AMÉLIORATION DE LA MÉTHODE EPS POUR UN CENTRE DE CALCUL

Mohamed Rezgui (Univ. Nice Sophia Antipolis)

Jean-Charles Régin (Univ. Nice Sophia Antipolis)

Arnaud Malapert (Univ. Nice Sophia Antipolis)

Plan

- Motivations : Paralléliser la recherche de solutions en CSP
- Etat de l’art (Programmation par contraintes)
- Etat de l’art (Parallélisme)
- Contributions : EPS & améliorations
- Résultats
- Conclusion & Perspectives

Motivations

- Proposer une méthode simple et efficace qui parallélise la résolution de problèmes avec des facteurs linéaires

  (des centaines voir des milliers de cœurs)

- Types d'instances concernés :
  - l'énumération de toutes les solutions,
  - la preuve l'optimalité,
  - Chercher la solution optimale et la prouver

- Appliquer la méthode sur tout type d'architecture
  - Multi-cœurs, multi-processeurs, cluster, cloud, etc.

PROGRAMMATION PAR CONTRAINTES

Programmation par contraintes (1)

- Paradigme né dans les années 70
- Un problème est défini par un modèle constitué de variables et de contraintes
- On associe à chaque variable du modèle un domaine de valeurs
- Les contraintes permettent de réduire l’espace de solutions à parcourir

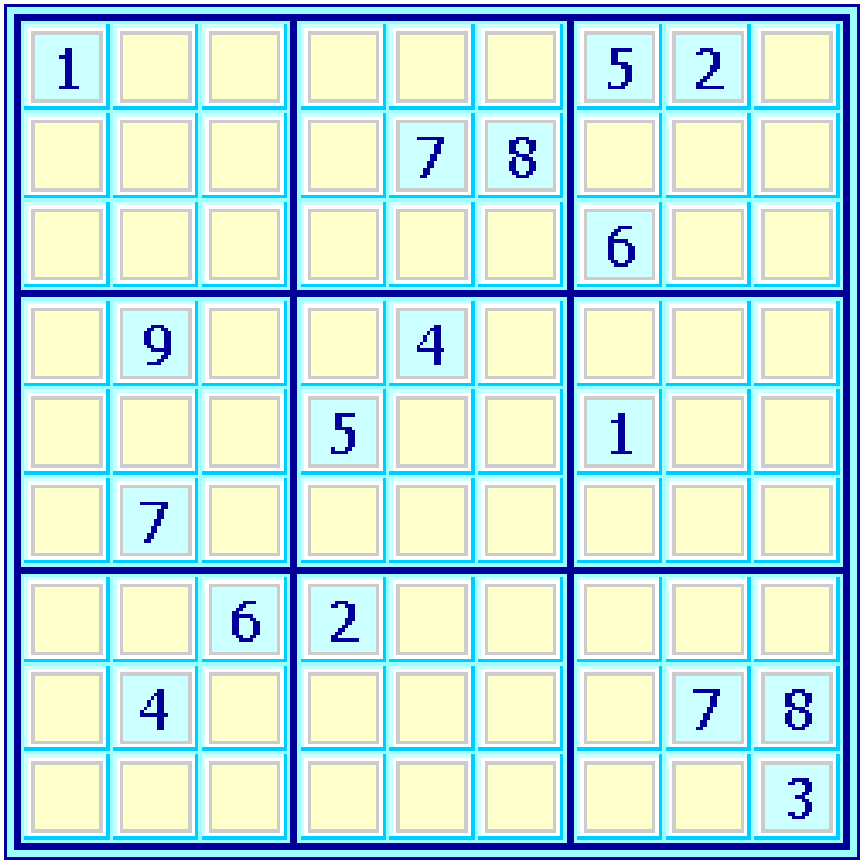
Programmation par contraintes (2)

- Chaque contrainte est associé à un algorithme de filtrage qui élimine les combinaisons de valeurs des domaines de variables qui ne sont pas admissibles dans une solution

