Retargeting PLAPACK to Clusters with Hardware Accelerators

Manuel Fogué[1] Francisco Igual[1]
Enrique S. Quintana-Ortí[1] Robert van de Geijn[2]

[1]Departamento de Ingeniería y Ciencia de los Computadores.
University Jaume I.
Castellón (Spain).

[2]Department of Computer Sciences.
Texas Institute for Computational Engineering and Sciences.
The University of Texas at Austin.

Motivation

Past, present and future of GPU computing

Systems with one GPU

Software:

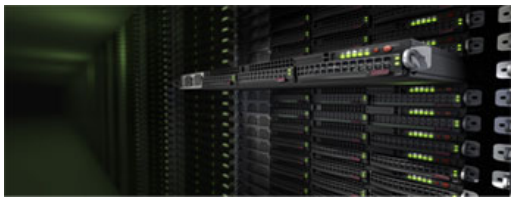
- CUDA
- OpenCL
- OpenGL, Cg, ...
- ATI Stream

Multi-GPU systems

Software:

- GPUSs
- Star-PU
- FLAME SuperMatrix

Clusters of GPUs



Software:

- ??
- Our approach: CUPLAPACK

Future is today...

China's new Nebulae Supercomputer is No. 2 at Top500 (June 2010)

Powered with Tesla C2050 GPUs

Motivation

Past, present and future of GPU computing

Systems with one GPU

Software:

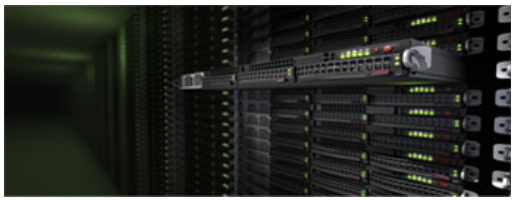
- CUDA
- OpenCL
- OpenGL, Cg, ...
- ATI Stream

Multi-GPU systems

Software:

- GPUSs
- Star-PU
- FLAME SuperMatrix

Clusters of GPUs



Software:

- ??
- Our approach: CUPLAPACK

Future is today...

China's new Nebulae Supercomputer is No. 2 at Top500 (June 2010)

Powered with Tesla C2050 GPUs

Motivation

Past, present and future of GPU computing

Systems with one GPU

Software:

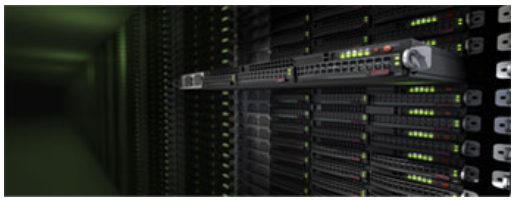
- CUDA
- OpenCL
- OpenGL, Cg, ...
- ATI Stream

Multi-GPU systems

Software:

- GPUSs
- Star-PU
- FLAME SuperMatrix

Clusters of GPUs



Software:

- ??
- Our approach: CUPLAPACK

Future is today...

China's new Nebulae Supercomputer is No. 2 at Top500 (June 2010)

Powered with Tesla C2050 GPUs

Outline

Introduction

- PLAPACK:
  - Parallel dense linear algebra package for message-passing architectures (i.e., clusters).
- CUPLAPACK:
  - Retarget of PLAPACK to clusters with graphics processors.
- Why a PLAPACK-based approach?:
  - Modularity: layered design.
  - Programmability: FLAME methodology. Object-based approach.
- Goals:
  1. Programmability: Transparent port to clusters of GPUs.
  2. Performance: Clusters of GPUs with hundreds of nodes.
  3. Scalability: Keep performance with large problems.

PLAPACK overview

PLAPACK: Parallel Linear Algebra Package

- *"Library infrastructure for parallel implementation of linear algebra algorithms on distributed memory machines".*
- Natural transition from algorithms to distributed-memory codes. Focuses on programmability.
- Layered design. Focuses on extensibility and flexibility.
- Physically Based Matrix Distribution (PBMD) and templates.

Cholesky factorization. Right-looking variant (I)

Cholesky factorization of a positive definite matrix A:

$$A = LL^T$$

Partition:

$$A = \left(\begin{array}{c|c} A_{11} & \star \\ \hline A_{21} & A_{22} \end{array}\right), \; L = \left(\begin{array}{c|c} L_{11} & 0 \\ \hline L_{21} & L_{22} \end{array}\right),$$

where A_{11} and L_{11} are $b \times b$ matrices.

From $A = LL^T$, we have that:

$$\left(\begin{array}{c|c} A_{11} & \star \\ \hline A_{21} & A_{22} \end{array}\right) = \left(\begin{array}{c|c} L_{11} & 0 \\ \hline L_{21} & L_{22} \end{array}\right)\left(\begin{array}{c|c} L_{11}^T & L_{21}^T \\ \hline 0 & L_{22}^T \end{array}\right)$$

$$= \left(\begin{array}{c|c} L_{11}L_{11}^T & \star \\ \hline L_{21}L_{11}^T & L_{21}L_{21}^T + L_{22}L_{22}^T \end{array}\right).$$

Cholesky factorization. Right-looking variant (I)

Cholesky factorization of a positive definite matrix A:

$$A = LL^T$$

Partition:

$$A = \left(\begin{array}{c|c} A_{11} & \star \\ \hline A_{21} & A_{22} \end{array}\right), \; L = \left(\begin{array}{c|c} L_{11} & 0 \\ \hline L_{21} & L_{22} \end{array}\right),$$

where A_{11} and L_{11} are $b \times b$ matrices.

From $A = LL^T$, we have that:

$$\begin{aligned} \left(\begin{array}{c|c} A_{11} & \star \\ \hline A_{21} & A_{22} \end{array}\right) &= \left(\begin{array}{c|c} L_{11} & 0 \\ \hline L_{21} & L_{22} \end{array}\right) \left(\begin{array}{c|c} L_{11}^T & L_{21}^T \\ \hline 0 & L_{22}^T \end{array}\right) \\ &= \left(\begin{array}{c|c} L_{11}L_{11}^T & \star \\ \hline L_{21}L_{11}^T & L_{21}L_{21}^T + L_{22}L_{22}^T \end{array}\right). \end{aligned}$$

Cholesky factorization. Right-looking variant (II)

This yields to equations:

$$\begin{aligned} L_{11}L_{11}^T &= A_{11}, \\ L_{21} &= A_{21}L_{11}^{-T}, \\ A_{22} - L_{21}L_{21}^T &= L_{22}L_{22}^T. \end{aligned}$$

Concluding with the algorithm:

1. Partition $A = \left(\begin{array}{c|c} A_{11} & \star \\ \hline A_{21} & A_{22} \end{array}\right)$, being A_{11} a $b \times b$ submatrix.
2. $A_{11} := L_{11}$ such that $A_{11} = L_{11}L_{11}^T$ (Cholesky factor of A_{11}).
3. $A_{21} := L_{21} = A_{21}L_{11}^{-T}$.
4. $A_{22} := A_{22} - L_{21}L_{21}^{-T}$.
5. Continue recursively with $A = A_{22}$ with step 1.

