Nouvelles fonctionnalités pour le parallélisme de données

Bertrand Ducourthial
HEUDIASYC UMR CNRS 6599
Université de Technologie de Compiègne
Bertrand.Ducourthial@utc.fr

Nicolas Sicard
IEF UMR CNRS 8622
Université de Paris-Sud
sicard@ief.u-psud.fr

Avril 2001

Résumé

- parallélisme de données
- \+ manipulation des données irrégulières `snet`, `vnet`
- \+ primitive asynchrone `scour`
- $\leadsto$ nouveaux outils pour la programmation
- $\leadsto$ nouvelles applications pour le parallélisme de données

Parallélisme de données

Exemples

- `matrice J; image I;`
- `int s;`
- `J = min(I,s);`
- `I = I + J;`
- `Where {I == 4} I = 5; EndWhere`

Intérêt

- parallélisation intuitive
- expressivité
- simplicité des constructions
- preuves de programmes

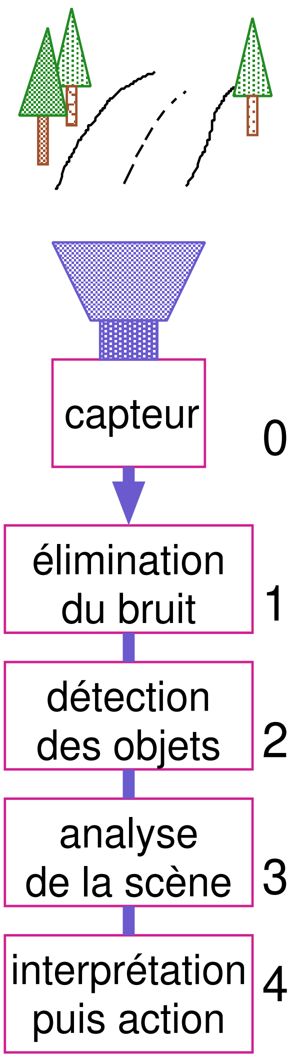
Utilisations

- architectures spécialisées
- circuits SIMD dans les processeurs standard
- environnements logiciels sur architectures standard
- matrices, géométrie, image...

Problèmes

- manipulations de données irrégulières (*e.g.*, régions)
- forte synchronisation de traitements élémentaires
- machine virtuelle sur architecture standard
- irrégularité des traitements

Origine de l'étude : image



Traitement d'images (phases 1 2)

- améliorer l'image captée
- y déceler des «objets»

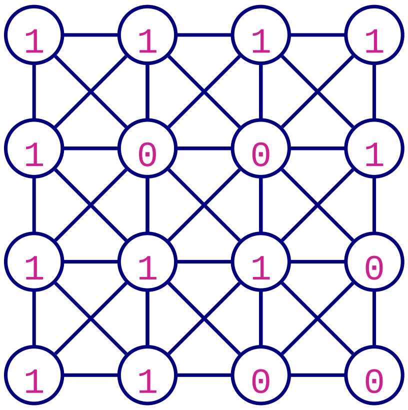
Données

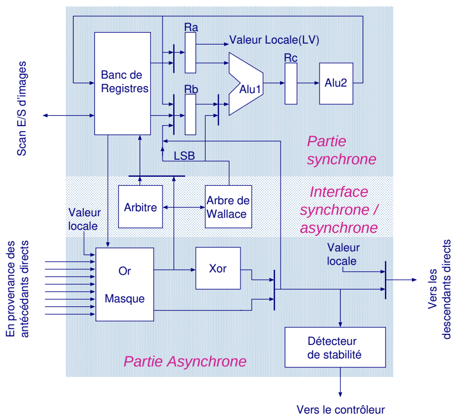
- grande quantité
- information géographique forte
- irrégulières

Calculs

- locaux : pixels dans le voisinage
- globaux : objets dans l'image
- irréguliers ($e.g.$, segmentation)

Origine de l'étude : architecture





Machines parallèles

- quantité de données
- délai de traitements

  $\rightsquigarrow$ tableaux de PE

Évolutions

- grille
- autres topologies
- grille + autres circuits
- archi. reconfigurables
- archi. asynchrones

Modèle

- parallélisme de données
- mélanger calcul et communications
- primitive asynchrone

