Embarrassingly Parallel Search

Itération 1 : Exploration de la piste EPS pour la parallélisation HBFS dans Toulbar2

Lieu : Unité Mathématiques et Informatique Appliquées de Toulouse, INRA, Castanet-Tolosan

Dates : du 18/03/2018 au 15/09/2018

Abdelkader BELDJILALI

3 juillet 2019

Table des matières

1 Généralités sur la recherche parallèle embarrassante (EPS)

Le terme pittoresque d'embarrasingly Parallel computation est dû à Geoffrey Fox et date de 1995. Ce paradigme désigne une parallélisation naturelle qui consiste à découper un programme, si ce dernier le permet, en parties indépendantes et à les assigner à des "workers" c'est à dire aux c œurs ou processeurs disponibles.

2 Génération a priori des sous problèmes

2.0.1 Génération par produit cartésien

Un programme C++ a été créé qui produit les assignations partielles donnant ainsi un nombre de sous problèmes tel que $|A_{k-1}| < q \leq |A_k|$ où q est la valeur cible du nombre de sous problèmes désirés $|A_k|$ correspond au cardinal du produit cartésien des domaines de k variables. cf **[Regin2013]**.

```
./generate name of input wcsp file, space,
name of output subproblems file,space,
approximate number of sub problem
Example: ./generate 404.wcsp job_file 250
Parallel command : cat job_file | parallel -j+0 --eta -k ./toulbar2 404.wcsp {}
```

Le programme génère des assignation/décisions de type égalité comme ci-dessous :

```
256 sous problèmes correspondant aux 4 premières variables
du problème 404 de domaines {0,1,2,3}

-x=",0=0,1=0,2=0,3=0"
-x=",0=0,1=0,2=0,3=1"
-x=",0=0,1=0,2=0,3=2"
-x=",0=0,1=0,2=0,3=3"
-x=",0=0,1=0,2=1,3=0"
-x=",0=0,1=0,2=1,3=1"
-x=",0=0,1=0,2=1,3=2"
...

qui correspondent au produit cartésien calculé via la fonction:

/** \brief compute the cartesian product of a vector vectors
domains : set of domains dm represented by c++ vectors
*/
vector<vector<long>> cartProd (const vector<vector<long>>& domains)
{
vector<vector<long>> vv = {{}}, tmp; // init vector of vectors
for (const vector<long> & dm : domains) { // for each domain dm in the set of domains dm
tmp= {};
for (const vector<long> & v : vv) { // for each sub vector v in vv (v is {} at first)
for (const long value : dm) { // for each value in domain dm
```

```
tmp.push_back(v); // add domain v in tmp here v ={} at first
// cout << "x ="<<x << endl;
tmp.back().push_back(value); // add value to the last domain in tmp, here v={} tmp ={{0},{

}
}

vv.swap(tmp); // vv become tmp and tmp become vv
}
//cout<< tmp << endl;
return vv;
}
```

2.0.2 Résultats

Les résultats obtenus sur les problèmes au format wcsp : 404, scen06 sont décevants. Le temps d'exécution est largement majoré par rapport au programme séquentiel.

3 Génération des sous problèmes via hbfs

L'algorithme HBFS de Toulbar2 utilise une file de priorité, nommée open, pour stocker les nœuds ouverts qui constituent les sous problèmes qui restent à résoudre. Donc naturellement, en fixant une taille maximale pour open qui lorsqu'elle est atteinte ou dépassée aboutit à un dump des sous problème et à la terminaison du programme, on obtient une série de sous problèmes de *bonne qualité* au sens où ils ont pu bénéficier des optimisations du programme séquentiel HBFS.

En outre de manière à obtenir un load balancing correct les sous problèmes doivent être de taille suffisamment petits. Pour ce faire, on se fixe une valeur N, qui donne de bons résultats, égale à environ 30 fois le nombre de cœurs[**Regin2013**]. Lorsque la taille de la file open dépasse cette valeur, on extrait les nœds et on reconstitue l'affectation partielle associée. On utilise ensuite cette liste d'affectations partielles en tant que paramètre dans toulbar2 pour produire un ensemble de processus qu'il s'agit d'exécuter en parallèle par exemple en utilisant GNU parallel.

