Rapport final - Projet GC

Mathis Verstrepen Remi Clement

25 Mai 2023

Contents

1 Contexte du projet et description du sujet

Le but de ce sujet est de créer une application pour résoudre des problèmes d'ordonnancement. Elle devra calculer les dates de début au plus tôt et au plus tard pour chaque tâche ainsi que calculer leur marge. Enfin elle devra identifier la durée minimale du projet et les tâches critiques.

L'application doit être capable de lire un fichier d'entrée avec un format spécifique (inspiré de DIMACS) contenant les informations sur les sommets, les arêtes, les poids et les contraintes de précédence. Il doit être également capable de traiter de grands ensembles de données en étant efficace en usage de la mémoire et CPU.

2 Analyse et conception du problème

L'objectif de ce projet est de créer un programme pour résoudre des problèmes d'ordonnancement de tâches simples en tenant compte des contraintes de précédence et de la durée des tâches individuelles.

2.1 Modélisation du problème

Le graphe sera représenté à l'aide de trois tables :

- *Head* : La table Head contiendra les informations de chaque sommets. Elle sera représenté à l'aide d'une liste de structure dont voici la forme :

  ```
  struct sommet {
      entier succIdx
      entier predIdx
      entier poids
      entier dateTot
      entier dateTard
      entier marge
      booléen estCritique
  }
  ```

  NB : Nous avions oublié le champ marge lors du premier rapport.

- *Succ* : Le tableau *Succ* est une liste rangée contenant successivement pour chaque sommet x_i l'ensemble de ses sommets successeurs.

- *Pred* : Le tableau *Pred* est une liste rangée contenant successivement pour chaque sommet x_i l'ensemble de ses sommets predecesseurs.

2.2 Méthode de résolution

Pour résoudre ce problème d'ordonnancement, nous devons déterminer, les dates de début au plus tôt et au plus tard pour chaque tâche, ainsi que la durée minimale du projet ,les tâches critiques et la marge de chacune des tâches. Nous utiliserons la méthode de Potentiel-Tâche pour calculer ces informations en utilisant l'algorithme de Bellman sur le graphe.

Cette méthode se déroule en plusieurs étapes :

2.2.1 Calcul l'ordre topologique

On utilisera une variante de l'algorithme d'exploration en profondeur pour calculer la liste *Sorted* correspondant à l'ordre topologique des sommets du graphe.

2.2.2 Calcul des potentiels maximaux

On applique l'algorithme de Bellman à partir du sommet de départ pour déterminer les potentiels maximaux de chaque sommet. Les potentiels maximaux correspondent aux dates de début au plus tôt pour chaque sommet.

2.2.3 Calcul des potentiels minimaux et identification des tâches critiques et des marges

On construit l'antiarborescence et on utilise l'algorithme de Bellman à partir du sommet de fin pour déterminer les potentiels minimaux de chaque sommet. Les potentiels minimaux correspondent aux dates de fin au plus tard pour chaque sommet.

Pendant cet algorithme on calculera et identifira également les tâches critiques et les marges de chaque sommet.

2.2.4 Détermination de la durée minimale du projet

La durée minimale du projet est déterminée en trouvant la date de fin au plus tard du sommet de fin du graphe.

3 Mode d'emploi

3.1 Commandes de compilation

3.1.1 Compilation classique

Série de commandes pour un lancement classique du programme :

```
$ gcc -c *.c
$ gcc *.o
```

3.1.2 Compilation orienté débogage

Série de commandes optimisés pour le débogage du programme et les test de valgrind :

```
$ gcc -c -g -Wall -pass-exit-codes *.c
$ gcc -g -Wall -pass-exit-codes *.o
```

3.2 Commandes de lancement

3.2.1 Lancement orienté performance

Ce lancement optimise le temps d'éxecution du programme et n'affiche que la valeur optimale du graphe.

```
$ ./a.out "../ressources/nom_du_fichier.txt"
```

3.2.2 Lancement orienté débogage et informations

Ce lancement affiche toutes les données du graphe (Table Head, Succ, Pred, date au plus tôt, au plus tard, tâches critiques et marge si non critique) (ajout du paramètre d dans la commande).

```
$ ./a.out "../ressources/nom_du_fichier.txt" d
```

