testing_sgetrf.cpp

```
/*
 *  -- MAGMA (version 1.3.0) --
 *     Univ. of Tennessee, Knoxville
 *     Univ. of California, Berkeley
 *     Univ. of Colorado, Denver
 *     November 2012
 *
 * @generated s Wed Nov 14 22:54:17 2012
 *
 **/
// includes, system
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <cuda_runtime_api.h>
#include <cublas.h>
// includes, project
#include "flops.h"
#include "magma.h"
#include "magma_lapack.h"
#include "testings.h"

// Initialize matrix to random.
// Having this in separate function ensures the same ISEED is always used,
// so we can re-generate the identical matrix.
void init_matrix( int m, int n, float *h_A, magma_int_t lda )
{
    magma_int_t ione = 1;
    magma_int_t ISEED[4] = {0,0,0,1};
    magma_int_t n2 = lda*n;
    lapackf77_slarnv( &ione, ISEED, &n2, h_A );
}

// On input, A and ipiv is LU factorization of A. On output, A is overwritten.
// Requires m == n.
// Uses init_matrix() to re-generate original A as needed.
// Generates random RHS b and solves Ax=b.
// Returns residual, |Ax - b| / (n |A| |x|).
float get_residual(
    magma_int_t m, magma_int_t n,
    float *A, magma_int_t lda,
    magma_int_t *ipiv )
{
    if ( m != n ) {
        printf( "\nERROR: residual check defined only for square matrices\n" );
        return -1;
    }

    const float c_one     = MAGMA_S_ONE;
    const float c_neg_one = MAGMA_S_NEG_ONE;
    const magma_int_t ione = 1;

    // this seed should be DIFFERENT than used in init_matrix
    // (else x is column of A, so residual can be exactly zero)
    magma_int_t ISEED[4] = {0,0,0,2};
    magma_int_t info = 0;
    float *x, *b;

    // initialize RHS
    TESTING_MALLOC( x, float, n );
    TESTING_MALLOC( b, float, n );
    lapackf77_slarnv( &ione, ISEED, &n, b );
    blasf77_scopy( &n, b, &ione, x, &ione );

    // solve Ax = b
    lapackf77_sgetrs( "Notrans", &n, &ione, A, &lda, ipiv, x, &n, &info );
```

```
68      if ( info != 0 )
69          printf( "lapackf77_sgetrs returned error %d\n", info );
70
71      // reset to original A
72      init_matrix( m, n, A, lda );
73
74      // compute r = Ax - b, saved in b
75      blasf77_sgemv( "Notrans", &m, &n, &c_one, A, &lda, x, &ione, &c_neg_one, b, &ione );
76
77      // compute residual |Ax - b| / (n*|A|*|x|)
78      float norm_x, norm_A, norm_r, work[1];
79      norm_A = lapackf77_slange( "F", &m, &n, A, &lda, work );
80      norm_r = lapackf77_slange( "F", &n, &ione, b, &n, work );
81      norm_x = lapackf77_slange( "F", &n, &ione, x, &n, work );
82
83      //printf( "r=\n" ); magma_sprint( 1, n, b, 1 );
84
85      TESTING_FREE( x );
86      TESTING_FREE( b );
87
88      //printf( "r=%.2e, A=%.2e, x=%.2e, n=%d\n", norm_r, norm_A, norm_x, n );
89      return norm_r / (n * norm_A * norm_x);
90  }
91
92  // On input, LU and ipiv is LU factorization of A. On output, LU is overwritten.
93  // Works for any m, n.
94  // Uses init_matrix() to re-generate original A as needed.
95  // Returns error in factorization, |PA - LU| / (n |A|)
96  // This allocates 3 more matrices to store A, L, and U.
