

Level-3 BLAS on a GPU

Picking the Low Hanging Fruit

Francisco Igual¹ Gregorio Quintana-Ortí¹
Robert A. van de Geijn²

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

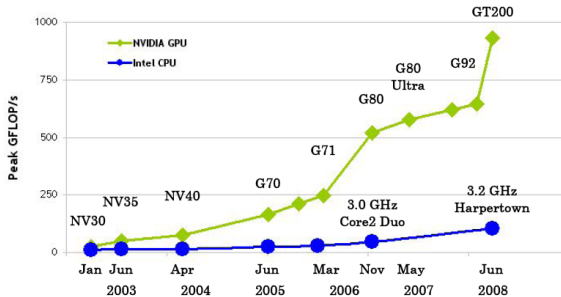
²Department of Computer Sciences.
The University of Texas at Austin





Motivation (I)

- GPU vendors promise spectacular **peak performances**

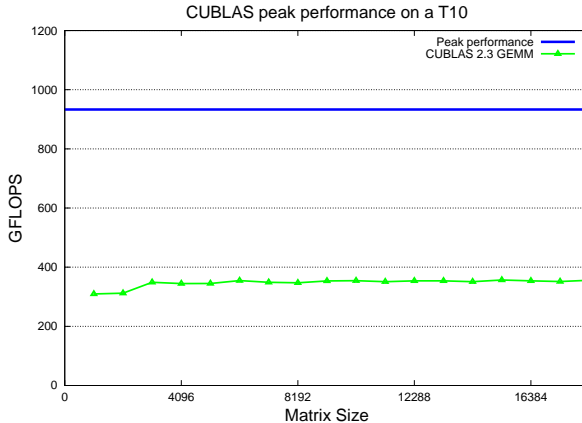


GT200 = GeForce GTX 280	G71 = GeForce 7900 GTX	NV35 = GeForce FX 5950 Ultra
G92 = GeForce 9800 GTX	G70 = GeForce 7800 GTX	NV30 = GeForce FX 5800
G80 = GeForce 8800 GTX	NV40 = GeForce 6800 Ultra	



Motivation (II)

- But **real performances** are not so optimistic. . .

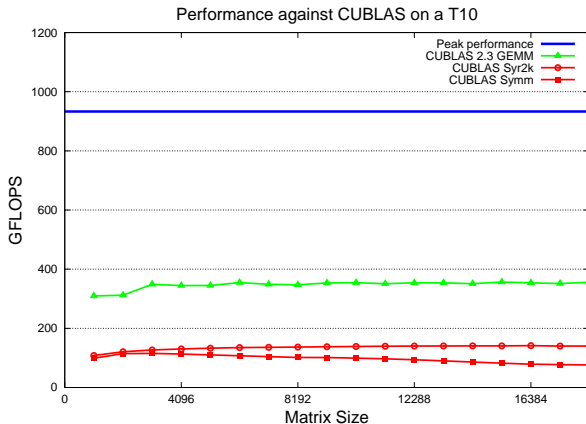


Limitations of the graphics-oriented architecture



Motivation (III)

- And **current implementations** can be quite poor...



Hard to **efficiently** program the GPU, even using CUDA



Outline

- 1 Introduction
- 2 Development of algorithms by blocks. The matrix-matrix product
- 3 Accelerating the Level-3 CUBLAS
- 4 Experimental results
- 5 Conclusions and future work



Contents

- 1 Introduction
- 2 Development of algorithms by blocks. The matrix-matrix product
- 3 Accelerating the Level-3 CUBLAS
- 4 Experimental results
- 5 Conclusions and future work



Introduction

BLAS

- BLAS: Basic Linear Algebra Subprograms
- Lie in the heart of complex dense linear algebra algorithms
- Key in their final performance
- Tuned implementations for many architectures
 - GotoBLAS, MKL, CUBLAS

