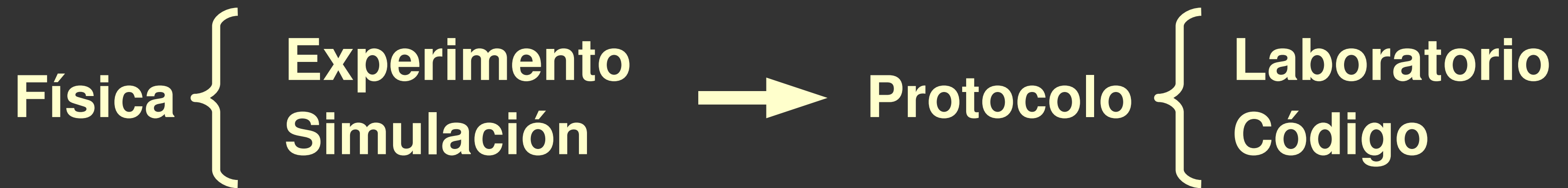
FaMAF | GPGPU Computing Group

No ignorarás el problema

Ezequiel Ferrero
(Juan Pablo De Francesco)

Física computacional



Muestra **Inicialización**

Ambiente/Perturbaciones **Parámetros**

Evolución **Medición**

Promedios **Gráficos**

Dimensión/Geometría
Condiciones de contorno

Grilla o Matriz

Modelo de Potts ferromagnético de q estados

$$H = -J \sum_{nn} \delta(s_i, s_j)$$

$$s_i = 1, \ldots, q$$

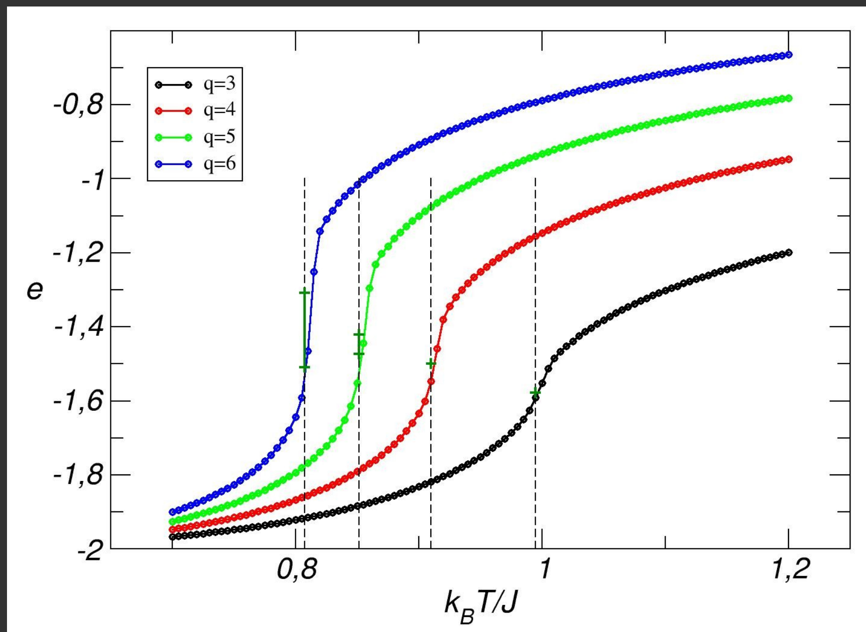
$$\delta(s, s') \begin{cases} 1 & \text{if } s = s' \\ 0 & \text{otherwise} \end{cases}$$

