Projet ALGOREP 2017

E. Renault

Ce projet est à rendre pour le 21 Décembre 2017 au plus tard à 23h59. Il s'effectue ***au minimum*** *par groupe de 2* ***au maximum*** *par groupe de 3. Le rendu s'effectuera par mail (un par groupe) avec comme objet [ALGOREP][projet] nom1 nom2 nom3. Le rendu se composera de code (commenté !), d'un Makefile, d'un README,* ***et d'un rapport de 10 à 20 pages (maximum).*** *Tout retard donnera lieu à une pénalité. Le projet participera à hauteur de 40% de la note finale (15% pour le code composant et 25% pour le rapport).*

1 Description du Projet

Ce projet est volontairement imprécis. L'objectif étant de vous faire réfléchir et regarder l'état de l'art. Pensez à mettre dans votre rapport vos remarques, vos lectures et les problèmes rencontrés/résolus.

Description. Nous souhaitons mettre en place une mémoire distribuée. Pour cela, chaque processus va maintenir localement une collection variables entières. L'ensemble des collections de tous les processus va constituer la mémoire partagée. A chaque variable est associé un identifiant. Lorsqu'un processus veut accéder en lecture à une variable il lui suffit d'envoyer un message au processus qui la possède ; ce dernier lui renverra alors la valeur de la variable. Pour effectuer un accès en écriture sur une variable, il faut envoyer un message au processus qui possède la variable pour qu'il propage la nouvelle valeur.

Vous écrirez des fonctions permettant :

1. de demander l'allocation d'une variable. Cette fonction doit retourner un identifiant permettant d'y accéder depuis n'importe quel processus.
2. de demander la lecture d'une variable. Cette fonction doit retourner la valeur de la variable.
3. de demander la modification d'une variable. Cette fonction doit retourner un booléen indiquant si la modification a réellement eu lieu.
4. de libérer une variable.
5. de demander l'allocation/la destruction d'un ensemble de variable, i.e. d'un tableau.

Notons que nous nous limitons ici au type entier pour se faciliter les choses. Les points 1, 2 et 4 sont relativement simple à implémenter via un échange de message. Libre à vous de choisir la topologie virtuelle de votre réseau qui minimise le nombre de messages échangés.

Le point 3, même s'il semble simple requiert une attention particulière : que se passe-t-il lorsque deux messages d'écriture arrivent simultanément ? que se passe-t-il lorsqu'un message de modification survient et que celui-ci a un temps logique très ancien ?

Le dernier point est sans doute le plus compliqué. Que se passe-t-il si la mémoire requise est supérieure à tout ce qui est disponible sur un processus ? Dans ce cas, l'identifiant doit permettre de référencer des blocs venant d'endroits différents. Le mécanisme doit aussi permettre d'aller modifier une variable particulière dans ce tableau d'éléments.

Tester son allocateur. Vous devez construire une application permettant de tester votre allocateur. Vous pouvez choisir n'importe quelle application du moment qu'elle stimule votre système. Une idée possible serait de construite un (très grand) tableau avec des nombres aléatoires, puis de trier ce tableau en répartissant le travail sur chaque processus (pensez aux algorithmes divide and conquer). Une autre idée possible serait de prendre une image et de faire une convolution, chaque processus traitant des parties précises de l'image.

Pour aller plus loin. Pensez que votre allocateur peut être intelligent : on peut par exemple vouloir déclarer une affinité entre certaines données et certains processus pour éviter des échanges coûteux de messages. Par ailleurs, un objectif dans ce genre de système est d'effectuer un équilibrage de charge pour éviter que toutes les allocations tombent sur le même processus.

2 Description du rapport

L'objectif de ce rapport est multiple. Tout d'abord il doit présenter votre topologie et expliquer vos choix. Ensuite le rapport doit présenter votre solution, vos API, et évaluer la complexité en terme de messages échangés pour chacune des opérations présentes dans votre API. Ensuite il est nécessaire de décrire l'application de test que vous avez choisi et d'expliquer pourquoi elle vous semble stimuler correctement votre mémoire partagée. Pensez à donner le pseudo-code du comportement de chaque processus pour plus de clarté.

Remarques

Dans les dernières versions, MPI offre déjà la possibilité d'avoir une mémoire partagée. Il est bien sur hors de question ici de se reposer la-dessus. De plus MPI offre de nombreux langage de programmation vous pouvez choisir celui que vous souhaitez du moment qu'il reste *mainstream* : si vous avez un doute sur ce qu'est un langage *mainstream* demandez moi ! Enfin (1) il ne vous est pas demandé de gérer les pannes et (2) pensez que votre application puisse être testée avec un nombre de thread différent de celui que vous utiliserez !