Goals

- Tune the performance of the latest implementation of CUBLAS
- Without low-level programming (CUDA)
- Improving programmability: FLAME methodology



Introduction

BLAS

- BLAS: Basic Linear Algebra Subprograms
- Lie in the heart of complex dense linear algebra algorithms
- Key in their final performance
- Tuned implementations for many architectures
 - GotoBLAS, MKL, CUBLAS

Goals

- Tune the performance of the latest implementation of CUBLAS
- Without low-level programming (CUDA)
- Improving programmability: FLAME methodology



Contents

- 1 Introduction
- 2 Development of algorithms by blocks. The matrix-matrix product
- 3 Accelerating the Level-3 CUBLAS
- 4 Experimental results
- 5 Conclusions and future work



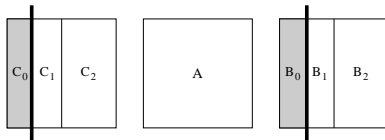
The FLAME methodology

- FLAME: high level abstraction and notation for dense linear algebra algorithms
- Not only a library:
 - Notation for expressing algorithms
 - Methodology for systematic derivation of algorithms
 - Application Program Interfaces (APIs) for representing the algorithms in code
 - Tools and more
- Example: Matrix-matrix multiplication

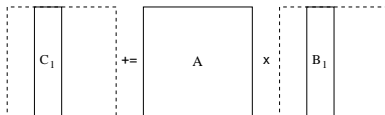


The matrix-matrix multiplication

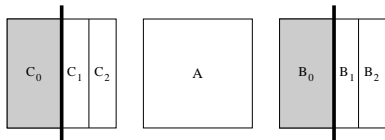
a) Partitioning before iteration



b) Computation in iteration



c) Advancement of partitioning for next iteration



The matrix-matrix multiplication. FLAME algorithm



Algorithm: GEMM_MP(A, B, C)

Partition $B \rightarrow \left(\begin{array}{c|c} B_L & B_R \end{array} \right), C \rightarrow \left(\begin{array}{c|c} C_L & C_R \end{array} \right)$

where B_L has 0 columns, C_L has 0
columns

while $n(B_L) < n(B)$ **do**

Determine block size b

Repartition

$\left(\begin{array}{c|c} B_L & B_R \end{array} \right) \rightarrow \left(\begin{array}{c|c|c} B_0 & B_1 & B_2 \end{array} \right),$
 $\left(\begin{array}{c|c} C_L & C_R \end{array} \right) \rightarrow \left(\begin{array}{c|c|c} C_0 & C_1 & C_2 \end{array} \right)$

where B_1 has b columns, C_1 has b
columns

$$C_1 := C_1 + AB_1$$

Continue with

$\left(\begin{array}{c|c} B_L & B_R \end{array} \right) \leftarrow \left(\begin{array}{c|c|c} B_0 & B_1 & B_2 \end{array} \right),$
 $\left(\begin{array}{c|c} C_L & C_R \end{array} \right) \leftarrow \left(\begin{array}{c|c|c} C_0 & C_1 & C_2 \end{array} \right)$