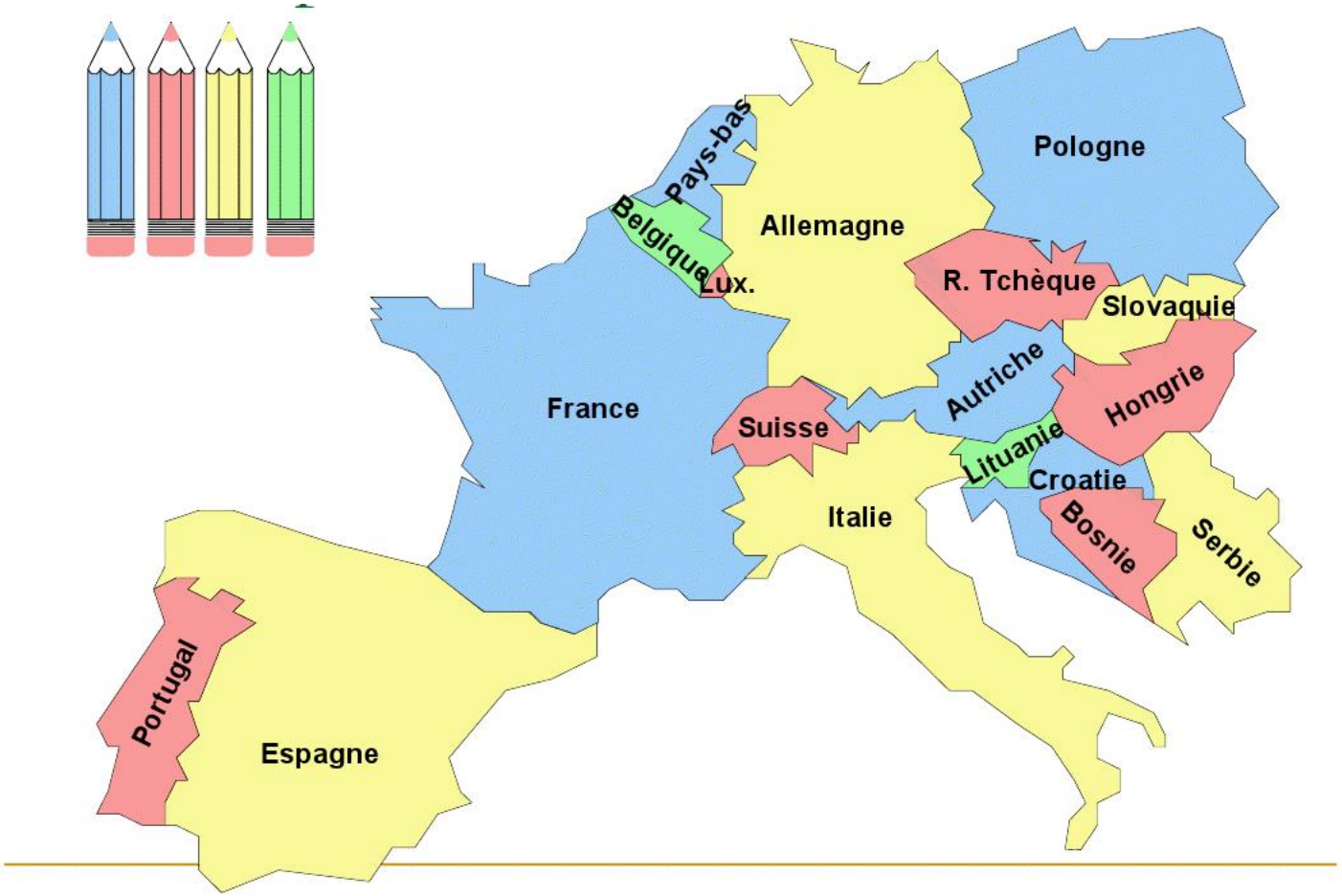
- Le mécanisme de propagation permet d'appeler chaque algorithme de filtrage pour éliminer tour à tour les valeurs non consistantes.

