DÉPARTEMENT D'INFORMATIQUE

Faculté des sciences

Université de Sherbrooke

GÉNÉRATION DE TERRAIN ALÉATOIRE

À L'AIDE DE CALCUL PARALLÈLE MPI

par

MAEL PERREAULT

travail présenté à

GABRIEL GIRARD

dans le cadre du cours

IFT 630

Processus concurrents et parallélisme

Sherbrooke

AVRIL 2012

Table des matières

Introduction

Ce projet vise à évaluer l'avantage du calcul parallèle lors de la génération de contenu aléatoire.

Problématique

Lors de la création de jeux vidéo, beaucoup de temps est accordé à la création de l'univers de jeu. Celui-ci est composé de contenu réutilisé en diverses dispositions afin de créer le divertissement. Les méthodes de création de jeu habituelles proposent de créer l'univers pièce par pièce à la main, car cela certifie que le contenu est disposé d'une manière à créer le maximum de divertissement. Cependant, cette méthode est très coûteuse en temps. De plus, le divertissement est souvent créé par l'agencement de pièces dont on n'aurait pas immédiatement pensé. Il est donc sensé d'essayer plusieurs agencements avant de se décider sur celui qui devrait être intégré au jeu.

Certains éléments du jeu sont longs à générer et produisent un facteur de divertissement négligeable notamment le terrain. Celui-ci n'est souvent qu'un médium sur lequel les autres éléments du jeu sont disposés. Il serait donc bénéfique de pouvoir générer des exemples de terrain selon certains critères et en choisir un qui convient à nos besoins de création plutôt que de tous les créer manuellement. Certains éléments du jeu sont longs à générer et produisent un facteur de divertissement négligeable notamment le terrain. Celui-ci n'est souvent qu'un médium sur lequel les autres éléments du jeu sont disposés. Il serait donc bénéfique de pouvoir générer des exemples de terrain selon certains critères et en choisir un qui convient à nos besoins de création plutôt que de tous les créer manuellement.

Dans ce projet, nous nous intéresserons au cas de génération de terrain pour un jeu de type « Turned-Based Strategy ». Il s'agit d'un cas simple qui peut démontrer les possibilités de MPI. Dans ce type de jeu, le terrain est une grille composée de cases, souvent une formes géométriques. Chacune de ces cases est liée avec ses voisins, un par côté. Sur cette grille, le joueur contrôle une armée de personnages qui naviguent tour à tour afin d'attaquer et vaincre l'armée adverse. Les personnages ont diverses caractéristiques, comme le nombre de cases qu'ils peuvent traverser en longueur ainsi qu'en hauteur.

Risk[1] (1957)

Inspiration des jeux « Turned-Based Strategy »

Civilization[2] (1997)

Un des premiers jeux « Turned-Based Strategy »

Nous allons choisir la disposition la plus simple, c'est-à-dire celle utilisée dans les jeux « Final Fantasy Tactics » ainsi que « Tactics Ogre ». La grille sera composée de cases carrées qui seront liées à 4 voisins. La principale caractéristique des cases dans ces jeux est leur hauteur.

Final Fantasy Tactics[3] (1997)

Tactics Ogre : The Knight of Lodis[4] (2001)

Objectifs

À l'aide de MPI, générer des grilles pour ce type de jeu. Intégrer un calcul de différence de hauteur entre les cases afin de permettre de créer des grilles plus ou moins plates. Les 4 voisins d'une case doivent donc avoir une différence de hauteur moindre ou égale au maximum défini.

