


AMS X02:
Task-based adaptive
multiresolution for time-space
multi-scale reaction-diffusion
systems on multi-core
architectures

Le cadre mathématique


Equation de diffusion-réaction:

$$\left. \begin{aligned} \frac{\partial u_i}{\partial t}(x, t) - \boxed{\operatorname{div}(\varepsilon_i(x) \operatorname{grad} u_i(x, t))} &= \boxed{f_i(u(x, t))}, & x \in \Omega \subset \mathbb{R}^d, t > 0, \\ u_i(x, 0) &= u_i^0(x), & x \in \Omega; \end{aligned} \right\}$$

$u = (u_1, \dots, u_m)^t$ $f(u) = (f_1(u), \dots, f_m(u))^t$



Terme de diffusion de
conductivité ε_i



Terme de
réaction

On prendra une condition de bord de Neumann homogène : $\frac{\partial u_i}{\partial \nu} = 0$ in $\partial\Omega$

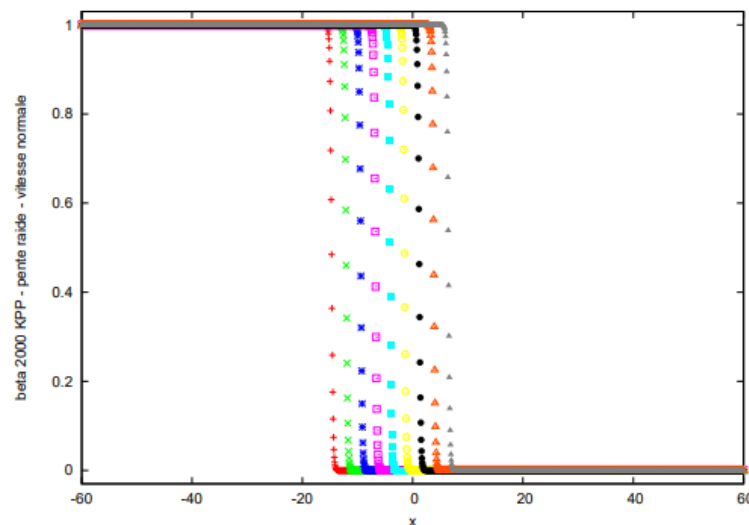
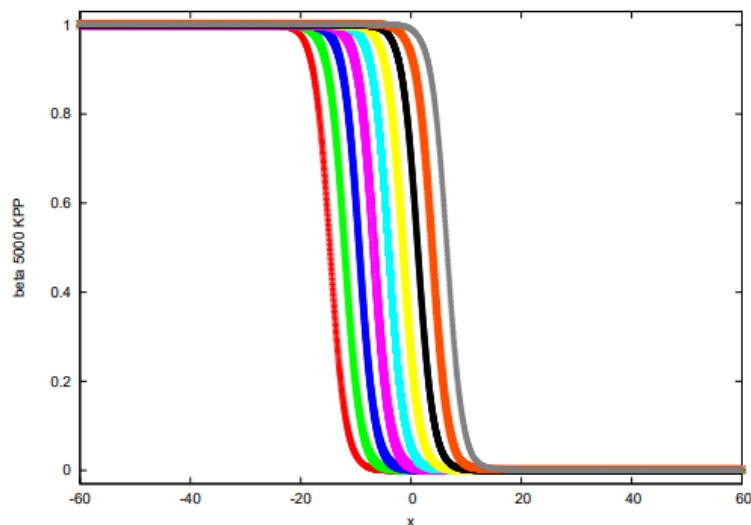
**Equation très générale prenant place dans
d'innombrable cas d'intérêt.**

Le cadre mathématique

- Equation de Nagumo:

$$\frac{\partial u}{\partial t} - \operatorname{div}(\varepsilon \nabla u) = ku^2(1 - u)$$

On prend ici $k = 10$, $\varepsilon = 0.1$ et $0 \leq u^0_1(x) \leq 1$. C'est une équation bistable, avec une solution en onde progressive et un fort gradient : $\|\nabla u\|_{L^\infty} \sim \frac{k}{\varepsilon} \Leftrightarrow$ besoin de multi-résolution adaptative



Implémentation peu coûteuse et peu de raideur

Le cadre mathématique

- Belusov-Zhabotinsky réaction:

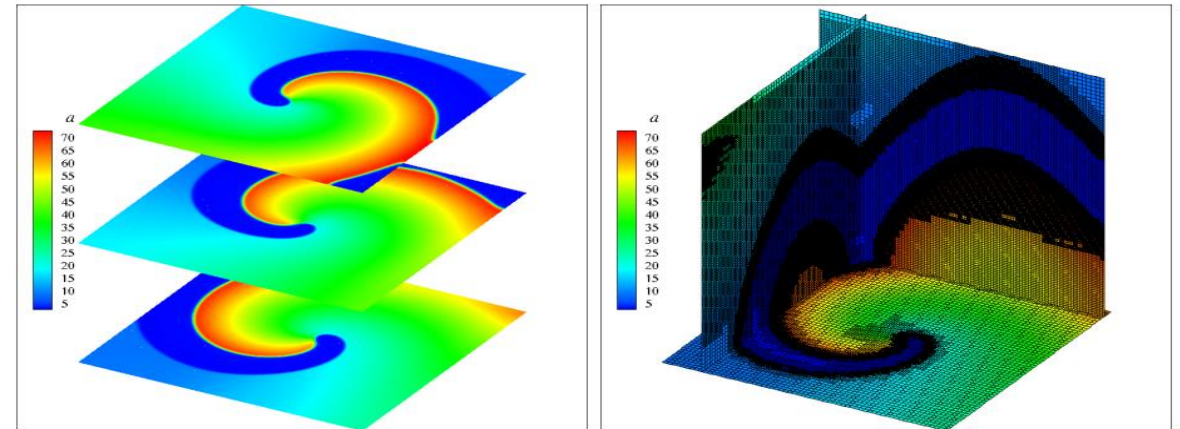
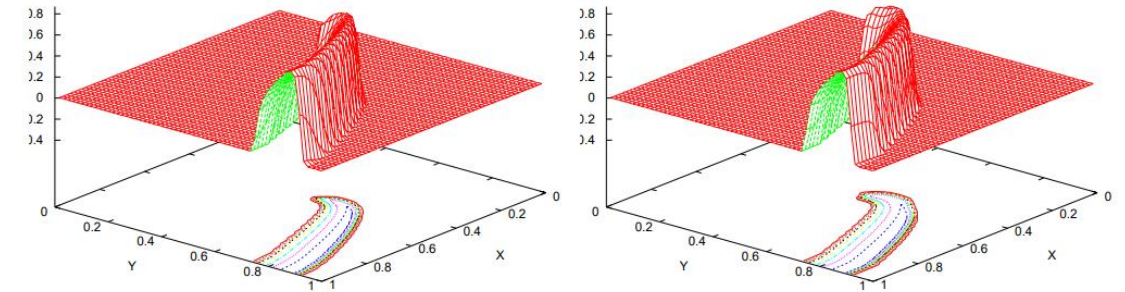
Système de 3 équations couplés:

$$\begin{cases} \frac{\partial u_1}{\partial t} - 2.5 \cdot 10^{-3} \Delta u_1 = 10^5 (-0.02 u_1 - u_1 u_2 + 1.6 u_3) \\ \frac{\partial u_2}{\partial t} - 2.5 \cdot 10^{-3} \Delta u_2 = 10^2 (u_2 - u_2^2 - u_1 (u_2 - 0.02)) \\ \frac{\partial u_3}{\partial t} - 1.5 \cdot 10^{-3} \Delta u_3 = u_2 - u_3 \end{cases}$$

Equilibré

- Stroke équations.

Système de 21 équations hautement raides.



Cas extrême d'une implémentation
chère et très raide!

Le cadre informatique

- Les équations de ce type peuvent présenter de la raideur (stiffness). Par définition, cette propriété caractérise les systèmes coûteux à simuler.
- Il n'est pas envisageable de réaliser le calcul numérique avec la même précision sur l'ensemble du domaine.
- Une grille cartésienne uniforme n'est pas adapté pour la résolution de ce genre de problème, car la raideur est localisé. Un tel choix multiplierait le temps de calcul inutilement.
- Le splitting, le MultiResolution Analysis ou l'Adaptative Mesh Refinement sont d'autant de moyens pour remédier à ce problème.
- Le code MBARETE met déjà en place le MRA-splitting : Cependant, **pas de parallélisme**.

