VOG project: Vlasov on GPU

Luminy, August 2012

Organization

- From Vlasov to GPU
  - Development and comparison of Vlasov-Poisson solvers
  - Implementation in GPU
- From GPU to Vlasov (Philippe Helluy)
  - Discontinuous Galerkin GPU code for hyperbolic systems
  - Velocity spectral approximation of the Vlasov-Poisson system

Vlasov-Poisson (1D $\times$ 1D)

Vlasov-Poisson system

$$\partial_t f(t,x,v) + v\partial_x f(t,x,v) + E(t,x)\partial_v f(t,x,v) = 0,$$

where the field E is solution of the Poisson equation

$$\partial_x E(t,x) = \int_{\mathbb{R}} f(t,x,v)dv - 1$$

with zero mean condition ($\int_0^L E(t,x)dx = 0$)

$\Rightarrow$ Simplified model ; first plasmas test cases
$\Rightarrow$ Smooth solution but development of small scales

Plan

1. FFT type algorithm
2. Implementation in GPU
3. Results

Strang splitting

Transport in x over $\Delta t/2$

$$\mathcal{T}_x : f \to f(x - v\Delta t/2, v)$$

Update of electric field E

Transport in v over Δt

$$\mathcal{T}_v : f \to f(x, v - E(x)\Delta t)$$

Transport in x over $\Delta t/2$

$$\mathcal{T}_x : f \to f(x - v\Delta t/2, v)$$

1. Constant advection along x direction for $f(\cdot, v_j)$
2. Constant advection along v direction for $f(x_i, \cdot)$

The constant advection equation

Continuous

$$u(t_{n+1}, z_k) = u(t_n, z_k - c\Delta t)$$

Numerics

$$u^{n+1} = Au^n, \; u_i^n \approx u(t_n, z_k)$$

with circulant matrix

$$A = \begin{pmatrix} a_0 & a_1 & \dots & \dots & a_{N-1} \\ a_{N-1} & a_0 & a_1 & \dots & a_{N-2} \\ \ddots & \ddots & \ddots & \ddots & \ddots \\ \ddots & \ddots & \ddots & \ddots & \ddots \\ a_1 & \dots & \dots & a_{N-1} & a_0 \end{pmatrix}$$

We use periodic boundary conditions

Examples

We define the CFL number

$$\beta = \frac{c\Delta t}{\Delta x}$$

1. Upwind ($0 \leq \beta \leq 1$)

$$a_0 = (1-\beta),\ a_{N-1} = \beta,\ a_1 = \cdots = a_{N-2} = 0.$$

2. LAG1 : upwind with possible shift
3. LAG-(2d+1) : $2d+2$ non vanishing terms

FFT implementation

Circulant matrices are diagonalizable in Fourier basis. So we can write

$$A = UDU^{\star}$$

where U is unitary and D diagonal :

$$\begin{aligned} U &= N^{-1/2}[e^{-2i\pi mk/N}, \quad m,k = 0 \dots N-1] \\ D &= diag\left(\sum_{k=0}^{N-1} a_k e^{-2i\pi mk/N}, \quad m = 0,...,N-1\right). \end{aligned}$$

In order to compute Au^n, we follow the algorithm :

1. compute $U^{\star}u^n$ by calculating $FFT^{-1}(u^n)$
2. compute D by calculating $FFT(a)$
3. compute Au^n by calculating $FFT(DU^{\star}u^n)$

Cuda GPU implementation

Kernels on GPU by using existing NVIDIA routines

- FFT by using cufft library
- transposition ($N = N_x = N_v$ required)

  $\Rightarrow$ different possible algorithms are provided
- compute charge density $\rho = \int f(t, x, v)dv$

  $\Rightarrow$ adaption of `ScalarProd` routine

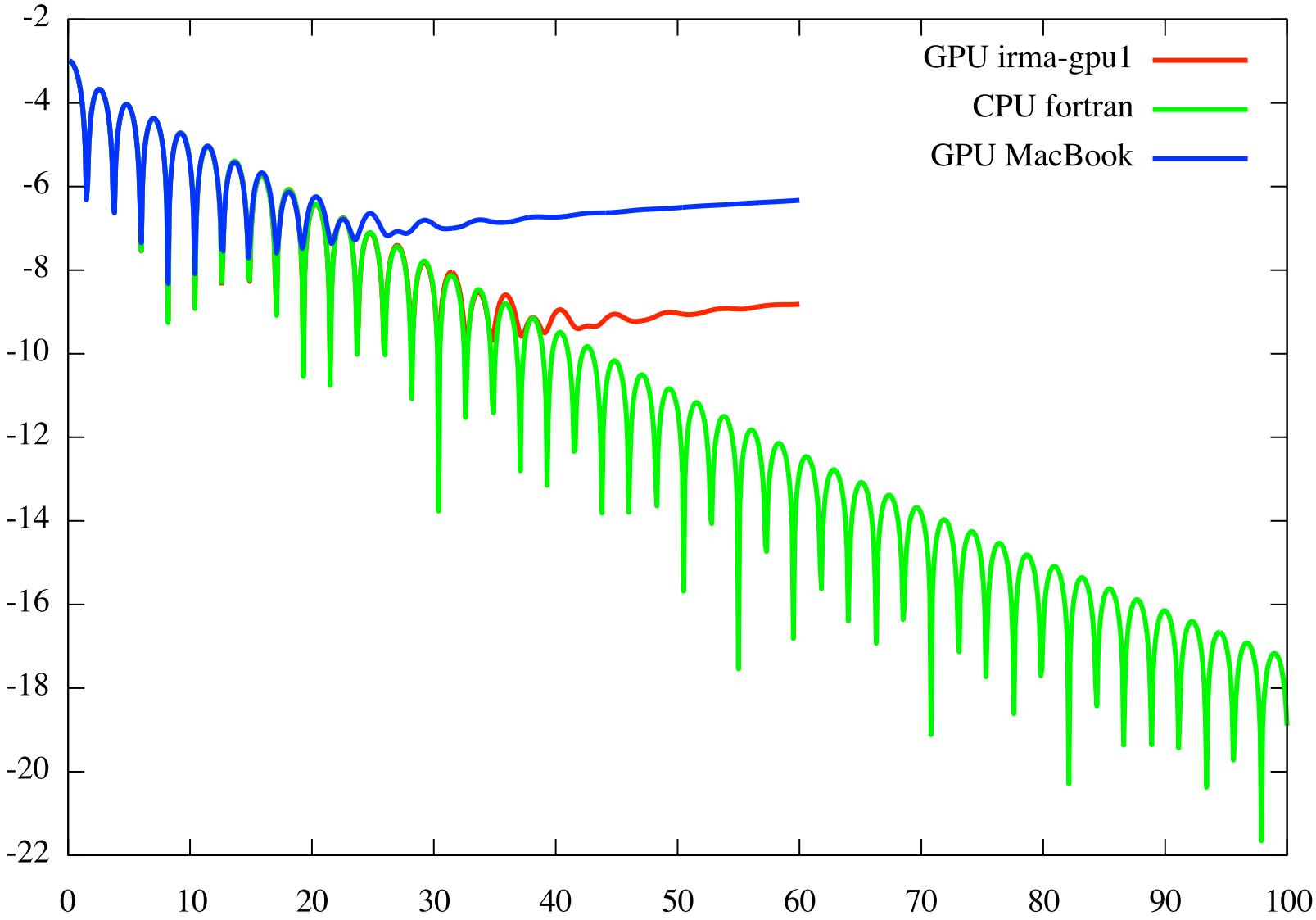
Kernel on GPU for computing coefficients of A matrix