Cholesky factorization. Right-looking variant (II)

This yields to equations:

$$\begin{aligned} L_{11}L_{11}^T &= A_{11}, \\ L_{21} &= A_{21}L_{11}^{-T}, \\ A_{22} - L_{21}L_{21}^T &= L_{22}L_{22}^T. \end{aligned}$$

Concluding with the algorithm:

1. Partition $A = \left(\begin{array}{c|c} A_{11} & \star \\ \hline A_{21} & A_{22} \end{array}\right)$, being A_{11} a $b \times b$ submatrix.
2. $A_{11} := L_{11}$ such that $A_{11} = L_{11}L_{11}^T$ (Cholesky factor of A_{11}).
3. $A_{21} := L_{21} = A_{21}L_{11}^{-T}$.
4. $A_{22} := A_{22} - L_{21}L_{21}^{-T}$.
5. Continue recursively with $A = A_{22}$ with step 1.

Using PLAPACK as a library

From algorithms to code

partition $A = \left(\begin{array}{c|c} A_{TL} & \star \\ \hline A_{BL} & A_{BR} \end{array}\right)$

where A_{TL} is 0×0

do until A_{BR} is 0×0

determine block size b

repartition

$$\left(\begin{array}{c|c} A_{TL} & \star \\ \hline A_{BL} & A_{BR} \end{array}\right) = \left(\begin{array}{c|c|c} A_{00} & \star & \star \\ \hline A_{10} & A_{11} & \star \\ \hline A_{20} & A_{21} & A_{22} \end{array}\right)$$

where A_{11} is $b \times b$

$A_{11} \leftarrow L_{11} = \text{Chol. fact.}(A_{11})$

$A_{21} \leftarrow L_{21} = A_{21} L_{11}^{-T}$

$A_{22} \leftarrow A_{22} - L_{21} L_{21}^{T}$

continue with

$$\left(\begin{array}{c|c} A_{TL} & \star \\ \hline A_{BL} & A_{BR} \end{array}\right) = \left(\begin{array}{c|c|c} A_{00} & \star & \star \\ \hline A_{10} & A_{11} & \star \\ \hline A_{20} & A_{21} & A_{22} \end{array}\right)$$

enddo

```
int PLA_Chol( int b, PLA_Obj A )
{
  PLA_Obj  ABR = NULL,
           A11 = NULL,      A21 = NULL;
  /* ... */

  /* View ABR = A */
  PLA_Obj_view_all( A, &ABR );

  while ( TRUE ) {
    /* Partition  ABR = / A11  ||   *  \
     *                  | =====  ===== |
     *                  \ A21  || ABR  /
     * where A11 is b x b                */
    PLA_Obj_split_4( ABR, b, b,
                     &A11, PLA_DUMMY,
                     &A21, &ABR );

    /* A11 := L11 = Cholesky Factor( A11 ) */
    PLA_Local_chol( PLA_LOWER_TRIANGULAR, A11 );

    /* Update A21 := L21 = A21 * inv( L11' ) */
    PLA_Trsm( PLA_SIDE_RIGHT, PLA_LOWER_TRIANGULAR,
              PLA_TRANSPOSE,  PLA_NONUNIT_DIAG,
              one, A11, A21 );

    /* Update A22 := A22 - L21 * L21' */
    PLA_Syrk( PLA_LOWER_TRIANGULAR, PLA_NO_TRANS,
              minus_one, A21,
              one,       ABR );
  }
  /* ... */
}
```

Using PLAPACK as a library

From algorithms to code

partition $A = \left(\begin{array}{c|c} A_{TL} & \star \\ \hline A_{BL} & A_{BR} \end{array}\right)$

where A_{TL} is 0×0

do until A_{BR} is 0×0

determine block size b

repartition

$$\left(\begin{array}{c|c} A_{TL} & \star \\ \hline A_{BL} & A_{BR} \end{array}\right) = \left(\begin{array}{c|c|c} A_{00} & \star & \star \\ \hline A_{10} & A_{11} & \star \\ \hline A_{20} & A_{21} & A_{22} \end{array}\right)$$

where A_{11} is $b \times b$

$A_{11} \leftarrow L_{11} = \text{Chol. fact.}(A_{11})$

$A_{21} \leftarrow L_{21} = A_{21} L_{11}^{-T}$

$A_{22} \leftarrow A_{22} - L_{21} L_{21}^{T}$

continue with

$$\left(\begin{array}{c|c} A_{TL} & \star \\ \hline A_{BL} & A_{BR} \end{array}\right) = \left(\begin{array}{c|c|c} A_{00} & \star & \star \\ \hline A_{10} & A_{11} & \star \\ \hline A_{20} & A_{21} & A_{22} \end{array}\right)$$

enddo

```
int PLA_Chol( int b, PLA_Obj A )
{
  PLA_Obj  ABR = NULL,
           A11 = NULL,      A21 = NULL;
  /* ... */

  /* View ABR = A */
  PLA_Obj_view_all( A, &ABR );

  while ( TRUE ) {
    /* Partition  ABR = / A11  ||   *  \
     *                 | =====  =====  |
     *                 \ A21  ||  ABR  /
     * where A11 is b x b                */
    PLA_Obj_split_4( ABR, b, b,
                     &A11, PLA_DUMMY,
                     &A21, &ABR );

    /* A11 := L11 = Cholesky Factor( A11 ) */
    PLA_Local_chol( PLA_LOWER_TRIANGULAR, A11 );

    /* Update A21 := L21 = A21 * inv( L11' ) */
    PLA_Trsm( PLA_SIDE_RIGHT, PLA_LOWER_TRIANGULAR,
              PLA_TRANSPOSE,  PLA_NONUNIT_DIAG,
              one, A11, A21 );

    /* Update A22 := A22 - L21 * L21' */
    PLA_Syrk( PLA_LOWER_TRIANGULAR, PLA_NO_TRANS,
              minus_one, A21,
              one,       ABR );
  }
  /* ... */
}
```

