Modèle Electrophysiologique sur GPU

Jean-Baptiste Keck
Gauthier Zirnhelt

18/06/2014

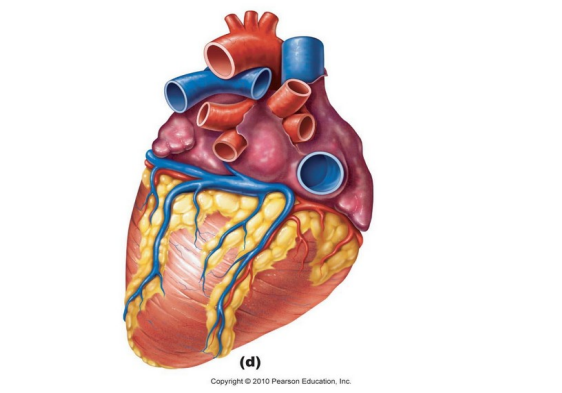
Encadrant: **Christophe Picard (LJK)**

Introduction

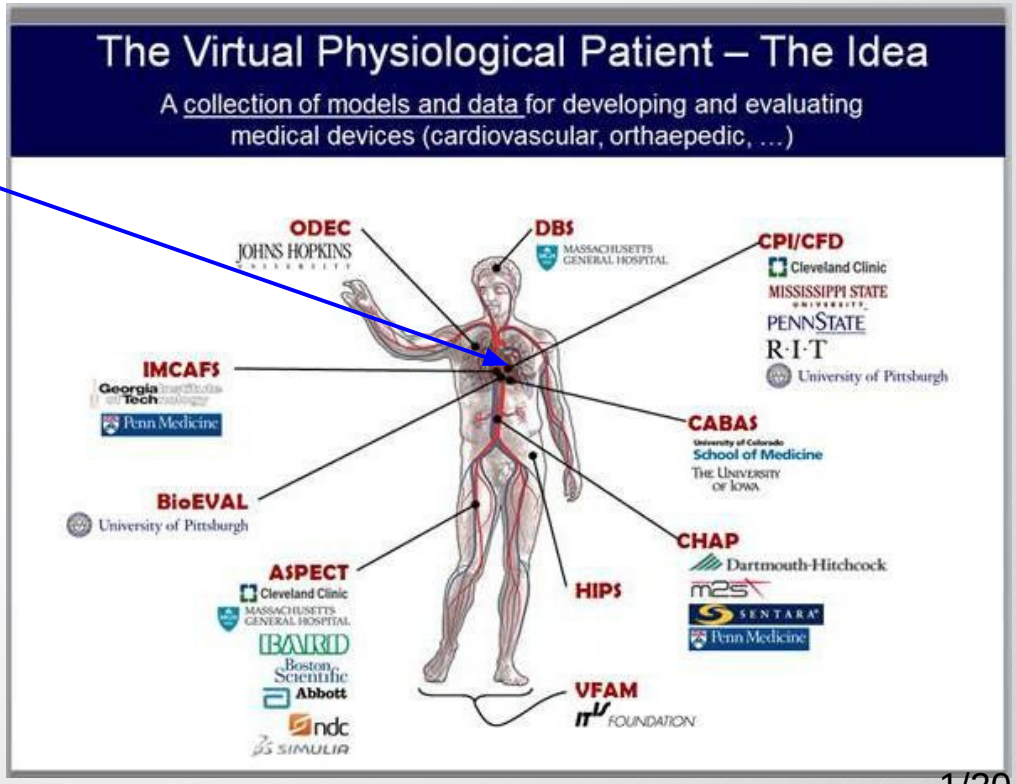
- **Projet européen** Virtual Physiological Human (VPH)
- **Objectif :** *simuler* un corps humain au niveau cellulaire
- Optique de **diagnostiques thérapeutiques** en médecine

Notre équipe :