Pour le problème 404.wcsp, les courbes ci-dessous décrivent les résultats obtenus qui exhibent un speedup [1] maximum de 4.9 sur une machine 4/8, 4 cœrs réels, 8 avec l'hyperthreading avec un nombre de sous problème par cœur entre 28 et 31. On retrouve le résultat de Régin et al. mais ce dernier est probablement architecture et problème dépendant. Dans le cas présent, le calculateur est mono processeur (i.e. socket unique).

1. Le speedup est ici défini comme le rapport entre le temps d'exécution chronométré du programme parallèle et le temps du programme séquentiel.

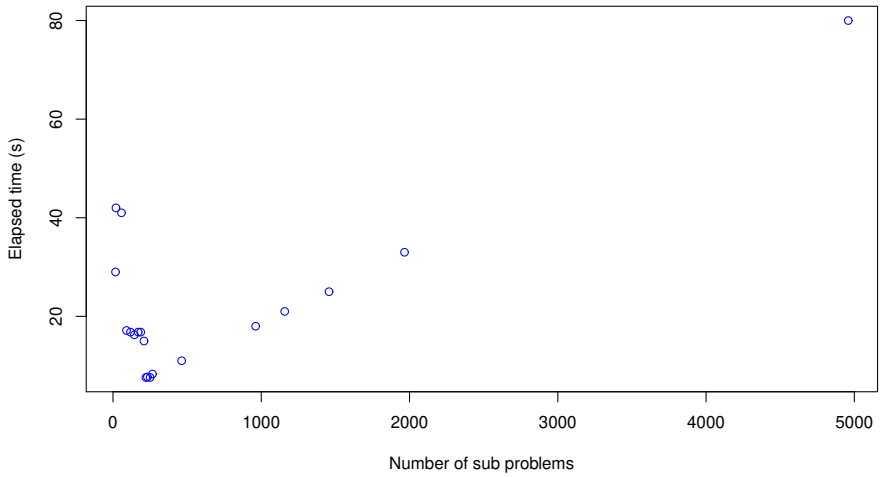

FIGURE 1 – a minimum for the elapse time is observed near 200 sub problems

Un zoom de la figure 10 donne une valeur plus précise de l'optimum.