Using PLAPACK as a library. Cholesky code

```
1   int PLA_Chol( int b, PLA_Obj A )
2   {
3     PLA_Obj  ABR = NULL,
4              A11 = NULL,      A21 = NULL;
5     /* ... */
6
7     /* View ABR = A */
8     PLA_Obj_view_all( A, &ABR );
9
10    while ( TRUE ) {
11      /* Partition  ABR = / A11  ||   *  \
12       *                 | =====  ===== |
13       *                 \ A21  || ABR  /
14       * where A11 is b x b              */
15      PLA_Obj_split_4( ABR, b, b,
16                       &A11, PLA_DUMMY,
17                       &A21, &ABR );
18
19      /* A11 := L11 = Cholesky Factor( A11 ) */
20      PLA_Local_chol( PLA_LOWER_TRIANGULAR, A11 );
21
22      /* Update A21 := L21 = A21 * inv( L11' ) */
23      PLA_Trsm( PLA_SIDE_RIGHT, PLA_LOWER_TRIANGULAR,
24                PLA_TRANSPOSE,  PLA_NONUNIT_DIAG,
25                one, A11,
26                     A21 );
27
28      /* Update A22 := A22 - L21 * L21' */
29      PLA_Syrk( PLA_LOWER_TRIANGULAR, PLA_NO_TRANS,
30                minus_one, A21,
31                one,       ABR );
32    }
33    /* ... */
34  }
```

```
int CUPLA_Chol( int b, PLA_Obj A )
{
  PLA_Obj  ABR = NULL,
           A11 = NULL,      A21 = NULL;
  /* ... */

  /* View ABR = A */
  PLA_Obj_view_all( A, &ABR );

  while ( TRUE ) {
    /* Partition  ABR = / A11  ||   *  \
     *                 | =====  ===== |
     *                 \ A21  || ABR  /
     * where A11 is b x b              */
    PLA_Obj_split_4( ABR, b, b,
                     &A11, PLA_DUMMY,
                     &A21, &ABR );

    /* A11 := L11 = Cholesky Factor( A11 ) */
    CUPLA_Local_chol( PLA_LOWER_TRIANGULAR, A11 );

    /* Update A21 := L21 = A21 * inv( L11' ) */
    CUPLA_Trsm( PLA_SIDE_RIGHT, PLA_LOWER_TRIANGULAR,
                PLA_TRANSPOSE,  PLA_NONUNIT_DIAG,
                one, A11,
                     A21 );

    /* Update A22 := A22 - L21 * L21' */
    CUPLA_Syrk( PLA_LOWER_TRIANGULAR, PLA_NO_TRANS,
                minus_one, A21,
                one,       ABR );
  }
  /* ... */
}
```

Using PLAPACK as a library. Cholesky code

```
1   int PLA_Chol( int b, PLA_Obj A )
2   {
3     PLA_Obj  ABR = NULL,
4              A11 = NULL,      A21 = NULL;
5     /* ... */
6
7     /* View ABR = A */
8     PLA_Obj_view_all( A, &ABR );
9
10    while ( TRUE ) {
11      /* Partition  ABR = / A11  ||   *  \
12       *                 | =====  ===== |
13       *                 \ A21  || ABR  /
14       * where A11 is b x b              */
15      PLA_Obj_split_4( ABR, b, b,
16                       &A11, PLA_DUMMY,
17                       &A21, &ABR );
18
19      /* A11 := L11 = Cholesky Factor( A11 ) */
20      PLA_Local_chol( PLA_LOWER_TRIANGULAR, A11 );
21
22      /* Update A21 := L21 = A21 * inv( L11' ) */
23      PLA_Trsm( PLA_SIDE_RIGHT, PLA_LOWER_TRIANGULAR,
24                PLA_TRANSPOSE,  PLA_NONUNIT_DIAG,
25                one, A11,
26                     A21 );
27
28      /* Update A22 := A22 - L21 * L21' */
29      PLA_Syrk( PLA_LOWER_TRIANGULAR, PLA_NO_TRANS,
30                minus_one, A21,
31                one,       ABR );
32    }
33    /* ... */
34  }
```

```
int CUPLA_Chol( int b, PLA_Obj A )
{
  PLA_Obj  ABR = NULL,
           A11 = NULL,      A21 = NULL;
  /* ... */

  /* View ABR = A */
  PLA_Obj_view_all( A, &ABR );

  while ( TRUE ) {
    /* Partition  ABR = / A11  ||   *  \
     *                 | =====  ===== |
     *                 \ A21  || ABR  /
     * where A11 is b x b              */
    PLA_Obj_split_4( ABR, b, b,
                     &A11, PLA_DUMMY,
                     &A21, &ABR );

    /* A11 := L11 = Cholesky Factor( A11 ) */
    CUPLA_Local_chol( PLA_LOWER_TRIANGULAR, A11 );

    /* Update A21 := L21 = A21 * inv( L11' ) */
    CUPLA_Trsm( PLA_SIDE_RIGHT, PLA_LOWER_TRIANGULAR,
                PLA_TRANSPOSE,  PLA_NONUNIT_DIAG,
                one, A11,
                     A21 );

    /* Update A22 := A22 - L21 * L21' */
    CUPLA_Syrk( PLA_LOWER_TRIANGULAR, PLA_NO_TRANS,
                minus_one, A21,
                one,       ABR );
  }
  /* ... */
}
```

GPU acceleration of PLAPACK

How to accelerate PLAPACK?

- *Naive implementation:*
  - Objects created in RAM.
  - GPU $\rightleftarrows$ CPU transfers bound to calculations.
  - Intercept calls to local BLAS and transfer data to/from GPU.
  - Straightforward implementation.
- *Tuned implementation:*
  - Objects created in GPU.
  - GPU $\rightleftarrows$ CPU transfers bound to communications.
  - Only transfer to RAM when necessary (communication).
  - Creation of objects involves allocation on GPUs.
  - Modification of communication layer in PLAPACK (`PLA_Copy`, `PLA_Reduce`).

GPU acceleration of PLAPACK

How to accelerate PLAPACK?

- *Naive implementation:*
  - Objects created in RAM.
  - GPU $\rightleftarrows$ CPU transfers bound to calculations.
  - Intercept calls to local BLAS and transfer data to/from GPU.
  - Straightforward implementation.
- *Tuned implementation:*
  - Objects created in GPU.
  - GPU $\rightleftarrows$ CPU transfers bound to communications.
  - Only transfer to RAM when necessary (communication).
  - Creation of objects involves allocation on GPUs.
  - Modification of communication layer in PLAPACK (`PLA_Copy`, `PLA_Reduce`).

