PRISM
UVSQ

Supélec

Simulation parallèle des systèmes multi-agent et son application au trafic routier

Eugen Dedu

PRiSM UVSQ

http://www.prism.uvsq.fr/~dedu/parssap

le 22 avril 2003

Centre
Charles Hermite

lorraine
conseil régional

Plan & Contributions

- Introduction
- 1. Nouveau modèle de simulation des SMA
- 2. Algorithmique parallèle du modèle
  - 2.0 Parallélisme dans l'implantation
  - 2.1 Percept de la vision
  - 2.2 Percept de la détection des potentiels
- 3. Implantation parallèle portable du modèle (bibliothèque), application au trafic routier
- Conclusions

Avant tout…

- Centré sur le thème :
  - algorithmique parallèle et performances
  - SMA et application (trafic routier)

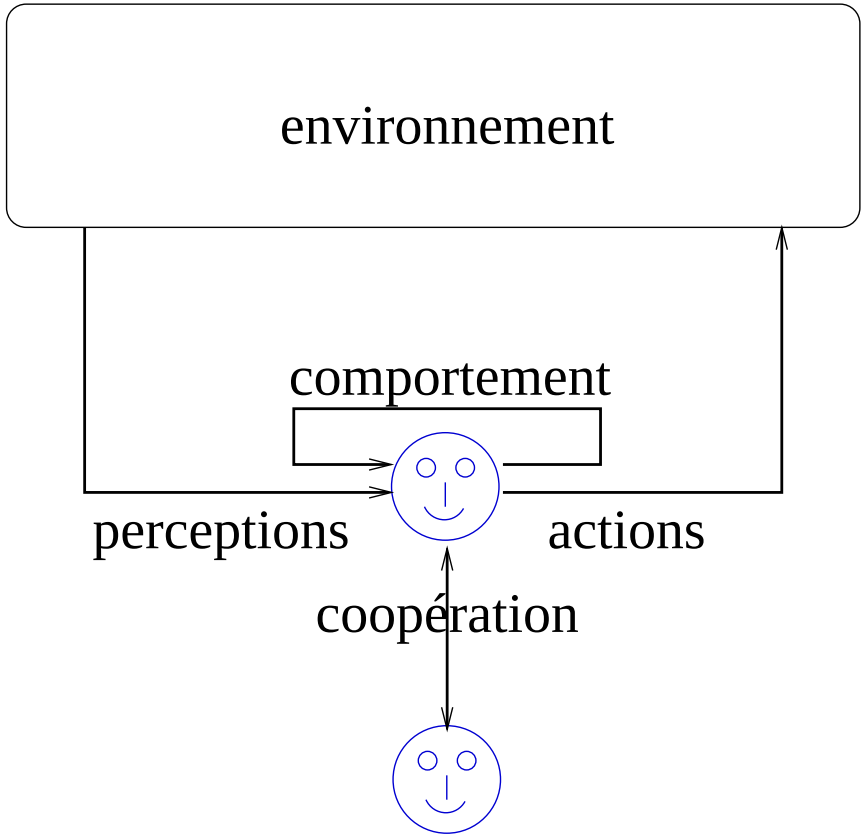
- Choix de la façon :
  - explications
  - informations nouvelles / intéressantes

Introduction : systèmes multi-agent

- Agent :
  - entité autonome
  - a un but
  - vit dans un environnement
  - communique avec l'environnement
  - agents situés

- Domaines d'applications :
  - société : populations urbaines, trafic routier
  - biologie, étude des comportements : populations de fourmis, d'oiseaux

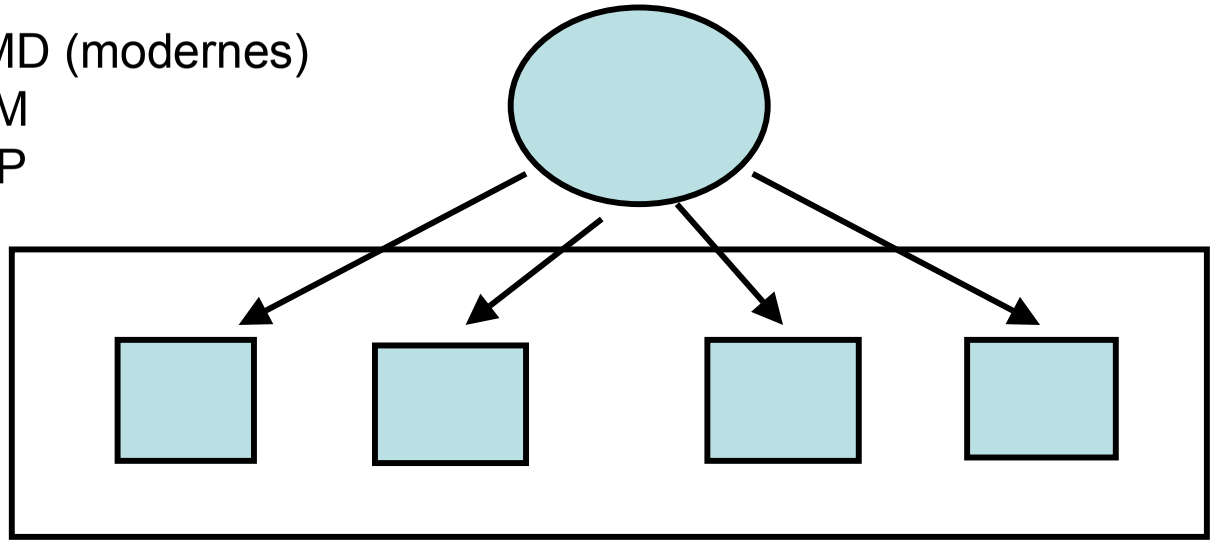
- Jacques Ferber (pionnier)



Introduction : parallélisme

Pour nous : exécution d'une application sur plusieurs processeurs (accélération)

- MIMD (modernes)
- DSM
- MVP



Application

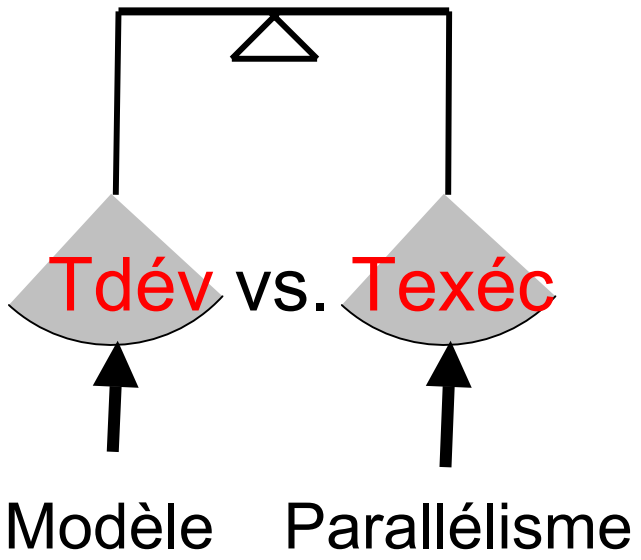
Exemple :
Ordinateur à
4 processeurs

- Origin 2000, 64 processeurs, 4 Mo cache, 24 Go mémoire
- Sun 450, 4 processeurs, 4 Mo cache, 1 Go mémoire
- IA-64 Itanium SMP, 4 processeurs
- PCs SMP bon marché

- OpenMP
- multi-threading
- envoi de messages
- multi-processus

Motivations

- comportement des agents
- diverses fonctionnalités
- gestion de l'environnement
- gestion de la simultanéité
- gestion des structures des entités



