From sequential to GPU, case study

Mehdi AMINI – PIPS DAYS 25/10/2010

On stage

Host	MAGGIE	RIDEE	VIVALDI
Cuda	3,0	3,2	3,1
CPU	2212 @2GHz	E6600 @2.40GHz	X5670@3GHz
GPU	8800 GTX	C1060	C2050
Multiproc	16	24	14
Cores	128	192	448
H2D	1.65GB/s	1.8GB/s	3.7GB/s
D2H	0.875GB/s	1.3GB/s	3.2GB/s
D2D	10GB/s	80GB/s	90GB/s
Registers/multiproc	8k	16k	32k
Shared	16k	16k	48k
Cache	Texture-only	Texture-only	768k L2 16k L1
Clock	1.35GHz	1.10GHz	1.15GHz

Example code 1

```
void part2data(float pos[N][3], int *data) {
  int i;
  float x, y, z;
  for (i = 0; i < N; i++) {
    x = pos[i][0];
    y = pos[i][1];
    z = pos[i][2];
    data[i] = (int)(x / DX) * N*N +
              (int)(y / DX) * N +
              (int)(z / DX);
  }
}
```

Maggie	56128
Ridee	81689
Vivaldi	47309

Example code 1
Kernel « naïve »

```
__global__ void Kpart2data(float pos[N][3], int *data) {
  int tx = threadIdx.x;
  int bx = blockIdx.x;
  int dimx = blockDim.x;

  int i = bx * dimx + tx; // indice de boucle
  float x, y, z;

  x = pos[i][0];
  y = pos[i][1];
  z = pos[i][2];
  data[i] = (int)(x / DX) *N*N +
            (int)(y / DX) *N +
            (int)(z / DX);
}
```

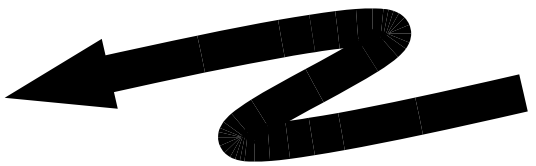
Maggie	3234	x17.36
Ridee	**935**	**x87.37**
Vivaldi	1018	x46.47

Example code 1
Kernel « naïve »

```
__global__ void Kpart2data(float pos[N][3], int *data) {
  int tx = threadIdx.x;
  int bx = blockIdx.x;
  int dimx = blockDim.x;

  int i = bx * dimx + tx; // indice de boucle
  float x, y, z;

  x = pos[i][0];
  y = pos[i][1];
  z = pos[i][2];
  data[i] = (int)(x / DX) *N*N +
            (int)(y / DX) * N +
            (int)(z / DX);
}
```



Unaligned access !

1	1	1	2	2	2	3	3	3	4	4	4

Example code 1
Transformation : shared memory

```
__global__ void Kpart2data(float pos[N][3], int *data) {
  int tx = threadIdx.x;
  int bx = blockIdx.x;
  int dimx = blockDim.x;
  int i = bx * dimx + tx; // indice de boucle

  extern __shared__ float spot[]; // Size define at call site : 3*dimx

  // Loading in shared memory
  int basePos = dataPos * 3;
  spot[tx] = ((float *)pos)[basePos + tx];
  spot[tx + dimx] = pos[basePos + dimx + tx];
  spot[tx + 2 * dimx] = pos[basePos + dimx * 2 + tx];

  // Sync thread to ensure shared memory is loaded
  __syncthreads();

  // Compute from shared memory
  data[i] = (int)(spot[tx*3+0] / DX) *N*N +
            (int)(spot[tx*3+1] / DX) * N +
            (int)(spot[tx*3+2] / DX);
}
```

Example code 1
Transformation : shared memory

```
__global__ void Kpart2data(int np, float pos[N][3], int *data) {
  int tx = threadIdx.x;
  int bx = blockIdx.x;
  int dimx = blockDim.x;
  int i = bx * dimx + tx; // indice de boucle

  extern __shared__ float spot[]; // Size define at call site : 3*dimx

  // Loading in shared memory
  int basePos = dataPos * 3;
  spot[tx] = ((float *)pos)[basePos + tx];
  spot[tx + dimx] = pos[basePos + dimx + tx];
  spot[tx + 2 * dimx] = pos[basePos + dimx * 2 + tx];

  // Sync thread to ensure shared memory is loaded
  __syncthreads();

  // Compute from shared memory
  data[i] = (int)(spot[tx*3+0] / DX)
            (int)(spot[tx*3+1] / DX)
            (int)(spot[tx*3+2] / DX);
}
```

