Maxima et la programmation en parallèle

Danielle Léger
Département de mathématiques et d'informatique
Université Laurentienne (Sudbury, Ontario)

La présence de l'ordinateur constitue une réalité palpable du monde d'aujourd'hui. Les ordinateurs sont utilisés dans toutes les sphères de l'activité humaine. Que ce soit en sciences, en génie ou en mathématiques, les ordinateurs sont très souvent capables de nous secourir, notamment en simplifiant l'exécution de diverses tâches. En fait, il existe une variété de logiciels qui facilitent la recherche d'un mathématicien, ces programmes étant apparus pour la première fois il y a plus de trente ans.

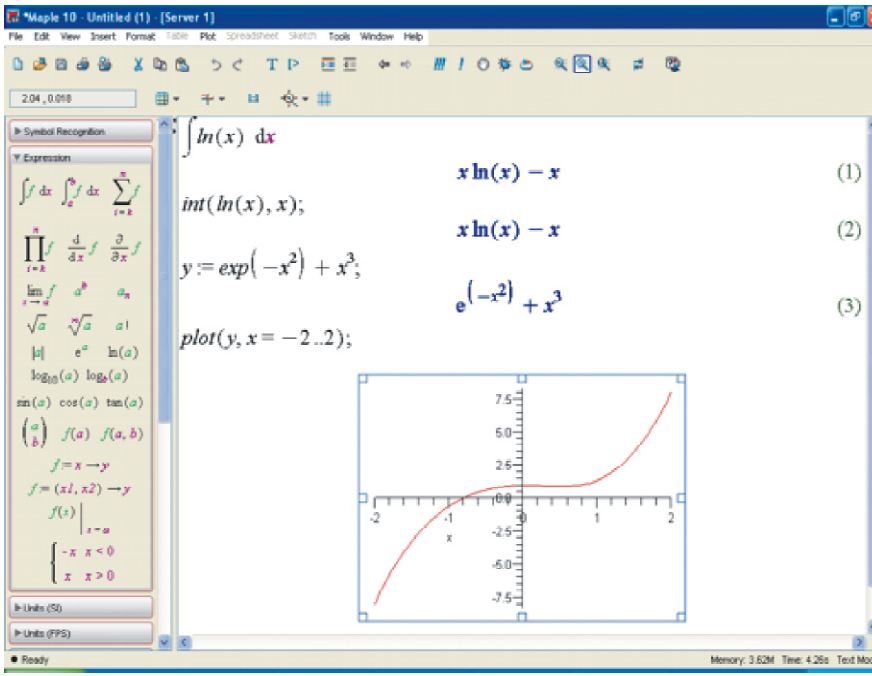
Après plusieurs années d'évolution et d'amélioration, il existe maintenant une vingtaine de logiciels différents susceptibles d'aider le travail du mathématicien, *Maple, Mathematica* et *Maxima* étant parmi les plus populaires (Fateman, s.d.). Par contre, il est indispensable de noter que, malgré l'utilité et l'efficacité de ces logiciels, le temps d'exécution de certaines procédures peut être de quelques heures et, parfois même, de quelques jours. Il faut alors trouver des moyens de réduire le temps d'exécution.

La programmation en parallèle est une méthode courante qui permet de réaliser un algorithme et qui, par ricochet, permet d'obtenir des résultats plus rapidement. C'est ainsi que cette nouvelle technologie devient une solution avantageuse qui offre plusieurs possibilités pour l'avenir.

Comparaison du logiciel *Maple* au logiciel *Maxima*

Dans la recherche mathématique, les logiciels de calcul symbolique sont fréquemment utilisés. Aujourd'hui, *Maple* est l'un des logiciels les plus connus. Créé en 1981 par l'Université de Waterloo (Encyclopédie Wiképédia, 2006), *Maple* existe depuis longtemps dans plusieurs universités canadiennes. Il possède un mode interactif, tel qu'illustré à la figure 1, ce qui constitue une caractéristique idéale pour les étudiants et les étudiantes. Le mode interactif facilite la résolution d'un problème en offrant une variété de menus qui indiquent ses diverses capacités. L'étudiant ou l'étudiante peut ainsi retrouver les commandes nécessaires pour réaliser tous ses calculs avec simplicité. Parmi ses nombreuses capacités, *Maple* permet de calculer des intégrales et des dérivées, de créer des graphiques en deux et trois dimensions ainsi que de résoudre des systèmes d'équation. Il permet également d'écrire des algorithmes sous forme de procédures et de les faire exécuter.

