TP1 - OSEM

Ilyas Toumlilt
Yassine Aroua

12/11/2015

Introduction :

Ce rapport propose une vérification expérimentale de l'efficacité de la distribution d'une structure de données en répartissant les traitements, dans une architecture Many-Cores à accès mémoire non uniformes : cc-NUMA.

Cette étude a été menée sur TSAR, un simulateur de processeur many-cores précis au cycle près et au bit, pouvant être configuré jusqu'à 256 processeurs, et sur ALMOS, un système d'exploitation conçu par Ghassan Almaless dans le cadre de sa thèse, spécifiquement pour TSAR et qui repose sur le même paradigme de programmation en mémoire partagée que d'autres systèmes monolithiques tels que Linux ou BSD.

Pour pouvoir mesurer la répartition de traitements respectant la localité des accès mémoire, nous avons décidé, dans le cadre de ce TP, de réaliser une application calculant de manière équitable sur un nombre défini (pour chaque exécution) de processus, la taille en mots d'un texte donné. Une application qu'on a décidé de nommer : dwc (distributed words counter).

De manière générale, ce TP propose de vérifier expérimentalement s'il est possible que la conception du noyau d'ALMOS et les applications actuelles, reposant sur la notion de threads et de mémoire partagée, puissent passer à l'échelle en nombre de coeurs.

Mise en place de l'environement :

Avant de commencer l'implémentation de notre programme, il a d'abord fallut mettre en place l'environement cible pour notre exécution, en l'occurence ici la distribution ALMOS pour TSAR.

On commence donc par télécharger la dernière distribution stable d'ALMOS à l'adresse :

.

```
https://www-soc.lip6.fr/trac/almos/chrome/site/almos-tsar-
    mipsel-1.0.tbz2
```

Ensuite, il faudra décompresser l'archive :

.

```
$ tar jxf almos-tsar-mipsel-1.0.tbz2
```

Dans notre cas, on a eu quelques soucis liés à l'absence de packets dont dépendait l'utilisation du simulateur, on a donc eu à installer la liste de packets suivante :

.

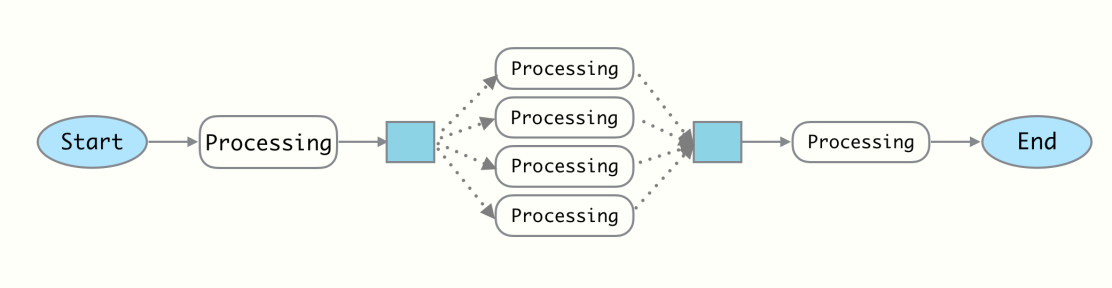
```
libc6-i386 lib32stdc++6 lib32gcc1 lib32ncurses5 lib32z1 xterm
```

Les répertoires importants dans notre cas sont :

```
dans almos-tsar-mipsel-1.0
-> ./test/pf1 : // contient le makefile launcheur de TSAR (
    make sim<nb_clusters> )
-> ./apps : // repertoire contenant les sources de nos
    applications
```

Description de l'application :

DWC (Distributed Words Counter) est une application de comptage distribué de nombre de mots dans un texte. Le choix de cette application, est motivé par la nécessité d'avoir une application parallèle dont on évaluera la performance (speedup). Cette application sera écrite en C, et utilisera l'API POSIX Threads pour exprimer le traitement parallèle, elle suivra le schèma de traitement illustré par la figure :



La phase d'initialisation permet de remplir la structure de calcul, représentée dans notre cas par un tableau de mots générés aléatoirement et dont la taille sera définie à l'avance (parce qu'on veut pouvoir manipuler la taille de mots, histoire de pouvoir évaluer les performances sur plusieurs plages de mots). On lancera ensuite autant de taches "workers" qu'il y a de processeurs dans la platforme matérielle, chaque tache "worker" aura sa plage de travail dans le tableau de mots, et produira un tableau correspondant au nombre de mots d'indice i du tableau de lettres. (ainsi array[3] correspondra au nombre de mots de taille 3). Dès que les "workers" auront fini leur calcul, le main rentrera en phase de merge pour rassembler tous les tableaux de résultats et faire les sommes de nombres de mots.

