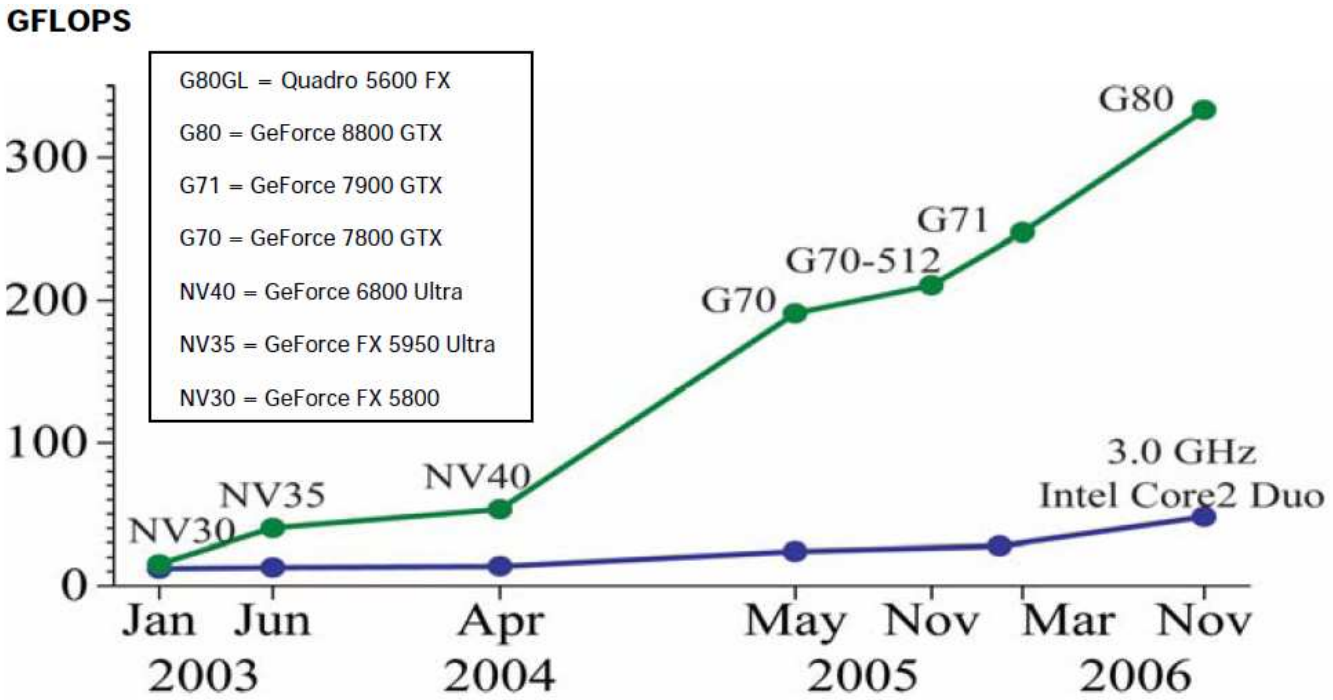
Evaluation and Tuning of Level 3 CUBLAS for Graphics Processors

Sergio Barrachina Maribel Castillo Francisco Igual
Rafael Mayo Enrique S. Quintana-Ortí

Universitat Jaume I
Spain

9th IEEE International Workshop on
Parallel and Distributed Scientific and Engineering Computing
Miami, April 2008

Motivation



Current graphics processors (GPUs) are gaining wide appeal as HPC accelerators

Motivation (II)

- GPUs are being transformed into general-purpose devices of wide appeal (GPGPU):
  1. High performance
  2. Low cost
  3. Programmability

