2nd Annual Meeting of EUROfusion HPC ACHs

BIT1, XTOR-K and ERO2.0

Xavier Sáez

Eduardo Cabrera

Augusto Maidana

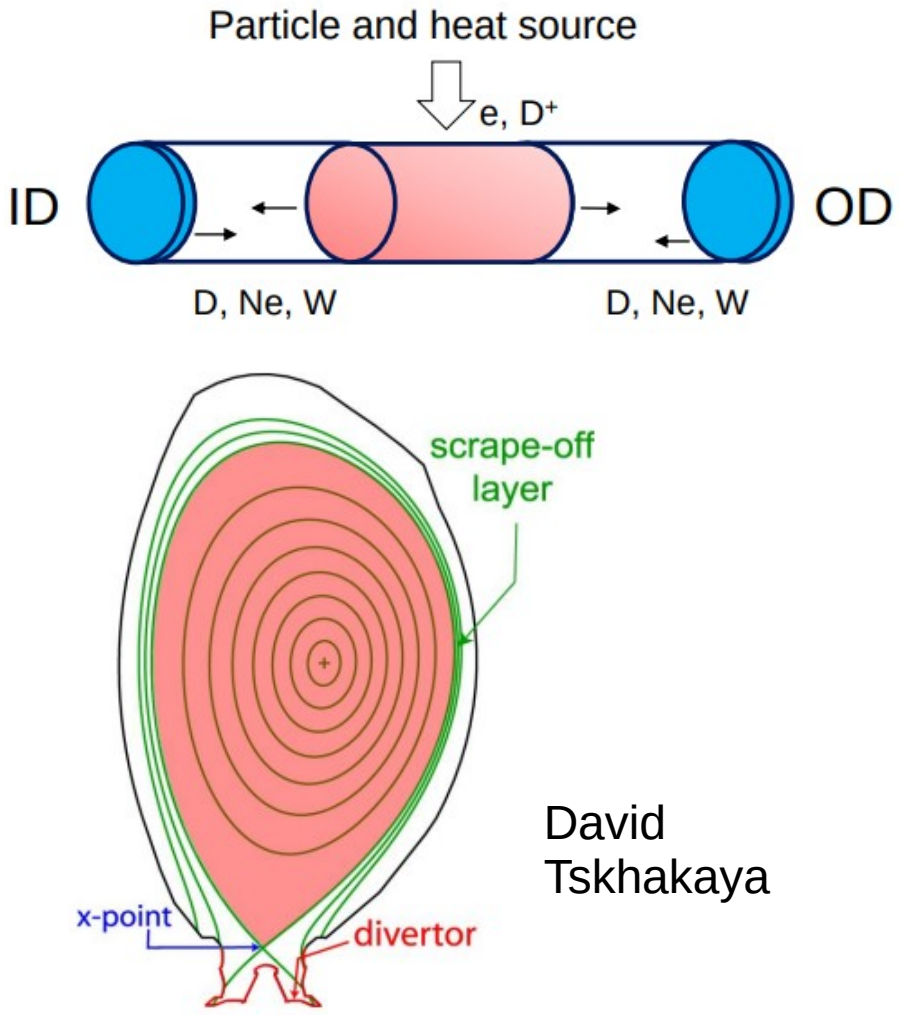
ACH@CIEMAT-BSC

This work has been carried out within the framework of the EUROfusion Consortium, funded by the European Union via the Euratom Research and Training Programme (Grant Agreement No 101052200 — EUROfusion). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Commission. Neither the European Union nor the European Commission can be held responsible for them.