[1] "risk-board." Photo. TheMovieDoc.com. 10 juin 2011. 25 avril 2012. <http://themoviedoc.com/risk/>
[2] "Civ 1." Screenshot. 101VideoGames.wordpress.com. 24 mars 2011. 25 Avril 2012. <http://101videogames.wordpress.com/2011/03/24/97-civilization-ii/>
[3] "A Battle in Final Fantasy Tactics." Final Fantasy Wiki. 18 mars 2012. 25 Avril 2012. <http://finalfantasy.wikia.com/wiki/Final_Fantasy_Tactics>
[4] "Tactics Ogre." Screenshot. Push cx. 24 mai 2009. 25 avril 2012. <http://push.cx/2009/game-influences-tactics-ogre>

Méthodologie

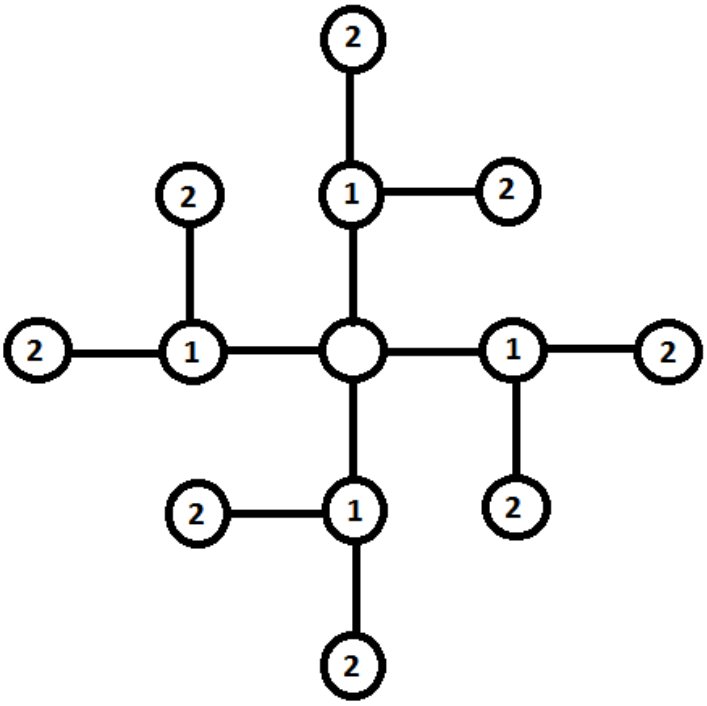
Parcours de la grille

À des fins de simplification et d'unification des calculs, nous supposons des grilles carrées comme 5x5 ou 100x100.

Comme la hauteur des cases adjacentes est importante, il faut calculer la hauteur d'une case uniquement lorsque les voisins préalables sont calculés. À l'inverse, on peut aussi définir les voisins si on connait une case. C'est pourquoi nous allons fixer la valeur d'une case initiale pour nos calculs. Voici un exemple d'exécution pour une grille simple 5x5. Les chiffres représentent à quelle itération on peut calculer la case et donc à la 5ème itération l'algorithme se termine.

4	3	2	3	4
3	2	1	2	3
2	1	(Départ)	1	2
3	2	1	2	3
4	3	2	3	4

Grâce à cet exemple, on peut déduire une grille de précédence entre les cases qui donnerait quelque chose comme ceci :



On suppose alors que les branches principales, celle partant du centre, vont produire 2 nouvelles cases par itération, tandis que les branches secondaires vont uniquement produire une case, et ce, jusqu'à l'épuisement du graphe.

Note : Alors si l'on choisit un des coins de la grille comme point de départ, on perd une grande partie des cases par itération. Il est donc préférable d'utiliser le milieu de la grille.

Accélération à l'aide de MPI

Pour accélérer le calcul, on passe donc chacune des cases à calculer à un fil de MPI. On doit attendre que toutes les cases d'une certaine itération soient calculées avant de passer à la prochaine. Par exemple, les cases 1 dans le graphe précédent doivent être calculées avant les cases 2.

Donc, plus la grille est grande, plus on risque d'arriver à un nombre important de cases à calculer en même temps.