- Il s'arrête lorsqu'on ne trouve plus de valeurs inconsistantes => point fixe

Exemple : Sudoku



Exemple : Coloriage d'une carte



Exemples concrets

- Vérification de programmes & localisation d'erreurs
- Rangement de fichiers sur des supports
- Découpe de barres d'acier en réduisant au maximum les pertes
- Rangement de cartons dans des palettes
- Etablir un emploi du temps en respectant les contraintes de ressources

Solveur

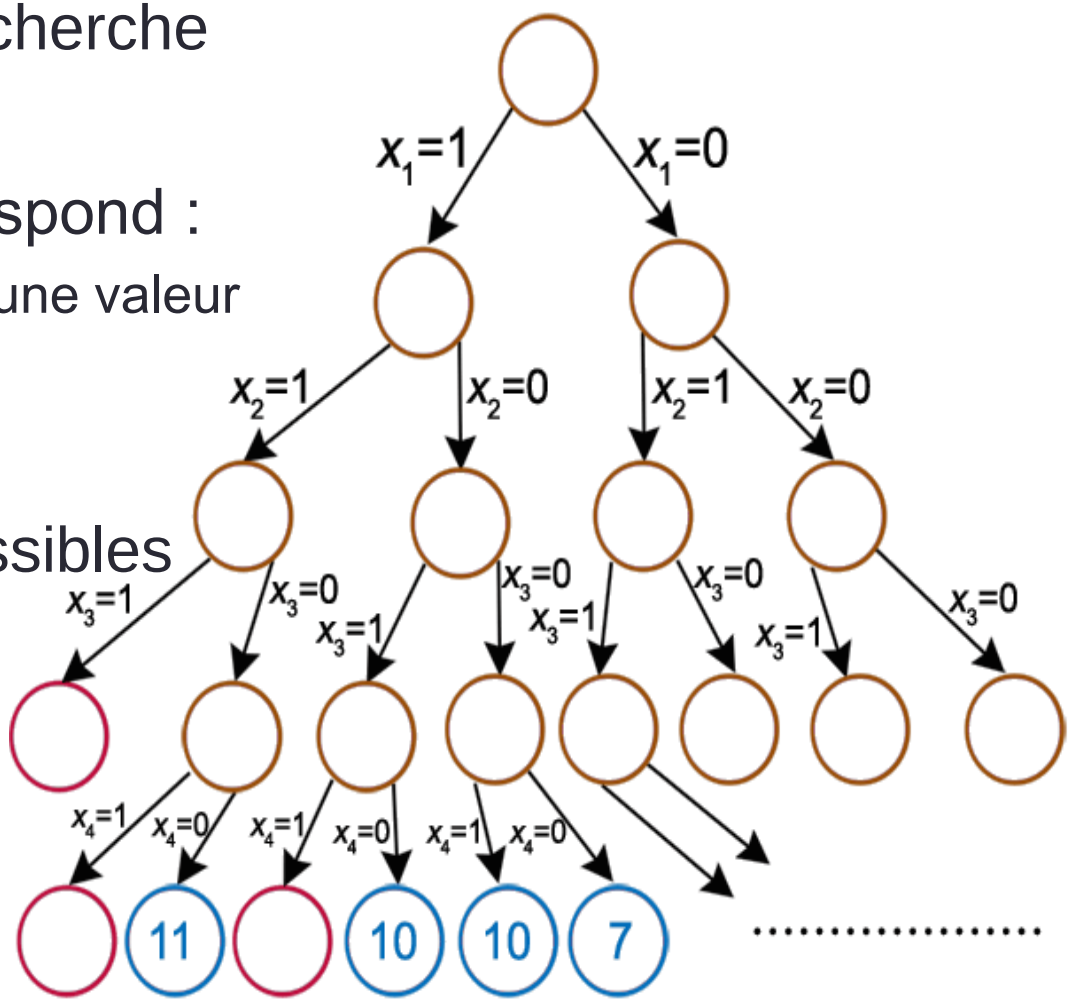
- Logiciel permettant d'utiliser la programmation par contraintes

- Plusieurs solveurs existent :
  CP Optimizer (ILOG (IBM), or-tools (Google), Choco (EMN), gecode, etc.