Réseaux Associatifs

Étude de l'architecture

- Maille associative
- multi-processeurs SIMD asynchrone

Modèle de calcul

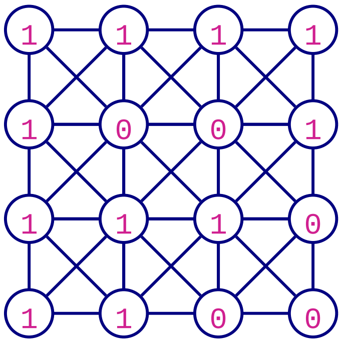
- primitive asynchrone
- conditions de terminaison
- structure de données
- algorithmique
- mise en œuvre logicielle

API

- Anet : bibliothèque parallèle multithread
  hiérarchie de classes C++, threads POSIX ou PM2
- programmation :
  parallélisme de données
  + nouvelles structures de données
  + nouvelles primitives

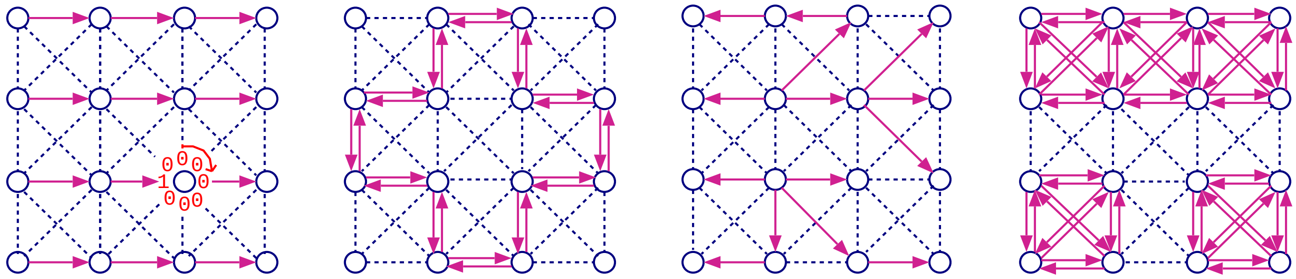
$\leadsto$ *nouvelles perspectives pour le parallélisme de données*

Structures de données (1/2)



- Graphe de base
  - quelconque mais fixé
  - machine virtuelle d'instanciation
- Variables parallèles
  pvar ⟨ int ⟩ image;

- Données irrégulières



a. lignes.
mask = 00000010
sur les liens entrants

b. contours

c. forêt

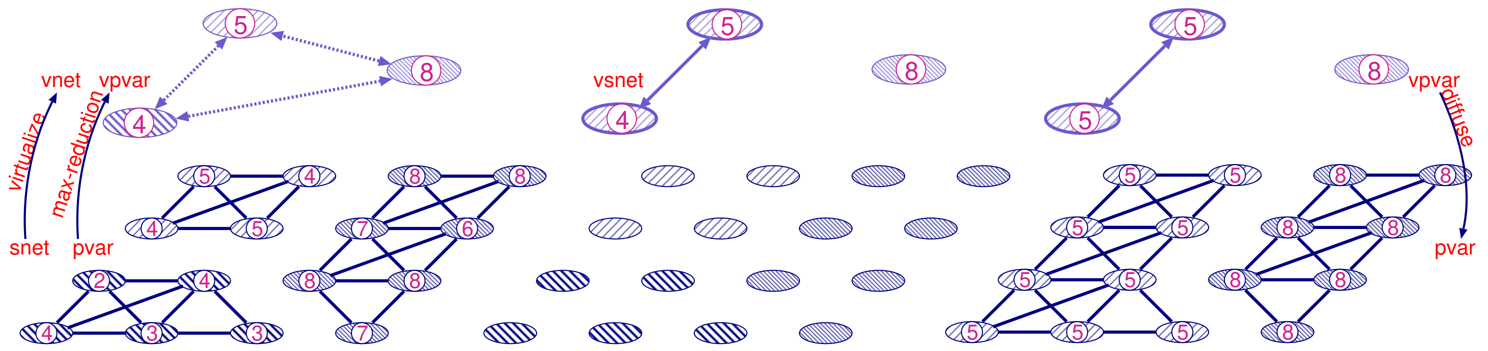
d. régions

- sous-graphe du graphe de base
- mask : mot binaire indiquant les arcs masqués
- pvar ⟨ mask ⟩ forêt
- `symetrize`, `link-with-equals`, `invert`...