BIT1

Xavier Sáez

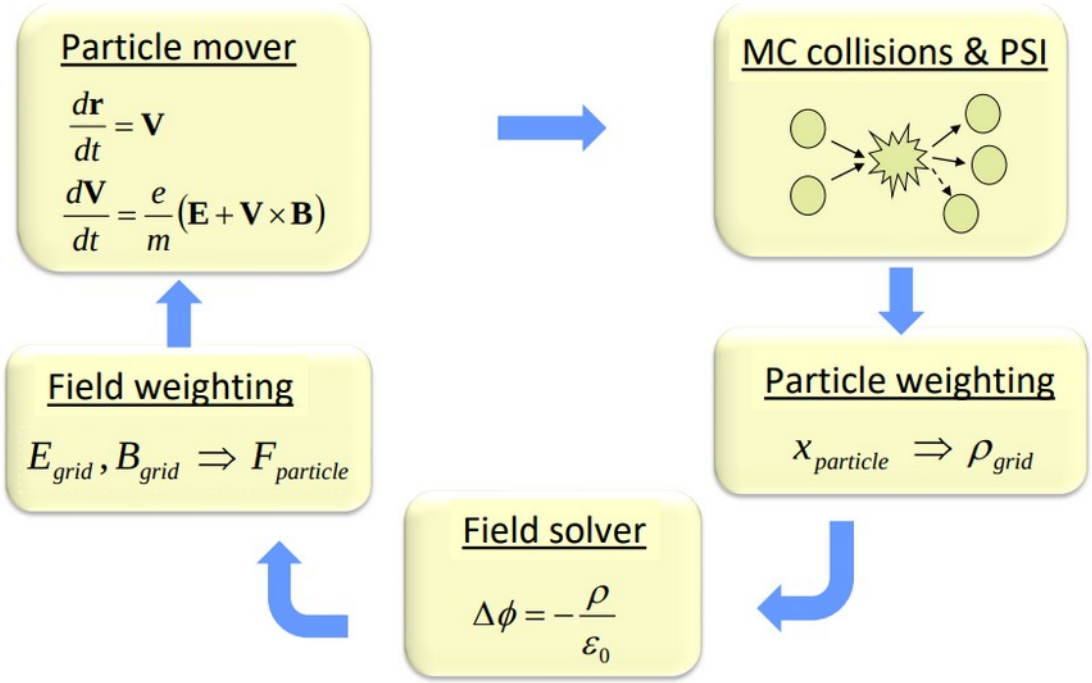
BIT1: Introduction



David Tskhakaya

- BIT1 is an electrostatic 1D3V PIC + 2D3V direct Monte Carlo code for plasma simulations.
- **Input:** Particle, heat sources, geometry and divertor material
- **Simulated particle species:** El, D (H, T), Li, He, Be, O, …
- Highly accurate → $5x10^6$ cells →capability to simulate whole SOL
- **Language:** C
- **Libraries**: No external libs
- **Parallelization**: MPI using domain decomposition

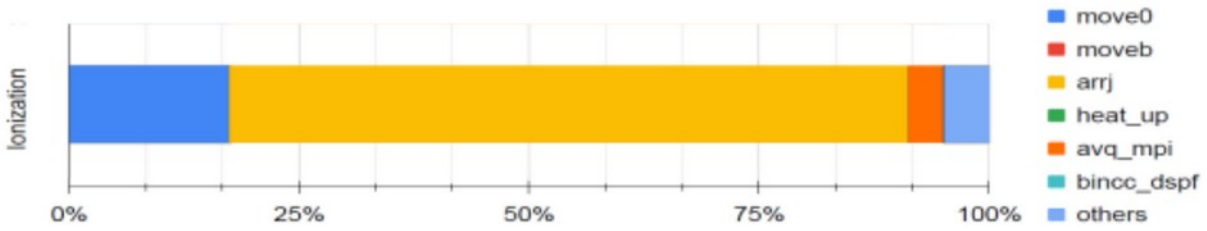
BIT1: Algorithm



- BIT1 is a **Particle-in-cell (PIC) code**

BIT1: Project

- **Project:** develop a GPU version of the BIT1
- **Work plan:**
  - OpenACC
  - Target routines:
    - **Move0:** particle pusher.
      Leap-Frog scheme, adjusts solver to the given magnetic field configuration
      No core communication is required
    - **Arrj:** arranjing particles into proper cells and cores
      No core communication is required



BIT1: Difficulties

- **Error:** Segmentation fault

```
Can't find input file /gpfs/projects/bsc21/bsc021331/BIT1/bit1-feature-GPU-OpenACC/tests/../BIT1_c8/at@N
Can't find input file /gpfs/projects/bsc21/bsc021331/BIT1/bit1-feature-GPU-OpenACC/tests/../BIT1_c8/at@@eH
```

bit1.h

```
#define MAXNM           80        /* maximum number of letters in names */
```

- **Complex data structure**

```
x  = (float ***)malloc(nsp*sizeof(float **));

for(i=0; i<nsp; i++)
{
  x[i]  = (float **)malloc((ng+1)*sizeof(float *));

  for(k=0; k<=ng; k++)
  {
     x[i][k]  = (float *)malloc(maxL[i]*sizeof(float));

     for (l=0; l<maxL[i]; l++)
        x[i][k][l]  = 0.0;
  }
}
```