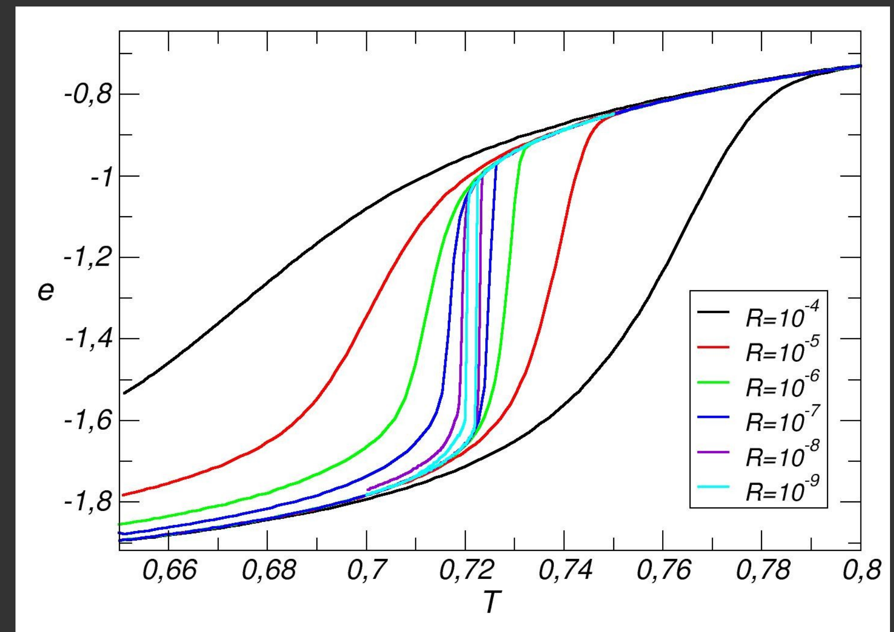
tendencia a alinearse vs. baño térmico

Transición de fase: orden-desorden

Sin dinámica propia

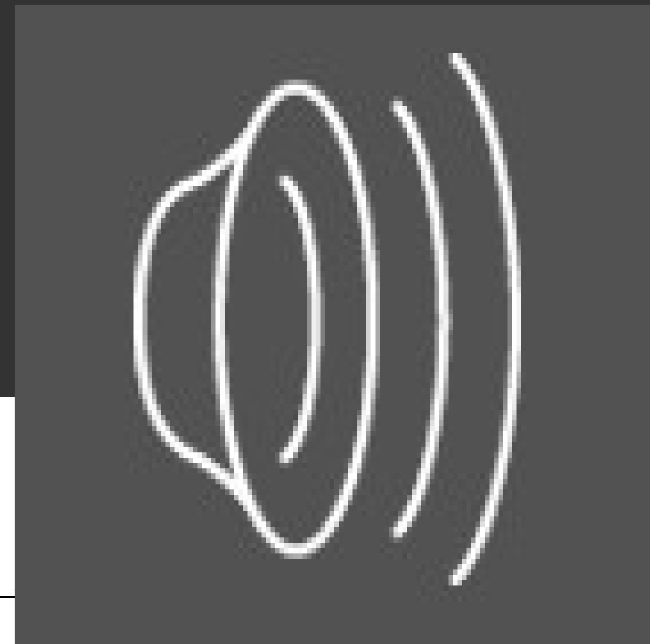
Monte Carlo





Después de un quench a T<<Tc

Coarsening y metaestables



L = 300, T = 0.3

e(t)