L'électrophysiologie du coeur

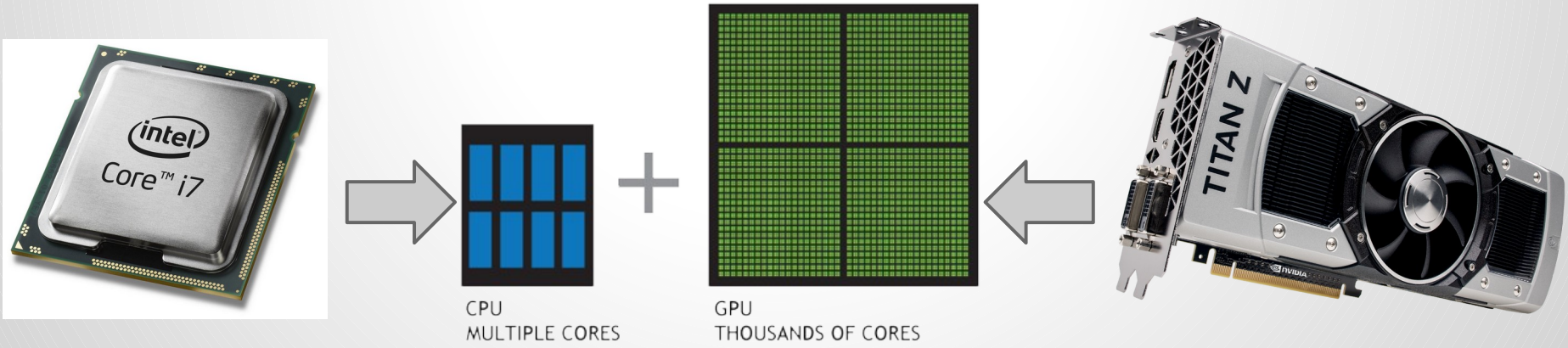


Étude des phénomènes électriques et électrochimiques qui se produisent dans les cellules.



Simulation

- **Simulations numériques en ingénierie :** Très gourmandes en ressources
- **Tendance actuelle :** GPGPU
- **GPU :** des milliers de coeurs de calcul



Périmètre et objectifs

- Périmètre
  - Systèmes cibles : GNU/Linux
  - Choix du langage : C++14 / Python
  - Technologies GPU : OpenCL / OpenGL
  - Interface graphique utilisateur : Qt

- Objectifs initialement fixés
  - Simulation d'un modèle simple en 3D en temps réel (grille $512 \times 512 \times 512$) sur la machine de référence, mais sans affichage.
  - Interface graphique minimale
  - Affichage de la simulation (au moins en 2D)

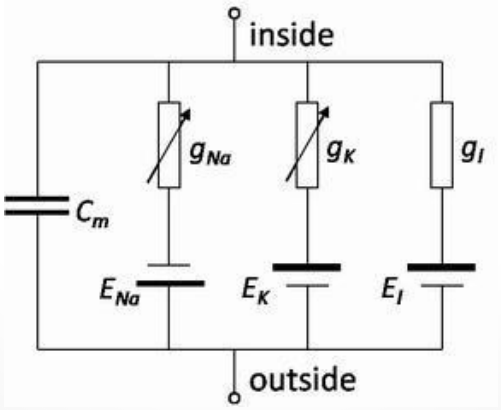
Périmètre et objectifs

- Objectifs additionnels
  - Utilisation de plusieurs cartes graphiques
  - Possibilité de sauvegarder les simulations
  - Possibilité de créer et d’ajouter simplement des modules supplémentaires:
    - Solveurs
    - Modèles
    - Conditions initiales
  - Possibilité de modifier les paramètres de simulation

Les différents modèles

- Modèle simple à deux variables (sur CPU et GPU)

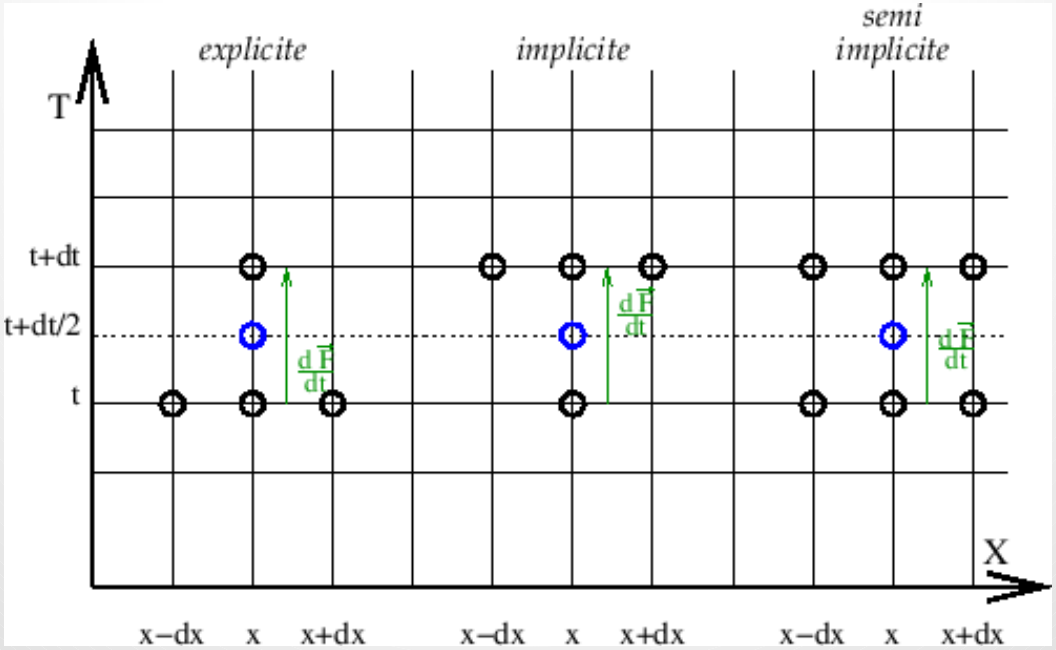
- Mais il en existe des dizaines d'autres
  - **Exemple: Modèle de Hodgkin–Huxley**



