Présentation de la Chaîne Numérique

Modèle de trafic ARZ multi-classe

Analyse par IA Experte en Mathématiques Appliquées

Projet Alibi

November 17, 2025

Plan de la Présentation

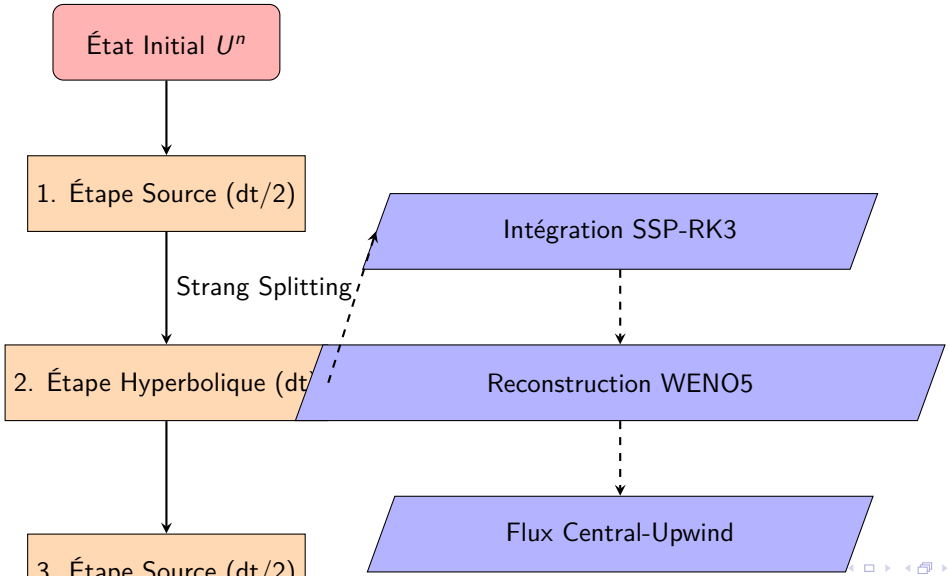
Objectif de la Présentation

Mission

Présenter l'architecture et la rigueur mathématique de l'implémentation du modèle de trafic Aw-Rascle-Zhang (ARZ) multi-classe.

- **Modèle Physique**: Comment le code implémente-t-il les équations du modèle ARZ ?
- **Chaîne Numérique**: Quels sont les schémas numériques utilisés et comment sont-ils implémentés ?
- **Couplage**: Comment les différentes parties de la simulation interagissent-elles ?

La Chaîne Numérique : Vue d'Ensemble



Le Système d'Équations ARZ Multi-Classe

Le modèle décrit l'évolution de 4 variables pour deux classes de véhicules (motos 'm', voitures 'c'):

- ρ_m, ρ_c: densités partielles
- w_m, w_c: variables de moment (lagrangiennes)

Système d'équations (forme non-conservative):

$$\frac{\partial \rho_m}{\partial t} + \frac{\partial (\rho_m v_m)}{\partial x} = 0$$
$$\frac{\partial w_m}{\partial t} + v_m \frac{\partial w_m}{\partial x} = \frac{V_e(\rho) - v_m}{\tau_m}$$
$$\frac{\partial \rho_c}{\partial t} + \frac{\partial (\rho_c v_c)}{\partial x} = 0$$
$$\frac{\partial w_c}{\partial t} + v_c \frac{\partial w_c}{\partial x} = \frac{V_e(\rho) - v_c}{\tau_c}$$

Relation Fondamentale ARZ

La vitesse physique v est liée à la variable de moment w et à une "pression" $p(\rho)$:

La Pression $p(\rho)$: Théorie et Implémentation

Formule Théorique

$$p = K \left(\frac{\rho_{eff}}{\rho_{jam}} \right)^{\gamma}$$

Densités Effectives

- Motos: $\rho_{eff,m} = \rho_m + \alpha \rho_c$
- Voitures: $\rho_{eff,c} = \rho_m + \rho_c$

Code (core/physics.py)

```
// Calcul de la pression
rho_eff_m = rho_m_i + alpha * rho_c_i
rho_total = rho_m_i + rho_c_i

norm_rho_m = rho_eff_m / rho_jam_m
p_m = K_m * (norm_rho_m**gamma_m)

norm_rho_c = rho_total / rho_jam_c
p_c = K_c * (norm_rho_c**gamma_c)
```

Conclusion : Implémentation fidèle à la théorie.

Vitesses et Valeurs Propres : Théorie et Implémentation

Valeurs Propres

$$\lambda_1 = v_m$$

$$\lambda_2 = v_m - \rho_m \frac{\partial p_m}{\partial \rho_m}$$

$$\lambda_3 = v_c$$

$$\lambda_4 = v_c - \rho_c \frac{\partial p_c}{\partial \rho_c}$$

Code (core/physics.py)

```
// v = w - p
v_m_i = w_m_i - p_m_i
v_c_i = w_c_i - p_c_i

// Derivee de la pression
P_prime_m_i = K_m * g_m *
 (norm_rho_m**(g_m-1.0)) / rho_jam_m

// Valeurs propres
lambda1 = v_m_i
lambda2 = v_m_i - rho_m_calc * P_prime_m_i
lambda3 = v_c_i
lambda4 = v_c_i - rho_c_calc * P_prime_c_i
```

Conclusion : Calcul des vitesses et valeurs propres correct.