endwhile



The matrix-matrix multiplication. FLAME code

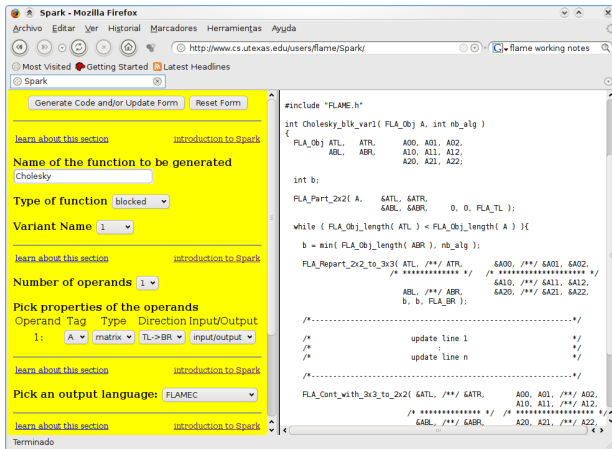
```

1  FLA_Obj  BL, BR,      B1, B2, B3;
2  FLA_Obj  CL, CR,      C1, C2, C3;
3
4  FLA_Part_1x2( B,
5                &BL, &BR,              0, FLA_LEFT );
6
7  FLA_Part_1x2( C,
8                &CL, &CR,              0, FLA_LEFT );
9
10 while ( FLA_Obj_width( BL ) < FLA_Obj_width( B ) ){
11
12     b = min( FLA_Obj_width( BR ), nb_alg );
13
14     FLA_Repart_1x2_to_1x3( BL, /* */ BR,
15                            &B0, /* */ &B1, &B2,
16                            b, FLA_RIGHT );
17
18     FLA_Repart_1x2_to_1x3( CL, /* */ CR,
19                            &C0, /* */ &C1, &C2,
20                            b, FLA_RIGHT );
21
22     /*-----*/
23     FLA_Gemm( FLA_NO_TRANSPOSE, FLA_TRANSPOSE, FLA_MINUS_ONE, A, B1, FLA_ONE, C1 );
24     /*-----*/
25
26     FLA_Cont_with_1x3_to_1x2( &BL, /* */ &BR,
27                               B0, B1, /* */ B2,
28                               FLA_LEFT );
29
30     FLA_Cont_with_1x3_to_1x2( &CL, /* */ &CR,
31                               C0, C1, /* */ C2,
32                               FLA_LEFT );
33 }

```

The matrix-matrix multiplication. Spark

SPARK: AUTOMATIC GENERATION OF CODE SKELETONS



Spark - Mozilla Firefox

Archivo Editar Ver Historial Marcadores Herramientas Ayuda

http://www.cs.utexas.edu/users/flame/Spark/

Most Visited Getting Started Latest Headlines

Spark

Generate Code and/or Update Form Reset Form

[learn about this section](#) [introduction to Spark](#)

Name of the function to be generated

Cholesky

Type of function blocked

Variant Name 1

[learn about this section](#) [introduction to Spark](#)

Number of operands 1

Pick properties of the operands

Operand	Tag	Type	Direction	Input/Output
1:	A	matrix	TL->BR	input/output

[learn about this section](#) [introduction to Spark](#)

Pick an output language: FLAMEC

[learn about this section](#) [introduction to Spark](#)

Terminado

```
#include "FLAME.h"

int Cholesky_blk_var1( FLA_Obj A, int nb_alg )
{
    FLA_Obj ATL,   ATR,   A00, A01, A02,
             ABL,   ABR,   A10, A11, A12,
             A20, A21, A22;

    int b;

    FLA_Part_2x2( A,      &ATL, &ATR,
                  &ABL, &ABR,   0, 0, FLA_TL );

    while ( FLA_Obj_length( ATL ) < FLA_Obj_length( A ) ){

        b = min( FLA_Obj_length( 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 );

        /*-----*/
        /*          update line 1          */
        /*          :                       */
        /*          update line n          */
        /*-----*/

        FLA_Cont_with_3x3_to_2x2( &ATL, /**/ &ATR,   A00, A01, /**/ A02,
                                  /* ***** */ /* ***** */
                                  &ABL, /**/ &ABR,   A20, A21, /**/ A22,
                                  <

```



Contents

- 1 Introduction
- 2 Development of algorithms by blocks. The matrix-matrix product
- 3 Accelerating the Level-3 CUBLAS**
- 4 Experimental results
- 5 Conclusions and future work



Accelerated Level-3 BLAS routines

SYMM $C := AB + C$, A symmetric and only the lower triangular part of this matrix is stored.

SYRK $C := C - AA^T$, C symmetric and only the lower triangular part of this matrix is stored and computed.

