 1D column-major
H	(inout) Double precision input/output matrix of (k ∗ n) number of elements stored using 1D column-major
expo	(input) Double precision value. The exponent beta of betadivergence method
uType	(input) It can be UpdateAll (W and H are updated), UpdateW (only H is updated) or UpdateH (only W is updated)
nIter	(input) Number of iterations

It returns 0 if all is OK.

Definition at line 87 of file bdiv_cpu.c.

2.1.2.3 dbdivone_cpu()

```
int dbdivone_cpu (
              const int m,
              const int n,
              const int k,
              const double * A,
              double * W,
              double * H,
              const int uType,
              const int nIter )
```

dbdivone_cpu performs NNMF using beta-divergence when beta=1, using double precision

The algorithm is
repit nIter times
STEP 1
y(i)=sum(W(j,i) for all j in range) for all i in range
L=W∗H
L=A(./)L
B=W'*L*
B(i,j)=B(i,j) / y(i) for all B elements
H=H(.)B

STEP 2
y(i)=sum(H(i,j) for all j in range) for all i in range
L=W∗H
L=A(./)L
D=L∗H'
B(i,j)=B(i,j) / y(j) for all B elements
W=W(.∗)D
end repit
End algorithm

In real live B is a (k∗n) matrix used in STEP 1, and D is a (m∗k) matrix used in STEP 2. B and D are independent. For this reason only 1 matrix of size max(m,n)∗k is declared/used

Parameters

m	(input) Number of rows of matrix A and matrix W
n	(input) Number of columns of matrix A and matrix H
k	(input) Number of columns of matrix W and rows of H
A	(input) Double precision input matrix of (m $*$ n) number of elements stored using 1D column-major
W	(inout) Double precision input/output matrix of (m $*$ k) number of elements stored using 1D column-major
H	(inout) Double precision input/output matrix of (k $*$ n) number of elements stored using 1D column-major
uType	(input) It can be UpdateAll (W and H are updated), UpdateW (only H is updated) or UpdateH (only W is updated)
nIter	(input) Number of iterations

It returns 0 if all is OK.

Definition at line 566 of file bdiv_cpu.c.

2.1.2.4 derrorbd0_x86()

```
_void derrorbd0_x86 (
             const int n,
             const double * x,
             double *__restrict__ y )
```

This function performs auxiliar double precision operations when error is computed using betadivergence error formula and beta = 0.

Parameters

n	(input) Number of elements of x and y
x	(input) Double precision input vector/matrix
y	(inout) Double precision input/output vector/matrix

Definition at line 1569 of file bdiv_cpu.c.

2.1.2.5 derrorbd1_x86()

```
void derrorbd1_x86 (
             const int n,
             const double * x,
             double *__restrict__ y )
```

This function performs auxiliar double precision operations when error is computed using betadivergence error formula and beta = 1.

Parameters

n	(input) Number of elements of x and y
x	(input) Double precision input vector/matrix
y	(inout) Double precision input/output vector/matrix

Definition at line 1635 of file bdiv_cpu.c.

2.1.2.6 derrorbd_x86()

```
void derrorbd_x86 (
             const int m,
             const int n,
             const int k,
             const double * A,
             const double * W,
             const double * H,
             const double beta )
```

This function returns double precision error when error is computed using betadivergence error formula.

Parameters

m	(input) Number of rows of A and W
n	(input) Number of columns of A and H
k	(input) Number of columns/rows of W/H
A	(input) Double precision input matrix A
W	(input) Double precision input matrix W
H	(input) Double precision input matrix H
beta	(input) Double precision beta value

Definition at line 1782 of file bdiv_cpu.c.

2.1.2.7 derrorbdg_x86()

```
void derrorbdg_x86 (
             const int n,
             const double * x,
             double *__restrict__ y,
             const double beta )
```

This function performs auxiliar double precision operations when error is computed using betadivergence error formula with (beta != 0) and (beta != 1)

Parameters

n	(input) Number of elements of x and y
x	(input) Double precision input vector/matrix
y	(inout) Double precision input/output vector/matrix
beta	(input) Double precision value of beta

Definition at line 1702 of file bdiv_cpu.c.

2.1.2.8 dkernelH_x86()

```
void dkernelH_x86 (
             const int m,
             const int n,
             const double * L,
             const double * A,
             double *__restrict__ R,
             const double expo )
```

Definition at line 1116 of file bdiv_cpu.c.

2.1.2.9 dkernelW_x86()

```
void dkernelW_x86 (
             const int m,
             const int n,
             const double * L,
             const double * A,
             double *__restrict__ R,
             const double expo )
```

This function computes double precision R(pos)=L(i)^expo)∗A(i) and R(pos+m)=L(i)∗(L(i)^expo) Note expo is a real number expo $<$ 0 or expo $>$ 0.

Parameters

m	(input) Number of rows of L, A and R matrices
n	(input) Number of columns of L, A and R matrices
L	(input) Double precision input matrix (1D column-major)
A	(input) Double precision input matrix (1D column-major)
R	(output) Double precision output matrix (1D column-major)
expo	(input) the "power of" for function pow(). It is a double precision value

Definition at line 1204 of file bdiv_cpu.c.

2.1.2.10 dupdate1H_x86()

```
dupdate1H_x86 (
             const int n,
             const double * X,
             double *__restrict__ H )
```

This function computes double precision H(i)=H(i)∗B(i)/C(i) where matrices B and C are stored in the same buffer (called X). All B 1st and after C.

Parameters

n	(input) Number of elements of H
X	(input) Simple precision input matrix (1D column-major)
H	(inout) Simple precision input/output matrix (1D column-major)

Definition at line 1293 of file bdiv_cpu.c.

2.1.2.11 dupdate1W_x86()

```
void dupdate1W_x86 (
             const int m,
             const int n,
             const double * X,
             double *__restrict__ W )
```

Definition at line 1359 of file bdiv_cpu.c.

2.1.2.12 dupdate2H_x86()

```
void dupdate2H_x86 (
             const int m,
             const int n,
             const double * y,
             const double * B,
             double *__restrict__ H )
```

This function computes double precision H(i)=H(i)∗(B(i)/y(j))

Parameters

m	(input) Number of rows of B and H, and number of elements of vector y
n	(input) Number of columns of B and A
y	(input) Double precision vector with the sum of W columns
B	(input) Double precision input matrix (1D column-major)
H	(inout) Double precision input/output matrix (1D column-major)

Definition at line 1435 of file bdiv_cpu.c.

2.1.2.13 dupdate2W_x86()

```
void dupdate2W_x86 (
             const int m,
             const int n,
             const double * y,
             const double * B,
             double *__restrict__ W )
```

This function performs double precision W(i)=W(i)∗(B(i)/y(j))

Parameters

m	(input) Number of rows of W and B,
n	(input) Number of columns of W and B, and number of elements of vector y
y	(input) Double precision vector with the sum of H rows
B	(input) Double precision input matrix (1D column-major)
W	(inout) Double precision input/output matrix (1D column-major)

Definition at line 1503 of file bdiv_cpu.c.

2.1.2.14 sbdiv_cpu()

```
int sbdiv_cpu (
             const int m,
             const int n,
             const int k,
             const float * A,
             float * W,
             float * H,
             const float beta,
             const int uType,
             const int nIter )
```

Definition at line 1089 of file bdiv_cpu.c.

2.1.2.15 sbdivg_cpu()

```
int sbdivg_cpu (
             const int m,
             const int n,
             const int k,
             const float * A,
```

```
            float * W,
            float * H,
            const float expo,
            const int uType,
            int nIter )
```

sbdivg_cpu performs NNMF using betadivergence for general case (beta <> 1 and 2) using simple precision

Parameters

m	(input) Number of rows of matrix A and matrix W
n	(input) Number of columns of matrix A and matrix H
k	(input) Number of columns of matrix W and rows of H
A	(input) Simple precision input matrix of (m ∗ n) number of elements stored using 1D column-major
W	(inout) Simple precision input/output matrix of (m ∗ k) number of elements stored using 1D column-major
H	(inout) Simple precision input/output matrix of (k ∗ n) number of elements stored using 1D column-major
expo	(input) Simple precision value. The exponent beta of betadivergence method
uType	(input) It can be UpdateAll (W and H are updated), UpdateW (only H is updated) or UpdateH (only W is updated)
nIter	(input) Number of iterations

Returns

: 0 if all is OK, -1 otherwise

Definition at line 313 of file bdiv_cpu.c.

2.1.2.16 sbdivone_cpu()

```
int sbdivone_cpu (
            const int m,
            const int n,
            const int k,
            const float * A,
            float * W,
            float * H,
            const int uType,
            const int nIter )
```

sbdivone_cpu performs NNMF using betadivergence when beta=1 using simple precision

Parameters

m	(input) Number of rows of matrix A and matrix W
n	(input) Number of columns of matrix A and matrix H
k	(input) Number of columns of matrix W and rows of H
A	(input) Simple precision input matrix of (m ∗ n) number of elements stored using 1D column-major

Parameters

W	(inout) Simple precision input/output matrix of (m ∗ k) number of elements stored using 1D column-major
H	(inout) Simple precision input/output matrix of (k ∗ n) number of elements stored using 1D column-major
uType	(input) It can be UpdateAll (W and H are updated), UpdateW (only H is updated) or UpdateH (only W is updated)
nIter	(input) Number of iterations

Returns

: 0 if all is OK, -1 otherwise

Definition at line 813 of file bdiv_cpu.c.

2.1.2.17 serrorbd0_x86()

```
void serrorbd0_x86 (
            const int n,
            const float * x,
            float *__restrict__ y )
```

This function performs auxiliar simple precision operations when error is computed using betadivergence error formula and beta = 0.

Parameters

n	(input) Number of elements of x and y
x	(input) Simple precision input vector/matrix
y	(inout) Simple precision input/output vector/matrix

Definition at line 1602 of file bdiv_cpu.c.

2.1.2.18 serrorbd1_x86()

```
void serrorbd1_x86 (
            const int n,
            const float * x,
            float *__restrict__ y )
```

This function performs auxiliar simple precision operations when error is computed using betadivergence error formula and beta = 1.

Parameters

n	(input) Number of elements of x and y
x	(input) Simple precision input vector/matrix
y	(inout) Simple precision input/output vector/matrix

Definition at line 1668 of file bdiv_cpu.c.

2.1.2.19 serrorbd_x86()

```
void serrorbd_x86 (
            const int m,
            const int n,
            const int k,
            const float * A,
            const float * W,
            const float * H,
            const float beta )
```

This function returns simple precision error when error is computed using betadivergence error formula.

Parameters

m	(input) Number of rows of A and W
n	(input) Number of columns of A and H
k	(input) Number of columns/rows of W/H
A	(input) Simple precision input matrix A
W	(input) Simple precision input matrix W
H	(input) Simple precision input matrix H
beta	(input) Simple precision beta value

Definition at line 1849 of file bdiv_cpu.c.

2.1.2.20 serrorbdg_x86()

```
void serrorbdg_x86 (
            const int n,
            const float * x,
            float *__restrict__ y,
            const float beta )
```

This function performs auxiliar simple precision operations when error is computed using betadivergence error formula with (beta != 0) and (beta != 1)

Parameters

n	(input) Number of elements of x and y
x	(input) Simple precision input vector/matrix
y	(inout) Simple precision input/output vector/matrix
beta	(input) Simple precision value of beta

Definition at line 1741 of file bdiv_cpu.c.

2.1.2.21 skernelH_x86()

```
void skernelH_x86 (
             const int m,
             const int n,
             const float * L,
             const float * A,
             float *__restrict__ R,
             const float expo )
```

This function computes simple precision R(i)=(L(i)^expo)∗A[i] and R(i+m∗n)=L[i]∗(L(i)^expo) Note "expo" is a real number expo $<$ 0 or expo $>$ 0.

Parameters

m	(input) Number of rows of L, A and R matrices
n	(input) Number of columns of L, A and R matrices
L	(input) Simpel precision input matrix (1D column-major)
A	(input) Simpel precision input matrix (1D column-major)
R	(output) Simpel precision output matrix (1D column-major)
expo	(input) the "power of" for function pow(). It is a simple precision value

Definition at line 1160 of file bdiv_cpu.c.

2.1.2.22 skernelW_x86()

```
void skernelW_x86 (
             const int m,
             const int n,
             const float * L,
             const float * A,
             float *__restrict__ R,
             const float expo )
```

This function computes simple precision R(pos)=L(i)^expo)∗A(i) and R(pos+m)=L(i)∗(L(i)^expo) Note expo is a real number expo $<$ 0 or expo $>$ 0.

Parameters

m	(input) Number of rows of A
n	(input) Number of columns of A
L	(input) Simple precision input matrix (1D column-major)
A	(input) Simple precision input matrix (1D column-major)
R	(output) Simple precision output matrix (1D column-major)
expo	(input) the "power of" for function pow(). It is a simple precision value

Definition at line 1250 of file bdiv_cpu.c.

2.1.2.23 supdate1H_x86()

```
void supdate1H_x86 (
            const int n,
            const float * X,
            float *__restrict__ H )
```

This function computes simple precision H(i)=H(i)∗B(i)/C(i) where matrices B and C are stored in the same buffer (called X). All B 1st and after C.

Parameters

n	(input) Number of elements of H
X	(input) Simple precision input matrix (1D column-major)
H	(inout) Simple precision input/output matrix (1D column-major)

Definition at line 1325 of file bdiv_cpu.c.

2.1.2.24 supdate1W_x86()

```
void supdate1W_x86 (
            const int m,
            const int n,
            const float * X,
            float *__restrict__ W )
```

This function computes double precision W[i]=W[i]∗D[i]/E[i] where matrices D and E are stored in the same buffer (called X) according beta-divergence general case (see dbdivg_x86(...))

Parameters

m	(input) Number of rows of W
n	(input) Number of columns of W
X	(input) Simple precision input matrix (1D column-major)
W	(inout) Simple precision input/output matrix (1D column-major)

Definition at line 1397 of file bdiv_cpu.c.

2.1.2.25 supdate2H_x86()

```
void supdate2H_x86 (
             const int m,
             const int n,
             const float * y,
             const float * B,
             float *__restrict__ H )
```

This function performs the simple H(i)=H(i)∗(B(i)/y(j))

Parameters

m	(input) Number of rows of B and H, and number of elements of vector y
n	(input) Number of columns of B and A
y	(input) Simple precision vector with the sum of W columns
B	(input) Simple precision input matrix (1D column-major)
H	(inout) Simple precision input/output matrix (1D column-major)

Definition at line 1469 of file bdiv_cpu.c.

2.1.2.26 supdate2W_x86()

```
void supdate2W_x86 (
             const int m,
             const int n,
             const float * y,
             const float * B,
             float *__restrict__ W )
```

This function computes simple precision W(i)=W(i)∗(B(i)/y(j))

Parameters

m	(input) Number of rows of W and B,
n	(input) Number of columns of W and B, and number of elements of vector y
y	(input) Simple precision vector with the sum of H rows
B	(input) Simple precision input matrix (1D column-major)
W	(inout) Simple precision input/output matrix (1D column-major)

Definition at line 1537 of file bdiv_cpu.c.

2.2 bdiv_cpu.h File Reference

Header file for using the betadivergence cuda functions with CPU.