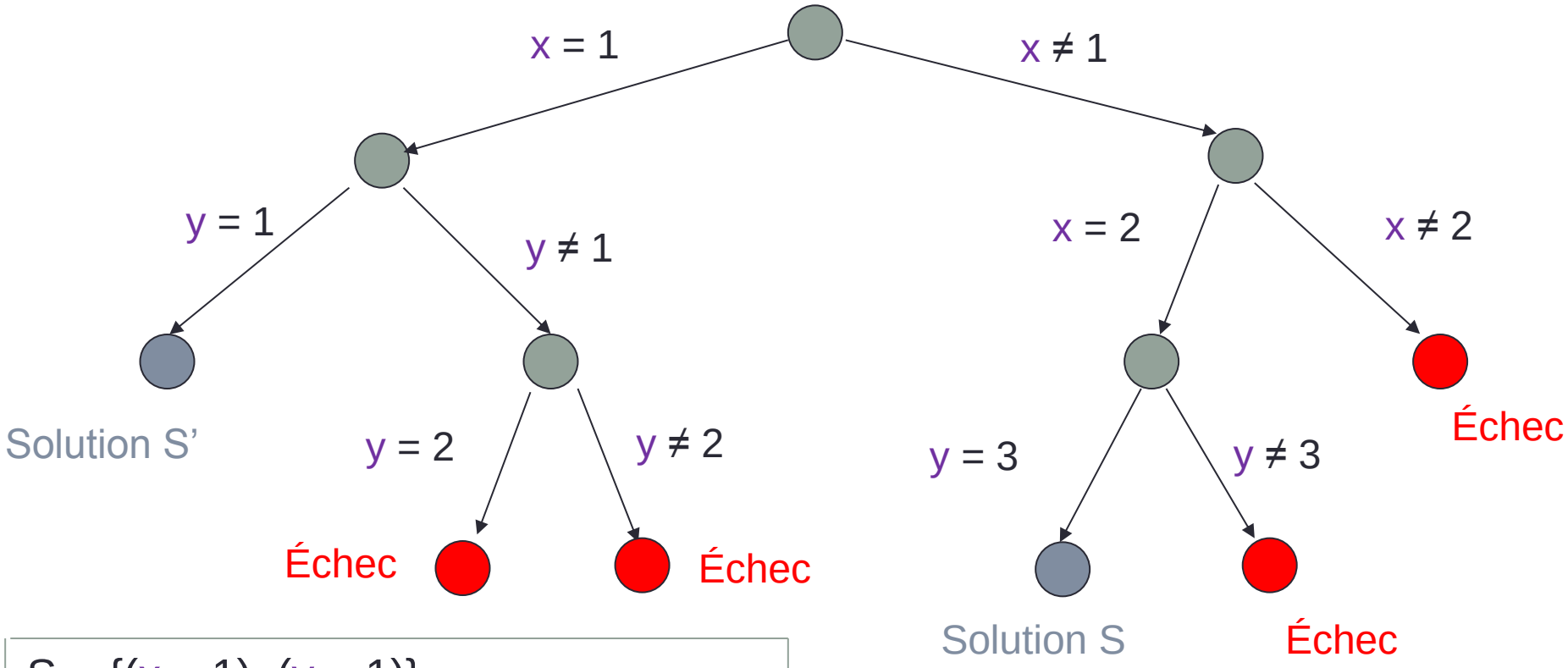
- A partir des variables et des contraintes,
  le solveur construit un arbre de recherche pour chercher les solutions

Parcours de l'arbre de recherche

- Parcours de l'arbre de recherche
- Un nœud de l'arbre correspond :
  - Affectation d'une variable à une valeur
  - Appel de la propagation
- Cherche les solutions possibles
- Backtrack



Parcours de l'arbre de recherche



S = {(x = 1), (y = 1)}
S' = {(x = 2), (y = 3)}

PARALLÉLISME EN PROGRAMMATION PAR CONTRAINTES

Plusieurs types de parallélisme

- Parallélisme dans la propagation ou dans le filtrage
- Parallélisme simulé (Restart)
- CSP distribués (Programmation par agent)
- Parallélisme de l'arbre de recherche
  - Un nœud de l'arbre de recherche correspond à un sous-problème
  - Work stealing
  - Self-splitting
  - EPS