Résultats

Exemple d'exécution

Les matrices inférieures à 25x25 sont affichées à la console afin de pouvoir vérifier l'algorithme.

```
Debut du calcul d'un terrain : 25 par 25.
5 %
10 %
15 %
20 %
25 %
30 %
35 %
40 %
45 %
50 %
55 %
60 %
65 %
70 %
75 %
80 %
85 %
90 %
95 %

Matrice resultat

10  8  6  7  8  6  4  4  3  1  1  3  2  0 -2  0  2  0 -2 -4 -6 -7 -7 -7 -7
11 10  8  8  9  7  6  4  3  1  0  1  0 -2 -3 -2  0 -2 -3 -2 -4 -5 -7 -8 -8
13 11 10  8  9  7  6  6  5  3  1 -1 -1 -3 -5 -4 -2 -1 -3 -2 -4 -3 -5 -7 -8
11 13 11 10  9  9  7  5  5  4  3  1  0 -2 -3 -2  0 -1 -3 -2 -3 -4 -6 -8 -10
11 12 10 11 11 10  8  7  7  5  4  3  2  0 -1 -1 -1 -1 -3 -3 -1 -3 -4 -6 -8
 9 11 12 12 13 11  9  9  8  6  4  3  3  1 -1  0  1  1 -1 -1 -2 -1 -2 -4 -6
11 13 14 13 11 13 11 11 10  8  6  5  4  2  1  2  3  2  1  0 -1  1 -1 -2 -4
10 12 12 11 11 11  9  9  8 10  8  6  5  3  2  0  2  0 -1 -2 -1  0  0 -1 -3
12 11 11 12 13 11  9  8  9  9  9  7  7  5  3  2  1  2  1  0 -2  0  2  1 -1
10 12 12 14 15 13 11 10  9  8  9  8  7  5  4  2  0  1  2  1  0 -1  1 -1 -1
 8 10 10 12 13 13 13 12 10  9 11 10  9  7  6  4  2  0  0  1  0  0  2  1  1
 7  9  8 10 11 13 11 10  9  7  9 11 11  9  7  5  4  2  2  2  0  1  3  1  0
 9 11  9 10 12 13 12 12 10  8 10 10 10 11  9  7  6  4  2  2  0  0  2  1  0
 8  9 11  9 11 13 12 12 11 10  9 11  9  9 11  9  7  5  4  2  1 -1  1  2  0
 9 11 13 11 10 12 12 14 12 11 10  9 10  9  9 11  9  7  6  4  3  1  0  2  1
11 12 14 13 11 11 12 12 14 13 12 11 11 10  8 10 11  9  7  5  3  2  2  0  2
 9 10 12 13 13 13 11 13 15 13 12 13 13 12 10  8  9 10  8  7  5  4  3  2  3
 7  9 10 12 12 12 10 11 13 15 14 15 15 14 12 10  9  8  8  9  7  6  4  3  2
 9 11 10 12 13 11  9 10 11 13 13 14 14 14 14 12 10 10  9  9  8  8  6  4  2
11 11 12 11 12 12 10  8  9 11 12 13 15 15 14 12 10  8  8  7  9  8  6  6  4
 9 11 11 12 13 13 11 10 10 10 12 12 13 14 12 13 12 10 10  9 11 10  8  8  6
 7  9 10 10 11 11  9 11 12 12 13 14 14 12 13 15 13 11 11  9  9 11  9 10  8
 7  9 10  9 11  9 10  9 10 10 11 13 14 12 14 16 14 13 13 11  9  9  8  9  7
 8  7  8  7  9 11 11 11 10 11  9 11 12 14 12 14 15 15 13 12 11  9 10  8  6
10  8  8  8  8 10 11  9  8 10  9 11 11 13 13 14 15 17 15 14 12 10 11 10  8
Temps d'execution: 0.026000
```

Temps d'exécution

Pour une matrice 1000x1000.

0 travailleur	**2 travailleurs**	**4 travailleurs**
7.169	20.914	21.587
6.812	20.361	22.654
6.797	19.851	21.012

Conclusion

Si les cases n'étaient pas dépendantes les uns des autres, on trouverait probablement un grand avantage à paralléliser le problème. Malheureusement, à mon grand étonnement, le coût de maintenir et attendre l'information de plusieurs thread est plus grand que d'effectuer le travail soi-même, car un nombre limité de cases peuvent être calculées en même temps.