

Università di Bologna
CdL in Informatica
Teorie e tecniche del riconoscimento - Pattern recognition
AA 2006/2007

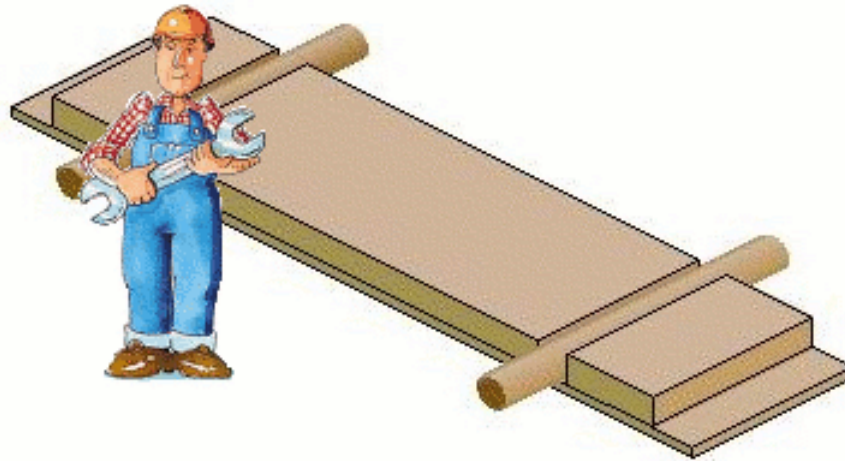
Computational Optimizations

Lesson 2
27 April 2007

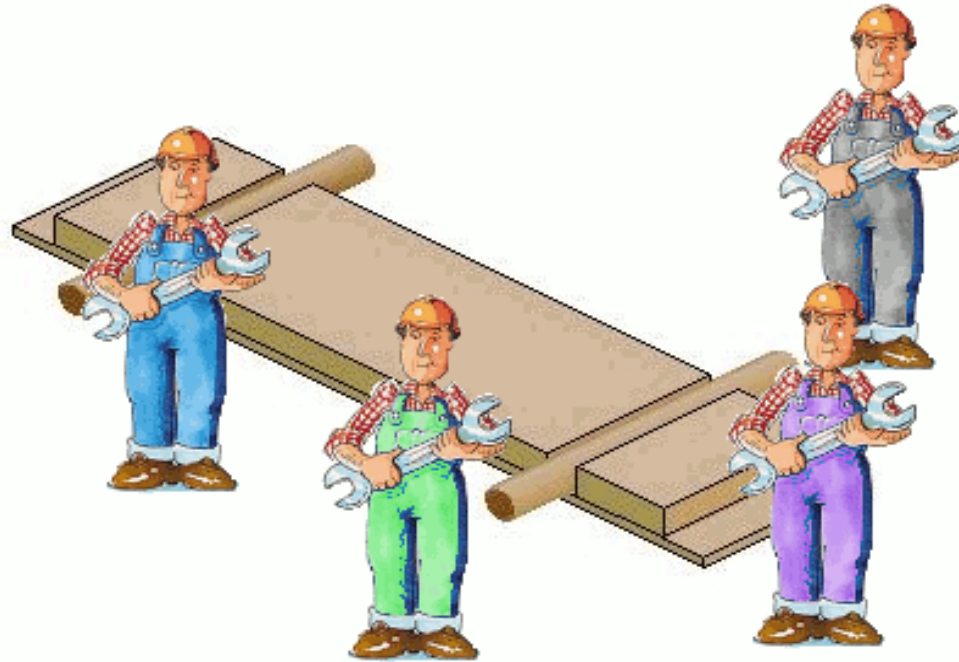
Matteo Roffilli

roffilli@csr.unibo.it
