Notas: kCSDA

KCSDA completo

$$C^*(\mathbf{x}) = \widetilde{\mathbf{K}}^T(\mathbf{x}) \cdot \mathbf{K}^{-1} \cdot \mathbf{V}.$$

kCSDA (la parte que se hizo)

- El cálculo largo, después de otras rutinas más sofisticadas, pero computacionalmente ligeras.

$$mat[j,k] = \sum_{l=0}^{elecUtiles} b_j(x_k - x_l + c) * b_j(x_j - x_l + c)$$

  - elecUtiles < 4096
  - Mat = [4096,4096]

- La versión en julia tarda 11 horas.

¿Qué se necesitaría?

- Una función en CUDA/C++ que se pueda llamar desde Julia.
- calculak(MatrixIntegrales, listaElectrodos): Matriz K

CUDA

- Primer versión, a la final le faltarían más parámetros para personalizar más.

```
__global__ void calculateK(float * d_out, float * d_in,float * d_jlist, float * d_klist,int block, int electrodes, int matdim, int origin){
        int k = block * blockIdx.x + threadIdx.x; //column
    int j = block * blockIdx.y + threadIdx.y; //row

    if(j >= k && j < electrodes)
    {
            float sum = 0;
            //Ver lo de los indices matriz julia vs c++
            for (int l = 0; l < electrodes; ++l)
            {
                //Coordenadas funcionan igual con desfase [1]
                //CoordenadasTotal[j]
                int xj1 = (int) d_jlist[j];//(int) ceil((double) j/ (double) matdim);
                int xj2 = (int) d_klist[j];//j % matdim;
                //CoordenasTodal[k]
                int xk1 = (int) d_jlist[k];//(int) ceil((double) k/ (double) matdim);
                int xk2 = (int) d_klist[k];//k % matdim;
                //Coordenastotal[l]
                int xl1 = (int) d_jlist[l];//(int) ceil((double) l/ (double) matdim);
                int xl2 = (int) d_klist[l];//l % matdim;
                //Matrix bj is stored as an array: Col + Row*dim;
                //xk-xl+const
                int idx1 = xk2-xl2+origin + (xk1-xl1+origin)*matdim;
                //xj-xl+const
                int idx2 = xj2-xl2+origin + (xj1-xl1+origin)*matdim;
                sum += d_in[idx1] * d_in[idx2];//d_in[xj2 + xj1*matdim];
            }
            //Equivalente a d_out[j,k] = sum
        d_out[k + electrodes*j] = sum;
    }
}
```