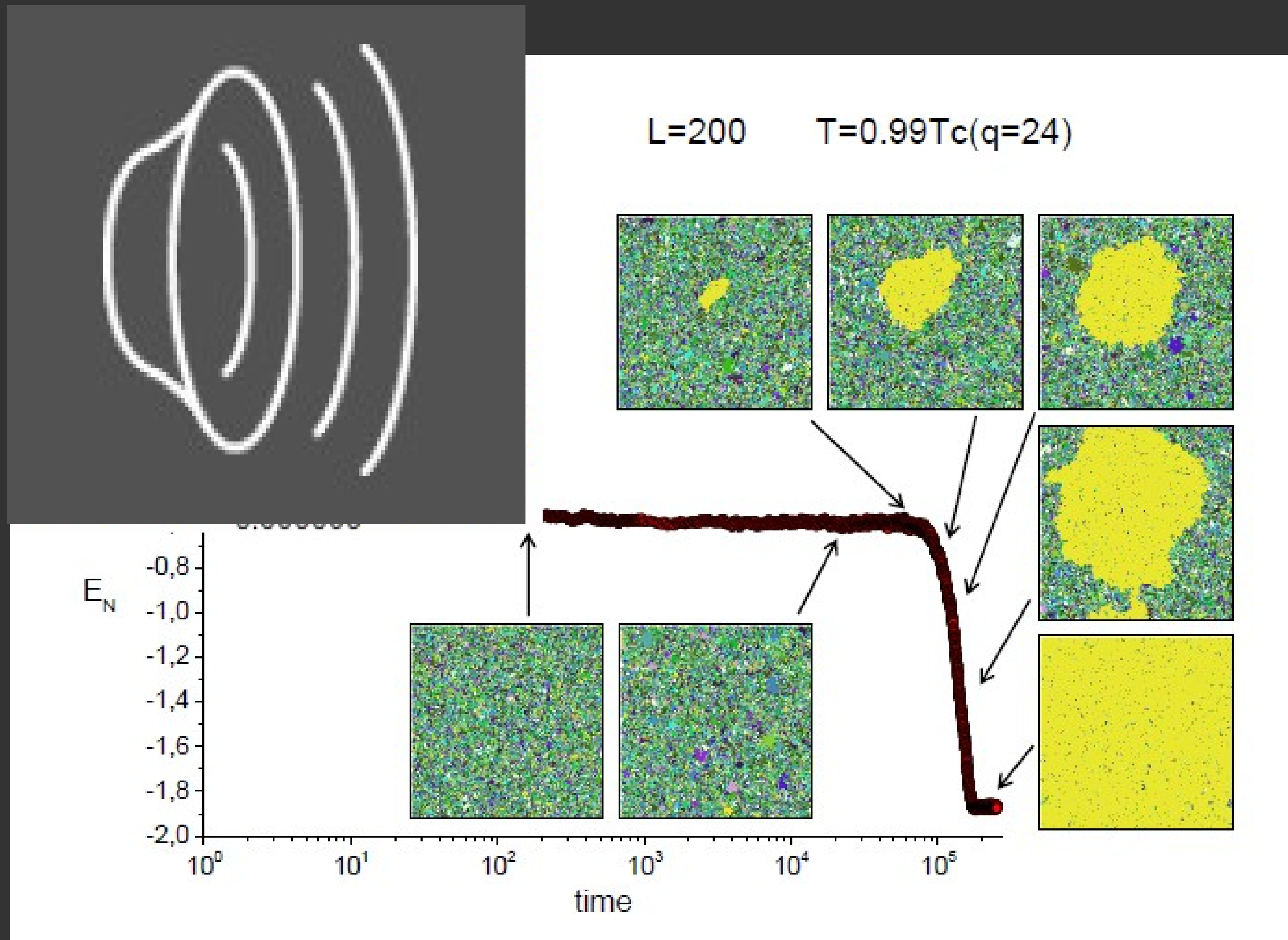
-1,75
-1,80
-1,85
1,90
-1,95
-2,00

10^3 10^4 10^5 10^6 10^7

time [MCS]

Después de un quench a T<~Tc

Nucleación



Monte Carlo

ecuación maestra

$$\frac{d}{dt}P(\mathrm{x},t) = -\sum_{\mathrm{x}'} W(\mathrm{x} \to \mathrm{x}')P(\mathrm{x},t) + \sum_{\mathrm{x}'} W(\mathrm{x}' \to \mathrm{x})P(\mathrm{x}',t)$$

balance detallado

$$P_{eq}(\mathrm{x})W(\mathrm{x} \to \mathrm{x}') = P_{eq}(\mathrm{x}')W(\mathrm{x}' \to \mathrm{x})$$

$$\frac{W(\mathrm{x} \to \mathrm{x}')}{W(\mathrm{x}' \to \mathrm{x})} = exp(-\frac{\delta\mathcal{H}}{k_B T})$$

Algoritmo

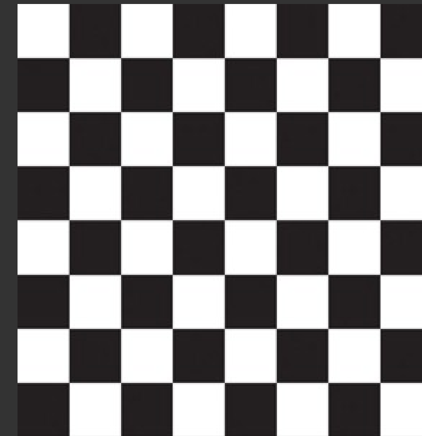
```
for (i=0; i<N; i++){
    choose S_i , backup
    calculate E_before
    "flip" S_i
    calculate E_after
    if (deltaE = E_after-E_before <=0){
    accept the move
    } else {
    pick a random number r and accept the move if
    r<=exp(-DeltaE/T)
    }
}
```