Figure 1
Le mode intéractif de *Maple*



C'est cette propriété du logiciel qui est la plus utile au moment de faire une recherche. L'efficacité du logiciel est la plus couramment mesurée par la durée d'exécution d'une procédure : plus le temps d'exécution est court, plus le logiciel est considéré comme étant efficace. En tenant compte de la vaste panoplie de logiciels de calcul symbolique qui existent sur le marché aujourd'hui, *Maple* est l'un des plus rapides et des plus puissants. Il est toutefois important de tenir compte du fait que, malgré son caractère efficace, *Maple* est un programme commercial, ce qui explique en grande partie son coût élevé.

En contrepartie, il existe des logiciels qui, contrairement à *Maple*, sont disponibles gratuitement pour le public : c'est le cas de *Maxima*. Ce programme est un descendant du logiciel Macsyma, créé dans les années soixante par l'Institut de technologie de Massachusetts (*Massachussetts' Technology Institute*), mieux connu sous l'acronyme MIT (Maxima, 2006). Macsyma a été l'un des premiers logiciels de calcul symbolique disponibles sur le marché. Ainsi, il a été véritablement un pionnier dans le développement de tels programmes. En 1982, William Schelter a développé la nouvelle version appelée *Maxima*. C'est grâce à son dévouement que l'entretien, l'amélioration et le succès de ce logiciel ont été possibles (Maxima, 2006).

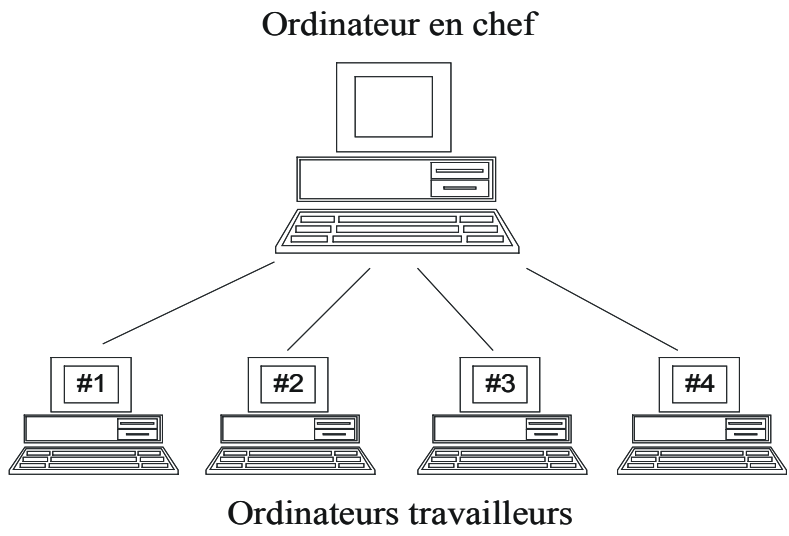
En fait, *Maxima* possède presque toutes les mêmes capacités que *Maple*, en plus d'être gratuit. Le programme est également « doté d'un langage de programmation, [donc] ses possibilités sont très riches » (Encyclopédie Wiképédia, 2006). Il est alors possible de traduire un algorithme écrit pour *Maple* et de le transformer en une procédure qui peut être exécutée par *Maxima*.

Comme les commandes se ressemblent d'un logiciel à l'autre, la traduction d'un algorithme s'accomplit assez facilement. Néanmoins, pour certains calculs, *Maxima* est moins efficace que *Maple*, ce qui entraîne souvent un temps d'exécution plus élevé lors de son utilisation. Pour régler ce problème, il suffit de trouver des techniques pour améliorer le rendement de ce logiciel afin d'obtenir de meilleurs résultats.

Programmation en parallèle

La programmation en parallèle est une méthode fréquemment utilisée pour réduire le temps d'exécution d'une procédure. C'est pourquoi elle peut être appliquée aux algorithmes créés pour le logiciel *Maxima*.

Figure 2
La programmation en parallèle



La première étape consiste à analyser le programme qui réalise l'algorithme, afin de déterminer quelles parties peuvent être calculées de manière parallèle. En d'autres termes, il faut identifier les sections de l'algorithme qui peuvent être décomposées en petites tâches capables d'être accomplies simultanément. De cette façon, il n'y a qu'un seul ordinateur en chef responsable de l'exécution

de tous les calculs afin de préparer les diverses tâches. Par la suite, celles-ci sont distribuées aux ordinateurs « travailleurs » qui sont responsables de les évaluer et d'envoyer leurs résultats à l'ordinateur en chef qui, à son tour, produira la solution finale. De ce fait, le programme initial créé pour *Maxima* doit être réécrit en deux parties : la première construira la liste de tâches à être évaluées, et la seconde contiendra le code nécessaire pour que les ordinateurs travailleurs puissent les calculer.

