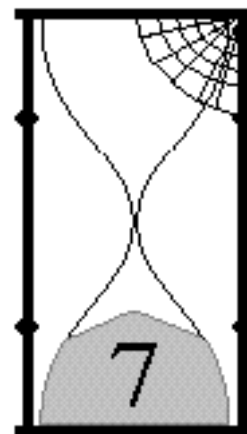


*Sur un chemin montant, sablonneux, malaisé,
et de tous les côtés au soleil exposé,
six forts chevaux tiraient un coche.*

*Jean de la Fontaine, Fables, Livre VII,
Le coche et la mouche*



Modèle de Cundall parallèle

7.1. Parallélisme



i la modélisation par éléments distincts permet de réaliser des simulations impressionnantes, elle est malheureusement très gourmande en calculs. On l'a vu, l'utilisation d'une triangulation permet d'améliorer les performances des programmes. Cependant, quelle que soit la technique de voisinage, on ne pourra jamais descendre en dessous d'une certaine limite. Pour augmenter encore la vitesse d'exécution, il n'y a que deux alternatives : utiliser un ordinateur plus puissant ou *paralléliser* le programme, i.e. répartir le travail sur plusieurs unités de calcul (les processeurs). Disons d'emblée qu'il ne suffit pas de lancer un programme séquentiel sur un ordinateur parallèle pour obtenir de bons résultats. Ce qu'il faut, c'est “penser parallèle” en écrivant le programme.

Nous avons mis en évidence au chapitre 3 les différences conceptuelles entre les modèles newtoniens des écoles des corps déformables et indéformables, et surtout le fait que l'on ne doit pas, dans le premier cas, calculer les temps exacts des chocs. Cette particularité nous facilite grandement la vie, car on peut se passer d'un échéancier, et on ne doit plus maintenir la triangulation de manière continue. Cela rendra plus facile la parallélisation du programme : en effet, le principal obstacle était justement que l'on ne pouvait pas répartir un échéancier sans une grande perte d'efficacité, comme on le verra plus loin.

Ce chapitre explique comment paralléliser la technique présentée au chapitre 6. Nous n'entrons pas trop dans les détails du programme parallèle, car il a recours à des bibliothèques spécifiques à la machine utilisée. Nous avons préféré ne donner que les idées générales programmables sur d'autres machines parallèles. On fera aussi un bref survol de la littérature, bref parce que les références sur le sujet ne sont pas légion, et nous présenterons les résultats que nous avons obtenus.

7.1.1. Le parallélisme

On peut se représenter un ordinateur parallèle comme un atelier où plusieurs ouvriers (les processeurs) travaillent et communiquent. Un programme parallèle peut se voir comme l'organisation de la répartition d'un travail entre ces personnes. Chaque ouvrier fera un certain nombre de tâches dans son coin et communiquera ses résultats aux autres ou à son chef une fois son travail terminé. En général, les ouvriers n'ont pas accès à toute l'information, mais seulement à la partie qui les intéresse directement. S'ils ont besoin d'autres données pour une raison ou pour une autre, ils doivent les demander à qui de droit. Le programme répartit à la fois la charge de travail et les données. La phase de discussion est la partie critique du programme. En effet, plus on discute et moins on travaille ! Le cas idéal est de ne communiquer qu'au début et à la fin, mais cela veut dire que toutes les tâches sont indépendantes, ce qui est rarement le cas, et certainement pas lors de la simulation d'un milieu granulaire puisque justement le mouvement de chaque grain est entravé par ses voisins. Des échanges d'informations seront donc inévitables, mais il faudra minimiser leur nombre et leur durée.

7.1.2. Utilisation de plusieurs ordinateurs à travers un réseau

Dans le cadre de son travail de diplôme, G. Mongelli avait tenté de paralléliser le programme décrit au chapitre 5 (voir 8-[Mon94]). L'idée était d'utiliser plusieurs ordinateurs connectés dans un réseau, afin "d'additionner" leur puissance de calcul. On a constaté que ce n'était pas une bonne idée, car les communications à travers le réseau étaient beaucoup trop lentes. Ce que l'on gagnait en utilisant plusieurs ordinateurs, on le perdait en grande partie à cause de la lenteur des communications. La première conclusion qui s'impose est donc d'utiliser un vrai ordinateur parallèle. Nous avons la chance à l'EPFL d'avoir un des ordinateurs les plus puissants du monde, le Cray T3D, que 6-[Hof95] plaçait au 31e rang mondial en 1995.