- analytical formula is used for each coefficient a_i
- complexity switched from $O(Nd)$ to $O(Nd^2)$ operations

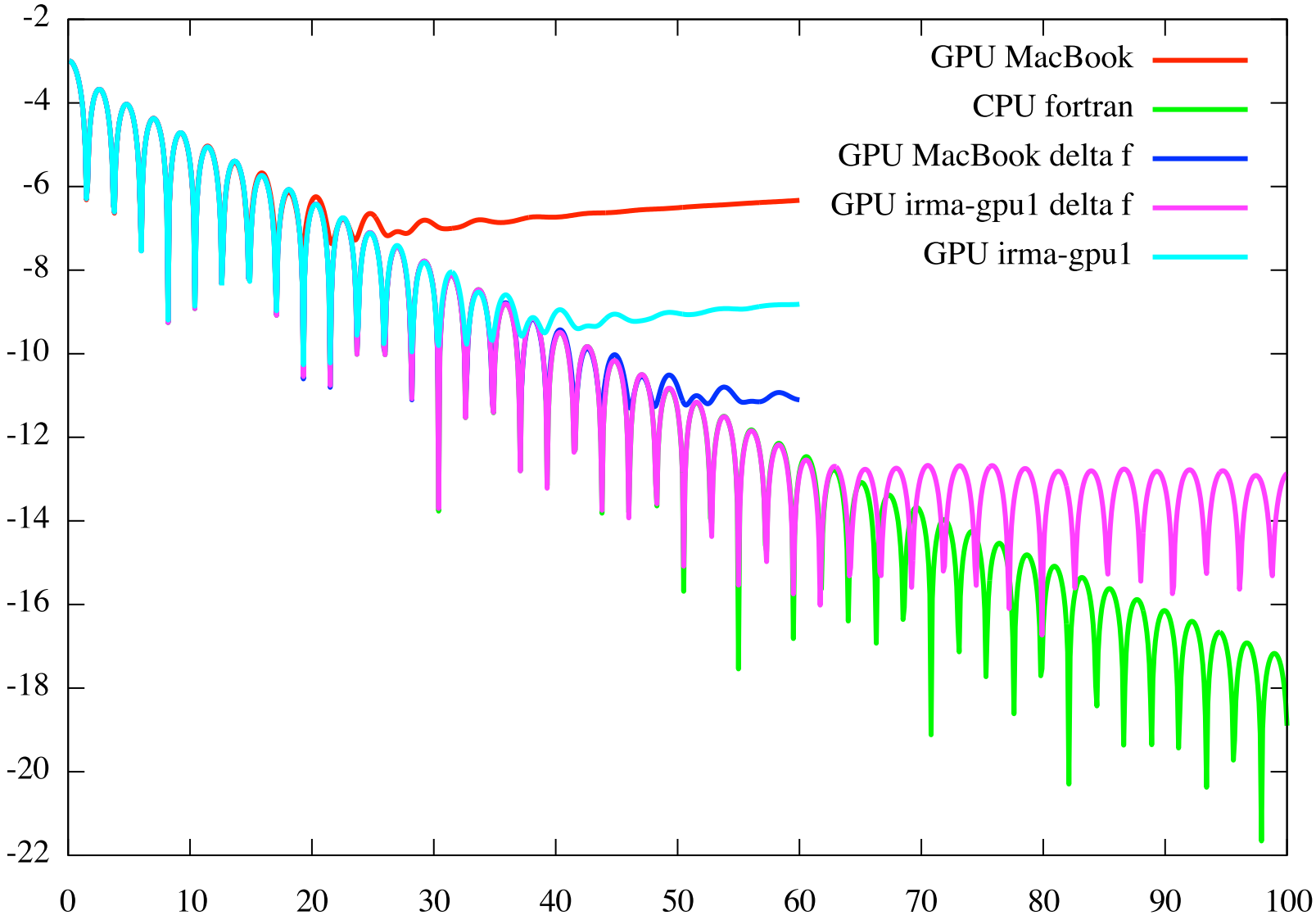
Other aspects

- Initial condition computed on CPU and transferred to GPU
- At each time step
  1. Advection in x with fft on GPU
  2. Transposition
  3. Computation of ρ on GPU
  4. Transfer of ρ to CPU
  5. Computation of electric field E on CPU
  6. Transfer of E on GPU
  7. Advection in v with fft on GPU
  8. Transposition

