FLAG/C

Una API para computación matricial sobre GPUs

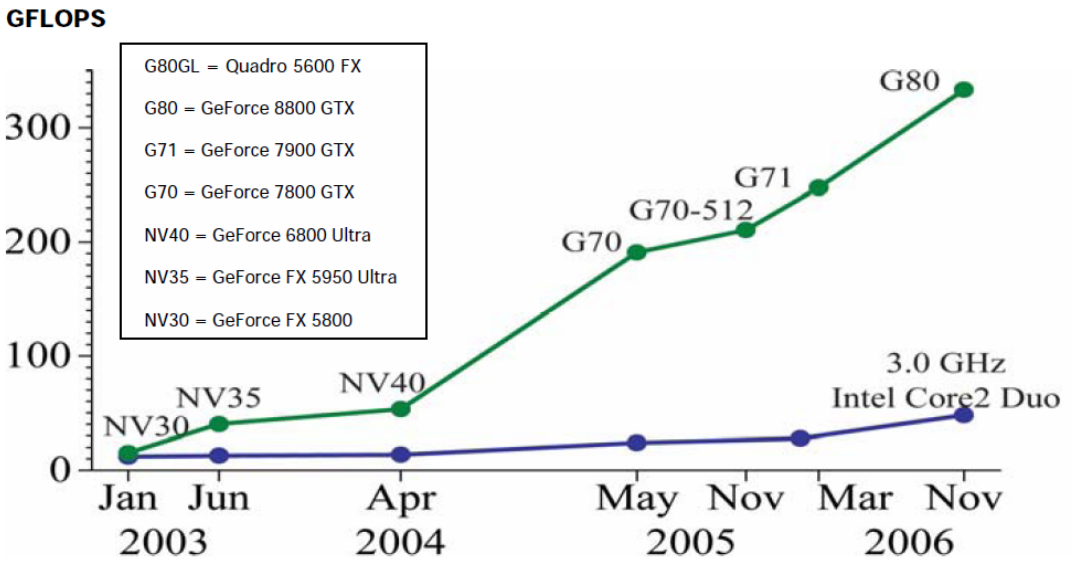
M. Jesús Zafont Alberto Martín
Francisco Igual Enrique S. Quintana-Ortí

High Performance Computing & Architectures Group
Universitat Jaume I de Castellón (Spain)

Motivación (I)

Los procesadores gráficos actuales se han convertido en una alternativa real y de bajo coste para HPC

Rendimiento



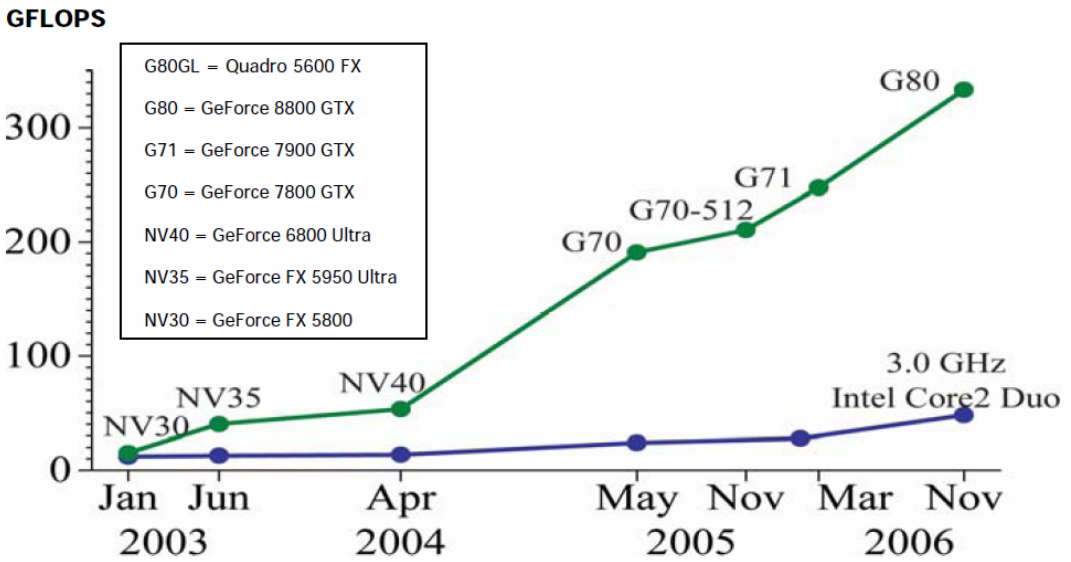
Facilidad de programación

- Hardware:
  - Mayor funcionalidad
- Software:
  - Cg/OpenGL vs. CUDA
  - CUBLAS/CUFFT
  - ...

Motivación (I)

Los procesadores gráficos actuales se han convertido en una alternativa real y de bajo coste para HPC

Rendimiento



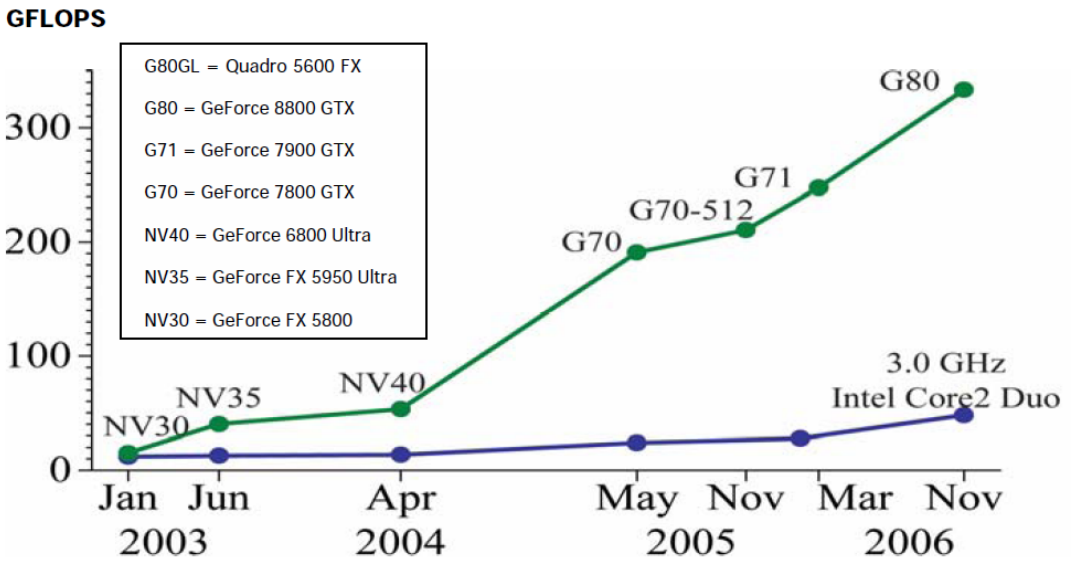
Facilidad de programación

- Hardware:
  - Mayor funcionalidad
- Software:
  - Cg/OpenGL vs. CUDA
  - CUBLAS/CUFFT
  - ...