Exemples de simulateurs de SMA

- Pengi [Ferber et al., 1991]
  - **simple**
  - **séquentiel**

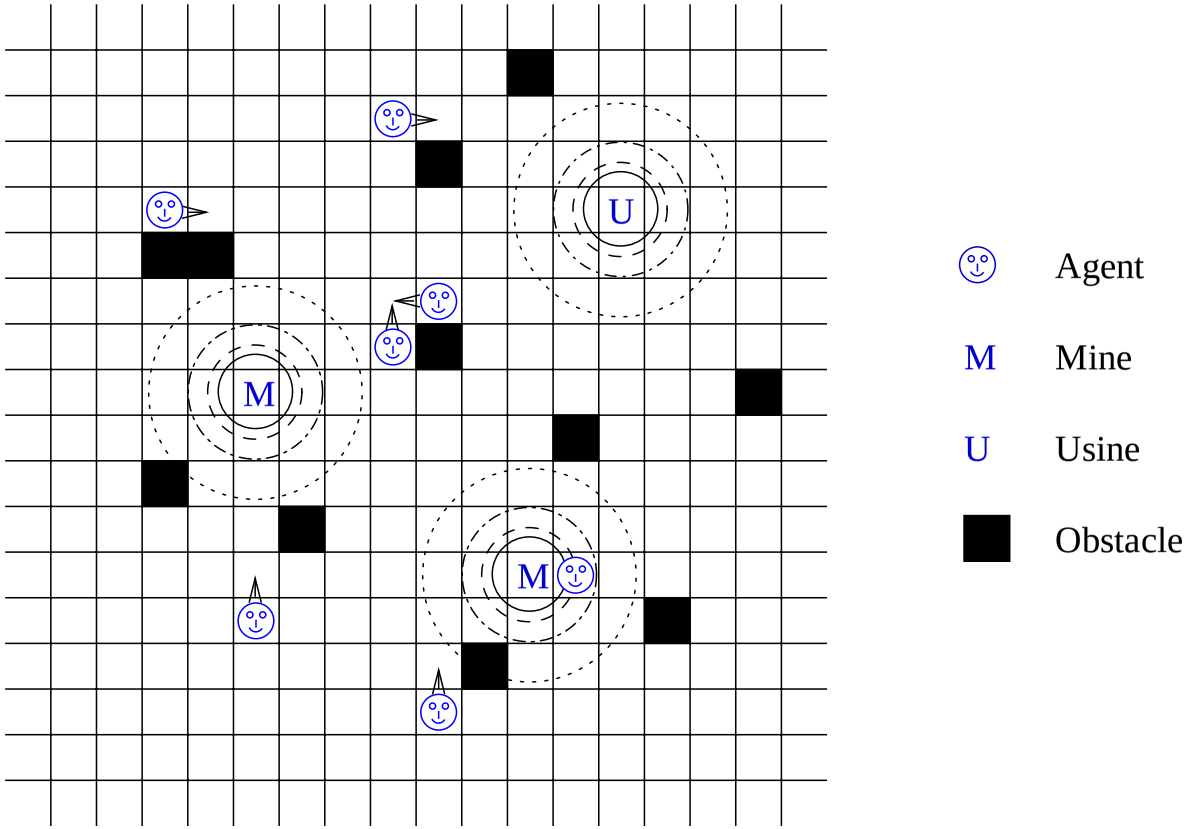
- BioLand [Werner & Dyer, 1994]
  - **massivement parallèle, beaucoup d'agents**
  - **application riche en fonctionnalités**
    - **réseaux de neurones**
  - **sans obstacles => algorithmique plus simple**
  - **peu flexible**
  - **machines SIMD (CM-2)**

- PIOMAS [Bouzid, 2001]
  - **parallèle, basé sur ParCeL-3**
  - **précis, modélisation des incertitudes dans perceptions et actions**
  - **peu d'agents**

1. Modèle : exemple

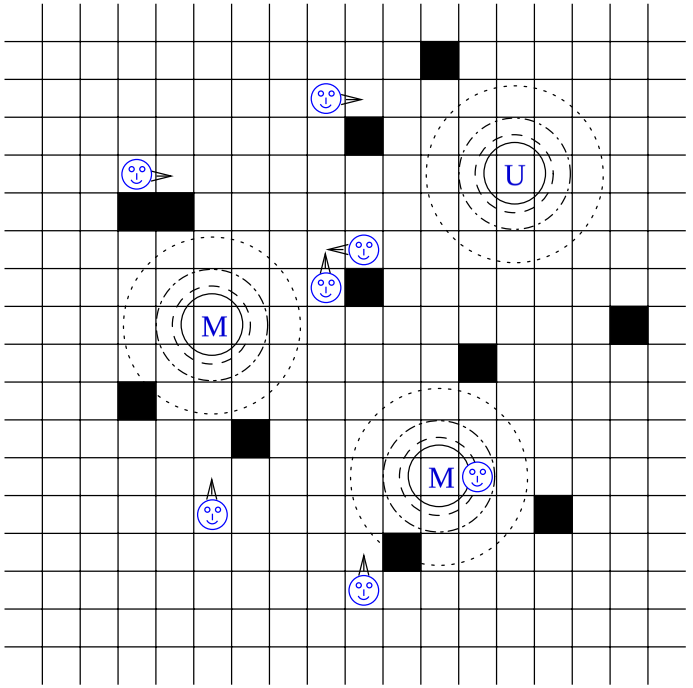
ParSSAP (Parallel Simulator of Situated Agent Populations)

- environnement
- ressources
- agents
- arbitre



1. Modèle : généralités

- Environnement
  - **discret**
  - **obstacles**
- Ressources
  - **propagent des potentiels décroissants**
  - **contiennent des objets**
  - **champs dynamiques de potentiel**
  - **plusieurs types**
- Agents
  - **entités mobiles**
  - **comportement**
  - **mémoire**
  - **perceptions locales : vision, odeur**
  - **actions : mouvement, dépôt/prise d'objets**
  - **création et destruction dynamique**
- Arbitre (virtuel)
  - **loi du système**
  - **résout les conflits spatiaux entre les agents**



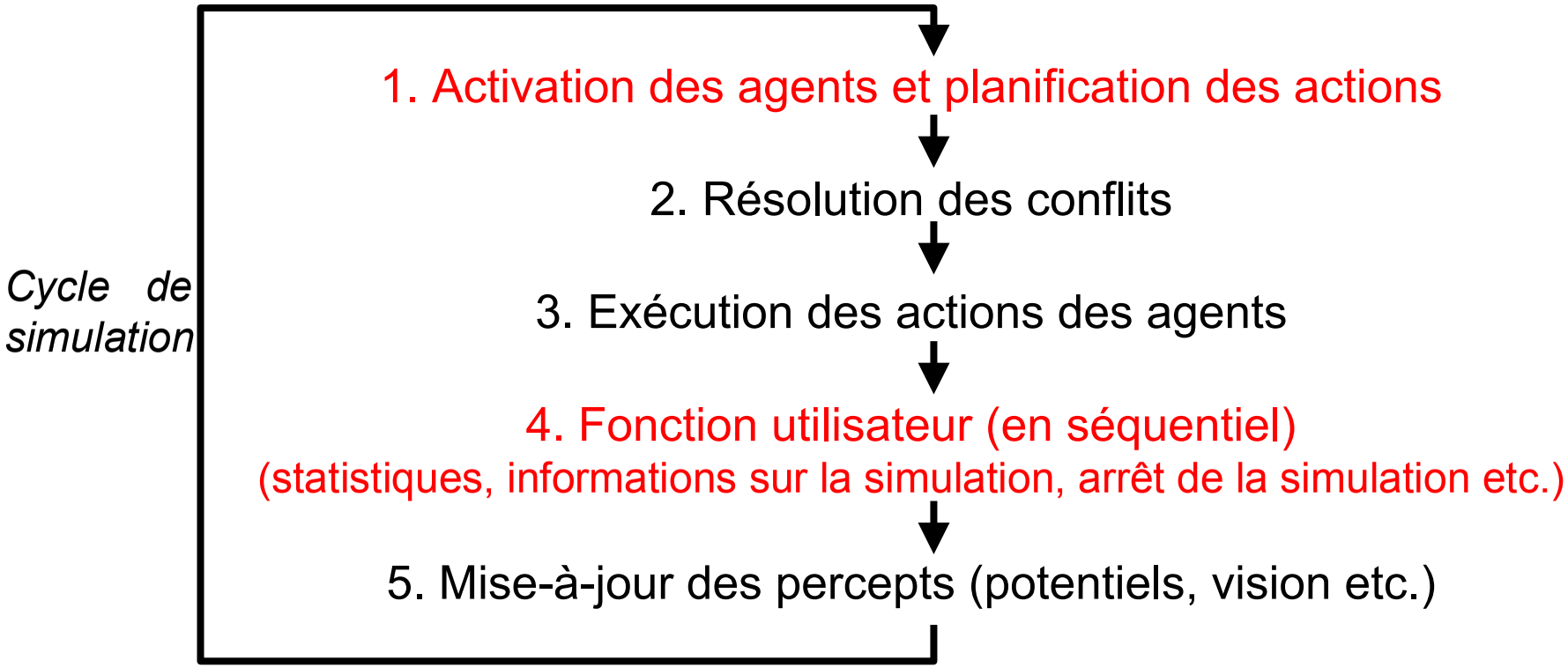
Sauvegardes
- **statistiques**
- **mouvement des agents**