Une fois cette étape accomplie, il reste à développer les programmes responsables de la communication entre l'ordinateur en chef et les travailleurs. Ce travail se fait assez facilement grâce au logiciel Java RMI (*Remote Method Invocation*), qui offre à l'utilisateur une technique simple et compréhensible pour réaliser la programmation en parallèle. Il suffit de créer trois classes[1], deux qui seront utilisées par l'ordinateur en chef et une qui le sera par les ordinateurs travailleurs.

L'ordinateur en chef

L'ordinateur en chef doit régler la gestion des tâches et des résultats. Il exige donc deux classes : une interface et une classe qui rendra l'interface efficace. Toutes les méthodes qu'exigeront des ordinateurs travailleurs doivent être déclarées dans l'interface. Ces méthodes peuvent être, par exemple, *recevoirTâche()* et *envoyerRésultat()*. Puisque ces méthodes se trouvent dans l'interface de l'ordinateur en chef, il sera maintenant possible pour les travailleurs d'y accéder sans difficulté sur un autre ordinateur.

Quant à la deuxième classe, nommée Chef.java, c'est elle qui contiendra la totalité des commandes utilisées par l'ordinateur en chef. En effet, c'est là que celui-ci pourra exécuter la première partie de l'algorithme de *Maxima* afin de créer la liste des tâches à calculer. Pour ce faire, il suffit d'employer les commandes suivantes :

```
Process p = Runtime.getRuntime().exec("Maxima –b programme.max");
p.waitFor();
```

où programme.max est le fichier qui contient l'algorithme. *Maxima* réalise la procédure, crée la liste des tâches et conserve ces résultats dans un nouveau fichier pour que Java puisse les retrouver. Par la suite, la classe Chef.java complète les méthodes qui sont présentes dans l'interface.

L'ordinateur en chef possède alors le code nécessaire pour distribuer les tâches aux travailleurs, recueillir les résultats et imprimer la solution finale. Toutefois, la synchronisation de ces événements est primordiale. En fait, c'est une synchronisation bien achevée qui permettra d'éviter des problèmes, comme la simultanéité d'activité de deux travailleurs qui veulent exécuter la

[1] En Java, tous les programmes que nous créons se nomment des classes.

même tâche en même temps. Finalement, la classe Chef.java doit enregistrer l'ordinateur en chef avec le registre RMI. Ce registre constitue un annuaire qui permet aux travailleurs de retrouver le chef dans le but de communiquer avec lui.

Les ordinateurs travailleurs

Il y a une même classe créée pour tous les ordinateurs travailleurs. Avec cette classe, nommée Travail.java, chaque travailleur peut chercher dans le registre RMI afin de localiser le chef. Une fois cette tâche réalisée, le travailleur se branche au chef et peut maintenant accéder aux méthodes de l'interface. Grâce au code dans Travail.java, le travailleur peut demander une tâche à l'ordinateur en chef, l'évaluer en utilisant la deuxième partie de l'algorithme de *Maxima*, et renvoyer le résultat; ce processus se poursuit jusqu'à ce que toutes les tâches aient été évaluées.

Exemple de chaque classe

Interface.java

```
import java.rmi.*;
public interface Interface extends Remote
{
    public String recevoirTâche( ) throws RemoteException;
    public void envoyerRésultat( int résultat ) throws RemoteException;
}
```

Chef.java

```
import java.rmi.*;
import java.rmi.server.*;
public class Chef extends UnicastRemoteObject implements Interface
{
    public Chef( )
    {
        super( );
        //Exécuter la première partie de l'algorithme de Maxima afin de créer
        //la liste des tâches à évaluer par les ordinateurs travailleurs.
    }
    public String recevoirTâche( ) throws RemoteException
    {
        synchronized(this.getClass( )) //assure la synchronization
        {
            //commandes qui vont distribuer les tâches parmi les travailleurs
            //lorsqu'il n'y a plus de tâches, le travailleur reçoit le mot « FIN »
        }
```

```
    }
    public void envoyerRésultat( int résultat ) throws RemoteException
    {
        synchronized(this.getClass( ))
        {
            //commandes qui vont recevoir les résultats des travailleurs
        }
    }
    public static void main(String[] args)
    {
        try
        {
            //Commencer l'ordinateur en chef et l'enregistrer avec le registre RMI
            //sous le nom ChefRMI afin que les travailleurs puissent le localiser
            //et communiquer avec lui.

            Chef rmiChef = new Chef( );
            Naming.rebind( "ChefRMI",rmiChef );

            System.out.println( "L'ordinateur chef attend les ordinateurs
                                        travailleurs pour distribuer les tâches…" );
        }

        catch(Exception e)
        {
            System.out.println( " Une erreur s'est produite." );
            e.printStackTrace( );
            System.out.println(e.getMessage( ));
        }
    }
}
```