- **Global variables**

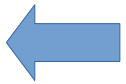
bit1.h

```
float xnc, nc2p, length, area, rhoback, backj, dde, epsilon, bmag,
      extc, w0, dcbias, acbias,
      exti, sigma, oldsigma, dx, dt, vxscale, vscale, dttx, se,
      aa, lsq_n, n_av, cc_coef, ht, ht1, cs, sn, dn, extr,
      df, ramp, theta0, jp, b0, risetime, LL_01, Ls_mpi, Ls1_mpi,
      Src, L_s, nc2, nc3, nc05, dshort, L_s1, nc21, nc31, nc051,
      ...
      ***vx, ***vy, ***vz, ***x,
      ...
      ***s_disem, ***s_disa,
      ***s_idem, ***s_ida,
      ****s_chxe, ****s_exte,
      *****s_ione, *****sd_rre, *****sd_rde, *****sd_dise,
      *****sd_ide, *****sd_tbre;
```

BIT1: Difficulties

- **Function pointers**

```
while (tstep < last_step)
{
  t += dt;
  tstep++;
  ...
  (*move_p)();   /* particle mover */
  ...
}
```



```
void parmover()
{

  /** Deciding which mover to use **/

  if (bmag>0.)
  {
    if (field)  move_p = moveb;
    else        move_p = moveb0;
  }
  else if (field)  move_p = move1;
  else move_p = move0;

} /* End of PARMOVER */
```

- **Demanding large amount of memory**

  slurmstepd: error: Detected 1 oom_kill event

- **Requires a large #steps to generate output file results**

- **OpenACC errors:**

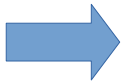
  Complex loop carried dependence of nstep->,vx->,vx->->,np->->,vx->->->,x->,x->->->,x->->,np-> prevents parallelization

  Loop not vectorized/parallelized: not countable

  Inner collapsed loop bounds are not invariant in outer loop.

BIT1: Implementation

```
void move0()
{
  int isp, i, j;

  for (isp=0; isp<nsp; isp++)
  {
   if (chsp[isp])
   {
    if(dinj[isp])    /* some particles are moving after nstep t.s.*/
     for (j=0; j< nc; j++)
      for (i=np[isp][j]-1; i>=0; i--)
        x[isp][j][i] += nstep[isp]*vx[isp][j][i];

   }  /* end of if (chsp[isp]) loop */

    else
     nmove(isp);                      /* mover for neutrals */
  }
}  /* END of move0() */

/*************************************************************/
/********* Particle mover for neutrals.   ********************/
/*************************************************************/

void nmove(int isp)
{
  int i, j;

    if(dinj[isp])    /* some particles are moving after nstep t.s.*/
    {
     if (sn2d[isp])        /* 2D case */
     {
      for (j=0; j< nc; j++)
       for (i=np[isp][j]-1; i>=0; i--)
       {
        x[isp][j][i] += nstep[isp]*vx[isp][j][i];
        yp[isp][j][i] += nstep[isp]*vy[isp][j][i];
       }
     }
     else
     {
      for (j=0; j< nc; j++)
       for (i=np[isp][j]-1; i>=0; i--)
        x[isp][j][i] += nstep[isp]*vx[isp][j][i];
     }
    }
}  /* END of nmove */
```



```
void move0()
{
  int isp, i, j;

 #pragma acc enter data copyin(chsp[:nsp], dinj[:nsp], np[:nsp][:nc], nstep[:nsp])

  for (isp=0; isp<nsp; isp++)
  {
   #pragma acc enter data copyin(x[isp:1][:nc][:maxL[isp]], vx[isp:1][:nc][:maxL[isp]],
                                 yp[isp:1][:nc][:maxL[isp]], vy[isp:1][:nc][:maxL[isp]])
   if (chsp[isp])
   {
    if(dinj[isp])
    {    /* some particles are moving after nstep t.s.*/
      printf("openacc dim x %d %d  np %d %d \n",nsp,ng+1,nsp,nc);

      #pragma acc parallel loop private (i,j)
      for (j=0; j< nc; j++)
      {
        #pragma acc loop
        for (i = 0; i < np[isp][j]; i++)
        {
          x[isp][j][i] += nstep[isp]*vx[isp][j][i];
        }
      }
    }
   }  /* end of if (chsp[isp]) loop */

    else {
    if(dinj[isp])    /* some particles are moving after nstep t.s.*/
    {
      ...
    }

   #pragma acc exit data copyout(x[isp:1][:nc][:maxL[isp]], yp[isp:1][:nc][:maxL[isp]])
  }
 }

}  /* END of move0() */
```

BIT1: Work in progress

- **Results:**

move0:	No acc:	43.58 s	Total:	No acc:	509.21s
	Acc:	2.16 s		Acc:	582.03s
Reduction=95.6%		Speedup ⇒ 20x	Increment=14.3%		Speedup ⇒ 0.87x

- **Next Steps:**
  - Improve data movement CPU ↔ GPU
  - Introduce OpenACC on arrj routine

XTOR-K

Eduardo Cabrera

XTOR-K: Background

- XTOR-K is a HPC totally implicit hybrid fluid-kinetic model with a Particle in Cell (PIC) Boris-Buneman particle pusher applied to one or more populations of kinetic ions making possible to simulate the couplings between the Alfvénique spectrum and ionic kinetic effects.