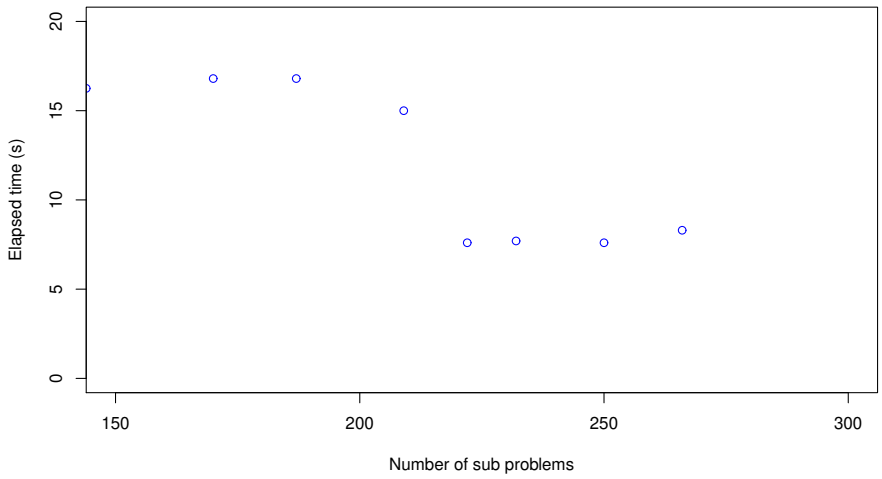

FIGURE 2 – Minimum clockwall time is reached for 220 to 250 subproblems

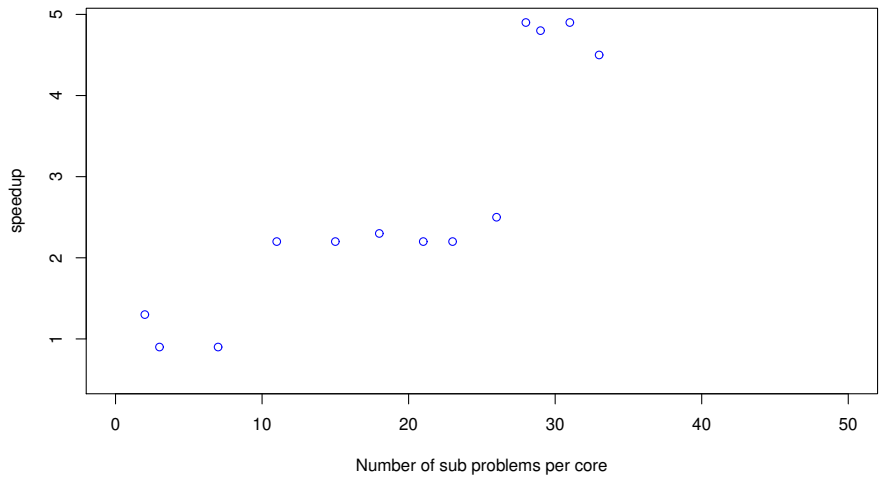

FIGURE 3 – Maximum speedup observé pour des valeurs de 28 à 31 voire 33 sous problème par cœur

4 Eléments descriptifs des résultats obtenus

4.1 Performances

4.1.1 Méthode de mesure

1. Toulbar2 est exécuté en séquentiel dans le dossier bin/Linux avec la commande ./toulbar2 pb.wcsp pour obtenir le temps réel qui serait chronométré du lancement à la terminaison du programme.
2. On lance Toulbar2 -eps pb.wcsp pour produire le script eps.sh et les sous problèmes dans subProblem.txt. Les sous problèmes sont constitués d'une suite de décisions associées au branchement dans l'arbre de recherche qu'on peut considérer comme des affectations partielles. A titre d'exemple, l'option à donner à toulbar2 peut être de la forme :

   ```
   -x=",44#0,46=1,60=1,45=1,50=1,71=0,58=1,88=1,61#1,59=3,49#1,9#3"
   ```

3. On utilise ensuite gnu parallel en lui passant en arguments les processus à exécuter en parallèle. L'option -j indique le nombre maximum de processus à exécuter en parallèle à un instant donné sur les N = 240. Sur une machine à 8 cœrs, on pourrait penser que 8 processus maximum donneraient un résultat optimum mais il a été observé sur une machine donnée et le problème 404.wcsp que le temps d'exécution est optimal avec $20 = 2.5 * 8$ processus en parallèle. Ce facteur de 2.5 optimal dans le cas précédent est malheureusement architecture et problème dépendant d'après quelques expérimentations. Ce comportement est probablement dû à la manière dont gnu parallel est programmé, au scheduler linux et à l'architecture.

cat subProblems.txt | time parallel -j20 –eta ./toulbar2 404.wcsp -ub=114

4.2 Nombre de nœuds totals traités vs nombre de nœuds ouverts

La file open varie en taille au cours de l'exécution du programme. Des nœuds sont retirés de la file pour alimenter l'algorithme de recherche en profondeur (DFS) et ce dernier retourne des nœuds ouverts qui alimentent la liste open. Les expériences faites sur différents problèmes au format wcsp que lors de la génération des sous problèmes seules une minorité de n œuds ont été traités en totalités : n œuds fermés. Le problème scen06.wcsp par exemple, avec un nombre de problèmes cibles fixé à 240, fournit 257 nœds ouverts pour 260 n œds explorés au total. rapport 257/260. 404.wcsp -> 251/264, nug12.wcsp -> 243/286.

Le temps de génération des sous problèmes ne dépasse pas la demi-seconde sur une processeur 8 cœurs hyperthreadé : 4 cœrs réels, 2 threads par cœur. Le système d'exploitation voit 8 cœurs donc la valeur qui déclenchera l'extraction des sous problèmes est $N = 30 * 8 = 240$. Ce temps est négligeable devant la phase parallèle.

4.3 Optimisation de la génération

Seuls les sous problèmes pouvant donner lieu à solution sont générés : condition d'élagage borne inf lb >= borne sup ub.

En outre, la meilleure solution ub trouvée au moment de la génération des sous problèmes (SP), ou incumbent solution, est utilisée dans les sous problèmes à exécuter en parallèle. Pour le problème 404.wcsp, la résolution sans utiliser l'option -ub=114 de toulbar2 pour tenir compte de la meilleure solution trouvée lors de la génération des SP est de 17s contre 7.5 secondes avec ub. Cette amélioration est bien sûre elle aussi problème dépendante.

