Parallélisation du suivi de marqueurs sur une séquence vidéo

Faycal TOUHAMI, Jean-Marc TRÉMEAUX

11 mars 2005

Résumé

Ce document présente une architecture de calcul parallèle permettant d'effectuer le suivi de nœuds d'une grille régulière sur une séquence video. Notre objectif est de réfléchir à la parallélisation de certaines techniques de suivi et de tester la validité de celles-ci.

Mots-clés : Parallélisme, Gros grain, MPI, Analyse d'image, Marqueurs, Suivi

Table des matières

1 Introduction

Notre travail se déroule dans le cadre suivant. Nous disposons d'une séquence d'images vidéo, chacune d'elle représentant une grille régulière projetée sur le thorax d'un patient. Un outil a précédement [1] été réa-lisé afin de suivre le déplacement de chaque nœud de la grille, c'est-à-dire ses coordonnées successives au cours du temps.

Nous allons explorer les techniques permettant d'exécuter le même algorithme en parallèle sur p processeurs. Puis nous mettrons en oeuvre une simulation de fonctionnement sur des données simplifiées.

Le calcul en parallèle permettra peut-être de gagner en performance et de traiter le flux d'entrée vidéo en temps réel. D'autre part, on peut espérer traiter des volumes de données plus importants, c'est-à-dire des images en plus haute résolution ou un nombre de nœuds plus important.

2 Méthode

2.1 Génération des données d'entrée

Afin de simplifier le problème, nous utilisons notre propre version de la donnée d'entrée, c'est-à-dire de la vidéo à analyser. Cela nous permet d'éviter les artefacts liés à l'acquisition de la vidéo. En particulier, nous n'avons pas à effectuer un traitement bas niveau des clichés afin d'améliorer la qualité de l'image.

La grille est générée procéduralement, ce qui nous permet de nous assurer de la cohérence de celle-ci. Par exemple, nous sommes certains que les nœuds ne se chevauchent pas. La première image générée permet une amorce de l'algorithme de suivi. Pour celle-ci, la position de nœuds est marquée "manuellement". Comme nous générons les images à partir d'une formule mathématique, nous pouvons écrire automatiquement la position initiale des nœuds dans un fichier. Les images restantes sont ensuite générées et sauvegardées.

2.2 Algorithme de suivi

L'algorithme de suivi mis en œuvre est un algorithme simplifié. Nous développons tout d'abord un programme de suivi séquentiel des nœuds de la grille afin de produire un premier résultat. La sortie de ce programme est une trajectoire associée à chaque nœud suivi.

Enfin, nous modifions ce programme pour mettre en oeuvre l'algorithme parallèle, et nous vérifions que celui-ci fournit les mêmes résultats que l'algorithme séquentiel.

3 Génération des clichés

Le premier programme réalisé est celui de la génération des clichés composant la séquence video fictive.

En effet, les images de la cage thoracique destinées à servir d'entrée au programme ne nous ont pas été fournies, dans un souci de simplification du problème.

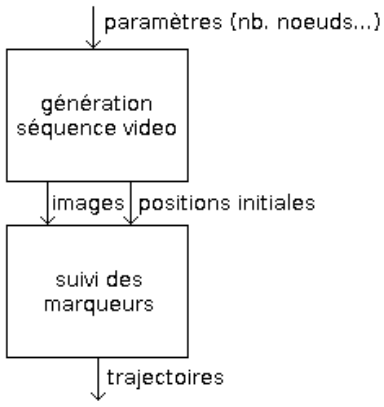

Fig. 1 – Génération des clichés puis suivi.

3.1 Placement des marqueurs

Les marqueurs sont placés sur une grille quasi-régulière de taille fournie par l'utilisateur (*cf.* figure 2). Chaque marqueur est représenté sur un cliché par un pixel blanc sur fond noir. Les coordonnées de celui-ci varient au cours du temps le long d'une courbe paramétrée comprise dans une région rectangulaire fixée, appelée *fenêtre de tracé*. Ces fenêtres sont choisies à priori de manière à ce qu'elles ne se chevauchent pas, un croisement des marqueurs pouvant poser problème lors de la phase de suivi.

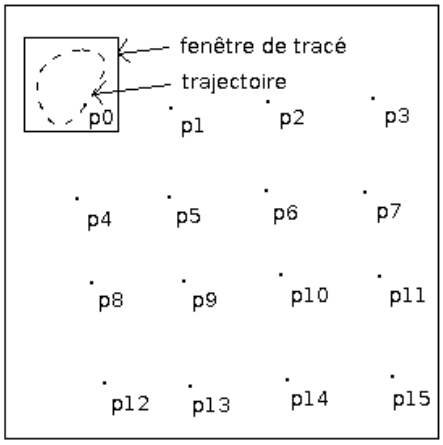

Fig. 2 – Grille quasi-régulière de 4x4 marqueurs.

