Simulaciones Astrofísicas en GPU's

CÓDIGO MAGNETOHIDRODINÁMICO FARGO3D

Pablo Benítez-Llambay
&
Frédéric Masset

WHPC 2014

IATE
CONICET
UNC

UNC
400 AÑOS

OAC

INSTITUTO DE
CIENCIAS
FÍSICAS

GPU's?

- Las GPU's están desarrolladas para el procesamiento eficiente de imágenes.
  - Operaciones entre vecinos cercanos.

- Y lo hacen extremadamente bien! (Mucho mejor que las CPU's)

Por lo tanto, si un problema físico puede ser transformado a un problema similar al procesamiento de imágenes, se puede resolver eficientemente utilizando la GPU's.

Por qué usar GPU's?

- Son extremadamente rápidas para tareas masivamente paralelas.
- Consumen muy poca energía.
- Son relativamente baratas.
- El rendimiento flop/watt es muy alto, y cada vez es mejor.
- Cada vez es más sencillo programar en ellas.
- MENDIETA

Por qué NO son tan usadas?

- Escribir “kernels” es una tarea muy tediosa.
- No siempre es trivial la transformación de un “do”/”for” hacia un kernel (de forma eficiente).
- Hay que ser cuidadosos en el manejo de la memoria.
- En general estamos interesados en hacer ciencia, no en estar programando todo el día!

Problemas típicos de la astrofísica

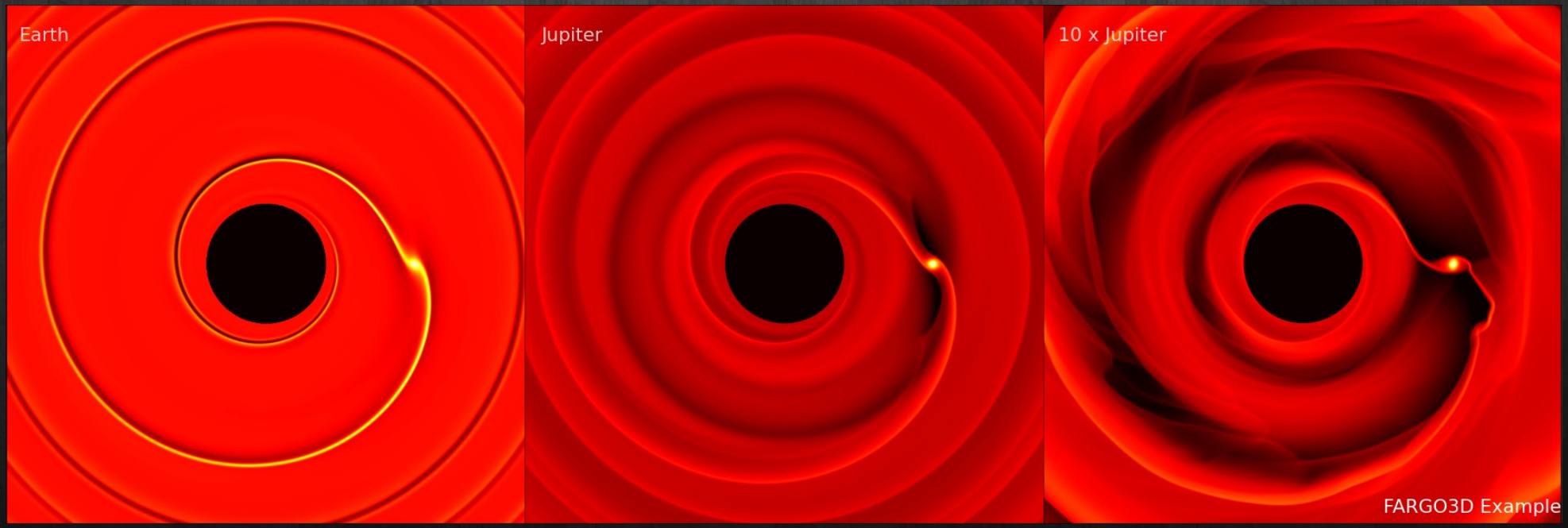
- Problemas de gravitación.*
- Problemas de radiación.
- Problemas hidrodinámicos.*
- Problemas magnetohidrodinámicos.*
- Procesamiento de datos.*

Ejemplos:

- Problemas en los que la mayoría de los cálculos sean locales.
  - Ecuaciones de la hidrodinámica. (~40x)
  - Ecuaciones de la MHD. (~40x)
  - Cálculo de potenciales. (~100x)
  - Cálculo de correlaciones.
  - Algunos problemas de optimización. (ej: Chi2)
- Problemas en los que los cálculos sean independientes. (Simulaciones planetarias, Montecarlo, etc.)