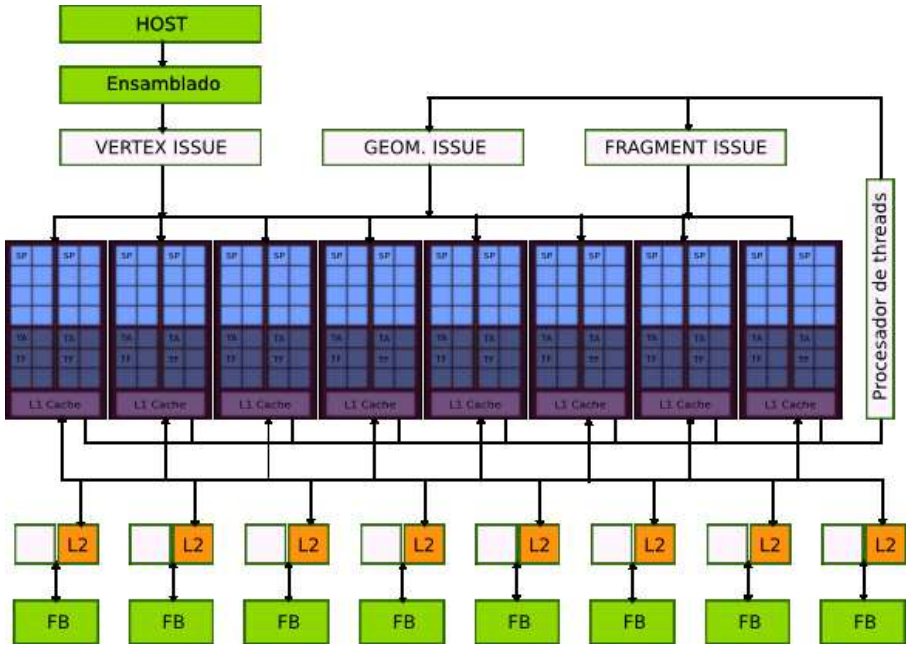
$\longrightarrow$

- Applied in many problems: physical simulations, real-time image processing, linear algebra, signal processing,...

Outline

CUDA Hardware

- A CUDA-enabled device is seen as a coprocessor to the CPU, capable of executing a very high number of threads in parallel
- Example: nVIDIA G80 as a set of SIMD Multiprocessors with On-Chip Shared Memory



- Up to 128 *Streaming Processors* (SP), grouped in clusters
- SP are SIMD processors
- Small and fast Shared Memory shared per SP cluster
- Local 32-bit registers per processor

CUDA Software

- The CUDA API provides a simple framework for writing C programs for execution on the GPU
- Consists of:
  - A minimal set of extensions to the C language
  - A runtime library of routines for controlling the transfers between video and main memory, run-time configuration, execution of device-specific functions, handling multiple GPUs,...

CUDA libraries

On top of CUDA, nVIDIA provides two optimized libraries: CUFFT and CUBLAS

CUBLAS Example

```
int main( void ){
  ...
  float* h_vector, * d_vector;

  h_vector = (float*)malloc(M*sizeof(float));

  ...
  cublasAlloc(M, sizeof(float),
              (void**) &d_vector);

  cublasSetVector(M, sizeof(float), h_vector,
                  d_vector, 1);
  cublasSscal(M, ALPHA, d_vector, 1);
  cublasGetVector(M, sizeof(float), d_vector,
                  h_vector, 1);

  cublasFree(d_vector);
  ...
}
```

A typical CUDA (and CUBLAS) program has 3 phases:

1. Allocation and transfer of data to GPU
2. Execution of the BLAS kernel
3. Transfer of results back to main memory

Level 3 CUBLAS Evaluation

- BLAS are the building blocks for more complex linear algebra operations (e.g., solution of dense linear systems)
- There exist optimized versions of the BLAS tuned for most current processors: GotoBLAS, Intel MKL, AMD ACML,...

Level 3 CUBLAS Evaluation

- BLAS are the building blocks for more complex linear algebra operations (e.g., solution of dense linear systems)
- There exist optimized versions of the BLAS tuned for most current processors: GotoBLAS, Intel MKL, AMD ACML,...