Functions

- int dbdiv_cpu (const int m, const int n, const int k, const double *A, double *W, double *H, const double beta, const int uType, const int nIter)

  dbdiv_cpu is a wrapper that calls the adequate function to performs NNMF using betadivergence using double precision with CPU
- int sbdiv_cpu (const int m, const int n, const int k, const float *A, float *W, float *H, const float beta, const int uType, const int nIter)
- int dbdivg_cpu (const int m, const int n, const int k, const double *A, double *W, double *H, const double beta, const int uType, const int nIter)

  dbdivg_cpu performs the NNMF using beta-divergence when beta is != 1 and !=2, using double precision.
- int sbdivg_cpu (const int m, const int n, const int k, const float *A, float *W, float *H, const float beta, const int uType, const int nIter)

  sbdivg_cpu performs NNMF using betadivergence for general case (beta <> 1 and 2) using simple precision
- int dbdivone_cpu (const int m, const int n, const int k, const double *A, double *W, double *H, const int uType, const int nIter)

  dbdivone_cpu performs NNMF using beta-divergence when beta=1, using double precision
- int sbdivone_cpu (const int m, const int n, const int k, const float *A, float *W, float *H, const int uType, const int nIter)

  sbdivone_cpu performs NNMF using betadivergence when beta=1 using simple precision
- void dupdate1H_x86 (const int n, const double *X, double *__restrict__ H)

  *This function computes double precision H(i)=H(i)*B(i)/C(i) where matrices B and C are stored in the same buffer (called X). All B 1st and after C.*
- void supdate1H_x86 (const int n, const float *X, float *__restrict__ H)

  *This function computes simple precision H(i)=H(i)*B(i)/C(i) where matrices B and C are stored in the same buffer (called X). All B 1st and after C.*
- void dupdate1W_x86 (const int m, const int n, const double *X, double *__restrict__ W)
- void supdate1W_x86 (const int m, const int n, const float *X, float *__restrict__ W)

  *This function computes double precision W[i]=W[i]*D[i]/E[i] where matrices D and E are stored in the same buffer (called X) according beta-divergence general case (see dbdivg_x86(...))*
- void dkernelH_x86 (const int m, const int n, const double *L, const double *A, double *__restrict__ R, const double expo)
- void skernelH_x86 (const int m, const int n, const float *L, const float *A, float *__restrict__ R, const float expo)

  *This function computes simple precision R(i)=(L(i)^expo)*A[i] and R(i+m*n)=L[i]*(L(i)^expo) Note "expo" is a real number expo < 0 or expo > 0.*
- void dkernelW_x86 (const int m, const int n, const double *L, const double *A, double *__restrict__ R, const double expo)

  *This function computes double precision R(pos)=L(i)^expo)*A(i) and R(pos+m)=L(i)*(L(i)^expo) Note expo is a real number expo < 0 or expo > 0.*
- void skernelW_x86 (const int m, const int n, const float *L, const float *A, float *__restrict__ R, const float expo)

  *This function computes simple precision R(pos)=L(i)^expo)*A(i) and R(pos+m)=L(i)*(L(i)^expo) Note expo is a real number expo < 0 or expo > 0.*
- void dupdate2H_x86 (const int m, const int n, const double *X, const double *B, double *__restrict__ H)

  This function computes double precision H(i)=H(i)(B(i)/y(j))*
- void supdate2H_x86 (const int m, const int n, const float *X, const float *B, float *__restrict__ H)

  This function performs the simple H(i)=H(i)(B(i)/y(j))*

- void dupdate2W_x86 (const int m, const int n, const double *X, const double *B, double *__restrict__ W)

  This function performs double precision W(i)=W(i)(B(i)/y(j))*

- void supdate2W_x86 (const int m, const int n, const float *X, const float *B, float *__restrict__ W)

  This function computes simple precision W(i)=W(i)(B(i)/y(j))*

- void derrorbd0_x86 (const int n, const double *x, double *__restrict__ y)

  This function performs auxiliar double precision operations when error is computed using betadivergence error formula and beta = 0.

- void serrorbd0_x86 (const int n, const float *x, float *__restrict__ y)

  This function performs auxiliar simple precision operations when error is computed using betadivergence error formula and beta = 0.

- void derrorbd1_x86 (const int n, const double *x, double *__restrict__ y)

  This function performs auxiliar double precision operations when error is computed using betadivergence error formula and beta = 1.

- void serrorbd1_x86 (const int n, const float *x, float *__restrict__ y)

  This function performs auxiliar simple precision operations when error is computed using betadivergence error formula and beta = 1.

- void derrorbdg_x86 (const int n, const double *x, double *__restrict__ y, const double beta)

  This function performs auxiliar double precision operations when error is computed using betadivergence error formula with (beta != 0) and (beta != 1)

- void serrorbdg_x86 (const int n, const float *x, float *__restrict__ y, const float beta)

  This function performs auxiliar simple precision operations when error is computed using betadivergence error formula with (beta != 0) and (beta != 1)

- double derrorbd_x86 (const int m, const int n, const int k, const double *A, const double *W, const double *H, const double betadiv)

  This function returns double precision error when error is computed using betadivergence error formula.

- float serrorbd_x86 (const int m, const int n, const int k, const float *A, const float *W, const float *H, const float betadiv)

  This function returns simple precision error when error is computed using betadivergence error formula.

2.2.1 Detailed Description

Header file for using the betadivergence cuda functions with CPU.

Author

Information Retrieval and Parallel Computing Group (IRPCG)
University of Oviedo, Spain
Interdisciplinary Computation and Communication Group (INCO2)
Universitat Politecnica de Valencia, Spain.
Contact: nnmfpack@gmail.com

Date

04/11/14

2.2.2 Function Documentation

2.2.2.1 dbdiv_cpu()

```
int dbdiv_cpu (
            const int m,
            const int n,
            const int k,
            const double * A,
            double * W,
            double * H,
            const double beta,
            const int uType,
            const int nIter )
```

dbdiv_cpu is a wrapper that calls the adequate function to performs NNMF using betadivergence using double precision with CPU

Parameters

m	(input) Number of rows of matrix A and matrix W
n	(input) Number of columns of matrix A and matrix H
k	(input) Number of columns of matrix W and rows of H
A	(input) Double precision input matrix of (m $*$ n) number of elements stored using 1D column-major
W	(inout) Double precision input/output matrix of (m $*$ k) number of elements stored using 1D column-major
H	(inout) Double precision input/output matrix of (k $*$ n) number of elements stored using 1D column-major
beta	(input) Double precision value. The parameter beta of betadivergence method
uType	(input) It can be UpdateAll (W and H are updated), UpdateW (only H is updated) or UpdateH (only W is updated)
nIter	(input) Number of iterations

Returns

: 0 if all is OK, -1 otherwise

Definition at line 1059 of file bdiv_cpu.c.

2.2.2.2 dbdivg_cpu()

```
int dbdivg_cpu (
            const int m,
            const int n,
            const int k,
            const double * A,
            double * W,
            double * H,
            const double expo,
            const int uType,
            int nIter )
```

dbdivg_cpu performs the NNMF using beta-divergence when beta is != 1 and !=2, using double precision.

The algorithm is
repit nIter times
STEP 1
L=W∗H
L1=L(.^)(beta-2)
L2=L1(.∗)A
L1=L1(.∗)L
B=W'*L2*
C=W'∗L1
*H=H(.)*B(./)C

STEP 2
L=W∗H
L1=L(.^)(beta-2)
L2=L1(.∗)A
L1=L1(.∗)L
D=L2∗H'
E=L1∗H'
W=W(.∗)D(./)E
end repit
End algorithm

In real live L1 and L2 are (m∗n) matrices used in STEP 1 and 2. For performance reasons only one 1D colum-mayor buffer, named R in next code, of size 2∗m∗n is used In STEP 1, first part of R (m∗n positions) is L2 and the second part is L1. In STEP 2, fisrt column of R (2∗m positions) is composed by the first column of L2 following first column of L1. Second column of R (2∗m positions) is composed by the sencond column of L2 following second column of L1. 3rd column of R ... and so on

In real live B and C are (k∗n) matrices used in STEP 1, and D and E are (m∗k) matrices used in STEP 2. B/C and D/E are independent. However we do not have L1 and L2, we have R, and we can do B=W'∗L2 and C=W'∗L1 (or D=L2∗H' and E=L1∗H') at the same time. For this reason only one matrix is declared to save space. This is matrix M with size 2∗max(m,n)∗k

Parameters

m	(input) Number of rows of matrix A and matrix W
n	(input) Number of columns of matrix A and matrix H
k	(input) Number of columns of matrix W and rows of H
A	(input) Double precision input matrix of (m ∗ n) number of elements stored using 1D column-major
W	(inout) Double precision input/output matrix of (m ∗ k) number of elements stored using 1D column-major
H	(inout) Double precision input/output matrix of (k ∗ n) number of elements stored using 1D column-major
expo	(input) Double precision value. The exponent beta of betadivergence method
uType	(input) It can be UpdateAll (W and H are updated), UpdateW (only H is updated) or UpdateH (only W is updated)
nIter	(input) Number of iterations

It returns 0 if all is OK.

Definition at line 87 of file bdiv_cpu.c.

2.2.2.3 dbdivone_cpu()

```
int dbdivone_cpu (
              const int m,
              const int n,
              const int k,
              const double * A,
              double * W,
              double * H,
              const int uType,
              const int nIter )
```

dbdivone_cpu performs NNMF using beta-divergence when beta=1, using double precision

The algorithm is
repit nIter times
STEP 1
y(i)=sum(W(j,i) for all j in range) for all i in range
L=W∗H
L=A(./)L
B=W'*L*
B(i,j)=B(i,j) / y(i) for all B elements
H=H(.)B

STEP 2
y(i)=sum(H(i,j) for all j in range) for all i in range
L=W∗H
L=A(./)L
D=L∗H'
B(i,j)=B(i,j) / y(j) for all B elements
W=W(.∗)D
end repit
End algorithm

In real live B is a (k∗n) matrix used in STEP 1, and D is a (m∗k) matrix used in STEP 2. B and D are independent. For this reason only 1 matrix of size max(m,n)∗k is declared/used

Parameters

m	(input) Number of rows of matrix A and matrix W
n	(input) Number of columns of matrix A and matrix H
k	(input) Number of columns of matrix W and rows of H
A	(input) Double precision input matrix of (m ∗ n) number of elements stored using 1D column-major
W	(inout) Double precision input/output matrix of (m ∗ k) number of elements stored using 1D column-major
H	(inout) Double precision input/output matrix of (k ∗ n) number of elements stored using 1D column-major
uType	(input) It can be UpdateAll (W and H are updated), UpdateW (only H is updated) or UpdateH (only W is updated)
nIter	(input) Number of iterations

It returns 0 if all is OK.

Definition at line 566 of file bdiv_cpu.c.

2.2.2.4 derrorbd0_x86()

```
void derrorbd0_x86 (
            const int n,
            const double * x,
            double *__restrict__ y )
```

This function performs auxiliar double precision operations when error is computed using betadivergence error formula and beta = 0.

Parameters

n	(input) Number of elements of x and y
x	(input) Double precision input vector/matrix
y	(inout) Double precision input/output vector/matrix

Definition at line 1569 of file bdiv_cpu.c.

2.2.2.5 derrorbd1_x86()

```
void derrorbd1_x86 (
            const int n,
            const double * x,
            double *__restrict__ y )
```

This function performs auxiliar double precision operations when error is computed using betadivergence error formula and beta = 1.

Parameters

n	(input) Number of elements of x and y
x	(input) Double precision input vector/matrix
y	(inout) Double precision input/output vector/matrix

Definition at line 1635 of file bdiv_cpu.c.

2.2.2.6 derrorbd_x86()

```
double derrorbd_x86 (
            const int m,
            const int n,
            const int k,
            const double * A,
```

```
            const double * W,
            const double * H,
            const double beta )
```

This function returns double precision error when error is computed using betadivergence error formula.

Parameters

m	(input) Number of rows of A and W
n	(input) Number of columns of A and H
k	(input) Number of columns/rows of W/H
A	(input) Double precision input matrix A
W	(input) Double precision input matrix W
H	(input) Double precision input matrix H
beta	(input) Double precision beta value

Definition at line 1782 of file bdiv_cpu.c.

2.2.2.7 derrorbdg_x86()

```
void derrorbdg_x86 (
            const int n,
            const double * x,
            double *__restrict__ y,
            const double beta )
```

This function performs auxiliar double precision operations when error is computed using betadivergence error formula with (beta != 0) and (beta != 1)

Parameters

n	(input) Number of elements of x and y
x	(input) Double precision input vector/matrix
y	(inout) Double precision input/output vector/matrix
beta	(input) Double precision value of beta

Definition at line 1702 of file bdiv_cpu.c.

2.2.2.8 dkernelH_x86()

```
void dkernelH_x86 (
            const int m,
            const int n,
            const double * L,
            const double * A,
```

```
            double *__restrict__ R,
            const double expo )
```

Definition at line 1116 of file bdiv_cpu.c.

2.2.2.9 dkernelW_x86()

```
void dkernelW_x86 (
            const int m,
            const int n,
            const double * L,
            const double * A,
            double *__restrict__ R,
            const double expo )
```

This function computes double precision R(pos)=L(i)^expo)∗A(i) and R(pos+m)=L(i)∗(L(i)^expo) Note expo is a real number expo $<$ 0 or expo $>$ 0.

Parameters

m	(input) Number of rows of L, A and R matrices
n	(input) Number of columns of L, A and R matrices
L	(input) Double precision input matrix (1D column-major)
A	(input) Double precision input matrix (1D column-major)
R	(output) Double precision output matrix (1D column-major)
expo	(input) the "power of" for function pow(). It is a double precision value

Definition at line 1204 of file bdiv_cpu.c.

2.2.2.10 dupdate1H_x86()

```
void dupdate1H_x86 (
            const int n,
            const double * X,
            double *__restrict__ H )
```

This function computes double precision H(i)=H(i)∗B(i)/C(i) where matrices B and C are stored in the same buffer (called X). All B 1st and after C.

Parameters

n	(input) Number of elements of H
X	(input) Simple precision input matrix (1D column-major)
H	(inout) Simple precision input/output matrix (1D column-major)

Definition at line 1293 of file bdiv_cpu.c.

2.2.2.11 dupdate1W_x86()

```
void dupdate1W_x86 (
            const int m,
            const int n,
            const double * X,
            double *__restrict__ W )
```

Definition at line 1359 of file bdiv_cpu.c.

2.2.2.12 dupdate2H_x86()

```
void dupdate2H_x86 (
            const int m,
            const int n,
            const double * y,
            const double * B,
            double *__restrict__ H )
```

This function computes double precision H(i)=H(i)∗(B(i)/y(j))

Parameters

m	(input) Number of rows of B and H, and number of elements of vector y
n	(input) Number of columns of B and A
y	(input) Double precision vector with the sum of W columns
B	(input) Double precision input matrix (1D column-major)
H	(inout) Double precision input/output matrix (1D column-major)

Definition at line 1435 of file bdiv_cpu.c.

2.2.2.13 dupdate2W_x86()

```
void dupdate2W_x86 (
            const int m,
            const int n,
            const double * y,
            const double * B,
            double *__restrict__ W )
```

This function performs double precision W(i)=W(i)∗(B(i)/y(j))

Parameters

m	(input) Number of rows of W and B,
n	(input) Number of columns of W and B, and number of elements of vector y
y	(input) Double precision vector with the sum of H rows
B	(input) Double precision input matrix (1D column-major)
W	(inout) Double precision input/output matrix (1D column-major)

Definition at line 1503 of file bdiv_cpu.c.

2.2.2.14 sbdiv_cpu()

```
int sbdiv_cpu (
              const int m,
              const int n,
              const int k,
              const float * A,
              float * W,
              float * H,
              const float beta,
              const int uType,
              const int nIter )
```

Definition at line 1089 of file bdiv_cpu.c.

2.2.2.15 sbdivg_cpu()

```
int sbdivg_cpu (
              const int m,
              const int n,
              const int k,
              const float * A,
              float * W,
              float * H,
              const float expo,
              const int uType,
              int nIter )
```

sbdivg_cpu performs NNMF using betadivergence for general case (beta $<>$ 1 and 2) using simple precision

Parameters

m	(input) Number of rows of matrix A and matrix W
n	(input) Number of columns of matrix A and matrix H
k	(input) Number of columns of matrix W and rows of H
A	(input) Simple precision input matrix of (m $*$ n) number of elements stored using 1D column-major
W	(inout) Simple precision input/output matrix of (m $*$ k) number of elements stored using 1D column-major

Parameters

H	(inout) Simple precision input/output matrix of (k ∗ n) number of elements stored using 1D column-major
expo	(input) Simple precision value. The exponent beta of betadivergence method
uType	(input) It can be UpdateAll (W and H are updated), UpdateW (only H is updated) or UpdateH (only W is updated)
nIter	(input) Number of iterations

Returns

: 0 if all is OK, -1 otherwise

Definition at line 313 of file bdiv_cpu.c.

2.2.2.16 sbdivone_cpu()

```
int sbdivone_cpu (
            const int m,
            const int n,
            const int k,
            const float * A,
            float * W,
            float * H,
            const int uType,
            const int nIter )
```

sbdivone_cpu performs NNMF using betadivergence when beta=1 using simple precision

Parameters

m	(input) Number of rows of matrix A and matrix W
n	(input) Number of columns of matrix A and matrix H
k	(input) Number of columns of matrix W and rows of H
A	(input) Simple precision input matrix of (m ∗ n) number of elements stored using 1D column-major
W	(inout) Simple precision input/output matrix of (m ∗ k) number of elements stored using 1D column-major
H	(inout) Simple precision input/output matrix of (k ∗ n) number of elements stored using 1D column-major
uType	(input) It can be UpdateAll (W and H are updated), UpdateW (only H is updated) or UpdateH (only W is updated)
nIter	(input) Number of iterations

Returns

: 0 if all is OK, -1 otherwise

Definition at line 813 of file bdiv_cpu.c.

2.2.2.17 serrorbd0_x86()

```
void serrorbd0_x86 (
            const int n,
            const float * x,
            float *__restrict__ y )
```

This function performs auxiliar simple precision operations when error is computed using betadivergence error formula and beta = 0.

Parameters

n	(input) Number of elements of x and y
x	(input) Simple precision input vector/matrix
y	(inout) Simple precision input/output vector/matrix

Definition at line 1602 of file bdiv_cpu.c.