7.1.3. Les difficultés de paralléliser un modèle de corps indéformables

On ne peut pas répartir un échéancier sans artifices; c'est le deuxième enseignement de 8-[Mon94]. Imaginons que l'on a deux processeurs à disposition, appelons-les Alice et Georges, ainsi que deux échéanciers (si on n'en utilise qu'un seul, on ne gagne rien). Le remplissage des échéanciers au début du cycle n'est pas difficile à paralléliser. Mais une fois que le cycle commence, les ennuis arrivent. Disons qu'Alice s'occupe de la collision entre deux disques d_1 et d_2 . Il faudra alors mettre à jour certaines valeurs dans les échéanciers. Tout va bien tant qu'il n'y a pas de répercussions dans l'échéancier de Georges, mais si tel est cas, alors il se peut que Georges soit plus en avance dans le temps qu'Alice, et que l'événement cherché n'existe plus. Par exemple Alice peut dire à Georges : «Je t'informe que le disque d_46 , que tu contrôles, entrera en collision avec mon disque d_2 au temps 0.00345». «Désolé», répond Georges, «mon temps local actuel est 0.00421. Ce que tu dis ne me concerne plus ! ». Le problème est donc que les différents processeurs doivent être synchrones, ce qui implique des

communications incessantes. Tous les espoirs de gain de temps partent en fumée.

La solution qu'avait adoptée G. Mongelli était de créer des frontières artificielles entre les régions contrôlées par les différents processeurs de façon à ce que les échéanciers soient indépendants les uns des autres. Malheureusement, cette méthode est très coûteuse en communications, et de plus, elle ne respecte plus les lois de la physique newtonienne.

7.1.4. Survol de la littérature

Le parallélisme a été jusqu'ici peu utilisé pour la simulation des milieux granulaires par la méthode des éléments distincts. Les seuls exemples que l'on connaît sont très récents puisque le plus ancien date de 1994.

Dans 6-[Mee94], Meegoda et Washington présentent leur programme nommé TPM (TRUBAL for Parallel Machines). Leur modèle 3D est basé sur le fameux programme TRUBAL développé entre autres par Cundall. Ils utilisent comme ordinateur massivement parallèle la *connection machine CM-2*, et obtiennent un facteur 5 d'accélération par rapport à un ordinateur séquentiel VAX 8800. Chaque sphère est assignée à un processeur qui stocke les informations dans sa mémoire, ce qui évidemment limite la taille du milieu granulaire au nombre de processeurs. La méthode de programmation est bien décrite dans 6-[Mee94], par contre on reste sur sa faim quant à la présentation des résultats. En effet, le seul test présenté est fait avec seulement 200 sphères. Il n'y a pas non plus les courbes d'accélération qui permettraient de se rendre compte si le programme est bien parallélisé ou non. Il n'en reste pas moins que c'est apparemment le premier article sur la parallélisation d'une méthode par éléments distincts (6-[Gha93] est un article uniquement théorique, voire philosophique, sur l'usage des réseaux de neurones).

Hustrulid expose son travail dans 6-[Hus95], travail qui a consisté à paralléliser le modèle de Cundall 2D sur un *Transputer* à 64 processeurs de Alta Technology. L'accélération obtenue est de 2.76 par rapport à un ordinateur séquentiel, ce qui est très décevant. Cet échec n'est pas dû aux communications, comme le pense l'auteur, mais au fait que la partie de l'algorithme qui détecte les contacts et calcule les forces est séquentielle et non pas répartie sur tous les processeurs. Comme cette phase est très longue, il est normal que l'accélération soit faible. Pour être plus clair, reprenons notre image de l'atelier et des ouvriers. Imaginons qu'un travail complet est accompli en une heure par un seul ouvrier. Supposons maintenant que l'on fasse faire la moitié de ce travail par le même ouvrier, puis que l'on donne ensuite la deuxième moitié à faire à toute l'équipe. Il est clair l'on ne pourra obtenir une accélération supérieure à deux. C'est exactement ce qui se passe dans 6-[Hus95]. Il apparaît d'ailleurs clairement sur sa courbe d'accélération qu'au-delà d'un certain nombre de processeurs, on ne gagne presque plus rien à en ajouter de nouveaux. On verra dans notre implémentation qu'il est possible d'exécuter la phase de détection des contacts sur plusieurs processeurs simultanément.