Goals

- Evaluate the result of the Level 3 BLAS in CUBLAS
- Tune the implementations in CUBLAS to attain higher performance

Experimental Setup

Intel Core 2 Duo with a nVIDIA 8800 Ultra board:

	CPU	GPU
Processor	Intel Core 2 Duo	NVIDIA 8800 Ultra
Codename	Crusoe E6320	G80
Clock frequency	1.86 GHz	575 MHz
Memory speed	2×333 MHz	2×900 MHz
Bus width	64 bits	384 bits
Max. bandwidth	5.3 GB/s	86.4 GB/s
Memory	1024 MB DDR2	768 MB GDDR3
Bus	PCI Express x16 (4 GB/s)	

CUDA and CUBLAS 1.0 and single precision arithmetic used in the experiments

Evaluated BLAS routines

GEMM

$$C := \beta \cdot C + \alpha \cdot op(A) \cdot op(B)$$

SYRK

$$C := \beta \cdot C + \alpha \cdot A \cdot A^T \quad \text{or}$$
$$C := \beta \cdot C + \alpha \cdot A^T \cdot A$$

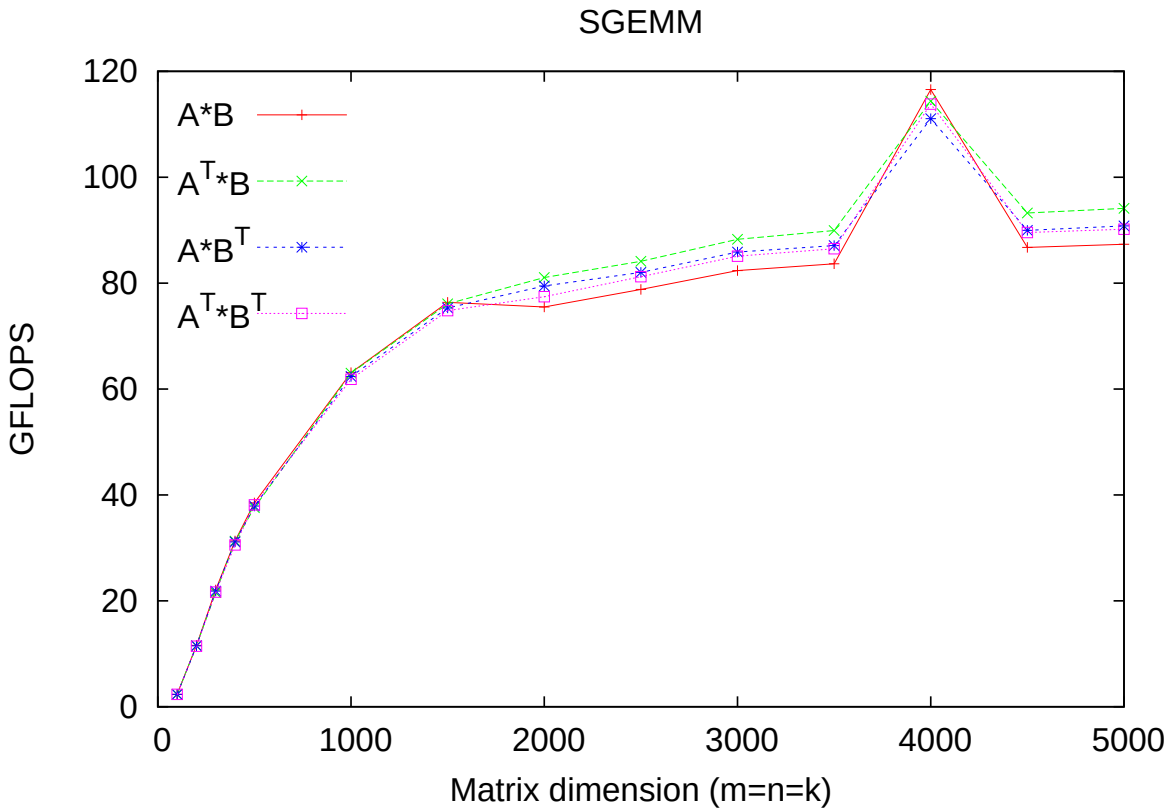
TRSM

$$op(A) \cdot X = \alpha \cdot B \quad \text{or}$$
$$X \cdot op(A) = \alpha \cdot B$$

where $op(X) = X$ or X^T

SYMM and TRMM similar

GEMM Evaluation



GEMM Evaluation

Main remarks

- Peak performance is $\sim$116 Gflops for GEMM

GEMM Evaluation

Main remarks

- Peak performance is ∼116 Gflops for GEMM
- Performance attained for large matrices is much better than that of small/medium problems