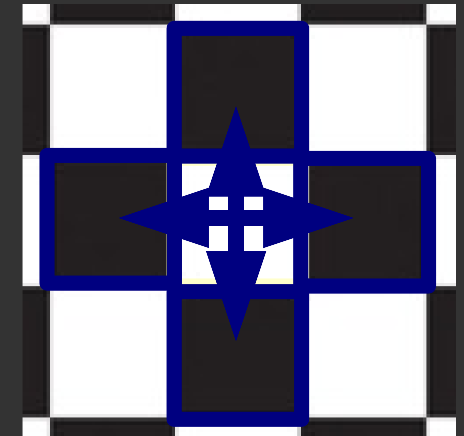
N attemps = 1 MCS (u. de tiempo)

Implementación

estrategia de paralelización

Tablero de ajedrez





c.c. periódicas toro

Subsistemas: black , white

Estructura del programa

```
__global__ void calculateCUDA() {                         }
__global__ void sumupCUDA() {                             }
void sumupRecursive() {
   sumupMCUDA<<<numBlocks, numThreads>>>();
   if () {sumupRecursive();}                              }
float calculate () {
   calculateCUDA<<<dimGrid, dimBlock>>>();
   sumupRecursive();                                      }
__global__ void updateCUDA () {                           }
void update() {
   updateCUDA<<<dimGrid, dimBlock>>>();}
int ciclo () {
      for () {update(); if () {*e = calculate()}; }       }
int muestra() {
   for () {update();}
   e = calculate();
   i = ciclo();                                           }
int ConfigureRandomNumbers(void) {
   int ret = init_RNG();                                  }
int main (void) {
   if (ConfigureRandomNumbers()) return 1;
   for () {muestra();}                                    }
```

update (en host)

```
void update(const float temp, Matrix S, float* energia, int* m) {
    /* Calculo array de probabilidades */
    double tmp_p[9];
    tmp_p[0] = exp(4/temp);
    tmp_p[1] = exp(3/temp);
    tmp_p[2] = exp(2/temp);
    tmp_p[3] = exp(1/temp);
    tmp_p[4] = 1;
    tmp_p[5] = exp(-1/temp);
    tmp_p[6] = exp(-2/temp);
    tmp_p[7] = exp(-3/temp);
    tmp_p[8] = exp(-4/temp);
    /* Copio el arreglo de probabilidades a la memoria constante */
    CUDA_SAFE_CALL(cudaMemcpyToSymbol(p, tmp_p, sizeof(tmp_p)));
    //necesito nFramesxnFrames de 512x512 cada uno
    //int nFrames = L >> 9; //en realidad lo "hardcodeo" en los for's
    /* 512 en vez de L (ya que largo cuadrados de 512x512) */
    dim3 dimBlock(16, 16);
    dim3 dimGrid((512 + dimBlock.x - 1)/dimBlock.x,
            (512/2 + dimBlock.y - 1)/dimBlock.y);
    /* Actualizo las blancas -> necesito los valores de las negras */
    updateCUDA<<<dimGrid, dimBlock>>>(temp, S, WHITES, S.blacks);
    CUDA_SAFE_CALL(cudaThreadSynchronize());
    /* Actualizo las negras -> necesito los valores de las blancas */
    updateCUDA<<<dimGrid, dimBlock>>>(temp, S, BLACKS, S.whites);
    CUDA_SAFE_CALL(cudaThreadSynchronize());
}
```

UNC Universidad Nacional de Córdoba

updateCUDA (en device) 1/2

```
__global__ void updateCUDA (float temp, Matrix S, int casilleros, byte *cas_lectura) {
  int jOriginal = blockIdx.x*blockDim.x + threadIdx.x;
  int iOriginal = blockIdx.y*blockDim.y + threadIdx.y;

  if (iOriginal<512/2 && jOriginal<512) {
    unsigned int tid = iOriginal*512 + jOriginal;
    int h_before, h_after, delta_E;
    byte old_color, color, q;
    byte tmp1, tmp2, tmp3, tmp4;
    double aleatorio;
    int i, j;

    /* Los limites de los for corresponden a cuantos frames de 512 necesito */
    for (int iFrame=0; iFrame<(L >> 9); iFrame++) {
      i = iOriginal + (512/2)*iFrame;
      for (int jFrame=0; jFrame<(L >> 9); jFrame++) {
        j = jOriginal + 512*jFrame;
        if (casilleros == WHITES) {
          old_color = S.whites[i*S.width + j];
        } else {
          old_color = S.blacks[i*S.width + j];
        }

        /* Calculo de h_before */
```