Une exécution de l'application rend le temps de traitement en nombre de cycles par phase du programme. on exécutera le programme, sur les mêmes données et pour la même configuration séquentiellement et de manière distribuée (pour les benchs).

Notre rendu :

L'archive rendue contient :

.

```
-> ./apps/dwc/* // sources du programme dwc
-> ./apps/dwc/makefile // makefile le compilant
-> ./execMe.sh // script compile le programme et lance une
   serie de simulations comme decrit dans le paragraphe
   precedant
```

Cette archive est à copier à la source de la distibution, c'est à dire dans le dossier almos-tsar-mipsel-1.0.

Description du code implémenté :

Contenus dans **./apps/dwc/** les sources du code de DWC se distinguent en 3 parties :

words_generator.c/.h : Un générateur de mots aléatoires, dont la fonction d'entrée

```
char** random_text_generator(int nbWords, int maxWordSize);
```

prend en paramètre le nombre de mots à générer et la taille maximale que peut prendre un mot, elle retournera un tableau de nbWords mots de tailles entre 0 et maxWordSize.
Ce fichier contient également deux autre fonctions pour désallouer un tableau de mots, et pour l'afficher :

```
void free_text(char** text, int nbWords);
void print_text(char** text, int nbWords);
```

distributed_words_counter.c/.h : Le coeur de l'application, contenant notament les deux fonctions principales :

```
struct dwc_bench {
  clock_t mono_time; /* temps passe dans la partie
     sequentielle */
  clock_t multi_time; /* temps passe dans la partie parallele
     */
};
struct dwc_bench* super_count(char** text, int nbThreads,
                int nbWords, int maxWordSize);
clock_t mono_count(char** text, int nbWords, int maxWordSize);
```

Ce sont ces deux fonctions (les seules publiques dans le .h) qui seront appellées pour exécuter nos calculs, les deux prennent en paramètre un tableau de mots (idéalement générés par le words_generator), la taille de ce tableau (le nombre de mots), et la taille max d'un mot. La différence entre les deux c'est que la première effectue un calcul distribué (le nombre de threads sera dans ce cas passé en param) et retourne donc de manière distincte le temps passé dans la partie parallèle et le temps passé dans la partie séquentielle, tandis que la deuxième fonction effectue le calcul de manière séquentielle et ne rend donc qu'une seule durée, celle de tout le traitement.
Le calcul parallèle ce fait à travers l'utilisation de la biliothèque Threads POSIX, où - comme décrit plus haut - chaque threads effectuera son propre calcul sur sa propre plage de données, le processus principal s'occupe donc de la création des Threads et de l'affectation de la plage de données dans le tableau, il attendra ensuite que tous les workers finissent le traitement pour faire la somme des calculs collectés.

Le fichier .c contient d'autres fonctions telles que celles qui gèrent la mémoire et celles qui seront affectés à chaque Thread.