Structures de données (2/2)

- Éliminer les traitements inutiles
  - moins de processeurs physiques que de processeurs virtuels dans la machine d'instanciation
  - `Where...ElseWhere...EndWhere`
  - calculs sur les objets $\Rightarrow$ bcp de calculs sur les pixels
- Simplifier l'algorithmique
  - exprimer les traitements sur les régions sans référence aux pixels sous-jacents.

- "Virtualisation"



Algorithme 1 :

```
subnet sn1, sn2;
vnet vn1(sn1), vn2;
vn2.Virtualize(sn2)
with vn1 do
  ...
end with
```

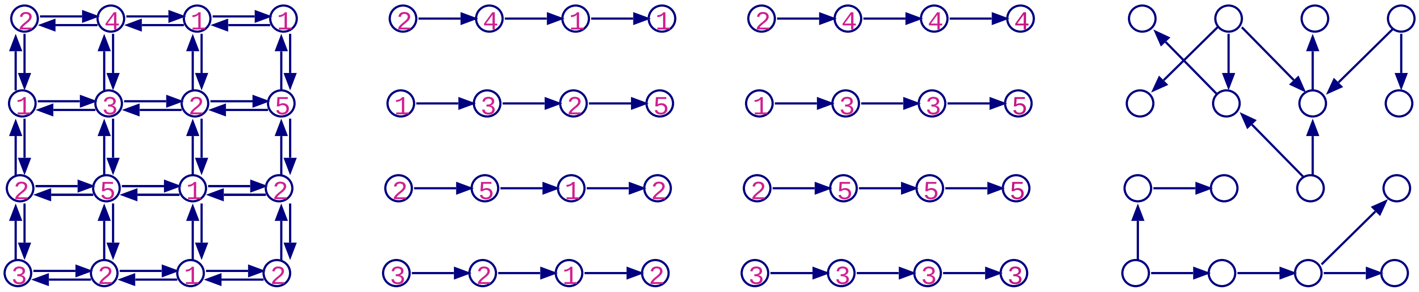
Algorithme 2 :

```
pvar  p;
vpvar  vp;
with vn do
  vp.MaxReduction();
  vp.Diffusion(p);
end with
```

Primitives de calcul (1/2)

Calcul sur un voisinage

- moyennage en C$\star$:
  `I = ([.-1][.]J + [.+1][.]J + [.][-1]J + [.][+1]J + J)/5`
- `snet vois = 10101010;`
  `I = +-step-scour(J, vois); I = I / 5`



a.snet pour calculer la moyenne des 4 plus proches voisins avec step-scour.

b.la primitive max-scour sur un snet en lignes correspond à un max-scan (état initial à g., résultat à d.).

c.exemple de snet préfixe (sur lequel la primitive scour généralise les scans).

Calcul sur une ligne

- cascade de step-scour $\equiv$ scour
- `snet lignes = 00000100`
  `I = max-scour(J, lignes)`
- *cf.* `max-scan`

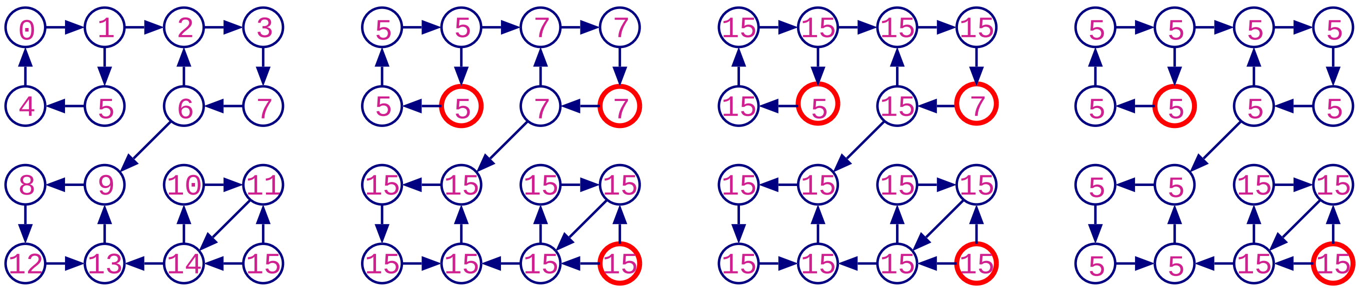
Graphes préfixes

- au plus un chemin entre deux sommets
- tout type d'opérateur ; extension des `scan`