SYR2K $C := C - AB^T - BA^T$, C symmetric and only the upper triangular part of this matrix is stored and computed.

TRMM $C := AB + C$, where A upper triangular.

TRSM $XA^T = B$, A lower triangular and B is overwritten with the solution X .



Accelerating the CUBLAS

Three main ideas

- 1 GEMM-based implementations
- 2 Multiple algorithmic variants
- 3 Multiple block sizes



GEMM-based SYRK

Algorithm: SYRK_GEMM(C, A)

Partition $C \rightarrow \left(\begin{array}{c|c} C_{TL} & C_{TR} \\ \hline C_{BL} & C_{BR} \end{array} \right), A \rightarrow \left(\begin{array}{c} A_T \\ \hline A_B \end{array} \right)$

where C_{TL} is 0×0 , A_T has 0 rows

while $m(C_{TL}) < m(C)$ **do**

Determine block size b

Repartition

$\left(\begin{array}{c|c} C_{TL} & C_{TR} \\ \hline C_{BL} & C_{BR} \end{array} \right) \rightarrow \left(\begin{array}{c|c|c} C_{00} & C_{01} & C_{02} \\ \hline C_{10} & C_{11} & C_{12} \\ \hline C_{20} & C_{21} & C_{22} \end{array} \right), \left(\begin{array}{c} A_T \\ \hline A_B \end{array} \right) \rightarrow \left(\begin{array}{c} A_0 \\ \hline A_1 \\ \hline A_2 \end{array} \right)$

where C_{11} is $b \times b$, A_1 has b rows

$C_{11} := C_{11} - A_1 A_1^T$ (SYRK)

$C_{21} := C_{21} - A_2 A_1^T$ (GEMM)

Continue with

$\left(\begin{array}{c|c} C_{TL} & C_{TR} \\ \hline C_{BL} & C_{BR} \end{array} \right) \leftarrow \left(\begin{array}{c|c|c} C_{00} & C_{01} & C_{02} \\ \hline C_{10} & C_{11} & C_{12} \\ \hline C_{20} & C_{21} & C_{22} \end{array} \right), \left(\begin{array}{c} A_T \\ \hline A_B \end{array} \right) \leftarrow \left(\begin{array}{c} A_0 \\ \hline A_1 \\ \hline A_2 \end{array} \right)$

endwhile



Multiple algorithmic variants

Algorithm: GEMM_MP(A, B, C)

Partition $B \rightarrow \left(\begin{array}{c|c} B_L & B_R \end{array} \right), C \rightarrow \left(\begin{array}{c|c} C_L & C_R \end{array} \right)$

where B_L has 0 columns, C_L has 0 columns

while $n(B_L) < n(B)$ **do**

Determine block size b

Repartition

$\left(\begin{array}{c|c} B_L & B_R \end{array} \right) \rightarrow \left(\begin{array}{c|c|c} B_0 & B_1 & B_2 \end{array} \right),$
 $\left(\begin{array}{c|c} C_L & C_R \end{array} \right) \rightarrow \left(\begin{array}{c|c|c} C_0 & C_1 & C_2 \end{array} \right)$

where B_1 has b columns, C_1 has b columns

$C_1 := C_1 + AB_1$

Continue with

$\left(\begin{array}{c|c} B_L & B_R \end{array} \right) \leftarrow \left(\begin{array}{c|c|c} B_0 & B_1 & B_2 \end{array} \right),$
 $\left(\begin{array}{c|c} C_L & C_R \end{array} \right) \leftarrow \left(\begin{array}{c|c|c} C_0 & C_1 & C_2 \end{array} \right)$

endwhile



Multiple algorithmic variants

Algorithm: GEMM_PM(A, B, C)