Profile para algunas matrices

```
Nes (master *) P2kCSDA $ nvprof ./kCSDAKCUDA.out BceroDura.dat
==74494== NVPROF is profiling process 74494, command: ./kCSDAKCUDA.out BceroDura.dat
==74494== Profiling application: ./kCSDAKCUDA.out BceroDura.dat
==74494== Profiling result:
Time(%)      Time     Calls       Avg       Min       Max  Name
 97.95%  41.334ms         1  41.334ms  41.334ms  41.334ms  [CUDA memcpy DtoH]
  1.92%  811.53us         1  811.53us  811.53us  811.53us  calculateK(float*, float*, float*, float*, int, int, int, int)
  0.13%  53.601us         3  17.867us  6.6240us  40.321us  [CUDA memcpy HtoD]

==74494== API calls:
Time(%)      Time     Calls       Avg       Min       Max  Name
 64.69%  79.375ms         4  19.844ms  4.2790us  78.708ms  cudaMalloc
 34.81%  42.713ms         4  10.678ms  31.906us  42.568ms  cudaMemcpy
  0.23%  285.29us         2  142.64us  137.54us  147.75us  cudaFree
  0.12%  147.74us        91  1.6230us     137ns  54.507us  cuDeviceGetAttribute
  0.11%  130.14us         1  130.14us  130.14us  130.14us  cudaLaunch
  0.03%  35.272us         1  36.272us  36.272us  36.272us  cuDeviceGetName
  0.01%  7.9710us         1  7.9710us  7.9710us  7.9710us  cuDeviceTotalMem
  0.00%  5.4500us         3  1.8160us     322ns  4.0250us  cuDeviceGetCount
  0.00%  3.8240us         3  1.2740us     354ns  3.1110us  cuDeviceGet
  0.00%  2.8680us         1  2.8680us  2.8680us  2.8680us  cudaConfigureCall
  0.00%  2.2310us         8     278ns     170ns     642ns  cudaSetupArgument
```

```
==75353== NVPROF is profiling process 75353, command: ./kCSDAKCUDA.out BceroDura.dat 256
==75353== Profiling application: ./kCSDAKCUDA.out BceroDura.dat 256
==75353== Profiling result:
Time(%)      Time     Calls       Avg       Min       Max  Name
 88.75%  41.005ms         1  41.005ms  41.005ms  41.005ms  [CUDA memcpy DtoH]
 11.14%  5.1449ms         1  5.1449ms  5.1449ms  5.1449ms  calculateK(float*, float*, float*, float*, int, int, int, int)
  0.12%  53.633us         3  17.877us  6.6560us  40.321us  [CUDA memcpy HtoD]

==75353== API calls:
Time(%)      Time     Calls       Avg       Min       Max  Name
 60.19%  71.923ms         4  17.981ms  4.1730us  71.269ms  cudaMalloc
 39.22%  46.864ms         4  11.716ms  31.649us  46.717ms  cudaMemcpy
  0.27%  320.94us         2  160.47us  150.46us  170.48us  cudaFree
  0.19%  228.31us        91  2.5080us     236ns  85.960us  cuDeviceGetAttribute
  0.08%  94.430us         1  94.430us  94.430us  94.430us  cudaLaunch
  0.04%  42.439us         1  42.439us  42.439us  42.439us  cuDeviceGetName
  0.01%  14.872us         1  14.872us  14.872us  14.872us  cuDeviceTotalMem
  0.00%  5.3630us         3  1.7870us     298ns  3.6800us  cuDeviceGetCount
  0.00%  3.1370us         3  1.0450us     410ns  2.1120us  cuDeviceGet
  0.00%  3.0260us         1  3.0260us  3.0260us  3.0260us  cudaConfigureCall
  0.00%  2.1150us         8     264ns     163ns     593ns  cudaSetupArgument
```

```
Nes (master *) P2kCSDA $ nvprof ./kCSDAKCUDA.out BceroDura.dat 1024
==75568== NVPROF is profiling process 75568, command: ./kCSDAKCUDA.out BceroDura.dat 1024
==75568== Profiling application: ./kCSDAKCUDA.out BceroDura.dat 1024
==75568== Profiling result:
Time(%)      Time     Calls       Avg       Min       Max  Name
 91.03%  261.66ms         1  261.66ms  261.66ms  261.66ms  calculateK(float*, float*, float*, float*, int, int, int, int)
  8.96%  25.742ms         1  25.742ms  25.742ms  25.742ms  [CUDA memcpy DtoH]
  0.02%  53.568us         3  17.856us  6.6240us  40.320us  [CUDA memcpy HtoD]

==75568== API calls:
Time(%)      Time     Calls       Avg       Min       Max  Name
 78.43%  288.12ms         4  72.031ms  33.383us  287.97ms  cudaMemcpy
 21.38%  78.538ms         4  19.634ms  7.8910us  77.843ms  cudaMalloc
  0.11%  403.54us         2  201.77us  175.35us  228.19us  cudaFree
  0.04%  145.47us        91  1.5980us     137ns  52.900us  cuDeviceGetAttribute
  0.02%  80.121us         1  80.121us  80.121us  80.121us  cudaLaunch
  0.01%  30.591us         1  30.591us  30.591us  30.591us  cuDeviceGetName
  0.00%  9.8280us         1  9.8280us  9.8280us  9.8280us  cuDeviceTotalMem
  0.00%  4.6280us         3  1.5420us     220ns  3.5480us  cuDeviceGetCount
  0.00%  4.0110us         3  1.3370us     264ns  3.3700us  cuDeviceGet
  0.00%  2.9340us         1  2.9340us  2.9340us  2.9340us  cudaConfigureCall
  0.00%  2.1570us         8     269ns     175ns     619ns  cudaSetupArgument
```

Profile 4096

```
Nes (master *) P2kCSDA $ nvprof ./kCSDAKCUDA.out BceroDura.dat 4096
==78225== NVPROF is profiling process 78225, command: ./kCSDAKCUDA.out BceroDura.dat 4096
==78225== Profiling application: ./kCSDAKCUDA.out BceroDura.dat 4096
==78225== Profiling result:
Time(%)      Time     Calls       Avg       Min       Max  Name
 99.60%  7.64961s         1  7.64961s  7.64961s  7.64961s  calculateK(float*, float*, float*, float*, int, int,
int, int)
  0.40%  30.432ms         1  30.432ms  30.432ms  30.432ms  [CUDA memcpy DtoH]
  0.00%  53.568us         3  17.856us  6.6240us  40.288us  [CUDA memcpy HtoD]

==78225== API calls:
Time(%)      Time     Calls       Avg       Min       Max  Name
 98.93%  7.68106s         4  1.92026s  32.677us  7.68088s  cudaMemcpy
  1.01%  78.243ms         4  19.561ms  4.5860us  77.608ms  cudaMalloc
  0.06%  4.4228ms         2  2.2114ms  320.88us  4.1019ms  cudaFree
  0.00%  160.30us        91  1.7610us     146ns  57.366us  cuDeviceGetAttribute
  0.00%  97.287us         1  97.287us  97.287us  97.287us  cudaLaunch
  0.00%  30.095us         1  30.095us  30.095us  30.095us  cuDeviceGetName
  0.00%  9.3390us         1  9.3390us  9.3390us  9.3390us  cuDeviceTotalMem
  0.00%  4.3640us         3  1.4540us     438ns  3.2420us  cuDeviceGetCount
  0.00%  3.0770us         3  1.0250us     238ns  2.1110us  cuDeviceGet
  0.00%  2.9710us         1  2.9710us  2.9710us  2.9710us  cudaConfigureCall
  0.00%  2.3310us         8     291ns     188ns     616ns  cudaSetupArgument
```