L'intérêt d'utiliser le ub obtenu à la génération des SP est illustré sur le problème 404 par la figure 4 ci-dessous. Seule une minorité de sous problème trouve la valeur optimale globale, les autres n'aboutissent qu'à une solution sub optimale imposée par leur assignation partielle qui ne permet pas d'atteindre l'optimum global. D'où, d'une part d'utiliser toute information préalable sur le ub, et d'autre part, d'échanger dynamiquement les ub trouvés entre les processus. Ce qui peut se faire soit par passage de messages entre processus ou par mémoire partagée avec mutuelle exclusion. Cependant le principe de l'EPS ne prévoit pas d'échanges entre processus durant le traitement.

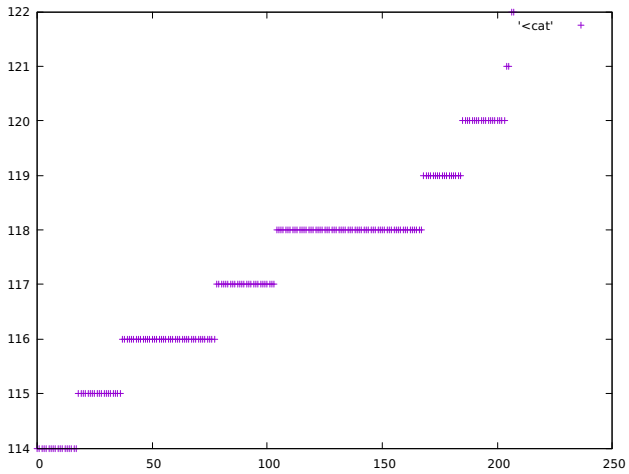

FIGURE 4 – Valeur de l'optimum trouvé par les 208 SP. Seule une minorité trouve l'optimum 114.

4.4 Distribution en nombre de variables non assignées

Le % de nombre de variables non assignées pour les problèmes 404.wcsp et scen06.wcsp fournissent une indication sur la taille du problème si les domaines des variables sont de taille comparable.

exemple : pour le problème 404 qui compte nbVar=100 variables , on calcule le % comme suit : ((nbVar - (nb.last-nb.first))/nbVar) * 100. Pour scen06, on trouve certains % négatifs !

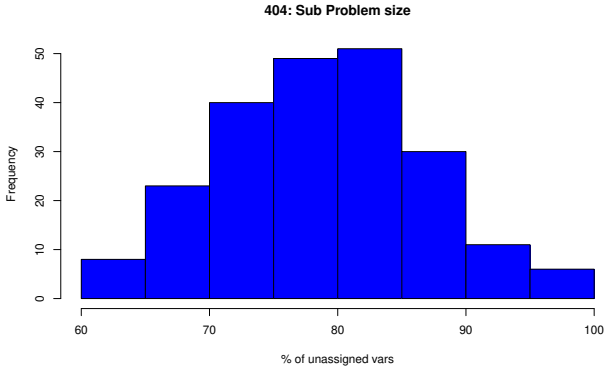

FIGURE 5 – Taille des sous problèmes en nombre de variables

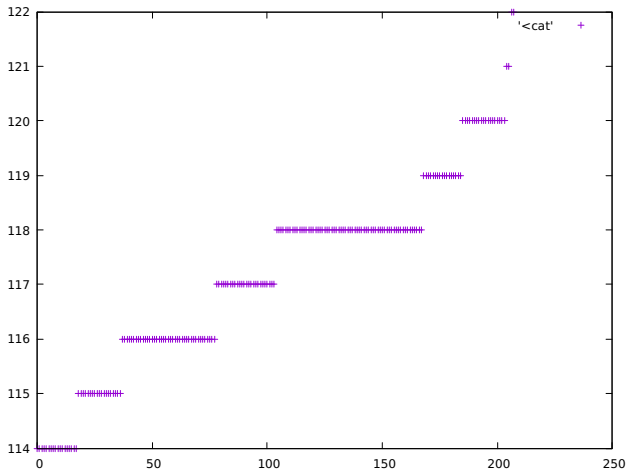

FIGURE 6 – Taille des sous problèmes en nombre de variables

ff

```
 SubProblem    size...
Min.   :61.00   Mode:logical
1st Qu.:74.00   NA's:218
Median :79.00
Mean   :79.36
3rd Qu.:85.00
Max.   :98.00
```