97  float get_LU_error(magma_int_t M, magma_int_t N,
98                     float *LU, magma_int_t lda,
99                     magma_int_t *ipiv)
100 {
101     magma_int_t min_mn = min(M,N);
102     magma_int_t ione   = 1;
103     magma_int_t i, j;
104     float alpha = MAGMA_S_ONE;
105     float beta  = MAGMA_S_ZERO;
106     float *A, *L, *U;
107     float work[1], matnorm, residual;
108
109     TESTING_MALLOC( A, float, lda*N    );
110     TESTING_MALLOC( L, float, M*min_mn );
111     TESTING_MALLOC( U, float, min_mn*N );
112     memset( L, 0, M*min_mn*sizeof(float) );
113     memset( U, 0, min_mn*N*sizeof(float) );
114
115     // set to original A
116     init_matrix( M, N, A, lda );
117     lapackf77_slaswp( &N, A, &lda, &ione, &min_mn, ipiv, &ione);
118
119     // copy LU to L and U, and set diagonal to 1
120     lapackf77_slacpy( MagmaLowerStr, &M, &min_mn, LU, &lda, L, &M      );
121     lapackf77_slacpy( MagmaUpperStr, &min_mn, &N, LU, &lda, U, &min_mn );
122     for(j=0; j<min_mn; j++)
123         L[j+j*M] = MAGMA_S_MAKE( 1., 0. );
124
125     matnorm = lapackf77_slange("f", &M, &N, A, &lda, work);
126
127     blasf77_sgemm("N", "N", &M, &N, &min_mn,
128                   &alpha, L, &M, U, &min_mn, &beta, LU, &lda);
129
130     for( j = 0; j < N; j++ ) {
131         for( i = 0; i < M; i++ ) {
132             LU[i+j*lda] = MAGMA_S_SUB( LU[i+j*lda], A[i+j*lda] );
133         }
```

```
  134       }
  135       residual = lapackf77_slange("f", &M, &N, LU, &lda, work);
  136
  137       TESTING_FREE(A);
  138       TESTING_FREE(L);
  139       TESTING_FREE(U);
  140
  141       return residual / (matnorm * N);
  142   }
  143
  144   /* ////////////////////////////////////////////////////////////////////////////
  145      -- Testing sgetrf
  146   */
  147   int main( int argc, char** argv)
  148   {
  149       TESTING_CUDA_INIT();
  150
  151       real_Double_t   gflops, gpu_perf, gpu_time, cpu_perf, cpu_time;
  152       float          error;
  153       float *h_A;
  154       magma_int_t     *ipiv;
  155
  156       /* Matrix size */
  157       magma_int_t M = 0, N = 0, n2, lda, ldda;
  158       const int MAXTESTS = 10;
  159       magma_int_t msize[MAXTESTS] = { 1024, 2048, 3072, 4032, 5184, 6016, 7040, 8064,
9088, 10112 };
  160       magma_int_t nsize[MAXTESTS] = { 1024, 2048, 3072, 4032, 5184, 6016, 7040, 8064,
9088, 10112 };
  161
  162       magma_int_t i, info, min_mn, nb;
  163
  164       int lapack   = getenv("MAGMA_RUN_LAPCAK")     != NULL;
  165       int checkres = getenv("MAGMA_TESTINGS_CHECK") != NULL;
  166
  167       // process command line arguments
  168       printf( "\nUsage: %s -N <m,n> -c -c2 -l\n"
  169               "  -N  can be repeated up to %d times. If only m is given, then m=n.\n"
  170               "  -c  or setting $MAGMA_TESTINGS_CHECK checks result, PA - LU.\n"
  171               "  -c2 for square matrices, solves one RHS and checks residual, Ax - b.\n"
  172               "  -l  or setting $MAGMA_RUN_LAPACK runs LAPACK.\n\n",
  173               argv[0], MAXTESTS );
  174
  175       int ntest = 0;
  176       for( int i = 1; i < argc; ++i ) {
  177           if ( strcmp("-N", argv[i]) == 0 && i+1 < argc ) {
  178               magma_assert( ntest < MAXTESTS, "error: -N repeated more than maximum %d
tests\n", MAXTESTS );
  179               int m, n;
  180               info = sscanf( argv[++i], "%d,%d", &m, &n );
  181               if ( info == 2 && m > 0 && n > 0 ) {
  182                   msize[ ntest ] = m;
  183                   nsize[ ntest ] = n;
  184               }
  185               else if ( info == 1 && m > 0 ) {
  186                   msize[ ntest ] = m;
  187                   nsize[ ntest ] = m;  // implicitly
  188               }
  189               else {
  190                   printf( "error: -N %s is invalid; ensure m > 0, n > 0.\n", argv[i] );
  191                   exit(1);
  192               }
  193               M = max( M, msize[ ntest ] );
  194               N = max( N, nsize[ ntest ] );
  195               ntest++;
  196           }
```

```
  197           else if ( strcmp("-M", argv[i]) == 0 ) {
  198               printf( "-M has been replaced in favor of -N m,n to allow -N to be
repeated.\n\n" );
  199               exit(1);
  200           }
  201           else if ( strcmp("-c", argv[i]) == 0 ) {
  202               checkres = 1;
  203           }
  204           else if ( strcmp("-c2", argv[i]) == 0 ) {
  205               checkres = 2;