Travail.java

```
import java.rmi.*;
public class Worker
{
    public static void main(String[ ] args)
    {
        String ordichef = args[0];
        // args[0] devrait être l'adresse IP de l'ordinateur en chef
        //chaque travailleur doit connaître ceci afin de localiser le chef
```

```
            String url = "//" + ordichef + "/ChefRMI";
            //exemple: //142.51.24.98/ChefRMI
            int résultat;
            try
            {
                //Trouver l'ordinateur en chef dans le registre RMI et s'y brancher
                Interface objetLointain = (Interface) Naming.lookup(url);

                //Demander au chef une tâche à évaluer
                String tâche = objetLointain.recevoirTâche( );

                while (! (tâche.equals("FIN")))
                {
                    //commande pour évaluer les tâches

                    //commande pour envoyer le résultat
                    objetLointain.envoyerRésultat(résultat);

                    //commande pour recevoir une nouvelle tâche
                    tâche = objetLointain.recevoirTâche( );
                }
            }
            catch(Exception e)
            {
                System.out.println( " Une erreur s'est produite." );
                e.printStackTrace( );
                System.out.println(e.getMessage( ));
            }
        }
}
```

Une fois que ces trois classes sont programmées, il suffit de les compiler en utilisant les commandes suivantes :

```
javac Interface.java
javac Chef.java
javac Travail.java
rmic Chef
```

En effectuant la dernière commande, Java RMI crée les liens nécessaires entre l'ordinateur chef et les ordinateurs travailleurs pour assurer la communication entre eux. Tout est maintenant prêt pour effectuer la programmation en

parallèle de l'algorithme. Il ne faut qu'activer le registre RMI et ensuite exécuter la classe Chef.java sur l'ordinateur en chef et la classe Travail.java sur chacun des ordinateurs travailleurs. De cette manière, l'algorithme de *Maxima* est calculé avec plusieurs ordinateurs au lieu d'un seul, ce qui a pour résultat de minimiser le temps d'exécution de la procédure.

Résultats et améliorations

La programmation en parallèle a été appliquée à plusieurs algorithmes en utilisant huit ordinateurs travailleurs. Chaque fois, le temps d'exécution de la procédure a diminué de 50 % à 75 %. De plus, un nombre minime de changements aux classes Interface.java, Chef.java et Travail.java ont été nécessaires pour accommoder un nouvel algorithme. Ces classes sont alors réutilisables : il suffit de les créer correctement une seule fois. Malgré ces excellents résultats, il reste certaines améliorations à apporter, ce dont il sera question dans les paragraphes suivants.

La technique développée oblige le programmeur à activer manuellement les ordinateurs travailleurs chaque fois qu'il désire exécuter un algorithme en parallèle pour que ceux-ci commencent à demander les tâches à l'ordinateur en chef. Avec seulement huit travailleurs, cette technique ne cause pas de graves ennuis, mais en augmentant ce nombre, le travail manuel peut devenir long et inefficace. Il serait alors mieux de trouver un méthode où l'ordinateur en chef chercherait des ordinateurs travailleurs qui sont libres pour ensuite leur attribuer une tâche. Ainsi, l'ordinateur en chef pourrait initier une communication avec les travailleurs, alors que, actuellement, seul l'inverse est possible.

L'amélioration de la technique pourrait également résoudre un deuxième problème. À l'heure actuelle, si un ordinateur fait défaut en calculant une tâche et est ainsi incapable de renvoyer un résultat, l'ordinateur en chef ne possède aucun moyen de le détecter. Il pense par conséquent que le travailleur est toujours en train de terminer l'évaluation de la tâche et attendra indéfiniment le résultat. Si l'ordinateur en chef pouvait communiquer avec ce travailleur défectueux, il saurait redonner la tâche à un autre travailleur.

Malgré les quelques lacunes dans la méthode que nous avons suivie, les résultats de notre recherche démontrent que la programmation en parallèle est avantageuse puisqu'elle permet de diminuer considérablement le temps d'exécution d'une procédure.

Références

FATEMAN, R.J. (s.d.). *Computer Algebra Systems,* Disponible à http://en.wikipedia.org/wiki/Computer_algebra_system, Consulté le 21 février 2006.

MAPLE SOFTWARE, Disponible à : http://en.wikipedia.org/wiki/Maple_computer_algebra_system, Consulté le 21 février 2006.

MAXIMA, Disponible à http://maxima.sourceforge.net/, Consulté le 21 février 2006.

MAXIMA, Disponible à http://fr.wikipedia.org/wiki/Maxima, Consulté le 21 février 2006.