Partition $A \rightarrow \left(\frac{A_T}{A_B} \right), C \rightarrow \left(\frac{C_T}{C_B} \right)$

where A_T has 0 rows, C_T has 0 rows

while $m(A_T) < m(A)$ **do**

Determine block size b

Repartition

$$\left(\frac{A_T}{A_B} \right) \rightarrow \left(\frac{A_0}{A_1}{A_2} \right), \left(\frac{C_T}{C_B} \right) \rightarrow \left(\frac{C_0}{C_1}{C_2} \right)$$

where A_1 has b rows, C_1 has b rows

$$C_1 := C_1 + A_1 B$$

Continue with

$$\left(\frac{A_T}{A_B} \right) \leftarrow \left(\frac{A_0}{A_1}{A_2} \right), \left(\frac{C_T}{C_B} \right) \leftarrow \left(\frac{C_0}{C_1}{C_2} \right)$$

endwhile



Multiple algorithmic variants

Algorithm: GEMM_PP(A, B, C)

Partition $A \rightarrow (A_L \mid A_R), B \rightarrow \left(\frac{B_T}{B_B}\right)$

where A_L has 0 columns, B_T has 0

while $n(A_L) < n(A)$ **do**

Determine block size b

Repartition

$(A_L \mid A_R) \rightarrow (A_0 \mid A_1 \mid A_2), \left(\frac{B_T}{B_B}\right) \rightarrow \left(\frac{B_0}{B_1} \mid \frac{B_0}{B_2}\right)$

where A_1 has b columns, B_1 has b rows

$C := C + A_1 B_1$

Continue with

$(A_L \mid A_R) \leftarrow (A_0 \mid A_1 \mid A_2), \left(\frac{B_T}{B_B}\right) \leftarrow \left(\frac{B_0}{B_1} \mid \frac{B_0}{B_2}\right)$

endwhile



Varying the block size

Algorithm: GEMM_PP(A, B, C)

Partition $A \rightarrow (A_L \mid A_R), B \rightarrow \left(\frac{B_T}{B_B}\right)$

where A_L has 0 columns, B_T has 0

while $n(A_L) < n(A)$ **do**

Determine block size b

Repartition

$(A_L \mid A_R) \rightarrow (A_0 \mid A_1 \mid A_2), \left(\frac{B_T}{B_B}\right) \rightarrow \left(\frac{B_0}{B_1} \mid \frac{B_0}{B_2}\right)$

where A_1 has b columns, B_1 has b rows

$C := C + A_1 B_1$

Continue with

$(A_L \mid A_R) \leftarrow (A_0 \mid A_1 \mid A_2), \left(\frac{B_T}{B_B}\right) \leftarrow \left(\frac{B_0}{B_1} \mid \frac{B_0}{B_2}\right)$

endwhile



Contents

- 1 Introduction
- 2 Development of algorithms by blocks. The matrix-matrix product
- 3 Accelerating the Level-3 CUBLAS
- 4 Experimental results**
- 5 Conclusions and future work



Experimental setup

Experimental setup

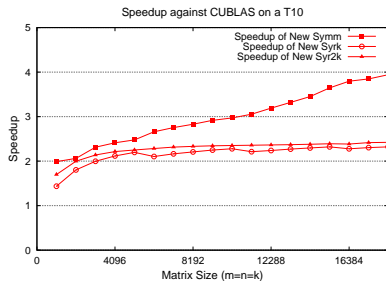
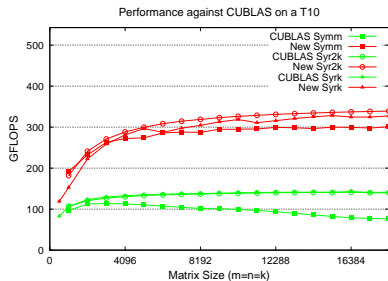
CPU	Dual Xeon QuadCore E5410
CPU frequency	2.33 Ghz
RAM memory	8 Gbytes
GPU	Tesla C1060
Processor	Nvidia GT200
GPU frequency	1.3 Ghz
Video memory	4 Gbytes DDR3
Interconnection	PCIExpress Gen2
CUDA (CUBLAS) version	2.3 (July 2009)
Driver version	185.18