  206           }
  207           else if ( strcmp("-l", argv[i]) == 0 ) {
  208               lapack = true;
  209           }
  210           else {
  211               printf( "invalid argument: %s\n", argv[i] );
  212               exit(1);
  213           }
  214       }
  215       if ( ntest == 0 ) {
  216           ntest = MAXTESTS;
  217           M = msize[ntest-1];
  218           N = nsize[ntest-1];
  219       }
  220
  221       ldda   = ((M+31)/32)*32;
  222       n2     = M * N;
  223       min_mn = min(M, N);
  224       nb     = magma_get_sgetrf_nb(min_mn);
  225
  226       /* Allocate memory for the matrix */
  227       TESTING_MALLOC(ipiv, magma_int_t, min_mn);
  228       TESTING_HOSTALLOC( h_A, float, n2 );
  229
  230       if ( checkres == 2 ) {
  231           printf("    M     N   CPU GFlop/s (sec)   GPU GFlop/s (sec)   |Ax-
b|/(N*|A|*|x|)\n");
  232       }
  233       else {
  234           printf("    M     N   CPU GFlop/s (sec)   GPU GFlop/s (sec)   |PA-
LU|/(N*|A|)\n");
  235       }
  236
printf("=========================================================================\n");
  237       for( i = 0; i < ntest; ++i ) {
  238           M = msize[i];
  239           N = nsize[i];
  240           min_mn = min(M, N);
  241           lda    = M;
  242           n2     = lda*N;
  243           ldda   = ((M+31)/32)*32;
  244           gflops = FLOPS_SGETRF( M, N ) / 1e9;
  245
  246           /* ====================================================================
  247              Performs operation using LAPACK
  248              =================================================================== */
  249           if ( lapack ) {
  250               init_matrix( M, N, h_A, lda );
  251
  252               cpu_time = magma_wtime();
  253               lapackf77_sgetrf(&M, &N, h_A, &lda, ipiv, &info);
  254               cpu_time = magma_wtime() - cpu_time;
  255               cpu_perf = gflops / cpu_time;
  256               if (info != 0)
  257                   printf("lapackf77_sgetrf returned error %d.\n", (int) info);
  258           }
```

```
259
260         /* ====================================================================
261            Performs operation using MAGMA
262            =================================================================== */
263         init_matrix( M, N, h_A, lda );
264
265         gpu_time = magma_wtime();
266         magma_sgetrf( M, N, h_A, lda, ipiv, &info);
267         gpu_time = magma_wtime() - gpu_time;
268         gpu_perf = gflops / gpu_time;
269         if (info != 0)
270             printf("magma_sgetrf returned error %d.\n", (int) info);
271
272         /* ====================================================================
273            Check the factorization
274            =================================================================== */
275         printf("%5d %5d", (int) M, (int) N );
276         if ( lapack ) {
277             printf("   %7.2f (%7.2f)", cpu_perf, cpu_time );
278         }
279         else {
280             printf("     ---   (  ---  )");
281         }
282         printf("   %7.2f (%7.2f)", gpu_perf, gpu_time );
283         if ( checkres == 2 ) {
284             error = get_residual( M, N, h_A, lda, ipiv );
285             printf("   %8.2e\n", error );
286         }
287         else if ( checkres ) {
288             error = get_LU_error(M, N, h_A, lda, ipiv );
289             printf("   %8.2e\n", error );
290         }
291         else {
292             printf("     ---   \n");
293         }
294     }
295
296     /* Memory clean up */
297     TESTING_FREE( ipiv );
298     TESTING_HOSTFREE( h_A );
299
300     TESTING_CUDA_FINALIZE();
301     return 0;
302 }
```

testing_zgetrf_gpu.cpp

```
/*
 *  -- MAGMA (version 1.3.0) --
 *     Univ. of Tennessee, Knoxville
 *     Univ. of California, Berkeley
 *     Univ. of Colorado, Denver
 *     November 2012
 *
 * @precisions normal z -> c d s
 *
 **/
// includes, system
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <cuda_runtime_api.h>
#include <cublas.h>

// includes, project
#include "flops.h"
#include "magma.h"
#include "magma_lapack.h"
#include "testings.h"


double get_LU_error(magma_int_t M, magma_int_t N,
                    cuDoubleComplex *A,  magma_int_t lda,
                    cuDoubleComplex *LU, magma_int_t *IPIV)
{
    magma_int_t min_mn = min(M,N);
    magma_int_t ione   = 1;
    magma_int_t i, j;
    cuDoubleComplex alpha = MAGMA_Z_ONE;
    cuDoubleComplex beta  = MAGMA_Z_ZERO;