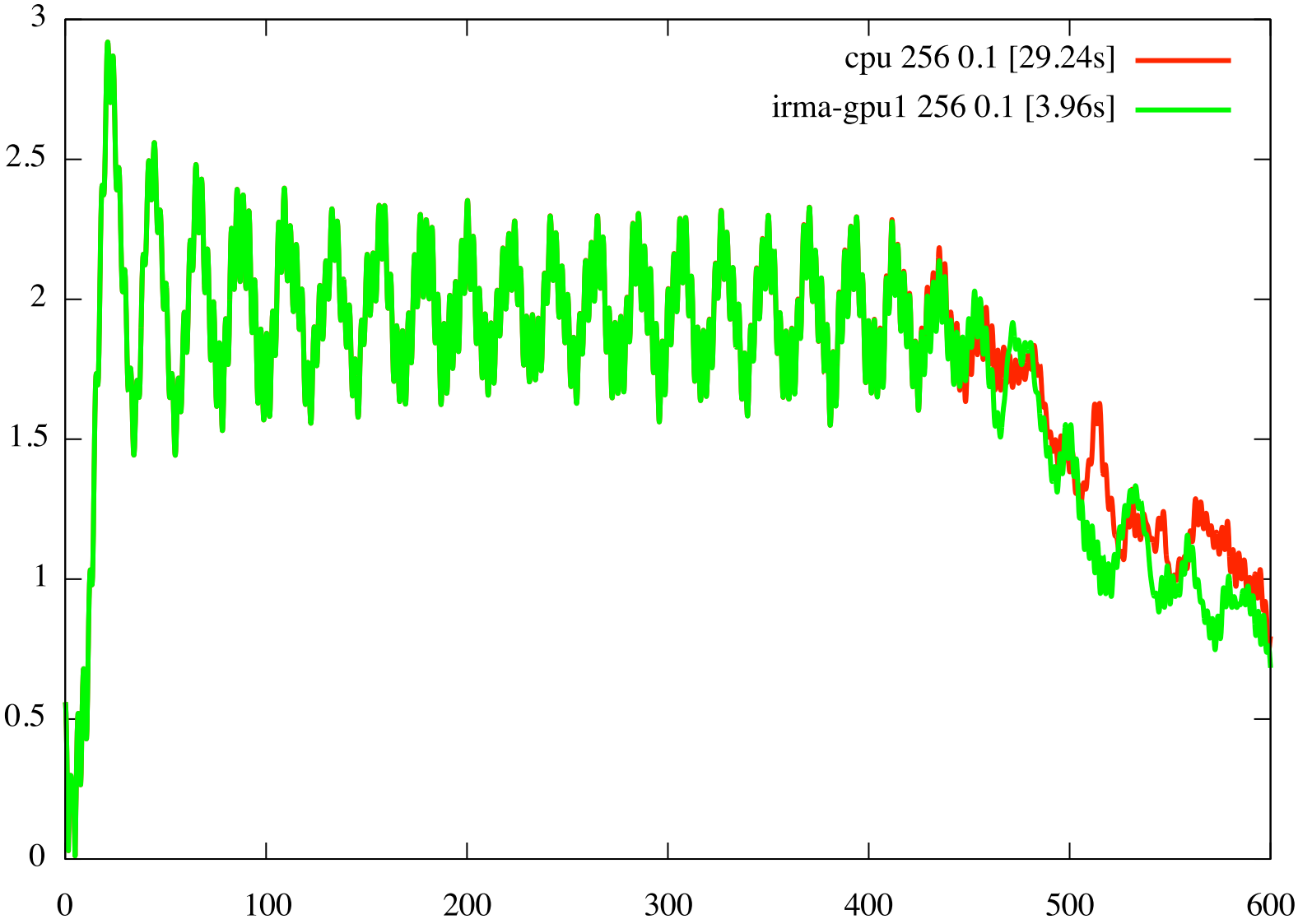
Landau $N = 1024$ $\Delta t = 0.1$ $\alpha = 0.01$