PLAPACK infrastructure

PLAPACK layered design

<table>
<tr><td colspan="5">Naive User Application</td><td>application layer</td></tr>
<tr><td rowspan="3">Application Program Interface
(PLA_API)</td><td colspan="4">Higher Level Global LA Routines</td><td rowspan="2">library layer</td></tr>
<tr><td colspan="4">PLA_Global BLAS</td></tr>
<tr><td colspan="2">PLA_Copy/Reduce</td><td>LA object manipulation</td><td>PLA_Local BLAS</td><td>PLAPACK abstraction layer</td></tr>
<tr><td>MMPI</td><td>PLA/MPI interface</td><td>PLA_malloc</td><td>PBMD Templates</td><td>PLA/ BLAS interface</td><td>machine/distribution independent layer</td></tr>
<tr><td colspan="2">Message-Passing Interface</td><td>malloc</td><td>cartesian distributions</td><td>vendor BLAS</td><td>machine/distribution specific layer</td></tr>
</table>

Necessary changes to use GPU

1. PLA_Local_BLAS: NVIDIA CUBLAS.
2. LA object manipulation: management of objects in GPU memory.
3. Communication layer: transfers to RAM prior to communications.

PLAPACK infrastructure

PLAPACK layered design

<table>
<tr><td colspan="5">Naive User Application</td><td>application layer</td></tr>
<tr><td rowspan="3">Application Program Interface
(PLA_API)</td><td colspan="4">Higher Level Global LA Routines</td><td rowspan="2">library layer</td></tr>
<tr><td colspan="4">PLA_Global BLAS</td></tr>
<tr><td colspan="2">PLA_Copy/Reduce</td><td>LA object manipulation</td><td>PLA_Local BLAS</td><td>PLAPACK abstraction layer</td></tr>
<tr><td>MMPI</td><td>PLA/MPI interface</td><td>PLA_malloc</td><td>PBMD Templates</td><td>PLA/ BLAS interface</td><td>machine/distribution independent layer</td></tr>
<tr><td colspan="2">Message-Passing Interface</td><td>malloc</td><td>cartesian distributions</td><td>vendor BLAS</td><td>machine/distribution specific layer</td></tr>
</table>

Necessary changes to use GPU

1. PLA_Local_BLAS: NVIDIA CUBLAS.
2. LA object manipulation: management of objects in GPU memory.
3. Communication layer: transfers to RAM prior to communications.

LA object manipulation. Cholesky driver using GPU

```
1   int main( void ){
2     /* ... */
3     // Object creation on GPU
4     CUPLA_Matrix_create(MPI_FLOAT, size, size, templ,
5                         PLA_ALIGN_FIRST, PLA_ALIGN_FIRST,
6                         &A_GPU);
7
8     // Object initialization on GPU
9     CUPLA_Obj_set_to_SPD_random( A_GPU );
10
11    /* ... */
12
13    // Cholesky factorization on GPU
14    CUPLA_Chol( nb_alg, A_GPU );
15    /* ... */
16
17    // Object destruction on GPU
18    CUPLA_Obj_free( &A_GPU );
19  }
```

- `CUPLA_Matrix_create` creates a L. A. object on GPU.
- `CUPLA_Obj_free` destroys an object on GPU.
- `CUPLA_Chol` operates with objects created on GPU.

Communication layer

Communication is restricted to routines `PLA_Copy` and `PLA_Reduce`:

int PLA_Copy (PLA_Obj obj_from, PLA_Obj obj_to)

Purpose: Copy contents between linear algebra objects.

IN	obj_from	Object to be copied
IN/OUT	obj_to	Object into which to copy

int PLA_Reduce (PLA_Obj obj_from, MPI_OP op, PLA_Obj obj_to)

Purpose: Reduce using `op` the contents in the duplicated object given by `obj_from` and overwrite `obj_to` with the result.

IN	obj_from	Object to be reduced
IN	op	Reduce operator to be used
IN/OUT	obj_to	Object into which to reduce result

Communication layer

Communication is restricted to routines `PLA_Copy` and `PLA_Reduce`:

int PLA_Copy (PLA_Obj obj_from, PLA_Obj obj_to)

Purpose: Copy contents between linear algebra objects.

IN	obj_from	Object to be copied
IN/OUT	obj_to	Object into which to copy

int PLA_Reduce (PLA_Obj obj_from, MPI_OP op, PLA_Obj obj_to)

Purpose: Reduce using `op` the contents in the duplicated object given by `obj_from` and overwrite `obj_to` with the result.

IN	obj_from	Object to be reduced
IN	op	Reduce operator to be used
IN/OUT	obj_to	Object into which to reduce result

Modifications to PLA_Copy (I)

```
1   /* PLAPACK PLA_Copy between column aligned matrices */
2
3   while ( TRUE ){
4     PLA_Obj_split_size( from_cur, PLA_SIDE_TOP, &size_from, &owner_from );
5     PLA_Obj_split_size( to_cur,   PLA_SIDE_TOP, &size_to,   &owner_to);
6
7     if ( 0 == ( size = min( size_from, size_to ) ) ) break;
8
9     PLA_Obj_horz_split_2( from_cur, size, &from_1, &from_cur );
10    PLA_Obj_horz_split_2( to_cur,   size, &to_1,   &to_cur );
11
12    if ( myrow == owner_from && owner_from == owner_to ){
13      PLA_Local_copy( from_1, to_1 );
14    }
15    else{
16      if ( myrow == owner_from ) {
17        PLA_Obj_get_local_contents( from_1, PLA_NO_TRANS, &dummy, &dummy,
18                                    buffer_temp, size, 1 );
19
20        MPI_Send( BF( buffer_temp ), size * local_width, datatype,
21                  owner_to, 0, comm_col );
22      }
23      if ( myrow == owner_to ) {
24        MPI_Recv( BF( buffer_temp ), size * local_width, datatype,
25                  owner_from, MPI_ANY_TAG, comm_col, &status );
26
27        PLA_Obj_set_local_contents( PLA_NO_TRANS, size, local_width,
28                                    buffer_temp, size, 1, to_1 );
29      }
30    }
31  }
32  PLA_free( buffer_temp );
```