<http://www.cs.unibo.it/people/phd-students/roffilli/>

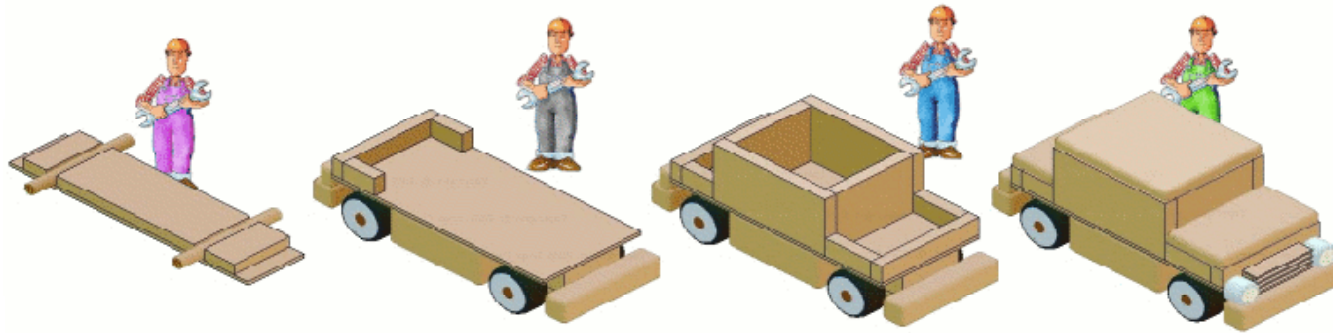
SISD CPU (Standard)



SIMD CPU (MMX,SSE,...)



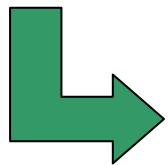
STREAM CPU (GPU)



SVM/RVM testing phase

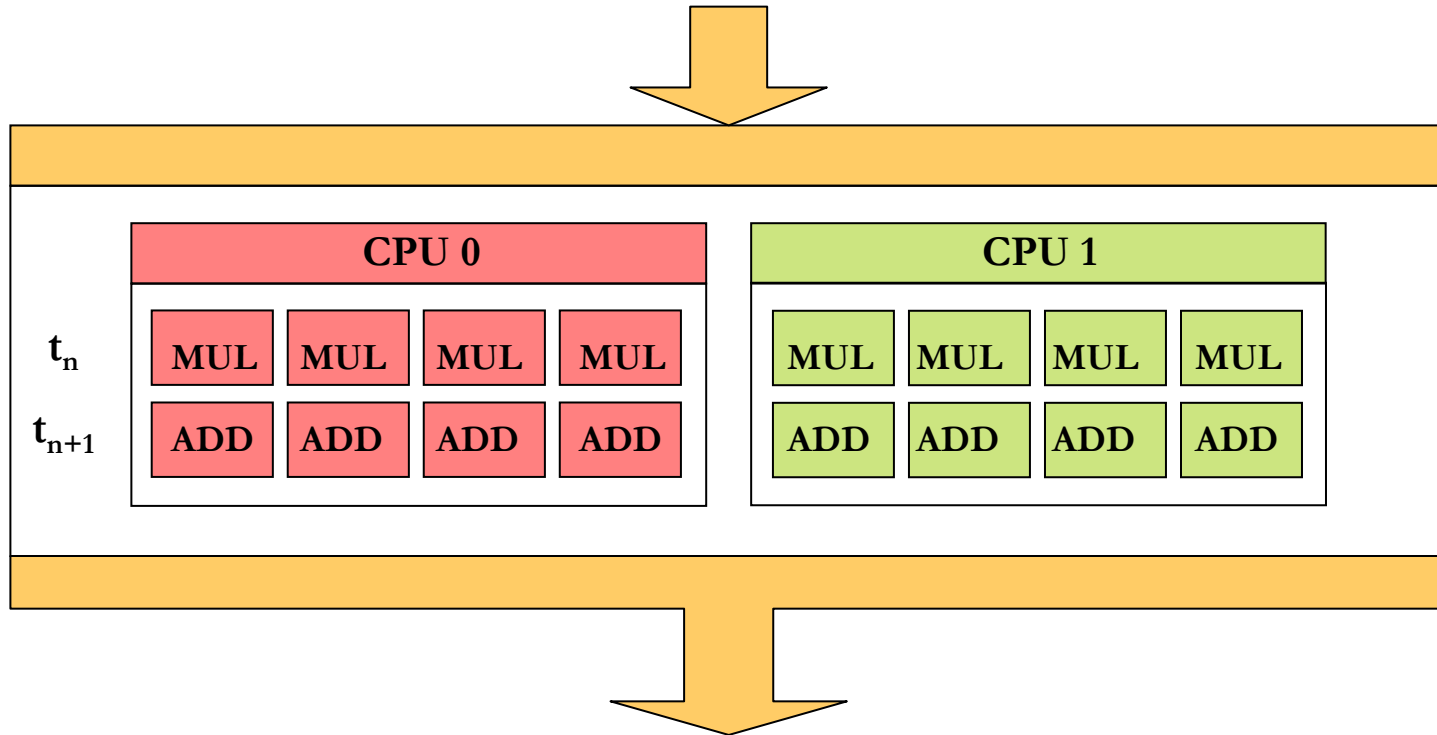
The new sample \mathbf{x} is assigned to class +1 or -1 according to