números

	CPU (Julia)	CUDA	Speedup
128	3.31	0.042198	78.4397364804019
256	25.70	0.046167877	556.664106517179
512		0	
1024	1615	0.28745357	5618.29863515002
4096	39600	7.7	5142.85714285714

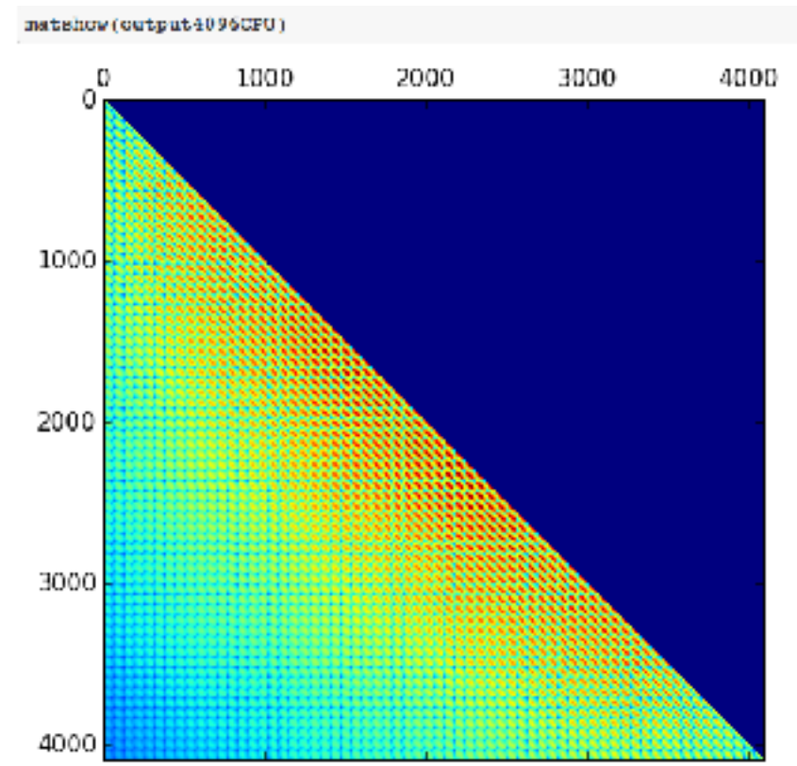
- Es injusto comparar Julia contra un código c++/CUDA compilado para cierta arquitectura