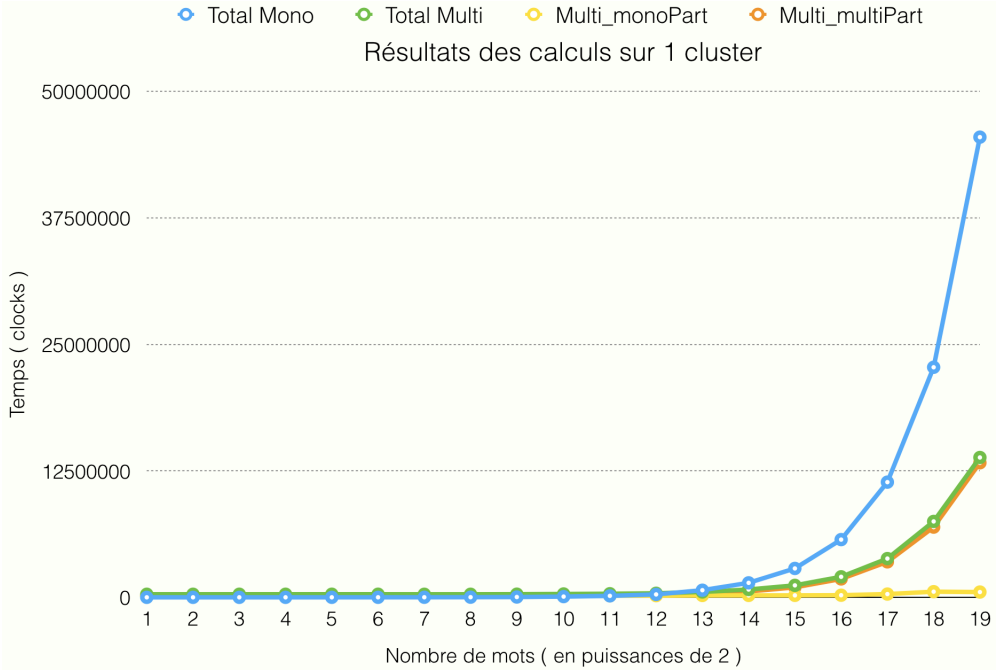
main.c : Point d'entrée principal de l'application, il prend en argument le nombre de mots à traiter, la taille maximale d'un mot est par contre déclarée en dur dans le code (32), il récupère d'abord le nombre de processeurs, grace à l'appel **sysconf(_SC_NPROCESSORS_ONLN)**, gènère un tableau de mots aléatoires, et lance les deux calculs : parallèle puis séquentiel. A la fin de son exécution il affiche les durées de calculs retournées par les deux fonctions de comptage.
Le but étant de calculer à chacune des exécutions de ce main les temps de calculs (séquentiel + distribué) sur une même portion de nombre de mots, et une même configuration du simulateur. On verra dans le paragraphe suivant qu'on va lancer des benchs sur plusieurs plages de mots différentes, et sur plusieurs configurations du simulateur.

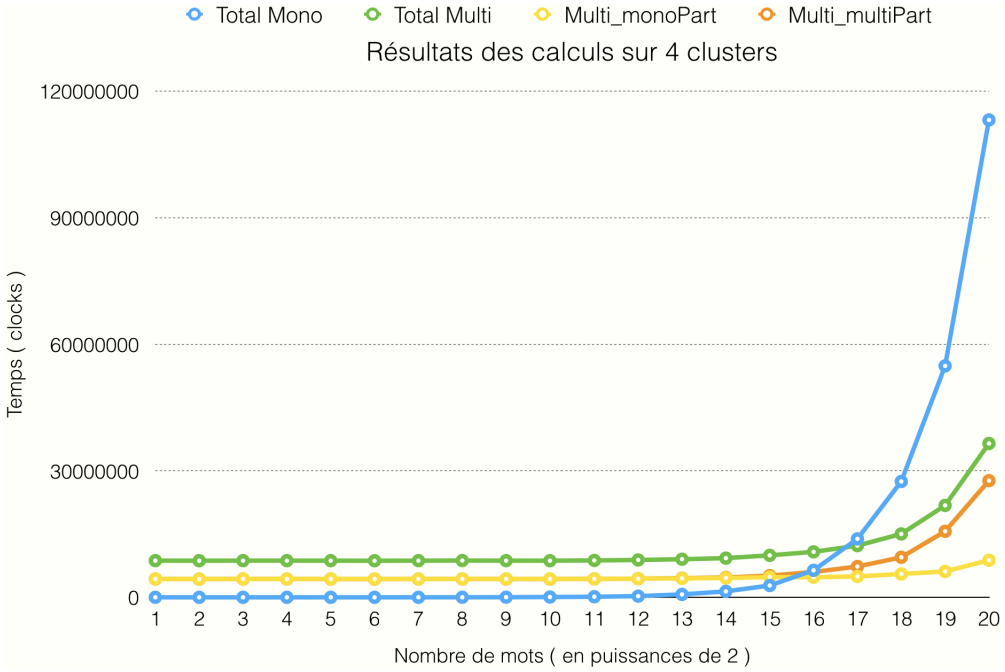
Description de l'expérience :