2.2.2.18 serrorbd1_x86()

```
void serrorbd1_x86 (
            const int n,
            const float * x,
            float *__restrict__ y )
```

This function performs auxiliar simple precision operations when error is computed using betadivergence error formula and beta = 1.

Parameters

n	(input) Number of elements of x and y
x	(input) Simple precision input vector/matrix
y	(inout) Simple precision input/output vector/matrix

Definition at line 1668 of file bdiv_cpu.c.

2.2.2.19 serrorbd_x86()

```
float serrorbd_x86 (
            const int m,
            const int n,
            const int k,
            const float * A,
            const float * W,
            const float * H,
            const float beta )
```

This function returns simple precision error when error is computed using betadivergence error formula.

Parameters

m	(input) Number of rows of A and W
n	(input) Number of columns of A and H
k	(input) Number of columns/rows of W/H
A	(input) Simple precision input matrix A
W	(input) Simple precision input matrix W
H	(input) Simple precision input matrix H
beta	(input) Simple precision beta value

Definition at line 1849 of file bdiv_cpu.c.

2.2.2.20 serrorbdg_x86()

```
void serrorbdg_x86 (
            const int n,
            const float * x,
            float *__restrict__ y,
            const float beta )
```

This function performs auxiliar simple precision operations when error is computed using betadivergence error formula with (beta != 0) and (beta != 1)

Parameters

n	(input) Number of elements of x and y
x	(input) Simple precision input vector/matrix
y	(inout) Simple precision input/output vector/matrix
beta	(input) Simple precision value of beta

Definition at line 1741 of file bdiv_cpu.c.

2.2.2.21 skernelH_x86()

```
void skernelH_x86 (
            const int m,
            const int n,
            const float * L,
            const float * A,
            float *__restrict__ R,
            const float expo )
```

This function computes simple precision R(i)=(L(i)^expo)∗A[i] and R(i+m∗n)=L[i]∗(L(i)^expo) Note "expo" is a real number expo $<$ 0 or expo $>$ 0.

Parameters

m	(input) Number of rows of L, A and R matrices
n	(input) Number of columns of L, A and R matrices
L	(input) Simpel precision input matrix (1D column-major)
A	(input) Simpel precision input matrix (1D column-major)
R	(output) Simpel precision output matrix (1D column-major)
expo	(input) the "power of" for function pow(). It is a simple precision value

Definition at line 1160 of file bdiv_cpu.c.

2.2.2.22 skernelW_x86()

```
void skernelW_x86 (
            const int m,
            const int n,
            const float * L,
            const float * A,
            float *__restrict__ R,
            const float expo )
```

This function computes simple precision R(pos)=L(i)^expo)∗A(i) and R(pos+m)=L(i)∗(L(i)^expo) Note expo is a real number expo $<$ 0 or expo $>$ 0.

Parameters

m	(input) Number of rows of A
n	(input) Number of columns of A
L	(input) Simple precision input matrix (1D column-major)
A	(input) Simple precision input matrix (1D column-major)
R	(output) Simple precision output matrix (1D column-major)
expo	(input) the "power of" for function pow(). It is a simple precision value

Definition at line 1250 of file bdiv_cpu.c.

2.2.2.23 supdate1H_x86()

```
void supdate1H_x86 (
            const int n,
            const float * X,
            float *__restrict__ H )
```

This function computes simple precision H(i)=H(i)∗B(i)/C(i) where matrices B and C are stored in the same buffer (called X). All B 1st and after C.

Parameters

n	(input) Number of elements of H
X	(input) Simple precision input matrix (1D column-major)
H	(inout) Simple precision input/output matrix (1D column-major)

Definition at line 1325 of file bdiv_cpu.c.

2.2.2.24 supdate1W_x86()

```
void supdate1W_x86 (
            const int m,
            const int n,
            const float * X,
            float *__restrict__ W )
```

This function computes double precision W[i]=W[i]∗D[i]/E[i] where matrices D and E are stored in the same buffer (called X) according beta-divergence general case (see dbdivg_x86(...))

Parameters

m	(input) Number of rows of W
n	(input) Number of columns of W
X	(input) Simple precision input matrix (1D column-major)
W	(inout) Simple precision input/output matrix (1D column-major)

Definition at line 1397 of file bdiv_cpu.c.

2.2.2.25 supdate2H_x86()

```
void supdate2H_x86 (
            const int m,
            const int n,
            const float * y,
            const float * B,
            float *__restrict__ H )
```

This function performs the simple H(i)=H(i)∗(B(i)/y(j))

Parameters

m	(input) Number of rows of B and H, and number of elements of vector y
n	(input) Number of columns of B and A
y	(input) Simple precision vector with the sum of W columns
B	(input) Simple precision input matrix (1D column-major)
H	(inout) Simple precision input/output matrix (1D column-major)

Definition at line 1469 of file bdiv_cpu.c.

2.2.2.26 supdate2W_x86()

```
void supdate2W_x86 (
            const int m,
            const int n,
            const float * y,
            const float * B,
            float *__restrict__ W )
```

This function computes simple precision W(i)=W(i)∗(B(i)/y(j))

Parameters

m	(input) Number of rows of W and B,
n	(input) Number of columns of W and B, and number of elements of vector y
y	(input) Simple precision vector with the sum of H rows
B	(input) Simple precision input matrix (1D column-major)
W	(inout) Simple precision input/output matrix (1D column-major)

Definition at line 1537 of file bdiv_cpu.c.

2.3 bdiv_cuda.cu File Reference

2.4 bdiv_cuda.h File Reference

Header file for using the betadivergence cuda functions with GPUs.

Functions

- int dbdiv_cuda (const int m, const int n, const int k, const double ∗A, double ∗W, double ∗H, const double beta, const int uType, const int nIter)
- int sbdiv_cuda (const int m, const int n, const int k, const float ∗A, float ∗W, float ∗H, const float beta, const int uType, const int nIter)
- int dbdivg_cuda (const int m, const int n, const int k, const double ∗A, double ∗W, double ∗H, const double beta, const int uType, const int nIter)
- int sbdivg_cuda (const int m, const int n, const int k, const float ∗A, float ∗W, float ∗H, const float beta, const int uType, const int nIter)
- int dbdivone_cuda (const int m, const int n, const int k, const double ∗A, double ∗W, double ∗H, const int uType, const int nIter)
- int sbdivone_cuda (const int m, const int n, const int k, const float ∗A, float ∗W, float ∗H, const int uType, const int nIter)
- void dkernelH_cuda (const int m, const int n, const double ∗L, const double ∗A, double ∗R, const double expo, cudaStream_t stream)

- void skernelH_cuda (const int m, const int n, const float *L, const float *A, float *R, const float expo, cuda↩ Stream_t stream)
- void dkernelW_cuda (const int m, const int n, const double *L, const double *A, double *R, const double expo, cudaStream_t stream)
- void skernelW_cuda (const int m, const int n, const float *L, const float *A, float *R, const float expo, cuda↩ Stream_t stream)
- void dupdate1H_cuda (const int n, const double *X, double *H, cudaStream_t stream)
- void supdate1H_cuda (const int n, const float *X, float *H, cudaStream_t stream)
- void dupdate2H_cuda (const int m, const int n, const double *X, const double *B, double *H, cudaStream_t stream)
- void supdate2H_cuda (const int m, const int n, const float *X, const float *B, float *H, cudaStream_t stream)
- void dupdate1W_cuda (const int m, const int n, const double *X, double *W, cudaStream_t stream)
- void supdate1W_cuda (const int m, const int n, const float *X, float *W, cudaStream_t stream)
- void dupdate2W_cuda (const int m, const int n, const double *X, const double *B, double *W, cudaStream_t stream)
- void supdate2W_cuda (const int m, const int n, const float *X, const float *B, float *W, cudaStream_t stream)
- __global__ void vdkernelH_cuda (const int m, const int n, const double *__restrict__ L, const double *__↩ restrict__ A, double *R, const double expo)
- __global__ void vskernelH_cuda (const int m, const int n, const float *__restrict__ L, const float *__restrict↩ __ A, float *R, const float expo)
- __global__ void vdkernelW_cuda (const int m, const int n, const double *__restrict__ L, const double *__↩ restrict__ A, double *R, const double expo)
- __global__ void vskernelW_cuda (const int m, const int n, const float *__restrict__ L, const float *__restrict↩ __ A, float *R, const float expo)
- __global__ void vdupdate1H_cuda (const int n, const double *__restrict__ X, double *H)
- __global__ void vsupdate1H_cuda (const int n, const float *__restrict__ X, float *H)
- __global__ void vdupdate1W_cuda (const int m, const int n, const double *__restrict__ X, double *W)
- __global__ void vsupdate1W_cuda (const int m, const int n, const float *__restrict__ X, float *W)
- __global__ void vdupdate2H_cuda (const int m, const int n, const double *__restrict__ X, const double *_↩ _restrict__ B, double *H)
- __global__ void vsupdate2H_cuda (const int m, const int n, const float *__restrict__ X, const float *__↩ restrict__ B, float *H)
- __global__ void vdupdate2W_cuda (const int m, const int k, const double *__restrict__ X, const double *_↩ _restrict__ B, double *W)
- __global__ void vsupdate2W_cuda (const int m, const int k, const float *__restrict__ X, const float *__↩ restrict__ B, float *W)

2.4.1 Detailed Description

Header file for using the betadivergence cuda functions with GPUs.

Author

Information Retrieval and Parallel Computing Group (IRPCG)
University of Oviedo, Spain
Interdisciplinary Computation and Communication Group (INCO2)
Universitat Politecnica de Valencia, Spain.
Contact: nnmfpack@gmail.com

Date

04/11/14

2.4.2 Function Documentation

2.4.2.1 dbdiv_cuda()

```
int dbdiv_cuda (
            const int m,
            const int n,
            const int k,
            const double * A,
            double * W,
            double * H,
            const double beta,
            const int uType,
            const int nIter )
```

2.4.2.2 dbdivg_cuda()

```
int dbdivg_cuda (
            const int m,
            const int n,
            const int k,
            const double * A,
            double * W,
            double * H,
            const double beta,
            const int uType,
            const int nIter )
```

2.4.2.3 dbdivone_cuda()

```
int dbdivone_cuda (
            const int m,
            const int n,
            const int k,
            const double * A,
            double * W,
            double * H,
            const int uType,
            const int nIter )
```

2.4.2.4 dkernelH_cuda()

```
void dkernelH_cuda (
            const int m,
            const int n,
            const double * L,
            const double * A,
            double * R,
            const double expo,
            cudaStream_t stream )
```

2.4.2.5 dkernelW_cuda()

```
void dkernelW_cuda (
            const int m,
            const int n,
            const double * L,
            const double * A,
            double * R,
            const double expo,
            cudaStream_t stream )
```

2.4.2.6 dupdate1H_cuda()

```
void dupdate1H_cuda (
            const int n,
            const double * X,
            double * H,
            cudaStream_t stream )
```

2.4.2.7 dupdate1W_cuda()

```
void dupdate1W_cuda (
            const int m,
            const int n,
            const double * X,
            double * W,
            cudaStream_t stream )
```

2.4.2.8 dupdate2H_cuda()

```
void dupdate2H_cuda (
             const int m,
             const int n,
             const double * X,
             const double * B,
             double * H,
             cudaStream_t stream )
```

2.4.2.9 dupdate2W_cuda()

```
void dupdate2W_cuda (
             const int m,
             const int n,
             const double * X,
             const double * B,
             double * W,
             cudaStream_t stream )
```

2.4.2.10 sbdiv_cuda()

```
int sbdiv_cuda (
             const int m,
             const int n,
             const int k,
             const float * A,
             float * W,
             float * H,
             const float beta,
             const int uType,
             const int nIter )
```

2.4.2.11 sbdivg_cuda()

```
int sbdivg_cuda (
             const int m,
             const int n,
             const int k,
             const float * A,
             float * W,
             float * H,
             const float beta,
             const int uType,
             const int nIter )
```

2.4.2.12 sbdivone_cuda()

```
int sbdivone_cuda (
            const int m,
            const int n,
            const int k,
            const float * A,
            float * W,
            float * H,
            const int uType,
            const int nIter )
```

2.4.2.13 skernelH_cuda()

```
void skernelH_cuda (
            const int m,
            const int n,
            const float * L,
            const float * A,
            float * R,
            const float expo,
            cudaStream_t stream )
```

2.4.2.14 skernelW_cuda()

```
void skernelW_cuda (
            const int m,
            const int n,
            const float * L,
            const float * A,
            float * R,
            const float expo,
            cudaStream_t stream )
```

2.4.2.15 supdate1H_cuda()

```
void supdate1H_cuda (
            const int n,
            const float * X,
            float * H,
            cudaStream_t stream )
```

2.4.2.16 supdate1W_cuda()

```
void supdate1W_cuda (
             const int m,
             const int n,
             const float * X,
             float * W,
             cudaStream_t stream )
```

2.4.2.17 supdate2H_cuda()

```
void supdate2H_cuda (
             const int m,
             const int n,
             const float * X,
             const float * B,
             float * H,
             cudaStream_t stream )
```

2.4.2.18 supdate2W_cuda()

```
void supdate2W_cuda (
             const int m,
             const int n,
             const float * X,
             const float * B,
             float * W,
             cudaStream_t stream )
```

2.4.2.19 vdkernelH_cuda()

```
__global__ void vdkernelH_cuda (
             const int m,
             const int n,
             const double *__restrict__ L,
             const double *__restrict__ A,
             double * R,
             const double expo )
```

2.4.2.20 vdkernelW_cuda()

```
__global__ void vdkernelW_cuda (
              const int m,
              const int n,
              const double *__restrict__ L,
              const double *__restrict__ A,
              double * R,
              const double expo )
```

2.4.2.21 vdupdate1H_cuda()

```
__global__ void vdupdate1H_cuda (
              const int n,
              const double *__restrict__ X,
              double * H )
```

2.4.2.22 vdupdate1W_cuda()

```
__global__ void vdupdate1W_cuda (
              const int m,
              const int n,
              const double *__restrict__ X,
              double * W )
```

2.4.2.23 vdupdate2H_cuda()

```
__global__ void vdupdate2H_cuda (
              const int m,
              const int n,
              const double *__restrict__ X,
              const double *__restrict__ B,
              double * H )
```

2.4.2.24 vdupdate2W_cuda()

```
__global__ void vdupdate2W_cuda (
              const int m,
              const int k,
              const double *__restrict__ X,
              const double *__restrict__ B,
              double * W )
```

2.4.2.25 vskernelH_cuda()

```
__global__ void vskernelH_cuda (
              const int m,
              const int n,
              const float *__restrict__ L,
              const float *__restrict__ A,
              float * R,
              const float expo )
```

2.4.2.26 vskernelW_cuda()

```
__global__ void vskernelW_cuda (
              const int m,
              const int n,
              const float *__restrict__ L,
              const float *__restrict__ A,
              float * R,
              const float expo )
```

2.4.2.27 vsupdate1H_cuda()

```
__global__ void vsupdate1H_cuda (
              const int n,
              const float *__restrict__ X,
              float * H )
```

2.4.2.28 vsupdate1W_cuda()

```
__global__ void vsupdate1W_cuda (
              const int m,
              const int n,
              const float *__restrict__ X,
              float * W )
```

2.4.2.29 vsupdate2H_cuda()

```
__global__ void vsupdate2H_cuda (
              const int m,
              const int n,
              const float *__restrict__ X,
              const float *__restrict__ B,
              float * H )
```

2.4.2.30 vsupdate2W_cuda()

```
__global__ void vsupdate2W_cuda (
              const int m,
              const int k,
              const float *__restrict__ X,
              const float *__restrict__ B,
              float * W )
```

2.5 bdiv_mic.c File Reference

File with functions to calcule NNMF using betadivergence methods with Intel Xeon Phi (MIC)

Functions

- int dbdivg_mic (const int m, const int n, const int k, const double *A, double *W, double *H, const double expo, const int uType, const int nIter)

  dbdivg_mic performs the NNMF using beta-divergence when beta is != 1 and !=2, using double precision
- int sbdivg_mic (const int m, const int n, const int k, const float *A, float *W, float *H, const float expo, const int uType, const int nIter)

  sbdivg_mic performs NNMF using betadivergence for general case (beta <> 1 and 2) using simple precision
- int dbdivone_mic (const int m, const int n, const int k, const double *A, double *W, double *H, const int uType, const int nIter)

  dbdivone_mic performs NNMF using betadivergence when beta=1 using double precision
- int sbdivone_mic (const int m, const int n, const int k, const float *A, float *W, float *H, const int uType, const int nIter)

  sbdivone_mic performs NNMF using betadivergence when beta=1 using simple precision
- int dbdiv_mic (const int m, const int n, const int k, const double *A, double *W, double *H, const double beta, const int uType, const int nIter)

  dbdiv_mic is a wrapper that calls the adequate function to performs NNMF using betadivergence using double precision with MIC
- int sbdiv_mic (const int m, const int n, const int k, const float *A, float *W, float *H, const float beta, const int uType, const int nIter)

2.5.1 Detailed Description

File with functions to calcule NNMF using betadivergence methods with Intel Xeon Phi (MIC)

Author

Information Retrieval and Parallel Computing Group (IRPCG)
University of Oviedo, Spain
Interdisciplinary Computation and Communication Group (INCO2)
Universitat Politecnica de Valencia, Spain.
Contact: nnmfpack@gmail.com

Date

04/11/14

2.5.2 Function Documentation

2.5.2.1 dbdiv_mic()

```
int dbdiv_mic (
            const int m,
            const int n,
            const int k,
            const double * A,
            double * W,
            double * H,
            const double beta,
            const int uType,
            const int nIter )
```

dbdiv_mic is a wrapper that calls the adequate function to performs NNMF using betadivergence using double precision with MIC

Parameters

m	(input) Number of rows of matrix A and matrix W
n	(input) Number of columns of matrix A and matrix H
k	(input) Number of columns of matrix W and rows of H
A	(input) Double precision input matrix of (m ∗ n) number of elements stored using 1D column-major
W	(inout) Double precision input/output matrix of (m ∗ k) number of elements stored using 1D column-major
H	(inout) Double precision input/output matrix of (k ∗ n) number of elements stored using 1D column-major
beta	(input) Double precision value. The parameter beta of betadivergence method
uType	(input) It can be UpdateAll (W and H are updated), UpdateW (only H is updated) or UpdateH (only W is updated)
nIter	(input) Number of iterations

Returns

: 0 if all is OK, -1 otherwise

Definition at line 274 of file bdiv_mic.c.