Maggie	888	**x3.64**
Ridee	**803**	x1.16
Vivaldi	1119	**x0.91**

Example code 1
Allocation && communications

For each kernel launch, you'll need to :

- Allocate two arrays on GPU (~83Mo & ~250Mo)
- Copy-in *via* PCI-Express (250Mo)
- Launch the kernel
- Copy-out *via* PCI-Express (80Mo)
- Free memory on GPU

Q: what'll be the speedup taking this into account ?

	Sequential	Kernel only	With transfert	Gain potentiel
Maggie	56128	888 (**x63.2**)	**?**	**?**
Ridee	81689	803 (**x101.7**)	**?**	**?**
Vivaldi	47309	1018 (**x46.5**)	**?**	**?**

Example code 1
Allocation && communications

For each kernel launch, you'll need to :

- Allocate two arrays on GPU (~83Mo & ~250Mo)
- Copy-in *via* PCI-Express (250Mo)
- Launch the kernel
- Copy-out *via* PCI-Express (80Mo)
- Free memory on GPU

Q: what'll be the speedup taking this into account ?

	Sequential	Kernel only	With transfert	Gain potentiel
Maggie	56128	888 (**x63.2**)	39044 **(x1.4)**	**x50**
Ridee	81689	803 (**x101.7**)	19451 **(x4.2)**	**x24**
Vivaldi	47309	1018 (**x46.5**)	15288 **(x3.1)**	**x15**

Example code 2

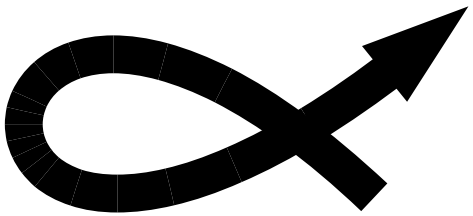
```
void forcex(float pot[N][N][N], float fx[N][N][N]) {
  int i,j,k;
  for (i = 0; i < N; i++)
    for (j = 0; j < N; j++)
      for (k = 0; k < N; k++)
        fx[i][j][k] = (pot[(i+1)&(N-1)][j][k] –
                       pot[(i-1)&(N-1)][j][k]
                      ) / (2. * DX);


}
```

Maggie	14393
Ridee	15014
Vivaldi	10089

Example code 2
Kernel « naïve »

```
__global__ void Kforcex(float pot[N][N][N], float fx[N][N][N]) {
  int i=threadIdx.x;
  int j=blockIdx.x;
  int k=blockIdx.y;
  fx[i][j][k] = (pot[(i+1)&(N-1)][j][k] –
                 pot[(i-1)&(N-1)][j][k]
                     ) / (2. * DX);
}
```



Unaligned access again !

Maggie	36875	x0.4
Ridee	**35301**	**X0.4**
Vivaldi	2537	x4

Example code 2
Kernel *improved*

```
#define STRIDE 32
__global__ void Kforcex(float pot[N][N][N], float fx[N][N][N]) {
  int i=threadIdx.x;
  int j=blockIdx.x;
  int k=blockIdx.y;
  fx[i][j][k] = (pot[(i+1)&(N-1)][j][k] –
                       pot[(i-1)&(N-1)][j][k]
                                  ) / (2. * DX);
  float a,b,c;
  int idx=((bx*STRIDE_FORCE64-1)&(ncell-1)) * ncell * ncell + by * ncell + tx;
  a = pot[idx];
  idx=(idx+ncell*ncell)&(ncell*ncell*ncell-1);
  b = pot[idx];

#pragma unroll
  for(int i=1; i<STRIDE_FORCE64+1;i++) {
    int oldIdx = idx;
    idx=(idx+ncell*ncell)&(ncell*ncell*ncell-1);
    c = pot[idx];

    fx[oldIdx] = (c-a )/(2.*DX);

    a = b;
    b = c;

  }
}
```

Loop interchange
+ kind of strip mining

	Sequential	Kernel only
Maggie	14393	**150 (x95)**
Ridee	15014	**118 (x127)**
Vivaldi	10089	**223 (x45)**