    cuDoubleComplex *L, *U;
    double work[1], matnorm, residual;

    TESTING_MALLOC( L, cuDoubleComplex, M*min_mn);
    TESTING_MALLOC( U, cuDoubleComplex, min_mn*N);
    memset( L, 0, M*min_mn*sizeof(cuDoubleComplex) );
    memset( U, 0, min_mn*N*sizeof(cuDoubleComplex) );

    lapackf77_zlaswp( &N, A, &lda, &ione, &min_mn, IPIV, &ione);
    lapackf77_zlacpy( MagmaLowerStr, &M, &min_mn, LU, &lda, L, &M      );
    lapackf77_zlacpy( MagmaUpperStr, &min_mn, &N, LU, &lda, U, &min_mn );

    for(j=0; j<min_mn; j++)
        L[j+j*M] = MAGMA_Z_MAKE( 1., 0. );

    matnorm = lapackf77_zlange("f", &M, &N, A, &lda, work);

    blasf77_zgemm("N", "N", &M, &N, &min_mn,
                  &alpha, L, &M, U, &min_mn, &beta, LU, &lda);

    for( j = 0; j < N; j++ ) {
        for( i = 0; i < M; i++ ) {
            LU[i+j*lda] = MAGMA_Z_SUB( LU[i+j*lda], A[i+j*lda] );
        }
    }
    residual = lapackf77_zlange("f", &M, &N, LU, &lda, work);

    TESTING_FREE(L);
    TESTING_FREE(U);

    return residual / (matnorm * N);
}

```

```
   68  /* ////////////////////////////////////////////////////////////////////////////
   69     -- Testing zgetrf
   70  */
   71  int main( int argc, char** argv)
   72  {
   73      TESTING_CUDA_INIT();
   74
   75      real_Double_t   gflops, gpu_perf, gpu_time, cpu_perf, cpu_time;
   76      double          error;
   77      cuDoubleComplex *h_A, *h_R;
   78      cuDoubleComplex *d_A;
   79      magma_int_t     *ipiv;
   80
   81      /* Matrix size */
   82      magma_int_t M = 0, N = 0, n2, lda, ldda;
   83      const int MAXTESTS = 10;
   84      magma_int_t msize[MAXTESTS] = { 1024, 2048, 3072, 4032, 5184, 6016, 7040, 8064,
9088, 10112 };
   85      magma_int_t nsize[MAXTESTS] = { 1024, 2048, 3072, 4032, 5184, 6016, 7040, 8064,
9088, 10112 };
   86
   87      magma_int_t i, info, min_mn, nb;
   88      magma_int_t ione     = 1;
   89      magma_int_t ISEED[4] = {0,0,0,1};
   90      magma_int_t checkres;
   91
   92      checkres = getenv("MAGMA_TESTINGS_CHECK") != NULL;
   93
   94      // process command line arguments
   95      printf( "\nUsage: %s -N <m,n> -c\n", argv[0] );
   96      printf( "  -N can be repeated up to %d times. If only m is given, then m=n.\n",
MAXTESTS );
   97      printf( "  -c or setting $MAGMA_TESTINGS_CHECK runs LAPACK and checks result.\n\n"
);
   98      int ntest = 0;
   99      for( int i = 1; i < argc; ++i ) {
  100          if ( strcmp("-N", argv[i]) == 0 && i+1 < argc ) {
  101              magma_assert( ntest < MAXTESTS, "error: -N repeated more than maximum %d
tests\n", MAXTESTS );
  102              int m, n;
  103              info = sscanf( argv[++i], "%d,%d", &m, &n );
  104              if ( info == 2 && m > 0 && n > 0 ) {
  105                  msize[ ntest ] = m;
  106                  nsize[ ntest ] = n;
  107              }
  108              else if ( info == 1 && m > 0 ) {
  109                  msize[ ntest ] = m;
  110                  nsize[ ntest ] = m;  // implicitly
  111              }
  112              else {
  113                  printf( "error: -N %s is invalid; ensure m > 0, n > 0.\n", argv[i] );
  114                  exit(1);
  115              }
  116              M = max( M, msize[ ntest ] );
  117              N = max( N, nsize[ ntest ] );
  118              ntest++;
  119          }
  120          else if ( strcmp("-M", argv[i]) == 0 ) {
  121              printf( "-M has been replaced in favor of -N m,n to allow -N to be
repeated.\n\n" );
  122              exit(1);
  123          }
  124          else if ( strcmp("-c", argv[i]) == 0 ) {
  125              checkres = true;
  126          }
  127          else {
```

```
  128               printf( "invalid argument: %s\n", argv[i] );
  129               exit(1);
  130           }
  131       }
  132       if ( ntest == 0 ) {
  133           ntest = MAXTESTS;
  134           M = msize[ntest-1];
  135           N = nsize[ntest-1];
  136       }
  137
  138       ldda   = ((M+31)/32)*32;
  139       n2     = M * N;
  140       min_mn = min(M, N);
  141       nb     = magma_get_zgetrf_nb(min_mn);
  142
  143       /* Allocate memory for the matrix */
  144       TESTING_MALLOC(ipiv, magma_int_t, min_mn);
  145       TESTING_MALLOC(    h_A, cuDoubleComplex, n2     );
  146       TESTING_HOSTALLOC( h_R, cuDoubleComplex, n2     );
  147       TESTING_DEVALLOC(  d_A, cuDoubleComplex, ldda*N );
  148
  149       printf("  M     N     CPU GFlop/s (sec)   GPU GFlop/s (sec)   ||PA-