Unification des entités => travaux futurs

Tous les principes existent pour augmenter la généricité

1. Modèle : moteur d'exécution

- Basé sur une discrétisation du temps
- Macroscopiquement synchrone, microscopiquement asynchrone



2.0 Algorithmique parallèle : parallélisme dans l'implantation

- Parallélisme interne, mais caché à l'utilisateur
- Décomposition en domaines rectangulaires
  - perspectives : si nécéssaire, décomposition explicite prenant en compte les zones à grande charge
- Aucune surcharge due au parallélisme, sauf :
  - fonction utilisateur (en séquentiel) <= transparence au parallélisme
  - gestion des frontières <= intrinsèque au parallélisme
  - déséquilibrage de charge => travaux futurs

- Langage C, threads POSIX et Irix, fonctionne aussi sur mono-processeur

- Mécanismes de génération de nombres aléatoires ad hoc => reproductibilité complète des simulations, même avec nombre différent de processeurs

2. Algorithmique parallèle des percepts des agents

- 2.1 Percept de la vision

  But : création des champs de visibilité

  - **les cases qu'un agent peut voir**
  - **rayon de vision**
  - **empêché par les obstacles, perception lointaine**

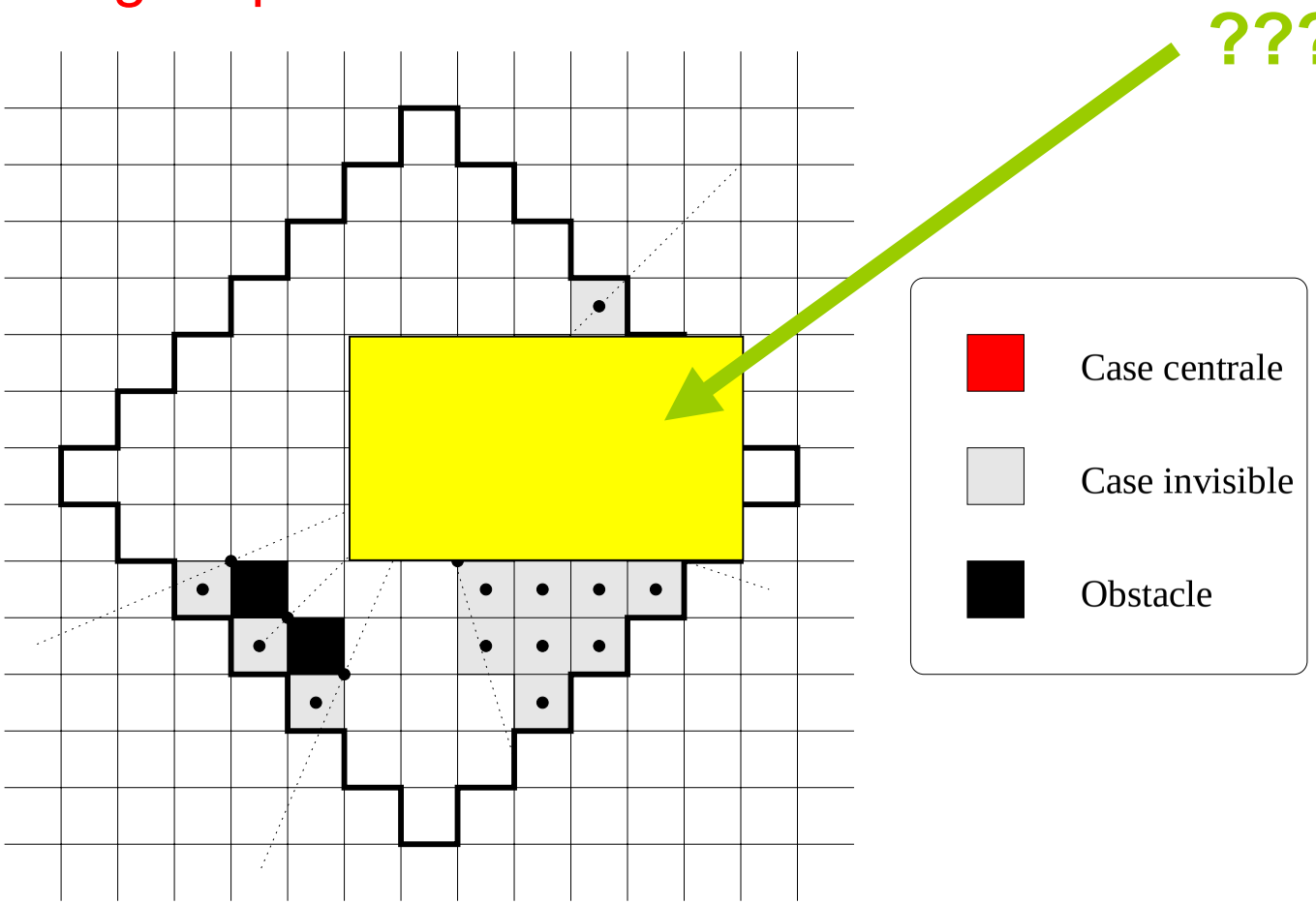
- 2.2 Percept de la détection des potentiels

  But : création des champs de potentiel générés par les ressources

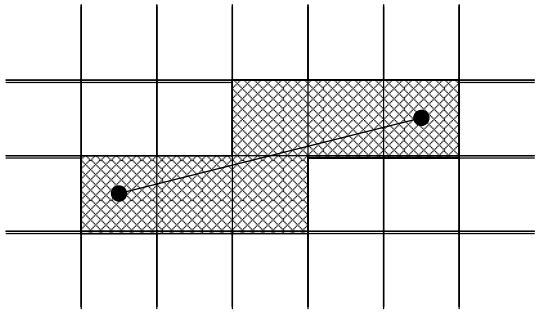
  - **début : le potentiel de la ressource**
  - **propagation par vagues**
  - **contourne les obstacles, perception de proximité seulement**