$$\Downarrow$$

Stream-oriented architecture

GEMM Evaluation

Main remarks

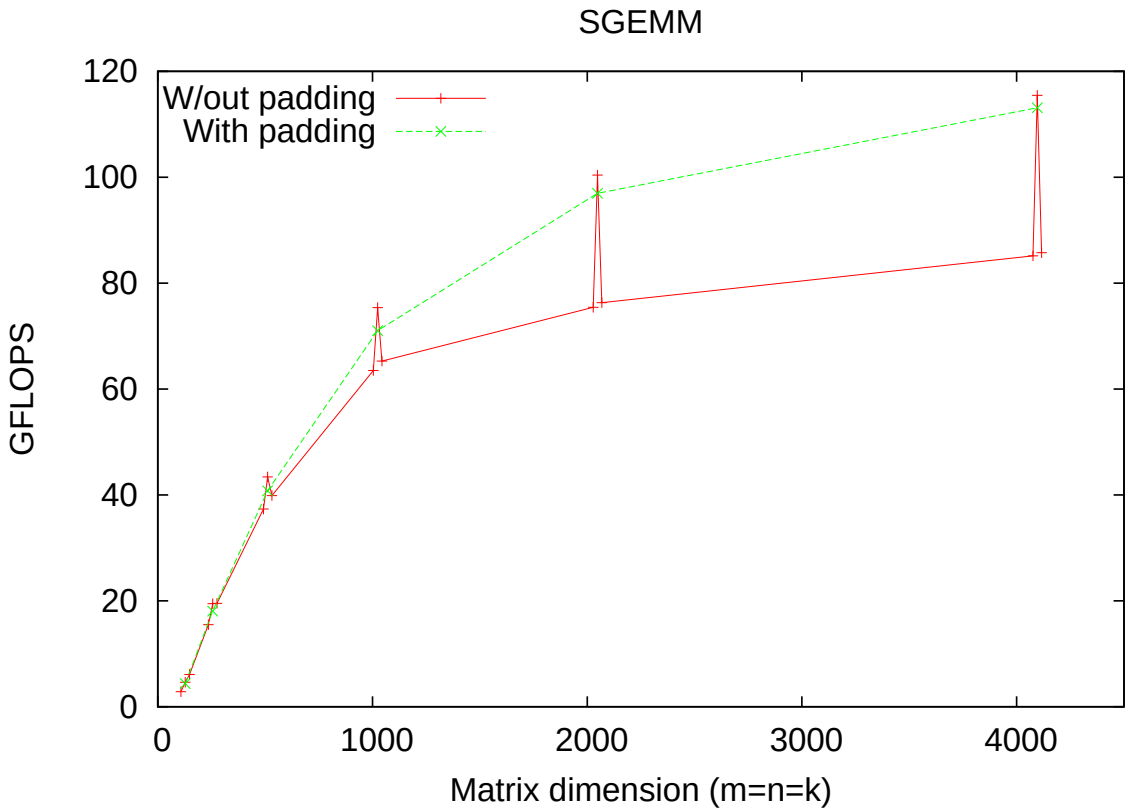
- Peak performance is ∼116 Gflops for GEMM
- Performance attained for large matrices is much better than that of small/medium problems