2.5.2.2 dbdivg_mic()

```
int dbdivg_mic (
            const int m,
            const int n,
            const int k,
            const double * A,
```

```
            double * W,
            double * H,
            const double expo,
            const int uType,
            const int nIter )
```

dbdivg_mic performs the NNMF using beta-divergence when beta is != 1 and !=2, using double precision

The algorithm is
repit nIter times
STEP 1
L=W∗H
L1=L(.^)(beta-2)
L2=L1(.∗)A
L1=L1(.∗)L
B=W'*L2*
C=W'∗L1
*H=H(.)*B(./)C

STEP 2
L=W∗H
L1=L(.^)(beta-2)
L2=L1(.∗)A
L1=L1(.∗)L
D=L2∗H'
E=L1∗H'
W=W(.∗)D(./)E
end repit
End algorithm

```
In real live L1 and L2 are (m*n) matrices used in STEP 1 and 2. For performance
reasons only one 1D colum-mayor buffer, named R in next code, of size 2*m*n is used
In STEP 1, first part of R (m*n positions) is L2 and the second part is L1.
In STEP 2, fisrt column of R (2*m positions) is composed by the first column of L2
following first column of L1. Second column of R (2*m positions) is composed by the
sencond column of L2 following second column of L1. 3rd column of R ... and so on

In real live B and C are (k*n) matrices used in STEP 1, and D and E are (m*k)
matrices used in STEP 2. B/C and D/E are independent. However we do not have L1
and L2, we have R, and we can do B=W'*L2 and C=W'*L1 (or D=L2*H' and E=L1*H') at
the same time. For this reason only one matrix is declared to save space. This is
matrix M with size 2*max(m,n)*k
```

Parameters

m	(input) Number of rows of matrix A and matrix W
n	(input) Number of columns of matrix A and matrix H
k	(input) Number of columns of matrix W and rows of H
A	(input) Double precision input matrix of (m ∗ n) number of elements stored using 1D column-major
W	(inout) Double precision input/output matrix of (m ∗ k) number of elements stored using 1D column-major
H	(inout) Double precision input/output matrix of (k ∗ n) number of elements stored using 1D column-major
expo	(input) Double precision value. The exponent beta of betadivergence method
uType	(input) It can be UpdateAll (W and H are updated), UpdateW (only H is updated) or UpdateH (only W is updated)
nIter	(input) Number of iterations

It returns 0 if all is OK.

Definition at line 88 of file bdiv_mic.c.

2.5.2.3 dbdivone_mic()

```
int dbdivone_mic (
            const int m,
            const int n,
            const int k,
            const double * A,
            double * W,
            double * H,
            const int uType,
            const int nIter )
```

dbdivone_mic performs NNMF using betadivergence when beta=1 using double precision

The algorithm is
repit nIter times
STEP 1
y(i)=sum(W(j,i) for all j in range) for all i in range
L=W∗H
L=A(./)L
B=W'*L*
B(i,j)=B(i,j) / y(i) for all B elements
H=H(.)B

STEP 2
y(i)=sum(H(i,j) for all j in range) for all i in range
L=W∗H
L=A(./)L
D=L∗H'
B(i,j)=B(i,j) / y(j) for all B elements
W=W(.∗)D
end repit
End algorithm

In real live B is a (k∗n) matrix used in STEP 1, and D is a (m∗k) matrix used in STEP 2. B and D are independent. For this reason only 1 matrix of size max(m,n)∗k is declared/used

Parameters

m	(input) Number of rows of matrix A and matrix W
n	(input) Number of columns of matrix A and matrix H
k	(input) Number of columns of matrix W and rows of H
A	(input) Double precision input matrix of (m ∗ n) number of elements stored using 1D column-major
W	(inout) Double precision input/output matrix of (m ∗ k) number of elements stored using 1D column-major
H	(inout) Double precision input/output matrix of (k ∗ n) number of elements stored using 1D column-major
uType	(input) It can be UpdateAll (W and H are updated), UpdateW (only H is updated) or UpdateH (only W is updated)
nIter	(input) Number of iterations

Returns

: 0 if all is OK, -1 otherwise

Definition at line 195 of file bdiv_mic.c.

2.5.2.4 sbdiv_mic()

```
int sbdiv_mic (
              const int m,
              const int n,
              const int k,
              const float * A,
              float * W,
              float * H,
              const float beta,
              const int uType,
              const int nIter )
```

Definition at line 304 of file bdiv_mic.c.

2.5.2.5 sbdivg_mic()

```
int sbdivg_mic (
              const int m,
              const int n,
              const int k,
              const float * A,
              float * W,
              float * H,
              const float expo,
              const int uType,
              const int nIter )
```

sbdivg_mic performs NNMF using betadivergence for general case (beta $<>$ 1 and 2) using simple precision

Parameters

m	(input) Number of rows of matrix A and matrix W
n	(input) Number of columns of matrix A and matrix H
k	(input) Number of columns of matrix W and rows of H
A	(input) Simple precision input matrix of (m $*$ n) number of elements stored using 1D column-major
W	(inout) Simple precision input/output matrix of (m $*$ k) number of elements stored using 1D column-major
H	(inout) Simple precision input/output matrix of (k $*$ n) number of elements stored using 1D column-major
expo	(input) Double precision value. The exponent beta of betadivergence method
uType	(input) It can be UpdateAll (W and H are updated), UpdateW (only H is updated) or UpdateH (only W is updated)
nIter	(input) Number of iterations

Returns

: 0 if all is OK, -1 otherwise

Definition at line 128 of file bdiv_mic.c.

2.5.2.6 sbdivone_mic()

```
int sbdivone_mic (
            const int m,
            const int n,
            const int k,
            const float * A,
            float * W,
            float * H,
            const int uType,
            const int nIter )
```

sbdivone_mic performs NNMF using betadivergence when beta=1 using simple precision

Parameters

m	(input) Number of rows of matrix A and matrix W
n	(input) Number of columns of matrix A and matrix H
k	(input) Number of columns of matrix W and rows of H
A	(input) Simple precision input matrix of (m $*$ n) number of elements stored using 1D column-major
W	(inout) Simple precision input/output matrix of (m $*$ k) number of elements stored using 1D column-major
H	(inout) Simple precision input/output matrix of (k $*$ n) number of elements stored using 1D column-major
uType	(input) It can be UpdateAll (W and H are updated), UpdateW (only H is updated) or UpdateH (only W is updated)
nIter	(input) Number of iterations

Returns

: 0 if all is OK, -1 otherwise

Definition at line 234 of file bdiv_mic.c.

2.6 bdiv_mic.h File Reference

Header file for using the betadivergence functions with MIC/MIC.

Functions

- int dbdiv_mic (const int m, const int n, const int k, const double *A, double *W, double *H, const double beta, const int uType, const int nIter)

  dbdiv_mic is a wrapper that calls the adequate function to performs NNMF using betadivergence using double precision with MIC
- int sbdiv_mic (const int m, const int n, const int k, const float *A, float *W, float *H, const float beta, const int uType, const int nIter)
- int dbdivg_mic (const int m, const int n, const int k, const double *A, double *W, double *H, const double beta, const int uType, const int nIter)

  dbdivg_mic performs the NNMF using beta-divergence when beta is != 1 and !=2, using double precision
- int sbdivg_mic (const int m, const int n, const int k, const float *A, float *W, float *H, const float beta, const int uType, const int nIter)

  sbdivg_mic performs NNMF using betadivergence for general case (beta <> 1 and 2) using simple precision
- int dmlsa_mic (const int m, const int n, const int k, const double *A, double *W, double *H, const int uType, const int nIter)

  dmlsa_mic performs NNMF using betadivergence when beta=2 using double precision
- int smlsa_mic (const int m, const int n, const int k, const float *A, float *W, float *H, const int uType, const int nIter)

  smlsa_mic performs NNMF using betadivergence when beta=2 using simple precision
- int dbdivone_mic (const int m, const int n, const int k, const double *A, double *W, double *H, const int uType, const int nIter)

  dbdivone_mic performs NNMF using betadivergence when beta=1 using double precision
- int sbdivone_mic (const int m, const int n, const int k, const float *A, float *W, float *H, const int uType, const int nIter)

  sbdivone_mic performs NNMF using betadivergence when beta=1 using simple precision

2.6.1 Detailed Description

Header file for using the betadivergence functions with MIC/MIC.

Author

Information Retrieval and Parallel Computing Group (IRPCG)
University of Oviedo, Spain
Interdisciplinary Computation and Communication Group (INCO2)
Universitat Politecnica de Valencia, Spain.
Contact: nnmfpack@gmail.com

Date

04/11/14

2.6.2 Function Documentation

2.6.2.1 dbdiv_mic()

```
int dbdiv_mic (
            const int m,
            const int n,
            const int k,
            const double * A,
            double * W,
            double * H,
            const double beta,
            const int uType,
            const int nIter )
```

dbdiv_mic is a wrapper that calls the adequate function to performs NNMF using betadivergence using double precision with MIC

Parameters

m	(input) Number of rows of matrix A and matrix W
n	(input) Number of columns of matrix A and matrix H
k	(input) Number of columns of matrix W and rows of H
A	(input) Double precision input matrix of (m ∗ n) number of elements stored using 1D column-major
W	(inout) Double precision input/output matrix of (m ∗ k) number of elements stored using 1D column-major
H	(inout) Double precision input/output matrix of (k ∗ n) number of elements stored using 1D column-major
beta	(input) Double precision value. The parameter beta of betadivergence method
uType	(input) It can be UpdateAll (W and H are updated), UpdateW (only H is updated) or UpdateH (only W is updated)
nIter	(input) Number of iterations

Returns

: 0 if all is OK, -1 otherwise

Definition at line 274 of file bdiv_mic.c.

2.6.2.2 dbdivg_mic()

```
int dbdivg_mic (
            const int m,
            const int n,
            const int k,
            const double * A,
            double * W,
            double * H,
            const double expo,
            const int uType,
            const int nIter )
```

dbdivg_mic performs the NNMF using beta-divergence when beta is != 1 and !=2, using double precision

The algorithm is
repit nIter times
STEP 1
L=W∗H
L1=L(.^)(beta-2)
L2=L1(.∗)A
L1=L1(.∗)L
B=W'*L2*
C=W'∗L1
*H=H(.)*B(./)C

STEP 2
L=W∗H
L1=L(.^)(beta-2)
L2=L1(.∗)A
L1=L1(.∗)L
D=L2∗H'
E=L1∗H'
W=W(.∗)D(./)E
end repit
End algorithm

```
In real live L1 and L2 are (m*n) matrices used in STEP 1 and 2. For performance
reasons only one 1D colum-mayor buffer, named R in next code, of size 2*m*n is used
In STEP 1, first part of R (m*n positions) is L2 and the second part is L1.
In STEP 2, fisrt column of R (2*m positions) is composed by the first column of L2
following first column of L1. Second column of R (2*m positions) is composed by the
sencond column of L2 following second column of L1. 3rd column of R ... and so on

In real live B and C are (k*n) matrices used in STEP 1, and D and E are (m*k)
matrices used in STEP 2. B/C and D/E are independent. However we do not have L1
and L2, we have R, and we can do B=W'*L2 and C=W'*L1 (or D=L2*H' and E=L1*H') at
the same time. For this reason only one matrix is declared to save space. This is
matrix M with size 2*max(m,n)*k
```

Parameters

m	(input) Number of rows of matrix A and matrix W
n	(input) Number of columns of matrix A and matrix H
k	(input) Number of columns of matrix W and rows of H
A	(input) Double precision input matrix of (m ∗ n) number of elements stored using 1D column-major
W	(inout) Double precision input/output matrix of (m ∗ k) number of elements stored using 1D column-major
H	(inout) Double precision input/output matrix of (k ∗ n) number of elements stored using 1D column-major
expo	(input) Double precision value. The exponent beta of betadivergence method
uType	(input) It can be UpdateAll (W and H are updated), UpdateW (only H is updated) or UpdateH (only W is updated)
nIter	(input) Number of iterations

It returns 0 if all is OK.

Definition at line 88 of file bdiv_mic.c.

2.6.2.3 dbdivone_mic()

```
int dbdivone_mic (
            const int m,
            const int n,
            const int k,
            const double * A,
            double * W,
            double * H,
            const int uType,
            const int nIter )
```

dbdivone_mic performs NNMF using betadivergence when beta=1 using double precision

The algorithm is
repit nIter times
STEP 1
y(i)=sum(W(j,i) for all j in range) for all i in range
L=W∗H
L=A(./)L
B=W'*L*
B(i,j)=B(i,j) / y(i) for all B elements
H=H(.)B

STEP 2
y(i)=sum(H(i,j) for all j in range) for all i in range
L=W∗H
L=A(./)L
D=L∗H'
B(i,j)=B(i,j) / y(j) for all B elements
W=W(.∗)D
end repit
End algorithm

In real live B is a (k∗n) matrix used in STEP 1, and D is a (m∗k) matrix used in STEP 2. B and D are independent. For this reason only 1 matrix of size max(m,n)∗k is declared/used

Parameters

m	(input) Number of rows of matrix A and matrix W
n	(input) Number of columns of matrix A and matrix H
k	(input) Number of columns of matrix W and rows of H
A	(input) Double precision input matrix of (m ∗ n) number of elements stored using 1D column-major
W	(inout) Double precision input/output matrix of (m ∗ k) number of elements stored using 1D column-major
H	(inout) Double precision input/output matrix of (k ∗ n) number of elements stored using 1D column-major
uType	(input) It can be UpdateAll (W and H are updated), UpdateW (only H is updated) or UpdateH (only W is updated)
nIter	(input) Number of iterations

Returns

: 0 if all is OK, -1 otherwise

Definition at line 195 of file bdiv_mic.c.

2.6.2.4 dmlsa_mic()

```
int dmlsa_mic (
            const int m,
            const int n,
            const int k,
            const double * A,
            double * W,
            double * H,
            const int uType,
            const int nIter )
```

dmlsa_mic performs NNMF using betadivergence when beta=2 using double precision

The algorithm is
repit nIter times
STEP 1
L=W∗H
B=W'*A*
C=W'∗L
H=H(.)B(./)C

STEP 2
L=W∗H
D=A∗H'
E=L∗H'
W=W(.∗)D(./)E
end repit
End algorithm

To save some FLOPs and RAM a modified Lee and Seung version is used, so we have Step 1: L=W'∗W, C=L∗H and then B=W'∗A; Step 2: L=H∗H', E=W∗L and D=A∗H'. W'∗W (H∗H') and L∗H (W∗L) can do in parallel with W'∗A (A∗H'). Because dgemm works fine we don't use threads to do in parallel.

In real live B and C are (k∗n) matrices used in STEP 1, and D and E are (m∗k) matrices used in STEP 2. B/C and D/E are independent. For this reason only 2 matrices are declared to save space. They are matrices B and C with size max(m,n)∗k

Parameters

m	(input) Number of rows of matrix A and matrix W
n	(input) Number of columns of matrix A and matrix H
k	(input) Number of columns of matrix W and rows of H
A	(input) Double precision input matrix of (m ∗ n) number of elements stored using 1D column-major
W	(inout) Double precision input/output matrix of (m ∗ k) number of elements stored using 1D column-major
H	(inout) Double precision input/output matrix of (k ∗ n) number of elements stored using 1D column-major
uType	(input) It can be UpdateAll (W and H are updated), UpdateW (only H is updated) or UpdateH (only W is updated)
nIter	(input) Number of iterations

Returns

: 0 if all is OK, -1 otherwise

Definition at line 72 of file mlsa_mic.c.