Le papier est sur l'introduction de nouvelles structures de données pour l'implémentation parallèle de MRA-splitting

Le cadre informatique

- Deux processeurs vont être utiles pour les simulations numériques.
- Haswell est la micro-architecture de processeurs x86-64 d'Intel qui succède à la micro-architecture Sandy Bridge.
 - 20 cœurs et 40 threads
- Xeon Phi (MIC) 5110P. Intel Many Integrated Core Architecture ou Intel MIC est une architecture multiprocesseur développée par Intel intégrant des travaux antérieurs
 - 60 cœurs et 240 threads

Important pour calculer l'efficacité du parallélisme!

Le cadre numérique

- Aussi puissant soit-il, l'ordinateur ne peut pas trouver de solution dans un espace de dimension infinie. Soit V l'espace dans lequel le problème de diffusion réaction est bien posé au sens de Hadamard. De le cas d'une approximation conforme, on prend $V_h \subset V$, $\dim(V_h) < \infty$.

Dans ce cadre, on arrive à l'ODE :

$$\begin{cases} \frac{dU(t)}{dt} = A_\varepsilon U(t) + F(U(t)) \\ U(0) = U_0 \end{cases}$$

De cette manière, étant dans un espace de dimension finie, le problème est immédiatement bien posé, modulo des hypothèses de régularités tout à fait raisonnable sur F .

La question est maintenant de savoir comment trouver la solution le plus rapidement possible.

Le splitting d'opérateur

- Plutôt que de calculer l'opérateur d'évolution temporel : $U(t) = K(t, t_0)U(t_0)$, on va se ramener, au prix de l'erreur de l'approximation, à deux systèmes découplés:

$$\begin{cases} \frac{dU(t)}{dt} = A_\varepsilon U(t) + F(U(t)) \\ U(0) = U_0 \end{cases}$$

$$\begin{cases} \frac{dV}{dt} = A_\varepsilon V \\ V(0) = V_0 \end{cases} \quad \text{ROCK4 methode}$$

$$\text{RADAU5 methode} \begin{cases} \frac{dW}{dt} = F(W) \\ W(0) = W_0 \end{cases}$$

En notant $D_{\Delta t}$ et $R_{\Delta t}$ les opérateurs d'évolutions respectifs du problème de diffusion et réaction, on a alors le schéma de Strang du second ordre : $U^{n+1} = R_{\Delta t/2} \circ D_{\Delta t} \circ R_{\Delta t/2} U^n$

On sait que l'erreur L^2 : $\left\| R_{\frac{\Delta t}{2}} \circ D_{\Delta t} \circ R_{\frac{\Delta t}{2}} - K(t, t_0) \right\|_{L^2}$ est en $O(t^3)$

Multi Resolution Analysis

- La décomposition multi-échelle est basée sur la théorie des ondelettes.

Un signal peut se décomposer sous la forme suivante :

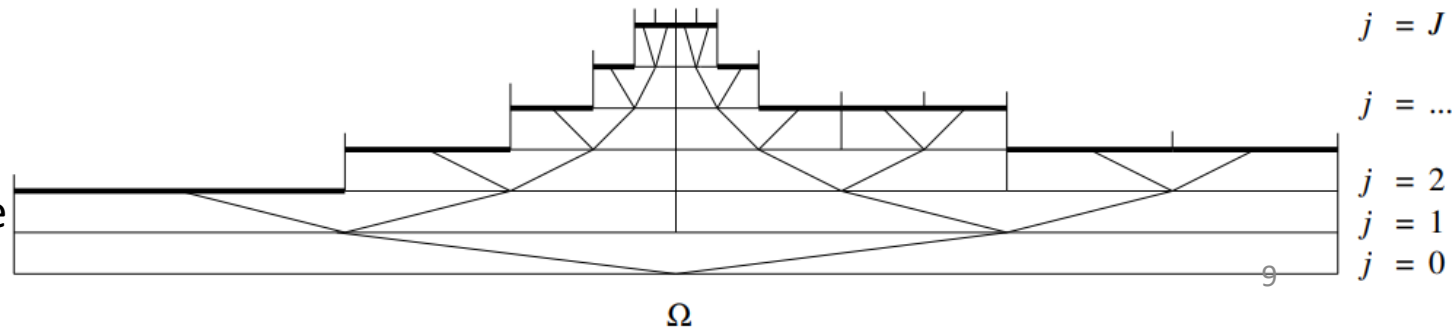
$$f = \sum_k c_{j_0,k} \varphi_{j_0,k} + \sum_{j \geq j_0} \sum_k d_{j,k} \psi_{j,k}$$