- It couples in a self-consistent manner the time advance of a complete set of bi-fluid equations in full tokamak geometry (thermalized plasma electron and ions background) with a 6D PIC method accurately integrating the trajectories of selected ion populations, making accessible new families of instabilities which are not allowed in a simplified framework such as magnetohydrodynamics.

- It is implemented in Fortran2008, parallelised with MPI and OpenMP already.

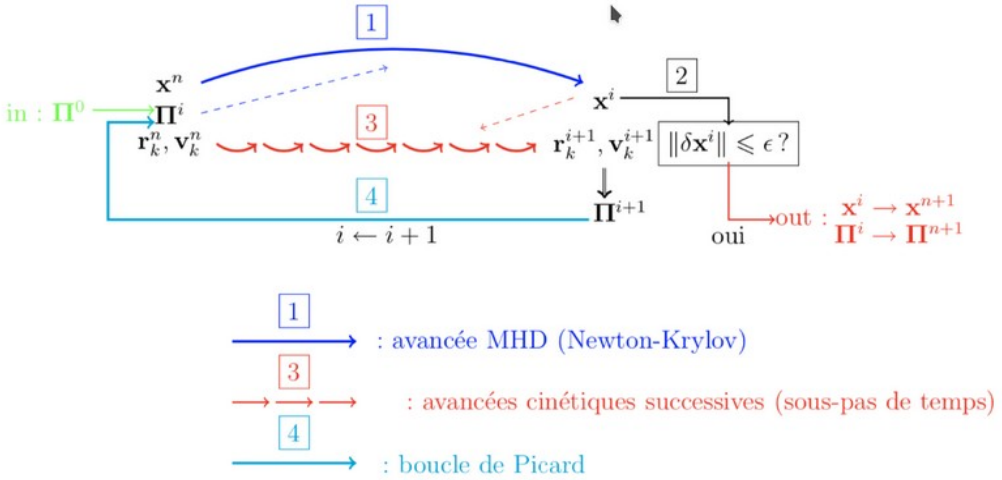
Hinrich Lütjens

XTOR-K: Current status and objective

- The kinetic part of the code is dominant in terms of computer time consumption.

  It uses a PIC time advance scheme.

  - **Interpolate_eb_fields** has been identified as the most consuming-time subroutine.



- Porting such kinetic parts onto GPU's is twofold:
  - To increase significantly the number of markers used in the PIC advance.
  - To reduce the numerical noise inherent to this method.

XTOR-K: Work done

- All kinetic related subroutines (6) were ported to GPU using OpenACC.

  The fluid part is still working properly. (CPU)

  Interpolate_eb_fields subroutine could not be *totally* ported since it has a **return** statement inside it.

XTOR-K: Work in progress

- XTOR-K is being refactored
    - To get rid of such return statement
    - Reordering some do-loops

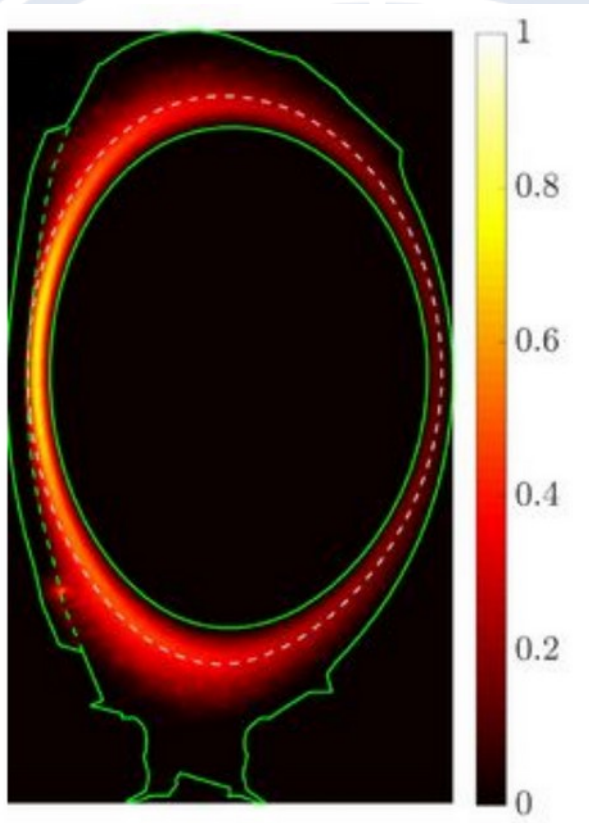
ERO2.0

Augusto Maidana

ERO2.0: Introduction

What is ERO2? Why Migrate?