  - **De 2 à 40 variables !**

Les méthodes de résolution

- Méthodes de résolution
  - Résolution explicite
  - Gradient conjugué
  - Gradient biconjugué stabilisé



Le modèle de base

Modèle 2D à deux variables :

$$\begin{cases} \frac{\partial e}{\partial t} = F(e,r) = \nabla^t D \nabla e - Ke(e-\alpha_1)(e-1) - er) \\ \frac{\partial r}{\partial t} = G(e,r) = \left(\varepsilon + \mu_1 \frac{r}{1+\mu_2}\right)(-r - Ke(e-\alpha_2 - 1)) \end{cases}$$

Que l'on simplifie en considérant la conductivité constante :

$$\begin{cases} \frac{\partial e}{\partial t} = D\Delta e - Ke(e-\alpha_1)(e-1) - er) \\ \frac{\partial r}{\partial t} = \left(\varepsilon + \mu_1 \frac{r}{1+\mu_2}\right)(-r - Ke(e-\alpha_2 - 1)) \end{cases}$$

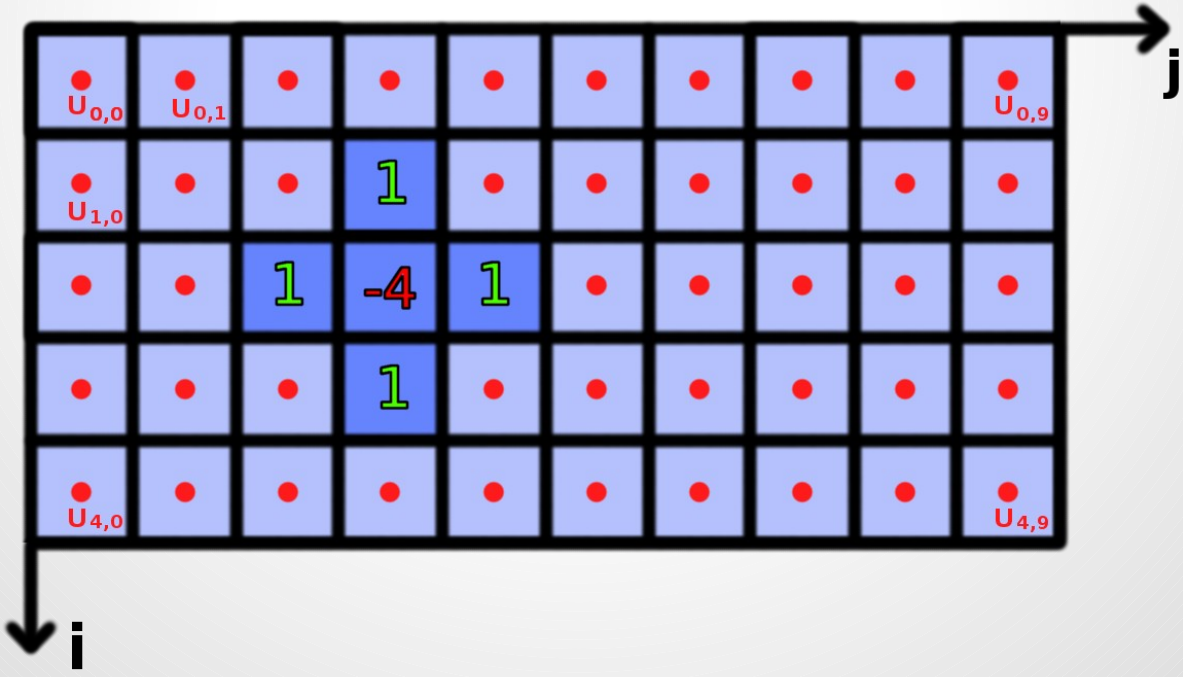
Résolution explicite : $\frac{du(t)}{dt} = f(t,u(t))$ avec $u(t=0) = u_0$

Taylor à l'ordre 1 : $u^{k+1} = u^k + dt\ f(t_k, u^k)$

$$\begin{cases} e^{k+1} = e^k + dt\ F(e^k, r^k) = e^k + dt\ \left(D\Delta e^k - Ke^k(e^k-\alpha_1)(e^k-1) - e^k r^k)\right) \\ r^{k+1} = r^k + dt\ G(e^k, r^k) = r^k + dt\ \left(\varepsilon + \mu_1 \frac{r^k}{1+\mu_2}\right)\left(-r^k - Ke(e^k - \alpha_2)\right) \end{cases}$$