$$f(\mathbf{x}) = \text{sgn} \left(\sum_{SV} (y_i \alpha_i^0 K(\vec{\mathbf{x}}_i, \vec{\mathbf{x}}) + b^0) \right)$$



Fast and parallelizable

Dot product by SWAR and SMP



1 CPU no SWAR

$\sim 3000 \times 1500 \times 10^5 \times 2$ ops

~ 1000 Giga ops =

~ 5 minutes on 3 GHz CPU

Number of SV ~ 1500

Number of features ~ 3000

Number of samples ~ 100000

2 CPUs with SWAR

$\sim 3000 \times 1500 \times 10^5 \times 2$ ops / 2 / 4

~ 125 Giga ops =

~ 0.5 minutes on 3 GHz CPUs

Assembler built-in

v4sf a,b,c,d,e,f,g,h,m,n;

[...]

a=__builtin_ia32_loadups(va);
b=__builtin_ia32_loadups(vb);

data loading

[...]

a=__builtin_ia32_mulps(a,b);

*Superscalar
multiplication*

[...]

__builtin_ia32_storess (&b2, a);

data store

Assembler hand-made

p_scalar:

```
movaps (%eax), %xmm2
movaps 32(%eax), %xmm3
movaps (%edx), %xmm6
movaps 32(%edx), %xmm7
movaps 16(%eax), %xmm1
movaps 16(%edx), %xmm5
movaps 48(%edx), %xmm0
movaps 48(%eax), %xmm4
```

```
mulps %xmm6, %xmm2
mulps %xmm5, %xmm1
mulps %xmm4, %xmm0
```

```
mulps %xmm7, %xmm3
addps %xmm1, %xmm2
addps %xmm3, %xmm0
subl $16, %ecx
addps -88(%ebp), %xmm2
addps -104(%ebp), %xmm0
addl $64, %eax
addl $64, %edx
testl %ecx, %ecx
```

```
movaps %xmm2, -88(%ebp)
movaps %xmm0, -104(%ebp)
jg .L12
```


ATLAS

ATLAS (**A**utomatically**T**uned**L**inear**A**lgebra **S**oftware)

API BLAS (**B**asic **L**inear**A**lgebra **S**ubroutine)

<http://math-atlas.sourceforge.net/>

```
(void)catlas_sset(ra*rb,1,c,1);
```

Memory set

[...]

```
(void)cblas_sgemm(CblasRowMajor, CblasNoTrans, CblasTrans,  
[...],coef_lin, coef_const, [...]);
```