- **ERO2.0** simulates plasma-wall interaction and global material migration in fusion devices.
- **The goal**: Accelerate computations by transitioning to GPUs, leveraging new accelerator technologies.
- **Benefits**: Faster simulations, better scalability, and alignment with modern computational trends.



Juri Romazanov

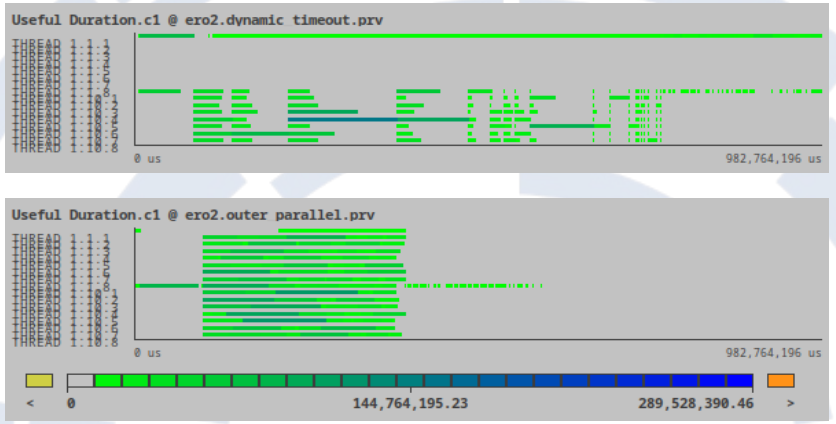
ERO2.0: Migration overview

Transitioning from CPU to GPU

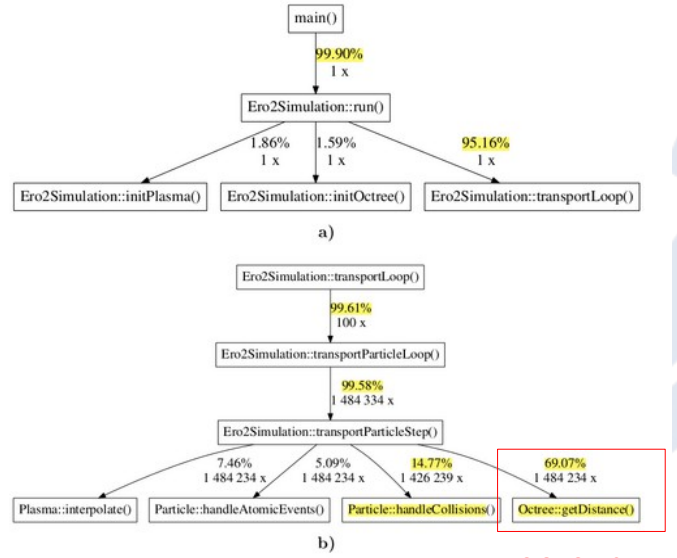
- Moved from CPU-based parallelism (**OpenMP**, **MPI**) to GPU-based design (**OpenACC**, **CUDA**).
- **Objective**: Enable the architecture to support massive parallelism for GPU environments.
- **Strategy**: Focus on adapting critical sections while preserving algorithmic integrity.

The load imbalance caused by the variability in the particle simulation times.

We develops several Proof of concept to fix the imbalance.
The better was the third one




```
g3d/
|
├── PolygonTypes.h
├── TrianglePolygon.h
├── TrianglePolygon.cpp
├── QuadPolygon.h
├── QuadPolygon.cpp
├── PolyMesh.h
├── PolyMesh.cpp
├── PolyMeshSet.h
├── PolyMeshSet.cpp
├── Ray.h
└── Octree.cu
```