2.1 Vision : exemple de champ

Ce qu'un agent peut voir

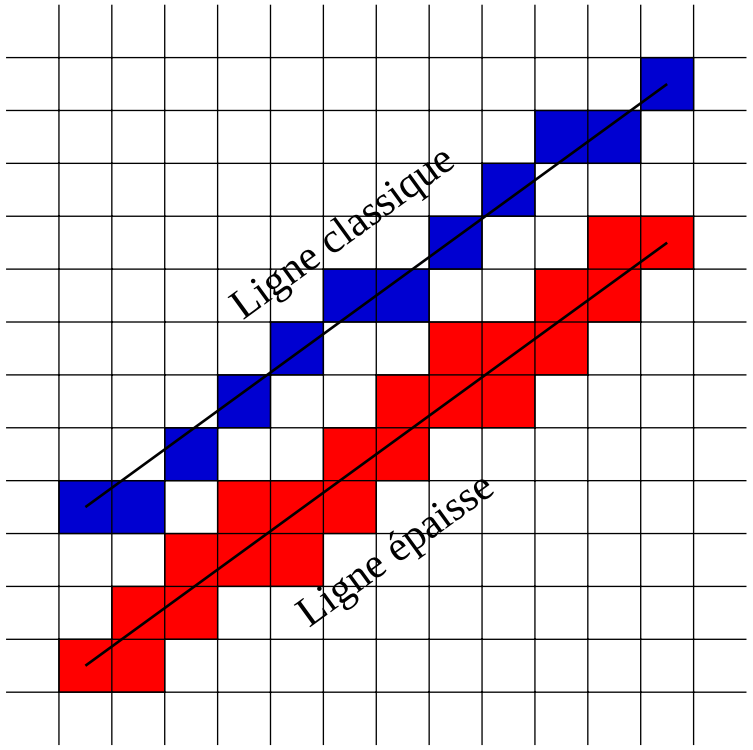


2.1 Vision : nouvel algorithme de lignes épaisses



ligne classique : ne considère pas l'obstacle

ligne épaisse : considère l'obstacle



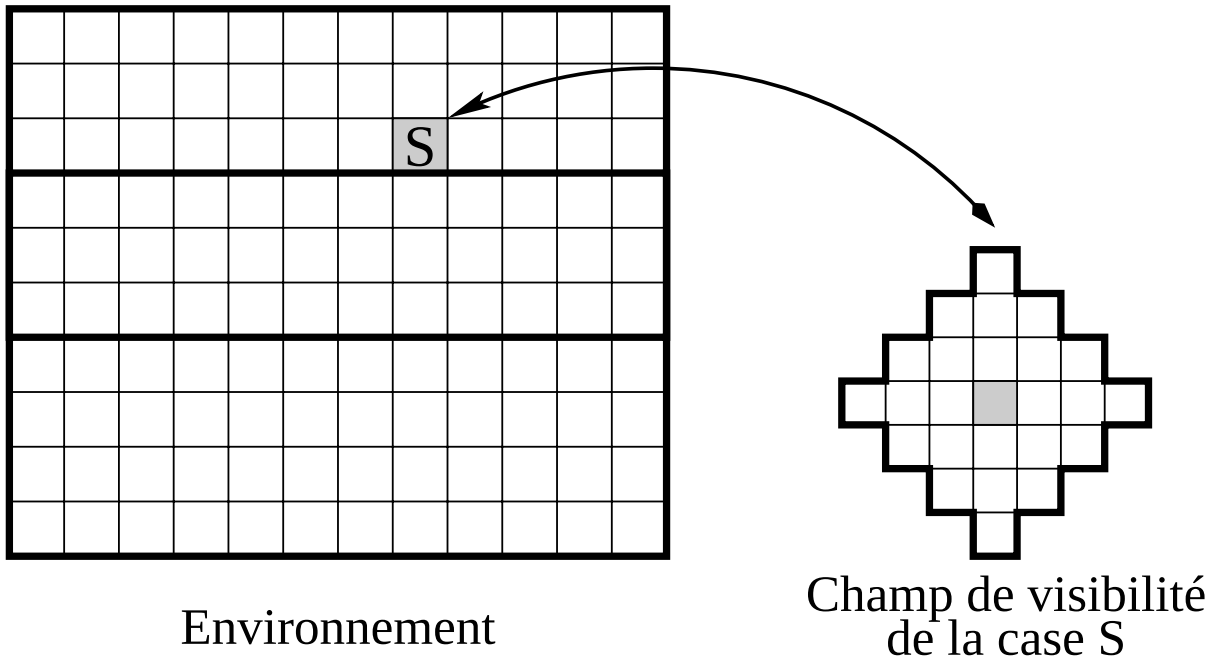
- TOUS les points que la ligne idéale intersecte

- basé sur l'algorithme de Bresenham
- prend en compte l'erreur précédente

2.1 Vision : parallélisme

- Calculer les champs de visibilité de chaque case
  - Lecture de la matrice des obstacles
  - Écriture dans la matrice de visibilité

Calcul identique pour toute case => parallélisation par décomposition en domaines