FIGURE 7 – Taille des sous problèmes en nombre de variables

```
> summary(size_scen06)
SubProblem    size...
Min.   :-4.00   Mode:logical
1st Qu.:32.00   NA's:252
```

```
Median :55.00
Mean   :53.37
3rd Qu.:76.00
Max.   :98.00
```

4.5 Distribution en complexité des sous problèmes

Les complexités sont évaluées sous la forme du cardinal des produits cartésiens des domaines des variables non assignées, c'est à dire des variables non concernées par des décisions de type : $=$, $\#$, $<$, $>$. Si les domaines sont réduits par propagation de contrainte au moment de l'extraction des sous problèmes alors ces complexités sont sur évaluées. Les résultats ci-dessous montre une grande disparité des complexités théoriques avec une majorité de problèmes complexes de l'ordre de $10^3 7$ à $10^3 8$.

```
   Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
7.556e+22 5.071e+30 6.490e+32 6.535e+36 4.154e+34 3.403e+38
```

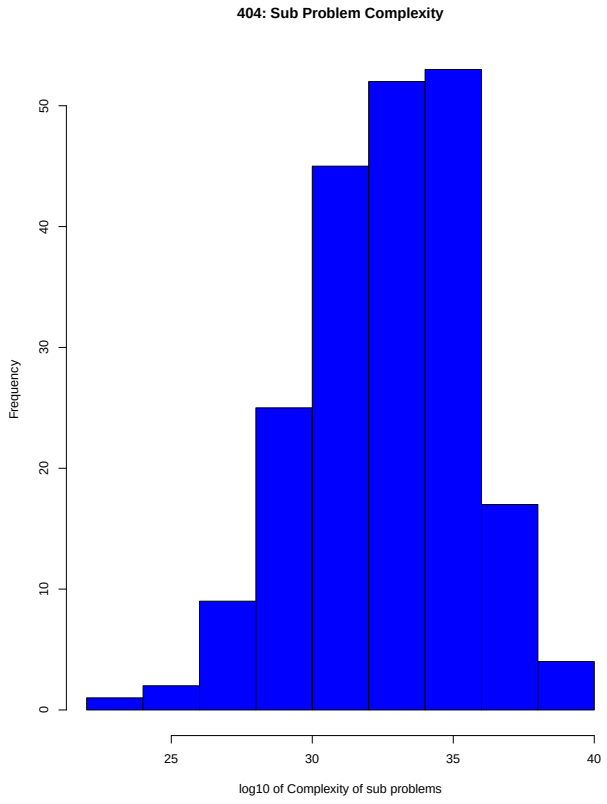

FIGURE 8 – Distibution du log10 des complexités

La complexité théorique est elle liée au temps de résolution du sous problème ? Ce qui permettrait de prévoir le temps de résolution. Le schéma ci-dessous indique que pour les complexités faibles, inférieures à 33 en échelle logarithmique le temps de résolution est faible. Par contre, au delà, les temps d'exécution des sous problèmes s'allongent et la prévision des temps associés est hasardeuse. On peut éventuellement inférer une application pratique pour la génération des sous problèmes à savoir calculer la complexité théorique des sous problèmes et remettre dans la file open ceux dont le CT dépassent un certain seuil. Encore faut-il pouvoir déterminer ce seuil et trouver le bon compromis entre la phase séquentielle de génération de SP et la phase parallèle.