Modifications to PLA_Copy (II)

```
1   /* PLAPACK PLA_Obj_get_local_contents */
2
3   int PLA_Obj_get_local_contents    (
4           PLA_Obj    obj,
5           int        trans,
6           int        *rows_in_buf,
7           int        *cols_in_buf,
8           void       *buf,
9           int        ldim_buf,
10          int        stride_buf)
11
12    /****************************/
13
14    for ( j=0; j<n; j++ ){
15      tmp_local = buf_local + j*ldim_buf*typesize;
16      tmp_obj   = buf_obj   + j*ldim*typesize;
17      memcpy( tmp_local, tmp_obj, m*typesize );
18    }
19
20    /****************************/
```

```
/* CUPLAPACK PLA_Obj_get_local_contents */

int CUPLA_Obj_get_local_contents    (
        PLA_Obj    obj,
        int        trans,
        int        *rows_in_buf,
        int        *cols_in_buf,
        void       *buf,
        int        ldim_buf,
        int        stride_buf)

  /****************************/

  cublasGetMatrix( m, n, typesize,
                   buf_obj,    ldim,
                   buf_local,  ldim_buf );

  /****************************/
```

Necessary GPU $\rightleftarrows$ CPU transfers

Data is transferred to RAM prior to a communication.
Data is transferred to GPU after each communication.

Modifications to PLA_Copy (II)

```
1   /* PLAPACK PLA_Obj_get_local_contents */
2
3   int PLA_Obj_get_local_contents    (
4           PLA_Obj    obj,
5           int        trans,
6           int        *rows_in_buf,
7           int        *cols_in_buf,
8           void       *buf,
9           int        ldim_buf,
10          int        stride_buf)
11
12    /****************************/
13
14    for ( j=0; j<n; j++ ){
15      tmp_local = buf_local + j*ldim_buf*typesize;
16      tmp_obj   = buf_obj   + j*ldim*typesize;
17      memcpy( tmp_local, tmp_obj, m*typesize );
18    }
19
20    /****************************/
```

```
/* CUPLAPACK PLA_Obj_get_local_contents */

int CUPLA_Obj_get_local_contents    (
        PLA_Obj    obj,
        int        trans,
        int        *rows_in_buf,
        int        *cols_in_buf,
        void       *buf,
        int        ldim_buf,
        int        stride_buf)

  /****************************/

  cublasGetMatrix( m, n, typesize,
                   buf_obj,    ldim,
                   buf_local, ldim_buf );

  /****************************/
```

Necessary GPU $\rightleftarrows$ CPU transfers

Data is transferred to RAM prior to a communication.
Data is transferred to GPU after each communication.

The LONGHORN visualization cluster

	Per Node	Per System (256 nodes)
Number of cores	8	2,048
CPU	Intel Xeon Nehalem @ 2.53 GHz	
Available memory	48 Gbytes	13.5 TBytes
Interconnection network	QDR Infiniband	
Graphics system	128 NVIDIA Quadro Plex S4s	
GPU	2 x NVIDIA Quadro FX5800	512 x NVIDIA Quadro FX5800
Interconnection bus	PCIExpress 2.0 (8x)	
Available video memory	8 Gbytes DDR3	2 TBytes DDR3
Peak performance (SP)	161.6 GFLOPS	41.40 TFLOPS
Peak performance (DP)	80.8 GFLOPS	20.70 TFLOPS

Table: Detailed features of the LONGHORN cluster.

PLAPACK: version R32.

BLAS: NVIDIA CUBLAS 2.2, MKL 10.1.

MPI: MVAPICH 1.4.

GEMM: performance

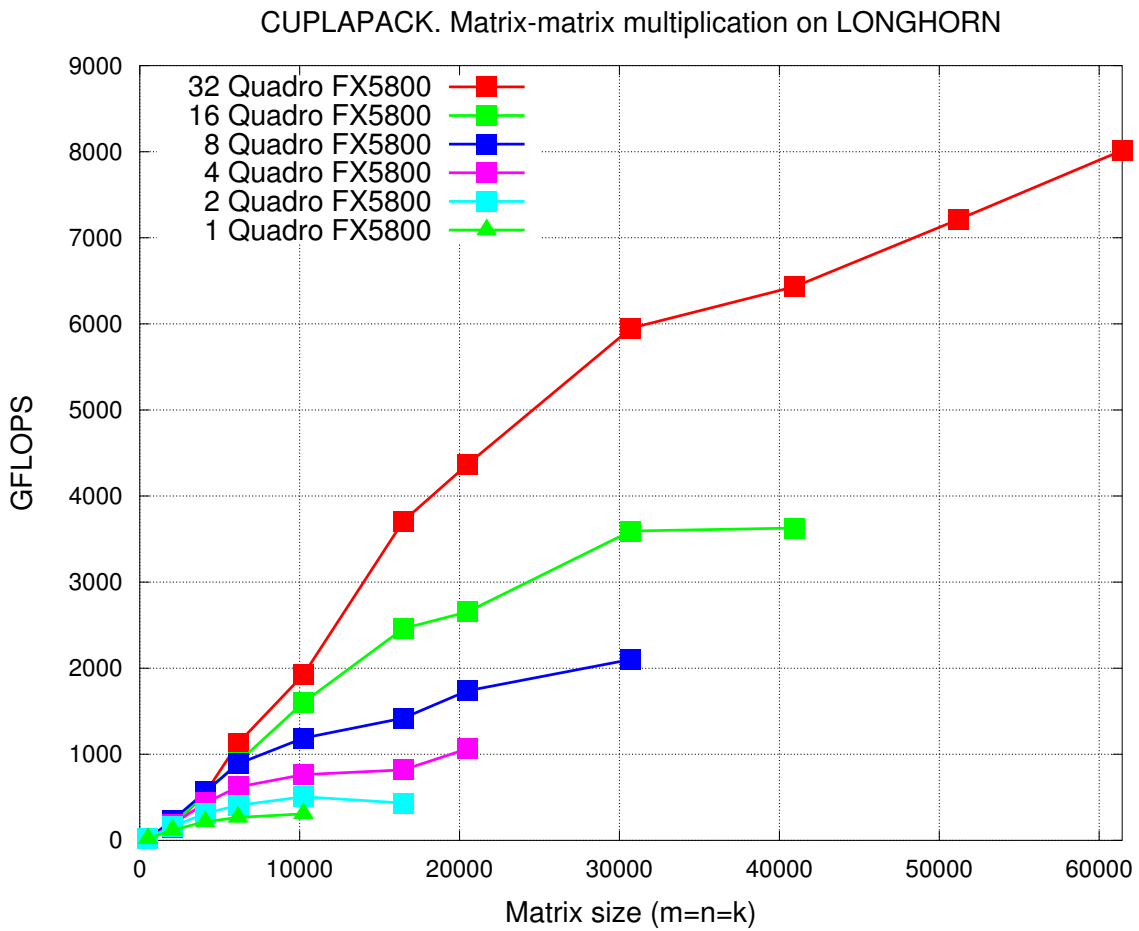


Figure: Performance of CUPLAPACK for the matrix-matrix multiplication on LONGHORN.