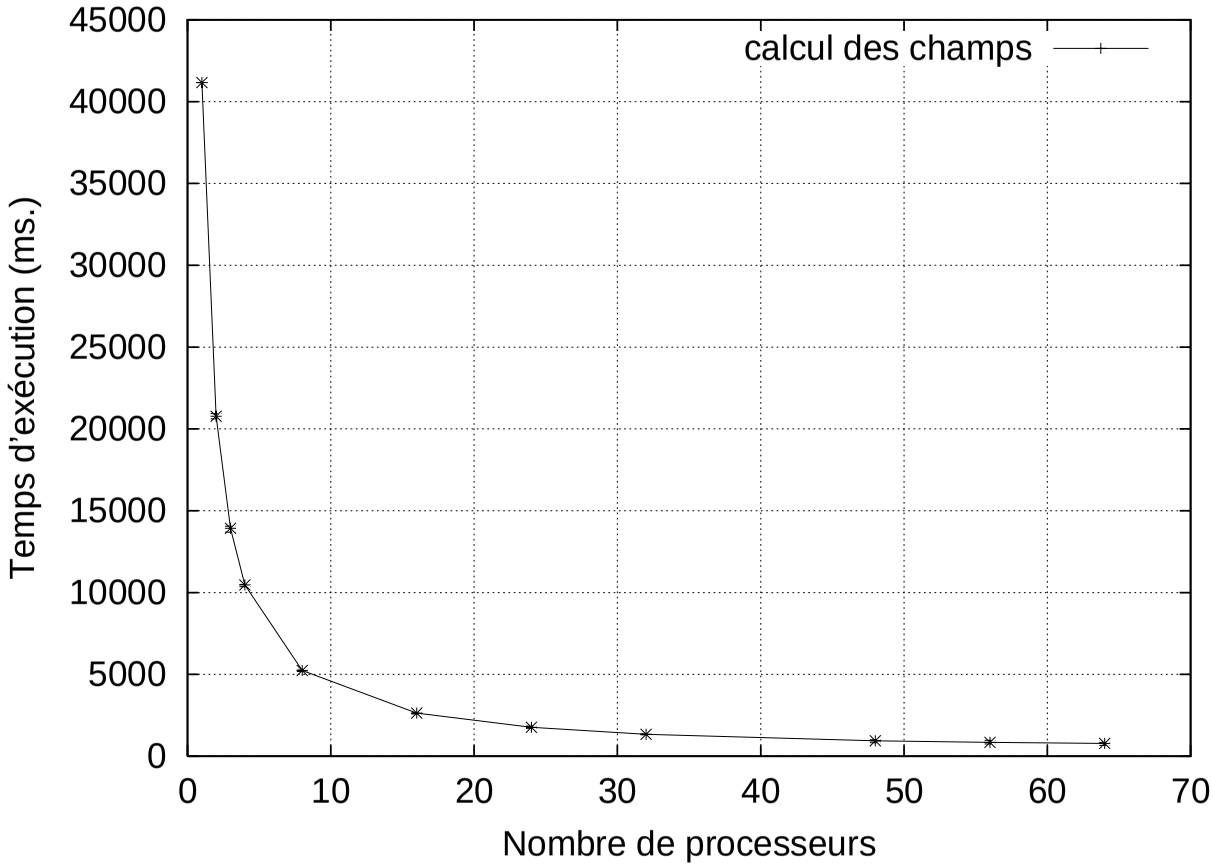


2.1 Vision : performances (1/3)

Mémoire : O $(N(2r_v+1)^2)$

Temps :
1. allocations mémoire
2. calcul des champs

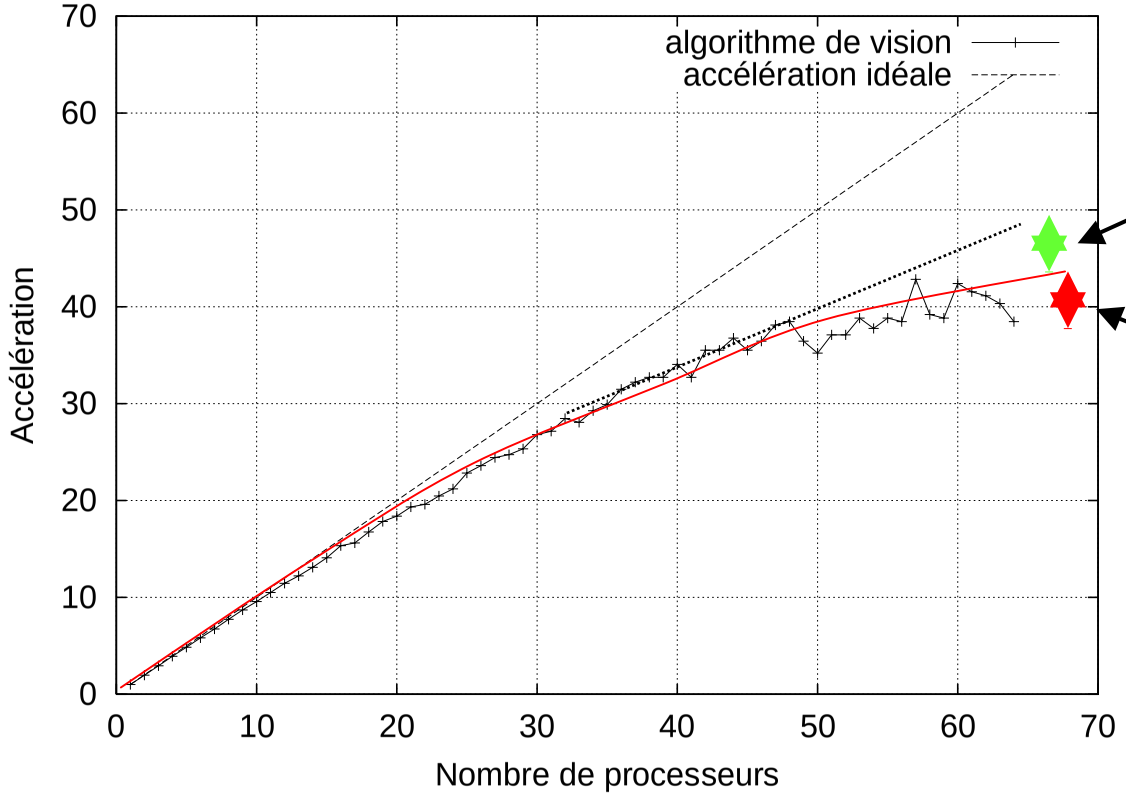
Temps d'exécution
du calcul des champs



1 sec. sur 64 processeurs, O2K, 512x512 cases, rayon de 8

2.1 Vision : performances (2/3)

Accélération, utilisant le temps du thread le plus lent
Tout l'algorithme (allocations + calcul des champs)



Diminution des performances :

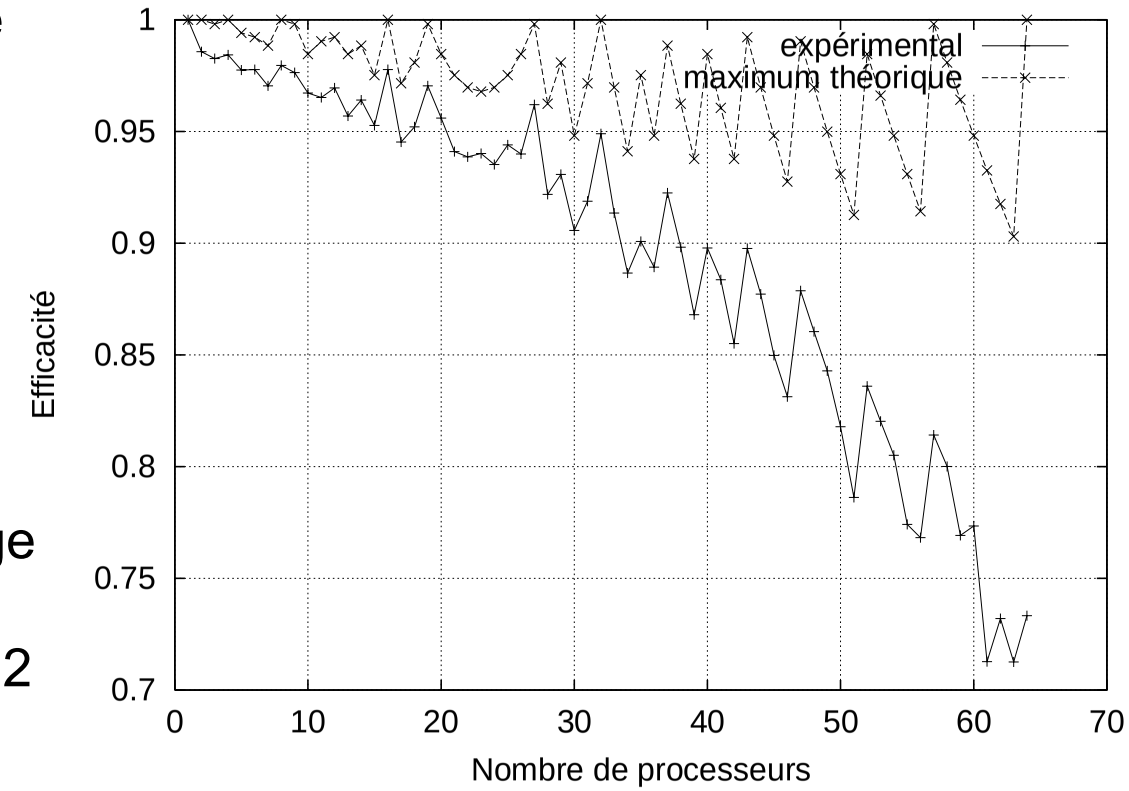
- allocations mémoire (100 ms)
- éfficacité
- [parallélisme]

Accélération de 39 sur 48 processeurs (O2K)

2.1 Vision : performances (3/3)

Efficacité du calcul des visibilités (courbe en zigzag)

- Zigzag <= déséquilibrage de charge
- Concordance des extrema locaux entre la courbe pratique de performances et la courbe théorique de déséquilibrage de charge



Qualité d'équilibrage de charge

Q_{ldbal} : N* -> (0, 1]

$$\begin{cases} 1 & \text{, si } P \mid 512 \\ \dfrac{512}{P \times ([512/P]+1)} & \text{, sinon} \end{cases}$$

2.2 Propagation par vagues : exemple de champ

Utilité :

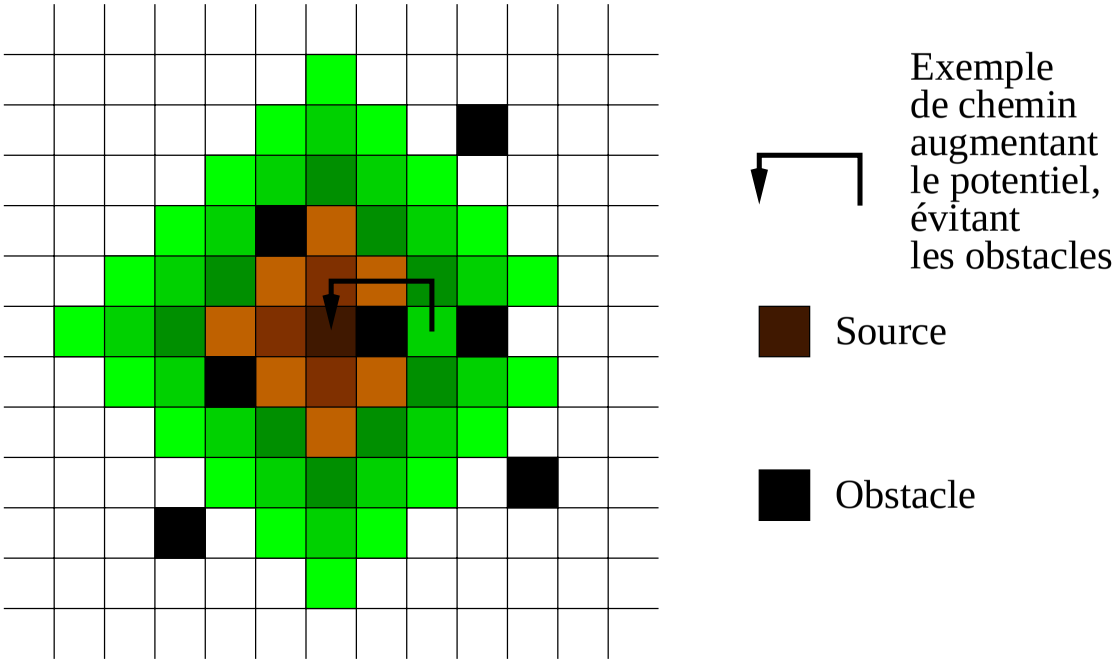
- retrouve le chemin vers la ressource

Raison :

- existence des obstacles

Propriétés :

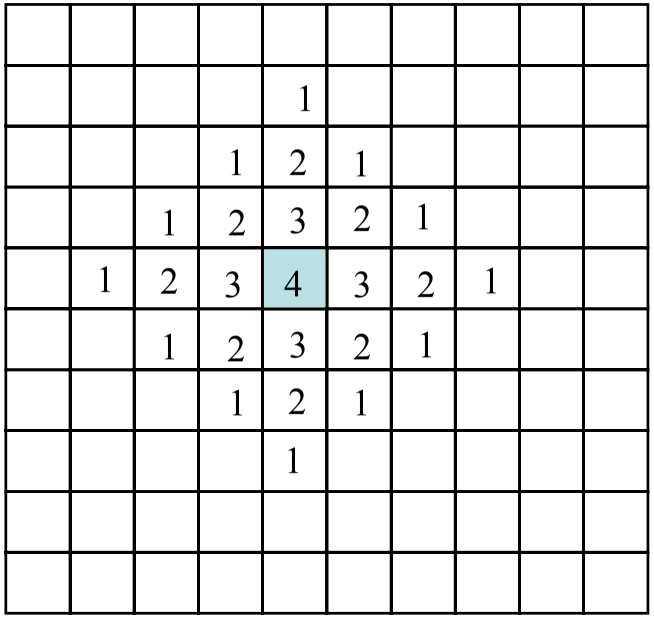
- contourne les obstacles
- décroît avec la distance
- potentiel de la ressource est une fonction de sa charge en objets
- fonction de superposition de champs