Primitives de calcul (2/2)

- Graphes quelconques
  - s-opérateurs (associatifs, commutatifs, idempotents) $e.g.$, min(x,y)
  - r-opérateurs idempotents $e.g.$, minc(x,y) $=$ min(x,y$+$1)
  - si non idempotent ($e.g.$, $+$), `spanning tree`
- `scour` : primitive asynchrone
  - dé-synchronisation des traitements élémentaires
  - `step-scour` sur le voisinage
  - `scour` sur une ligne
  - `scour` sur un graphe préfixe
  - `scour` sur un graphe quelconque
    s-opérateurs, r-opérateurs idempotents
- Intérêt
  - circuits électroniques asynchrones :
    communication en temps moyen au lieu du pire temps
  - calcul parallèle sur multi-processeurs ($e.g.$, matrices) :
    limitation des barrières de synchronisation
  - bibliothèque data-parallèle : diminution du contrôle
  - algorithmique : expressivité

Algorithmique (1/4)



a. pvar index.

b. après max-scour
3 candidates.

c. après Where
(INT_MAX=15).

d. après min-scour,
2 candidates.

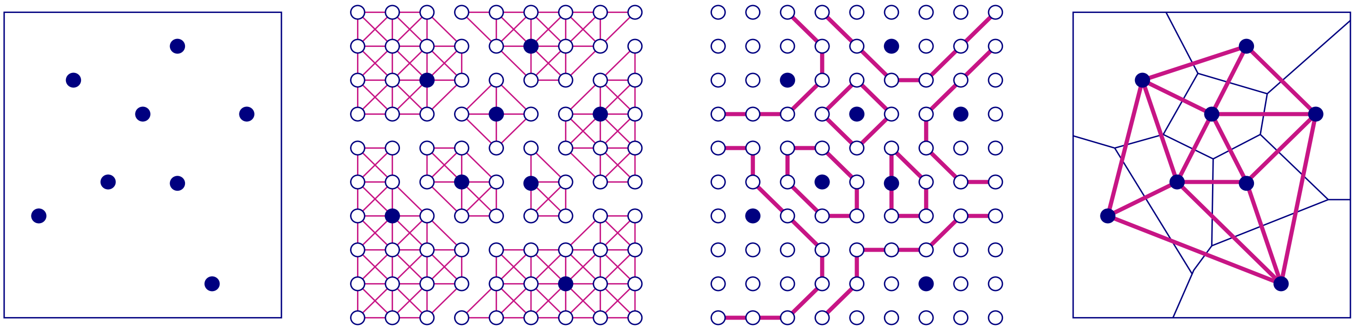
Algorithme 3 : **SourceLogique(**pvar ⟨ mask ⟩ graphe**)**

```
pvar ⟨ int ⟩  candidates = max-scour(graphe, index)
where candidates ≠ index do
  candidates = INT_MAX
  ▷ élément neutre pour le min sur les entiers.
end where
candidates = min-scour(graphe, candidates)
where candidates == index do
  candidates = 0 ▷ Les candidates restantes sont les nœuds à zéro.
end where
```

Algorithme 4 : **TriTopologique(**pvar ⟨ mask ⟩ graphe**)**

```
pvar ⟨ int ⟩  p = 1        ▷ p vaut 1 sur tous les nœuds.
p = maxinc-scour(graphe,p) ▷ Le résultat est dans la pvar p.
```

Algorithmique (2/4)



a. graines dans le plan **b**. snet codant les régions. **c**. snet codant les polygones. **d**. vnet codant la triangulation