Cholesky factorization: performance

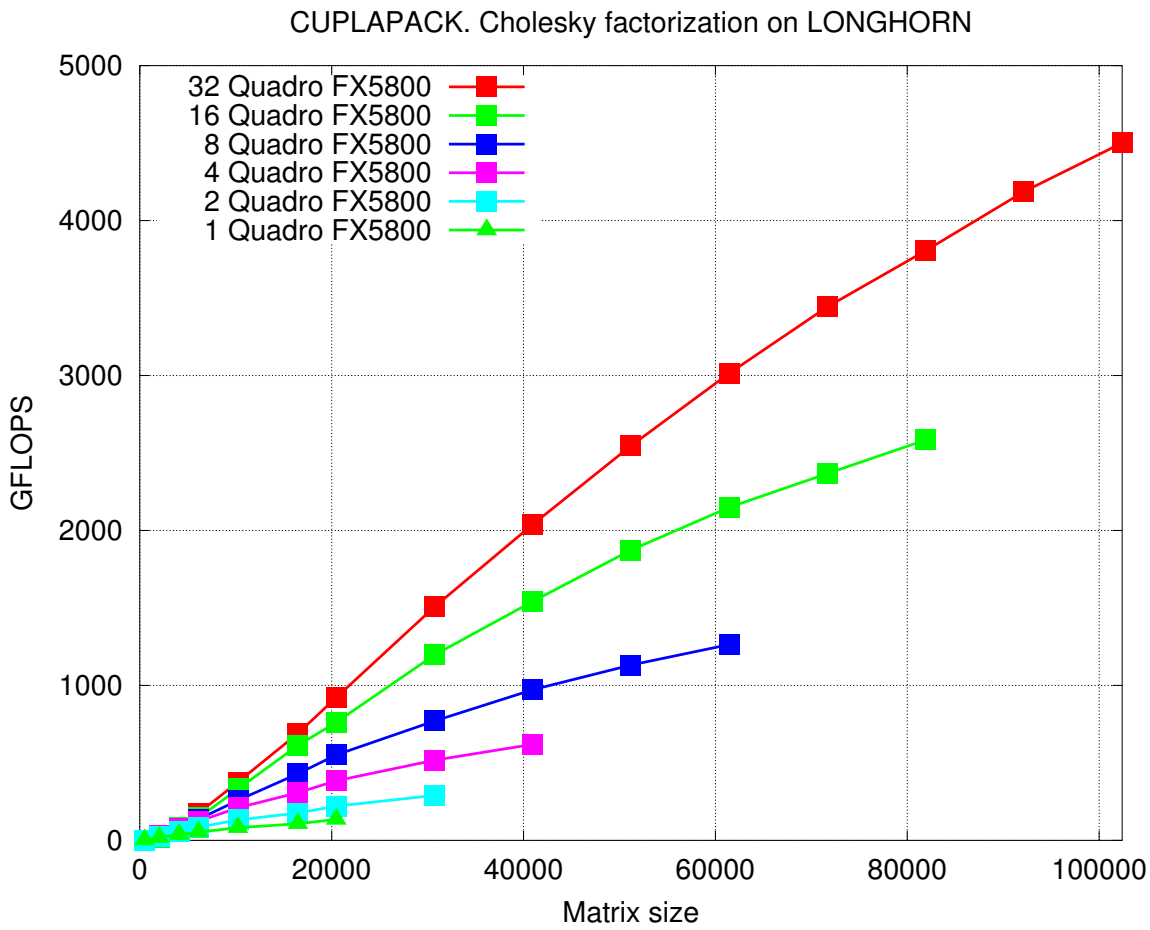


Figure: Performance of CUPLAPACK for the Cholesky factorization on LONGHORN.