2.2 Propagation par vagues : méthodes séquentielles

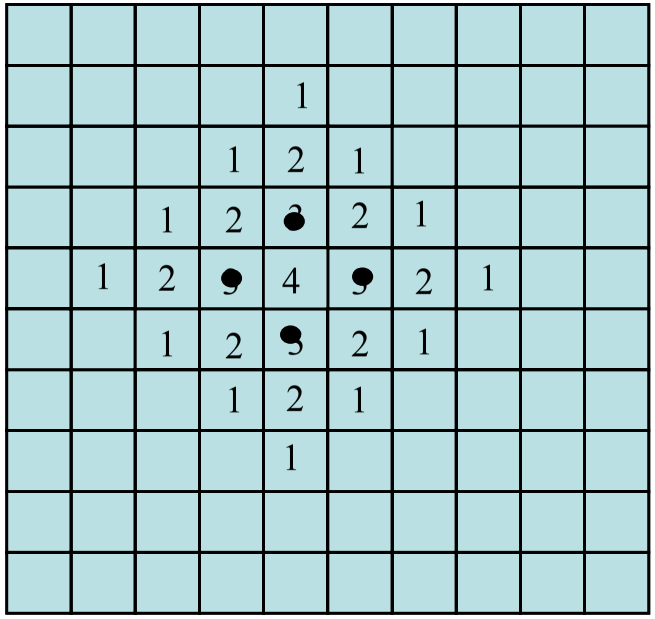
Récursion (en largeur)

pour chaque ressource
    propage-la récursivement



Itérative

met potential des resources
répéter
    pour chaque case
        $p = \max p_i - 1$
jusqu'à pas de modification

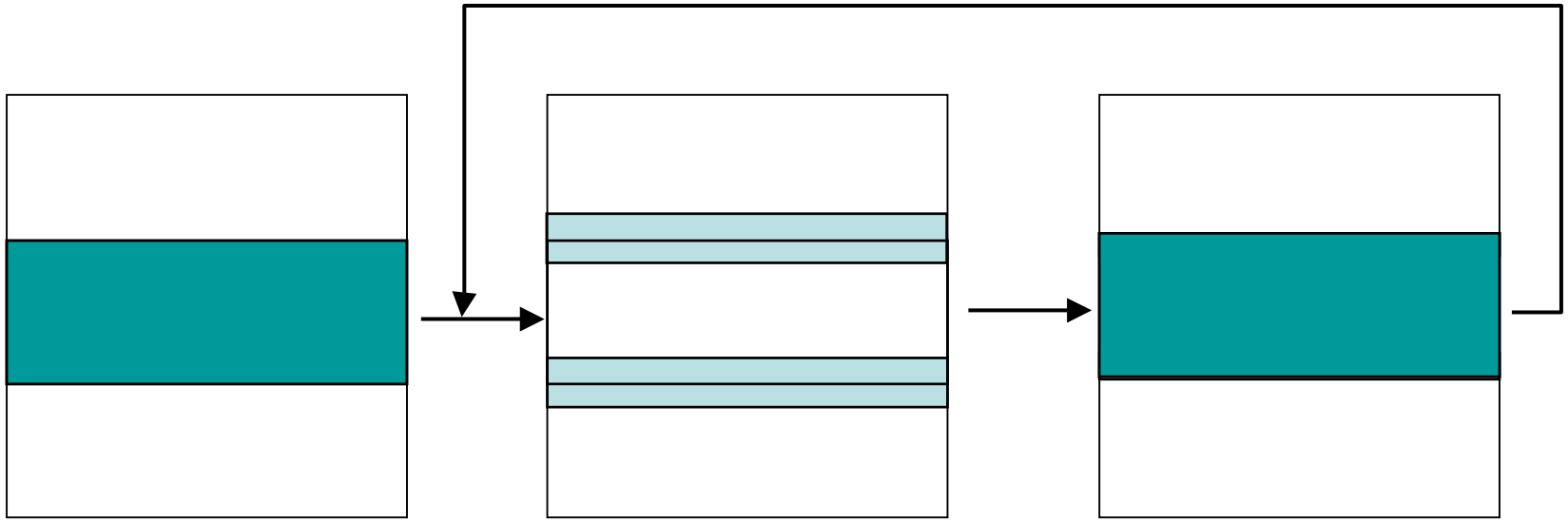


balayage haut-bas

balayage bas-haut

2.2 Propagation par vagues : méthodes parallèles (1/2)

Méthode de décomposition fixe du domaine

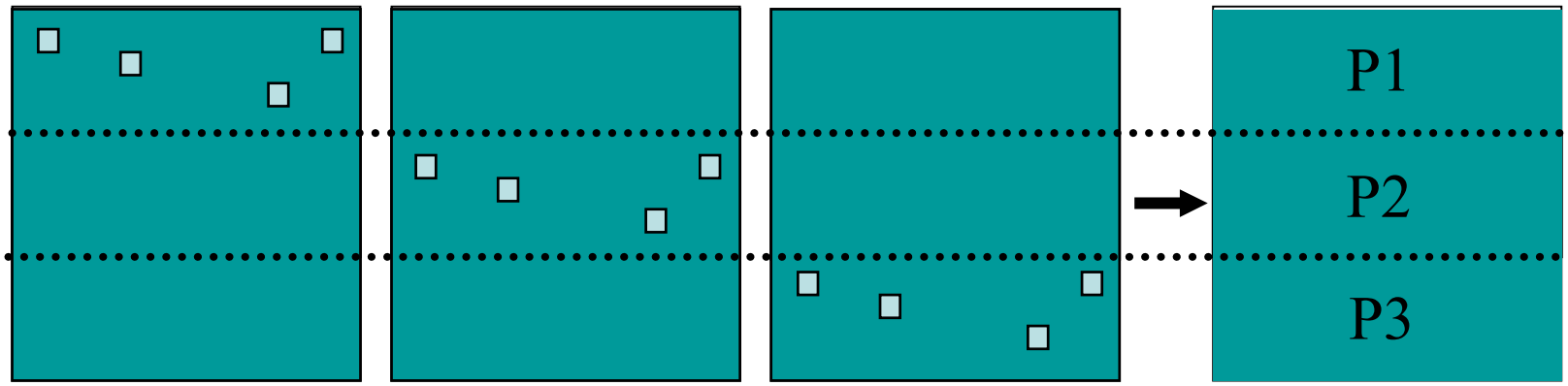


Avantage : peu de transferts de données

Inconvénient : plusieurs repropagations

2.2 Propagation par vagues : méthodes parallèles (2/2)

Méthode des environnements privés



Avantage : évite les repropagations

Inconvénients : défauts de cache

plus de mémoire

2.2 Propagation par vagues : performances (1/3)

Temps d'exécution des méthodes séquentielles

Paramètre \ Méthode	Récursive	Itérative
Taille environnement	+++	+++
Connectivité (4/8)	+++	+
Nombre de ressources	++	=
Potentiel des ressources	++	=
Nombre des obstacles	=	++