2.6.2.5 sbdiv_mic()

```
int sbdiv_mic (
              const int m,
              const int n,
              const int k,
              const float * A,
              float * W,
              float * H,
              const float beta,
              const int uType,
              const int nIter )
```

Definition at line 304 of file bdiv_mic.c.

2.6.2.6 sbdivg_mic()

```
int sbdivg_mic (
              const int m,
              const int n,
              const int k,
              const float * A,
              float * W,
              float * H,
              const float expo,
              const int uType,
              const int nIter )
```

sbdivg_mic performs NNMF using betadivergence for general case (beta $<>$ 1 and 2) using simple precision

Parameters

m	(input) Number of rows of matrix A and matrix W
n	(input) Number of columns of matrix A and matrix H
k	(input) Number of columns of matrix W and rows of H
A	(input) Simple precision input matrix of (m $*$ n) number of elements stored using 1D column-major
W	(inout) Simple precision input/output matrix of (m $*$ k) number of elements stored using 1D column-major
H	(inout) Simple precision input/output matrix of (k $*$ n) number of elements stored using 1D column-major
expo	(input) Double precision value. The exponent beta of betadivergence method
uType	(input) It can be UpdateAll (W and H are updated), UpdateW (only H is updated) or UpdateH (only W is updated)
nIter	(input) Number of iterations

Returns

: 0 if all is OK, -1 otherwise

Definition at line 128 of file bdiv_mic.c.

2.6.2.7 sbdivone_mic()

```
int sbdivone_mic (
            const int m,
            const int n,
            const int k,
            const float * A,
            float * W,
            float * H,
            const int uType,
            const int nIter )
```

sbdivone_mic performs NNMF using betadivergence when beta=1 using simple precision

Parameters

m	(input) Number of rows of matrix A and matrix W
n	(input) Number of columns of matrix A and matrix H
k	(input) Number of columns of matrix W and rows of H
A	(input) Simple precision input matrix of (m $*$ n) number of elements stored using 1D column-major
W	(inout) Simple precision input/output matrix of (m $*$ k) number of elements stored using 1D column-major
H	(inout) Simple precision input/output matrix of (k $*$ n) number of elements stored using 1D column-major
uType	(input) It can be UpdateAll (W and H are updated), UpdateW (only H is updated) or UpdateH (only W is updated)
nIter	(input) Number of iterations

Returns

: 0 if all is OK, -1 otherwise

Definition at line 234 of file bdiv_mic.c.

2.6.2.8 smlsa_mic()

```
int smlsa_mic (
            const int m,
            const int n,
            const int k,
            const float * A,
```

```
float * W,
float * H,
const int uType,
const int nIter )
```

smlsa_mic performs NNMF using betadivergence when beta=2 using simple precision

Parameters

m	(input) Number of rows of matrix A and matrix W
n	(input) Number of columns of matrix A and matrix H
k	(input) Number of columns of matrix W and rows of H
A	(input) Simple precision input matrix of (m ∗ n) number of elements stored using 1D column-major
W	(inout) Simple precision input/output matrix of (m ∗ k) number of elements stored using 1D column-major
H	(inout) Simple precision input/output matrix of (k ∗ n) number of elements stored using 1D column-major
uType	(input) It can be UpdateAll (W and H are updated), UpdateW (only H is updated) or UpdateH (only W is updated)
nIter	(input) Number of iterations

Returns

: 0 if all is OK, -1 otherwise

Definition at line 111 of file mlsa_mic.c.

2.7 config.h File Reference

2.8 dbdiv1_cpu.c File Reference

Beta Divergence (bdiv) example of usage. Matrices A, W and H are randomly generated. Double precision is used. Layout 1D column-mayor.

Functions

- int main (int argc, char ∗argv[ ])

2.8.1 Detailed Description

Beta Divergence (bdiv) example of usage. Matrices A, W and H are randomly generated. Double precision is used. Layout 1D column-mayor.

Author

Information Retrieval and Parallel Computing Group (IRPCG)
University of Oviedo, Spain
Interdisciplinary Computation and Communication Group (INCO2)
Universitat Politecnica de Valencia, Spain.
Contact: nnmfpack@gmail.com

Date

04/11/14

2.8.2 Function Documentation

2.8.2.1 main()

```
int main (
              int argc,
              char * argv[] )
```

Definition at line 35 of file dbdiv1_cpu.c.

2.9 dbdiv1_cuda.cu File Reference

2.10 dbdiv1_mic.c File Reference

Beta Divergence (bdiv) example of usage. Matrices A, W and H are randomly generated. Double precision is used. Layout 1D column-mayor.

Functions

- int main (int argc, char ∗argv[ ])

2.10.1 Detailed Description

Beta Divergence (bdiv) example of usage. Matrices A, W and H are randomly generated. Double precision is used. Layout 1D column-mayor.

Author

Information Retrieval and Parallel Computing Group (IRPCG)
University of Oviedo, Spain
Interdisciplinary Computation and Communication Group (INCO2)
Universitat Politecnica de Valencia, Spain.
Contact: nnmfpack@gmail.com

Date

04/11/14

2.10.2 Function Documentation

2.10.2.1 main()

```
int main (
              int argc,
              char * argv[] )
```

Definition at line 35 of file dbdiv1_mic.c.

2.11 dbdiv2_cpu.c File Reference

Beta Divergence (bdiv) example of usage. It reads matrices A, W and H from files and writes updated matrices W and H to files. Double precision is used. Layout 1D column-mayor.

Functions

- int main (int argc, char ∗argv[ ])

2.11.1 Detailed Description

Beta Divergence (bdiv) example of usage. It reads matrices A, W and H from files and writes updated matrices W and H to files. Double precision is used. Layout 1D column-mayor.

Author

Information Retrieval and Parallel Computing Group (IRPCG)
University of Oviedo, Spain
Interdisciplinary Computation and Communication Group (INCO2)
Universitat Politecnica de Valencia, Spain.
Contact: nnmfpack@gmail.com

Date

04/11/14

2.11.2 Function Documentation

2.11.2.1 main()

```
int main (
              int argc,
              char * argv[] )
```

Definition at line 38 of file dbdiv2_cpu.c.

2.12 dbdiv2_cuda.cu File Reference

2.13 dbdiv2_mic.c File Reference

Beta Divergence (bdiv) example of usage. It reads matrices A, W and H from files and writes updated matrices W and H to files. Double precision is used. Layout 1D column-mayor.

Functions

- int main (int argc, char ∗argv[ ])

2.13.1 Detailed Description

Beta Divergence (bdiv) example of usage. It reads matrices A, W and H from files and writes updated matrices W and H to files. Double precision is used. Layout 1D column-mayor.

Author

Information Retrieval and Parallel Computing Group (IRPCG)
University of Oviedo, Spain
Interdisciplinary Computation and Communication Group (INCO2)
Universitat Politecnica de Valencia, Spain.
Contact: nnmfpack@gmail.com

Date

18/07/14

2.13.2 Function Documentation

2.13.2.1 main()

```
int main (
            int argc,
            char * argv[] )
```

Definition at line 38 of file dbdiv2_mic.c.

2.14 dgenerate.c File Reference

Auxiliar code to randomly generate matrices. Matrices are written to a binary file. Double precision is used. Layout: 1D Column mayor.

Functions

- int main (int argc, char ∗argv[ ])

2.14.1 Detailed Description

Auxiliar code to randomly generate matrices. Matrices are written to a binary file. Double precision is used. Layout: 1D Column mayor.

Author

Information Retrieval and Parallel Computing Group (IRPCG)
University of Oviedo, Spain
Interdisciplinary Computation and Communication Group (INCO2)
Universitat Politecnica de Valencia, Spain.
Contact: nnmfpack@gmail.com

Date

04/11/14

2.14.2 Function Documentation

2.14.2.1 main()

```
int main (
            int argc,
            char * argv[] )
```

Definition at line 44 of file dgenerate.c.

2.15 mex_bdivCpu.c File Reference

mex_bdivCpu is the nnmfpack's CPU-driver (nnmfpack's functions runs on CPU) for using it from from Matlab/↩ Octave. Layout 1D column-mayor.

Functions

- void mexFunction (int nlhs, mxArray ∗plhs[ ], int nrhs, const mxArray ∗prhs[ ])

  Standar MEX interface. Double precision.

2.15.1 Detailed Description

mex_bdivCpu is the nnmfpack's CPU-driver (nnmfpack's functions runs on CPU) for using it from from Matlab/↩ Octave. Layout 1D column-mayor.

Author

Information Retrieval and Parallel Computing Group (IRPCG)
University of Oviedo, Spain
Interdisciplinary Computation and Communication Group (INCO2)
Universitat Politecnica de Valencia, Spain.
Contact: nnmfpack@gmail.com

Date

04/11/14

2.15.2 Function Documentation

2.15.2.1 mexFunction()

```
void mexFunction (
             int nlhs,
             mxArray * plhs[],
             int nrhs,
             const mxArray * prhs[] )
```

Standar MEX interface. Double precision.

Parameters

nlhs	(output) Number of expected output positions of mxArrays with data
plhs	(output) Outpur parameters. They are: W and H matrices
nrhs	(input) Number of input positions of mxArray with data
prhs	(input) Input parameters. They are: A, W and H matrices, beta value, uType and iterations uType can be UpdateAll (W and H are updated), UpdateW (only H) or UpdateH (only W)

Definition at line 47 of file mex_bdivCpu.c.

2.16 mex_bdivCuda.cu File Reference

2.17 mex_bdivMic.c File Reference

mex_bdivMic is the nnmfpack's MIC-driver (nnmfpack's functions runs on Intel Xeon Phi) for using it from from Matlab/Octave. Layout 1D column-mayor.

Functions

- void mexFunction (int nlhs, mxArray ∗plhs[ ], int nrhs, const mxArray ∗prhs[ ])

2.17.1 Detailed Description

mex_bdivMic is the nnmfpack's MIC-driver (nnmfpack's functions runs on Intel Xeon Phi) for using it from from Matlab/Octave. Layout 1D column-mayor.

Author

Information Retrieval and Parallel Computing Group (IRPCG)
University of Oviedo, Spain
Interdisciplinary Computation and Communication Group (INCO2)
Universitat Politecnica de Valencia, Spain.
Contact: nnmfpack@gmail.com

Date

04/11/14

2.17.2 Function Documentation

2.17.2.1 mexFunction()

```
void mexFunction (
             int nlhs,
             mxArray * plhs[],
             int nrhs,
             const mxArray * prhs[] )
```

Definition at line 47 of file mex_bdivMic.c.

2.18 mlsa_cpu.c File Reference

Functions

- int dmlsa_cpu (const int m, const int n, const int k, const double ∗A, double ∗W, double ∗H, const int uType, const int nIter)

  dmlsa_cpu performs NNMF using betadivergence when beta=2 using double precision
- int smlsa_cpu (const int m, const int n, const int k, const float ∗A, float ∗W, float ∗H, const int uType, const int nIter)

  smlsa_cpu performs NNMF using betadivergence when beta=2 using simple precision
- void ddotdiv_x86 (const int n, const double ∗x, const double ∗y, double ∗__restrict__ z)

  This function calls the appropiate funtions to performs double precision element-wise z[i]=z[i]∗x[i]/y[i] for all positions of x, y and z.
- void sdotdiv_x86 (const int n, const float ∗x, const float ∗y, float ∗__restrict__ z)

  This function calls the appropiate funtions to performs simple precision element-wise z[i]=z[i]∗x[i]/y[i] for all positions of x, y and z.

2.18.1 Function Documentation

2.18.1.1 ddotdiv_x86()

```
void ddotdiv_x86 (
             const int n,
             const double * x,
             const double * y,
             double *__restrict__ z )
```

This function calls the appropiate funtions to performs double precision element-wise z[i]=z[i]∗x[i]/y[i] for all positions of x, y and z.

Parameters

n	(input) Number of elements of x, y and z
x	(input) Double precision input vector/matrix (1D column-major)
y	(input) Double precision input vector/matrix (1D column-major)
z	(inout) Double precision input/output vector/matrix (1D column-major)

Definition at line 471 of file mlsa_cpu.c.

2.18.1.2 dmlsa_cpu()

```
int dmlsa_cpu (
             const int m,
             const int n,
             const int k,
             const double * A,
             double * W,
             double * H,
             const int uType,
             const int nIter )
```

dmlsa_cpu performs NNMF using betadivergence when beta=2 using double precision

The algorithm is
repit nIter times
STEP 1
L=W∗H
B=W'*A*
C=W'∗L
H=H(.)B(./)C

STEP 2
L=W∗H
D=A∗H'

E=L∗H'
W=W(.∗)D(./)E
end repit
End algorithm

To save some FLOPs and RAM a modified Lee and Seung version is used, so we have Step 1: L=W'∗W, C=L∗H and then B=W'∗A; Step 2: L=H∗H', E=W∗L and D=A∗H'. W'∗W (H∗H') and L∗H (W∗L) can do in parallel with W'∗A (A∗H'). Because dgemm works fine we don't use threads to do in parallel.

In real live B and C are (k∗n) matrices used in STEP 1, and D and E are (m∗k) matrices used in STEP 2. B/C and D/E are independent. For this reason only 2 matrices are declared to save space. They are matrices B and C with size max(m,n)∗k

Parameters

m	(input) Number of rows of matrix A and matrix W
n	(input) Number of columns of matrix A and matrix H
k	(input) Number of columns of matrix W and rows of H
A	(input) Double precision input matrix of (m ∗ n) number of elements stored using 1D column-major
W	(inout) Double precision input/output matrix of (m ∗ k) number of elements stored using 1D column-major
H	(inout) Double precision input/output matrix of (k ∗ n) number of elements stored using 1D column-major
uType	(input) It can be UpdateAll (W and H are updated), UpdateW (only H is updated) or UpdateH (only W is updated)
nIter	(input) Number of iterations

Returns

: 0 if all is OK, -1 otherwise

Definition at line 74 of file mlsa_cpu.c.

2.18.1.3 sdotdiv_x86()

```
void sdotdiv_x86 (
             const int n,
             const float * x,
             const float * y,
             float *__restrict__ z )
```

This function calls the appropiate funtions to performs simple precision element-wise z[i]=z[i]∗x[i]/y[i] for all positions of x, y and z.

Parameters

n	(input) Number of elements of x, y and z
x	(input) Simple precision input vector/matrix (1D column-major)
y	(input) Simple precision input vector/matrix (1D column-major)
z	(inout) Simple precision input/output vector/matrix (1D column-major)

Definition at line 509 of file mlsa_cpu.c.

2.18.1.4 smlsa_cpu()

```
int smlsa_cpu (
            const int m,
            const int n,
            const int k,
            const float * A,
            float * W,
            float * H,
            const int uType,
            const int nIter )
```

smlsa_cpu performs NNMF using betadivergence when beta=2 using simple precision

Parameters

m	(input) Number of rows of matrix A and matrix W
n	(input) Number of columns of matrix A and matrix H
k	(input) Number of columns of matrix W and rows of H
A	(input) Simple precision input matrix of (m ∗ n) number of elements stored using 1D column-major
W	(inout) Simple precision input/output matrix of (m ∗ k) number of elements stored using 1D column-major
H	(inout) Simple precision input/output matrix of (k ∗ n) number of elements stored using 1D column-major
uType	(input) It can be UpdateAll (W and H are updated), UpdateW (only H is updated) or UpdateH (only W is updated)
nIter	(input) Number of iterations

Returns

: 0 if all is OK, -1 otherwise

Definition at line 275 of file mlsa_cpu.c.

2.19 mlsa_cpu.h File Reference

File with functions to calcule NNMF using the mlsa algorithm for CPUs.

Functions

- int dmlsa_cpu (const int m, const int n, const int k, const double ∗A, double ∗W, double ∗H, const int uType, const int nIter)

  dmlsa_cpu performs NNMF using betadivergence when beta=2 using double precision

- int smlsa_cpu (const int m, const int n, const int k, const float *A, float *W, float *H, const int uType, const int nIter)

  smlsa_cpu performs NNMF using betadivergence when beta=2 using simple precision
- void ddotdiv_x86 (const int n, const double *x, const double *y, double *__restrict__ z)

  *This function calls the appropiate funtions to performs double precision element-wise z[i]=z[i]*x[i]/y[i] for all positions of x, y and z.*
- void sdotdiv_x86 (const int n, const float *x, const float *y, float *__restrict__ z)

  *This function calls the appropiate funtions to performs simple precision element-wise z[i]=z[i]*x[i]/y[i] for all positions of x, y and z.*

2.19.1 Detailed Description

File with functions to calcule NNMF using the mlsa algorithm for CPUs.