$$\Downarrow$$

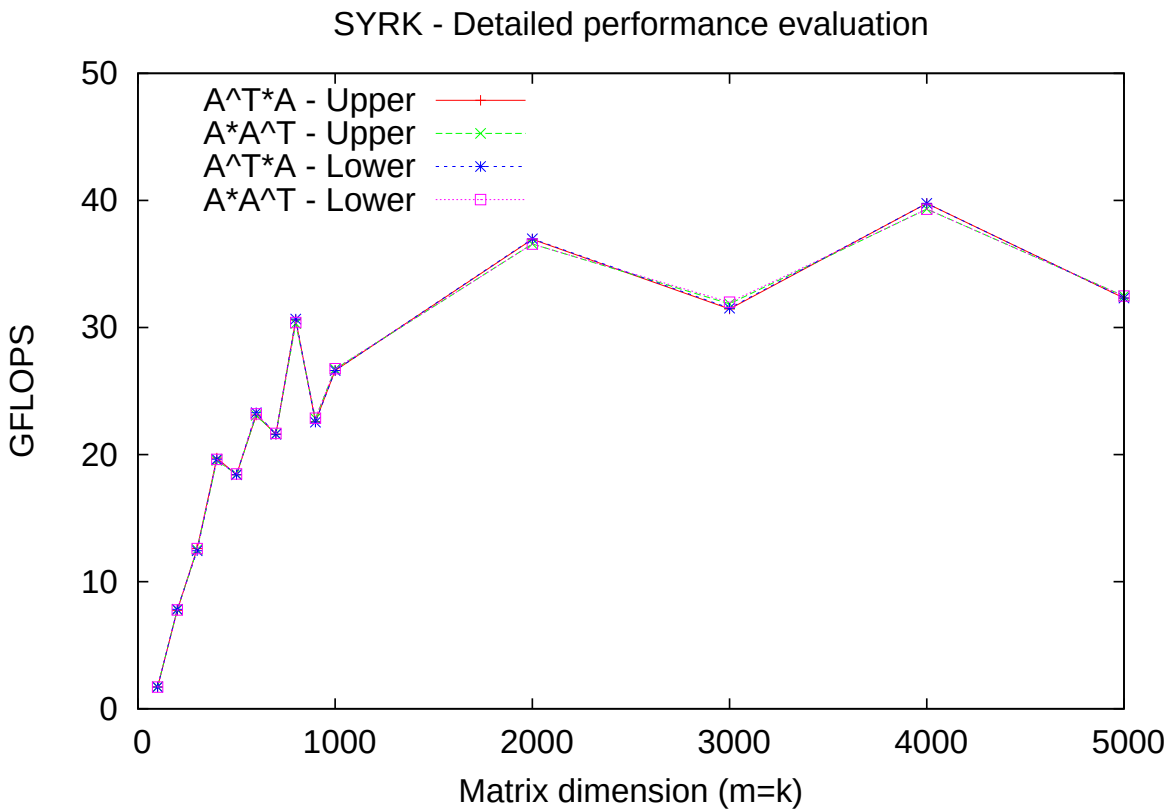
Stream-oriented architecture

- The peak observed for $m = 4000$ is also observed for all dimensions multiple of 32
- Our proposal: padding to improve performance

GEMM Tuning: Padding



SYRK Evaluation



SYRK Evaluation

Main remarks

- Peak performance is ∼40 Gflops for SYRK ⇒ Suboptimal implementation

SYRK Evaluation

Main remarks

- Peak performance is ∼40 Gflops for SYRK ⇒ Suboptimal implementation
- As for GEMM, results attained for big matrices are better

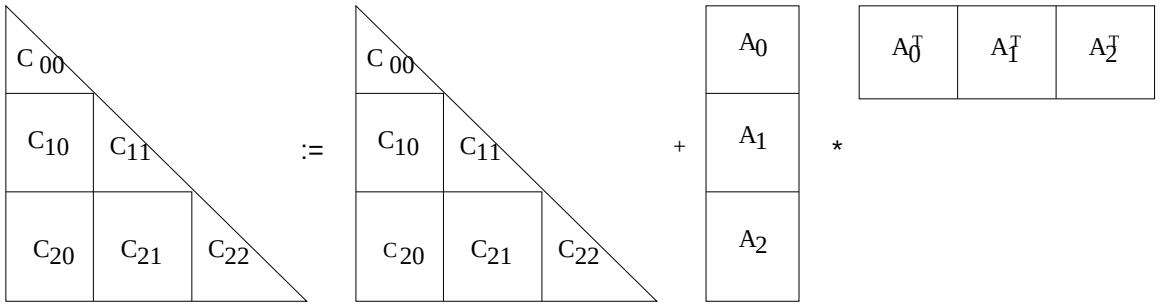
SYRK Evaluation

Main remarks

- Peak performance is ∼40 Gflops for SYRK ⇒ Suboptimal implementation
- As for GEMM, results attained for big matrices are better
- Our proposal: padding and build on top of GEMM