Motivación (I)

Los procesadores gráficos actuales se han convertido en una alternativa real y de bajo coste para HPC

Rendimiento



Facilidad de programación

- Hardware:
  - Mayor funcionalidad
- Software:
  - Cg/OpenGL vs. CUDA
  - CUBLAS/CUFFT
  - ...

Motivación (II)

Objetivos

1. Posibilitar la creación de códigos robustos
2. Reducir tiempos de desarrollo
3. Reducir tiempos de ejecución

Nuestra propuesta: FLAG/C

Interfaz de programación para operaciones de álgebra lineal densa sobre GPUs

- **Códigos robustos**: metodología FLAME
- **Facilidad de uso**: oculta al programador la arquitectura subyacente
- **Prestaciones**: basado en GPUs

Motivación (II)

Objetivos

1. Posibilitar la creación de códigos robustos
2. Reducir tiempos de desarrollo
3. Reducir tiempos de ejecución

Nuestra propuesta: FLAG/C

Interfaz de programación para operaciones de álgebra lineal densa sobre GPUs

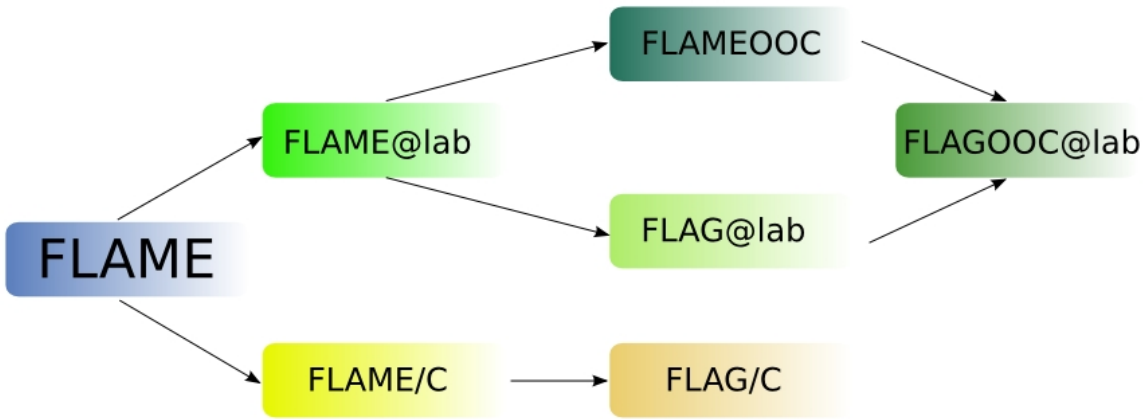
- **Códigos robustos**: metodología FLAME
- **Facilidad de uso**: oculta al programador la arquitectura subyacente
- **Prestaciones**: basado en GPUs

Índice de contenidos

La API FLAG/C

FLAG/C

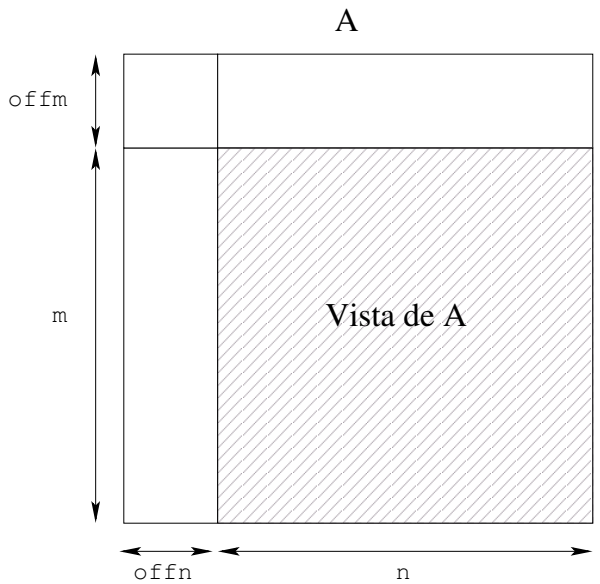
- FLAG/C: API de alto nivel, escrita en lenguaje C, para el desarrollo rápido de códigos eficientes de álgebra lineal densa
- Posibilidad de ser combinado con cualquier tipo de acelerador: GPUs, tarjetas ClearSpeed, Cell B.E.
- Requisito: implementación eficiente de BLAS disponible para la arquitectura



Objetos en FLAG/C

Objetos en FLAG/C

- Representación de una matriz en la memoria del acelerador
- FLAG/C permite al usuario:
  1. Crear, destruir y consultar las propiedades de un objeto
  2. Inicializar el contenido de un objeto
  3. Definir *vistas*
  4. Operar con objetos a través de BLAS



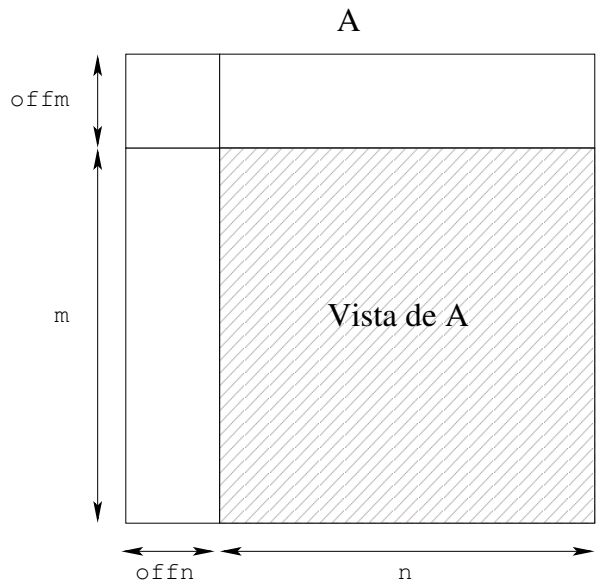
Vistas

- Referencias a bloques dentro de un objeto
- Flexibilidad para manipular matrices y bloques de matrices
- Sin coste adicional de almacenamiento

Objetos en FLAG/C

Objetos en FLAG/C

- Representación de una matriz en la memoria del acelerador
- FLAG/C permite al usuario:
  1. Crear, destruir y consultar las propiedades de un objeto
  2. Inicializar el contenido de un objeto
  3. Definir *vistas*
  4. Operar con objetos a través de BLAS



Vistas

- Referencias a bloques dentro de un objeto
- Flexibilidad para manipular matrices y bloques de matrices
- Sin coste adicional de almacenamiento

Estructura típica de un programa FLAG/C (I)

Inicialización

`FLA_Init()` / `FLA_Finalize()`

Inicialización / finalización necesaria del entorno

↓

Creación de Objetos

`FLAG_Obj_create( FLA_TYPE, int m, int n, FLA_Obj * )`

Reserva espacio para el objeto en la memoria de la GPU

↓

Estructura típica de un programa FLAG/C (I)

Inicialización

`FLA_Init()` / `FLA_Finalize()`

Inicialización / finalización necesaria del entorno

↓

Creación de Objetos

```
FLAG_Obj_create( FLA_TYPE, int m, int n, FLA_Obj * )
```