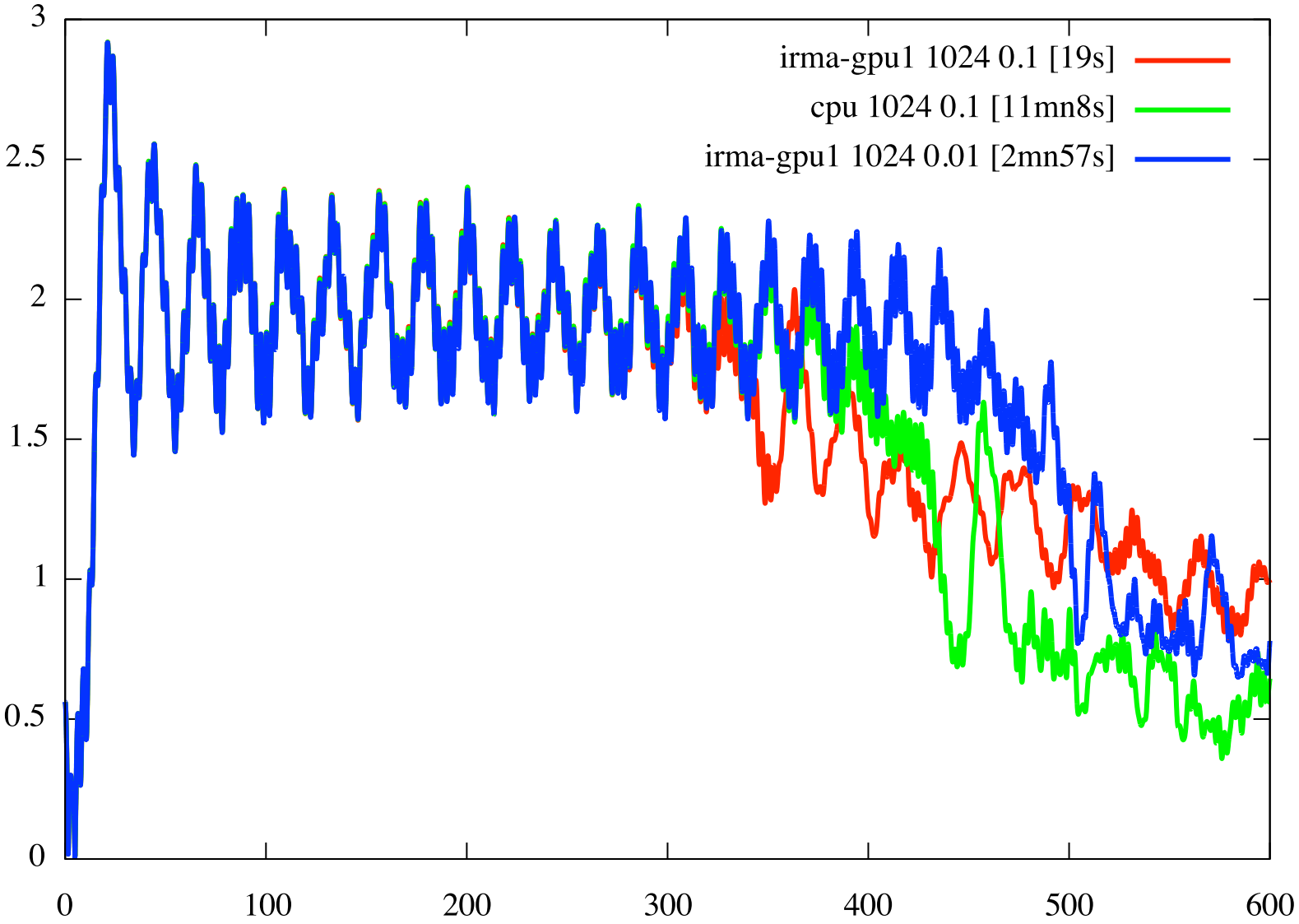
Landau delta f method



Bump on tail LAG17 $N = 256$



Bump on tail LAG9 $N = 1024$



performances Landau $N = 1024$, $nbstep = 600$

Time TOTAL : 1 507 405 microsec

Ratio TRANSFERTS : 92.3 %
Ratio FFT : 4.42 %
Ratio TRANSPOSITION : 0.798 %
Ratio COEFFICIENTS COMPUTE : 1.32 %

SUM RATIOS : 98.8 %

---

performances Landau (no transfer)

Transfer subroutines are commented in the code

Time TOTAL : 1 350 468 microsec

Ratio FFT : 65.3 %
Ratio TRANSPOSITION : 21.8 %
Ratio COEFFICIENTS COMPUTE : 11.7 %

SUM RATIOS : 98.8 %

Conclusions/ Perspectives

- Valid approach (speed-up of 25 for $N = 1024$ simulation)
- Delta f method validated in this framework

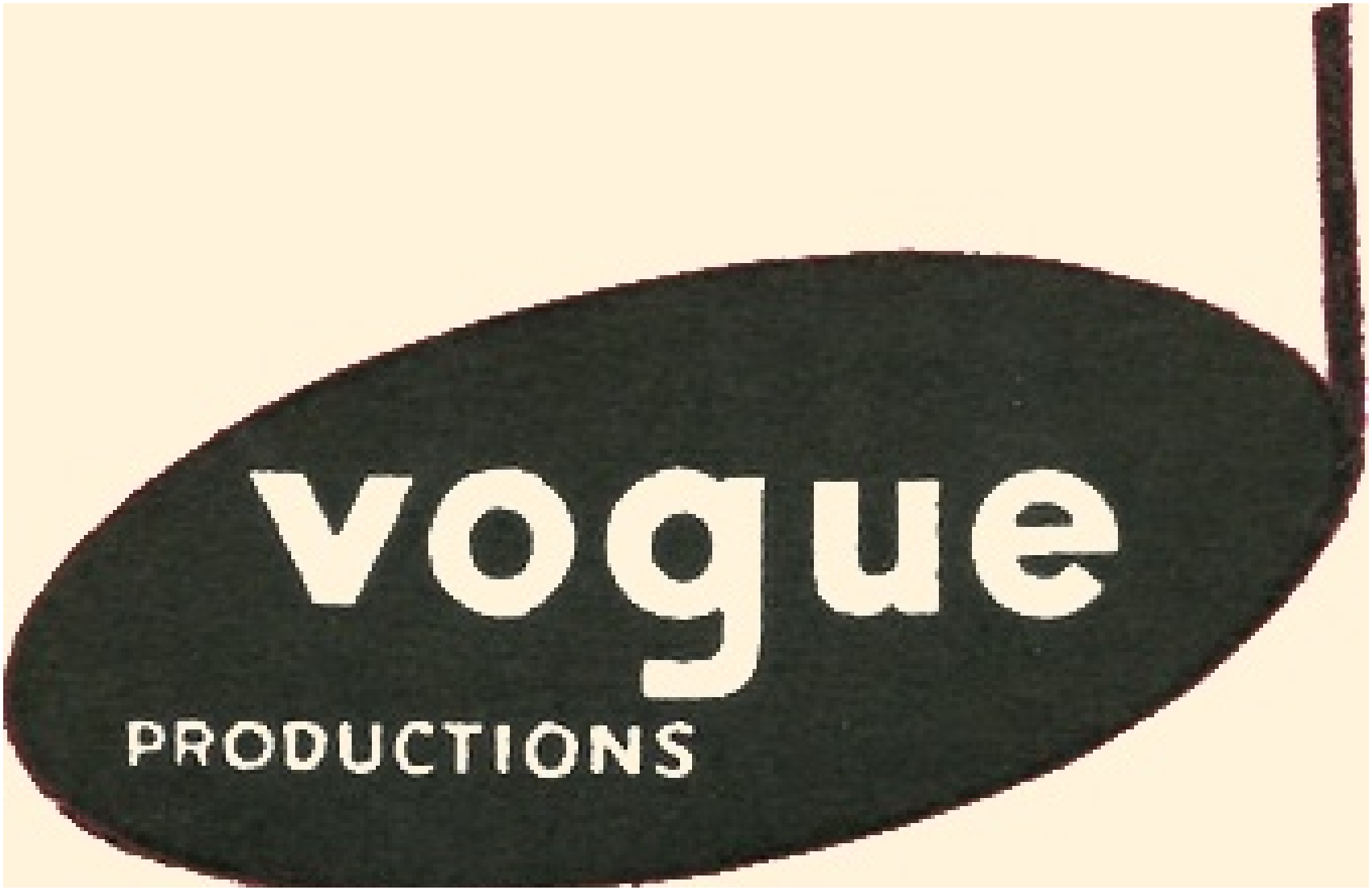
$\Rightarrow$ Suppress transfers

$\Rightarrow$ Go to double precision

$\Rightarrow$ Adapt to Keen code

$\Rightarrow$ GPU Visualization

Thank you for your attention



Other performance results

irma-gpu1 landau 4.6s
irma-gpu1 landau delta f 5.5s
MacBook fortran cpu landau 1mn52s
MacBook landau 56s
MacBook landau delta f 59.6s
irma-gpu1 fortran cpu landau 2m07s
MacBook Pro fortran cpu landau 1mn42s
MacBook = 2.4 GHz Intel Core 2 Duo Memory
MacBook Pro= 2.4 GHz Intel Core i5
Nx=Nv=1024 dt=0.1 nbstep=1000 LAG17
(fft used for E)

Annex : delta f method

- We write

$$f(x,v) = \delta f(x,v) + f_{\text{ref}}(v),\ f_{\text{ref}}(v) = \frac{1}{\sqrt{2\pi}} \exp(-v^2/2).$$

- We advect δf instead of f
- In v advection, we have

$$\delta f^{\text{new}}(x,v) = \delta f^{\text{old}}(x,v^*) + f_{\text{ref}}(v^*) - f_{\text{ref}}(v),$$

with $v^* = v - \Delta t E(x)$.