Work stealing

- On a k workers,
  - On coupe le problème en k sous-problèmes,
  - On donne un sous-problème à chaque worker
  - Quand un worker a fini son travail il en demande à un autre worker qui travaille. Celui-ci lui donne une part du travail restant.
- Avantages
  - Meilleure répartition du travail (dynamique)
- Inconvénients
  - Très intrusif dans le solveur
    (évité en partie avec les travaux de B. Le Cun sur Bob++)
  - Il ne faut pas donner un travail trop facile
  - A la fin, tout le monde demande du travail à tout le monde. Il faut bien gérer cela

Self splitting (Matteo Fischetti et al.)

- on a k workers (on attribue à chaque worker une couleur),
  - On coupe le problème en k sous-problèmes,
  - Chaque worker génère des nœuds de l'arbre de recherche durant un temps limité (phase de sampling)
  - On colorie les nœuds de manière déterministe
  - On regroupe les nœuds de même couleur et on les attribue au worker correspondant qui les résout par la suite.
  - Les nœuds jugés difficiles sont mis dans une queue
  - Lorsqu'un worker termine ses nœuds, il prend un nœud difficile dans la queue et le résout
  - Lorsque la queue est vide, les workers qui ont terminé le travail s'arrêtent.
  - Un worker est assigné pour rassembler les résultats lorsqu'il termine son travail.

- Avantages
  - Gestion des nœuds difficiles pour mieux les répartir
  - Pas de communications entre les workers
- Inconvénients
  - Très intrusif dans le solveur (implémentation sur le propagateur de gecode)
  - Ne passe pas à l'échelle au-delà de 64 workers d'après les résultats

CONTRIBUTIONS : MÉTHODE EPS & AMÉLIORATIONS

Méthode EPS

- On a 1 master et k workers,
  - On découpe le problème en un grand nombre de sous-problèmes
    - Plus de 30 sspb / worker
  - On ajoute les sous-problèmes dans un queue
  - Les workers se servent dans la queue à tour de rôle
    - => méthode préemptive
  - Lorsque tous les sous-problèmes sont résolus, la résolution est terminée
- Avantages
  - Simple
  - Pas ou très peu de communications
  - Pas intrusif dans le solveur
    (on a juste besoin de découper le sous-problème et de tester la propagation)
  - On peut rejouer facilement la résolution
    - il suffit de mémoriser l'ordre dans lequel les problèmes ont été résolus
      et par quel worker

Résultats sur 40 coeurs

Instance	Seq.	Work stealing		EPS	
	t	t	s	t	s
allinterval_15	262.5	9.7	27.0	8.8	**29.9**
magicsequence_40000	328.2	592.6	0.6	37.3	**8.8**
sportsleague_10	172.4	7.6	22.5	6.8	**25.4**
sb_sb_13_13_6_4	135.7	9.2	14.7	7.8	**17.5**
quasigroup7_10	292.6	14.5	20.1	10.5	**27.8**
non_non_fast_6	602.2	271.3	2.2	56.8	**10.6**
golombruler_13	1355.2	54.9	24.7	44.3	**30.6**
warehouses	148.0	25.9	5.7	21.1	**7.0**
setcovering	94.4	16.1	5.9	11.1	**8.5**
2DLevelPacking_Class5_20_6	22.6	13.8	1.6	0.7	**30.2**
depot_placement_att48_5	125.2	19.1	6.6	10.2	**12.3**
depot_placement_rat99_5	21.6	6.4	3.4	2.6	**8.3**
fastfood_ff58	23.1	4.5	5.1	3.8	**6.0**
open_stacks_01_problem_15_15	102.8	6.1	16.9	5.8	**17.8**
open_stacks_01_wbp_30_15_1	185.7	15.4	12.1	11.2	**16.6**
sugiyama2_g5_7_7_7_7_2	286.5	22.8	12.6	10.8	**26.6**
pattern_set_mining_k1_german-credit	113.7	22.3	5.1	13.8	**8.3**
radiation_03	129.1	33.5	3.9	25.6	**5.0**
bacp-7	227.2	15.6	14.5	9.5	**23.9**
talent_scheduling_alt_film116	254.3	13.5	**18.8**	35.6	**7.1**
total (t) or geometric mean (s)	488.2	1174.8	7.7	334.2	**13.8**