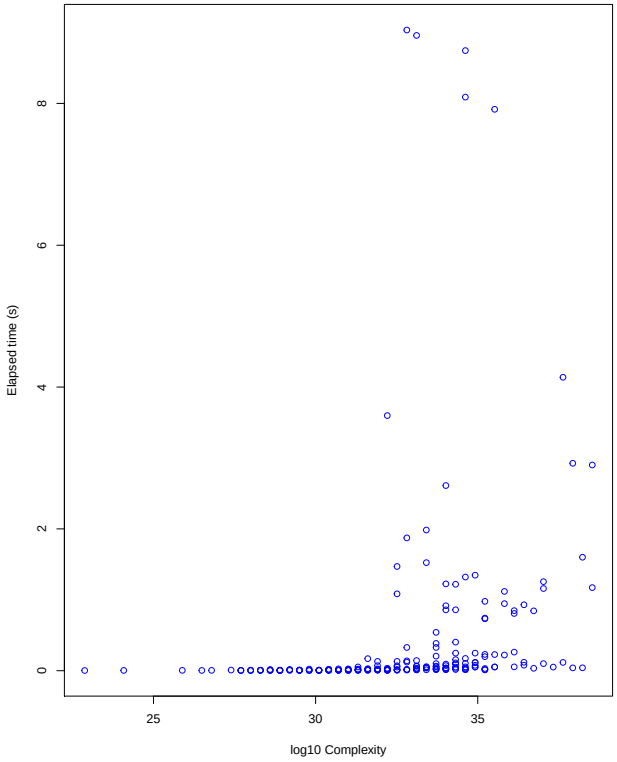

FIGURE 9 – time versus complexity of sub problems

4.6 Temps individuel de résolution des sous problèmes

Les mesures sont effectuées sur le problème 404. Le temps total d'exécution de toulbar2 est de l'ordre de 17s sans utiliser le ub produit par lors de la production des sous problèmes. La majorité des sous problèmes prennent une fraction de seconde. Seuls quelques sous problèmes prennent plus de 7s. Une génération de sous problèmes de taille plus homogène permettrait sans doute d'améliorer le speedup en évitant que des cœurs ne soient inoccupés si le nombre de "petits" problèmes ne sont pas suffisamment nombreux pour compenser le temps nécessaire aux "gros" problèmes. Dans le problème 404, la visualisation de la charge

des cœurs montre que ces derniers restent occupés jusqu'en fin de programme sans doute du fait d'une distribution de "taille" des sous problèmes qui permet de tenir occupé chaque cœr. (nb : similitude avec le problème du bin packing à ceci près qu'on ne cherche pas à minimiser le nombre de bin/cœrs utilisés mais à minimiser les différences de "remplissage" ; la dimension spatiale du bin packing étant remplacée par la dimension temporelle : métriques analogues).

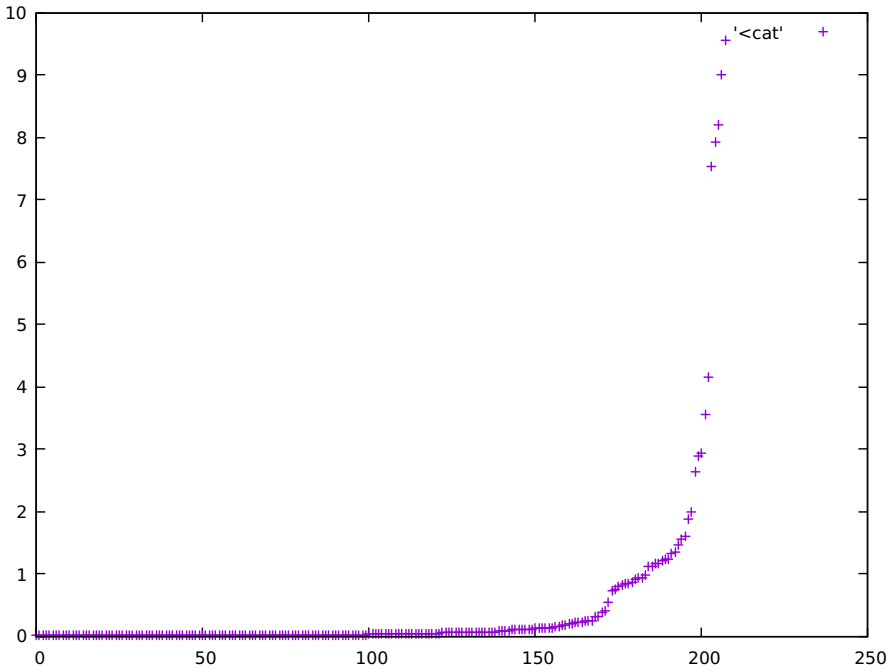

FIGURE 10 – courbe des temps de résolution en fonction des numéros des 208 SP du problème 404 sans utilisation de la borne ub calculée lors de la génération des SP

En récupérant et en utilisant l'incumbent solution ub obtenue au moment du dump, les performances sur le problème 404 sont fortement améliorées : gain de temps sur les problèmes les plus gros e.g. le 5ème sous problème passe de 9s à 4s. Ce qui au passage montre l'importance de pouvoir partager dynamiquement et facilement le ub entre les processus.