Header file for using the mlsa algorithm with CPU.

Author

Information Retrieval and Parallel Computing Group (IRPCG)
University of Oviedo, Spain
Interdisciplinary Computation and Communication Group (INCO2)
Universitat Politecnica de Valencia, Spain.
Contact: nnmfpack@gmail.com

Date

12/03/18

2.19.2 Function Documentation

2.19.2.1 ddotdiv_x86()

```
void ddotdiv_x86 (
            const int n,
            const double * x,
            const double * y,
            double *__restrict__ z )
```

This function calls the appropiate funtions to performs double precision element-wise z[i]=z[i]*x[i]/y[i] for all positions of x, y and z.

Parameters

n	(input) Number of elements of x, y and z
x	(input) Double precision input vector/matrix (1D column-major)
y	(input) Double precision input vector/matrix (1D column-major)
z	(inout) Double precision input/output vector/matrix (1D column-major)

Definition at line 471 of file mlsa_cpu.c.

2.19.2.2 dmlsa_cpu()

```
int dmlsa_cpu (
            const int m,
            const int n,
            const int k,
            const double * A,
            double * W,
            double * H,
            const int uType,
            const int nIter )
```

dmlsa_cpu performs NNMF using betadivergence when beta=2 using double precision

The algorithm is
repit nIter times
STEP 1
L=W∗H
B=W'*A*
C=W'∗L
H=H(.)B(./)C

STEP 2
L=W∗H
D=A∗H'
E=L∗H'
W=W(.∗)D(./)E
end repit
End algorithm

To save some FLOPs and RAM a modified Lee and Seung version is used, so we have Step 1: L=W'∗W, C=L∗H and then B=W'∗A; Step 2: L=H∗H', E=W∗L and D=A∗H'. W'∗W (H∗H') and L∗H (W∗L) can do in parallel with W'∗A (A∗H'). Because dgemm works fine we don't use threads to do in parallel.

In real live B and C are (k∗n) matrices used in STEP 1, and D and E are (m∗k) matrices used in STEP 2. B/C and D/E are independent. For this reason only 2 matrices are declared to save space. They are matrices B and C with size max(m,n)∗k

Parameters

m	(input) Number of rows of matrix A and matrix W
n	(input) Number of columns of matrix A and matrix H
k	(input) Number of columns of matrix W and rows of H
A	(input) Double precision input matrix of (m ∗ n) number of elements stored using 1D column-major
W	(inout) Double precision input/output matrix of (m ∗ k) number of elements stored using 1D column-major
H	(inout) Double precision input/output matrix of (k ∗ n) number of elements stored using 1D column-major
uType	(input) It can be UpdateAll (W and H are updated), UpdateW (only H is updated) or UpdateH (only W is updated)
nIter	(input) Number of iterations

Returns

: 0 if all is OK, -1 otherwise

Definition at line 74 of file mlsa_cpu.c.

2.19.2.3 sdotdiv_x86()

```
void sdotdiv_x86 (
            const int n,
            const float * x,
            const float * y,
            float *__restrict__ z )
```

This function calls the appropiate funtions to performs simple precision element-wise z[i]=z[i]∗x[i]/y[i] for all positions of x, y and z.

Parameters

n	(input) Number of elements of x, y and z
x	(input) Simple precision input vector/matrix (1D column-major)
y	(input) Simple precision input vector/matrix (1D column-major)
z	(inout) Simple precision input/output vector/matrix (1D column-major)

Definition at line 509 of file mlsa_cpu.c.

2.19.2.4 smlsa_cpu()

```
int smlsa_cpu (
            const int m,
            const int n,
            const int k,
            const float * A,
            float * W,
            float * H,
            const int uType,
            const int nIter )
```

smlsa_cpu performs NNMF using betadivergence when beta=2 using simple precision

Parameters

m	(input) Number of rows of matrix A and matrix W
n	(input) Number of columns of matrix A and matrix H
k	(input) Number of columns of matrix W and rows of H
A	(input) Simple precision input matrix of (m ∗ n) number of elements stored using 1D column-major
W	(inout) Simple precision input/output matrix of (m ∗ k) number of elements stored using 1D column-major

Parameters

H	(inout) Simple precision input/output matrix of (k ∗ n) number of elements stored using 1D column-major
uType	(input) It can be UpdateAll (W and H are updated), UpdateW (only H is updated) or UpdateH (only W is updated)
nIter	(input) Number of iterations

Returns

: 0 if all is OK, -1 otherwise

Definition at line 275 of file mlsa_cpu.c.

2.20 mlsa_cuda.cu File Reference

2.21 mlsa_cuda.h File Reference

Header file for using the mlsa algorithm using cuda functions with GPUs.

Functions

- int dmlsa_cuda (const int m, const int n, const int k, const double ∗A, double ∗W, double ∗H, const int uType, const int nIter)
- int smlsa_cuda (const int m, const int n, const int k, const float ∗A, float ∗W, float ∗H, const int uType, const int nIter)
- void ddotdiv_cuda (const int n, const double ∗x, const double ∗y, double ∗z, cudaStream_t stream)
- void sdotdiv_cuda (const int n, const float ∗x, const float ∗y, float ∗z, cudaStream_t stream)
- __global__ void vddotdiv_cuda (const int n, const double ∗__restrict__ x, const double ∗__restrict__ y, double ∗z)
- __global__ void vsdotdiv_cuda (const int n, const float ∗__restrict__ x, const float ∗__restrict__ y, float ∗z)

2.21.1 Detailed Description

Header file for using the mlsa algorithm using cuda functions with GPUs.

Author

Information Retrieval and Parallel Computing Group (IRPCG)
University of Oviedo, Spain
Interdisciplinary Computation and Communication Group (INCO2)
Universitat Politecnica de Valencia, Spain.
Contact: nnmfpack@gmail.com

Date

04/11/14

2.21.2 Function Documentation

2.21.2.1 ddotdiv_cuda()

```
void ddotdiv_cuda (
            const int n,
            const double * x,
            const double * y,
            double * z,
            cudaStream_t stream )
```

2.21.2.2 dmlsa_cuda()

```
int dmlsa_cuda (
            const int m,
            const int n,
            const int k,
            const double * A,
            double * W,
            double * H,
            const int uType,
            const int nIter )
```

2.21.2.3 sdotdiv_cuda()

```
void sdotdiv_cuda (
            const int n,
            const float * x,
            const float * y,
            float * z,
            cudaStream_t stream )
```

2.21.2.4 smlsa_cuda()

```
int smlsa_cuda (
            const int m,
            const int n,
            const int k,
            const float * A,
            float * W,
            float * H,
            const int uType,
            const int nIter )
```

2.21.2.5 vddotdiv_cuda()

```
__global__ void vddotdiv_cuda (
              const int n,
              const double *__restrict__ x,
              const double *__restrict__ y,
              double * z )
```

2.21.2.6 vsdotdiv_cuda()

```
__global__ void vsdotdiv_cuda (
              const int n,
              const float *__restrict__ x,
              const float *__restrict__ y,
              float * z )
```

2.22 mlsa_mic.c File Reference

File with functions to calcule NNMF using mlsa algorithm with Intel Xeon Phi (MIC)

Functions

- int dmlsa_mic (const int m, const int n, const int k, const double *A, double *W, double *H, const int uType, const int nIter)

  dmlsa_mic performs NNMF using betadivergence when beta=2 using double precision
- int smlsa_mic (const int m, const int n, const int k, const float *A, float *W, float *H, const int uType, const int nIter)

  smlsa_mic performs NNMF using betadivergence when beta=2 using simple precision

2.22.1 Detailed Description

File with functions to calcule NNMF using mlsa algorithm with Intel Xeon Phi (MIC)

Author

Information Retrieval and Parallel Computing Group (IRPCG)
University of Oviedo, Spain
Interdisciplinary Computation and Communication Group (INCO2)
Universitat Politecnica de Valencia, Spain.
Contact: nnmfpack@gmail.com

Date

12/03/18

2.22.2 Function Documentation

2.22.2.1 dmlsa_mic()

```
int dmlsa_mic (
            const int m,
            const int n,
            const int k,
            const double * A,
            double * W,
            double * H,
            const int uType,
            const int nIter )
```

dmlsa_mic performs NNMF using betadivergence when beta=2 using double precision

The algorithm is
repit nIter times
STEP 1
L=W∗H
B=W'*A*
C=W'∗L
H=H(.)B(./)C

STEP 2
L=W∗H
D=A∗H'
E=L∗H'
W=W(.∗)D(./)E
end repit
End algorithm

To save some FLOPs and RAM a modified Lee and Seung version is used, so we have Step 1: L=W'∗W, C=L∗H and then B=W'∗A; Step 2: L=H∗H', E=W∗L and D=A∗H'. W'∗W (H∗H') and L∗H (W∗L) can do in parallel with W'∗A (A∗H'). Because dgemm works fine we don't use threads to do in parallel.

In real live B and C are (k∗n) matrices used in STEP 1, and D and E are (m∗k) matrices used in STEP 2. B/C and D/E are independent. For this reason only 2 matrices are declared to save space. They are matrices B and C with size max(m,n)∗k

Parameters

m	(input) Number of rows of matrix A and matrix W
n	(input) Number of columns of matrix A and matrix H
k	(input) Number of columns of matrix W and rows of H
A	(input) Double precision input matrix of (m ∗ n) number of elements stored using 1D column-major
W	(inout) Double precision input/output matrix of (m ∗ k) number of elements stored using 1D column-major
H	(inout) Double precision input/output matrix of (k ∗ n) number of elements stored using 1D column-major
uType	(input) It can be UpdateAll (W and H are updated), UpdateW (only H is updated) or UpdateH (only W is updated)
nIter	(input) Number of iterations

Returns

: 0 if all is OK, -1 otherwise

Definition at line 72 of file mlsa_mic.c.

2.22.2.2 smlsa_mic()

```
int smlsa_mic (
            const int m,
            const int n,
            const int k,
            const float * A,
            float * W,
            float * H,
            const int uType,
            const int nIter )
```

smlsa_mic performs NNMF using betadivergence when beta=2 using simple precision

Parameters

m	(input) Number of rows of matrix A and matrix W
n	(input) Number of columns of matrix A and matrix H
k	(input) Number of columns of matrix W and rows of H
A	(input) Simple precision input matrix of (m $*$ n) number of elements stored using 1D column-major
W	(inout) Simple precision input/output matrix of (m $*$ k) number of elements stored using 1D column-major
H	(inout) Simple precision input/output matrix of (k $*$ n) number of elements stored using 1D column-major
uType	(input) It can be UpdateAll (W and H are updated), UpdateW (only H is updated) or UpdateH (only W is updated)
nIter	(input) Number of iterations

Returns

: 0 if all is OK, -1 otherwise

Definition at line 111 of file mlsa_mic.c.

2.23 mlsa_mic.h File Reference

Header file for using the mlsa algorithm with MIC/MIC.

Functions

- int dmlsa_mic (const int m, const int n, const int k, const double ∗A, double ∗W, double ∗H, const int uType, const int nIter)

  dmlsa_mic performs NNMF using betadivergence when beta=2 using double precision
- int smlsa_mic (const int m, const int n, const int k, const float ∗A, float ∗W, float ∗H, const int uType, const int nIter)

  smlsa_mic performs NNMF using betadivergence when beta=2 using simple precision

2.23.1 Detailed Description

Header file for using the mlsa algorithm with MIC/MIC.

Author

Information Retrieval and Parallel Computing Group (IRPCG)
University of Oviedo, Spain
Interdisciplinary Computation and Communication Group (INCO2)
Universitat Politecnica de Valencia, Spain.
Contact: nnmfpack@gmail.com

Date

12/03/18

2.23.2 Function Documentation

2.23.2.1 dmlsa_mic()

```
int dmlsa_mic (
            const int m,
            const int n,
            const int k,
            const double * A,
            double * W,
            double * H,
            const int uType,
            const int nIter )
```

dmlsa_mic performs NNMF using betadivergence when beta=2 using double precision

The algorithm is
repit nIter times
STEP 1
L=W∗H
B=W'*A*
C=W'∗L
*H=H(.)*B(./)C

STEP 2
L=W∗H
D=A∗H'
E=L∗H'
W=W(.∗)D(./)E
end repit
End algorithm

To save some FLOPs and RAM a modified Lee and Seung version is used, so we have Step 1: L=W'∗W, C=L∗H and then B=W'∗A; Step 2: L=H∗H', E=W∗L and D=A∗H'. W'∗W (H∗H') and L∗H (W∗L) can do in parallel with W'∗A (A∗H'). Because dgemm works fine we don't use threads to do in parallel.

In real live B and C are (k∗n) matrices used in STEP 1, and D and E are (m∗k) matrices used in STEP 2. B/C and D/E are independent. For this reason only 2 matrices are declared to save space. They are matrices B and C with size max(m,n)∗k

Parameters

m	(input) Number of rows of matrix A and matrix W
n	(input) Number of columns of matrix A and matrix H
k	(input) Number of columns of matrix W and rows of H
A	(input) Double precision input matrix of (m ∗ n) number of elements stored using 1D column-major
W	(inout) Double precision input/output matrix of (m ∗ k) number of elements stored using 1D column-major
H	(inout) Double precision input/output matrix of (k ∗ n) number of elements stored using 1D column-major
uType	(input) It can be UpdateAll (W and H are updated), UpdateW (only H is updated) or UpdateH (only W is updated)
nIter	(input) Number of iterations

Returns

: 0 if all is OK, -1 otherwise

Definition at line 72 of file mlsa_mic.c.

2.23.2.2 smlsa_mic()

```
int smlsa_mic (
             const int m,
             const int n,
             const int k,
             const float * A,
             float * W,
             float * H,
             const int uType,
             const int nIter )
```

smlsa_mic performs NNMF using betadivergence when beta=2 using simple precision

Parameters

m	(input) Number of rows of matrix A and matrix W
n	(input) Number of columns of matrix A and matrix H
k	(input) Number of columns of matrix W and rows of H
A	(input) Simple precision input matrix of (m $*$ n) number of elements stored using 1D column-major
W	(inout) Simple precision input/output matrix of (m $*$ k) number of elements stored using 1D column-major
H	(inout) Simple precision input/output matrix of (k $*$ n) number of elements stored using 1D column-major
uType	(input) It can be UpdateAll (W and H are updated), UpdateW (only H is updated) or UpdateH (only W is updated)
nIter	(input) Number of iterations

Returns

: 0 if all is OK, -1 otherwise

Definition at line 111 of file mlsa_mic.c.

2.24 nnmfpackCpu.h File Reference

General header file for using NnmfPack with CPUs.

2.24.1 Detailed Description

General header file for using NnmfPack with CPUs.

Author

Information Retrieval and Parallel Computing Group (IRPCG)
University of Oviedo, Spain
Interdisciplinary Computation and Communication Group (INCO2)
Universitat Politecnica de Valencia, Spain.
Contact: nnmfpack@gmail.com

Date

04/11/14

2.25 nnmfpackCuda.h File Reference

General header file for using NnmfPack with CUDA/GPUs.

2.25.1 Detailed Description

General header file for using NnmfPack with CUDA/GPUs.

Author

Information Retrieval and Parallel Computing Group (IRPCG)
University of Oviedo, Spain
Interdisciplinary Computation and Communication Group (INCO2)
Universitat Politecnica de Valencia, Spain.
Contact: nnmfpack@gmail.com

Date

04/11/14

2.26 nnmfpackMic.h File Reference

General header file for using NnmfPack with Intel Xeon Phi (MIC).

2.26.1 Detailed Description

General header file for using NnmfPack with Intel Xeon Phi (MIC).

Author

Information Retrieval and Parallel Computing Group (IRPCG)
University of Oviedo, Spain
Interdisciplinary Computation and Communication Group (INCO2)
Universitat Politecnica de Valencia, Spain.
Contact: nnmfpack@gmail.com

Date

04/11/14

2.27 sbdiv1_cpu.c File Reference

Beta Divergence (bdiv) example of usage. Matrices A, W and H are randomly generated. Simple precision is used. Layout 1D column-mayor.

Functions

- int main (int argc, char *argv[ ])

2.27.1 Detailed Description

Beta Divergence (bdiv) example of usage. Matrices A, W and H are randomly generated. Simple precision is used. Layout 1D column-mayor.

Author

Information Retrieval and Parallel Computing Group (IRPCG)
University of Oviedo, Spain
Interdisciplinary Computation and Communication Group (INCO2)
Universitat Politecnica de Valencia, Spain.
Contact: nnmfpack@gmail.com

Date

04/11/14

2.27.2 Function Documentation

2.27.2.1 main()

```
int main (
              int argc,
              char * argv[] )
```

Definition at line 35 of file sbdiv1_cpu.c.