Problème du passage à l'échelle

- Speedup non linéaires à partir d'une centaine de cœurs
  - **Mais** Speedup linéaires pour la résolution des sous-problèmes
- Donc la décomposition séquentielle freine la résolution dans son ensemble
- Solution : Paralléliser la décomposition !

Décomposition parallèle

- Les workers décomposent le problème en sous-problèmes qui seront ensuite résolus par ces derniers
- Problème de répartition de charge. Or un travail bien distribué est bien parallélisable !
- Une première parallélisation naïve :
  - On a k workers et on veut décomposer en q sous-problèmes
  - On décompose dans un premier temps en k sous-problèmes pour que chaque worker décompose par la suite en q/w pour atteindre q
  - Problème : Mauvaise répartition de charge

Décomposition parallèle

- Autre solution (plus efficace): Faire des arrêts pour redistribuer le travail.
- Ex: On veut décomposer en 30 sspb/worker.
- Idée on fait des étapes On cherche à faire 5 sspb/worker, puis 10, puis 20 puis 30.
- Arrêt haut (rapide) = meilleure répartition du travail
- Inconvénient des arrêts : nécessite synchronisation => ralentit le processus

Les étapes intermédiaires

- Après tests, on a trouvé qu'il est préférable de réaliser les étapes suivantes :
  - On fait du séquentiel pour couper le problème en XXX sspb
  - On recherche Y1 = ??? sspb / worker
  - On recherche Y2 = ??? sspb / worker
  - On rechercher Y3 = 30 sspb / worker

- Remarque :
  - Pas besoin de synchronisation pour la dernière étape
  - C'est plutôt robuste et ne dépend pas du type de problème traité

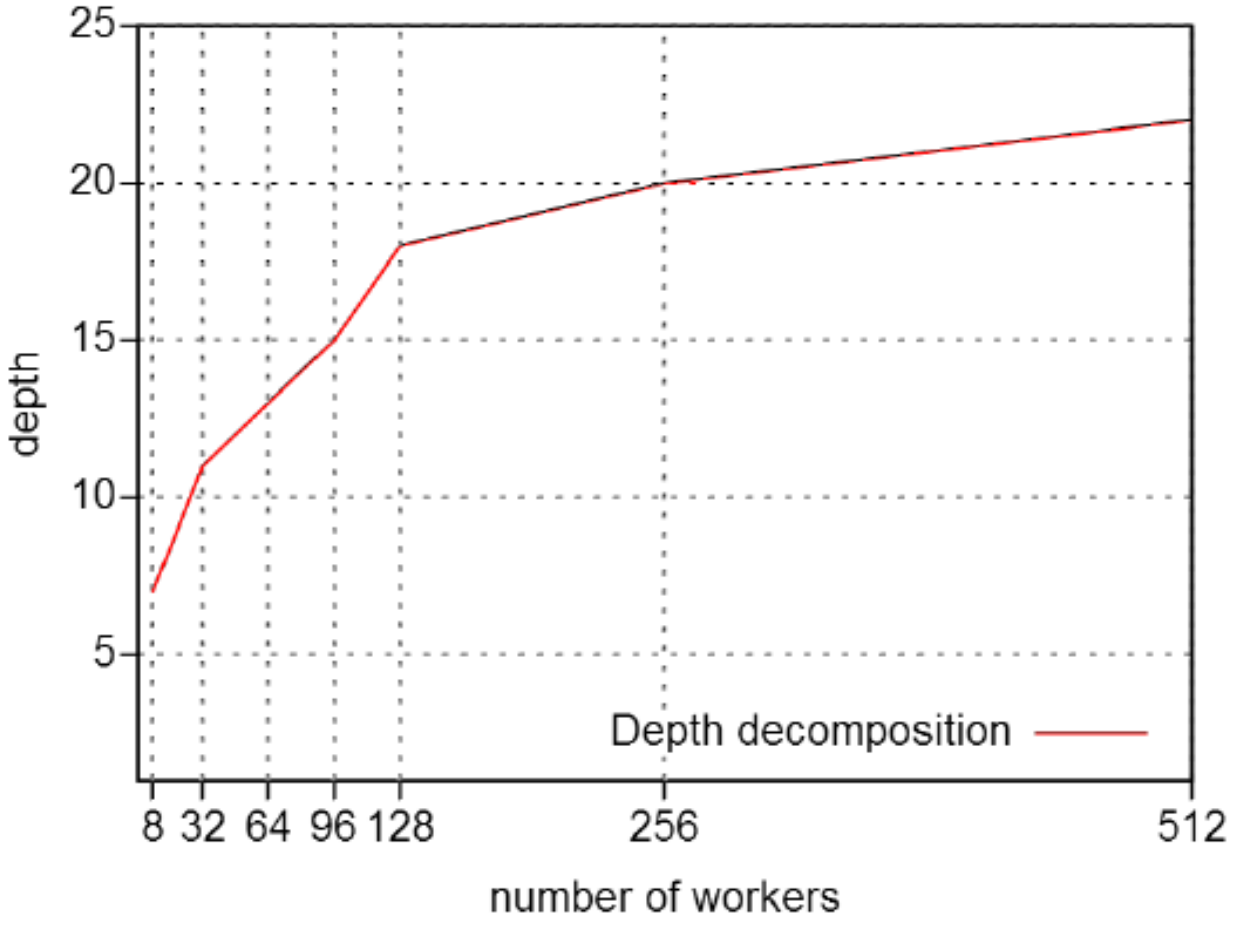
Résultats (1)