UNC Universidad Nacional de Córdoba

updateCUDA (en device) 2/2

```
/* Calculo de h_before */
tmp1 = cas_lectura[i*L + j];
tmp2 = cas_lectura[i*L + (j+1+L)%L];
tmp3 = cas_lectura[i*L + (j-1+L)%L];
tmp4 = cas_lectura[((i+(2*(casilleros^(j%2))-1)+L/2)%(L/2))*L + j];
h_before = (old_color == tmp1 ? 1 : 0)
        + (old_color == tmp2 ? 1 : 0)
        + (old_color == tmp3 ? 1 : 0)
        + (old_color == tmp4 ? 1 : 0);
/* Propongo un nuevo valor */
q = 1 + rand_MWC_co(&d_x[tid], &d_a[tid])*(Q-1);
color = (old_color + q) % Q;
/* Calculo de h_after (con el nuevo color) */
h_after = (color == tmp1 ? 1 : 0)
       + (color == tmp2 ? 1 : 0)
       + (color == tmp3 ? 1 : 0)
       + (color == tmp4 ? 1 : 0);
/* Decido si acepto el cambio o no */
delta_E = h_before - h_after;
aleatorio = rand_MWC_co(&d_x[tid], &d_a[tid]);
if (!(delta_E>0 &&  aleatorio>p[delta_E+4])) {
    if (casilleros == WHITES) {
        S.whites[i*S.width + j] = color;
    } else {
        S.blacks[i*S.width + j] = color;
    }
}
```

UNC Universidad Nacional de Córdoba

calculate (en host)

```
float calculate (const Matrix S, int* M, int* max_m) {
	float energia = 0.0;

	dim3 dimBlock(16, 16);
	dim3 dimGrid((L + dimBlock.x - 1)/dimBlock.x,
	             (L/2 + dimBlock.y - 1)/dimBlock.y);

	unsigned int numBlocks = ((L + dimBlock.x - 1)/dimBlock.x)
	                         *((L/2 + dimBlock.y - 1)/dimBlock.y);

	int *energia_matrix;
	size_t size = L * (L/2) * sizeof(int);
	CUDA_SAFE_CALL(cudaMalloc((void**) &energia_matrix, size));

	unsigned int *M_matrix;
	size = numBlocks * Q * sizeof(unsigned int);
	CUDA_SAFE_CALL(cudaMalloc((void**) &M_matrix, size));

	calculateCUDA<<<dimGrid, dimBlock>>>(S, energia_matrix, M_matrix);
	cudaThreadSynchronize();

	sumupRecursive(energia_matrix, L*L/2);
	sumupMRecursive(M_matrix, numBlocks);

	int tmp_energia=0;
	CUDA_SAFE_CALL(cudaMemcpy(&tmp_energia, energia_matrix, sizeof(int),
	                          cudaMemcpyDeviceToHost));
	energia = (float) -tmp_energia;

	unsigned int M_inner[Q];
	CUDA_SAFE_CALL(cudaMemcpy(M_inner, M_matrix, Q*sizeof(unsigned int),
	                          cudaMemcpyDeviceToHost));

	unsigned int tmp_max_m=0;
	for (int k=0; k < Q; k++) {
		tmp_max_m = (M_inner[k] > tmp_max_m) ? M_inner[k] : tmp_max_m;
	}
	*max_m = (int) tmp_max_m;


	cudaFree(M_matrix);
	cudaFree(energia_matrix);

	return(energia/2.0);
}
```