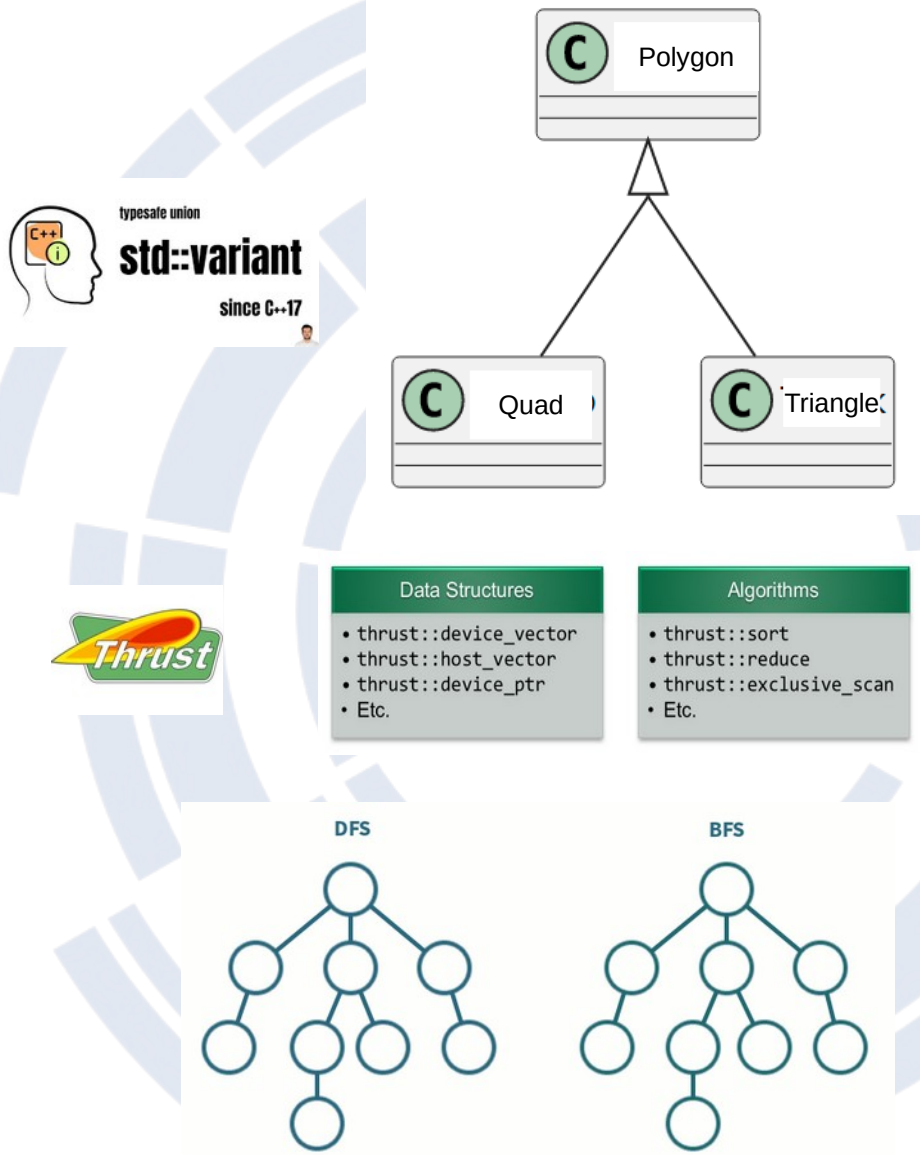


69.07%

ERO2.0: Challenges and solutions

Overcoming Migration Hurdles

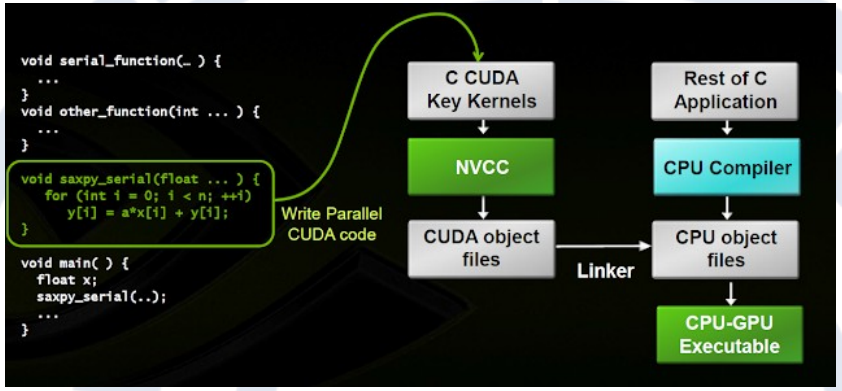
- **Virtual Functions**:
  - **Problem**: Inefficient and unsupported on GPUs.
  - **Solution**: Static polymorphism using **std::variant** and **std::visit**.
- **STL Containers**:
  - **Problem**: GPU incompatibility with **std::vector**.
  - ***Solution***: Replaced with **thrust::device_vector** or custom GPU-friendly wrappers.
- **Recursion & Exceptions**:
  - **Problem**: Recursion and **throw** are problematic on GPUs.
  - **Solution**: Converted **recursion** to **iteration** and replaced exceptions with **return** codes.