Poids de la décomposition séquentielle (96 workers)

Instance	$ratio_{dec/run}$	$speedup_{res}$
	p	s
allinterval_15	17%	83.5
magicsequence_40000	76%	87.8
sportsleague_10	49%	74.4
sb_sb_13_13_6_4	46%	82.4
quasigroup7_10	50%	76.5
non_non_fast_6	64%	80.9
golombruler_13	25%	94.2
warehouses	89%	219.9
setcovering	67%	74.6
2DLevelPacking_Class5_20_6	34%	230.9
depot_placement_att48_5	48%	74.9
depot_placement_rat99_5	58%	76.9
fastfood_ff58	69%	84.7
open_stacks_01_problem_15_15	60%	78.1
open_stacks_01_wbp_30_15_1	56%	72.2
sugiyama2_g5_7_7_7_7_2	63%	80.5
pattern_set_mining_k1_german-credit	61%	73.4
radiation_03	52%	70.6
bacp-7	91%	223.4
talent_scheduling_alt_film116	81%	97.8
geometric average (p or s)	54%	93.3

Résultats (2)

Évolution de la profondeur en fonction du nombre de sous-problèmes à générer (30 sous-problèmes par worker)



Résultats (3)

Comparaison des différentes décompositions (séquentielle vs parallèle (96 workers)