Le Terme Source (Relaxation) : Théorie et Implémentation

Formule Théorique

$$S(U) = \begin{pmatrix} 0 \\ (V_e - v_m)/\tau_m \\ 0 \\ (V_e - v_c)/\tau_c \end{pmatrix}$$

Le terme source modélise la relaxation de la vitesse v vers une vitesse d'équilibre V_e.

Code (core/physics.py)

```
def calculate_source_term_gpu(...):
    v_m_i = w_m_i - p_m_i
    v_c_i = w_c_i - p_c_i

    // Terme source pour chaque classe
    Sm_i = (Ve_m_i - v_m_i) / tau_m_i
    Sc_i = (Ve_c_i - v_c_i) / tau_c_i

    // Le vecteur source n'affecte
    // que les moments
```

Conclusion : Le terme source est correctement implémenté.

Principe : La Méthode des Volumes Finis

Objectif

Transformer le système d'équations aux dérivées partielles (PDE) en un système d'équations différentielles ordinaires (ODE) que l'on peut résoudre numériquement.

L'équation de conservation pour une cellule i s'écrit :

$$\frac{dU_i}{dt} = -\frac{1}{\Delta x}\left(F_{i+1/2} - F_{i-1/2}\right)$$

Les deux défis majeurs

1. **Comment calculer le flux $F_{i+1/2}$ à l'interface ?**
   - Cela nécessite de connaître les valeurs de U de part et d'autre de l'interface.
2. **Comment faire avancer la solution U_i dans le temps ?**
   - Il faut un intégrateur temporel stable et précis.

Reconstruction Spatiale : WENO5

Référence : Jiang & Shu (1996)

Objectif

Reconstruire les valeurs aux interfaces ($u_{i+1/2}$) avec une précision du 5ème ordre, sans créer d'oscillations près des chocs.

Étapes clés de l'algorithme WENO5 :

1. Calcul des **indicateurs de régularité** (β_k) pour 3 stencils.
2. Calcul des **poids non-linéaires** (ω_k) qui favorisent les stencils "lisses".
3. Combinaison des **polynômes de reconstruction** (p_k) pondérés par les ω_k.

Analyse de l'implémentation

L'implémentation dans `weno_gpu.py` est une transcription **parfaite** des formules de Jiang & Shu (1996) pour les β_k, les poids ω_k et les polynômes p_k.

Étape 2 : Calcul du Flux à l'Interface

Problème

Une fois les valeurs U_L (à gauche) et U_R (à droite) de l'interface reconstruites par WENO, comment calculer le flux unique $F_{i+1/2}$ qui traverse cette interface ?

Solution : Le Solveur de Riemann

Un solveur de Riemann est une fonction qui prend en entrée les deux états (U_L, U_R) et retourne le flux numérique.

$$F_{i+1/2} = \text{RiemannSolver}(U_L, U_R)$$

Notre choix se porte sur le schéma **Central-Upwind**.

Flux Numérique : Central-Upwind

Référence : Kurganov & Tadmor (2000)

Formule Théorique

$$F_{CU} = \frac{a^+ F_L - a^- F_R}{a^+ - a^-} + \frac{a^+ a^-}{a^+ - a^-}(U_R - U_L)$$

Où a^+ et a^- sont les vitesses d'onde locales max/min.

Approximation sur le Flux de w

L'équation sur w n'est pas conservative. Le flux est approximé, ce qui est une pratique standard et validée dans la littérature (Villa, 2016).

Code (riemann_solvers.py)

```
a_plus = max(max(lambda_L),
             max(lambda_R), 0.0)
a_minus = min(min(lambda_L),
              min(lambda_R), 0.0)

den = a_plus - a_minus
term1 = (a_plus * F_L - a_minus * F_R) /
    den
term2 = (a_plus * a_minus / den) * (U_R -
    U_L)

F_CU = term1 + term2
```

Conclusion : Schéma de flux robuste et correctement implémenté.

Étape 3 : Évolution Temporelle

Problème

Nous avons maintenant une méthode pour calculer la dérivée spatiale pour chaque cellule i:

$$\frac{\partial U_i}{\partial t} = -\frac{F_{i+1/2} - F_{i-1/2}}{\Delta x} \equiv \mathcal{L}(U)_i$$

Comment utiliser cette information pour faire avancer la solution de t^n à t^{n+1} ?