Matrix Multiply

[...]

```
(void)cblas_scopy(ra*rb,c,1,tmp,1);
```

Memory copy

```
for(i=0;i<rb;i++)
```

Scalar multiply

```
(void)cblas_sscal(ra,alfa[i],c+i,rb);
```

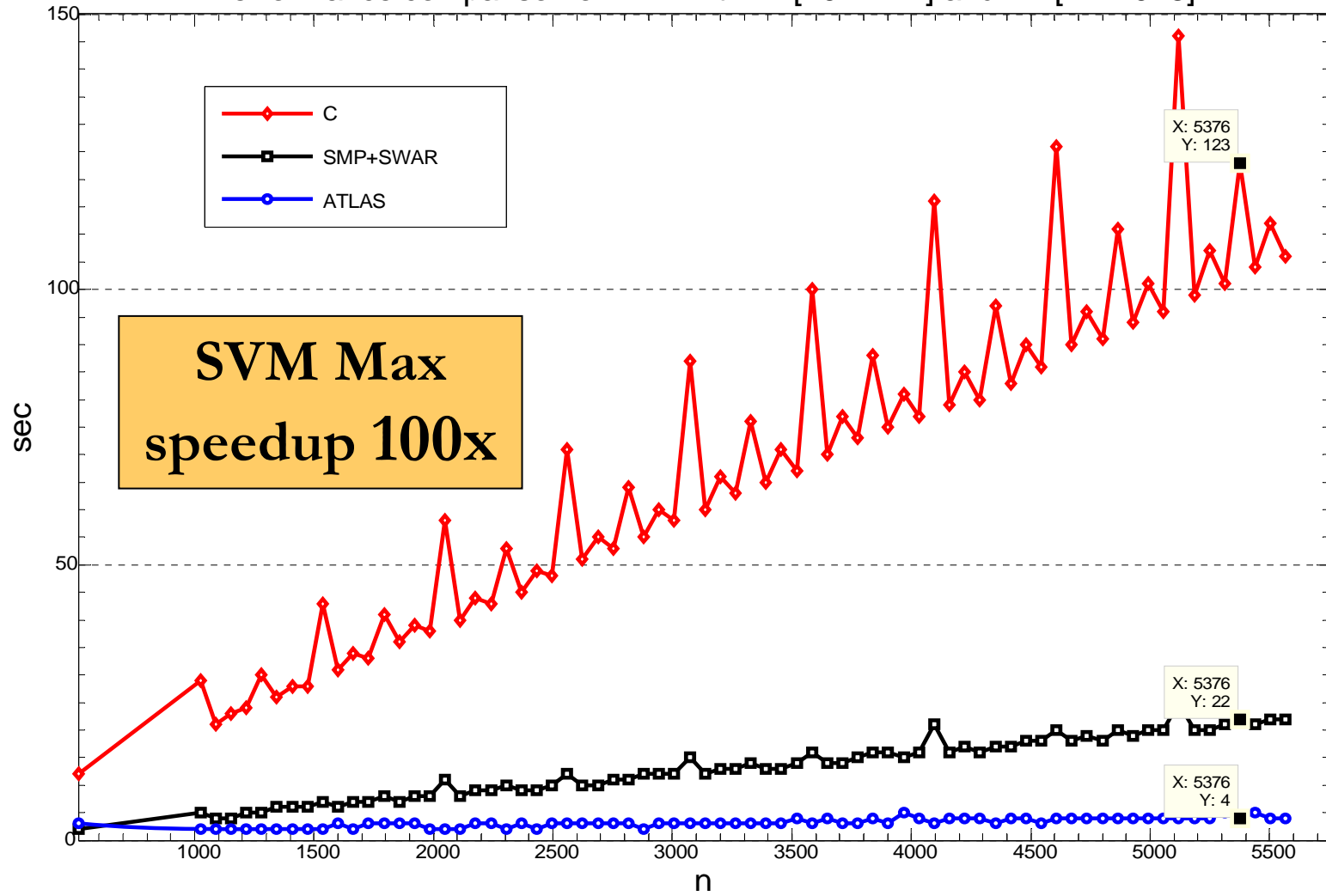
```
for(i=0;i<ra;i++)
```

Dot multiply

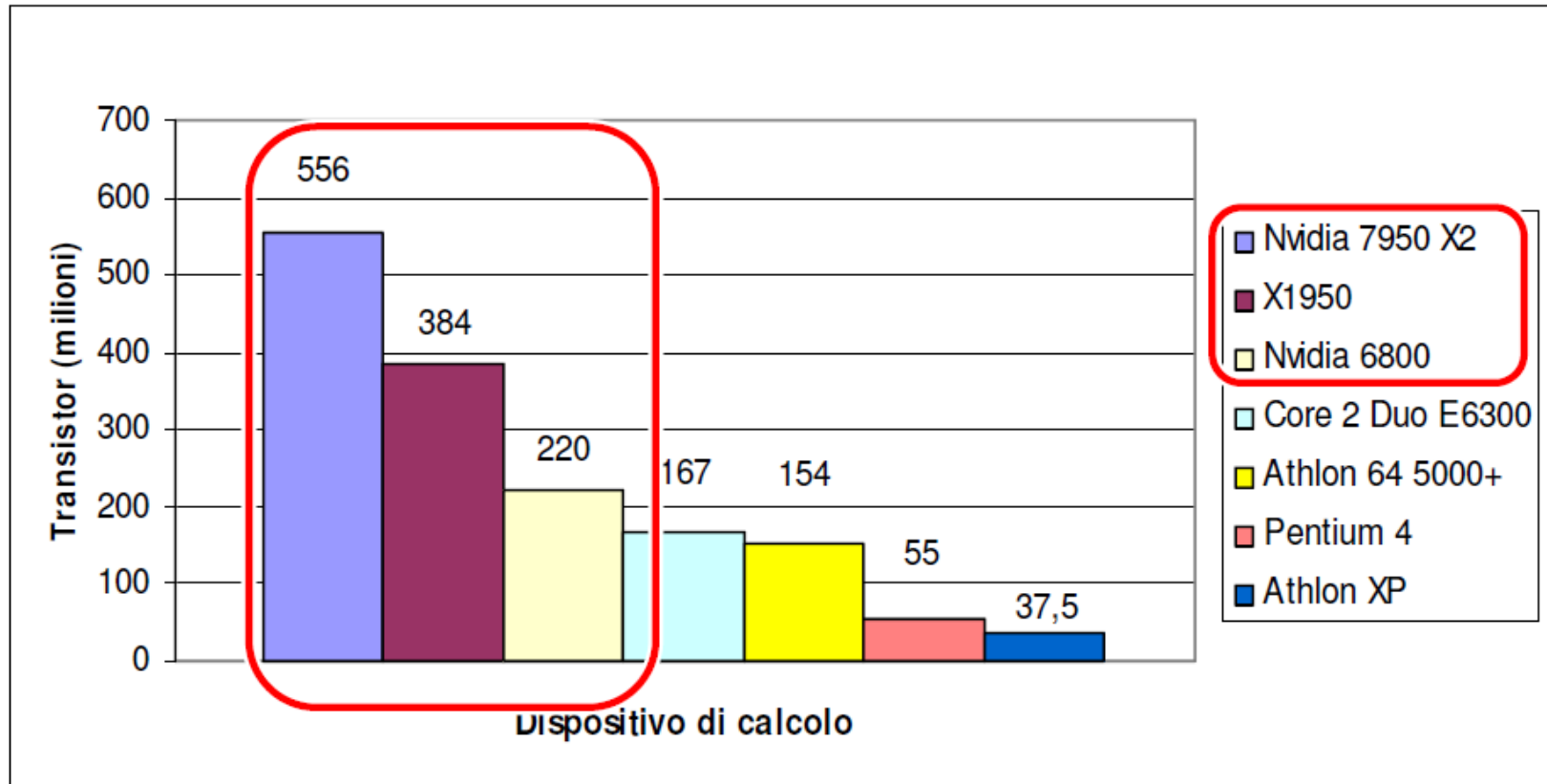
```
result[i]=cblas_sdot(rb,c+i*rb,1,tmp+i*rb,1)-threshold;
```