ERO2.0: Parallelization Strategy

Maximizing Parallel Performance

- **Dual approach**: OpenACC for loop parallelization and CUDA for kernel optimization.
- Migration of **G3D** classes to support GPU operations.
- Optimized **Octree** functions for efficient polygon distance computations.
- **Goal**: Achieve high utilization of GPU resources.




```
cudaMalloc(&d_proj, sizeof(Vector));
cudaMemcpy(d_proj, &proj, sizeof(Vector), cudaMemcpyHostToDevice);

double* d_t;
cudaMalloc(&d_t, sizeof(double));
cudaMemcpy(d_t, &t, sizeof(double), cudaMemcpyHostToDevice);

const PolygonVariant** d_poly;
cudaMalloc(&d_poly, sizeof(PolygonVariant*));

// Definir el tamaño del bloqqeey grid para CUDA
int blockSize = 256;
int gridSize = (numPolygons + blockSize - 1) / blockSize;

// Lanzar el kernel en la GPU
intersectKernel<<<gridSize, blockSize>>>(d_polygons, *d_segment, d_proj, d_t, d_poly, numPolygons);

// Check for errors after kernel launch
CUDA_CHECK_ERROR(cudaPeekAtLastError());
CUDA_CHECK_ERROR(cudaDeviceSynchronize());

// Copiar el resultado de vuelta desde la GPU al host
cudaMemcpy(&proj, d_proj, sizeof(Vector), cudaMemcpyDeviceToHost);
cudaMemcpy(&t, d_t, sizeof(double), cudaMemcpyDeviceToHost);
cudaMemcpy(&poly, d_poly, sizeof(PolygonVariant*), cudaMemcpyDeviceToHost);

// Liberar la memoria en la GPU
cudaFree(d_polygons);
```

ERO2.0: Current work status – Static Polymorphism

Replacing Dynamic Polymorphism

- **Manual adaptation**: Replaced calls to *Polygon* with *PolygonType* using **std::variant** and **std::visit**.
- **Files updated**: *Ero2Simulation.cpp*, *Particle.cpp*, *Pfc.cpp*, *PsiManager.cpp*, *TelescopeCam.cpp*, *WideViewCam.cpp*.
- **Result**: Eliminated dynamic polymorphism in favor of static polymorphism.

```
1763 void Ero2Simulation::psiAddErodedParticle (
1764     size_t id,
1765     size_t face_id,
1766     const Pfc& pfc,
1767     double atomsPerSecond,
1768     SpeciesType targetType,
1769     SpeciesType projectileType,
1770     const ErosionProcess& erosionProcess,
1771     std::vector<Particle>& buf
1772 )
1773 {
1774     // Obtener el poligooo uuaado std::variant
1775     const auto& face = pfc.polyMesh->polygons[face_id];
1776
1777     // Usar std::visit para obtener el normal del poligono
1778     g3d::Vector normal = std::visit([](const auto& poly) -> g3d::Vector {
1779         if constexpr (std::is_same_v<std::decay_t<decltype(poly)>, g3d::Triangle>) {
1780             return poly.getNormal();
1781         } else {
1782             // Para QuadPolygon, si no tiene getNormal, calcular el normal aquí o manejarlo de manera distinta.
1783             return getNormal(poly);  // Implementar esta función o usar otro método.
1784         }
1785     }, face);
1786
1787     // Usar std::visit para obtener el punto aleatorio en el poligono
1788     g3d::Vector position;
1789     switch (config.particles.initialCellPositionMethod)
1790     {
1791         case 0: // Usar un punto aleatorio en la superficie del poligono
1792             position = std::visit([&](const auto& poly) {
1793                 return poly.getRandomPoint(generator);
1794             }, face);
1795             break;
1796
1797         case 1: // Usar el centroide del poligono
1798             position = std::visit([](const auto& poly) {
1799                 return poly.getCentroid();
1800             }, face);
1801             break;
1802
1803         default:
1804             throw std::runtime_error("Unknown initialCellPositionModel");
1805     }
1806     //Print position (face->getCentroid()); // only for testing: start all particles from the same point in cent
```

std::visit

```
// Using std::variant to store different polygon types
 using PolygonVariant = std::variant<TrianglePolygon, QuadPolygon>;

  // Function that applies the correct method based on the polygon type
  using PolygonVariant = std::variant<TrianglePolygon, QuadPolygon>;
  double calculateDistanceSq(const PolygonVariant& polygon) {
      return std::visit([ ](const auto& poly) {
          return poly.getDistanceSq();
      }, polygon);
  }
```