GEMM: scalability

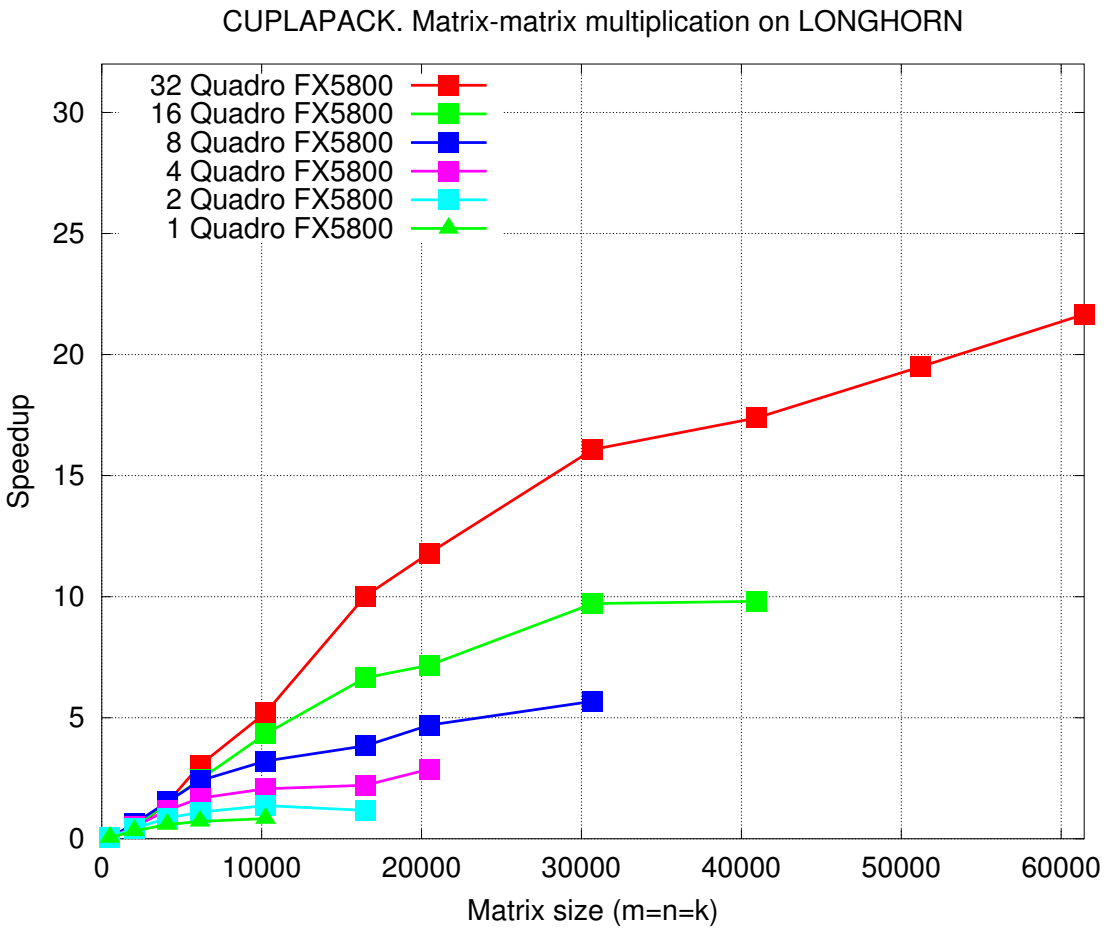


Figure: Scalability of the PLAPACK-based codes for the matrix-matrix multiplication on LONGHORN. The reference is the performance of CUBLAS SGEMM on one GPU.

GEMM: PLAPACK vs. CUPLAPACK

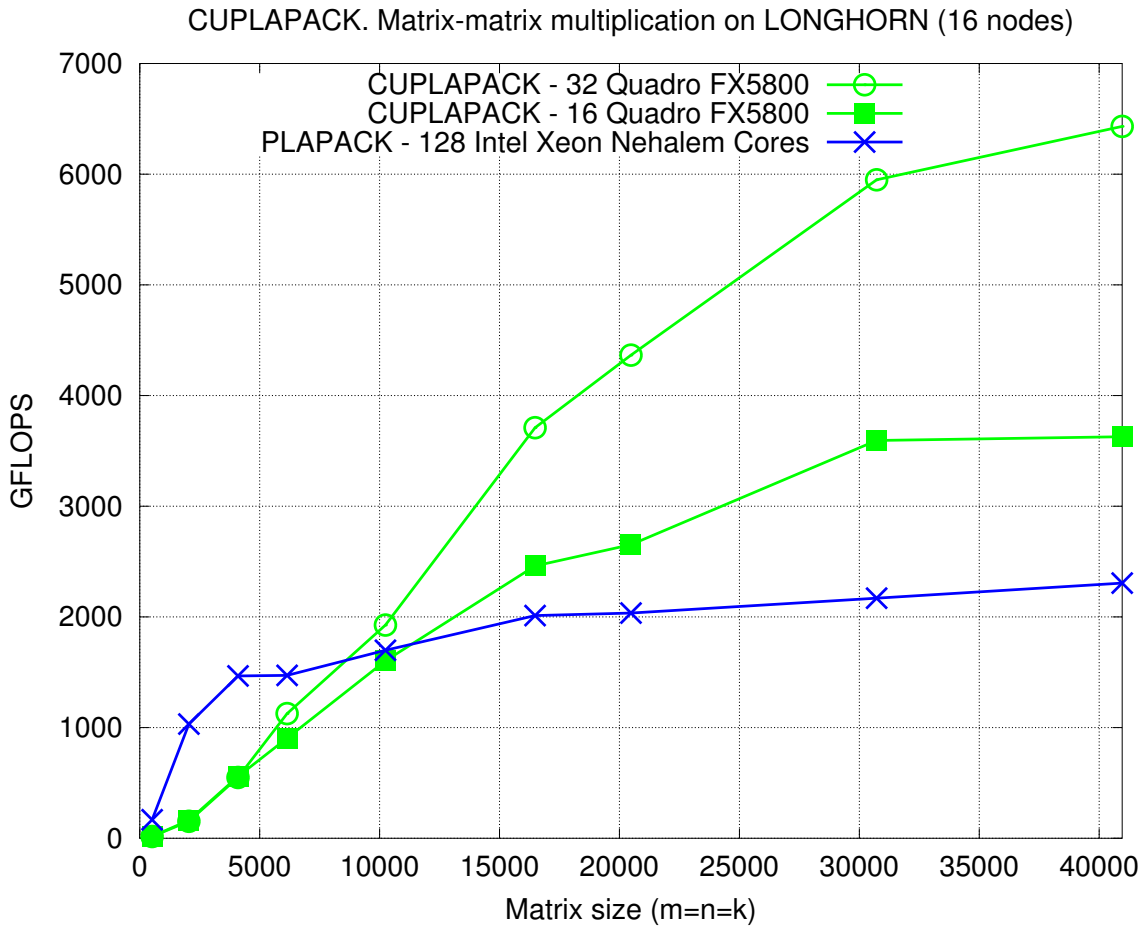


Figure: GEMM on 16 nodes of LONGHORN. PLAPACK vs CUPLAPACK.