Approximation par différence finies

On se place sur $\Omega = [0,1]^2$ pour simplifier et on divise chacune des directions de l'espace en un nombre fini d' intervalles $(x_i, y_j) = (i\,h, j\,h)$ avec $h = \frac{1}{n-1}$ et $0 \leq i, j \leq n-1$



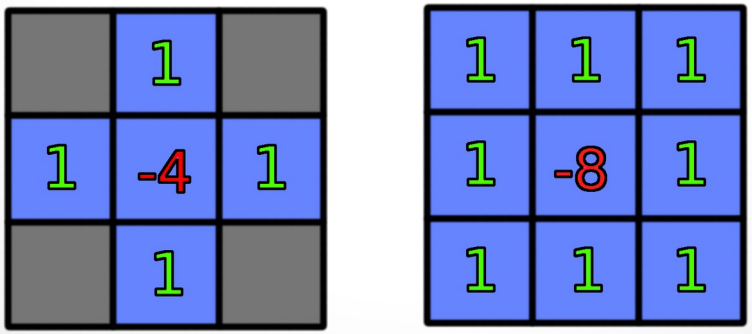
Approximation par différence finies

On peut approximer le laplacien 2D avec Taylor à l'ordre 2:

$$\Delta_h u = \frac{1}{h^2}\left(u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} - 4u_{i,j}\right)$$

$$\Delta_h u = \frac{1}{h^2}\left(u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} + u_{i+1,j+1} + u_{i-1,j+1} + u_{i+1,j+1} + u_{i+1,j-1} - 8u_{i,j}\right)$$

Que l'on peut résumer dans un stencil:

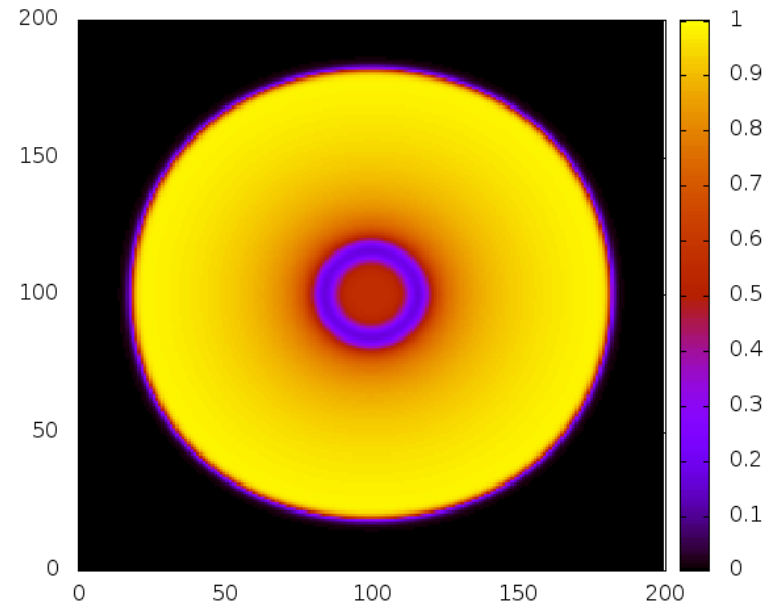


=> Une étape du modèle:

$$\forall (i,j) \in [[0, n-1]] \quad \begin{cases} e_{i,j}^{k+1} = e_{i,j}^{k} + dt\ \left(\frac{D}{h^2}\left(e_{i+1,j}^{k} + e_{i-1,j}^{k} + e_{i,j+1}^{k} + e_{i,j-1}^{k} - 4e_{i,j}^{k}\right) - K e_{i,j}^{k}(e_{i,j}^{k} - \alpha_1)(e_{i,j}^{k} - 1) - e_{i,j}^{k} r_{i,j}^{k})\right) \\ r_{i,j}^{k+1} = r_{i,j}^{k} + dt\ \left(\varepsilon + \mu_1 \frac{r_{i,j}^{k}}{1+\mu_2}\right)\left(-r_{i,j}^{k} - K e_{i,j}^{k}(e_{i,j}^{k} - \alpha_2)\right) \end{cases}$$