Reserva espacio para el objeto en la memoria de la GPU

↓

Estructura típica de un programa FLAG/C (II)

↓

Transferencias CPU -> GPU

`FLAG_Obj_set_with_buffer( void* buffer, FLA_Obj A )`

Inicializa los elementos del objeto A en memoria de GPU con una copia del contenido del *buffer* alojado en memoria central

↓

Computación Matricial

Implementación BLAS completa

`FLAG_Gemm( FLAG_Trans transA, FLAG_Trans transB, FLA_Obj alpha, FLAG_Obj A, FLAG_Obj B, FLA_Obj beta, FLAG_Obj C )`

Estructura típica de un programa FLAG/C (II)

↓

Transferencias CPU -> GPU

`FLAG_Obj_set_with_buffer( void* buffer, FLA_Obj A )`

Inicializa los elementos del objeto A en memoria de GPU con una copia del contenido del *buffer* alojado en memoria central

↓

Computación Matricial

Implementación BLAS completa

```
FLAG_Gemm( FLAG_Trans transA, FLAG_Trans transB,
FLA_Obj alpha, FLAG_Obj A, FLAG_Obj B, FLA_Obj beta,
FLAG_Obj C  )
```

Estructura típica de un programa FLAG/C (III)

↓

Utilidades

```
FLAG_Obj_show( char *s1, FLAG_Obj A, char * format,
                                    char *s2  )
```

Muestra el contenido del objeto A

↓

Transferencias GPU -> CPU

```
FLAG_Obj_transfer_to_FLA_Obj( FLAG_Obj A,FLA_Obj B  )
```

Inicializa los elementos de un objeto B con una copia del contenido de A, alojado en GPU

Estructura típica de un programa FLAG/C (III)

↓

Utilidades

```
FLAG_Obj_show( char *s1, FLAG_Obj A, char * format,
                char *s2  )
```

Muestra el contenido del objeto A

↓

Transferencias GPU -> CPU

```
FLAG_Obj_transfer_to_FLA_Obj( FLAG_Obj A,FLA_Obj B  )
```

Inicializa los elementos de un objeto B con una copia del contenido de A, alojado en GPU

Example: $C = A^T \cdot B$

```
FLAG_Init();

FLAG_Obj AObj, BObj, CObj;

FLAG_Obj_create( 'FLAG_FLOAT', n, n, &AObj );
FLAG_Obj_create( 'FLAG_FLOAT', n, n, &BObj );
FLAG_Obj_create( 'FLAG_FLOAT', n, n, &CObj );

FLAG_Obj_set_with_buffer( (void *) A, &AObj );
FLAG_Obj_set_with_buffer( (void *) B, &BObj );
FLAG_Obj_set_with_buffer( (void *) C, &CObj );

FLAG_Gemm( 'FLAG_TRANSPOSE',...
           'FLAG_NO_TRANSPOSE',...
           1.0, &Aobj, &Bobj, 0.0, &Cobj );

FLAG_Obj_show( "C = [", Cobj, "%f", "]" );

FLAG_Obj_free( &Aobj );
FLAG_Obj_free( &Bobj );
FLAG_Obj_free( &Cobj );

FLAG_Finalize();
```

Inicialización

Creación de objetos

Transferencia CPU -> GPU

Operaciones BLAS en GPU

Liberación de espacio en memoria de GPU

Example: $C = A^T \cdot B$

```
FLAG_Init();

FLAG_Obj AObj, BObj, CObj;

FLAG_Obj_create( 'FLAG_FLOAT', n, n, &AObj );
FLAG_Obj_create( 'FLAG_FLOAT', n, n, &BObj );
FLAG_Obj_create( 'FLAG_FLOAT', n, n, &CObj );

FLAG_Obj_set_with_buffer( (void *) A, &AObj );
FLAG_Obj_set_with_buffer( (void *) B, &BObj );
FLAG_Obj_set_with_buffer( (void *) C, &CObj );

FLAG_Gemm( 'FLAG_TRANSPOSE',...
           'FLAG_NO_TRANSPOSE',...
           1.0, &Aobj, &Bobj, 0.0, &Cobj );

FLAG_Obj_show( "C = [", Cobj, "%f", "]" );

FLAG_Obj_free( &Aobj );
FLAG_Obj_free( &Bobj );
FLAG_Obj_free( &Cobj );

FLAG_Finalize();
```

Inicialización

Creación de objetos

Transferencia CPU -> GPU

Operaciones BLAS en GPU

Liberación de espacio en memoria de GPU

Example: $C = A^T \cdot B$

```
FLAG_Init();

FLAG_Obj AObj, BObj, CObj;

FLAG_Obj_create( 'FLAG_FLOAT', n, n, &AObj );
FLAG_Obj_create( 'FLAG_FLOAT', n, n, &BObj );
FLAG_Obj_create( 'FLAG_FLOAT', n, n, &CObj );

FLAG_Obj_set_with_buffer( (void *) A, &AObj );
FLAG_Obj_set_with_buffer( (void *) B, &BObj );
FLAG_Obj_set_with_buffer( (void *) C, &CObj );

FLAG_Gemm( 'FLAG_TRANSPOSE',...
           'FLAG_NO_TRANSPOSE',...
           1.0, &Aobj, &Bobj, 0.0, &Cobj );

FLAG_Obj_show( "C = [", Cobj, "%f", "]" );

FLAG_Obj_free( &Aobj );
FLAG_Obj_free( &Bobj );
FLAG_Obj_free( &Cobj );

FLAG_Finalize();
```

Inicialización

Creación de objetos

Transferencia CPU -> GPU

Operaciones BLAS en GPU

Liberación de espacio en memoria de GPU

Example: $C = A^T \cdot B$

```
FLAG_Init();

FLAG_Obj AObj, BObj, CObj;

FLAG_Obj_create( 'FLAG_FLOAT', n, n, &AObj );
FLAG_Obj_create( 'FLAG_FLOAT', n, n, &BObj );
FLAG_Obj_create( 'FLAG_FLOAT', n, n, &CObj );

FLAG_Obj_set_with_buffer( (void *) A, &AObj );
FLAG_Obj_set_with_buffer( (void *) B, &BObj );
FLAG_Obj_set_with_buffer( (void *) C, &CObj );

FLAG_Gemm( 'FLAG_TRANSPOSE',...
           'FLAG_NO_TRANSPOSE',...
           1.0, &Aobj, &Bobj, 0.0, &Cobj );

FLAG_Obj_show( "C = [", Cobj, "%f", "]" );

FLAG_Obj_free( &Aobj );
FLAG_Obj_free( &Bobj );
FLAG_Obj_free( &Cobj );

FLAG_Finalize();
```