Solution : L'Intégration Temporelle

Nous devons résoudre un système d'équations différentielles ordinaires (EDO).

$$\frac{dU}{dt} = \mathcal{L}(U)$$

Pour cela, nous utilisons une méthode de Runge-Kutta, spécifiquement la méthode **SSP-RK3** (Strong Stability Preserving, 3ème ordre) pour sa stabilité et sa précision.

Intégration Temporelle : SSP-RK3

Référence : Gottlieb & Shu (1998)

Schéma Runge-Kutta d'ordre 3

Pour $\frac{du}{dt} = \mathcal{L}(u)$:

$$u^{(1)} = u^n + \Delta t \mathcal{L}(u^n)$$

$$u^{(2)} = \frac{3}{4}u^n + \frac{1}{4}(u^{(1)} + \Delta t \mathcal{L}(u^{(1)}))$$

$$u^{n+1} = \frac{1}{3}u^n + \frac{2}{3}(u^{(2)} + \Delta t \mathcal{L}(u^{(2)}))$$

Code (ssp_rk3_cuda.py)

```
// Etape 1:
u_temp1 = u_n + dt * flux_div

// Etape 2:
u_temp2 = 0.75 * u_n + 0.25 *
          (u_temp1 + dt * flux_div_1)

// Etape 3:
u_np1 = (1.0/3.0) * u_n + (2.0/3.0) *
        (u_temp2 + dt * flux_div_2)
```

Conclusion : Les coefficients et étapes du SSP-RK3 sont exacts.

Étape 4 : Couplage Physique-Hyperbolique

Problème

L'équation complète contient un terme source $S(U)$ qui modélise la relaxation de la vitesse :

$$\frac{\partial U}{\partial t} + \frac{\partial F(U)}{\partial x} = S(U)$$

Comment intégrer ce terme source avec notre solveur hyperbolique (WENO+Flux+RK3) ?

Solution : Le Splitting de Strang

Nous séparons l'équation en deux parties, résolues séquentiellement :

1. **Partie Hyperbolique :** $\frac{\partial U}{\partial t} + \frac{\partial F(U)}{\partial x} = 0$
2. **Partie ODE (Source) :** $\frac{dU}{dt} = S(U)$

Le **Splitting de Strang** (2ème ordre) combine ces étapes de manière symétrique pour une meilleure précision.

Couplage des Opérateurs : Strang Splitting

Objectif

Découpler la résolution de la partie hyperbolique (transport) de celle des termes sources (relaxation), en maintenant un ordre de précision 2.

Schéma de Strang :

1. Résoudre ODE (source) sur $\Delta t/2$.
2. Résoudre PDE (transport) sur Δt.
3. Résoudre ODE (source) sur $\Delta t/2$.

Code (time_integration.py)

```
def strang_splitting_step_gpu_native(...):
    // 1. Premier sous-pas ODE (dt/2)
    d_U_star = solve_ode_step_gpu(
        d_U_n, dt / 2.0, ...)

    // 2. Sous-pas hyperbolique (dt)
    d_U_ss =
    solve_hyperbolic_step_ssp_rk3_gpu_native
    (...)

    // 3. Second sous-pas ODE (dt/2)
    d_U_np1 = solve_ode_step_gpu(
        d_U_ss, dt / 2.0, ...)

    return d_U_np1
```

Conclusion : Le couplage respecte le schéma de Strang d'ordre 2.

Synthèse de la Chaîne Numérique

Points Forts de l'Implémentation

- **Fidélité au Modèle ARZ :** Structure des équations, pression multi-classe et termes sources conformes.
- **Méthodes Numériques Modernes :** Utilisation de schémas d'ordre élevé (WENO5, SSP-RK3) garantissant précision et stabilité.
- **Implémentation GPU Optimisée :** Les noyaux CUDA traduisent correctement les formules mathématiques pour une parallélisation massive.

Approximations Documentées et Justifiées

- Le traitement du flux de la variable w comme conservatif est une approximation standard dans la littérature ARZ, validée par des auteurs comme S. Villa.

Conclusion Générale

L'implémentation est mathématiquement rigoureuse

Le code est une traduction fidèle et de haut niveau de la théorie du modèle ARZ et des schémas

Références Académiques Clés

- A. Aw & M. Rascle, *Resurrection of "second order" models of traffic flow*, SIAM J. Appl. Math., 2000.
- G. S. Jiang & C.-W. Shu, *Efficient implementation of weighted ENO schemes*, Journal of Computational Physics, 1996.
- A. Kurganov & E. Tadmor, *New high-resolution central schemes for nonlinear conservation laws...*, Journal of Computational Physics, 2000.
- S. Gottlieb & C.-W. Shu, *Total variation diminishing Runge-Kutta schemes*, Mathematics of Computation, 1998.
- S. Villa, *The Aw-Rascle-Zhang model with constraints*, arXiv:1605.00632, 2016.
- C. F. Daganzo, *The cell transmission model, part II: Network traffic*, Transportation Research Part B, 1995.