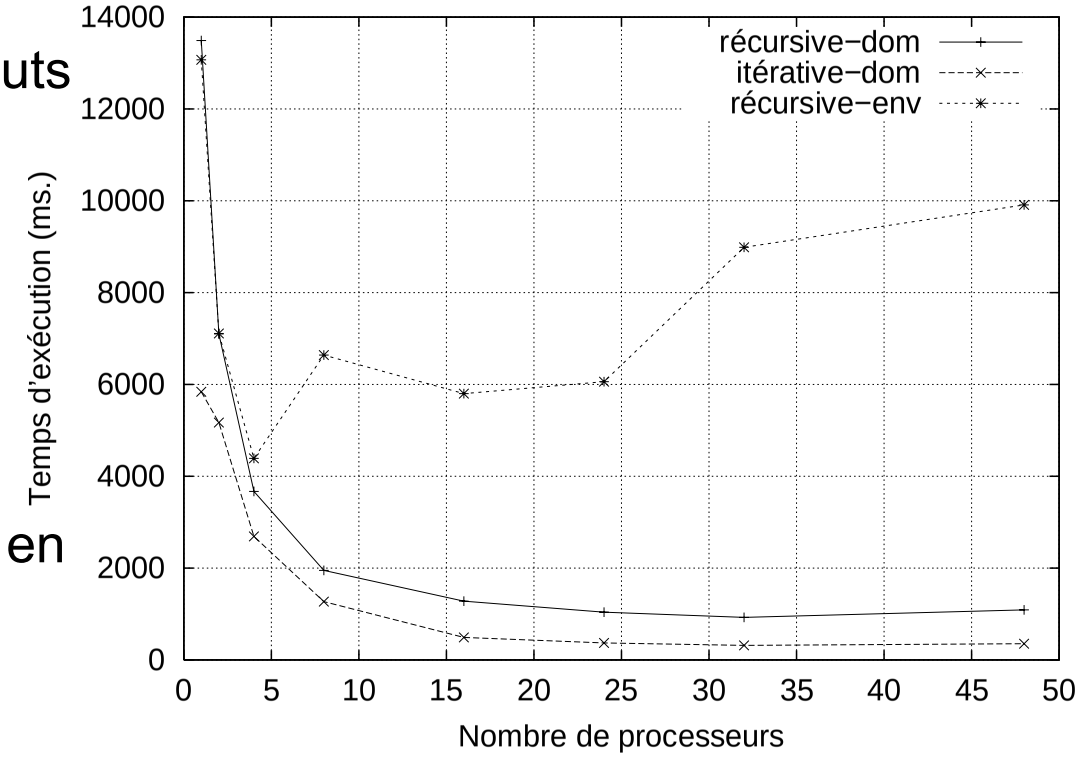
- Performances égales sur O2K :
  - connectivité de 4
  - sans obstacles
  - 1,1-1,45 ressources / case

2.2 Propagation par vagues : performances (2/3)

Performances des méthodes parallèles

1024x1024, sans obstacles, 1 % de ressources, potentiel des ressources de 16

- Environnements privés : défauts de cache => élimination
- Temps séquentiels :
  - **récursion : 13,5 sec.**
  - **itérative : 6 sec.**
  - **Tis < Trs**
- Temps parallèles :
  - **Ti < Tr**
- Idée : séparer la propagation en propagation du domaine et propagation des frontières :
  - **Tr = Tsr + Tpr**
  - **Ti = Tsi + Tpi**
  - **Tm = Tsi + Tpr**

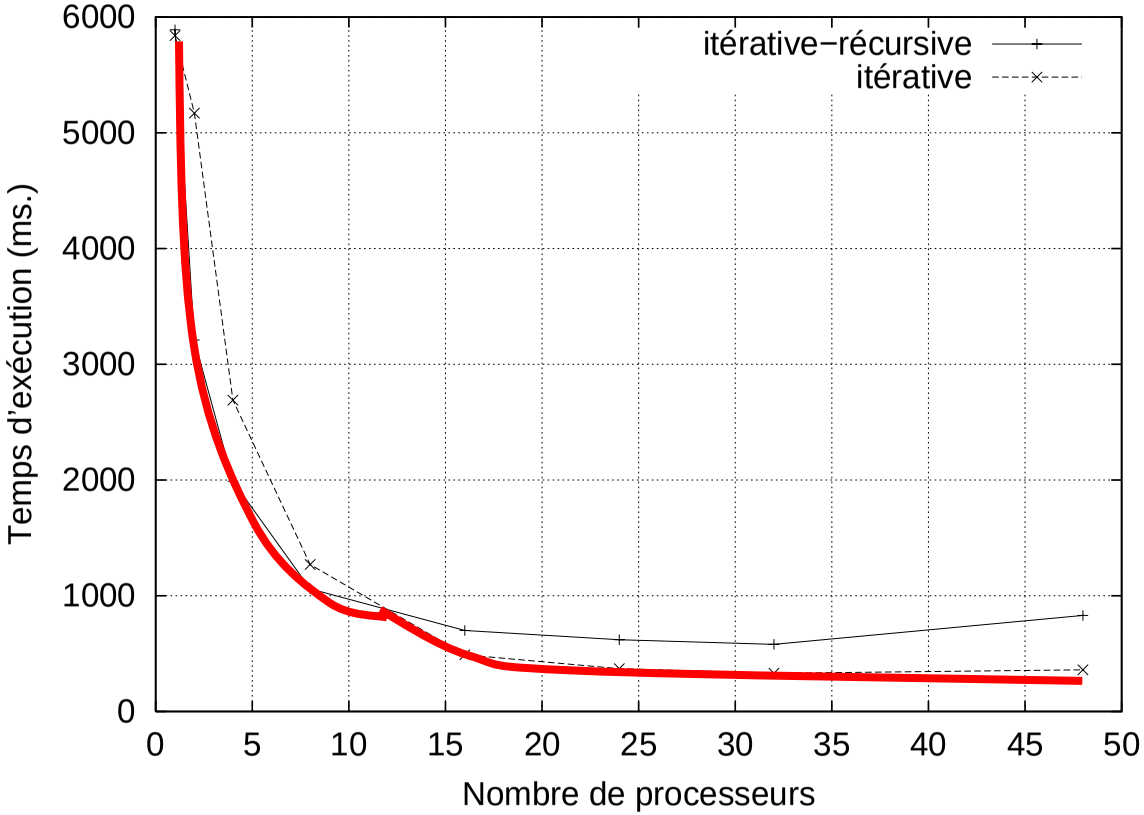


2.2 Propagation par vagues : performances (3/3)

Performances des deux meilleures méthodes parallèles

1024x1024, sans obstacles, 1 % de ressources, potentiel des ressources de 16

- Itérative
- Combinaison :
  - propagation du domaine : itérative
  - propagation des frontières : récursive