Inicialización

Creación de objetos

Transferencia CPU -> GPU

Operaciones BLAS en GPU

Liberación de espacio en memoria de GPU

Example: $C = A^T \cdot B$

```
FLAG_Init();

FLAG_Obj AObj, BObj, CObj;

FLAG_Obj_create( 'FLAG_FLOAT', n, n, &AObj );
FLAG_Obj_create( 'FLAG_FLOAT', n, n, &BObj );
FLAG_Obj_create( 'FLAG_FLOAT', n, n, &CObj );

FLAG_Obj_set_with_buffer( (void *) A, &AObj );
FLAG_Obj_set_with_buffer( (void *) B, &BObj );
FLAG_Obj_set_with_buffer( (void *) C, &CObj );

FLAG_Gemm( 'FLAG_TRANSPOSE',...
           'FLAG_NO_TRANSPOSE',...
           1.0, &Aobj, &Bobj, 0.0, &Cobj );

FLAG_Obj_show( "C = [", Cobj, "%f", "]" );

FLAG_Obj_free( &Aobj );
FLAG_Obj_free( &Bobj );
FLAG_Obj_free( &Cobj );

FLAG_Finalize();
```

Inicialización

Creación de objetos

Transferencia CPU -> GPU

Operaciones BLAS en GPU

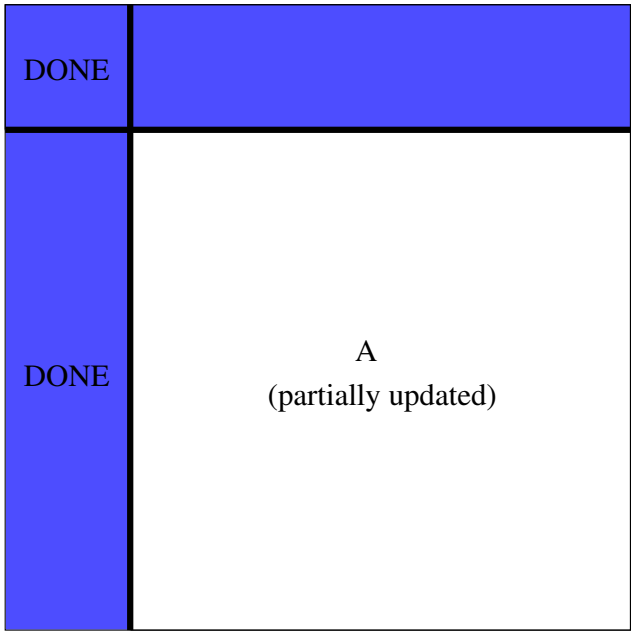
Liberación de espacio en memoria de GPU

FLAME

FLAME

- FLAME: desarrollo de bibliotecas robustas y eficientes con un alto nivel de abstracción
- Alto nivel de abracción: oculta detalles de implementación
- No es únicamente una biblioteca:
  - Notación para expresar algoritmos de álgebra lineal densa
  - Metodología para derivación sistemática de algoritmos
  - Interfaces de programación (APIs) para representar algoritmos en forma de código
  - Herramientas para facilitar el proceso de derivación algorítmica
- Ejemplo: factorización de Cholesky

Factorización de Cholesky con FLAME



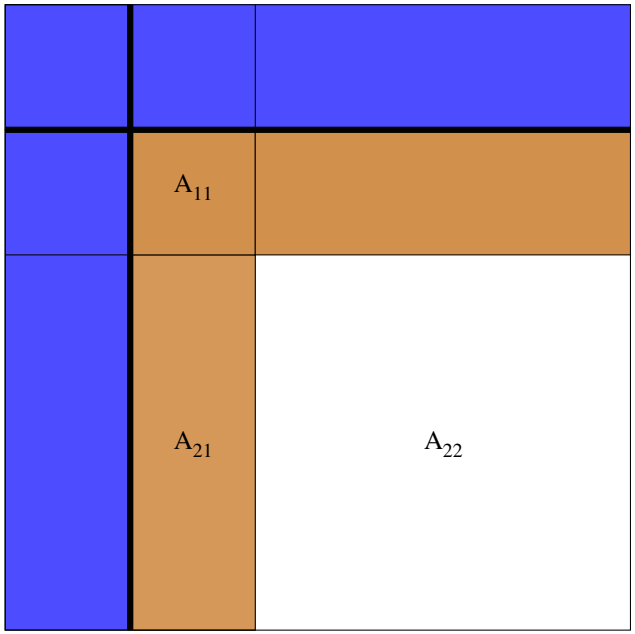
Definición

Dada $A \rightarrow n \times n$ s.p.d.

$$A = LL^T$$

con $L \rightarrow n \times n$ triangular inferior

Factorización de Cholesky con FLAME

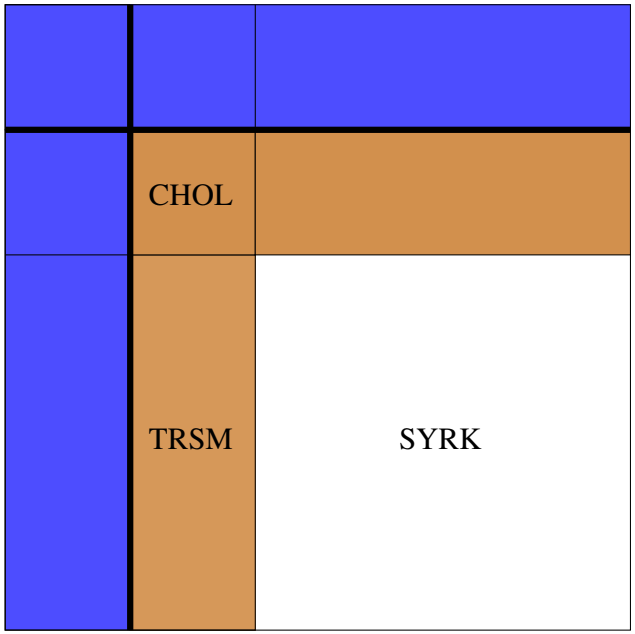


Repartition

$$\begin{pmatrix} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{pmatrix} \rightarrow \left(\begin{array}{c|c|c} A_{00} & A_{01} & A_{02} \\ \hline A_{10}^{\mathrm{T}} & A_{11} & A_{12}^{\mathrm{T}} \\ \hline A_{20} & A_{21} & A_{22} \end{array}\right)$$

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

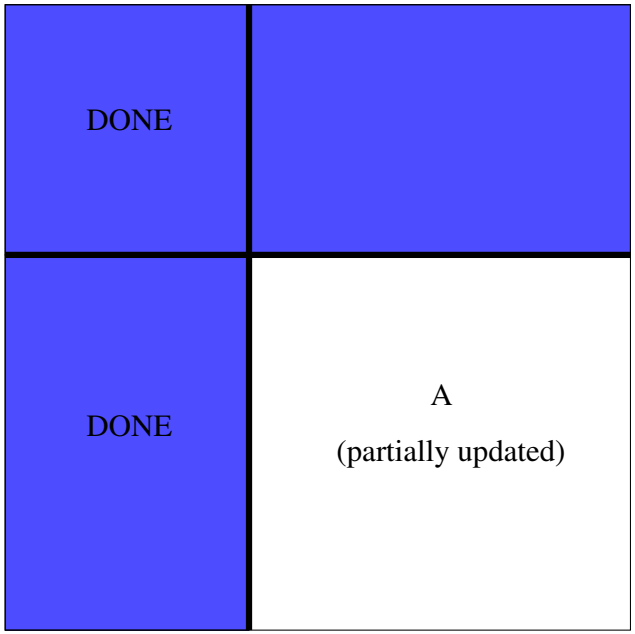
Factorización de Cholesky con FLAME



Operaciones por iteración:

```
%================%
A11 = chol( A11 );
A21 = A21 / tril( A11 )';
A22 = A22 - tril( A21 * A21' );
%================%
```