- Results in terms of GFLOPS (single precision)
- Transfer times not included in results



Experimental results

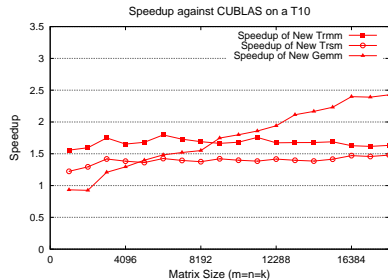
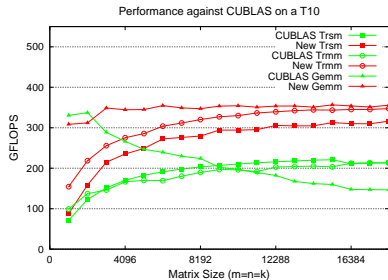
Results for square matrices: SYRK, SYMM and SYR2K.





Experimental results

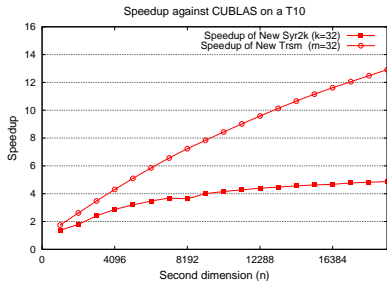
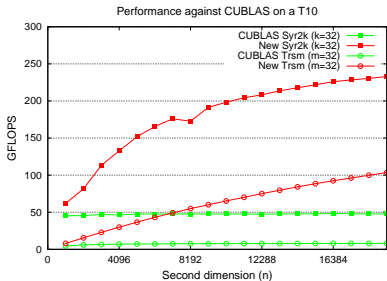
Results for square matrices: TRSM and TRMM.



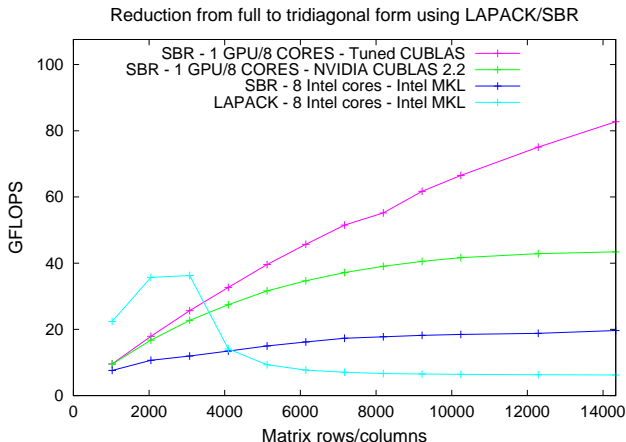


Experimental results

Results for rectangular matrices: TRSM and SYR2K.



Application: Symmetric Eigenvalue Problem (PPAM09)



- Using our SYMM and SYR2K implementations
- Speedup 2.2x when using GPU acceleration for SBR (19.2 vs 42 GFLOPS)
- Speedup 4.3x when using GPU acceleration and tuned BLAS (19.2 vs 82 GFLOPS)



Contents

- 1 Introduction
- 2 Development of algorithms by blocks. The matrix-matrix product
- 3 Accelerating the Level-3 CUBLAS
- 4 Experimental results
- 5 Conclusions and future work**



Conclusions and future work

- Performance boost with little effort for BLAS routines
- Attain considerable speedups compared with tuned CUBLAS
- Without writing one line of CUDA code
- Methodology applicable to other linear algebra routines



Conclusions and future work

Thank you!

More information. . .



[Level-3 BLAS on a GPU: Picking the Low Hanging Fruit]

FLAME Working Note #37

May, 2009

- HPCA Group at University Jaume I (<http://www.hpca.uji.es>)

figual@icc.uji.es