Pour changer de niveau de grille, on dispose de l'opérateur de **projection** et de **prédiction**.

$$P_{j-1}^j U^{j-1} = |\Omega_\gamma| \sum_{|\mu|=|\gamma|+1, \Omega_\mu \subset \Omega_\gamma} |\Omega_\mu| u_\mu \quad \text{projection}$$

$$\hat{u}_{j+1,2k} = u_{j,k} + \frac{1}{8}(u_{j,k-1} - u_{j,k+1}), \quad \hat{u}_{j+1,2k+1} = u_{j,k} + \frac{1}{8}(u_{j,k+1} - u_{j,k-1}) \quad \text{prédiction}$$

Structure d'arbre gradué dans le cas d'une partition dyadique



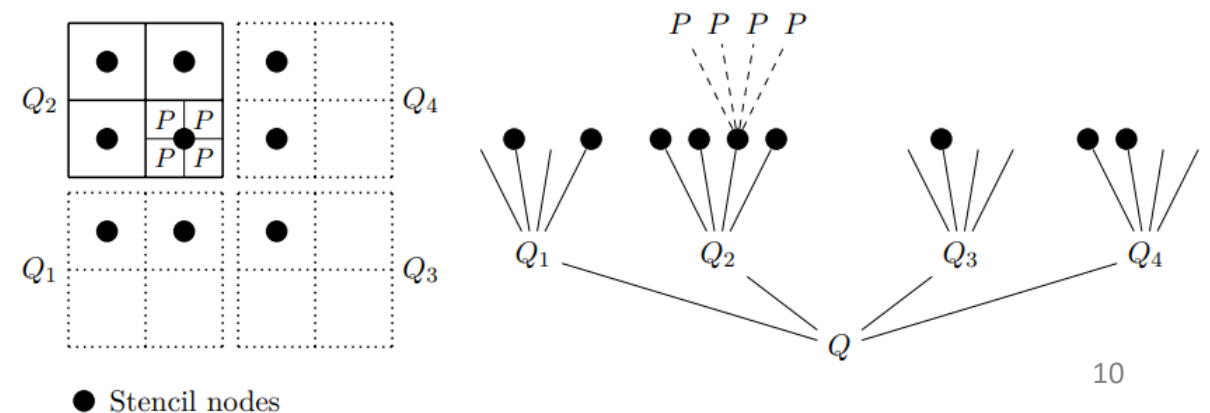
Parallélisme des tâches

- La multi résolution est bien adapté pour le parallélisme des tâches, de par la nature des opérations mis en jeu.
- Une librairie adapté pour ce genre d'approche est la librairie Intel Threading Building Blocks (TBB).

“Cet outil développé en C++ permet d'abstraire au maximum les détails complexes de la programmation sur microprocesseur multicœur. Ainsi un développeur n'a plus à se soucier d'écrire son code pour les threads POSIX ou pour les threads Windows car c'est TBB qui s'occupe de tous les détails spécifiques.” source : Wikipedia

- L'algorithme MBARETE est un algorithme qui met en place
 - La multi résolution spatiale adaptative
 - Strang splitting opérateur pour le temps

Dans le contexte de la parallélisation,
un des enjeux principal est la localisation
données \Leftrightarrow importance de la structure de donnée



Structure de donnée

- On considère un domaine $\Omega = [0,1]^d$. Avec une partition dyadique, on peut coder les nœuds de l'arbre à l'aide d'un seul entier de 64 bits.
 - Généralisation simple pour le 2D et 3D.
 - Abscisse en $0.s$

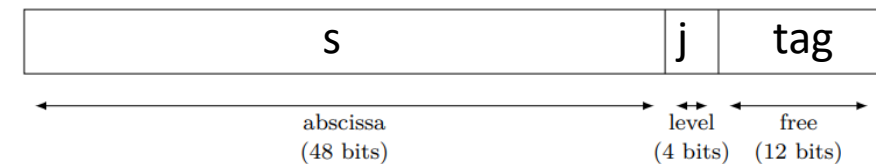
| | | | | | | | |
|-----|-----|-----|-----|-----|-----|-----|-----|
| 0 | | | | 1 | | | |
| 00 | | 01 | | 10 | | 11 | |
| 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 |

- On regroupe les nœuds sous forme de cluster aussi appelé blocks.