Factorización de Cholesky con FLAME

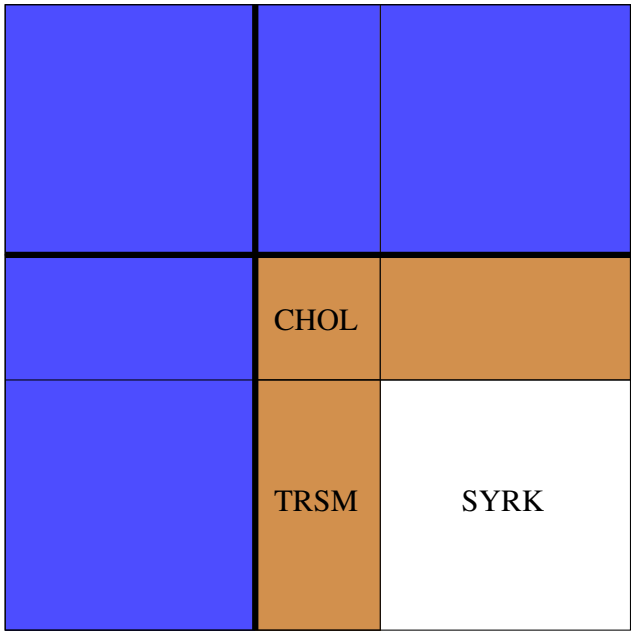


Continue with

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} A_{00} & A_{01} & A_{02} \\ \hline A_{10} & A_{11} & A_{12} \\ \hline A_{20} & A_{21} & A_{22} \end{array}\right)$$

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

Factorización de Cholesky con FLAME



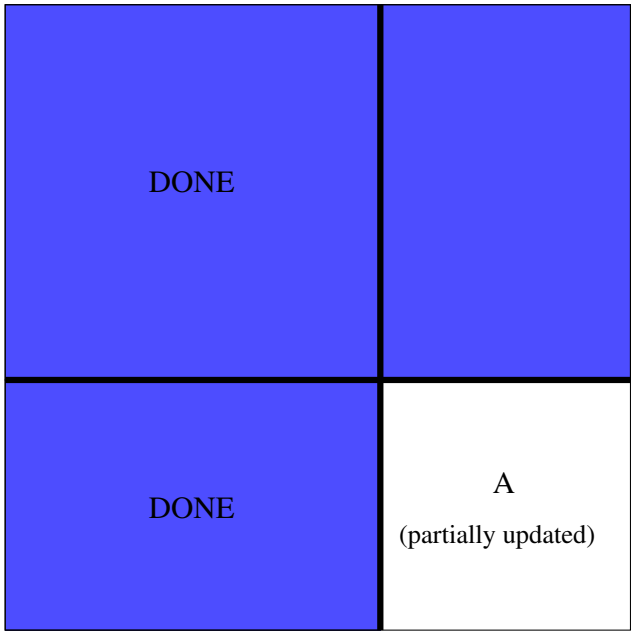
Repartition

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} A_{00} & A_{01} & A_{02} \\ \hline A_{10}^{T} & A_{11} & A_{12}^{T} \\ \hline A_{20} & A_{21} & A_{22} \end{array}\right)$$

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

```
FLA_Repart_2x2_to_3x3(
   ATL, ATR,        &A00, &A01, &A02,
                    &A10, &A11, &A12,
   ABL, ABR,        &A20, &A21, &A22,
                    b,   b, FLA_BR );
```

Factorización de Cholesky con FLAME



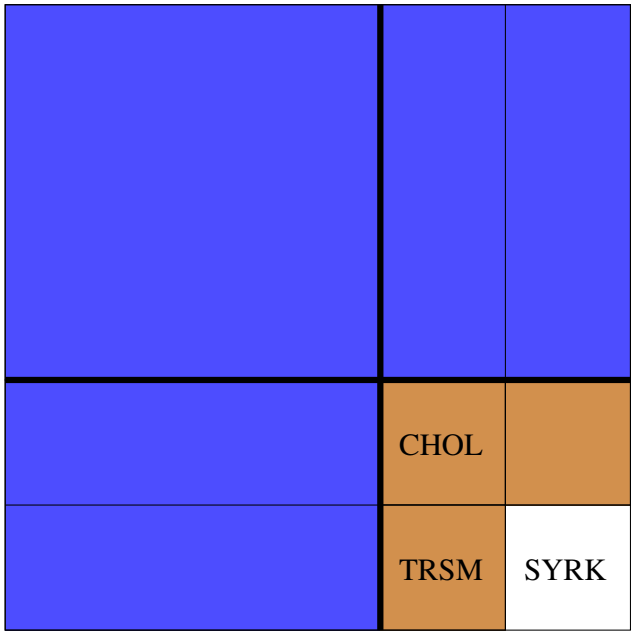
Continue with

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} A_{00} & A_{01} & A_{02} \\ \hline A_{10} & A_{11} & A_{12} \\ \hline A_{20} & A_{21} & A_{22} \end{array}\right)$$

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

```
FLA_Cont_with_3x3_to_2x2(
  &ATL, &ATR,       A00, A01, A02,
                     A10, A11, A12,
  &ABL, &ABR,       A20, A21, A22,
                     FLA_TL );
```

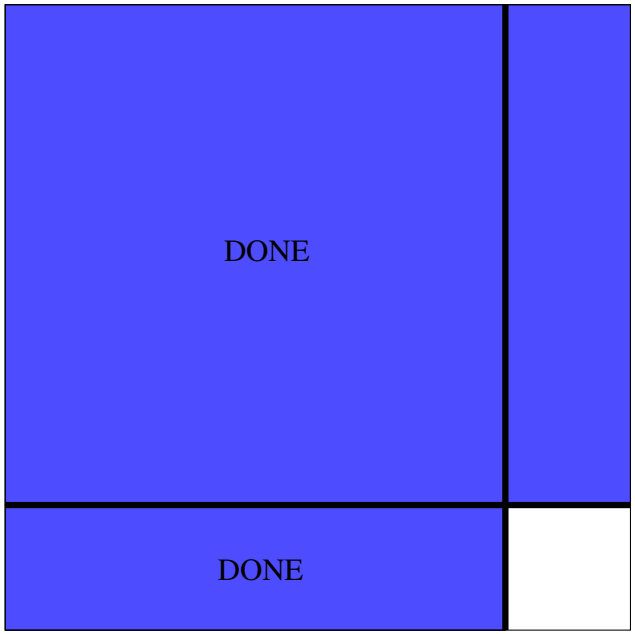
Factorización de Cholesky con FLAME



El algoritmo itera...

```
while ( size( ATL, 1 ) < size( A, 1 ) )
...
```

Factorización de Cholesky con FLAME



El algoritmo itera...

```
while ( size( ATL, 1 ) < size( A, 1 ) )
...
```

Notación FLAME y FLAME/C

Partition $A \rightarrow \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right)$

where A_{TL} is 0×0

while $m(A_{TL}) < m(A)$ **do**

Determine block size n_b

Repartition

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} A_{00} & A_{01} & A_{02} \\ \hline A_{10} & A_{11} & A_{12} \\ \hline A_{20} & A_{21} & A_{22} \end{array}\right)$$

