Ecole Polytechnique

Cours : "Parallélisme"
Travaux dirigés
E. Goubault & M. Martel

TD 6

4 mars 2003

On commencera par finir l'exercice sur les rotations de Givens.

1 Objets activables à distance et RMI

On souhaite améliorer la façon dont pi est calculé par la méthode vue au TD 6, en faisant en sorte que les serveurs de calcul soient activables à distance. Modifier le programme de calcul de pi du TD 5 (ou du TD 6) comme indiqué précédemment.

2 Prise en main de JAVAPARTY

- Commencer par générer (sauf si déjà fait par ailleurs) une clé pour `ssh`. Pour savoir si cela est nécessaire ou pas, faire `ssh` sur les machines que vous comptez utiliser pendant les TDs (à gauche, à droite etc.): s'il se connecte sans rien demander, alors c'est bon, sinon:

  ```
  > ssh -keygen -b1024 -tdsa
  ```

  Quand on vous demande `enter paraphrase`: tapez return (de même pour la confirmation). Maintenant, faites une fois `ssh` sur les machines voisines (il vous demandera quelque chose uniquement cette fois-ci).

- Mettre `/users/labos/lix/martel/jp-rmi/bin` dans votre `PATH` (dans `.cshrc` par exemple)

- Modifier le script `jpexec` (sur le web):

  ```
  jprm &
  jpvm -verbose &
  ssh loire "jpvm -verbose" &
  ```

  en changeant le nom `loire` par une autre machine, proche. De même, modifier les droits (et éventuellement le `PATH`, si nécessaire), pour que ce script soit exécutable.

- Commencer par essayer d'utiliser JAVAPARTY sur le programme `HelloJP.java` fourni (sur le site web):

  ```
  package examples;

  import jp.lang.DistributedRuntime;

  public remote class HelloJP {
    public void hello() {
  ```

```
    // Print on the console of the virtual machine where the
    // object lives
    System.out.println("Hello JavaParty!");
  }

  public static void main(String[] args) {

    int cnt = DistributedRuntime.getMachineCnt();
    System.out.println(cnt);
    for (int n = 0; n < cnt; n++) {
      /** @at n */
      HelloJP world = new HelloJP();
      // Remotely invoke a method
      world.hello();
    }
  }
}
```

- Pour ce faire: on compile par `jpc -d . HelloJP.java`, puis on lance `jpexec` (on attend un peu de voir les identificateurs 0 et 1 créés), puis dans la même console, on fait `jp examples.HelloJP`.
- Qu'est-ce que cela donne? L'affichage est redirigé sur la console d'origine par `ssh`; modifier le code pour que l'on soit sûr que l'on s'éxécute bien sur la bonne machine. On donne la méthode:

  ```
  try {
    System.out.println(java.net.InetAddress.getLocalHost().getHostName());
  } catch (Exception e) {;};
  ```

  qui permet d'afficher le nom de la machine hôte.
- Reprendre un des TDs RMI précédent, pour l'écrire en JAVAPARTY (π, anneau etc.)

3 Enveloppe convexe parallèle

On souhaite calculer l'enveloppe convexe d'un ensemble de points sur n machines en parallèle, à l'aide de JavaParty. Il est rappelé que l'algorithme séquentiel fonctionne suivant le principe du "diviser pour régner" :

- On se restreint au calcul de la demie enveloppe supérieure, la demie-enveloppe inférieure pouvant être calculée de façon similaire.
- On suppose que l'on dispose au départ de 2^n points $\{p_1, \ldots, p_{2^n}\}$, d'abcisses et d'ordonnées différentes deux à deux et tels que trois points ne sont jamais alignés.
- Le calcul de l'enveloppe convexe des 2^n points consiste à calculer récursivement les enveloppes des 2^{n-1} points de plus petites abcisses, celle des 2^{n-1} points de plus grandes abcisses et à fusionner les deux enveloppes obtenues.

Nous allons développer la classe `ConvexHullHP` suivante :

```
remote class ConvexHullJP {
    int[] x,y; /* coordonnees des points */
    int[] ch;  /* tableau contenant, les indices des points appartenant a l'enveloppe */
    int nMax;  /* nombre de points = taille de x et y */
    int nCh;   /* nombre de points de l'enveloppe = taille de ch <= nMax */

    ConvexHullJP(int[] tx,int[] ty);
```

```
    void merge(ConvexHullJP uh1,ConvexHullJP uh2);
    void compute();
    void printGraph();

    public static void main(String[] args) {
        int[] tx = { 8,15,19,22,33,41,47,50 };
        int[] ty = { 15,11,25,31,27,32,21,9 };
        ...
    }
}
```

La méthode `merge` permet de réaliser la fusion (voir page web). La méthode `printGraph` permet d'obtenir une représentation graphique du résultat par la commande `java ConvexHullJP | ch` (son code est aussi donné sur la page web).

1. Réfléchir à l'organisation des classes (`remote` ou pas)

2. Ecrire le reste de la classe de façon à ce que les appels récursifs soient effectués sur des machines différentes par JavaParty.

3. Afin de limiter le nombre de tâches, modifier le programme précédent pour que seuls les p premiers niveaux d'appels soient distribués, les suivant étant exécutés en séquentiel sur le noeud courant (en pratique on pourra poser $p = 2$ pour utiliser 4 machines).

4. Modifier le programme afin d'être sûr que chaque calcul s'effectuera sur une machine différente.

On suppose maintenant que les points donnés en entrée ne sont pas triées. Implémenter, en JavaParty, un algorithme de tri vu en cours (par exemple le tri pair-impair sur réseau lináire).