Implémentations

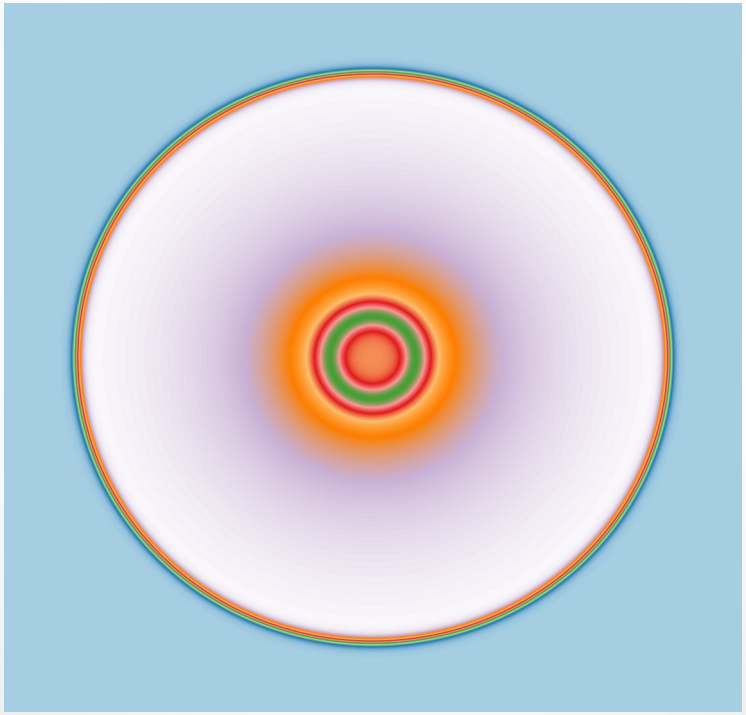
- Premiers tests sur CPU avec sortie en gif (gnuplot)



- Passage en multithread (implémentation de reférence).
- Beaucoup plus rapide que scilab !

Implémentation mono-GPU

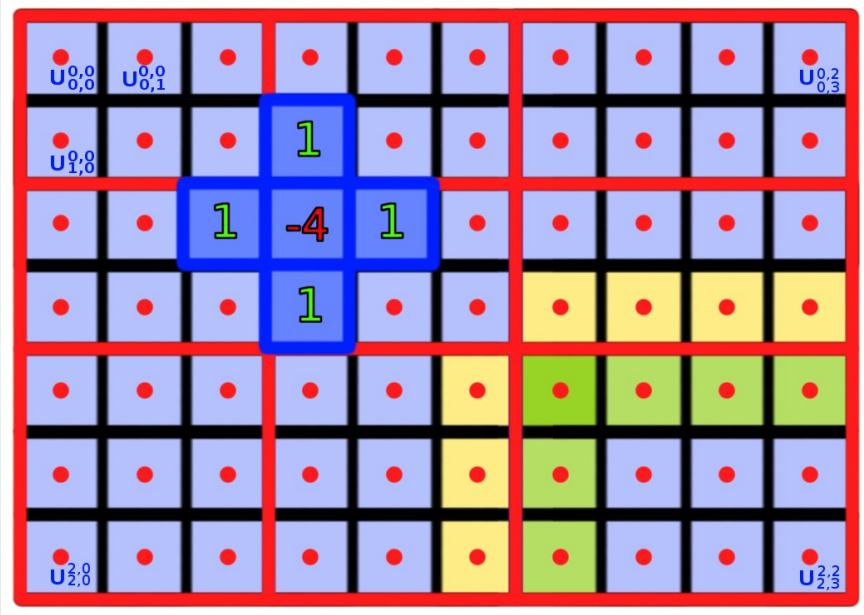
- Passage à OpenCL sur une seule carte



- Gains très significatifs !

Implémentation multi-GPU

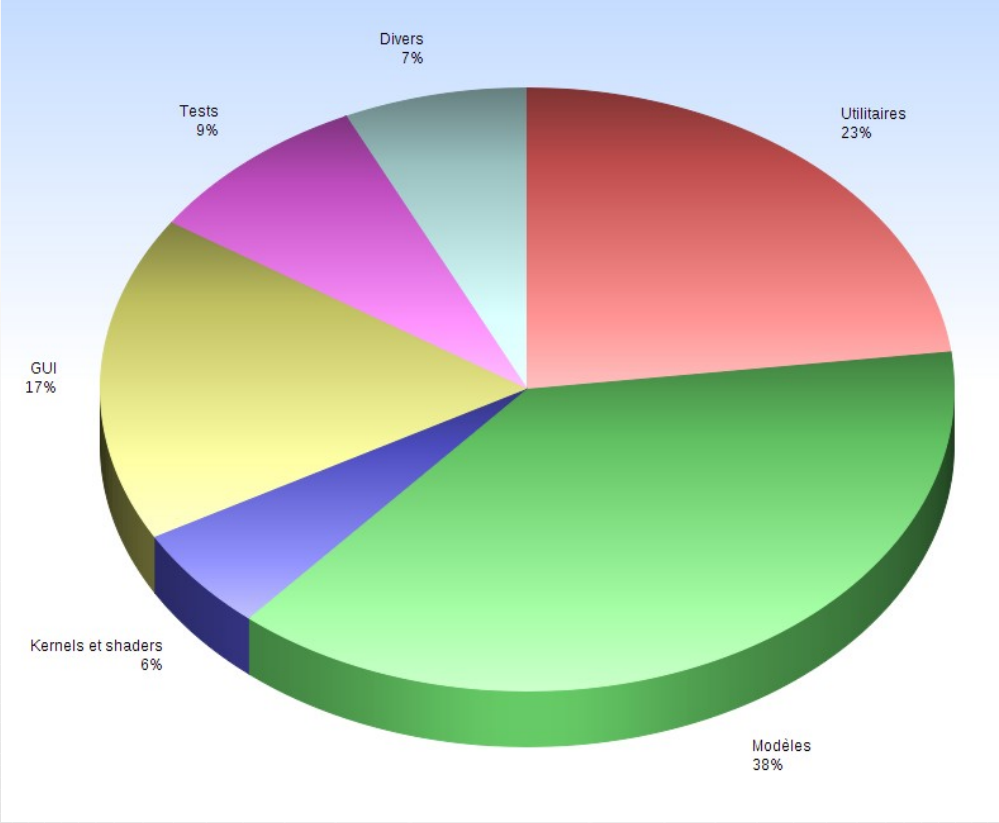
- Découpe de la grille:



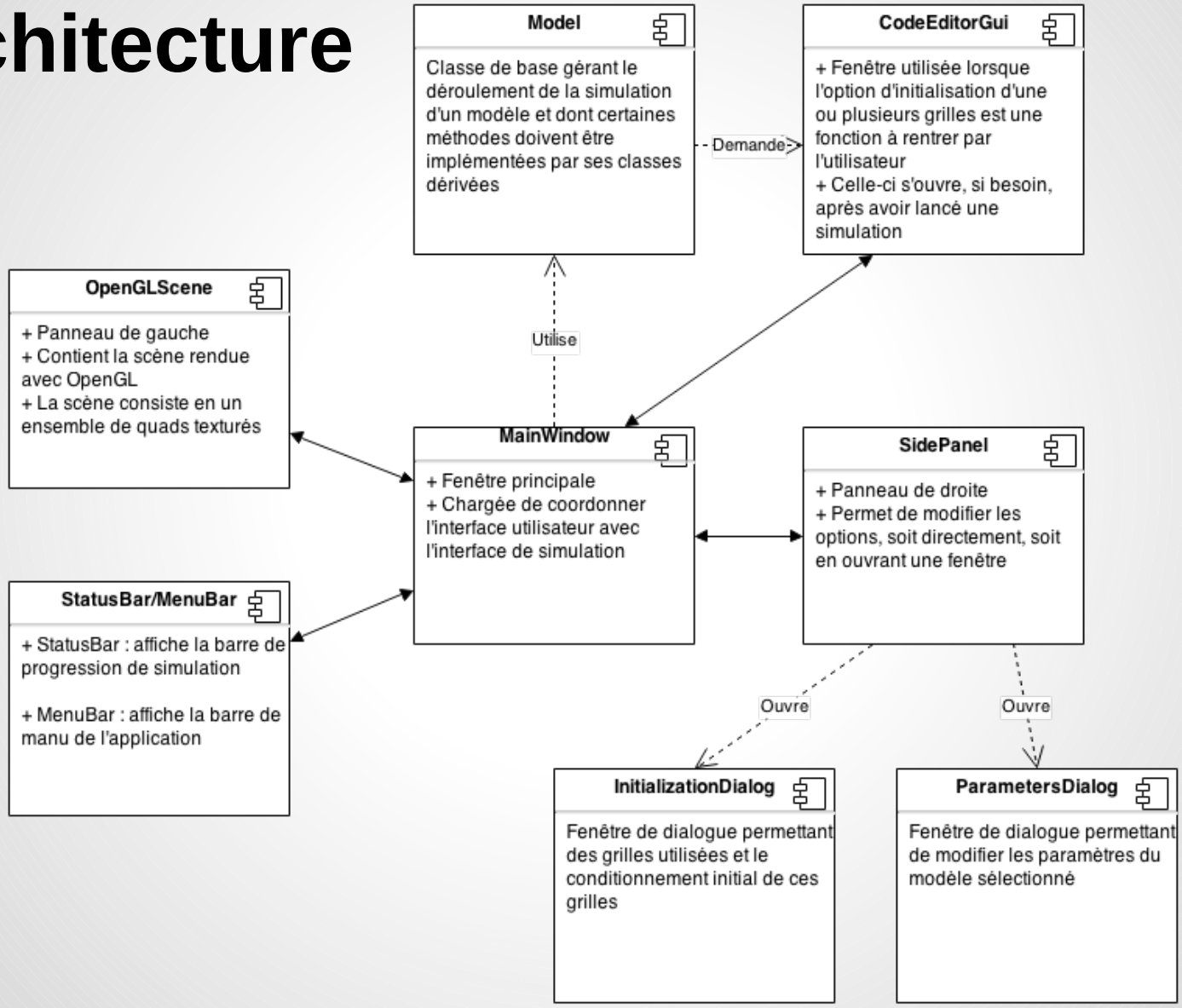
- Nécessité d’échanger les bords entre les sous-grilles.
- Un thread par GPU, synchronisation lourde
- Modèle producteur-consomateur