calculateCUDA (en device) 1/2

```
__global__ void calculateCUDA(Matrix S, int *energia_matrix,  unsigned int *M_matrix) {
    int j = blockIdx.x*blockDim.x + threadIdx.x;
    int i = blockIdx.y*blockDim.y + threadIdx.y;

    __shared__ unsigned int M[Q];

    if (threadIdx.x<Q) {
        M[threadIdx.x]=0;
    }
    __syncthreads();

    if (i<L/2 && j<L) {
        byte color;
        byte tmp1, tmp2, tmp3, tmp4;
        int restando;
        int casilleros = WHITES;
        color = S.whites[i*L + j];
        tmp1 = S.blacks[i*L + j];
        tmp2 = S.blacks[i*L + (j+1+L)%L];
        tmp3 = S.blacks[i*L + (j-1+L)%L];
        tmp4 = S.blacks[(((i+(2*(casilleros^(j%2))-1)*1+L/2)%(L/2))*L+j];
        restando = (color == tmp1 ? 1 : 0) + (color == tmp2 ? 1 : 0)
                + (color == tmp3 ? 1 : 0) + (color == tmp4 ? 1 : 0);

        atomicAdd(&(M[color]), 1);
```

calculateCUDA (en device) 1/2

```
        casilleros = BLACKS;
        color = S.blacks[i*L + j];
        tmp1 = S.whites[i*L + j];
        tmp2 = S.whites[i*L + (j+1+L)%L];
        tmp3 = S.whites[i*L + (j-1+L)%L];
        tmp4 = S.whites[((i+(2*(casilleros^(j%2))-1)*1+L/2)%(L/2))*L+j];
        restando += (color == tmp1 ? 1 : 0) + (color == tmp2 ? 1 : 0)
        + (color == tmp3 ? 1 : 0) + (color == tmp4 ? 1 : 0);

        atomicAdd(&(M[color]), 1);

        energia_matrix[i*L+j] = restando;   /* toda la energia junta de las casillas blancas y negras */
    }

    __syncthreads();
    if (threadIdx.x<Q) {
        M_matrix[(blockIdx.y*gridDim.x + blockIdx.x)*Q + threadIdx.x]
            = M[threadIdx.x];
    }
}
```

UNC Universidad Nacional de Córdoba

sumupRecursive (en host)

```
void sumupRecursive(int *array, unsigned long numElements) {
    unsigned int blockSize = 256; /* HARCODEADO EN sumupCUDA */
    /* numElements y blockSize seran siempre potencia de 2 */
    unsigned int numBlocks = MAX(1, numElements / (2 * blockSize));
    unsigned int numThreads = blockSize;

    if (numBlocks == 1)
        numThreads = numElements / 2;

    if (numBlocks > 1) {
        sumupCUDA<<<numBlocks, numThreads>>>(array);
        cudaThreadSynchronize(); //creo que no hace falta

        sumupRecursive(array, numBlocks);
    } else {
        sumupCUDA<<<numBlocks, numThreads>>>(array);
        cudaThreadSynchronize(); //creo que no hace falta
    }
}
```