Blocks : In computing (specifically data transmission and data storage), a **block**, sometimes called a **physical record**, is a sequence of bytes or bits, usually containing some whole number of records, having a maximum length; a *block size*. Data thus structured are said to be *blocked*. Source : Wikipedia

Le block est une structure efficace au sens où :

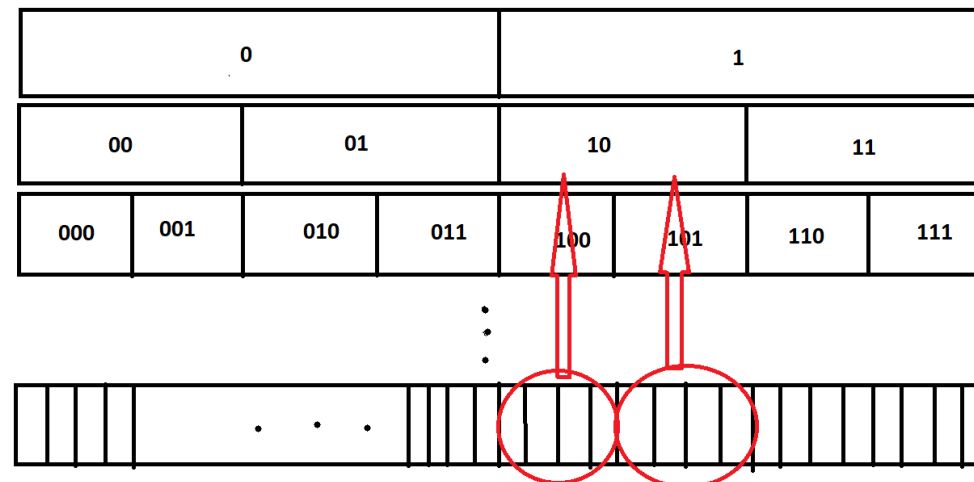
- Fusion et séparation simple
- Permet la recherche par dichotomie (en $O(\log(N_b))$)
- Opérations facilement parallélisable avec TBB.
- Contient tous les nœuds dans un intervalle de la forme $[s_{min}, s_{max}[$



Codage d'un block

Adaptation de la grille

- Le vecteur détail $d_{j,k}$ permet de rendre compte de la régularité d'un signal. Si, à une certaine profondeur J , les détails ne sont pas négligeables, alors il est intéressant de raffiner le maillage. On s'en sert comme des **estimateurs locaux**. Les détails se calculent avec l'opérateur : $\left[P_{j-1}^j \circ P_j^{j-1} \right]$
 - Différence principale avec le raffinement de maillage classique : estimation d'erreur *a priori* et *a posteriori* (difficulté théorique)
- Calcul des détails en deux étapes:
 - Projection : par récursivité, on propage les données de la profondeur j à la profondeur $j - 1$.



Parallélisation de chaque appel récursif

Parallel_for de TBB

Adaptation de la grille

- Calculs des détails en deux étapes:
 - Projection : par récursivité, on propage les données de la profondeur j à la profondeur $j - 1$.
 - Prédiction : décrit plus haut.
- On stocke les détails pour chacun des nœuds. L'adaptation de la grille se fait aussi par parallélisme.
 - Raffinement : On étiquète les nœuds qui ont besoin d'être raffiné (stockage prévue). On les raffine ensuite, en mettant à jour les blocks, quitte à les dédoubler en des blocks temporaires. **On garde cependant la contrainte d'un arbre gradué.**
 - Coarsening : Les feuilles qui n'ont pas été tagué pour raffinement sont à leur tour étiquetées pour être enlevées dans l'arbre. Les nœuds parents seront alors les nouvelles feuilles de l'arbre (sans compromettre la propriété de graduation de l'arbre). On les considère en block par block.