I. VELOCITY SPECTRAL APPROXIMATION OF THE VLASOV-POISSON SYSTEM

Plasma mathematical model

We are interested in the Vlasov equation with the equivalent equation of the Poisson equation which we called the Ampere equation following :

$$\partial_t E = -\int_v f v dv \tag{1}$$

In this work, we consider that $x \in [0, L]$, and generally, we have $v \in \mathbb{R}$ but we suppose that for every $|v| > V$ we have $f(x, v, t) = 0$. So, we can take $v \in [-V, V]$.

Plasma mathematical model

We are interested in the Vlasov equation with the equivalent equation of the Poisson equation which we called the Ampere equation following :

$$\partial_t E = -\int_v f v dv \tag{1}$$

In this work, we consider that $x \in [0, L]$, and generally, we have $v \in \mathbb{R}$ but we suppose that for every $|v| > V$ we have $f(x, v, t) = 0$. So, we can take $v \in [-V, V]$.

Plasma mathematical model

We are interested in the Vlasov equation with the equivalent equation of the Poisson equation which we called the Ampere equation following :

$$\partial_t E = -\int_v f v dv \tag{1}$$

In this work, we consider that $x \in [0, L]$, and generally, we have $v \in \mathbb{R}$ but we suppose that for every $|v| > V$ we have $f(x, v, t) = 0$. So, we can take $v \in [-V, V]$.

Plasma mathematical model

We are interested in the Vlasov equation with the equivalent equation of the Poisson equation which we called the Ampere equation following :

$$\partial_t E = -\int_v f v dv \tag{1}$$

In this work, we consider that $x \in [0, L]$, and generally, we have $v \in \mathbb{R}$ but we suppose that for every $|v| > V$ we have $f(x, v, t) = 0$. So, we can take $v \in [-V, V]$.

Change of variables

We put

$$\begin{cases} \overline{v} & = & \frac{v}{V}, \\ \overline{x} & = & \frac{x}{L}, \\ \overline{f}(\overline{x}, \overline{v}, t) & = & f(x, v, t). \end{cases}$$

So, $\overline{v} \in [-1, 1]$, $\overline{x} \in [0, 1]$ and

$$\overline{t} = \frac{\overline{x}}{\overline{v}} = \frac{x}{v} \cdot \frac{V}{L} = \frac{t}{T}, \tag{2}$$

with $T = \frac{L}{V}$. The Vlasov-Ampere equations become

$$\begin{cases} \partial_t f + v\partial_x f + E\partial_v f & = & 0, \\ \partial_t E & = & -\frac{L^2}{V} \int_{-1}^{1} f v dv, \end{cases} \tag{3}$$

where

$$x \in [0, 1], \ v \in [-1, 1],$$

Change of variables

We put

$$\begin{cases} \overline{v} & = & \frac{v}{V}, \\ \overline{x} & = & \frac{x}{L}, \\ \overline{f}(\overline{x},\overline{v},t) & = & f(x,v,t). \end{cases}$$

So, $\overline{v} \in [-1,1]$, $\overline{x} \in [0,1]$ and

$$\overline{t} = \frac{\overline{x}}{\overline{v}} = \frac{x}{v}.\frac{V}{L} = \frac{t}{T}, \tag{2}$$

with $T = \frac{L}{V}$. The Vlasov-Ampere equations become

$$\begin{cases} \partial_t f + v\partial_x f + E\partial_v f & = & 0, \\ \partial_t E & = & -\frac{L^2}{V}\int_{-1}^{1} f v dv, \end{cases} \tag{3}$$

where

$$x \in [0,1], \ v \in [-1,1],$$

Change of variables

We put

$$\begin{cases} \overline{v} & = & \frac{v}{V}, \\ \overline{x} & = & \frac{x}{L}, \\ \overline{f}(\overline{x},\overline{v},t) & = & f(x,v,t). \end{cases}$$

So, $\overline{v} \in [-1,1]$, $\overline{x} \in [0,1]$ and

$$\overline{t} = \frac{\overline{x}}{\overline{v}} = \frac{x}{v}.\frac{V}{L} = \frac{t}{T}, \tag{2}$$

with $T = \frac{L}{V}$. The Vlasov-Ampere equations become

$$\begin{cases} \partial_t f + v\partial_x f + E\partial_v f & = & 0, \\ \partial_t E & = & -\frac{L^2}{V}\int_{-1}^{1} f v dv, \end{cases} \tag{3}$$

where

$$x \in [0,1], \ v \in [-1,1],$$

Change of variables

We put

$$\begin{cases} \overline{v} & = & \frac{v}{V}, \\ \overline{x} & = & \frac{x}{L}, \\ \overline{f}(\overline{x},\overline{v},t) & = & f(x,v,t). \end{cases}$$

So, $\overline{v} \in [-1,1]$, $\overline{x} \in [0,1]$ and

$$\overline{t} = \frac{\overline{x}}{\overline{v}} = \frac{x}{v}.\frac{V}{L} = \frac{t}{T}, \tag{2}$$

with $T = \frac{L}{V}$. The Vlasov-Ampere equations become

$$\begin{cases} \partial_t f + v\partial_x f + E\partial_v f & = & 0, \\ \partial_t E & = & -\frac{L^2}{V}\int_{-1}^{1} f v dv, \end{cases} \tag{3}$$

where

$$x \in [0,1], \ v \in [-1,1],$$

Change of variables

We put

$$\begin{cases} \overline{v} & = & \frac{v}{V}, \\ \overline{x} & = & \frac{x}{L}, \\ \overline{f}(\overline{x},\overline{v},t) & = & f(x,v,t). \end{cases}$$