2.28 sbdiv1_cuda.cu File Reference

2.29 sbdiv1_mic.c File Reference

Beta Divergence (bdiv) example of usage. Matrices A, W and H are randomly generated. Simple precision is used. Layout 1D column-mayor.

Functions

- int main (int argc, char ∗argv[ ])

2.29.1 Detailed Description

Beta Divergence (bdiv) example of usage. Matrices A, W and H are randomly generated. Simple precision is used. Layout 1D column-mayor.

Author

Information Retrieval and Parallel Computing Group (IRPCG)
University of Oviedo, Spain
Interdisciplinary Computation and Communication Group (INCO2)
Universitat Politecnica de Valencia, Spain.
Contact: nnmfpack@gmail.com

Date

04/11/14

2.29.2 Function Documentation

2.29.2.1 main()

```
int main (
            int argc,
            char * argv[] )
```

Definition at line 35 of file sbdiv1_mic.c.

2.30 sbdiv2_cpu.c File Reference

Beta Divergence (bdiv) example of usage. It reads matrices A, W and H from files and writes updated matrices W and H to files. Simple precision is used. Layout 1D column-mayor.

Functions

- int main (int argc, char ∗argv[ ])

2.30.1 Detailed Description

Beta Divergence (bdiv) example of usage. It reads matrices A, W and H from files and writes updated matrices W and H to files. Simple precision is used. Layout 1D column-mayor.

Author

Information Retrieval and Parallel Computing Group (IRPCG)
University of Oviedo, Spain
Interdisciplinary Computation and Communication Group (INCO2)
Universitat Politecnica de Valencia, Spain.
Contact: nnmfpack@gmail.com

Date

04/11/14

2.30.2 Function Documentation

2.30.2.1 main()

```
int main (
            int argc,
            char * argv[] )
```

Definition at line 38 of file sbdiv2_cpu.c.

2.31 sbdiv2_cuda.cu File Reference

2.32 sbdiv2_mic.c File Reference

Beta Divergence (bdiv) example of usage. It reads matrices A, W and H from files and writes updated matrices W and H to files. Simple precision is used. Layout 1D column-mayor.

Functions

- int main (int argc, char ∗argv[ ])

2.32.1 Detailed Description

Beta Divergence (bdiv) example of usage. It reads matrices A, W and H from files and writes updated matrices W and H to files. Simple precision is used. Layout 1D column-mayor.

Author

Information Retrieval and Parallel Computing Group (IRPCG)
University of Oviedo, Spain
Interdisciplinary Computation and Communication Group (INCO2)
Universitat Politecnica de Valencia, Spain.
Contact: nnmfpack@gmail.com

Date

18/07/14

2.32.2 Function Documentation

2.32.2.1 main()

```
int main (
              int argc,
              char * argv[] )
```

Definition at line 38 of file sbdiv2_mic.c.

2.33 sgenerate.c File Reference

Auxiliar code to randomly generate matrices. Matrices are written to a binary file. Simple precision is used. Layout: 1D Column mayor.

Functions

- int main (int argc, char ∗argv[ ])

2.33.1 Detailed Description

Auxiliar code to randomly generate matrices. Matrices are written to a binary file. Simple precision is used. Layout: 1D Column mayor.

Author

Information Retrieval and Parallel Computing Group (IRPCG)
University of Oviedo, Spain
Interdisciplinary Computation and Communication Group (INCO2)
Universitat Politecnica de Valencia, Spain.
Contact: nnmfpack@gmail.com

Date

04/11/14

2.33.2 Function Documentation

2.33.2.1 main()

```
int main (
              int argc,
              char * argv[] )
```

Definition at line 45 of file sgenerate.c.

2.34 utils.c File Reference

Some transversal auxiliar functions. Double and simple precision.

Functions

- int read_file_header (char *path, int *rows, int *cols)

  read_file_header fills rows & cols with the dimensions of a matrix stored in the binary file. It returns 0 if not problems were found.
- int dread_file (char *path, int rows, int cols, double *M)

  dread_file fills the double precision matix M stored in the binary file. It returns 0 if not problems were found.
- int sread_file (char *path, int rows, int cols, float *M)

  sread_file fills the simple precision matix M stored in the binary file. It returns 0 if not problems were found.
- int dwrite_file (char *path, int *rows, int *cols, double *M)

  dwrite_file stores the double precision matix M in the binary file. It returns 0 if not problems were found.
- int swrite_file (char *path, int *rows, int *cols, float *M)

  swrite_file stores the simple precision matix M in the binary file. It returns 0 if not problems were found.
- int dwrite_ascii_file (char *path, int rows, int cols, double *M)

  dwrite_ascii_file stores the double precision matix M in the text file. It returns 0 if not problems were found.
- int swrite_ascii_file (char *path, int rows, int cols, float *M)

  swrite_ascii_file stores the simple precision matix M in the text file. It returns 0 if not problems were found.

2.34.1 Detailed Description

Some transversal auxiliar functions. Double and simple precision.

Author

Information Retrieval and Parallel Computing Group (IRPCG)
University of Oviedo, Spain
Interdisciplinary Computation and Communication Group (INCO2)
Universitat Politecnica de Valencia, Spain.
Contact: nnmfpack@gmail.com

Date

04/11/14

2.34.2 Function Documentation

2.34.2.1 dread_file()

```
int dread_file (
            char * path,
            int rows,
            int cols,
            double * M )
```

dread_file fills the double precision matix M stored in the binary file. It returns 0 if not problems were found.

Parameters

path	(input) Path to the file
rows	(input) Number of rows of the matrix M
cols	(input) Number of columns of the matrix M
M	(output) Double precision matrix M stored as 1D layout

Definition at line 69 of file utils.c.

2.34.2.2 dwrite_ascii_file()

```
int dwrite_ascii_file (
            char * path,
            int rows,
            int cols,
            double * M )
```

dwrite_ascii_file stores the double precision matix M in the text file. It returns 0 if not problems were found.

Parameters

path	(input) Path to the file
rows	(input) Number of rows of the matrix M
cols	(input) Number of columns of the matrix M
M	(input) Simple precision matrix M stored as 1D layout

Definition at line 191 of file utils.c.

2.34.2.3 dwrite_file()

```
int dwrite_file (
            char * path,
            int * rows,
            int * cols,
            double * M )
```

dwrite_file stores the double precision matix M in the binary file. It returns 0 if not problems were found.

Parameters

path	(input) Path to the file
rows	(input) Number of rows of the matrix M
cols	(input) Number of columns of the matrix M
M	(input) Double precision matrix M stored as 1D layout

Definition at line 141 of file utils.c.

2.34.2.4 read_file_header()

```
int read_file_header (
             char * path,
             int * rows,
             int * cols )
```

read_file_header fills rows & cols with the dimensions of a matrix stored in the binary file. It returns 0 if not problems were found.

Parameters

path	(input) Path to the file
rows	(output) Number of rows of the stored matrix in the file
cols	(output) Number of columns of the stored matrix in the file

Definition at line 45 of file utils.c.

2.34.2.5 sread_file()

```
int sread_file (
             char * path,
             int rows,
             int cols,
             float * M )
```

sread_file fills the simple precision matix M stored in the binary file. It returns 0 if not problems were found.

Parameters

path	(input) Path to the file
rows	(input) Number of rows of the matrix M
cols	(input) Number of columns of the matrix M
M	(output) Simple precision matrix M stored as 1D layout

Definition at line 105 of file utils.c.

2.34.2.6 swrite_ascii_file()

```
int swrite_ascii_file (
             char * path,
```

```
            int rows,
            int cols,
            float * M )
```

swrite_ascii_file stores the simple precision matix M in the text file. It returns 0 if not problems were found.

Parameters

path	(input) Path to the file
rows	(input) Number of rows of the matrix M
cols	(input) Number of columns of the matrix M
M	(input) Simple precision matrix M stored as 1D lahout

Definition at line 218 of file utils.c.

2.34.2.7 swrite_file()

```
int swrite_file (
            char * path,
            int * rows,
            int * cols,
            float * M )
```

swrite_file stores the simple precision matix M in the binary file. It returns 0 if not problems were found.

Parameters

path	(input) Path to the file
rows	(input) Number of rows of the matrix M
cols	(input) Number of columns of the matrix M
M	(input) Simple precision matrix M stored as 1D layout

Definition at line 166 of file utils.c.

2.35 utils.h File Reference

Header file for transversal auxiliar functions. Double and simple precision.

Functions

- int read_file_header (char ∗, int ∗, int ∗)

  read_file_header fills rows & cols with the dimensions of a matrix stored in the binary file. It returns 0 if not problems were found.
- int dread_file (char ∗, int, int, double ∗)

  dread_file fills the double precision matix M stored in the binary file. It returns 0 if not problems were found.

- int sread_file (char *, int, int, float *)

  sread_file fills the simple precision matix M stored in the binary file. It returns 0 if not problems were found.
- int dwrite_file (char *, int *, int *, double *)

  dwrite_file stores the double precision matix M in the binary file. It returns 0 if not problems were found.
- int swrite_file (char *, int *, int *, float *)

  swrite_file stores the simple precision matix M in the binary file. It returns 0 if not problems were found.
- int dwrite_ascii_file (char *, int, int, double *)

  dwrite_ascii_file stores the double precision matix M in the text file. It returns 0 if not problems were found.
- int swrite_ascii_file (char *, int, int, float *)

  swrite_ascii_file stores the simple precision matix M in the text file. It returns 0 if not problems were found.

2.35.1 Detailed Description

Header file for transversal auxiliar functions. Double and simple precision.

Author

Information Retrieval and Parallel Computing Group (IRPCG)
University of Oviedo, Spain
Interdisciplinary Computation and Communication Group (INCO2)
Universitat Politecnica de Valencia, Spain.
Contact: nnmfpack@gmail.com

Date

04/11/14

2.35.2 Function Documentation

2.35.2.1 dread_file()

```
int dread_file (
            char * path,
            int rows,
            int cols,
            double * M )
```

dread_file fills the double precision matix M stored in the binary file. It returns 0 if not problems were found.

Parameters

path	(input) Path to the file
rows	(input) Number of rows of the matrix M
cols	(input) Number of columns of the matrix M
M	(output) Double precision matrix M stored as 1D layout

Definition at line 69 of file utils.c.

2.35.2.2 dwrite_ascii_file()

```
int dwrite_ascii_file (
            char * path,
            int rows,
            int cols,
            double * M )
```

dwrite_ascii_file stores the double precision matix M in the text file. It returns 0 if not problems were found.

Parameters

path	(input) Path to the file
rows	(input) Number of rows of the matrix M
cols	(input) Number of columns of the matrix M
M	(input) Simple precision matrix M stored as 1D layout

Definition at line 191 of file utils.c.

2.35.2.3 dwrite_file()

```
int dwrite_file (
            char * path,
            int * rows,
            int * cols,
            double * M )
```

dwrite_file stores the double precision matix M in the binary file. It returns 0 if not problems were found.

Parameters

path	(input) Path to the file
rows	(input) Number of rows of the matrix M
cols	(input) Number of columns of the matrix M
M	(input) Double precision matrix M stored as 1D layout

Definition at line 141 of file utils.c.

2.35.2.4 read_file_header()

```
int read_file_header (
            char * path,
```

```
            int * rows,
            int * cols )
```

read_file_header fills rows & cols with the dimensions of a matrix stored in the binary file. It returns 0 if not problems were found.

Parameters

path	(input) Path to the file
rows	(output) Number of rows of the stored matrix in the file
cols	(output) Number of columns of the stored matrix in the file

Definition at line 45 of file utils.c.

2.35.2.5 sread_file()

```
int sread_file (
            char * path,
            int rows,
            int cols,
            float * M )
```

sread_file fills the simple precision matix M stored in the binary file. It returns 0 if not problems were found.

Parameters

path	(input) Path to the file
rows	(input) Number of rows of the matrix M
cols	(input) Number of columns of the matrix M
M	(output) Simple precision matrix M stored as 1D layout

Definition at line 105 of file utils.c.

2.35.2.6 swrite_ascii_file()

```
int swrite_ascii_file (
            char * path,
            int rows,
            int cols,
            float * M )
```

swrite_ascii_file stores the simple precision matix M in the text file. It returns 0 if not problems were found.

Parameters

path	(input) Path to the file
rows	(input) Number of rows of the matrix M
cols	(input) Number of columns of the matrix M
M	(input) Simple precision matrix M stored as 1D lahout

Definition at line 218 of file utils.c.

2.35.2.7 swrite_file()

```
int swrite_file (
            char * path,
            int * rows,
            int * cols,
            float * M )
```

swrite_file stores the simple precision matix M in the binary file. It returns 0 if not problems were found.

Parameters

path	(input) Path to the file
rows	(input) Number of rows of the matrix M
cols	(input) Number of columns of the matrix M
M	(input) Simple precision matrix M stored as 1D layout

Definition at line 166 of file utils.c.

2.36 utils_cuda.cu File Reference

2.37 utils_cuda.h File Reference

Header file for using utility modules from CUDA source codes.

Functions

- __global__ void vdmemset_cuda (const int n, double *x, const double val)
- __global__ void vsmemset_cuda (const int n, float *x, const float val)
- __global__ void vddiv_cuda (const int n, const double *__restrict__ x, const double *__restrict__ y, double *z)
- __global__ void vsdiv_cuda (const int n, const float *__restrict__ x, const float *__restrict__ y, float *z)
- __global__ void vdsub_cuda (const int n, const double *__restrict__ x, double *y)
- __global__ void vssub_cuda (const int n, const float *__restrict__ x, float *y)
- __global__ void vderrorbd0_cuda (const int n, const double *__restrict__ x, double *y)
- __global__ void vserrorbd0_cuda (const int n, const float *__restrict__ x, float *y)
- __global__ void vderrorbd1_cuda (const int n, const double *__restrict__ x, double *y)
- __global__ void vserrorbd1_cuda (const int n, const float *__restrict__ x, float *y)
- __global__ void vderrorbdg_cuda (const int n, const double *__restrict__ x, double *y, const double beta)
- __global__ void vserrorbdg_cuda (const int n, const float *__restrict__ x, float *y, const double beta)
- void dmemset_cuda (const int n, double *x, const double val, cudaStream_t stream)
- void smemset_cuda (const int n, float *x, const float val, cudaStream_t stream)
- void ddiv_cuda (const int n, const double *x, double *y, cudaStream_t stream)
- void sdiv_cuda (const int n, const float *x, float *y, cudaStream_t stream)

- void dsub_cuda (const int n, const double ∗x, double ∗y)
- void ssub_cuda (const int n, const float ∗x, float ∗y)
- void dlarngenn_cuda (const int m, const int n, const int seed, double ∗x)
- void slarngenn_cuda (const int m, const int n, const int seed, float ∗x)
- double derror_cuda (const int m, const int n, const int k, const double ∗x, const double ∗y, const double ∗z)
- float serror_cuda (const int m, const int n, const int k, const float ∗x, const float ∗y, const float ∗z)
- double derrorbd_cuda (const int m, const int n, const int k, const double ∗A, const double ∗W, const double ∗H, const double beta)
- float serrorbd_cuda (const int m, const int n, const int k, const float ∗A, const float ∗W, const float ∗H, const float beta)

2.37.1 Detailed Description

Header file for using utility modules from CUDA source codes.