3-[O'Co96] est la plus remarquable des trois références. Cette thèse écrite au MIT par R.M. O'Connor est intéressante à deux points de vue. Premièrement, la modélisation des grains est

tout à fait originale puisque ce sont des superquadriques (on a déjà parlé de cette représentation au chapitre 1). Deuxièmement, plusieurs programmes parallèles ont été testés sur différentes machines parallèles ou connectées en réseau avec moult comparaisons. Chaque processeur possède une copie de toutes les informations disponibles : les données ne sont pas réparties, ce qui limite un peu le nombre de grains à cause de la mémoire nécessaire, mais cette redondance accélère les choses car il n'y a presque plus de communications. Par contre, on perdra du temps pour mettre à jour les données sur tous les processeurs. Tous les contacts sont enregistrés dans une table : à chaque case de cette table se trouvent tous les points de contact et les forces s'exerçant sur le grain en question. Chaque processeur a sa propre table et calcule les forces de contact de N/P grains, où N est le nombre de grains et P le nombre de processeurs. Les forces résultantes sont obtenues en cumulant les forces correspondantes dans chaque table. C'est la phase de *comméragé* (*folding* en anglais) où les processeurs rassemblent leurs données pour que chacun ait la même table. Cette phase peut se faire en $O(\log_2(P))$. Les résultats sont impressionnants. Pour n'en citer que le plus spectaculaire, une simulation mettant en jeu 10'000 disques dure sur une SUN Sparc 271 secondes et sur une IBM SP2 avec huit processeurs 9 secondes, soit une accélération de 30. Évidemment, il ne faut pas se laisser leurrer par ce chiffre car, à programme parallèle identique, il y a beaucoup de différences entre les machines, mais la preuve que le parallélisme peut apporter quelque chose à la simulation des milieux granulaires est bien là.

Tous ces programmes parallèles sont des rejetons de l'école D, ce qui confirme notre analyse du § 7.1.3. Notre programme sera aussi de cette école. Nous resterons en 2D car notre but est de continuer à utiliser la triangulation de Delaunay pondérée dans le programme parallèle, afin de tirer parti de ses avantages et de pouvoir faire des comparaisons sur des bases identiques avec le programme séquentiel.

7.2. Paralléliser le modèle de Cundall



n l'a vu au paragraphe précédent, paralléliser efficacement un modèle de l'école I relève du tour de force. Par contre, la parallélisation d'un modèle de corps déformables est relativement facile, principalement parce qu'il n'y a pas trop de problèmes de synchronisation. Il faut cependant adapter le programme aux caractéristiques de la machine, pour en tirer le maximum de profit. Nous disposons à l'EPFL d'un Cray T3D. Ce superordinateur a été conçu pour que les communications entre les processeurs (qui sont appelés PE par Cray) soient très rapides. Il ne devrait donc pas y avoir de problèmes de ce côté-là.