3.2.3 Lancement Valgrind

Commande utilisé pour tester les fuites de mémoires et les accès mémoire non correct avec Valgrind (la compilation orienté débogage est recommandée) :

```
$ valgrind --trace-children=yes --track-fds=yes --track-origins=yes
  --leak-check=full --show-leak-kinds=all ./a.out
  "../ressources/nom_du_fichier.txt"
```

3.3 Fichiers d'entrées sorties

Nous utilisons les 5 fichiers de test fournis pour le projet qui sont nommés :

- "*Ex_300000.txt*" : graphe de 300 000 points
- "*Ex_TD.txt*" : graphe de l'exemple de TD
- "*ordo30.txt*" : graphe de 30 points
- "*ordo120.txt*" : graphe de 120 points
- "*Sondage.txt*" : graphe de l'exemple du sujet

Les noms de fichiers sont mis en arguments de la commande de lancement et non en fichier d'entrées (cf. Commandes de lancement).

Une sortie dans un fichier en sortie est possible avec un ajout en fin de commande d'une redirection de la sortie ($> out.txt$)

4 Résultats et performance

4.1 Résultats

Voici les valeurs optimales trouvées pour chaque fichier de test :

- "$Ex\_300000.txt$" : 1 350 000
- "$Ex\_TD.txt$" : 43
- "$ordo30.txt$" : 54
- "$ordo120.txt$" : 95
- "$Sondage.txt$" : 43

Elles correspondent toutes aux valeurs optimales attendu.

Les valeurs des dates au plus tot, au plus tard, des tâches critiques, des marges ont été comparées aux valeurs connues des fichiers $Sondage.txt$ et $Ex\_TD.txt$ et correspondent aux valeurs attendu.

4.2 Performance

4.2.1 Performance temps CPU

- Les fichiers "$Ex\_TD.txt$", "$ordo30.txt$", "$ordo120.txt$", "$Sondage.txt$" s'éxecute tous en dessous de la barre des 1 milliseconde.
- Le fichier "$Ex\_300000.txt$" est traité en environ 125 millisecondes.

```
Performance counter stats for './a.out ../ressources/Ex_300000.txt' (100 runs):

          129,23 msec task-clock                #    0,993 CPUs utilized            ( +-  0,30% )
              18      context-switches          #  138,938 /sec                     ( +-  7,28% )
               1      cpu-migrations            #    7,719 /sec                     ( +-  4,51% )
           6 407      page-faults               #   49,454 K/sec                    ( +-  0,00% )
     590 136 106      cycles                    #    4,555 GHz                      ( +-  0,29% )  (81,49%)
       5 986 360      stalled-cycles-frontend   #    1,01% frontend cycles idle     ( +-  0,36% )  (82,34%)
          75 854      stalled-cycles-backend    #    0,01% backend cycles idle      ( +-218,90% )  (84,55%)
   2 502 942 775      instructions              #    4,23  insn per cycle
                                                #    0,00  stalled cycles per insn  ( +-  0,05% )  (84,61%)
     500 385 125      branches                  #    3,862 G/sec                    ( +-  0,07% )  (84,55%)
         135 252      branch-misses             #    0,03% of all branches          ( +-  0,35% )  (82,46%)

        0,130188 +- 0,000397 seconds time elapsed  ( +-  0,31% )
```

Figure 1: Détails indicateurs de performances et de leurs variations sur 100 éxecutions du programme sur le fichier de test du graphe de 300 000 sommets.

NB : les temps d'éxecution totaux du programme ont été testé à l'aide de la commande $perf stat$ de linux (man7.org/linux/man-pages/man1/perf-stat.1.html). Les tests ont été effectué sur un processeur AMD Ryzen 7 5800x cadencé à 4.6 GHz.

4.2.2 Utilisation mémoire

La majorité de nos structures et de nos listes étant allouées avec malloc ou calloc, il est assez représentatif de regarder le nombre de bytes allouées que nous indique Valgrind pour avoir une idée de l'ordre de grandeur de la quantité de mémoire utilisée.