Building SYRK on top of GEMM

- Both SYRK and TRSM yield poor performance compared with that of the GEMM
- Assuming the partitioning for SYRK:

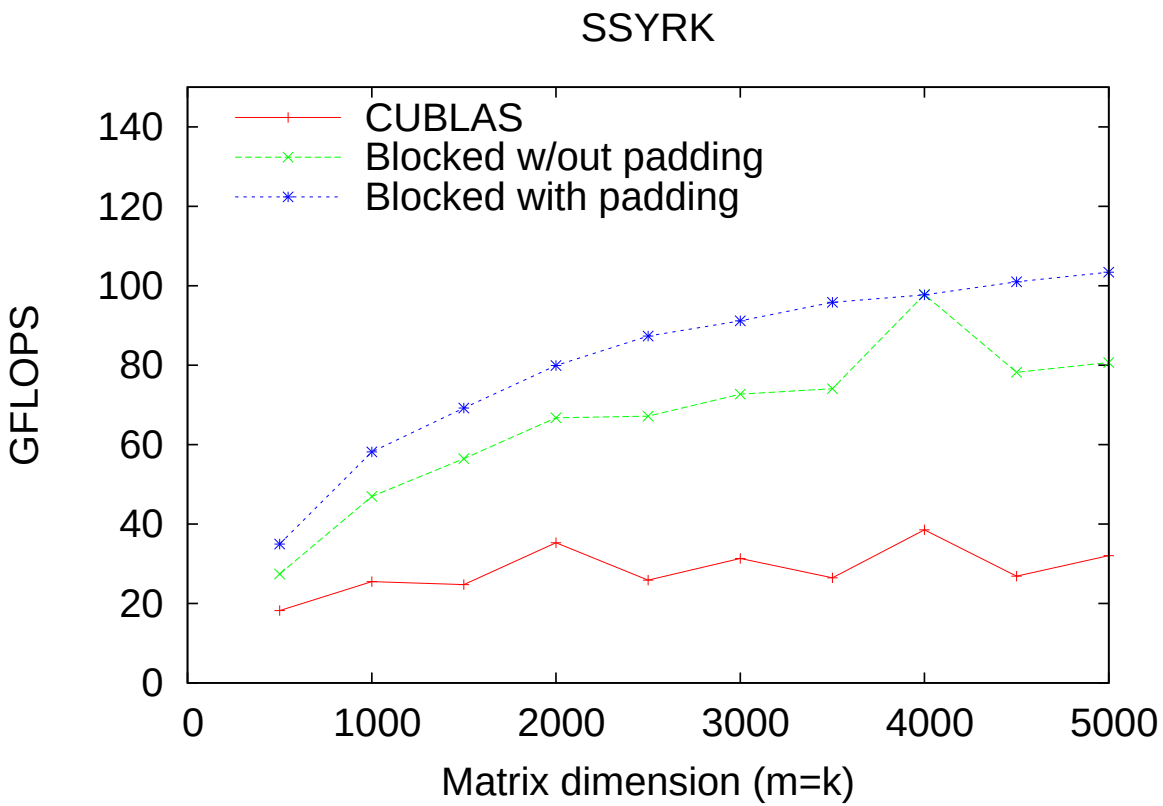


- if the first block of columns of C has been computed, we can proceed by:

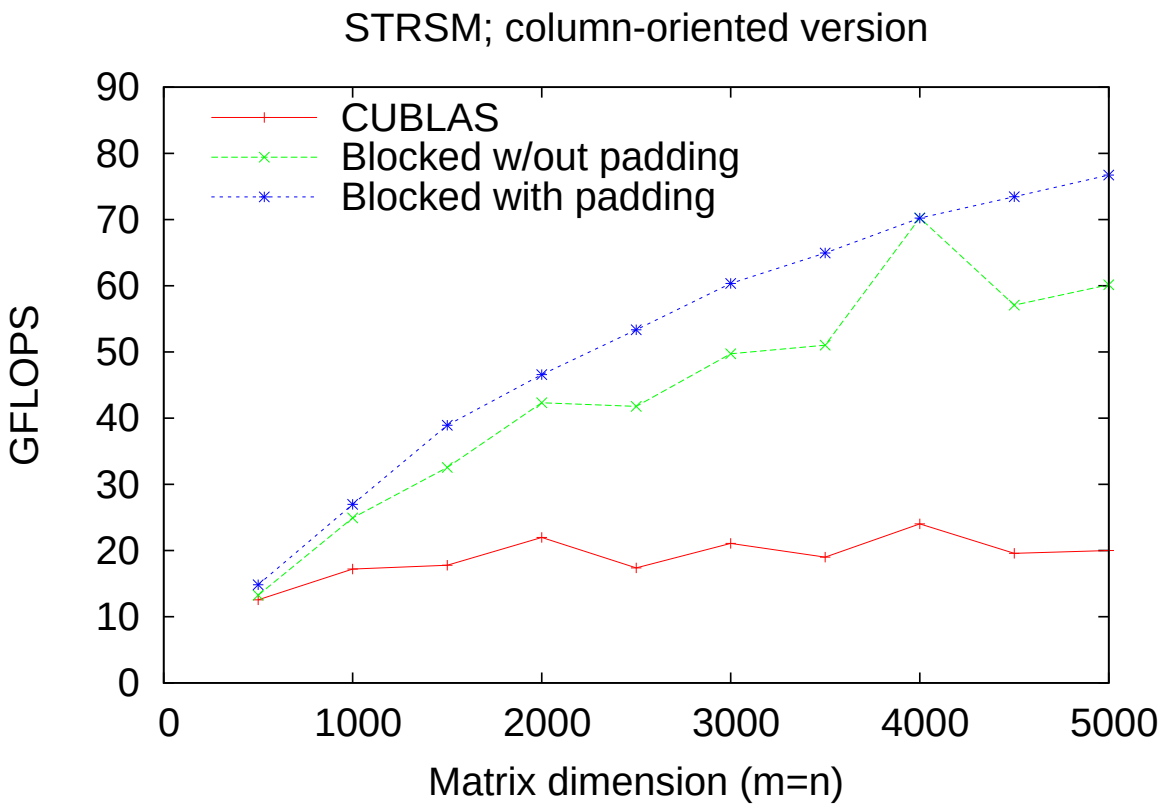
$$C_{11} := \beta \cdot C_{11} + \alpha \cdot A_1 \cdot A_1^T$$

$$C_{21} := \beta \cdot C_{21} + \alpha \cdot A_2 \cdot A_1^T$$

Building SYRK on top of GEMM



Building TRSM on top of GEMM

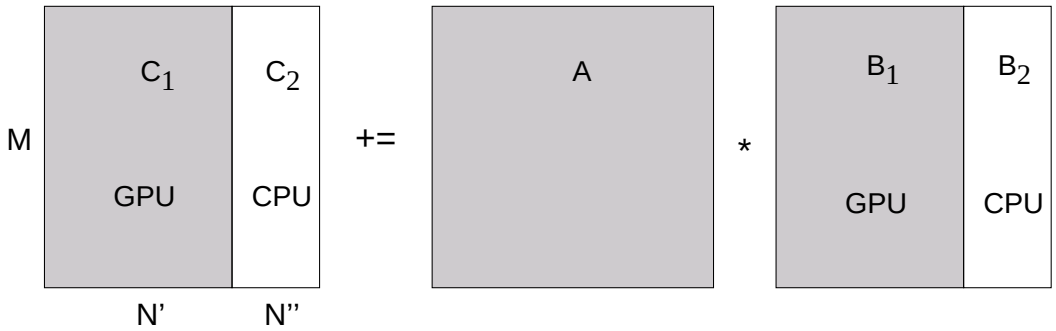


Partitioning for Larger Matrices

- Transfer times GPU$\leftrightsquigarrow$CPU are an important bottleneck
- Our blocked version of GEMM allows to overlap:
  - The partial multiplication A_p and B_p and
  - The transference of the next pair of blocks A_{p+1} and B_{p+1}, being A_p and B_p blocks of columns of A and B, respectively
- Nor CUDA 1.0 neither G80 allow the overlapping of communication and calculation on GPU (supported by more recent systems!)
- An orthogonal benefit: it enables computation with larger matrices that do not fit on GPU memory