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

$$A_{11} := \text{CHOL\_UNB}(A_{11})$$
$$A_{21} := A_{21}\,\text{TRIL}(A_{11})^{-\text{T}}$$
$$A_{22} := A_{22} - A_{21}A_{21}^{\text{T}}$$

Continue with

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} A_{00} & A_{01} & A_{02} \\ \hline A_{10} & A_{11} & A_{12} \\ \hline A_{20} & A_{21} & A_{22} \end{array}\right)$$

endwhile

```
int FLA_Chol_l_blk_var3_imp1( FLAG_Obj A, int nb_alg )
{
  FLA_Part_2x2( A,   &ATL, &ATR,
                     &ABL,  &ABR,
                     0, 0, FLAG_TL );

  while ( FLA_Obj_width( ATL ) < FLA_Obj_width( A ) ){
    b = min( min( FLA_Obj_length( ABR ),
                  FLA_Obj_width( ABR ) ), nb_alg);

    FLA_Repart_2x2_to_3x3(
        ATL, ATR,       &A00, &A01, &A02,
                        &A10, &A11, &A12,
        ABL, ABR,       &A20, &A21, &A22,
                        b,  b, FLA_BR );

    /* ********************************************* */
    FLA_Chol_unb( A11 );

    FLA_Trsm( FLA_RIGHT, FLA_LOWER_TRIANGULAR,
              FLA_TRANSPOSE, FLA_NONUNIT_DIAG,
              FLA_ONE, A11, A21 );

    FLA_Syrk( FLA_LOWER_TRIANGULAR, FLA_NO_TRANSPOSE,
              FLA_MINUS_ONE, A21, FLA_ONE, A22 );
    /* ********************************************* */

    FLA_Cont_with_3x3_to_2x2(
        &ATL, &ATR,     A00, A01, A02,
                        A10, A11, A12,
        &ABL, &ABR,     A20, A21, A22,
                        FLA_TL );
  }
}
```

Comparación FLAME/C - FLAG/C

```
int FLAG_Chol_l_blk_var3_imp1( FLAG_Obj A, int nb_alg )
{
  FLAG_Part_2x2( A,   &ATL, &ATR,
                      &ABL,  &ABR,
                      0, 0, FLAG_TL );

  while ( FLAG_Obj_width( ATL ) < FLAG_Obj_width( A ) ){
    b = min( min( FLAG_Obj_length( ABR ),
                  FLAG_Obj_width( ABR ) ), nb_alg);

    FLAG_Repart_2x2_to_3x3(
        ATL, ATR,       &A00, &A01, &A02,
                        &A10, &A11, &A12,
        ABL, ABR,       &A20, &A21, &A22,
                        b,  b, FLAG_BR );

    /* ********************************************* */
    FLAG_Chol_unb( A11 );

    FLAG_Trsm( FLAG_RIGHT, FLAG_LOWER_TRIANGULAR,
               FLAG_TRANSPOSE, FLAG_NONUNIT_DIAG,
               FLA_ONE, A11, A21 );

    FLAG_Syrk( FLAG_LOWER_TRIANGULAR, FLAG_NO_TRANSPOSE,
               FLA_MINUS_ONE, A21, FLA_ONE, A22 );
    /* ********************************************* */

    FLAG_Cont_with_3x3_to_2x2(
        &ATL, &ATR,     A00, A01, A02,
                        A10, A11, A12,
        &ABL, &ABR,     A20, A21, A22,
                        FLA_TL );
  }
}
```

Principales diferencias:

1. Rutinas BLAS proporcionadas por FLAG/C
2. Las matrices deben haber sido transferidas previamente a la GPU

Evaluación de FLAG/C

Partition $A \rightarrow \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right)$
where A_{TL} is 0×0

while $m(A_{TL}) < m(A)$ **do**

Repartition

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array}\right)$$

where α_{11} is 1×1

Variant 1:
$a_{10}^T := a_{10}^T \text{TRIL}(A_{00})^{-T}$
$\alpha_{11} := \alpha_{11} - a_{10}^T a_{10}$
$\alpha_{11} := \sqrt{\alpha_{11}}$

Variant 2:
$\alpha_{11} := \alpha_{11} - a_{10}^T a_{10}$
$\alpha_{11} := \sqrt{\alpha_{11}}$
$a_{21} := a_{21} - A_{20} a_{10}$
$a_{21} := a_{21} / \alpha_{11}$

Variant 3:
$\alpha_{11} := \sqrt{\alpha_{11}}$
$a_{21} := a_{21} / \alpha_{11}$
$A_{22} := A_{22} - a_{21} a_{21}^T$

Continue with

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array}\right)$$

endwhile

Partition $A \rightarrow \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right)$
where A_{TL} is 0×0

while $m(A_{TL}) < m(A)$ **do**

Determine block size n_b
Repartition

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} A_{00} & A_{01} & A_{02} \\ \hline A_{10} & A_{11} & A_{12} \\ \hline A_{20} & A_{21} & A_{22} \end{array}\right)$$

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

Variant 1:
$A_{10} := A_{10} \text{TRIL}(A_{00})^{-T}$
$A_{11} := A_{11} - A_{10} A_{10}^T$
$A_{11} := \text{CHOL\_UNB}(A_{11})$

Variant 2:
$A_{11} := A_{11} - A_{10} A_{10}^T$
$A_{11} := \text{CHOL\_UNB}(A_{11})$
$A_{21} := A_{21} - A_{20} A_{10}^T$
$A_{21} := A_{21} \text{TRIL}(A_{11})^{-T}$

Variant 3:
$A_{11} := \text{CHOL\_UNB}(A_{11})$
$A_{21} := A_{21} \text{TRIL}(A_{11})^{-T}$
$A_{22} := A_{22} - A_{21} A_{21}^T$

Continue with

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} A_{00} & A_{01} & A_{02} \\ \hline A_{10} & A_{11} & A_{12} \\ \hline A_{20} & A_{21} & A_{22} \end{array}\right)$$

endwhile

- Cálculo de raíces cuadradas → CPU

Evaluación de FLAG/C

Familia de implementaciones Imp1

- Transferencia de elementos diagonales
  - Cálculos
    - si $n_b \ll n \rightarrow$ multitud de llamadas a rutinas de nivel 1 y 2 de CUBLAS con poca carga computacional
    - si n_b es grande se reduce el núm. de operaciones expresadas en términos de llamadas al nivel 3 de CUBLAS
  - Comunicaciones
    - $T_1(n) = n(2\alpha + 8\beta)$ donde $\alpha \gg \beta$
    - $2n$ transferencias de 4 bytes