Performance comparison

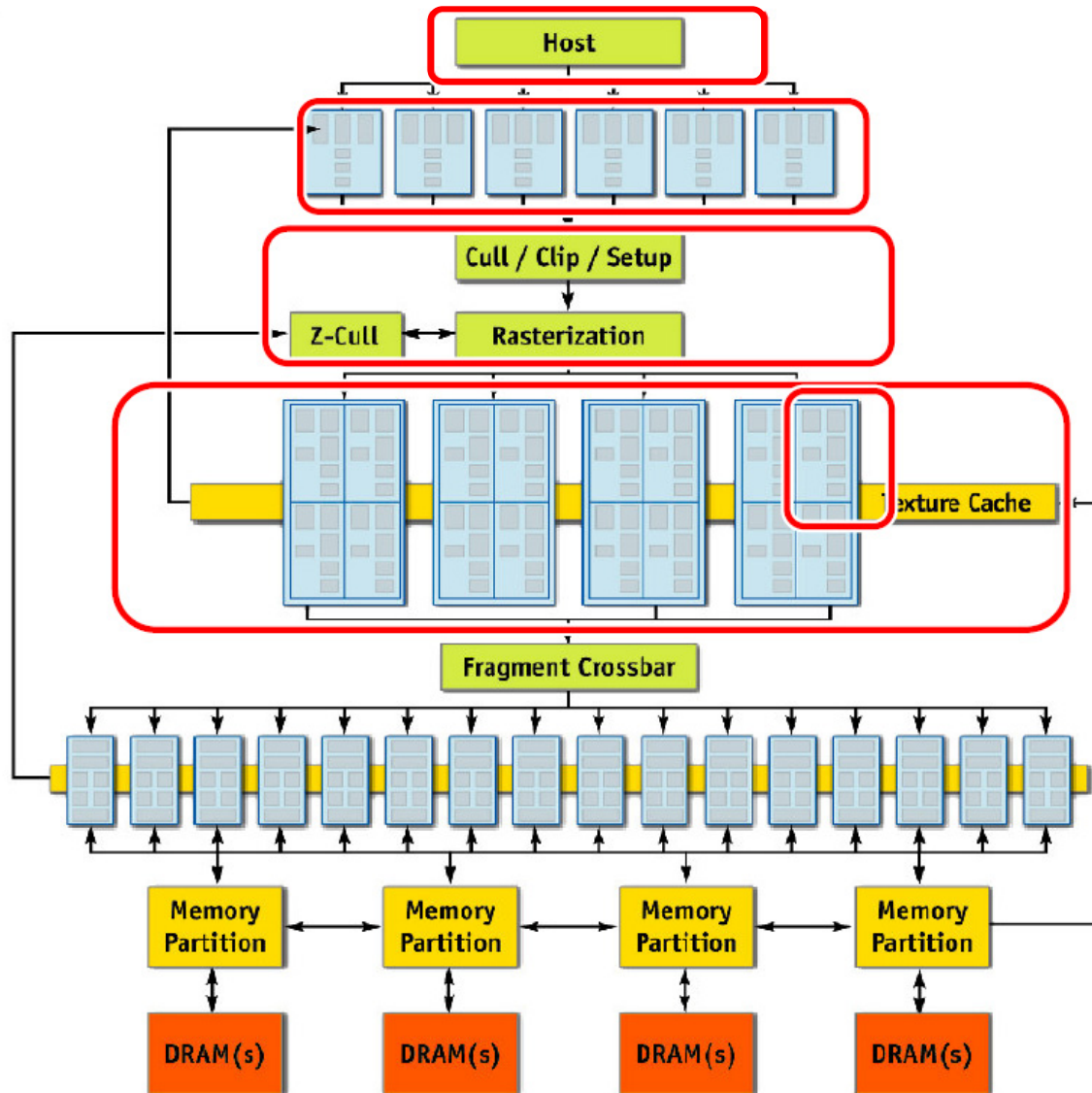
Performance comparison on $A \times B$ with $A=[1024 \times n]$ and $B=[n \times 2048]$