7.2.1. Caractéristiques du Cray T3D

Le système Cray installé à l'EPFL est composé de 256 PE (Processing Elements) connectés sur un tore 3D. Chaque PE consiste en un processeur DEC Alpha à 150 MHz et en une mémoire de 64 Mégabytes, fournissant ainsi une puissance globale estimée à 38 Gigaflops* et une mémoire

totale de 16 Gigabytes. Plusieurs utilisateurs peuvent occuper en même temps différents sous-ensembles disjoints de PE sans que cela ait une influence sur les temps de calcul. Par contre, un PE ne peut traiter qu'un seul programme à la fois. Ainsi, un système de files d'attente gère les accès au T3D, ce qui a pour conséquence qu'il faut parfois attendre que des PE se libèrent avant que notre programme ne commence son exécution. Une application tournant sur le T3D reçoit un sous-ensemble (une partition) de PE, mais elle n'a pas de contrôle sur la géométrie de ce sous-ensemble, ce qui veut dire qu'on ne peut pas choisir explicitement l'emplacement des PE, mais seulement leur nombre. Heureusement, il a été montré dans 6-[Num94] que la géométrie de la partition n'a pratiquement aucune influence sur les temps de communication inter-PE. Une description plus détaillée du système peut être trouvée dans 6-[Cray94a].

Le Cray T3D possède une librairie spécialisée `shmem` qui permet d'accéder à la mémoire des PE distants 6-[Bar94]. Les deux fonctions de base de cette librairie sont `shmem_get` et `shmem_put`. La première permet de lire rapidement des informations dans la mémoire d'un processeur distant, la seconde d'écrire dans cette mémoire. Nous avons préféré utiliser ces fonctionnalités plutôt que des méthodes plus classiques où les processeurs interagissent en s'envoyant des messages (PVM par exemple); l'inconvénient principal de cette façon de faire est qu'elle n'est pas directement portable sur d'autres machines parallèles, mais comme on l'a dit plus haut, chaque machine a ses propres spécificités avec lesquelles il faut jouer pour en tirer le meilleur parti.

7.2.2. Hypothèses sur le milieu à simuler

Avant de commencer la parallélisation proprement dite, nous allons faire quelques hypothèses sur le milieu granulaire que nous modéliserons :

1. Le milieu sera quasi statique. Nous n'aurons ainsi pas le souci de devoir rééquilibrer la charge de calcul en cours de simulation.
2. Il sera dense, uniformément réparti et confiné dans une cuve rectangulaire.
3. On n'aura pas besoin de construire la triangulation explicitement. On lira simplement depuis un fichier les positions et les caractéristiques des grains, ainsi que les arêtes de la triangulation.

Ces hypothèses nous placent dans un cas idéal qui sera facile à paralléliser pour les raisons que l'on va voir ci-après.

7.2.3. Répartition des données

Un des principaux facteurs pour une parallélisation efficace est une répartition équilibrée de la charge sur les processeurs. Étant données les hypothèses que nous avons faites, on divisera la cuve en bandes verticales d'égale largeur et chaque processeur sera responsable d'une bande. Il s'occupera des grains et des arêtes dont le centre de gravité se trouve dans sa bande. Cette répartition se fait une fois pour toutes avant de commencer la simulation. Comme le milieu est quasi statique, il sera inutile de refaire cette répartition plus tard. Pour des configurations de

milieux plus irrégulières, il faudrait évidemment une politique de répartition plus appropriée.

À l'intérieur d'une bande, toutes les données sont disponibles localement et donc chaque PE peut travailler indépendamment. Cependant, dans les bords des bandes, une arête peut relier deux disques se trouvant sur deux bandes distinctes. De même, un disque peut avoir des arêtes incidentes se trouvant sur des PE différents. Dans ces cas-là, il faudra faire des accès à des mémoires de PE distants et être particulièrement attentifs à conserver l'intégrité des données. On aurait aussi pu donner aux PE toute l'information relative au milieu granulaire, comme l'a fait O'Connor. On n'aurait sans doute pas beaucoup gagné du point de vue du temps de calcul, puisque ce que l'on économisera en temps de communication inter-PE, on le perdra lorsqu'il faudra mettre à jour les informations sur tous les PE.

Les disques et les arêtes sont stockés dans des tableaux sur chaque PE. Là où le programme séquentiel utilisait des pointeurs pour repérer les arêtes ou disques voisins, le programme parallèle utilisera ce que nous avons baptisé dans 8-[Fer95b] des *métapointeurs*. Un pointeur n'est rien d'autre qu'une adresse dans la mémoire de l'ordinateur. Dans notre cas, puisque les données sont réparties, nous avons besoin de *deux* adresses : le numéro du PE et l'indice du tableau, d'où ce nom de métapointeur.