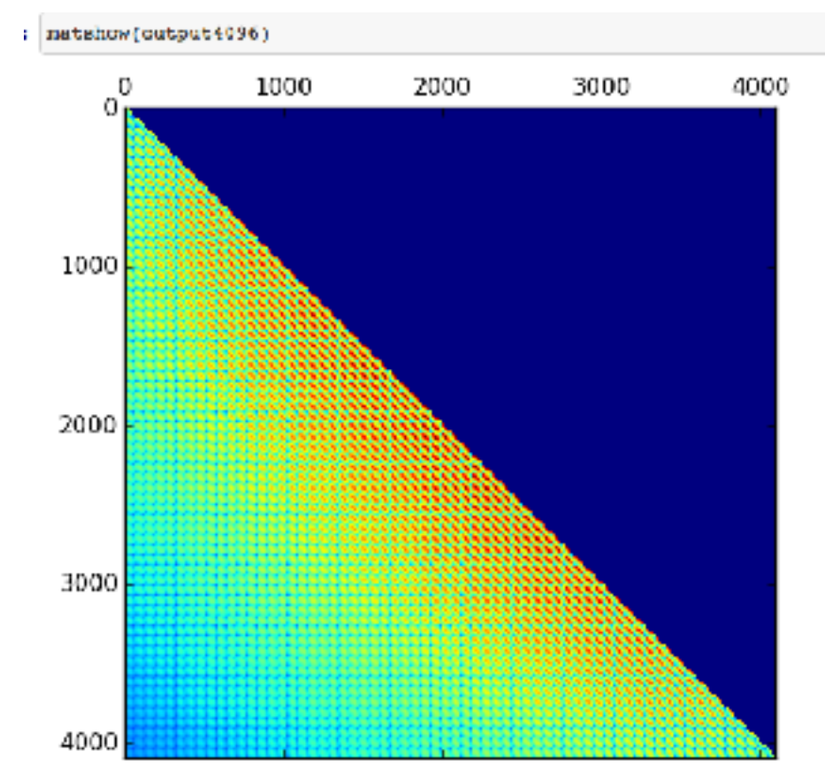
Comparamos Resultados

- La diferencia entre las matrices es menor a 5 en cantidades algoE6

```
output4096CPU-output4096

4096x4096 Array{Float64,2}:
 -1.11196    -1.61951e-40   0.0        …  0.0        0.0         0.0
  0.269068    5.06764       0.0           0.0        0.0         0.0
  1.45398    -7.49404       3.11137       0.0        0.0         0.0
 -1.23431     2.64183      -1.74425       0.0        0.0         0.0
  3.32749    -9.6563        6.68857       0.0        0.0         0.0
 -4.19083    -1.45836      -6.33655   …  0.0        0.0         0.0
  5.57403    -4.14337       6.54169       0.0        0.0         0.0
  2.71655    -0.0572879    -8.2542        0.0        0.0         0.0
 -0.324091    4.69657       6.67286       0.0        0.0         0.0
 -7.4143     -1.66378       5.92073       0.0        0.0         0.0
 -8.88219    -2.07926       2.70686   …  0.0        0.0         0.0
 -7.10682    -5.53716      -1.20834       0.0        0.0         0.0
 -2.33938    -6.26615      -3.8959        0.0        0.0         0.0
  ⋮                                    ⋱                         ⋮
 -2.54955    -3.45059      -4.20973       0.0        0.0         0.0
 -2.77231     0.497615     -1.15312   …  0.0        0.0         0.0
  2.13094    -6.64792      -5.66806       0.0        0.0         0.0
 -0.657779    2.2682       -0.623351      0.0        0.0         0.0
  1.70721     0.0311962    -3.29167       0.0        0.0         0.0
 -2.83673    -5.52027       4.06964       0.0        0.0         0.0
 -1.91733    -2.15462       3.55422   …  0.0        0.0         0.0
  0.555747   -3.96857       0.880967      0.0        0.0         0.0
  3.22677    -2.15645       5.02301       0.0        0.0         0.0
 -1.83375    -3.6373        0.085442      3.11137    0.0         0.0
  2.85997     0.470067     -3.6373        2.50596   -4.93236     0.0
  0.405453    2.85997      -1.83375   …  1.45398    0.269068   -1.11196
```





Problemas: A veces

- Luego hay que ejecutar la función varias veces para que haga algo

- Al hacer el profile a veces no sirve.

```
==75456== NVPROF is profiling process 75456, command: ./kCSDAKCUDA.out BceroDura.dat 512
==75456== Profiling application: ./kCSDAKCUDA.out BceroDura.dat 512
==75456== Warning: Found 1 invalid records in the result.
==75456== Warning: This can happen if device ran out of memory or if a device kernel was stopped due to an assertion.
==75456== Profiling result:
Time(%)      Time     Calls       Avg       Min       Max  Name
100.00%  135.82us         3  45.273us  29.660us  61.167us  [CUDA memcpy HtoD]

==75456== API calls:
Time(%)      Time     Calls       Avg       Min       Max  Name
 98.90%  72.635ms         4  18.159ms  5.0560us  71.501ms  cudaMalloc
  0.43%  317.99us         2  159.00us  2.5760us  315.42us  cudaFree
  0.23%  167.83us         4  41.958us  8.5550us  64.309us  cudaMemcpy
  0.20%  145.27us        91  1.5960us     135ns  54.299us  cuDeviceGetAttribute
  0.17%  126.22us         1  126.22us  126.22us  126.22us  cudaLaunch
  0.04%  27.363us         1  27.363us  27.363us  27.363us  cuDeviceGetName
  0.01%  8.8080us         1  8.8080us  8.8080us  8.8080us  cuDeviceTotalMem
  0.01%  4.9410us         3  1.6470us     271ns  3.7270us  cuDeviceGetCount
  0.00%  3.4330us         1  3.4330us  3.4330us  3.4330us  cudaConfigureCall
  0.00%  3.0250us         3  1.0080us     322ns  2.3360us  cuDeviceGet
  0.00%  2.2260us         8     278ns     165ns     673ns  cudaSetupArgument
```

Dudas a investigar

- ¿Float vs double?
- ¿Es julia realmente tan lento? Debería compararlo con una versión de c++ como benchmark inicial
- Tamaño del bloque en CUDA
- Memoria?
- ¿Por qué es tan rápido?¿BW de la tarjeta?¿Miles de cores?¿Qué parte de la memoria se está usando?
- Aquí no se puede usar Amdahl??
- El tamaño del problema sí cambia