Implémentation

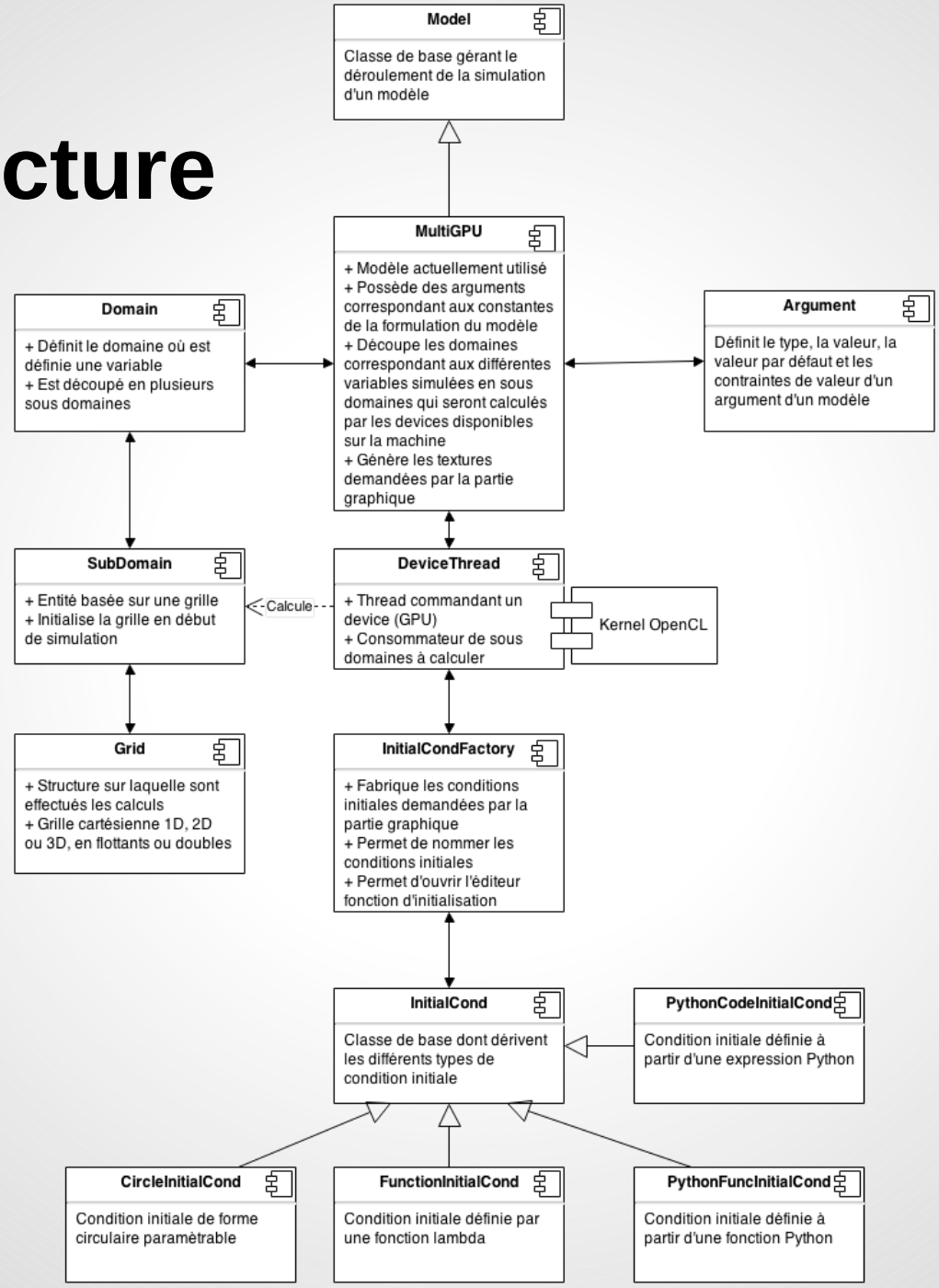
- Architecture modulaire
- Grilles génériques