LU||/(||A||*N)\n");
  150
printf("=========================================================================\n");
  151       for( i = 0; i < ntest; ++i ) {
  152           M = msize[i];
  153           N = nsize[i];
  154           min_mn= min(M, N);
  155           lda   = M;
  156           n2    = lda*N;
  157           ldda  = ((M+31)/32)*32;
  158           gflops = FLOPS_ZGETRF( M, N ) / 1e9;
  159
  160           /* Initialize the matrix */
  161           if ( checkres ) {
  162               lapackf77_zlarnv( &ione, ISEED, &n2, h_A );
  163               lapackf77_zlacpy( MagmaUpperLowerStr, &M, &N, h_A, &lda, h_R, &lda );
  164           }
  165           else
  166               lapackf77_zlarnv( &ione, ISEED, &n2, h_R );
  167           magma_zsetmatrix( M, N, h_R, lda, d_A, ldda );
  168
  169           /* ====================================================================
  170              Performs operation using LAPACK
  171              =================================================================== */
  172           if ( checkres ) {
  173               cpu_time = magma_wtime();
  174               lapackf77_zgetrf(&M, &N, h_A, &lda, ipiv, &info);
  175               cpu_time = magma_wtime() - cpu_time;
  176               cpu_perf = gflops / cpu_time;
  177               if (info != 0)
  178                   printf("lapackf77_zgetrf returned error %d.\n", (int) info);
  179           }
  180
  181           /* ====================================================================
  182              Performs operation using MAGMA
  183              =================================================================== */
  184           gpu_time = magma_wtime();
  185           magma_zgetrf_gpu( M, N, d_A, ldda, ipiv, &info);
  186           gpu_time = magma_wtime() - gpu_time;
  187           gpu_perf = gflops / gpu_time;
  188           if (info != 0)
  189               printf("magma_zgetrf_gpu returned error %d.\n", (int) info);
  190
  191           /* ====================================================================
```

```
192             Check the factorization
193             =================================================================== */
194         if ( checkres ) {
195             magma_zgetmatrix( M, N, d_A, ldda, h_A, lda );
196             error = get_LU_error(M, N, h_R, lda, h_A, ipiv);
197 
198             printf("%5d %5d   %7.2f (%7.2f)   %7.2f (%7.2f)   %8.2e\n",
199                    (int) M, (int) N, cpu_perf, cpu_time, gpu_perf, gpu_time, error);
200         }
201         else {
202             printf("%5d %5d     ---   (  ---  )   %7.2f (%7.2f)     ---  \n",
203                    (int) M, (int) N, gpu_perf, gpu_time);
204         }
205     }
206 
207     /* Memory clean up */
208     TESTING_FREE( ipiv );
209     TESTING_FREE( h_A );
210     TESTING_HOSTFREE( h_R );
211     TESTING_DEVFREE( d_A );
212 
213     TESTING_CUDA_FINALIZE();
214     return 0;
215 }
216 
```

testing_zgetrf_mgpu.cpp

```
/*
 *  -- MAGMA (version 1.3.0) --
 *     Univ. of Tennessee, Knoxville
 *     Univ. of California, Berkeley
 *     Univ. of Colorado, Denver
 *     November 2012
 *
 * @precisions normal z -> c d s
 *
 **/
// includes, system
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <cuda_runtime_api.h>
#include <cublas.h>

// includes, project
#include "flops.h"
#include "magma.h"
#include "magma_lapack.h"
#include "testings.h"

// Flops formula
#define PRECISION_z
#if defined(PRECISION_z) || defined(PRECISION_c)
#define FLOPS(m, n) ( 6. * FMULS_GETRF(m, n) + 2. * FADDS_GETRF(m, n) )
#else
#define FLOPS(m, n) (      FMULS_GETRF(m, n) +      FADDS_GETRF(m, n) )
#endif

/* ========= definition of multiple GPU code ========= */
extern "C" magma_int_t
magma_zgetrf_mgpu(magma_int_t num_gpus,
                      magma_int_t m, magma_int_t n,
                  cuDoubleComplex **d_lA, magma_int_t ldda,
                          magma_int_t *ipiv, magma_int_t *info);
/* ================================================== */



double get_LU_error(magma_int_t M, magma_int_t N,
                    cuDoubleComplex *A,  magma_int_t lda,
                    cuDoubleComplex *LU, magma_int_t *IPIV)
{
    magma_int_t min_mn = min(M,N);
    magma_int_t ione   = 1;
    magma_int_t i, j;
    cuDoubleComplex alpha = MAGMA_Z_ONE;
    cuDoubleComplex beta  = MAGMA_Z_ZERO;
    cuDoubleComplex *L, *U;
    double work[1], matnorm, residual;

    TESTING_MALLOC( L, cuDoubleComplex, M*min_mn);