Author

Information Retrieval and Parallel Computing Group (IRPCG)
University of Oviedo, Spain
Interdisciplinary Computation and Communication Group (INCO2)
Universitat Politecnica de Valencia, Spain.
Contact: nnmfpack@gmail.com

Date

04/11/14

2.37.2 Function Documentation

2.37.2.1 ddiv_cuda()

```
void ddiv_cuda (
            const int n,
            const double * x,
            double * y,
            cudaStream_t stream )
```

2.37.2.2 derror_cuda()

```
double derror_cuda (
            const int m,
            const int n,
            const int k,
            const double * x,
            const double * y,
            const double * z )
```

2.37.2.3 derrorbd_cuda()

```
double derrorbd_cuda (
            const int m,
            const int n,
            const int k,
            const double * A,
            const double * W,
            const double * H,
            const double beta )
```

2.37.2.4 dlarngenn_cuda()

```
void dlarngenn_cuda (
            const int m,
            const int n,
            const int seed,
            double * x )
```

2.37.2.5 dmemset_cuda()

```
void dmemset_cuda (
            const int n,
            double * x,
            const double val,
            cudaStream_t stream )
```

2.37.2.6 dsub_cuda()

```
void dsub_cuda (
            const int n,
            const double * x,
            double * y )
```

2.37.2.7 sdiv_cuda()

```
void sdiv_cuda (
            const int n,
            const float * x,
            float * y,
            cudaStream_t stream )
```

2.37.2.8 serror_cuda()

```
float serror_cuda (
            const int m,
            const int n,
            const int k,
            const float * x,
            const float * y,
            const float * z )
```

2.37.2.9 serrorbd_cuda()

```
float serrorbd_cuda (
            const int m,
            const int n,
            const int k,
            const float * A,
            const float * W,
            const float * H,
            const float beta )
```

2.37.2.10 slarngenn_cuda()

```
void slarngenn_cuda (
            const int m,
            const int n,
            const int seed,
            float * x )
```

2.37.2.11 smemset_cuda()

```
void smemset_cuda (
            const int n,
            float * x,
            const float val,
            cudaStream_t stream )
```

2.37.2.12 ssub_cuda()

```
void ssub_cuda (
            const int n,
            const float * x,
            float * y )
```

2.37.2.13 vddiv_cuda()

```
__global__ void vddiv_cuda (
              const int n,
              const double *__restrict__ x,
              const double *__restrict__ y,
              double * z )
```

2.37.2.14 vderrorbd0_cuda()

```
__global__ void vderrorbd0_cuda (
              const int n,
              const double *__restrict__ x,
              double * y )
```

2.37.2.15 vderrorbd1_cuda()

```
__global__ void vderrorbd1_cuda (
              const int n,
              const double *__restrict__ x,
              double * y )
```

2.37.2.16 vderrorbdg_cuda()

```
__global__ void vderrorbdg_cuda (
              const int n,
              const double *__restrict__ x,
              double * y,
              const double beta )
```

2.37.2.17 vdmemset_cuda()

```
__global__ void vdmemset_cuda (
              const int n,
              double * x,
              const double val )
```

2.37.2.18 vdsub_cuda()

```
__global__ void vdsub_cuda (
              const int n,
              const double *__restrict__ x,
              double * y )
```

2.37.2.19 vsdiv_cuda()

```
__global__ void vsdiv_cuda (
              const int n,
              const float *__restrict__ x,
              const float *__restrict__ y,
              float * z )
```

2.37.2.20 vserrorbd0_cuda()

```
__global__ void vserrorbd0_cuda (
              const int n,
              const float *__restrict__ x,
              float * y )
```

2.37.2.21 vserrorbd1_cuda()

```
__global__ void vserrorbd1_cuda (
              const int n,
              const float *__restrict__ x,
              float * y )
```

2.37.2.22 vserrorbdg_cuda()

```
__global__ void vserrorbdg_cuda (
              const int n,
              const float *__restrict__ x,
              float * y,
              const double beta )
```

2.37.2.23 vsmemset_cuda()

```
__global__ void vsmemset_cuda (
              const int n,
              float * x,
              const float val )
```

2.37.2.24 vssub_cuda()

```
__global__ void vssub_cuda (
              const int n,
              const float *__restrict__ x,
              float * y )
```

2.38 utils_x86.c File Reference

Some auxiliar functions. Double and simple precision for CPU and MIC.

Functions

- void dmemset_x86 (const int n, double *__restrict__ x, const double val)

  This function fills all positions of x with val.
- void smemset_x86 (const int n, float *__restrict__ x, const float val)

  This function fills all positions of x with val.
- void ddiv_x86 (const int n, const double *x, double *__restrict__ y)

  This function calls the appropiate funtions to performs double precision element-wise y[i]=x[i]/y[i] for all positions of x and y.
- void sdiv_x86 (const int n, const float *x, float *__restrict__ y)

  This function calls the appropiate funtions to performs simple precision element-wise x[i]=x[i]/y[i] for all positions of x and y.
- void dsub_x86 (const int n, const double *x, double *__restrict__ y)

  This function performs double precision element-wise substraction y[i]=x[i]-y[i].
- void ssub_x86 (const int n, const float *x, float *__restrict__ y)

  This function performs simple precision element-wise substraction y[i]=x[i]-y[i].
- void dlarngenn_x86 (const int m, const int n, const int seed, double *X)

  dlarngenn_x86 returns an (m x n) random double precision matrix. An uniform (0, 1) distribution is used to generate the values
- void slarngenn_x86 (const int m, const int n, const int seed, float *X)

  slarngenn_x86 returns an (m x n) random simple precision matrix. An uniform (0, 1) distribution is used to generate the values
- double derror_x86 (const int m, const int n, const int k, const double *A, const double *W, const double *H)

  derror_x86 returns double precision "2norm(A - WH) / sqrt(m x n)"
- float serror_x86 (const int m, const int n, const int k, const float *A, const float *W, const float *H)

  serror_x86 returns simple precision "2norm(A - WH) / sqrt(m x n)"

2.38.1 Detailed Description

Some auxiliar functions. Double and simple precision for CPU and MIC.

Author

Information Retrieval and Parallel Computing Group (IRPCG)
University of Oviedo, Spain
Interdisciplinary Computation and Communication Group (INCO2)
Universitat Politecnica de Valencia, Spain.
Contact: nnmfpack@gmail.com

Date

04/11/14

2.38.2 Function Documentation

2.38.2.1 ddiv_x86()

```
void ddiv_x86 (
            const int n,
            const double * x,
            double *__restrict__ y )
```

This function calls the appropiate funtions to performs double precision element-wise y[i]=x[i]/y[i] for all positions of x and y.

Parameters

n	(input) Number of elements of x and y
x	(input) Double precision input vector/matrix (1D column-major)
y	(inout) Double precision input/output vector/ matrix (1D column-major)

Definition at line 91 of file utils_x86.c.

2.38.2.2 derror_x86()

```
double derror_x86 (
            const int m,
            const int n,
            const int k,
            const double * A,
            const double * W,
            const double * H )
```

derror_x86 returns double precision "2norm(A - WH) / sqrt(m x n)"

Parameters

m	(input) Number of rows of matrix A and number of rows of W
n	(input) Number of columns of matrix A and number of columns of H
k	(input) Number of columns of matrix W and number of rows of H
A	(input) Double precision matrix, dimension (m x n), 1D layout column major
W	(input) Double precision matrix, dimension (m x k), 1D layout column major
H	(input) Double precision matrix, dimension (k x n), 1D layout column major

Definition at line 283 of file utils_x86.c.

2.38.2.3 dlarngenn_x86()

```
void dlarngenn_x86 (
            const int m,
            const int n,
            const int seed,
            double * X )
```

dlarngenn_x86 returns an (m x n) random double precision matrix. An uniform (0, 1) distribution is used to generate the values

Parameters

m	(input) Number of rows of matrix X
n	(input) Number of columns of matrix X
seed	(input) Initial seed for the random numbers
X	(output) Double precision matrix (1D column major)

Definition at line 220 of file utils_x86.c.

2.38.2.4 dmemset_x86()

```
void dmemset_x86 (
            const int n,
            double *__restrict__ x,
            const double val )
```

This function fills all positions of x with val.

Parameters

n	(input) Number of elements of x
x	(output) Double precision output matrix (1D column-major) or vector
val	(input) Double precision value

Definition at line 42 of file utils_x86.c.

2.38.2.5 dsub_x86()

```
void dsub_x86 (
            const int n,
            const double * x,
            double *__restrict__ y )
```

This function performs double precision element-wise substraction y[i]=x[i]-y[i].

Parameters

n	(input) Number of elements of x and y
x	(input) Double precision input vector/matrix (1D column-major)
y	(inout) Double precision input/output vector/matrix (1D column-major)

Definition at line 162 of file utils_x86.c.

2.38.2.6 sdiv_x86()

```
void sdiv_x86 (
            const int n,
            const float * x,
            float *__restrict__ y )
```

This function calls the appropiate funtions to performs simple precision element-wise x[i]=x[i]/y[i] for all positions of x and y.

Parameters

n	(input) Number of elements of x and y
x	(input) Simple precision input vector/matrix (1D column-major)
y	(inout) Simple precision input/output vector/ matrix (1D column-major)

Definition at line 127 of file utils_x86.c.

2.38.2.7 serror_x86()

```
float serror_x86 (
            const int m,
            const int n,
            const int k,
```

```
            const float * A,
            const float * W,
            const float * H )
```

serror_x86 returns simple precision "2norm(A - WH) / sqrt(m x n)"

Parameters

m	(input) Number of rows of matrix A and number of rows of W
n	(input) Number of columns of matrix A and number of columns of H
k	(input) Number of columns of matrix W and number of rows of H
A	(input) Simple precision matrix, dimension (m x n), 1D layout column major
W	(input) Simple precision matrix, dimension (m x k), 1D layout column major
H	(input) Simple precision matrix, dimension (k x n), 1D layout column major

Definition at line 329 of file utils_x86.c.

2.38.2.8 slarngenn_x86()

```
void slarngenn_x86 (
            const int m,
            const int n,
            const int seed,
            float * X )
```

slarngenn_x86 returns an (m x n) random simple precision matrix. An uniform (0, 1) distribution is used to generate the values

Parameters

m	(input) Number of rows of matrix X
n	(input) Number of columns of matrix X
seed	(input) Initial seed for the random numbers
X	(output) Simple precision matrix (1D column major)

Definition at line 251 of file utils_x86.c.

2.38.2.9 smemset_x86()

```
void smemset_x86 (
            const int n,
            float *__restrict__ x,
            const float val )
```

This function fills all positions of x with val.

Parameters

n	(input) Number of elements of x
x	(output) Simple precision output matrix (1D column-major) or vector
val	(input) Simple precision value

Definition at line 66 of file utils_x86.c.

2.38.2.10 ssub_x86()

```
void ssub_x86 (
            const int n,
            const float * x,
            float *__restrict__ y )
```

This function performs simple precision element-wise substraction y[i]=x[i]-y[i].

Parameters

n	(input) Number of elements of x and y
x	(input) Simple precision input vector/matrix (1D column-major)
y	(inout) Simple precision input/output vector/matrix (1D column-major)

Definition at line 190 of file utils_x86.c.

2.39 utils_x86.h File Reference

Header file for using utility modules from CPU/MIC source codes.

Functions

- void dlarnv_ (int *, int *, int *, double *)
- void slarnv_ (int *, int *, int *, float *)
- void dmemset_x86 (const int n, double *__restrict__ x, const double val)

  This function fills all positions of x with val.
- void smemset_x86 (const int n, float *__restrict__ x, const float val)

  This function fills all positions of x with val.
- void ddiv_x86 (const int n, const double *x, double *__restrict__ y)

  This function calls the appropiate funtions to performs double precision element-wise y[i]=x[i]/y[i] for all positions of x and y.
- void sdiv_x86 (const int n, const float *x, float *__restrict__ y)

  This function calls the appropiate funtions to performs simple precision element-wise x[i]=x[i]/y[i] for all positions of x and y.
- void dsub_x86 (const int n, const double *x, double *__restrict__ y)

  This function performs double precision element-wise substraction y[i]=x[i]-y[i].

- void ssub_x86 (const int n, const float ∗x, float ∗__restrict__ y)

  This function performs simple precision element-wise substraction y[i]=x[i]-y[i].
- void dlarngenn_x86 (const int m, const int n, const int seed, double ∗X)

  dlarngenn_x86 returns an (m x n) random double precision matrix. An uniform (0, 1) distribution is used to generate the values
- void slarngenn_x86 (const int m, const int n, const int seed, float ∗X)

  slarngenn_x86 returns an (m x n) random simple precision matrix. An uniform (0, 1) distribution is used to generate the values
- double derror_x86 (const int m, const int n, const int k, const double ∗A, const double ∗W, const double ∗H)

  derror_x86 returns double precision "2norm(A - WH) / sqrt(m x n)"
- float serror_x86 (const int m, const int n, const int k, const float ∗A, const float ∗W, const float ∗H)

  serror_x86 returns simple precision "2norm(A - WH) / sqrt(m x n)"

2.39.1 Detailed Description

Header file for using utility modules from CPU/MIC source codes.

Author

Information Retrieval and Parallel Computing Group (IRPCG)
University of Oviedo, Spain
Interdisciplinary Computation and Communication Group (INCO2)
Universitat Politecnica de Valencia, Spain.
Contact: nnmfpack@gmail.com

Date

04/011/14

2.39.2 Function Documentation

2.39.2.1 ddiv_x86()

```
void ddiv_x86 (
            const int n,
            const double * x,
            double *__restrict__ y )
```

This function calls the appropiate funtions to performs double precision element-wise y[i]=x[i]/y[i] for all positions of x and y.

Parameters

n	(input) Number of elements of x and y
x	(input) Double precision input vector/matrix (1D column-major)
y	(inout) Double precision input/output vector/ matrix (1D column-major)

Definition at line 91 of file utils_x86.c.

2.39.2.2 derror_x86()

```
double derror_x86 (
             const int m,
             const int n,
             const int k,
             const double * A,
             const double * W,
             const double * H )
```

derror_x86 returns double precision "2norm(A - WH) / sqrt(m x n)"

Parameters

m	(input) Number of rows of matrix A and number of rows of W
n	(input) Number of columns of matrix A and number of columns of H
k	(input) Number of columns of matrix W and number of rows of H
A	(input) Double precision matrix, dimension (m x n), 1D layout column major
W	(input) Double precision matrix, dimension (m x k), 1D layout column major
H	(input) Double precision matrix, dimension (k x n), 1D layout column major

Definition at line 283 of file utils_x86.c.

2.39.2.3 dlarngenn_x86()

```
void dlarngenn_x86 (
             const int m,
             const int n,
             const int seed,
             double * X )
```

dlarngenn_x86 returns an (m x n) random double precision matrix. An uniform (0, 1) distribution is used to generate the values

Parameters

m	(input) Number of rows of matrix X
n	(input) Number of columns of matrix X
seed	(input) Initial seed for the random numbers
X	(output) Double precision matrix (1D column major)

Definition at line 220 of file utils_x86.c.

2.39.2.4 dlarnv_()

```
void dlarnv_ (
              int * ,
              int * ,
              int * ,
              double *  )
```

2.39.2.5 dmemset_x86()

```
void dmemset_x86 (
              const int n,
              double *__restrict__ x,
              const double val )
```

This function fills all positions of x with val.

Parameters

n	(input) Number of elements of x
x	(output) Double precision output matrix (1D column-major) or vector
val	(input) Double precision value

Definition at line 42 of file utils_x86.c.

2.39.2.6 dsub_x86()

```
void dsub_x86 (
              const int n,
              const double * x,
              double *__restrict__ y )
```

This function performs double precision element-wise substraction y[i]=x[i]-y[i].

Parameters

n	(input) Number of elements of x and y
x	(input) Double precision input vector/matrix (1D column-major)
y	(inout) Double precision input/output vector/matrix (1D column-major)

Definition at line 162 of file utils_x86.c.

2.39.2.7 sdiv_x86()

```
void sdiv_x86 (
            const int n,
            const float * x,
            float *__restrict__ y )
```

This function calls the appropiate funtions to performs simple precision element-wise x[i]=x[i]/y[i] for all positions of x and y.

Parameters

n	(input) Number of elements of x and y
x	(input) Simple precision input vector/matrix (1D column-major)
y	(inout) Simple precision input/output vector/ matrix (1D column-major)

Definition at line 127 of file utils_x86.c.

2.39.2.8 serror_x86()

```
float serror_x86 (
            const int m,
            const int n,
            const int k,
            const float * A,
            const float * W,
            const float * H )
```

serror_x86 returns simple precision "2norm(A - WH) / sqrt(m x n)"

Parameters

m	(input) Number of rows of matrix A and number of rows of W
n	(input) Number of columns of matrix A and number of columns of H
k	(input) Number of columns of matrix W and number of rows of H
A	(input) Simple precision matrix, dimension (m x n), 1D layout column major
W	(input) Simple precision matrix, dimension (m x k), 1D layout column major
H	(input) Simple precision matrix, dimension (k x n), 1D layout column major

Definition at line 329 of file utils_x86.c.

2.39.2.9 slarngenn_x86()

```
void slarngenn_x86 (
            const int m,
```

```
            const int n,
            const int seed,
            float * X )
```

slarngenn_x86 returns an (m x n) random simple precision matrix. An uniform (0, 1) distribution is used to generate the values

Parameters

m	(input) Number of rows of matrix X
n	(input) Number of columns of matrix X
seed	(input) Initial seed for the random numbers
X	(output) Simple precision matrix (1D column major)

Definition at line 251 of file utils_x86.c.

2.39.2.10 slarnv_()

```
void slarnv_ (
            int * ,
            int * ,
            int * ,
            float *  )
```

2.39.2.11 smemset_x86()

```
void smemset_x86 (
            const int n,
            float *__restrict__ x,
            const float val )
```

This function fills all positions of x with val.

Parameters

n	(input) Number of elements of x
x	(output) Simple precision output matrix (1D column-major) or vector
val	(input) Simple precision value

Definition at line 66 of file utils_x86.c.

2.39.2.12 ssub_x86()

```
void ssub_x86 (
            const int n,
```

```
const float * x,
float *__restrict__ y )
```

This function performs simple precision element-wise substraction y[i]=x[i]-y[i].

Parameters

n	(input) Number of elements of x and y
x	(input) Simple precision input vector/matrix (1D column-major)
y	(inout) Simple precision input/output vector/matrix (1D column-major)

Definition at line 190 of file utils_x86.c.