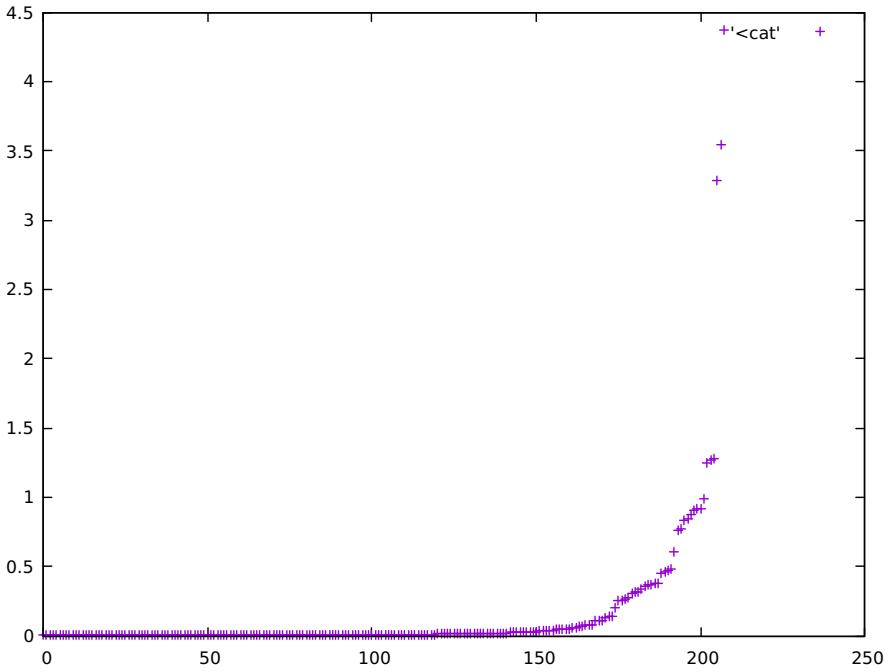

FIGURE 11 – courbe des temps de résolution en fonction des numéros des 208 SP du problème 404 avec -ub=114

4.7 Performances globales de la parallélisation : speed-up

Si des speed-up de 4.9 sur une machine 4/8 ont pu être atteint sur le pb 404.wcsp, les résultats sont décevants pour scen06.wcsp (speed-up = 1.2 à 2). Par ailleurs, sur un serveur à 2 processeurs avec 12 cœrs réels par processeur physique, les résultats ne sont pas améliorés en terme de speed-up, voire d'échelonnabilité (scalability) : perf moins bonnes des cœrs individuels, com. plus lentes entre les deux processeurs physiques (sockets), caches et processeurs moins bien exploités pour une cible de 30 SP/c.

Cependant, des résultats plus intéressants sont obtenus en fixant un seuil cible de 150 pour le nombre de nœds dans la file open au momentdu dump. Ce qui correspond plus ou moins à 6 SP/c.

Les problèmes Graph11, capmp1 et pedigree18 sont les plus récalcitrants avec des speedup inférieurs à 1 et des temps de résolution en mode parallèle importants.

4.7.1 Benchmarks sur serveur 4/8 : 4 cœurs réels, 8 avec multithreading

Benchmark sur architecture 4-8 coeurs

wcsp file	Time (s) of subproblem (sp) generation	Serial Time (s)	parallel time (s)	speedup	% CPU	obs
1PGB.11p.19aa.usingEref_self_digit2	1,6	4,1	10,4	0,4	740	53 sub problems
graph11	1,0	574,0	trop long*	< 1		464 sub problems
capmp1	152,3	446,6	397,4	1,1	787	172 sp – speed up total : 0,81
scen06	0,6	944,0	792,0	1,2	610	257 sp
capmo1	5,3	14,1	18,9	0,7	744	85 sub problem
pedigree18	0,3	348,0		<1		
pedigree7	0,3	2,7	13,7	0,2	583	
nug12	0,3	202,0	52,8	3,8	467	
404	0,3	35,1	7,7	4,6	530	