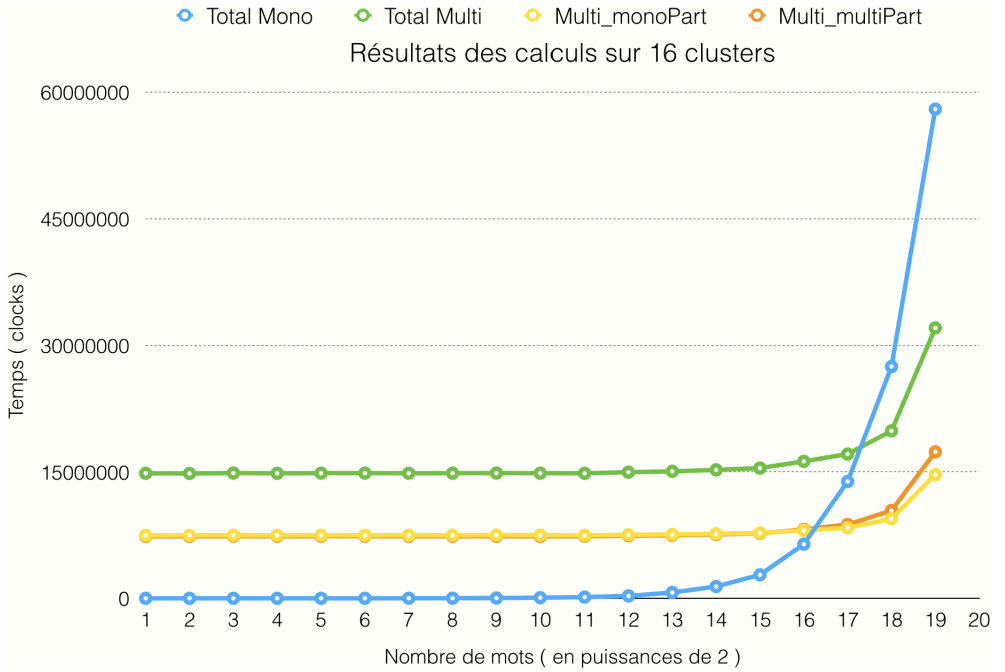
Le but de cette expérience est d'évaluer les temps d'exécution obtenus grace au code décrit ci-dessus, sur plusieurs plages de mots et plusieurs configuration, on dispose dans almos des 4 configurations suivantes :
- sim1 : Configuration avec un cluster, et donc 4 processeurs.
- sim4 : Configuration avec 4 clusters, 16 processeurs.
- sim16 : Configuration avec 16 clusters, 64 processeurs.
- sim64 : Configuration avec 64 clusters, dont on pourra pas évaluer les résultats car le traitement ne s'est jamais lancé (après une semaine d'attente), en plus du fait que ça renvoyait des erreurs de défauts mémoire.
Pour chaque configuration on lancera à chaque fois une exécution avec un nombre exponentiellement variant de mots : 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192 et 16384.
L'expérience est faite via le script fournis : **./ExecMe.sh**
Ce Script doit se trouver à la racine d'almos, il compilera l'application DWC puis lancera les différentes configurations de simulation avec les différentes configurations de clusters et nombre de mots, comme décrit plus haut.
Les résultats des calculs seront sauvegardés et rangés par dossiers de la forme : logs.<nb_clusters>/tty1.<nb_clusters>.<nb_words>

Analyse des résultats :

Le but de cette section est d'analyser les résultats des expériences qu'on a mennées, afin de valider l'efficacité du passage à l'échelle du noyau d'ALMOS.
Les 3 premières figures illustrent les résultats en temps (clocks) et en nombre de mots (en puissances de 2), pour chaque configuration (1, 4 et 16 clusters), ces résultats montrent clairement que l'exécution multi-process est plus lente au début, ce qui signifie que la création et synchronisation des différents threads provoque des ralentissements de l'exécution multi par rapport à la mono, mais dès que le nombre de mots devient important, l'éxécution multi est largement plus efficace.
Le point intéressant dans ces graphes est donc celui du croisement des courbes multi et mono, là où la distribution des taches devient efficace :