    TESTING_MALLOC( U, cuDoubleComplex, min_mn*N);
    memset( L, 0, M*min_mn*sizeof(cuDoubleComplex) );
    memset( U, 0, min_mn*N*sizeof(cuDoubleComplex) );

    lapackf77_zlaswp( &N, A, &lda, &ione, &min_mn, IPIV, &ione);
    lapackf77_zlacpy( MagmaLowerStr, &M, &min_mn, LU, &lda, L, &M      );
    lapackf77_zlacpy( MagmaUpperStr, &min_mn, &N, LU, &lda, U, &min_mn );

    for(j=0; j<min_mn; j++)
        L[j+j*M] = MAGMA_Z_MAKE( 1., 0. );

    matnorm = lapackf77_zlange("f", &M, &N, A, &lda, work);
```

```
   68
   69       blasf77_zgemm("N", "N", &M, &N, &min_mn,
   70                     &alpha, L, &M, U, &min_mn, &beta, LU, &lda);
   71
   72       for( j = 0; j < N; j++ ) {
   73           for( i = 0; i < M; i++ ) {
   74               LU[i+j*lda] = MAGMA_Z_SUB( LU[i+j*lda], A[i+j*lda] );
   75           }
   76       }
   77       residual = lapackf77_zlange("f", &M, &N, LU, &lda, work);
   78
   79       TESTING_FREE(L);
   80       TESTING_FREE(U);
   81
   82       return residual / (matnorm * N);
   83   }
   84
   85   /* ////////////////////////////////////////////////////////////////////////////
   86      -- Testing zgetrf
   87   */
   88   int main( int argc, char** argv)
   89   {
   90       TESTING_CUDA_INIT();
   91
   92       magma_timestr_t  start, end;
   93       double           flops, gpu_perf, cpu_perf, error;
   94       cuDoubleComplex *h_A, *h_R;
   95       cuDoubleComplex *d_lA[4];
   96       magma_int_t     *ipiv;
   97
   98       /* Matrix size */
   99       magma_int_t M = 0, N = 0, flag = 0, n2, lda, ldda, num_gpus, num_gpus0 = 1, n_local;
  100       magma_int_t size[10] = {1024,2048,3072,4032,5184,6016,7040,8064,9088,10112};
  101       magma_int_t n_size = 10;
  102
  103       magma_int_t i, k, info, min_mn, nb0, nb, nk, maxn, ret, ldn_local;
  104       magma_int_t ione     = 1;
  105       magma_int_t ISEED[4] = {0,0,0,1};
  106
  107       if (argc != 1){
  108           for(i = 1; i<argc; i++){
  109               if (strcmp("-N", argv[i])==0) {
  110                       N = atoi(argv[++i]);
  111                       flag = 1;
  112               } else if (strcmp("-M", argv[i])==0) {
  113                       M = atoi(argv[++i]);
  114                       flag = 1;
  115               } else if (strcmp("-NGPU", argv[i])==0)
  116                   num_gpus0 = atoi(argv[++i]);
  117           }
  118       }
  119       if( flag != 0 ) {
  120           if (M>0 && N>0 && num_gpus0>0)
  121               printf("  testing_zgetrf_mgpu -M %d -N %d -NGPU %d\n\n", (int) M, (int) N,
(int) num_gpus0);
  122           else {
  123               printf("\nError: \n");
  124               printf("  (m, n, num_gpus)=(%d, %d, %d) must be positive.\n\n", (int) M,
(int) N, (int) num_gpus0);
  125               exit(1);
  126           }
  127       } else {
  128           M = N = size[n_size-1];
  129           printf("\nDefault: \n");
  130           printf("  testing_zgetrf_mgpu -M %d -N %d -NGPU %d\n\n", (int) M, (int) N, (int)
num_gpus0);
```

```
131       }
132
133       ldda   = ((M+31)/32)*32;
134       maxn   = ((N+31)/32)*32;
135       n2     = M * N;
136       min_mn = min(M, N);
137       nb     = magma_get_zgetrf_nb(M);
138           num_gpus = num_gpus0;
139
140       /* Allocate host memory for the matrix */
141       TESTING_MALLOC(ipiv, magma_int_t, min_mn);
142       TESTING_MALLOC(    h_A, cuDoubleComplex, n2     );
143       TESTING_HOSTALLOC( h_R, cuDoubleComplex, n2     );
144           /* allocate device memory, assuming fixed nb and num_gpus */
145       for(i=0; i<num_gpus; i++){
146             n_local = ((N/nb)/num_gpus)*nb;
147             if (i < (N/nb)%num_gpus)
148               n_local += nb;
149             else if (i == (N/nb)%num_gpus)
150               n_local += N%nb;
151             ldn_local = ((n_local+31)/32)*32;
152         cudaSetDevice(i);
153         //TESTING_DEVALLOC( d_lA[i], cuDoubleComplex, ldda*n_local );
154         TESTING_DEVALLOC( d_lA[i], cuDoubleComplex, ldda*ldn_local );
155       }
156       cudaSetDevice(0);
157       nb0 = nb;
158
159       printf("  M     N   CPU GFlop/s    GPU GFlop/s   ||PA-LU||/(||A||*N)\n");
160       printf("=========================================================\n");
161       for(i=0; i<n_size; i++){
162           if (flag == 0){
163                 M = N = size[i];
164           }
165               min_mn= min(M, N);
166               lda   = M;
167               n2    = lda*N;
168               ldda  = ((M+31)/32)*32;