Etiquetage et suppression
parallélisé

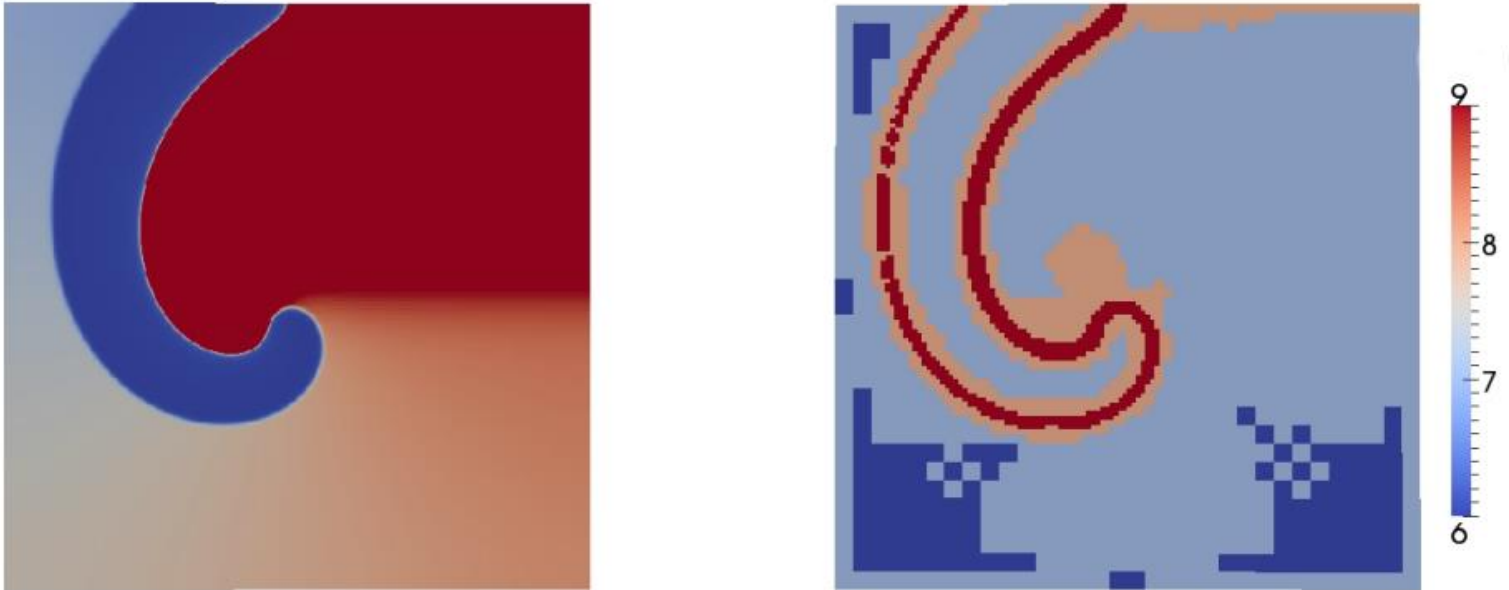
| 0 | | | | 1 | | | |
|-----|-----|-----|-----|-----|-----|-----|-----|
| 00 | | 01 | | 10 | | 11 | |
| 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 |
| | | X | X | X | X | | |

Splitting solveur pour le parallélisme

- Chaque nœud porte une inconnue : la valeur de la solution à cet instant t et à cet endroit. Deux façons de stocker la solution est envisageable :
 - Array of structure : Chaque nœud est traité individuellement, et lui est associé sa propre structure propre, contenant alors toutes les données relatives \Leftrightarrow Plus approprié pour l'évaluation de $F(U)$.
 - Structure of array : Chaque inconnu des nœuds P sont stockés dans un seul et unique array \Leftrightarrow Plus approprié pour l'évaluation de $A_\varepsilon U$ car besoin des nœuds voisins.
- Le solveur de réaction étant plus coûteux, la première solution a été adopté. Sa parallélisation se fait de la manière suivante : grâce à TBB, on parallélise l'évaluation de $F(U)$ pour chacun des block.
- Comme souvent en diffusion, on prendra un solveur ROCK4. Parallélisation possible pour l'évaluation des produits matrices-vecteurs.

Présentation du code

- Le code C++ est nommé Z-code. ROCK4 et RADAU5 ont été réécrit pour être plus adapté à la structure de donnée mis en place et à la parallélisation.