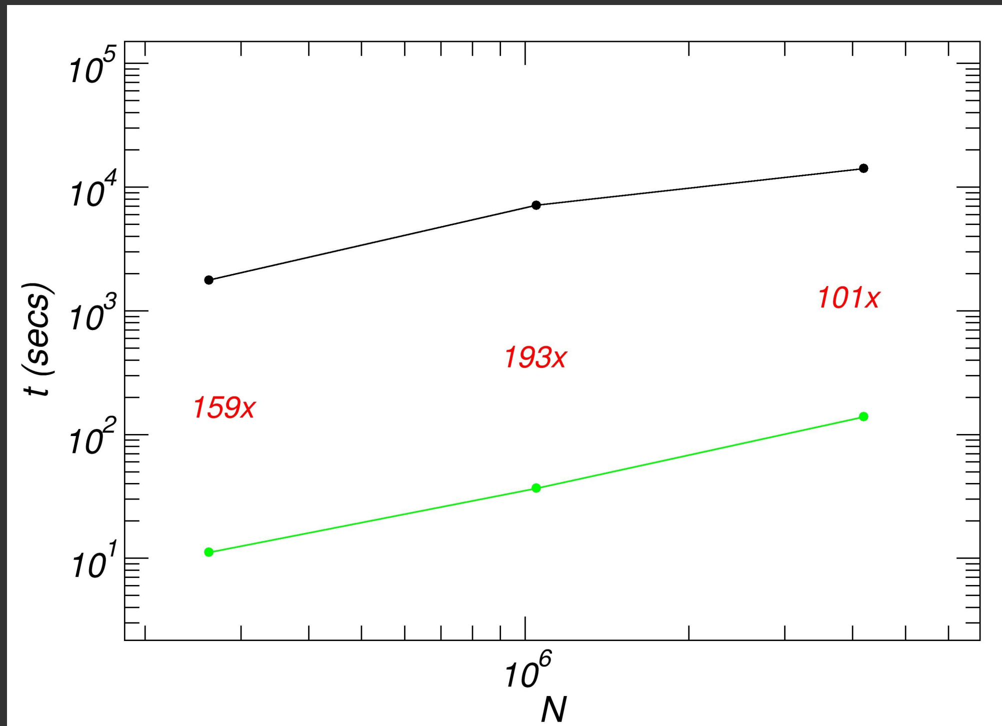
sumupCUDA (en device)

```
__global__ void sumupCUDA(int *array) {
    int i = threadIdx.x;
    int k = blockIdx.x;
    int dim = blockDim.x;

    /* traer a shared los 512 del array que se necesita
    __shared__ int perBlockArray[512];

    perBlockArray[i      ] = array[k*2*dim + i];
    perBlockArray[i + dim] = array[k*2*dim + i + dim];
    __syncthreads();

    unsigned int stride;
    for (stride=dim; stride>0; stride>>=1) {
        if (i < stride) {
            perBlockArray[i] = perBlockArray[i]
                        + perBlockArray[i+stride];
        }
        __syncthreads();
    }

   //en el i=0 esta la suma de todo el bloque
   if (i == 0)
       array[k] = perBlockArray[0];
}
```

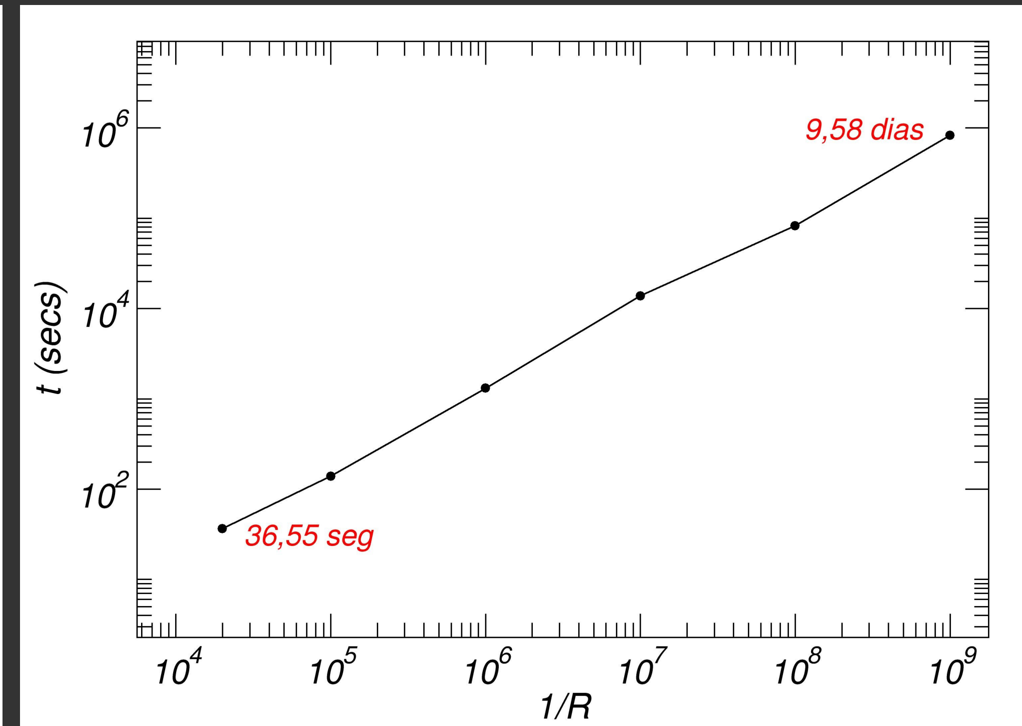
Universidad Nacional de Córdoba

Performance

¿100x?¿200x?



Lo que ahora podemos hacer



UNC
Universidad
Nacional
de Córdoba

end