Algorithme 5: **PolygVoronoï(** pvar ⟨ mask ⟩ grille, pvar ⟨ mask ⟩ régions**)**

```
pvar ⟨ bool ⟩  bord = 0           ▷ 0 partout
where  régions ≠ grille  do
  bord = 1                          ▷ sauf sur les bords des régions.
end where
pvar ⟨ mask ⟩  poly = link-with-equals(grille, bord, bord)
▷ poly inclut les connexions entre bords mais aussi les
connexions entre régions.
poly = poly & régions
▷ Suppression des connexions qui ne sont pas dans régions.
return  poly
```

Algorithme 6: **TriangVoronoï(** pvar ⟨ mask ⟩ grille, pvar ⟨ mask ⟩ régions**)**

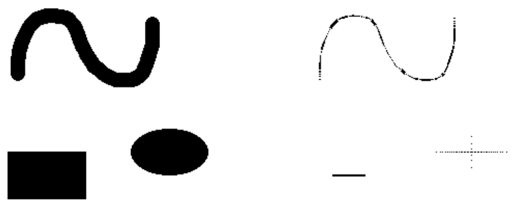
```
vsnet  triangles = virtualize(régions)
return  triangles
```

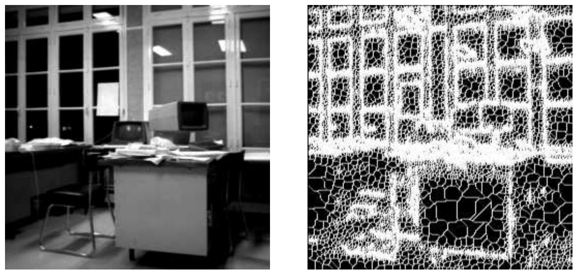
Algorithmique (3/4)

Algorithme 7 : **RégionsVoronoï(** pvar ⟨ mask ⟩ grille, pvar ⟨ bool ⟩ graines**)**

```
pvar ⟨ int ⟩  dist = INT_MAX                         ▷ INT_MAX partout
where  graines == 1  do
  dist = 0                                      ▷ sauf sur les graines.
end where
dist = minc-scour(grille, dist)    ▷ Distance à la graine la plus proche.
pvar ⟨ mask ⟩  régions = link-with-equals(grille, dist - 1, dist)
  ▷ Liens en provenance des voisins plus proches d'une graine.
tmp = min-step-exclusive-scour(régions, index)
  ▷ Plus petit des plus proches voisins.
régions = link-with-equals(grille, tmp, index)
  ▷ Forêt enracinée sur les graines ; un seul lien par nœud, en
  provenance d'un voisin plus proche d'une graine.
where  graines ≠ 1  do
  tmp = index                  ▷ index des graines sur les graines,
elsewhere
  tmp = 0                                          ▷ et 0 ailleurs.
end where
tmp = max-scour(régions, tmp)
  ▷ Diffusion de l'index des graines sur la forêt.
régions = link-with-equals(grille, tmp, tmp)
  ▷ Connexion des nœuds ayant la même graine.
return  régions            ▷ Graphe des régions de Voronoï.
```

Algorithmique (4/4)





Algorithme 8: **Squelettisation(**pvar ⟨ bool ⟩ image, pvar ⟨ bool ⟩ squel**)**

```
pvar ⟨ int ⟩  dist = 0                          ▷ 0 sur le fond
where  image == 1 do
  dist = INT_MAX  ▷ et l'élément neutre de minc sur les formes.
end where
dist = minc-scour(grille, dist)        ▷ Distance aux contours.
pvar ⟨ int ⟩  maxima = max-step-scour(grille, dist)
▷ maxima locaux.

where  maxima == dist  do
  squel = 1                  ▷ résultat dans l'image binaire squel.
elsewhere
  squel = 0 ▷ 1 sur le squelette (maxima locaux) et 0 ailleurs.
end where
```

Conclusion

Parallélisme de données

- parallélisation intuitive, expressivité, simplicité
- naturel pour l'image

Problèmes

- manipulations de données irrégulières (*e.g.*, régions)
- forte synchronisation de traitements élémentaires
- machine virtuelle sur architecture standard
- irrégularité des traitements

Contribution

- manipulation des données irrégulières `snet, vnet`
- primitive asynchrone `scour`
- $\leadsto$ nouveaux outils pour la programmation

  $\leadsto$ nouvelles applications pour le parallélisme de données

Perspectives

- Algorithmique
- Anet séquentielle, mémoire partagée
- Anet mémoire répartie ?