ERO2.0: Current work status – GPU compatibility

Adapting to GPU Constraints

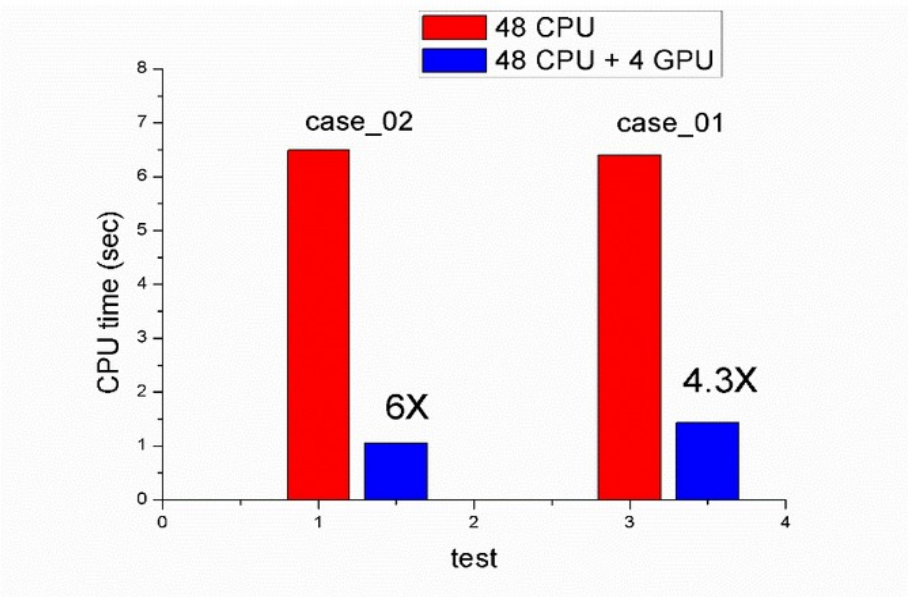
- **STL Replacement**:
  - Replaced all STL containers like **std::vector** in GPU-parallelized areas with **thrust::device_vector**.
  - Developed custom GPU-compatible wrappers where necessary.
- **Recursion and Exceptions**:
  - Converted recursive methods to iterative implementations (e.g., BFS).
  - Removed exceptions, using return codes for error handling on GPU.

```cpp
#ifndef MYVECTOR_H
#define MYVECTOR_H

#include <initializer_list>
//#include <stdexcept>     // Para std::out_of_range
#include <algorithm>     // Para std::copy
#include <vector>        // Para el constructor que acepta std::vector
#include <iterator>      // Para std::distance


namespace g3d {

template <typename T>
class MyVector {
public:
    // Constructor por defecto
    MyVector() : data(nullptr), sz(0), cap(0) {}

    // Constructor con lista de inicialización
    MyVector(std::initializer_list<T> init) : sz(init.size()), cap(init.size()) {
        data = new T[cap];
        std::copy(init.begin(), init.end(), data);
    }

    // Constructor que permite construir un MyVector a partir de un std::vector
    MyVector(const std::vector<T>& other) : sz(other.size()), cap(other.size()) {
        data = new T[cap];
        std::copy(other.begin(), other.end(), data);
    }

    // Constructor que permite construir un MyVector a partir de dos iteradores
    template <typename InputIt>
    MyVector(InputIt first, InputIt last) {
        sz = std::distance(first, last);
        cap = sz;
        data = new T[cap];
        std::copy(first, last, data);
    }

    // Constructor que permite definir tamaño y valor de inicialización opcionalon
    MyVector(size_t count, const T& value = T()) : sz(count), cap(count) {
        data = new T[cap];
        std::fill(data, data + sz, value);
    }
```

wrapper classes

ERO2.0: Next Steps

- **Finalize code adjustments**: Debug, refine, and compile remaining tasks.
- **Test and validate**: Run comprehensive tests, comparing GPU and CPU results for accuracy and consistency.
- **Optimize for performance**: Fine-tune the code to maximize computational efficiency and resource usage.
- **Deliverable**: A functional, optimized GPU version ready for deployment.



Thanks for your attention !