So, $\overline{v} \in [-1,1]$, $\overline{x} \in [0,1]$ and

$$\overline{t} = \frac{\overline{x}}{\overline{v}} = \frac{x}{v}.\frac{V}{L} = \frac{t}{T}, \tag{2}$$

with $T = \frac{L}{V}$. The Vlasov-Ampere equations become

$$\begin{cases} \partial_t f + v\partial_x f + E\partial_v f & = & 0, \\ \partial_t E & = & -\frac{L^2}{V}\int_{-1}^{1} f v dv, \end{cases} \tag{3}$$

where

$$x \in [0,1], \ v \in [-1,1],$$

Change of variables

We put

$$\begin{cases} \overline{v} & = & \frac{v}{V}, \\ \overline{x} & = & \frac{x}{L}, \\ \overline{f}(\overline{x},\overline{v},t) & = & f(x,v,t). \end{cases}$$

So, $\overline{v} \in [-1,1]$, $\overline{x} \in [0,1]$ and

$$\overline{t} = \frac{\overline{x}}{\overline{v}} = \frac{x}{v}.\frac{V}{L} = \frac{t}{T}, \tag{2}$$

with $T = \frac{L}{V}$. The Vlasov-Ampere equations become

$$\begin{cases} \partial_t f + v\partial_x f + E\partial_v f & = & 0, \\ \partial_t E & = & -\frac{L^2}{V}\int_{-1}^{1} f v dv, \end{cases} \tag{3}$$

where

$$x \in [0,1], \ v \in [-1,1],$$

Change of variables

We put

$$\begin{cases} \overline{v} & = & \frac{v}{V}, \\ \overline{x} & = & \frac{x}{L}, \\ \overline{f}(\overline{x},\overline{v},t) & = & f(x,v,t). \end{cases}$$

So, $\overline{v} \in [-1,1]$, $\overline{x} \in [0,1]$ and

$$\overline{t} = \frac{\overline{x}}{\overline{v}} = \frac{x}{v}.\frac{V}{L} = \frac{t}{T}, \tag{2}$$

with $T = \frac{L}{V}$. The Vlasov-Ampere equations become

$$\begin{cases} \partial_t f + v\partial_x f + E\partial_v f & = & 0, \\ \partial_t E & = & -\frac{L^2}{V}\int_{-1}^{1} f v dv, \end{cases} \tag{3}$$

where

$$x \in [0,1], \ v \in [-1,1],$$

Legendre polynomials

The (normalized) Legendre polynomials are defined, for $n \geq 0$, by

$$L_n(x) = \frac{\sqrt{n+\frac{1}{2}}}{n!2^n}\frac{d^n}{dx^n}((x^2-1)^n).$$

They satisfy the orthogonality condition

$$\int_{v=-1}^{1} L_i(v)L_j(v)dv = \delta_{ij}. \tag{4}$$

The k zeros of L_k are distincts and in $]-1,1[$. We denote by $(\xi_i)_{i=1\cdots N}$ the zeros of L_N and by

$$\omega_i = \frac{-\sqrt{2N+1}\sqrt{2N+3}}{(N+1)L'_N(\xi_i)L_{N+1}(\xi_i)}$$

the integration weights. Then, we have the quadrature formula

$$\int_{-1}^{1} f(v)dv \simeq \sum_{i-1}^{N} \omega_i f(\xi_i)$$

Legendre polynomials

The (normalized) Legendre polynomials are defined, for $n \geq 0$, by

$$L_n(x) = \frac{\sqrt{n+\frac{1}{2}}}{n!2^n}\frac{d^n}{dx^n}((x^2-1)^n).$$

They satisfy the orthogonality condition

$$\int_{v=-1}^{1} L_i(v)L_j(v)dv = \delta_{ij}. \tag{4}$$

The k zeros of L_k are distincts and in $]-1,1[$. We denote by $(\xi_i)_{i=1\cdots N}$ the zeros of L_N and by

$$\omega_i = \frac{-\sqrt{2N+1}\sqrt{2N+3}}{(N+1)L'_N(\xi_i)L_{N+1}(\xi_i)}$$

the integration weights. Then, we have the quadrature formula

$$\int_{-1}^{1} f(v)dv \simeq \sum_{i-1}^{N} \omega_i f(\xi_i)$$

Legendre polynomials

The (normalized) Legendre polynomials are defined, for $n \geq 0$, by

$$L_n(x) = \frac{\sqrt{n+\frac{1}{2}}}{n!2^n}\frac{d^n}{dx^n}((x^2-1)^n).$$

They satisfy the orthogonality condition

$$\int_{v=-1}^{1} L_i(v)L_j(v)dv = \delta_{ij}. \tag{4}$$

The k zeros of L_k are distincts and in $]-1,1[$. We denote by $(\xi_i)_{i=1\cdots N}$ the zeros of L_N and by

$$\omega_i = \frac{-\sqrt{2N+1}\sqrt{2N+3}}{(N+1)L'_N(\xi_i)L_{N+1}(\xi_i)}$$

the integration weights. Then, we have the quadrature formula

$$\int_{-1}^{1} f(v)dv \simeq \sum_{i-1}^{N} \omega_i f(\xi_i)$$

Legendre polynomials

The (normalized) Legendre polynomials are defined, for $n \geq 0$, by

$$L_n(x) = \frac{\sqrt{n+\frac{1}{2}}}{n!2^n}\frac{d^n}{dx^n}((x^2-1)^n).$$

They satisfy the orthogonality condition

$$\int_{v=-1}^{1} L_i(v)L_j(v)dv = \delta_{ij}. \tag{4}$$

The k zeros of L_k are distincts and in $]-1,1[$. We denote by $(\xi_i)_{i=1\cdots N}$ the zeros of L_N and by

$$\omega_i = \frac{-\sqrt{2N+1}\sqrt{2N+3}}{(N+1)L'_N(\xi_i)L_{N+1}(\xi_i)}$$

the integration weights. Then, we have the quadrature formula

$$\int_{-1}^{1} f(v)dv \simeq \sum_{i-1}^{N} \omega_i f(\xi_i)$$

Legendre polynomials

The (normalized) Legendre polynomials are defined, for $n \geq 0$, by

$$L_n(x) = \frac{\sqrt{n+\frac{1}{2}}}{n!2^n}\frac{d^n}{dx^n}((x^2-1)^n).$$

They satisfy the orthogonality condition

$$\int_{v=-1}^{1} L_i(v)L_j(v)dv = \delta_{ij}. \tag{4}$$

The k zeros of L_k are distincts and in $]-1,1[$. We denote by $(\xi_i)_{i=1\cdots N}$ the zeros of L_N and by