Instance	Seq.		$Dec_{//2}$		$Dec_{//1}$		Dec_{seq}	
	t_0	su_{res}	t_{dec}	su	t_{dec}	su	t_{dec}	su
	s	r	s	r	s	r	s	r
allinterval_15	220.0	83.5	0.3	75.1	0.6	69.1	3.1	38.2
magicsequence_40000	316.6	116.8	1.4	76.6	8.5	28.2	21.5	13.1
sportsleague_10	170.1	74.4	0.5	61.6	2.2	38.1	9.4	14.5
sb_sb_13_13_6_4	135.1	82.4	0.7	57.5	1.4	44.6	12.9	9.3
quasigroup7_10	287.0	76.5	0.6	65.6	3.8	38.1	13.1	17.0
non_non_fast_6	582.4	80.9	8.0	38.3	13.0	28.8	29.7	15.8
golombruler_13	1303.9	94.2	0.3	92.1	4.5	71.0	8.9	57.2
warehouses	139.4	219.9	0.7	108.3	4.9	25.1	14.8	9.1
setcovering	88.4	74.6	0.2	65.3	2.4	24.4	4.7	15.1
depot_placement_att48_5	113.3	74.9	0.4	60.3	1.4	38.9	6.2	14.7
depot_placement_rat99_5	20.3	76.9	0.4	30.9	0.4	32.1	6.9	2.9
fastfood_ff58	22.3	84.7	0.6	27.4	0.6	26.4	5.7	3.7
open_stacks_01_problem_15_15	99.1	78.1	0.3	62.1	1.9	31.4	14.1	6.4
open_stacks_01_wbp_30_15_1	180.7	72.2	0.9	53.0	3.2	31.9	21.7	7.5
sugiyama2_g5_7_7_7_7_2	237.5	80.5	1.3	56.5	5.0	29.7	33.0	6.6
pattern_set_mining_k1_german-credit	103.9	73.4	0.9	44.7	2.2	28.7	11.6	8.0
radiation_03	108.1	70.6	0.3	59.0	1.7	33.9	10.6	8.9
talent_scheduling_alt_film116	243.3	97.8	2.5	48.3	10.5	18.7	31.1	7.3
total(s) and geom. average(r)	4371.1	85.8	**20.2**	**57.0**	68.1	33.5	259.0	10.7

Résultats (4)

Avantages du stop dans la décomposition parallèle (96 workers)

Instance	2 étapes				
	3	4	5	6	7
	s	*s*	*s*	*s*	*s*
sb_sb_13_13_6_4	16.5	25.0	18.3	17.5	13.1
sugiyama	13.1	24.4	26.6	25.1	21.4
patternSetMiningGerman	8.1	9.0	12.7	14.4	10.5
radiation_03	18.6	24.8	35.2	26.9	17.3
geometric average (s)	13.45	19.22	**21.6**	20.3	15.0

Résultats (5)

Work stealing vs EPS (512 workers)

Instance	Seq.	Work stealing		EPS	
	t	t	s	t	s
market_split_s5-02	3314.4	-	-	8.2	**405.9**
market_split_u5-09	3266.6	-	-	7.9	**411.8**
market_split_s5-06	3183.9	-	-	8.3	**384.0**
prop_stress_0600	2729.2	1426.4	1.9	14.1	**193.1**
nmseq_400	2505.8	-	-	10.4	**240.4**
prop_stress_0500	1350.6	670.0	2.0	8.4	**161.6**
fillomino_18	763.9	-	-	5.1	**150.7**
steiner-triples_09	604.9	79.0	7.7	1.8	**332.0**
nmseq_300	555.3	-	-	4.2	**131.7**
golombruler_13	1303.9	15.5	83.9	3.0	**427.9**
cc_base_mzn_rnd_test.11	3279.5	-	-	26.8	**122.6**
ghoulomb_3-7-20	2993.8	575.4	5.2	22.8	**131.1**
pattern_set_mining_k1_yeast	2871.3	299.8	9.6	15.7	**183.2**
still_life_free_8x8	2808.9	1672.8	1.7	16.8	**166.9**
bacp-6	2763.3	330.1	8.4	7.3	**378.9**
depot_placement_st70_6	2665.1	1902.9	1.4	11.3	**235.1**
open_stacks_01_wbp_20_20_1	1523.2	153.9	9.9	9.5	**160.8**
bacp-27	1499.7	579.6	2.6	4.6	**326.5**
still_life_still_life_9	1145.1	140.1	8.2	6.3	**182.9**
talent_scheduling_alt_film117	566.1	95.5	5.9	3.2	**175.8**
total (t) or geometric average (s)	41694.5	7941	5.4	195.7	**223.9**

Conclusion & Perspectives

- Décomposition parallèle
- EPS passe à l'échelle sur des centaines de cœurs
- Machines distribuées (machines hétérogènes)
- Cloud : Windows Azure
- Estimations de la résolution
- Passage à l'échelle dynamique

Merci de votre attention !
Des questions ?