Mémoire allouée par fichier :

- "$Ex\_300000.txt$" : 22 565 780 bytes
- "$Ex\_TD.txt$" : 6 644 bytes
- "$ordo30.txt$" : 8 236 bytes
- "$ordo120.txt$" : 15 212 bytes
- "$Sondage.txt$" : 6 468 bytes

En sachant que le fichier des données du graphe de 300 000 sommets pèse 9 657 809 bytes, il semble que nous soyons assez efficace en terme d'utilisation mémoire.

5 Conclusion

5.1 Etat du projet

Le projet est entièrement terminé et chaque consignes du cahier des charges a été respecté.

5.2 Améliorations possibles

5.2.1 Optimisation mémoire

Une des pistes d'amélioration pourrait être l'utilisation de la mémoire par le programme. Même si l'on reste bien moins couteux que la majorité des autres structures de données choisies pour ce projet comme par exemple l'utilisation de listes chainées au lieu d'une simple liste d'entier, nous utilisons des champs qui peuvent sembler inutiles sur une utilisation en production. En effet les champs *marge* et *estCritique* de la table $Head$ ne semblent pas essentiel puisqu'il peuvent être déduit très facilement et très rapidement à l'aide du champ *dateTot* et *dateTard*. Cela nous ferait économiser une quantité de mémoire proportionnelle au nombre sommets du graphe.

5.2.2 Multithreading

Lors de l'éxecution du programme on remarque vite que le programme ne s'éxecute que sur un seul coeur CPU et ne prend pas avantage des autres coeurs disponible du processeur. Une des pistes d'améliorations serait d'implémenter une version du programme utilisant le multithreading afin de répartir la charge

CPU sur plusieurs coeurs. Cela pourrait être fait par exemple en utilisant la librairie *pthread.h* qui permet d'implementer le multithreading en C. Cependant pour prendre avantage au maximum de cette fonctionnalité, il serait peut être nécessaire de changer l'architecture globale des algorithmes.

6 Bilan Personnel

6.1 Mathis Verstrepen

Ce projet m'a permis d'appliquer tout ce que l'on avait appris en théorie pendant les cours et de l'appliquer à un exemple réel. Pendant ce projet nous avons fait face à de nombreux obstacles notamment lors de l'implémentation de l'algorithme de chargement des données dont je me suis chargés et que j'ai du recommencer de zéro au milieu du projet par manque d'efficacité. Une fois le chargement des données mis en place nous avons pu tester et corriger les fonctions de résolution de graphe codé par Remi. Puis nous avons essayé d'optimiser au maximum ce qui pouvait l'être dans le programme.

Ce projet m'a permis de découvrir de nouvelles choses en C comme l'utilisation de *calloc*.

Au final le projet est un succès, il marche sans erreur et est très bien optimisé que ce soit en utilisation mémoire ou CPU.

6.2 Remi Clement

Au début du projet, le travail demandé ne me semblait pas évident puisque l'on n'avait pas encore implanté de graphe sur ordinateur. De plus, contrairement à une résolution sur papier, ici, dès qu'il y a un grand nombre de points, il est impossible de tracer le graphe pour voir à quoi il ressemble. Il n'y a pas de fonction d'affichage et cela rend plus difficile l'implantation des fonctions pour sa manipulation.

Une fois les structures fixées, il m'a été bien plus simple de visualiser la façon dont nous allions résoudre le problème. La principale difficulté pour moi était l'utilisation des indices des sommets dans le tableau Head pour les représenter, puisque nous avions décidé de ne pas leur attribuer d'identifiant et il a été difficile pour moi de passer du pseudo code à l'écriture des fonctions à l'aide du langage C à cause de ce moyen de procéder. Mais au final j'ai pu écrire le corps des fonctions principales et avec l'aide de Mathis nous avons corrigé les erreurs d'indices et de manipulation de tableau que j'ai pu faire.

Au final, ce projet qui me paraissait assez difficile s'est plutôt bien déroulé et le résultat obtenu est vraiment satisfaisant. Le programme marche correctement et avec un temps d'exécution vraiment court sur le graphe de 300 000 poins.