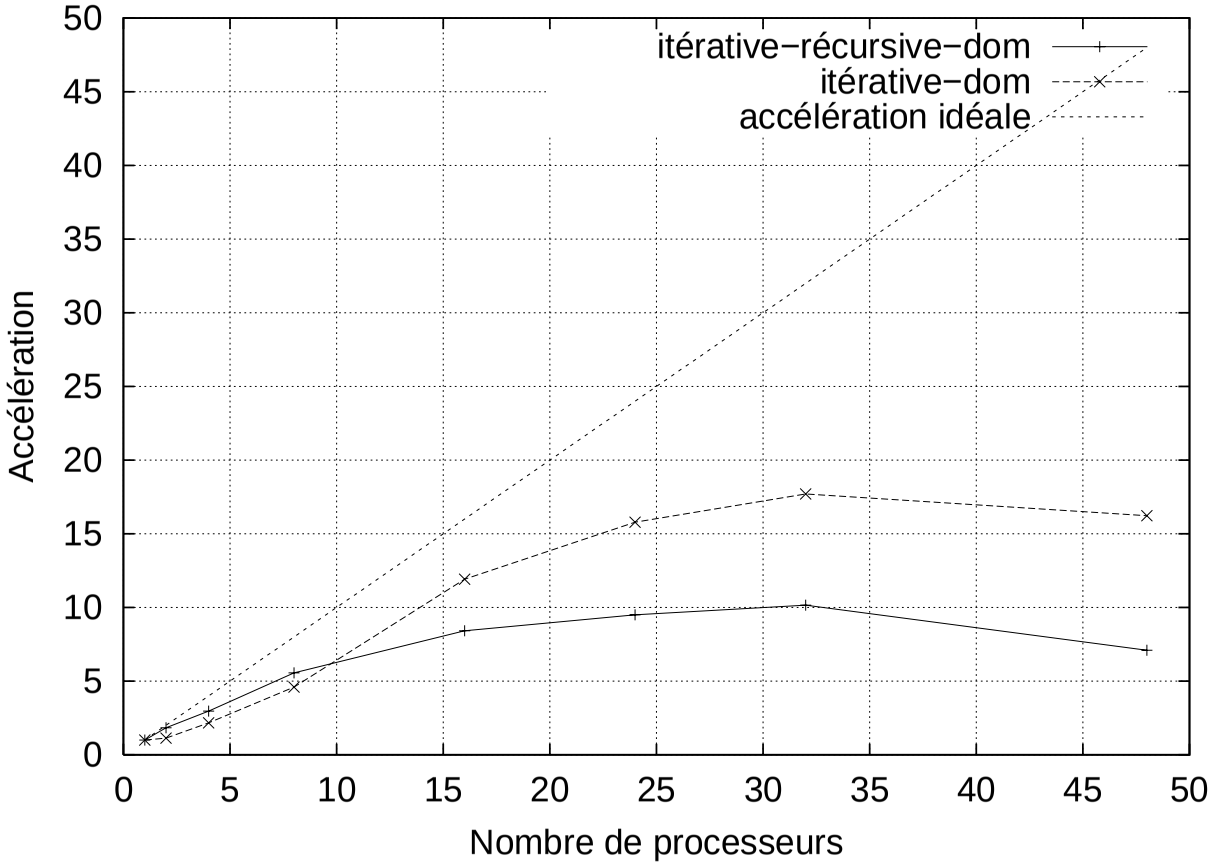


2.2 Propagation par vagues : conclusions

Meilleure méthode :

- propagation du domaine :
  - itérative
- propagation des frontières :
  - P petit : itérative
  - P grand : récursive

Accélération
(même Tséq) (O2K)

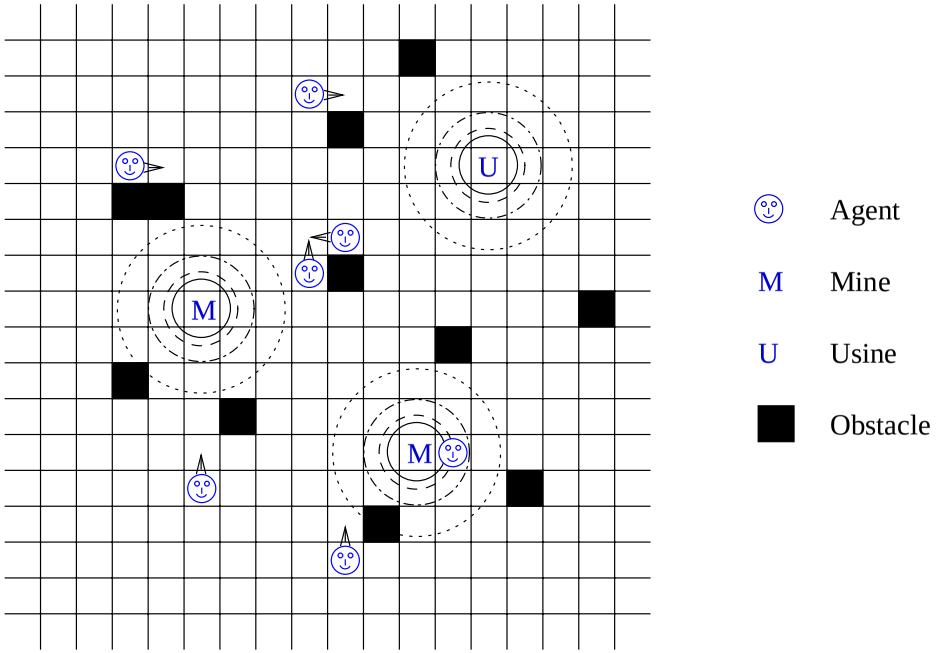


=> Propagation inexacte des potentiels => travaux futurs

3. Implantation du modèle : utilisation

- Écrire une application, étapes :
  - initialisation de l'environnement
  - fonction utilisateur
    - sauvegardes pour visualisation
  - ressources :

    charge => potentiel
  - comportements des agents
- Visualiser une simulation
  - outil graphique



3. Implantation du modèle : application au trafic routier

- Motivation
  - temps, argent, concerne tout le monde
- Simulations macroscopiques, microscopiques
- Autres travaux
  - Mathematical and Computer Modelling, 35 (2002)
  - Transportation Research Board
  - automates cellulaires
  - simulateurs spécifiques : Smartest Project
- Est-ce que le modèle MA est approprié pour la simulation du trafic routier ?
  - oui, et une implantation bien fonctionnelle

3. Trafic routier : simulation des objets

- E.g. 1 case = 3m x 3m, 1 cycle = 0.1 sec.
  - vitesse maximum : 30 m/s ⇔ 110 km/h
- Bords des rues = obstacles
  - rues à plusieurs files
- Voitures = agents
- Feux = ressources, couleur = charge
- Comportement des voitures, trajectoire = fonction spécifique aux agents
  - destination = position finale (x,y)

3. Trafic routier : comportement des voitures

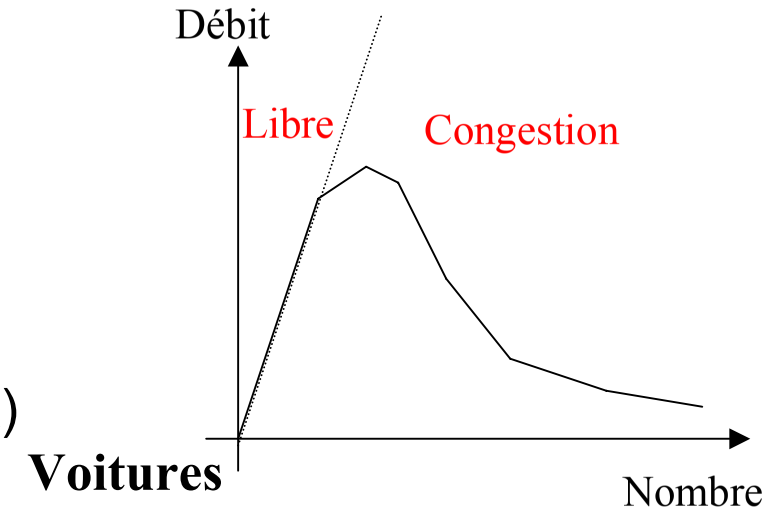
- Mets à jour ma vitesse
- Choisis la direction (trajectoire)
  - distance avec la prochaine voiture
- Calcule la vitesse sûre
- Accélère / décélère
- Si je bouge
  - suis-je à la destination ?
  - fais-je un accident / infraction ?
  - bouge

3.Trafic routier : simulation de la vitesse

- Problème :
  - la voiture bouge tous les N cycles ⇔ vitesse = 1/N
  - il faut la fonction continue (rationnel) -> discret (entier)
- Idée :
  - ressemblance avec l'algorithme de Bresenham
- Solution :
  - toute voiture a un compteur de « pas » effectués, initialisé à 0
  - à chaque cycle :
    - compteur += vitesse
    - si compteur >= VITESSE_MAX
      - compteur -= VITESSE_MAX
      - bouge la voiture
    - sinon, voiture reste sur place

3. Trafic routier : patterns observés

- Diagramme fondamental



- Feux
  - optimisation de l'attente aux feux
- Bouchons
  - propagation (en arrière ou en avant)
  - changement de taille
- Accidents et infractions
  - perspectives : identification *automatique* des zones accidentogènes
- Trajectoire
  - optimisation
  - changement dynamique, adaptabilité

3. Trafic routier : simplicité de programmation

- Coder le comportement / les paramètres, pas la simulation (style Prolog)
  - simulation prise en compte par la machine virtuelle de la bibliothèque
- Programmation rapide et plus facile
  - en total, 300-500 lignes de code source
- Performance
  - séquentielle : moins d'1 sec. pour plusieurs rues et quelques dizaines de voitures pendant 1000 cycles
  - parallèle : à voir sur un exemple de plus grande taille (ville)

Bilan : contributions

- Modèle de simulation de SMA
- Algorithmique parallèle des percepts des agents :
  - vision
  - propagation des potentiels
- Implantation parallèle du modèle
  - bonnes performances pour les percepts et les applications
- Application au trafic routier
  - bouchons
  - carrefours
  - optimisation des trajectoires

Bilan : perspectives

- Extension du modèle
- Optimisation des algorithmes
  - équilibrage dynamique de charge
- Extension de l'environnement de programmation
- Simulation de trafic :
  - taille plus grande, e.g. quartier d’une ville, optimisation de trajectoire
  - voitures de plusieurs cases
- Autres applications