169               flops = FLOPS( (double)M, (double)N ) / 1000000;
170
171           /* Initialize the matrix */
172           lapackf77_zlarnv( &ione, ISEED, &n2, h_A );
173           lapackf77_zlacpy( MagmaUpperLowerStr, &M, &N, h_A, &lda, h_R, &lda );
174
175          /* ====================================================================
176              Performs operation using LAPACK
177              =================================================================== */
178           start = get_current_time();
179           lapackf77_zgetrf(&M, &N, h_A, &lda, ipiv, &info);
180           end = get_current_time();
181           if (info < 0) {
182                           printf("Argument %d of zgetrf had an illegal value.\n", (int) -
info);
183                           break;
184                   } else if (info != 0 ) {
185                           printf("zgetrf returned info=%d.\n", (int) info);
186                           break;
187                   }
188           cpu_perf = flops / GetTimerValue(start, end);
189                   lapackf77_zlacpy( MagmaUpperLowerStr, &M, &N, h_R, &lda, h_A, &lda );
190
191           /* ====================================================================
192              Performs operation using MAGMA
193              =================================================================== */
194                   /* == distributing the matrix == */
195                   //cudaSetDevice(0);
```

```
  196           //cublasSetMatrix( M, N, sizeof(cuDoubleComplex), h_R, lda, d_lA[0], ldda);
  197           nb = magma_get_zgetrf_nb(M);
  198   #ifdef TESTING_ZGETRF_MGPU_CHECK
  199                   if( nb != nb0 ) {
  200                     printf( " different nb used for memory-allocation (%d vs. %d)\n",
(int) nb, (int) nb0 );
  201                   }
  202   #endif
  203           if( num_gpus0 > N/nb ) {
  204                     num_gpus = N/nb;
  205                     if( N%nb != 0 ) num_gpus ++;
  206                     printf( " * too many GPUs for the matrix size, using %d GPUs\n", (int)
num_gpus );
  207                   } else {
  208                     num_gpus = num_gpus0;
  209                   }
  210
  211                   for(int j=0; j<N; j+=nb){
  212                     k = (j/nb)%num_gpus;
  213                     cudaSetDevice(k);
  214                     nk = min(nb, N-j);
  215                     magma_zsetmatrix( M, nk,
  216                                       h_R+j*lda,                          lda,
  217                                       d_lA[k]+j/(nb*num_gpus)*nb*ldda, ldda );
  218                   }
  219                   cudaSetDevice(0);
  220
  221                   /* == calling MAGMA with multiple GPUs == */
  222           start = get_current_time();
  223           magma_zgetrf_mgpu( num_gpus, M, N, d_lA, ldda, ipiv, &info);
  224           end = get_current_time();
  225           gpu_perf = flops / GetTimerValue(start, end);
  226           if (info < 0) {
  227               printf("Argument %d of magma_zgetrf_mgpu had an illegal value.\n", (int) -
info);
  228                           break;
  229               } else if (info != 0 ) {
  230               printf("magma_zgetrf_mgpu returned info=%d.\n", (int) info);
  231                           break;
  232                   }
  233                   /* == download the matrix from GPUs == */
  234                   //cudaSetDevice(0);
  235                   //cublasGetMatrix( M, N, sizeof(cuDoubleComplex), d_lA[0], ldda, h_R,
M);
  236           for(int j=0; j<N; j+=nb){
  237                     k = (j/nb)%num_gpus;
  238                     cudaSetDevice(k);
  239                     nk = min(nb, N-j);
  240                     magma_zgetmatrix( M, nk,
  241                                       d_lA[k]+j/(nb*num_gpus)*nb*ldda, ldda,
  242                                       h_R+j*lda,                          lda );
  243                   }
  244                   cudaSetDevice(0);
  245
  246           /* =====================================================================
  247              Check the factorization
  248              ===================================================================== */
  249           error = get_LU_error(M, N, h_A, lda, h_R, ipiv);
  250
  251           printf("%5d %5d  %6.2f         %6.2f         %e\n",
  252                  (int) M, (int) N, cpu_perf, gpu_perf, error);
  253
  254           if (flag != 0)
  255               break;