Familia de implementaciones Imp2 (Generalización de Imp1)

- Transferencia de bloques diagonales
- Cálculos: llamadas a rutinas de nivel 3 de CUBLAS
- Comunicaciones
  - $T_2(n, n_b) = \frac{n}{n_b}(2\alpha + 8\beta n_b^2)$
  - $2\frac{n}{n_b}$ transferencias de $4n_b^2$ bytes

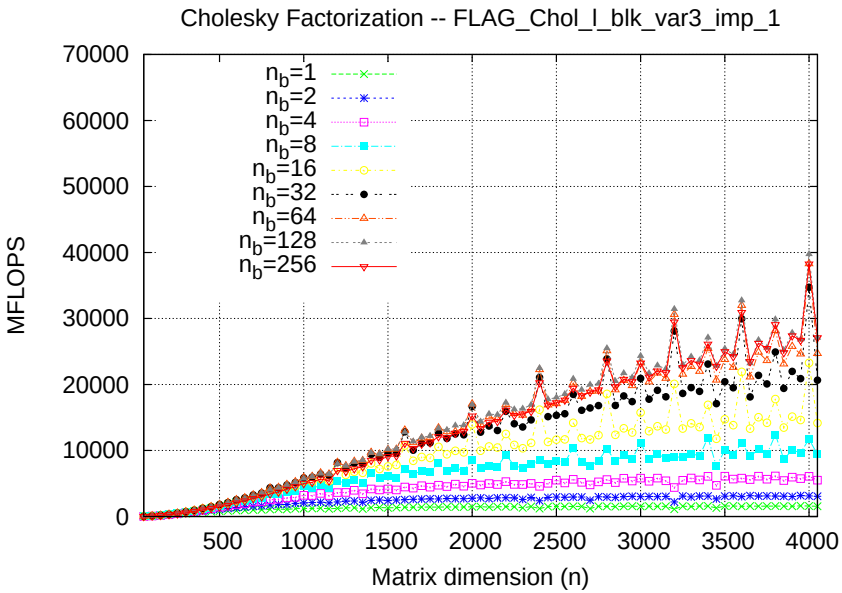
Evaluación de FLAG/C

- Métrica empleada: $\text{MFLOPs} = \frac{n^3}{3 \times 10^6 \times T_{ejec}}$
- T_{ejec} es el tiempo empleado por el procesador para calcular la factorización. Incluye:
  - Tiempo de transmisión de la matriz (CPU $\rightarrow$ GPU)
  - Tiempo de transmisión del factor de Cholesky (GPU $\rightarrow$ CPU)

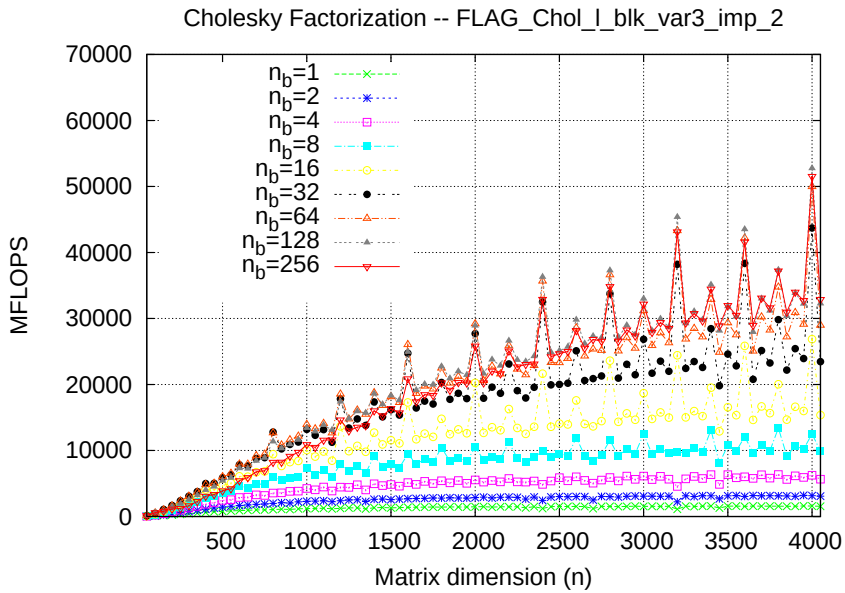
	CPU	GPU
Procesador	Intel Core 2 Duo	NVIDIA 8800 Ultra
Modelo	Crusoe E6320	G80
Frecuencia de reloj	1.86 GHz	575 MHz
Frecuencia memoria	2×333 MHz	2×900 MHz
Ancho banda bus	64 bits	384 bits
Máx. ancho de banda	5.3 GB/s	86.4 GB/s
Memoria	1024 MB DDR2	768 MB GDDR3
Tipo de bus	PCI Express x16 (4 GB/s)	
Versión BLAS	Intel MKL 10	CUBLAS 2.0