Simulation BZ en dimension 2

Comparaison entre grille cartésienne uniforme et adaptée

- Le même code, modulo certaines implémentations, est exécuté sur une grille cartésienne uniforme et adaptée.

L'indicateur de l'efficacité du code est le taux de compression CR

$$CR = \left(1 - \frac{N}{N_J}\right) \times 100$$

N : Nombre de nœud de la grille adapté au niveau J .

$$N_J = 2^{dJ}$$

| | | J | | |
|--------|----|---|---|---|
| | | 8 | 9 | 10 |
| NAGUMO | MR | 2.51×10^{-3} (1.7) | 3.02×10^{-3} | 4.97×10^{-3} |
| | CM | 1.49×10^{-3} | 8.27×10^{-3} (2.7) | 2.09×10^{-2} (4.2) |
| BZ | MR | 1.14×10^{-2} | 2.51×10^{-2} | 4.06×10^{-2} |
| | CM | 1.31×10^{-2} (1.2) | 4.06×10^{-2} (1.6) | 1.78×10^{-1} (4.4) |
| STROKE | MR | 2.4×10^{-2} | 4.03×10^{-2} | 1.03×10^{-1} |
| | CM | 3.2×10^{-1} (13.3) | 1.21 (30.0) | 4.80 (46.6) |

2D

| | | J | |
|--------|----|---------------------|----------------------|
| | | 8 | 9 |
| NAGUMO | MR | 0.13 | 0.31 |
| | CM | 0.54 (2.5) | 2.47 (4.5) |
| BZ | MR | 0.97 | 5.75 |
| | CM | 3.05 (3.1) | 23.60 (4.1) |
| STROKE | MR | 1.53 | 10.81 |
| | CM | 76.68 (50.0) | 610.50 (56.5) |

3D

Efficacité de la méthode

- On définit naturellement l'efficacité du parallélisme de la manière suivante:

$$s_k = \frac{T_1}{T_k n_k}$$

- Utilisation du multithreading sur

- Haswell : $n_k = \min(k, 20)$
- et Xeon Phi : $n_k = \min(60, k)$

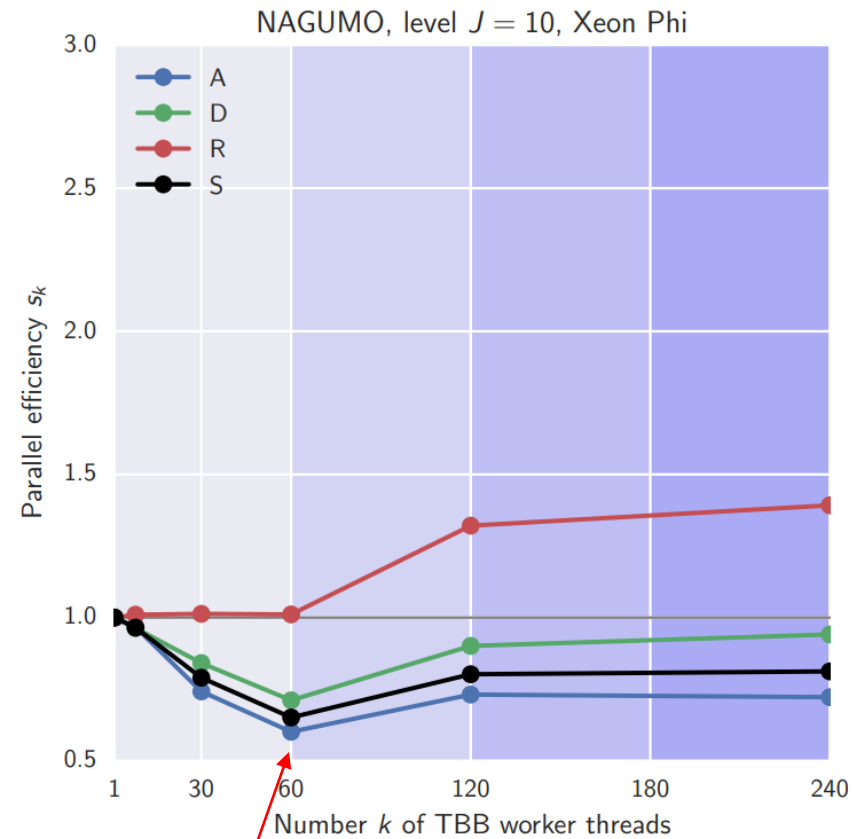
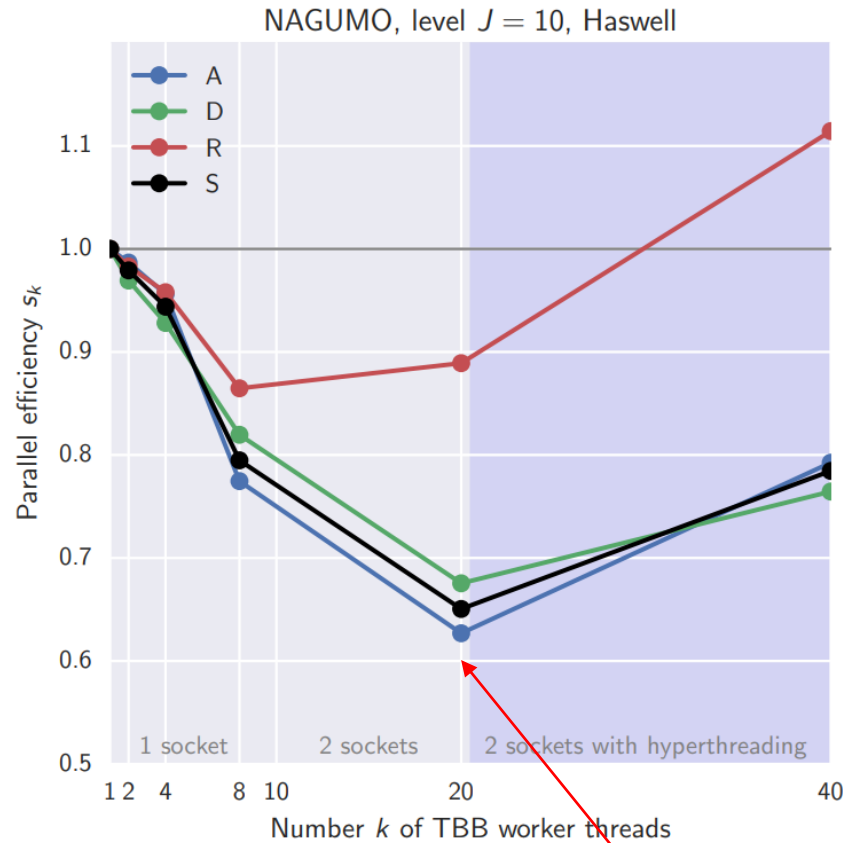
Basé sur le nombre de cœurs car unité de calcul indépendante \Leftrightarrow parallélisme

| | NAGUMO | BZ | STROKE |
|------------------|---------------|--------------|---------------|
| Mesh Adaptation | 68.00 | 19.23 | 7.20 |
| Reaction Solver | 0.92 | 56.23 | 88.89 |
| Diffusion Solver | 31.08 | 24.54 | 3.91 |

Sur le processeur Haswell, J = 10

Pourcentage de temps passé pour chacun des processus.

Efficacité de la méthode



A : Adaptative Mesh
D : Diffusion
R : Reaction
S : Total time Step

Multi-threading

Efficacité de la méthode

- Deux raisons que le parallélisme de diffusion et d'adaptation de maillage soient bien moins efficaces.
 - Sont plus aptes à demander beaucoup de mémoire, et donc de temps de lecture. De plus, la mémoire est partagée entre les cœurs : création d'un bottleneck \Leftrightarrow ralentissement.
 - Parallélisation intrinsèquement plus compliqué que pour la réaction \Leftrightarrow limitation par la loi d'Ahmdal.
- Importance du multi-threading
 - + d'un thread par core. Particulièrement efficace lorsqu'il y a beaucoup d'interruption dans la réalisation d'un thread : cache misses, branches, or bad pipelining.
 - Nécessité de connaître l'architecture profonde du processeur utilisé.

Conclusion

- Cadre mathématique d'une réaction chimique (diffusion + terme source)
- Technique de parallélisation combiné avec parallélisme.
- Présentation de la structure de donnée utilisée.
- Particulièrement adapté pour des problèmes temps-espaces complexes et multi dimensionnel.
- Efficacité du parallélisme reste relatif et très dépendant de l'implémentation informatique.
- Efficacité du multi-threading.

BACKSLIDES