7.2.4. Algorithme général

Les parties en italique de l'algorithme 7.1. indiquent les opérations s'effectuant sur tous les PE, tandis que les parties en romain ne sont effectuées que sur un seul. Le mot BARRIÈRE indique un point de synchronisation : tous les processeurs doivent avoir achevé leur travail avant que cette barrière puisse être franchie. La situation idéale serait qu'ils aient tous terminé en même temps, ou presque, sans quoi certains processeurs demeureront inactifs inutilement.

Algorithme 7.1 : le modèle de Cundall parallèle avec TDP

1. Lire le fichier de données et répartir les grains et les arêtes.
2. BARRIÈRE
3. *no_du_cycle:=no_du_cycle+1.*
4. **Pour chaque** arête de la triangulation:
 *déterminer les forces s'exerçant sur les grains en contact
 situés à ses extrémités.*
5. BARRIÈRE
6. **Pour chaque** grain:
 *déterminer les contacts avec les murs et calculer les forces
 correspondantes;
 calculer la résultante des forces et le moment.*
7. *Calculer les déplacements des grains en fonction de la résultante
 des forces (pour la vitesse de translation) et du moment (pour la
 vitesse de rotation) et déplacer les grains.*

```

8. Si no_du_cycle mod x = 0 alors
  8.1. BARRIÈRE
  8.2. Vérifier la validité des arêtes.
  8.3. BARRIÈRE
  8.4. Appliquer l'algorithme de flip (algorithme 4.2)
  8.5. BARRIÈRE
9. Si no_du_cycle mod y = 0 alors sauver la situation.
10. Aller à 3 tant que le temps de fin n'est pas atteint.

```

Il faut bien se rendre compte que chaque processeur exécute le même code, mais avec ses propres données. On n'a écrit qu'un seul programme qui doit être exécutable quel que soit le processeur et le nombre de ceux-ci.

Reprenons point par point cet algorithme et commentons-le.

1. Le fichier initial est créé sur SiliconGraphics en utilisant le programme séquentiel décrit au chapitre 6. C'est le PE no 0 qui lit le fichier de données et qui répartit les disques et les arêtes sur les autres PE. Chaque PE possède ses disques dans un tableau et ses arêtes dans un autre tableau.
2. Il faut attendre que l'initialisation soit terminée sur tous les PE avant de commencer la simulation proprement dite.
3. Début d'un nouveau cycle.
4. Chaque PE parcourt son tableau d'arêtes et calcule les éventuels incréments de forces associés en examinant les cailloux se trouvant aux extrémités. Il se peut que certains PE doivent aller chercher des informations sur les disques chez d'autres PE.
5. Pour être sûr que toutes les forces incrémentales ont été recalculées, il faut attendre ici que tous les PE aient fini.
6. Chaque PE parcourt son tableau de disques et, pour chacun d'entre eux, calcule les éventuelles forces provenant des murs, puis la résultante des forces afin de pouvoir l'intégrer pour connaître les déplacements. Il se peut qu'il faille aller chercher sur d'autres PE les forces mémorisées dans les arêtes.
7. Les grains sont déplacés en fonction de la résultante des forces et du moment de forces.
8. Tous les x cycles, on vérifiera que la triangulation est toujours une triangulation de Delaunay pondérée, et on la réparera le cas échéant.
 - 8.1. Il faut attendre que tous les disques aient été déplacés avant de commencer.
 - 8.2. La vérification des arêtes peut se faire localement sur chaque PE. Les arêtes illégales seront placées dans une pile sur le processeur 0.
 - 8.3. Avant de commencer à réparer, il faut que toutes les arêtes aient été vérifiées.
 - 8.4. La réparation n'est pas faite en parallèle. En fait, cela pourrait se faire, mais cela compliquerait les choses pour rien, surtout que le nombre d'arêtes à flipper est toujours faible (moins de 1%). Ici le PE 0 travaille sur toute la mémoire, i.e. la

sienne et celle des autres PE, qu'il peut modifier à distance grâce à la fonction `shmem_put`.