FARGO3D

- En nuestro grupo estamos estudiando planetas embebidos en discos de acreción magnetizados.



FARGO3D

- Desarrollamos un código MHD capáz de funcionar en un cluster de CPU's/ GPU's. (Paralelizado con MPI)

- Después de escribir muchos Kernels para un mismo tipo de ecuaciones, uno se da cuenta de que es una tarea muy repetitiva.

```
1  //<FLAGS>
2  //#define __GPU
3  //#define __NOPROTO
4  //<\FLAGS>
5
6  //<INCLUDES>
7  #include "fargo3d.h"
8  //<\INCLUDES>
9
10 void ComputePressureFieldIso_cpu() {
11 //<USER_DEFINED>
12   INPUT(Energy);
13   INPUT(Density);
14   OUTPUT(Pressure);
15 //<\USER_DEFINED>
16
17 //<EXTERNAL>
18   real* dens = Density->field_cpu;
19   real* cs   = Energy->field_cpu;
20   real* pres = Pressure->field_cpu;
21   int pitch  = Pitch_cpu;
22   int stride = Stride_cpu;
23   int size_x = Nx;
24   int size_y = Ny+2*NGHY;
25   int size_z = Nz+2*NGHZ;
26 //<\EXTERNAL>
27
28 //<INTERNAL>
29   int i;
30   int j;
31   int k;
32 //<\INTERNAL>
33
34 //<MAIN_LOOP>
35       (k=0; k<size_z; k++) {
36         (j=0; j<size_y; j++) {
37           (i=0; i<size_x; i++ ) {
38 //<#>
39           pres[ll] = dens[ll]*cs[ll]*cs[ll];
40 //<\#>
41       }
42     }
43   }
44 //<\MAIN_LOOP>
45 }
```

CPU

```
#define __GPU
#define __NOPROTO
#include "fargo3d.h"

            void ComputePressureFieldIso_kernel(real* dens,
                                                real* cs,
                                                real* pres,
                                                int pitch,
                                                int stride,
                                                int size_x,
                                                int size_y,
                                                int size_z) {

  int i;
  int j;
  int k;
  int ll;

i = threadIdx.x + blockIdx.x * blockDim.x;
j = threadIdx.y + blockIdx.y * blockDim.y;
k = threadIdx.z + blockIdx.z * blockDim.z;

   (k>=0 && k<size_z) {
     (j>=0 && j<size_y) {
       (i<size_x) {
       ll = l;
       pres[ll] = dens[ll]*cs[ll]*cs[ll];
     }
   }
 }
}

extern "C" void ComputePressureFieldIso_gpu() {

  INPUT(Energy);
  INPUT(Density);
  OUTPUT(Pressure);

dim3 block (BLOCK_X, BLOCK_Y, BLOCK_Z);
dim3 grid ((Nx+block.x-1)/block.x,
((Ny+2*NGHY)+block.y-1)/block.y,
((Nz+2*NGHZ)+block.z-1)/block.z);

#ifdef BIGMEM
#define xmin_d &Xmin_d
#define ymin_d &Ymin_d
#define zmin_d &Zmin_d
#define Sxj_d &Sxj_d
#define Syj_d &Syj_d
#define Szj_d &Szj_d
#define Sxk_d &Sxk_d
#define Syk_d &Syk_d
#define Szk_d &Szk_d
#define InvVj_d &InvVj_d
#endif

 cudaFuncSetCacheConfig(ComputePressureFieldIso_kernel, cudaFuncCachePreferL1 );
 ComputePressureFieldIso_kernel<<<grid,block>>>(Density->field_gpu,
                                                Energy->field_gpu,
                                                Pressure->field_gpu,
                                                Pitch_gpu,
                                                Stride_gpu,
                                                Nx,
                                                Ny+2*NGHY,
                                                Nz+

 check_errors("ComputePressureFieldIso_kernel");
}
```

GPU

Funciones de un código MHD típico

- Headers.
- Tipo de función y nombre.
- Argumentos.
- Variables globales (o argumentos de funciones, vienen desde afuera).
- Variables locales (internas a la función).
- Un loop sobre la malla.
- Alguno que otro detalle menor...

Hay que resolver:

- Cómo generar un header compatible con CUDA. (Librerías adicionales)
- Desarrollar el kernel.
- Desarrollar un lanzador (wrapper para el lanzamiento del kernel) compatible con la estructura de C de FARGO3D.
- Generar las comunicaciones entre CPU/GPU de forma consistente.
- Separar el loop principal en la cantidad de threads necesarios.
- Desarrollar un método para pasar las variables globales hacia adentro del kernel.
- Desarrollar un método para pasar estructuras mas generales hacia el kernel.
- Un método para cambiar fácilmente entre rutinas CPU/GPU.