$$\omega_i = \frac{-\sqrt{2N+1}\sqrt{2N+3}}{(N+1)L'_N(\xi_i)L_{N+1}(\xi_i)}$$

the integration weights. Then, we have the quadrature formula

$$\int_{-1}^{1} f(v)dv \simeq \sum_{i-1}^{N} \omega_i f(\xi_i)$$

Legendre polynomials

The (normalized) Legendre polynomials are defined, for $n \geq 0$, by

$$L_n(x) = \frac{\sqrt{n+\frac{1}{2}}}{n!2^n}\frac{d^n}{dx^n}((x^2-1)^n).$$

They satisfy the orthogonality condition

$$\int_{v=-1}^{1} L_i(v)L_j(v)dv = \delta_{ij}. \tag{4}$$

The k zeros of L_k are distincts and in $]-1,1[$. We denote by $(\xi_i)_{i=1\cdots N}$ the zeros of L_N and by

$$\omega_i = \frac{-\sqrt{2N+1}\sqrt{2N+3}}{(N+1)L'_N(\xi_i)L_{N+1}(\xi_i)}$$

the integration weights. Then, we have the quadrature formula

$$\int_{-1}^{1} f(v)dv \simeq \sum_{i=1}^{N} \omega_i f(\xi_i)$$

Legendre polynomials

The (normalized) Legendre polynomials are defined, for $n \geq 0$, by

$$L_n(x) = \frac{\sqrt{n+\frac{1}{2}}}{n!2^n}\frac{d^n}{dx^n}((x^2-1)^n).$$

They satisfy the orthogonality condition

$$\int_{v=-1}^{1} L_i(v)L_j(v)dv = \delta_{ij}. \tag{4}$$

The k zeros of L_k are distincts and in $]-1,1[$. We denote by $(\xi_i)_{i=1\cdots N}$ the zeros of L_N and by

$$\omega_i = \frac{-\sqrt{2N+1}\sqrt{2N+3}}{(N+1)L'_N(\xi_i)L_{N+1}(\xi_i)}$$

the integration weights. Then, we have the quadrature formula

$$\int_{-1}^{1} f(v)dv \simeq \sum_{i=1}^{N} \omega_i f(\xi_i)$$

Legendre polynomials

The (normalized) Legendre polynomials are defined, for $n \geq 0$, by

$$L_n(x) = \frac{\sqrt{n+\frac{1}{2}}}{n!2^n}\frac{d^n}{dx^n}((x^2-1)^n).$$

They satisfy the orthogonality condition

$$\int_{v=-1}^{1} L_i(v)L_j(v)dv = \delta_{ij}. \tag{4}$$

The k zeros of L_k are distincts and in $]-1,1[$. We denote by $(\xi_i)_{i=1\cdots N}$ the zeros of L_N and by

$$\omega_i = \frac{-\sqrt{2N+1}\sqrt{2N+3}}{(N+1)L'_N(\xi_i)L_{N+1}(\xi_i)}$$

the integration weights. Then, we have the quadrature formula

$$\int_{-1}^{1} f(v)dv \simeq \sum_{i=1}^{N} \omega_i f(\xi_i)$$

Legendre polynomials

The (normalized) Legendre polynomials are defined, for $n \geq 0$, by

$$L_n(x) = \frac{\sqrt{n+\frac{1}{2}}}{n!2^n}\frac{d^n}{dx^n}((x^2-1)^n).$$

They satisfy the orthogonality condition

$$\int_{v=-1}^{1} L_i(v)L_j(v)dv = \delta_{ij}. \tag{4}$$

The k zeros of L_k are distincts and in $]-1,1[$. We denote by $(\xi_i)_{i=1\cdots N}$ the zeros of L_N and by

$$\omega_i = \frac{-\sqrt{2N+1}\sqrt{2N+3}}{(N+1)L'_N(\xi_i)L_{N+1}(\xi_i)}$$

the integration weights. Then, we have the quadrature formula

$$\int_{-1}^{1} f(v)dv \simeq \sum_{i-1}^{N} \omega_i f(\xi_i)$$

Spectral velocity expansion

We now approximate the distribution function by a finite expansion on the Legendre basis

$$f(x,v,t) \simeq \sum_{k=0}^{N-1} w_k(x,t) L_k(v), \tag{5}$$

after computing, we obtain, for $m = 0 \cdots N-1$

$$\partial_t w_m + \sum_{k=0}^{N-1} \left(\int_v v L_m(v) L_k(v) \right) \partial_x w_k + E \sum_{k=0}^{N-1} \left(\int_v L_m(v) L_k'(v) \right) w_k = 0. \tag{6}$$