Evaluación de FLAG/C: Imp1 vs. Imp2

Imp1 Var3 CHOL_BLK



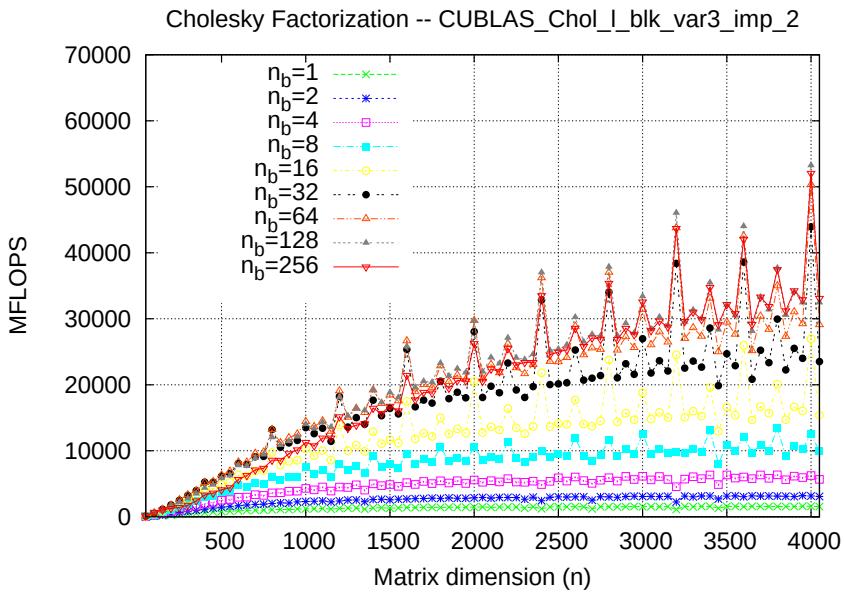
Imp2 Var3 CHOL_BLK



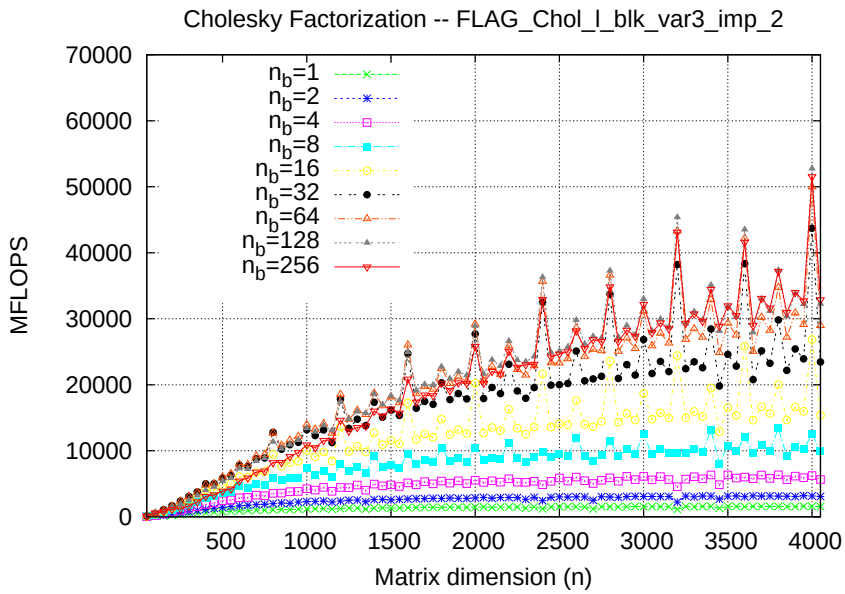
- Mejora significativa cuando $n \in [0, 2000]$
- Conclusiones:
  - Versatilidad para obtener implementaciones de altas prestaciones

Evaluación de FLAG/C: sobrecoste

CUBLAS
Imp2 Var3 CHOL_BLK



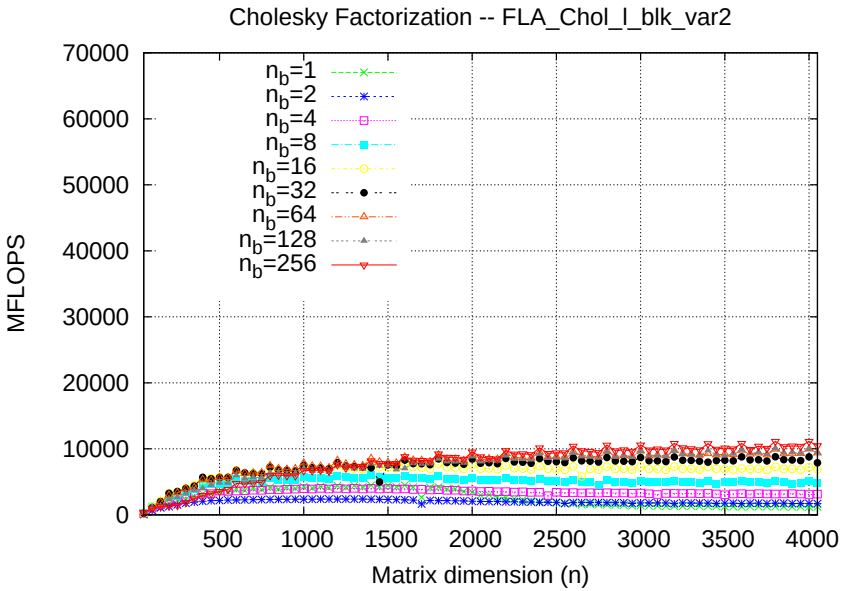
FLAG/C
Imp2 Var3 CHOL_BLK



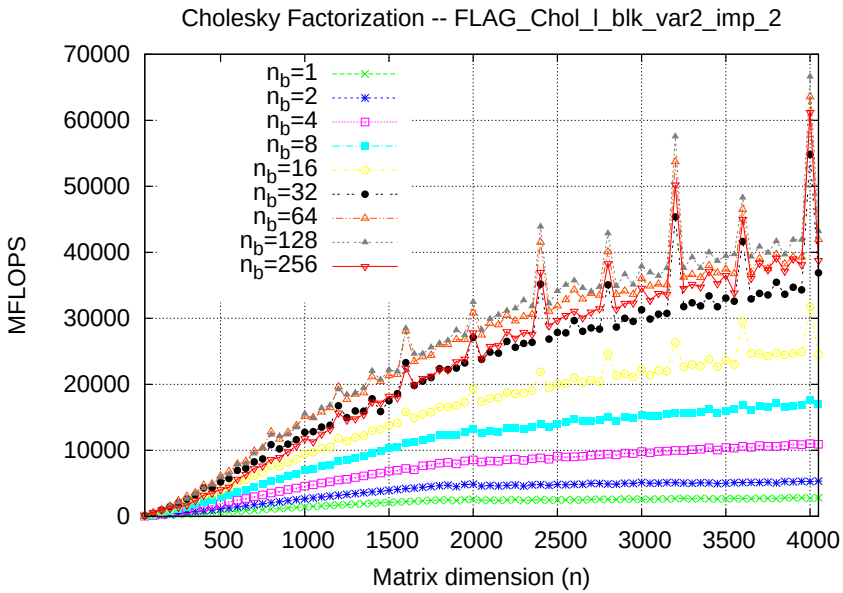
- El sobrecoste de FLAG/C es despreciable

Evaluación de FLAG/C: FLAME/C vs. FLAG/C

FLAME/C
Var2 CHOL_BLK



FLAG/C
Imp2 Var2 CHOL_BLK



- La GPU acelera significativamente el cálculo

Conclusiones

- FLAG/C es fácil de usar:
  - La curva de aprendizaje de FLAG/C es mínima
  - Alto nivel de abstracción para el desarrollo de códigos
- FLAG/C es eficiente:
  - Permite explotar eficientemente el paralelismo multihebra masivo disponible
  - Se puede acelerar hasta 7 veces el rendimiento obtenido por la CPU (factorización de Cholesky)
- FLAG/C (FLAME) es flexible:
  - Permite adaptar la interfaz a cualquier tipo de acelerador *hardware*

Conclusiones

- FLAG/C es fácil de usar:
  - La curva de aprendizaje de FLAG/C es mínima
  - Alto nivel de abstracción para el desarrollo de códigos
- FLAG/C es eficiente:
  - Permite explotar eficientemente el paralelismo multihebra masivo disponible
  - Se puede acelerar hasta 7 veces el rendimiento obtenido por la CPU (factorización de Cholesky)
- FLAG/C (FLAME) es flexible:
  - Permite adaptar la interfaz a cualquier tipo de acelerador *hardware*

Conclusiones

- FLAG/C es fácil de usar:
  - La curva de aprendizaje de FLAG/C es mínima
  - Alto nivel de abstracción para el desarrollo de códigos
- FLAG/C es eficiente:
  - Permite explotar eficientemente el paralelismo multihebra masivo disponible
  - Se puede acelerar hasta 7 veces el rendimiento obtenido por la CPU (factorización de Cholesky)
- FLAG/C (FLAME) es flexible:
  - Permite adaptar la interfaz a cualquier tipo de acelerador *hardware*

Gracias...

Más información...

http://www3.uji.es/~figual
http://www.hpca.uji.es