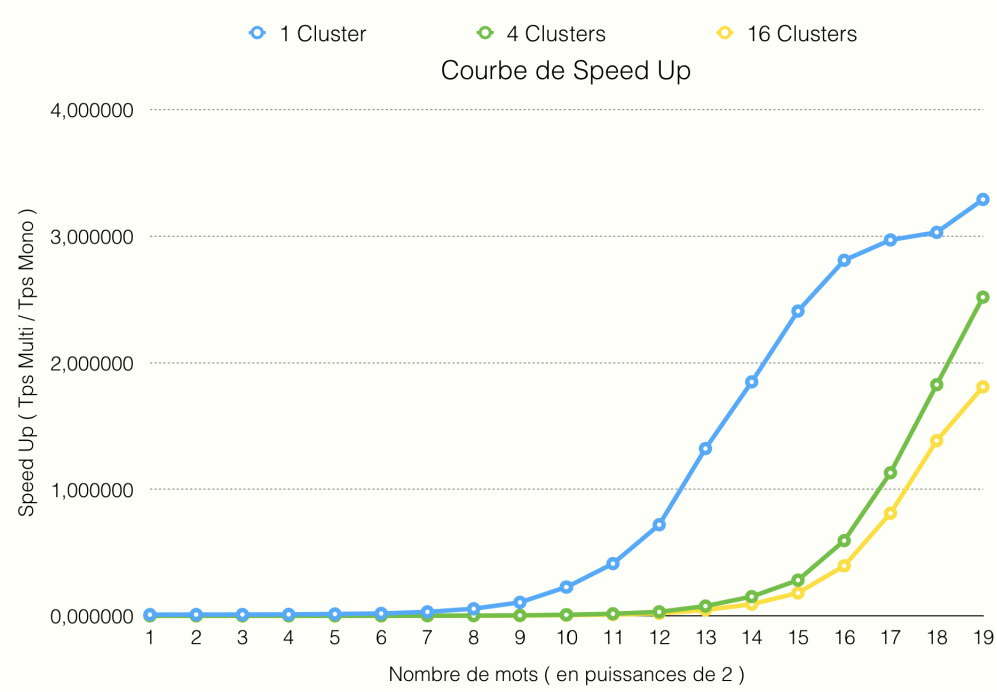




On remarque également que, la différence en temps d'éxécution au départ est plus importante quand on a plus de coeurs, ce qui est normal vu la durée que prennent les différents coeurs pour leur initialisation, leur synchronisation ainsi que pour le déplacement des données (auto-next-touch).

Le point d'efficacité de la distribution des taches se trouve à 2$\hat{1}$2 pour 1 cluster, 2$\hat{1}$6 pour 4 clusters et 2$\hat{1}$7 pour 16 clusters.

Pour pouvoir évaluer l'efficacité de la distribution, et donc valider le passage à l'échelle en nombre de coeurs, on a réalisé un 4ème graphe représentant les Speed Up par configuration de clusters, c'est à dire le résultat obtenu pour chaque exécution, en appliquant un facteur (Temps total en multi) divisé par (Temps total en mono), ce qu'on a représenté dans la figure suivante :



Malheureusement, et pour des raisons que je n'ai pas réussi à identifier, on n'arrive pas à simuler le programme pour plus de 1Mo de données, et du coup on a pas réussit à atteindre le point où dans les courbes des SpeedUps le 16 serait plus efficace que le 4 qui serait plus efficace que le 1. Notre expérimentation s'est donc arrêtée ici. Ce qui est un bon point d'avance par raport aux autres, d'après ce qu'on a pu constater.

Temps de simulation :
La simulation a du prendre pour l'ensemble des configurations a du prendre 2 semaines sur les machines de la fac.

Conclusion :

Ce TP est venu complèter nos connaissances architecture Many-Cores, il nous a permis de découvrir le simulateur TSAR et du coup de faire notre étude dessus, on a également pris en main le système d'exploitation ALMOS.

On a également pu observer les limites de la répartition des traitements et leur efficacité par rapport à la localité des accès mémoire, ceci grace au dévelloppement d'une application simple, distribuée, puis production d'une série de benchs pour appuyer la notion de passage à l'échelle du noyau d'ALMOS.