*pas de résultats après + de 12 heures

Page 1

FIGURE 12 – Benchmarks sur architecture 4-8 cœurs

4.7.2 Benchmarks sur serveur 2*12 cœurs réels

Benchmark sur architecture 24 coeurs

wcsp file	Time (s) of subproble m (sp) generation	Serial Time (s)	Parallel time (s)	speedup	% CPU	obs	Gub : ub at generation time	optimal ub	gap	relative gap (%)
1PGB.11p.19	2,1	4,3	5,1	0,83	1388	52 sp	286561	286135	426	0,1
graph11	1,7	325,0	1294,1	0,25	138	488 sp* -A	30703	3080	27623	896,9
capmp1	61,2	266,0	101,9	2,61	2149	172 sp – spe	2491566	2460099	31467	1,3
scen06	0,6	945,0	437,0	2,16	619	175 sp	8433	3389	5044	148,8
capmo1	4,6	14,2	3,2	4,46	276	85 sp	1163083	1156908	6175	0,5
pedigree18	0,3	340,1	trop long	<1		126 sp	768569542	620 119 799	148 449 743	23,9
pedigree7	0,9	2,4	4,5	0,54	393	82 sp	539491253	295341819	244149434	82,7
pedigree7	0,9	2,4	6,7	0,36	397	137 sp	517834901	295341819	222493082	75,3
nug12	1,0	207,3	49,5	4,19	385	716 sp	608,0	578,0	30,0	5,2
nug12	0,9	207,0	35,9	5,77	510	150 sp	620	578	42	7,3
404.wcsp	0,4	36,7	20,9	1,8	203	699 sp -j+0	114	114	0	0
404,wcsp	0,3	37,1	4,8	7,73	541	141 sp -j+0	120	114	6	5,3

cat subProblems.txt | time parallel -j24 --eta -k ./toulbar2 404.wcsp -ub=114 {}
cat subProblems.txt | time parallel -j24 --eta -k ./toulbar2 nug12.wcsp -ub=608 {}
cat subProblems.txt | time parallel -j24 --eta -k ./toulbar2 graph11.wcsp.xz -A -ub=30703 {}
Machine 2 * 12 cores : sullo avec cores partiellement occupés par d'autres process

graph11 remplit très rapidement la file open
temps bcp trop long : programme interrompu
Le speed varie beaucoup avec le nombre de sous problèmes. Cf mesures multiples notamment 404.wcsp
%CPU = (temps cumulé des processus + temps system i.e. process en mode noyau)/ elapse time –
donne une inforamtion sur l'utilisation des ressources CPU. Ici Majoré par 2400 %

pour les problemes cap, le temps de preprocessing est très lent d'où l'absence de gain.
pour pedigree7, le probleme vient peut etre d'un mauvais majorant ?

Page 1

FIGURE 13 – Benchmarks sur architecture 24 cœurs

5 Conclusion

Globalement, l'itération 1 (EPS) ne donne pas de résultats satisfaisants en pratique malgré des speedups intéressants dans certains cas. En effet, il apparaît que le nombre de sous problèmes générés impacte de façon importante sur le speedup en fonction de l'architecture : alors que les meilleures performances sur une machine 4/8 sont obtenues en générant environ 30 SP/cœur (nbProcess.txt 240) avec option -j20, sur un serveur 24 cœurs, les expériences effectués montrent de meilleurs speedup vers 6 SP/cœur (nbProcess.txt 240) avec option -j24. Les processeurs sont aussi mieux exploités. En effet, les %CPU sont plus faibles avec 30 sp/c, avec 6 sp/c ils sont plus élevés (500% pour le 404). On reste loin des 2400 % théoriques sur une machine 24 cœurs mais à sa décharge un process matlab multithreadé était en cours d'exécution. La mesure reste cependant pertinente car le temps séquentiel est mesuré dans les mêmes conditions. Sur ce serveur, l'augmentation du nombre max de jobs à lancer en parallèle fixés par l'option "-j " vs le nombre de cœrs limite les performances.

L'itération 2 MPI/Master-Worker pourrait s'avérer plus prometteuse en évitant les paramétrages de création de SP et surtout d'échanger l'incumbent solution dynamiquement.