Architecture



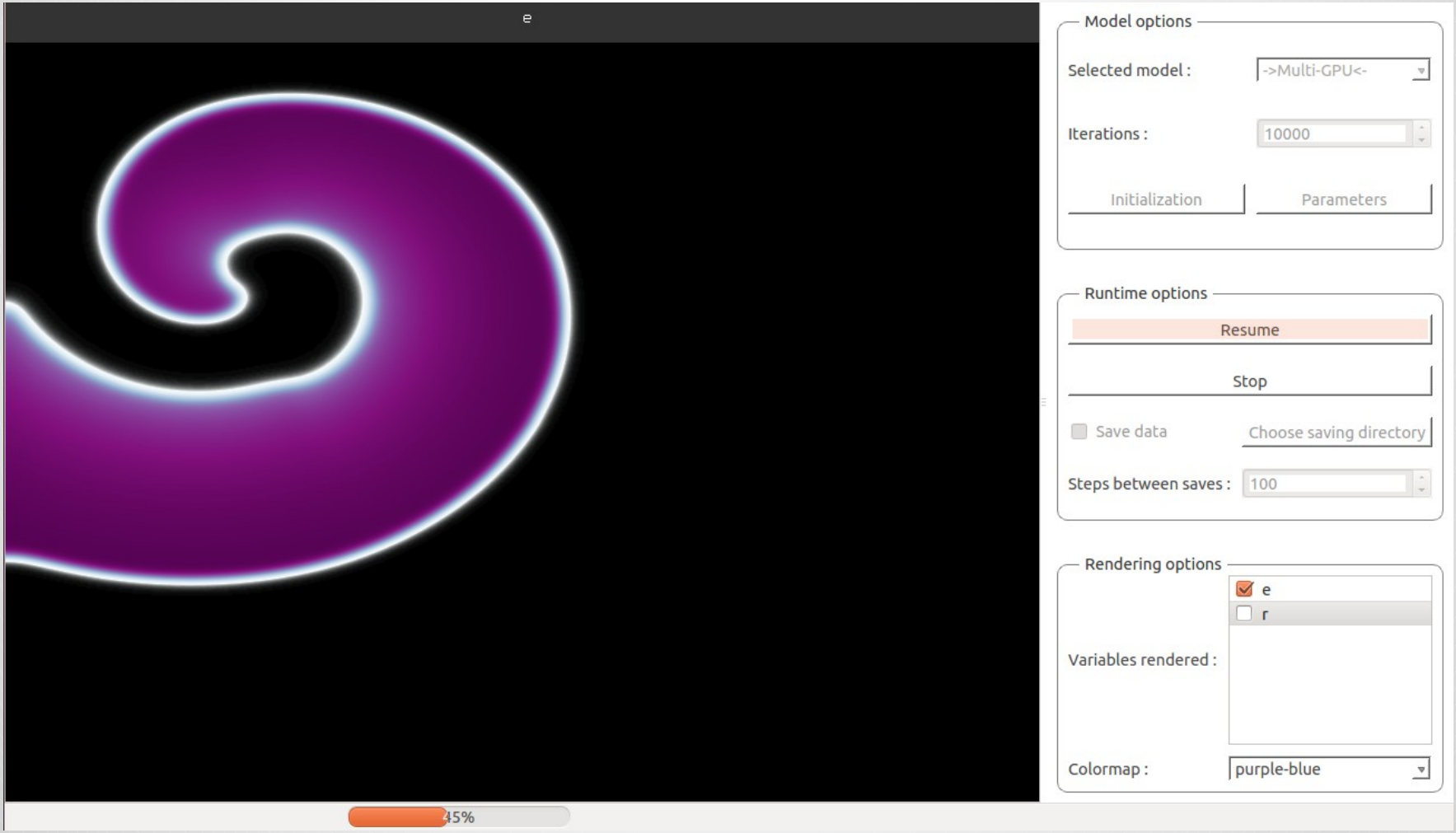
Architecture



Interface utilisateur

- Choix du modèle
- Nombre d’itérations à calculer
- Paramètres du modèle
- Conditions d’initialisation de chaque variable
- Pause de la simulation
- Sauvegarde des données de simulation
- Affichage temps réel
- Choix des variables à afficher

Démonstration



Résultats

- Environnement multi-GPU (deux GTX770 4Go 1536 coeurs) et processeur i7-3770K@3.50GHz
- 1000 pas de simulation

W	H	CPU -O2 multithread (ms)	Mono GPU (ms)	gains/CPU	Multi GPU (ms)	gains/CPU
128	128	3916	67	5 840%	1372	290%
256	256	12 360	136	9 090%	1 489	831%
512	512	71 531	452	15 825%	1 404	5 095%
1024	1024	274 602	1 213	22 638%	2 436	11 273%
2048	2048	1 103 000	4 154	26 553%	8 585	12 848%
4096	4096	3 454 700	14 329	24 110%	30 891	11 183%
8192	8192	-	55 844	-	102 907	-

- Gains impressionnants: jusqu’à 266x en mono-GPU
- Le multi-GPU se paye cher (synchronisation, partage des bords => passage par le CPU)

Perspectives d’améliorations

- Mesures de performance sur la machine de référence et optimisations si nécessaire
- Implémentation d’autres modèles
- Implémentation d’une autre méthode de résolution
- Choix de la coupe à visualiser avec une simulation en 3D

Conclusion