- 8.5. On attend que le PE 0 ait terminé l'algorithme de flip.
9. Chaque PE sauve la situation de sa bande tous les y cycles. Un petit programme spécial se charge ensuite de rassembler les morceaux pour construire un fichier global qui sera ensuite relu sur SiliconGraphics pour l'affichage graphique.
10. Le temps de fin est fixé par l'utilisateur au lancement de la simulation. Tant que cette date limite n'est pas atteinte, on continue la simulation. Le temps courant est donné par le numéro du cycle fois la durée du cycle.

On le voit, le modèle de Cundall (mais ceci est valable pour tous les modèles de l'école de corps déformables) se parallélise facilement. Tout l'algorithme 6.1 a pu être parallélisé, exception faite de la réparation de la TDP, mais cela n'a que très peu d'influence néfaste.

7.2.5. Accélération

Pour avoir une idée des performances, on utilise généralement deux mesures : *l'accélération* (speedup en anglais) $S_n = T_1/T_n$ où T_n est le temps d'exécution obtenu avec n PE, et *l'efficacité* $E_n = S_n/n$. Comme test, on a repris la simulation "1-10" du § 6.5 à une plus grande échelle, puisque 20'000 disques compose le remblai. On a simulé 5 millisecondes réelles avec un pas de temps de 10^{-5} seconde, soit 500 cycles de l'algorithme 7.1, sans faire de sauvegardes et en vérifiant la triangulation tous les 10 cycles. À l'exception de la zone d'impact, le milieu est quasi statique et la triangulation nécessite donc peu de réparations.

Nombre de PE utilisés sur le T3D	Temps d'exécution (en secondes)	Accélérations par rapport à 1 seul PE	Accélérations par rapport à la SGI
SGI	738	-	-
1	1674	-	0.45
2	852	1.97	0.87
4	436	3.84	1.69
8	222	7.54	3.32
16	115	14.56	6.41
32	61	27.44	12.10
64	35	47.83	21.09
128	22	76.09	33.55

Tableau 7.1. Performances

Ces performances pourraient être légèrement améliorées en répartissant mieux les arêtes et les disques. On a en effet noté qu'avec l'algorithme simpliste de répartition que l'on a utilisé, il

pouvait y avoir une variation de 10% du nombre d'arêtes et de disques entre les PE.

On peut aussi comparer ces temps de calcul avec le temps mis sur une station de travail SiliconGraphics Indigo avec un processeur de 100 MHz (SGI). Curieusement, il faut plus de deux PE pour être plus rapide que la SGI. Les deux processeurs sont pourtant pratiquement équivalents. L'explication vient entre autres du fait que la SGI gère mieux la mémoire cache* ou que les changements que nous avons été contraints de faire dans la structure de données (remplacement des pointeurs par des métapointeurs) ont affaibli les performances sur le T3D.