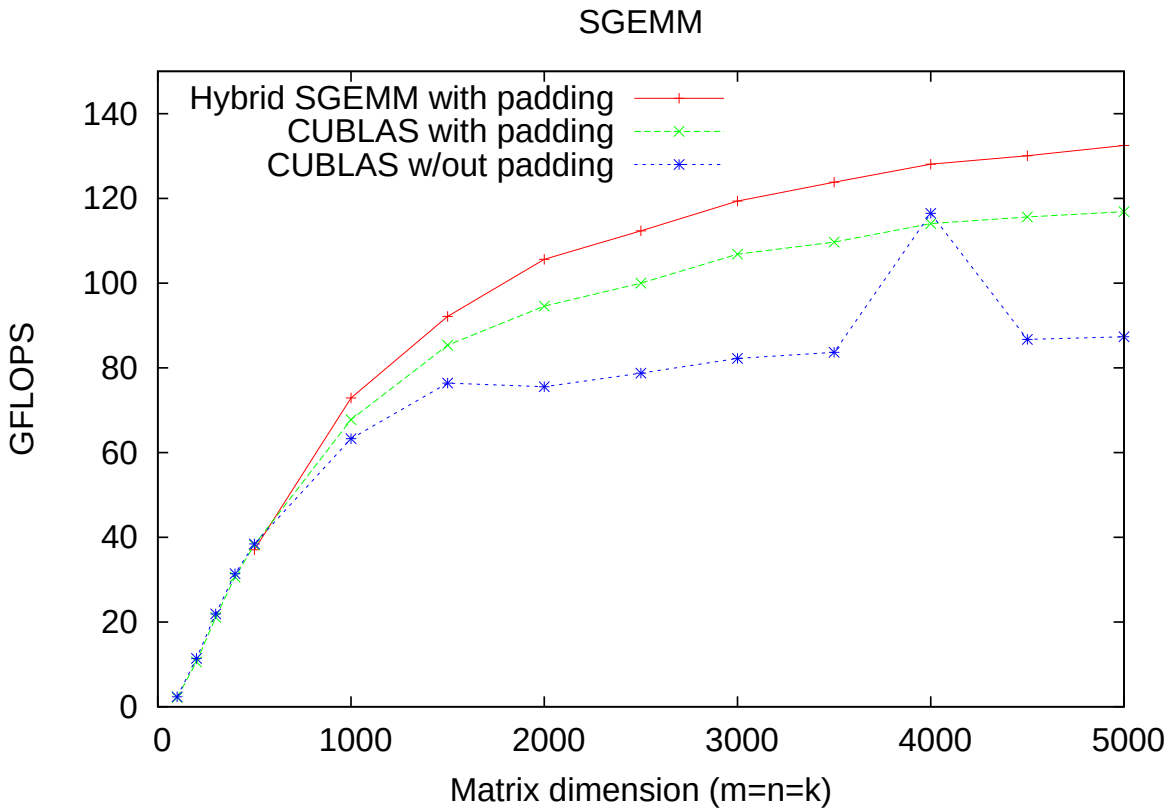
Hybrid Computation

- While GPU is performing a calculation, CPU can also be executing part of the computation
- We implement a hybrid GEMM implementation following the decomposition:



- Values for N' and N'' must be selected carefully to balance the computation load
- Same approach applied to SYRK and TRSM with similar results

Hybrid Computation



Conclusions and Future Work

- Although CUBLAS is a vendor-specific library, it is not highly optimized. In particular, not all routines are equally optimized (GEMM is the best)
- Applying simple ideas, it is possible to attain higher and more predictable results without modifying CUBLAS
- **The source code of CUBLAS has been recently released!!**

$\longrightarrow$

Possible to apply our improvements directly to CUBLAS, and even apply more fine-grained optimizations

Thanks for your interest!

Enrique S. Quintana-Ortí

More information

quintana@icc.uji.es

http://www3.uji.es/~figual/publications.html