C2CUDA PARSER

```
//<FLAGS>
//#define __GPU
//#define __NOPROTO
//<\FLAGS>

//<INCLUDES>
#include "fargo3d.h"
//<\INCLUDES>

void ComputePressureFieldIso_cpu() {
//<USER_DEFINED>
  INPUT(Energy);
  INPUT(Density);
  OUTPUT(Pressure);
//<\USER_DEFINED>

//<EXTERNAL>
  real* dens = Density->field_cpu;
  real* cs   = Energy->field_cpu;
  real* pres = Pressure->field_cpu;
  int pitch  = Pitch_cpu;
  int stride = Stride_cpu;
  int size_x = Nx;
  int size_y = Ny+2*NGHY;
  int size_z = Nz+2*NGHZ;
//<\EXTERNAL>

//<INTERNAL>
  int i;
  int j;
  int k;
//<\INTERNAL>

//<MAIN_LOOP>
      (k=0; k<size_z; k++) {
        (j=0; j<size_y; j++) {
          (i=0; i<size_x; i++ ) {
//<#>
        pres[ll] = dens[ll]*cs[ll]*cs[ll];
//<\#>
     }
    }
  }
//<\MAIN_LOOP>
}
```

c2cuda.py

```
#define __GPU
#define __NOPROTO
#include "fargo3d.h"

          void ComputePressureFieldIso_kernel(real* dens,
                                              real* cs,
                                              real* pres,
                                              int pitch,
                                              int stride,
                                              int size_x,
                                              int size_y,
                                              int size_z) {

  int i;
  int j;
  int k;
  int ll;

i = threadIdx.x + blockIdx.x * blockDim.x;
j = threadIdx.y + blockIdx.y * blockDim.y;
k = threadIdx.z + blockIdx.z * blockDim.z;

   (k>=0 && k<size_z) {
     (j>=0 && j<size_y) {
       (i<size_x) {
       ll = l;
       pres[ll] = dens[ll]*cs[ll]*cs[ll];
     }
   }
 }
}

extern "C" void ComputePressureFieldIso_gpu() {

  INPUT(Energy);
  INPUT(Density);
  OUTPUT(Pressure);

dim3 block (BLOCK_X, BLOCK_Y, BLOCK_Z);
dim3 grid ((Nx+block.x-1)/block.x,
((Ny+2*NGHY)+block.y-1)/block.y,
((Nz+2*NGHZ)+block.z-1)/block.z);

#ifdef BIGMEM
#define xmin_d &Xmin_d
#define ymin_d &Ymin_d
#define zmin_d &Zmin_d
#define Sxj_d &Sxj_d
#define Syj_d &Syj_d
#define Szj_d &Szj_d
#define Sxk_d &Sxk_d
#define Syk_d &Syk_d
#define Szk_d &Szk_d
#define InvVj_d &InvVj_d
#endif

 cudaFuncSetCacheConfig(ComputePressureFieldIso_kernel, cudaFuncCachePreferL1 );
 ComputePressureFieldIso_kernel<<<grid,block>>>(Density->field_gpu,
                                                Energy->field_gpu,
                                                Pressure->field_gpu,
                                                Pitch_gpu,
                                                Stride_gpu,
                                                Nx,
                                                Ny+2*NGHY,
                                                Nz+2*NGHZ);
 check_errors("ComputePressureFieldIso_kernel");
}
```

C2CUDA PARSER

- Cualquier usuario puede programar FARGO3D en GPU, incluso sin saber nada de ellas.

- Gracias a la posibilidad de hacer un switch entre CPU/GPU, hacer un benchmark por función es muy fácil.

Puntos a destacar

- Hace más de un año que no escribimos kernels.
- Posibilidad de elegir en tiempo real funciones CPU/GPU. (Punteros a funciones gemelas en CPU/GPU).
- Las comunicaciones se hacen de forma automática.
- Si funciona en la CPU, funciona en la GPU!
- Speedup muy bueno (**40x!!!!!**), mientras que para el mismo problema realizado con OpenACC tenemos reportes mucho mas pobres (10x).
- Optimización de tamaño de bloques de forma automática para cada función. (20%)
- Posibilidad de utilizar memoria de constantes o global para arreglos pequeños.
- Utilización de la tecnología GPU-direct.

FARGO3D en la web

GRACIAS