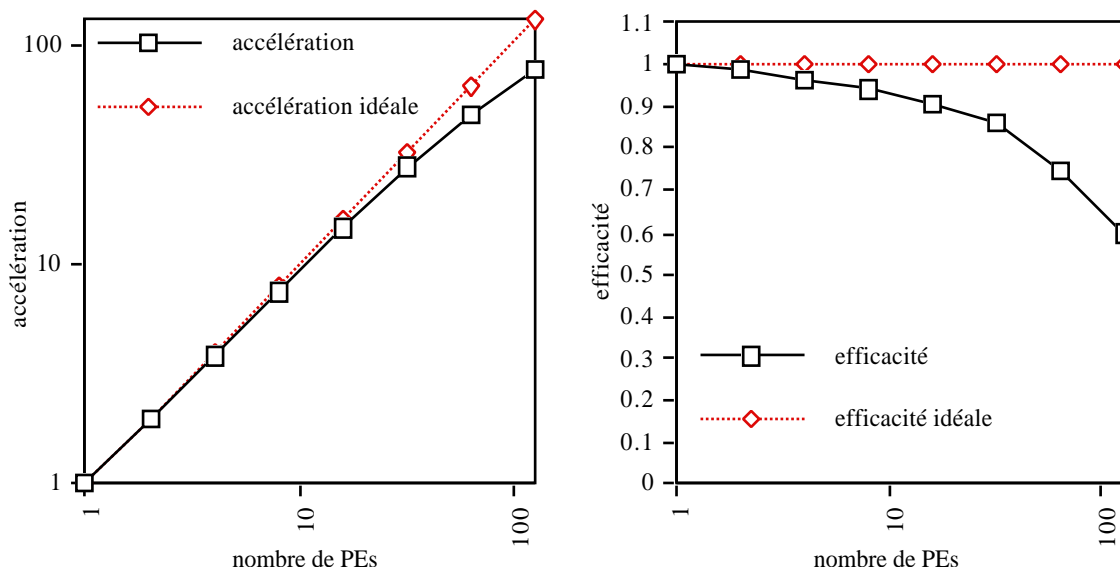


Figure 7.1. Accélération et efficacité

L'accélération est proche de l'idéal. L'efficacité diminue avec le nombre de PE parce que les bandes deviennent trop minces, ce qui a pour conséquence de faire augmenter les communications. Quand le nombre de PE atteint une certaine limite, il faudrait essayer de répartir les disques et les arêtes dans un quadrillage plutôt que dans des bandes.

7.3. Conclusion



Le parallélisme semble être la voie la plus prometteuse pour la simulation numérique des milieux granulaires par une méthode d'éléments distincts, quelle que soit la fonction de voisinage choisie. Il reste cependant beaucoup de travail pour faire un programme général. Nous nous sommes volontairement placés dans une situation très avantageuse : simuler un milieu quasi statique confiné dans une cuve rectangulaire. Un programme parallèle efficace serait plus difficile à obtenir si l'on voulait simuler par exemple l'écoulement de disques dans un sablier. En effet, comme le milieu évoluerait rapidement, il faudrait redistribuer la charge de travail fréquemment. Même la répartition initiale des disques n'est pas évidente si les murs latéraux ne sont pas verticaux, car la découpe en bandes verticales ne répartit plus équitablement les données.

Il faut peut-être aussi relativiser un peu les choses. Comme le Cray T3D est un superordinateur très cher, il est à la disposition du plus grand nombre d'utilisateurs possible afin de le rentabiliser. Cette politique implique que l'on ne peut pas monopoliser un grand nombre de processeurs trop longtemps. Typiquement, si on a besoin de 64 processeurs, on ne les aura que pendant 30 minutes. Il faut donc prévoir de sauvegarder la situation régulièrement dans un fichier. Ensuite, on pourra relancer une requête avec ce nouveau fichier, mais il est possible que l'on doive attendre un certain temps avant que notre tour ne revienne. Les accélérations que l'on a présentées ne tiennent pas compte de cette contrainte.

Notre triangulation s'adapte parfaitement au parallélisme, mais des techniques plus classiques fonctionneront tout aussi bien. Le principe de voisinage ne doit certainement pas être éliminé sous prétexte que l'on utilise un ordinateur puissant.

Il semblerait que seule l'école des corps déformables soit facilement parallélisable. Cependant, dans le cas de l'école des corps indéformables, on pourrait sans doute utiliser la méthode que Caroline Hogue expose dans sa thèse¹. C'est en fait une méthode hybride entre les deux écoles. En effet, elle considère les chocs comme instantanés, mais elle ne scrute le milieu qu'à intervalles de temps réguliers. Évidemment surgit parfois le problème dont nous avons déjà parlé, à savoir que plusieurs grains peuvent en chevaucher un autre. Elle adopte alors une technique pour classer les chocs par temps d'occurrence afin de pouvoir utiliser les formules du chapitre 3. On peut ainsi se débarrasser de l'échéancier qui nous contrarie tant pour la parallélisation.

En résumé, le parallélisme pourrait bien être le véritable détonateur de la méthode des éléments distincts, puisque le nombre de grains peut facilement être de plusieurs dizaines de milliers, tout en gardant des temps de calcul raisonnables.

¹voir 3-[Hog93a], chapitre 6