The ampere equation becomes

$$\partial_t E = -\frac{L^2}{V} \sqrt{2/3} w_1.$$

PDE system

We introduce the vector

$$W = \left(\; w_0, \quad w_1, \cdots \quad , w_{N-1}, E \; \right)^T,$$

The Vlasov-Ampere equations become

$$\partial_t W + A \partial_x W = S, \tag{7}$$

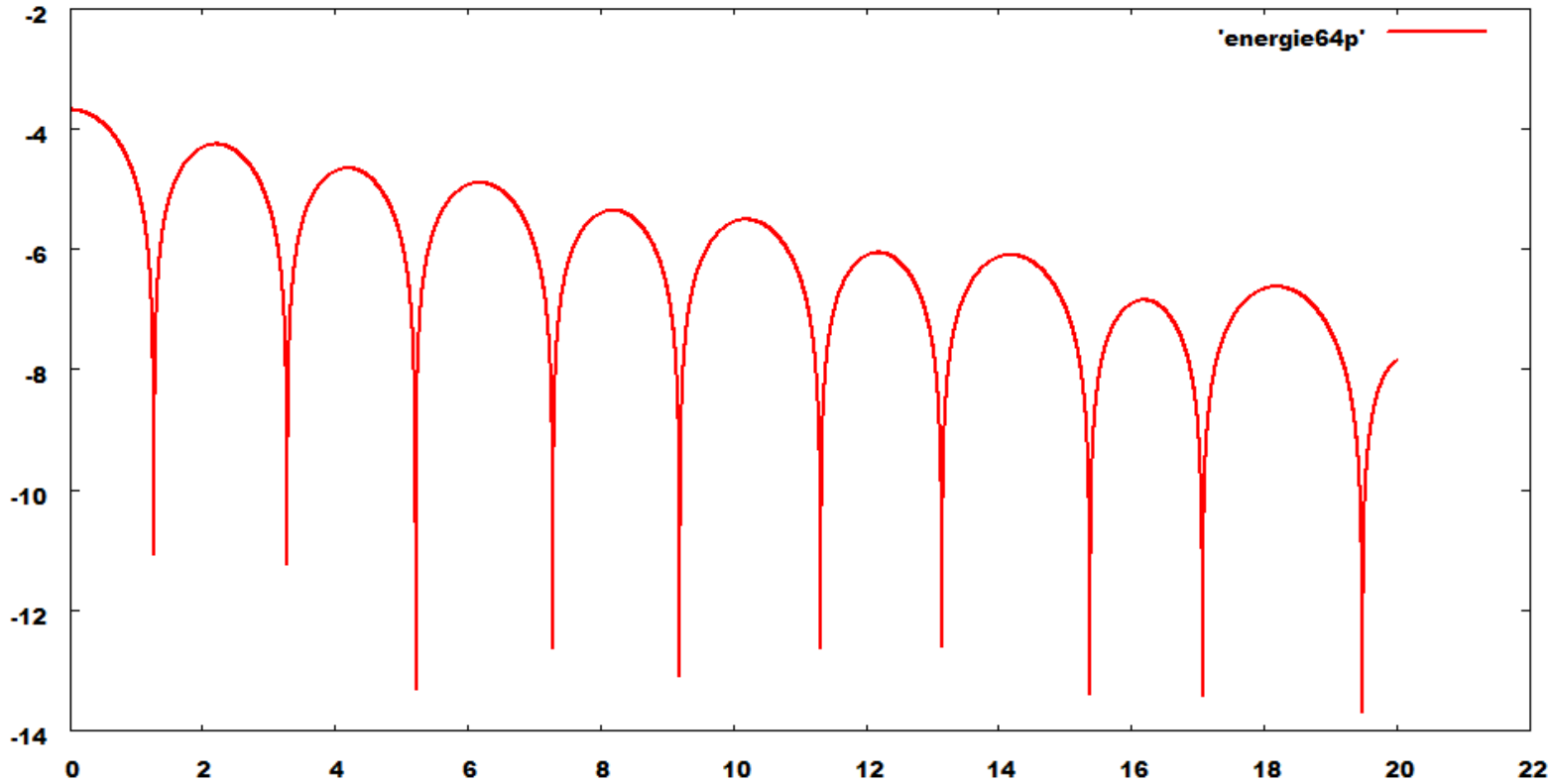
This system is hyperbolic because A is a symmetric matrix.

Numerical schemes

We programmed and tested the system with :

- The centered scheme with Runge-Kutta method
- The upwind scheme with Euler method

Result :



Discontinuous Galerkin GPU code for hyperbolic systems

What is done before Cemrac's ?

- Philippe Helluy and Thomas Strub developped a 3 dimensional code that solves the 3 dimensional Maxwell equation with a Galerkin Discontinuous method.
- The code works on an unstructured mesh.
- We have different zones where the approximation can be at different degree.

Problem

Consider a system of conservation law defined on a open space Ω and a familly of open space $(L_k)_{k=1\dots N}$ such that $\overline{\cup_k L_k} = \overline{\Omega}$, for which the Galerkin Discontinuous method can be written in the form

$$0 = \int_L dt\ W \cdot \varphi_L - \int_L f(W, W, \nabla\varphi_L) + \int_{\partial L \cap \Omega} f(W_L, W_R, n) \cdot \varphi_L$$

where φ_L is basis function that has its support in L.

- Assume that $(L_k)_{k=1\dots N}$ is a familly of hexahedral that constitutes a conform mesh.
- Assume that $\Omega = [0,1]^3$ to simplify the notation.

To compute the time derivative we need

- $\int_{\partial L \cap \Omega} f(W_L, W_R, n) \cdot \varphi_L$,
- $\int_L f(W, W, \nabla\varphi_L)$.

Quadrature

We approach the volume integration by

$$\int_L h(X)dX \simeq \sum_{k=0}^{(d+1)^3-1} \omega_k h(\chi_k)$$

and the surface integration by

$$\int_{\partial L} h(X)ds \simeq \sum_{f=1}^{6} \sum_{k=0}^{(d+1)^2-1} \omega_k^f h(\chi_k^f)$$

where

- $\chi_i = (\hat{x}_p, \hat{x}_q, \hat{x}_r)^T$ and $\omega_i = \hat{\omega}_p \hat{\omega}_q \hat{\omega}_r$ are the points and weights of quadrature in K,
- χ_i^f and $\omega_i^f = \hat{\omega}_p \hat{\omega}_q$, the points and weights of quadrature on the faces of K.

"Old" version : unstructured mesh

- The code uses a MPI parallelization : each process treats one part of the mesh then there are exchange zones.
- For each MPI domain we use an OpenCL parallelization :
  For the time derivative computation, the parallelization is done on the surface or volume integration points
  - Number of work-items = max(number of surface points, number of volume points).
  - the work-items that correspond to a same element are grouped in a work group.
- Problem of this algorithm : the memory access in global memory are not always coalescent (example of $degree = 0$).

"New" version

Definition

A structured zone is a zone without hole where the cells can be referenced by

$$L_{i,j,k},\ 0 \leq i \leq N_x - 1,\ 0 \leq j \leq N_y - 1,\ 0 \leq k \leq N_z - 1.$$

The idea of the new version is

- to test if there are structured or not,
- reorder all the structured zones such that element i and $i+1$ are neighbour in the x direction and write the vector W such that E_{i+1} is the next value after E_i in W, etc for the other component,
- apply the "old" algorithm on the unstructured zone and a new algorithm for structured zones.

Algorithm for structured zone

On a structured zone the parallelization is done on the element

- we compute the face fluxes in "x" direction,
- we transpose the vector W to have coalescent acces in "y" direction,
- we compute the face fluxes in "y" direction,
- we do the same thing for the "z" direction.

- Number of work-items = number of elements in the zone= $N_x \times N_y \times N_z$.
- the work-items corresponding to a same line are grouped in a work group.

Work

During the Cemracs, we

- wrote the algorithm that detect the structured zone and reorder it,
- wrote the algorithm for structured zone,
- wrote the transposition to have coalescent access between x and y direction.

Job to do

- optimize the transposition (use the cash memory),
- hide the communications between each MPI domain.