The future computing devices



GPU architecture



GPU programming



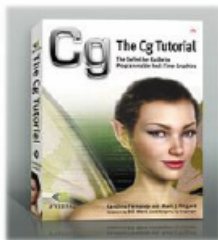
GCC 4.1.1
OpenGL

```
glGenFramebuffersEXT(1, &fb_id);  
glBindFramebufferEXT(GL_FRAMEBUFFER_EXT,  
                     fb_id);  
glEnable (GL_TEXTURE_RECTANGLE_ARB);
```



Pentium 4 a 3.00GHz
1024 Kb di cache

NVIDIA
Cg Compiler
Release 1.4



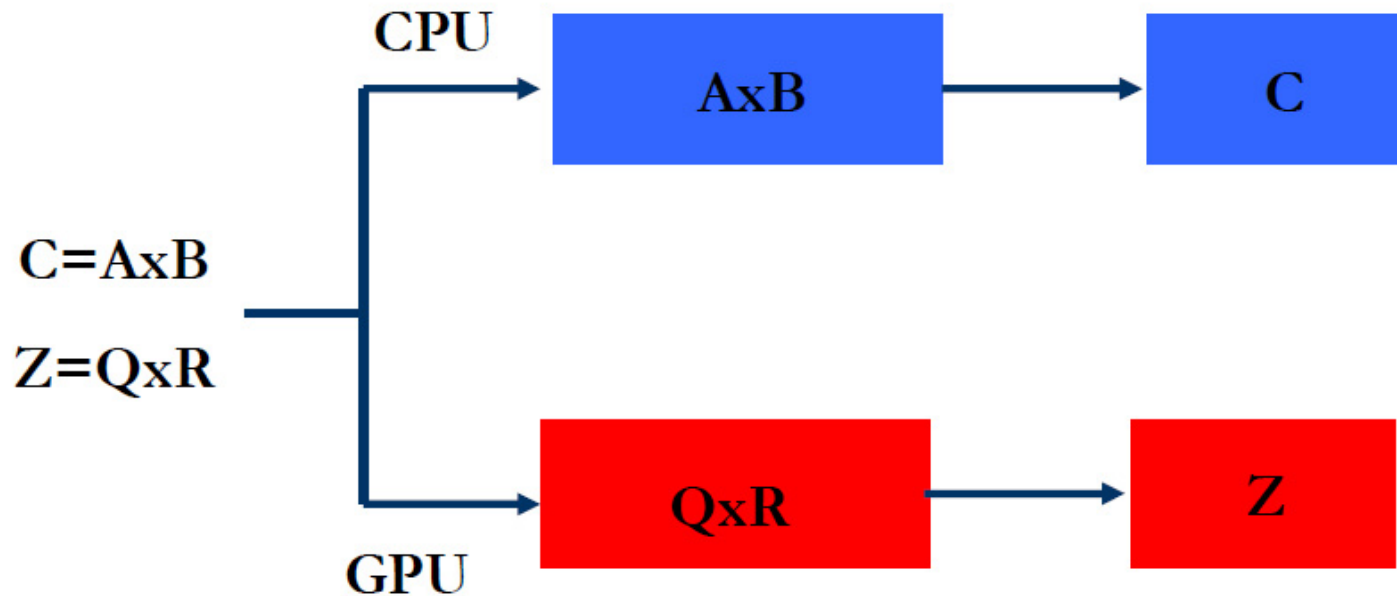
```
void matrixm(
    uniform samplerRECT image,
    float2 texcoord : TEXCOORD0,
    out float4 color0 : COLOR0,
    out float4 color1 : COLOR1)
```



NVIDIA
7800 GT
256Mb di RAM

CPU-GPU mixed programming

1. The idea is to use jointly both the CPU and the GPU (as a math coprocessor)
2. GPU's floating number precision is the same of CPU's one
3. As example: matrix-matrix multiplication as below:



CPU-GPU results