Cholesky factorization: PLAPACK vs. CUPLAPACK

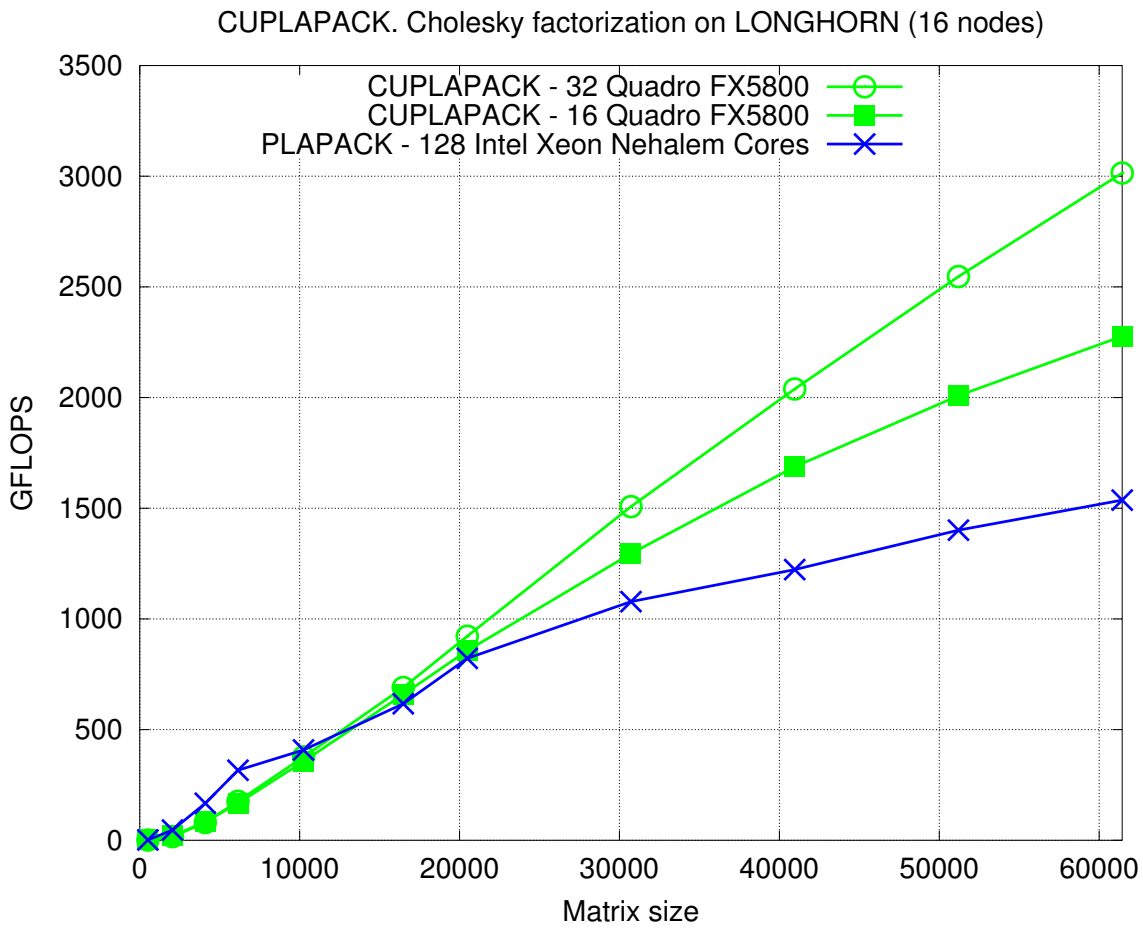


Figure: Cholesky factorization on 16 nodes of LONGHORN. PLAPACK vs CUPLAPACK.

Multiple GPUs per node vs. One GPU per node

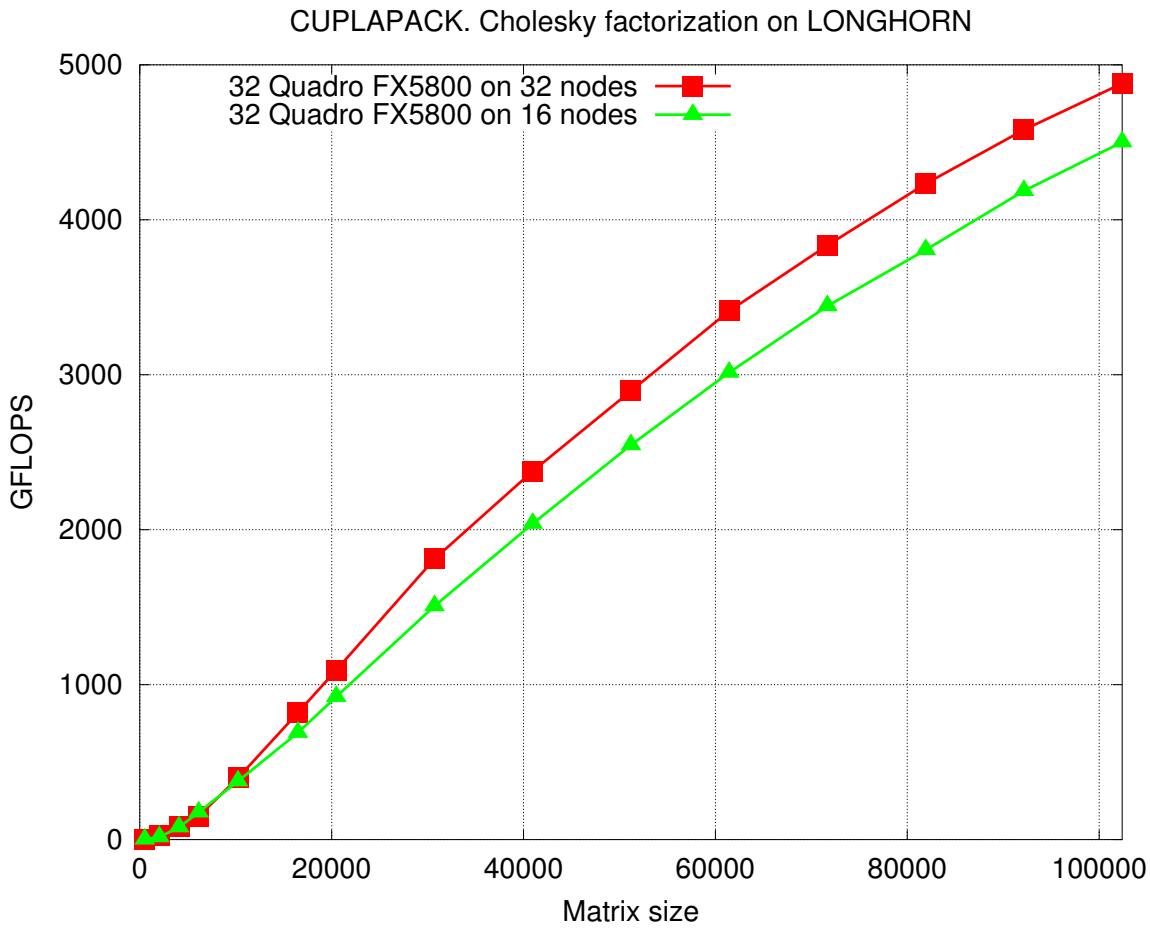


Figure: Cholesky factorization on 32 GPUs of LONGHORN, using **one** or **two** GPUs per node.

Conclusions and future work

Conclusions

- Retarget of PLAPACK to hybrid architectures.
- Transparent for programmers and library developers.
- Benefits from the layered design of PLAPACK.
- Remarkable performance and scalability on large GPU clusters.

Future work

- Port the Elemental framework to GPUs.
- Out-of-core + GPU support to deal with even larger problems.

Acknowledgements

Thank you. . .

Questions?

We thank the `Texas Advanced Computing Center` for granting access to the platform where the experiments were conducted[a].

[a] http://www.tacc.utexas.edu/resources/visualization/#longhorn

TACC TEXAS ADVANCED COMPUTING CENTER
Powering Discoveries That Change The World
THE UNIVERSITY OF TEXAS AT AUSTIN