  256       }
  257
```

```
258       /* Memory clean up */
259       TESTING_FREE( ipiv );
260       TESTING_FREE( h_A );
261       TESTING_HOSTFREE( h_R );
262       for(i=0; i<num_gpus0; i++){
263                   TESTING_DEVFREE( d_lA[i] );
264           }
265
266       /* Shutdown */
267       TESTING_CUDA_FINALIZE();
268   }
269
```

example_dgesv.c

```c
/**
 *
 * @file example_dgesv.c
 *
 *  PLASMA testing routines
 *  PLASMA is a software package provided by Univ. of Tennessee,
 *  Univ. of California Berkeley and Univ. of Colorado Denver
 *
 * @brief Example for solving a system of linear equations using LU factorization
 *
 * @version 2.5.0
 * @author Bilel Hadri
 * @date 2010-11-15
 * @generated d Thu Nov  8 11:44:46 2012
 *
 **/
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>

#include <plasma.h>
#include <cblas.h>
#include <lapacke.h>
#include <core_blas.h>

int check_solution(int, int , double *, int, double *, double *, int);

int IONE=1;
int ISEED[4] = {0,0,0,1};   /* initial seed for dlarnv() */

int main ()
{

    int cores = 2;
    int N     = 10;
    int LDA   = 10;
    int NRHS  = 5;
    int LDB   = 10;
    int info;
    int info_solution;
    int i,j;
    int LDAxN = LDA*N;
    int LDBxNRHS = LDB*NRHS;

    double *A1 = (double *)malloc(LDA*N*(sizeof*A1));
    double *A2 = (double *)malloc(LDA*N*(sizeof*A2));
    double *B1 = (double *)malloc(LDB*NRHS*(sizeof*B1));
    double *B2 = (double *)malloc(LDB*NRHS*(sizeof*B2));
    PLASMA_desc *L;
    int *IPIV;

    /* Check if unable to allocate memory */
    if ((!A1)||(!A2)||(!B1)||(!B2)){
        printf("Out of Memory \n ");
        return EXIT_SUCCESS;
    }

    /*Plasma Initialize*/
    PLASMA_Init(cores);
    printf("-- PLASMA is initialized to run on %d cores. \n",cores);

    /* Initialize A1 and A2 Matrix */
    LAPACKE_dlarnv_work(IONE, ISEED, LDAxN, A1);
    for ( i = 0; i < N; i++)
        for (  j = 0; j < N; j++)
            A2[LDA*j+i] = A1[LDA*j+i];
```

```
68
69      /* Initialize B1 and B2 */
70      LAPACKE_dlarnv_work(IONE, ISEED, LDBxNRHS, B1);
71      for ( i = 0; i < N; i++)
72          for ( j = 0; j < NRHS; j++)
73              B2[LDB*j+i] = B1[LDB*j+i];
74
75      /* PLASMA DGESV */
76      info = PLASMA_Alloc_Workspace_dgesv_incpiv(N, &L, &IPIV);
77      info = PLASMA_dgesv_incpiv(N, NRHS, A2, LDA, L, IPIV, B2, LDB);
78
79      /* Check the factorization and the solution */
80      info_solution = check_solution(N, NRHS, A1, LDA, B1, B2, LDB);
81
82      if ((info_solution != 0)|(info != 0))
83         printf("-- Error in DGESV example ! \n");
84      else
85         printf("-- Run of DGESV example successful ! \n");
86
87      free(A1); free(A2); free(B1); free(B2); free(IPIV); free(L);
88
89      PLASMA_Finalize();
90
91      return EXIT_SUCCESS;
92  }
93
94  /*------------------------------------------------------------------------
95   *  Check the accuracy of the solution of the linear system
96   */
97
98  int check_solution(int N, int NRHS, double *A1, int LDA, double *B1, double *B2, int
LDB)
99  {
100     int info_solution;
101     double Rnorm, Anorm, Xnorm, Bnorm;
102     double alpha, beta;
103     double *work = (double *)malloc(N*sizeof(double));
104     double eps;
105
106     eps = LAPACKE_dlamch_work('e');
107
108     alpha = 1.0;
109     beta  = -1.0;
110
111     Xnorm = LAPACKE_dlange_work(LAPACK_COL_MAJOR, lapack_const(PlasmaInfNorm), N, NRHS,
B2, LDB, work);
112     Anorm = LAPACKE_dlange_work(LAPACK_COL_MAJOR, lapack_const(PlasmaInfNorm), N, N, A1,
LDA, work);
113     Bnorm = LAPACKE_dlange_work(LAPACK_COL_MAJOR, lapack_const(PlasmaInfNorm), N, NRHS,
B1, LDB, work);
114
115     cblas_dgemm(CblasColMajor, CblasNoTrans, CblasNoTrans, N, NRHS, N, (alpha), A1, LDA,
B2, LDB, (beta), B1, LDB);
116     Rnorm = LAPACKE_dlange_work(LAPACK_COL_MAJOR, lapack_const(PlasmaInfNorm), N, NRHS,
B1, LDB, work);
117
118     printf("============\n");
119     printf("Checking the Residual of the solution \n");
120     printf("-- ||Ax-B||_oo/((||A||_oo||x||_oo+||B||_oo).N.eps) = %e
\n",Rnorm/((Anorm*Xnorm+Bnorm)*N*eps));
121
122     if ( isnan(Rnorm/((Anorm*Xnorm+Bnorm)*N*eps)) || (Rnorm/((Anorm*Xnorm+Bnorm)*N*eps)
> 10.0) ){
123         printf("-- The solution is suspicious ! \n");
124         info_solution = 1;
125     }
```

```
126       else{
127           printf("-- The solution is CORRECT ! \n");
128           info_solution = 0;
129       }
130
131       free(work);
132
133       return info_solution;
134   }
135
```