Puisque l'on peut considérer que la respiration est un phénomène périodique, il nous a semblé intéressant de déplacer chaque nœud au cours du temps le long d'une courbe paramétrée au comportement périodique.

Ainsi, nous allons utiliser des équations de la forme :

$$\begin{cases} x = a * cos(\omega_1 * t) + x_0 & \text{pour l'abcisse} \\ y = b * cos(\omega_2 * t) + y_0 & \text{pour l'ordonnée} \end{cases}$$

Les coefficients a et b déterminent la taille de la fenêtre de tracé. Ces deux paramètres, ainsi que ω_1 et ω_2 (vitesses angulaires selon les axes horizontaux et verticaux) sont fixés aléatoirement à l'initialisation pour chaque nœud. La variable t est la date du cliché.

Le point (x_0, y_0) représente le centre de la fenêtre de tracé.

Pour chaque nœud généré, le centre (x_0, y_0) de sa fenêtre de tracé est placé sur un quadrillage régulier de l'image de taille spécifiée par l'utilisateur.

3.2 Sauvegarde de la séquence vidéo

Comme nous l'avons mentionné précédemment, les clichés sont composés d'un ensemble de points blancs sur fond noir. Pour cette raison, il est inutile de choisir un format de sauvegarde trop complexe.

Notre choix c'est porté sur le format PBM (Portable Bitmap Format, Black & White). Ce format à l'avantage d'être très simple à manipuler et léger : chaque pixel est représenté par un bit seulement. Lors du suivi, les opérations d'entrée/sortie ne seront ainsi pas pénalisantes par rapport au reste de l'algorithme.

Les clichés sont sauvegardés au format PBM dans des fichiers nommés successivement $1, 2, ..., N$ afin d'être relus ultérieurement dans le même ordre.

3.3 Sauvegarde de la position initiale de chaque marqueur

Le cliché 0 correspond à la position initiale de chaque nœud. Pour celui-ci, nous sauvegardons au lieu du fichier PBM un fichier *0.pos*, pour *position*.

Celui-ci contient tout d'abord la taille de la grille, spécifiée par l'utilisateur. Ensuite, elle contient pour chaque nœud de la grille un couple (x_0, y_0) de coordonnées initiales.

3.4 Utilisation du programme

La ligne de commande pour effectuer la génération de la séquence vidéo est de la forme suivante :

gene -p <prefixe> -n <nb clichés> -l <largeur image>
-h <hauteur image> -m <nb points>

- *prefixe* est l'emplacement de sauvegarde des clichés générés.
- *nb clichés* est le nombre de clichés à génerer.
- *largeur image, hauteur image* spécifient la taille des clichés en pixels.
- *nb points* spécifie la taille de la grille horizontalement et verticalement, en nombre de marqueurs.

4 Suivi séquentiel des marqueurs

4.1 Algorithme général

Le premier programme réalisé consiste à résoudre le problème de suivi en employant un seul processeur. Ce programme prend en paramètres le nombre de clichés à analyser et l'emplacement de ceux-ci sur le système de fichiers.

L'algorithme est résumé comme suit :

- Charger la taille de la grille et la position initiale (cliché 0) de chaque nœud.
- Pour i de 1 à *nombre de clichés* :
  - Charger le cliché i
  - Pour j de 1 à *nombre de nœuds*
    - Calculer le déplacement du nœud j du cliché $i-1$ au cliché i
  - Sauvegarder le déplacement de chaque nœud

4.2 Calcul du déplacement d'un marqueur

Cet algorithme a pour but de calculer le vecteur de déplacement d'un nœud donné entre deux clichés successifs.

L'algorithme original ne travaille pas sur des pixels isolés mais sur des blocs, dont la similarité est calculée par une fonction de corrélation croisée normalisée [2]. Dans cette version simplifiée du problème, on recherche simplement la position d'un pixel allumé représentant le marqueur, les pixels éteints représentant le fond de l'image.

Pour cela, nous définissons, sur le cliché i, une fenêtre de recherche autour du point de coordonnées (x_{i-1}, y_{i-1}). Nous supposons d'une part que le point recherché se situe à l'intérieur de cette fenêtre (ce qui est le cas si nous avons généré les clichés convenablement), et d'autre part que le point aura peu bougé d'un cliché sur l'autre.

Nous parcourons alors la fenêtre de recherche *en spirale* (*cf.* figure 3) en partant du centre de celle-ci. L'algorithme s'arrête lorsque le pixel testé est allumé. Si la spirale sort de la fenêtre de recherche, alors l'algorithme échoue.

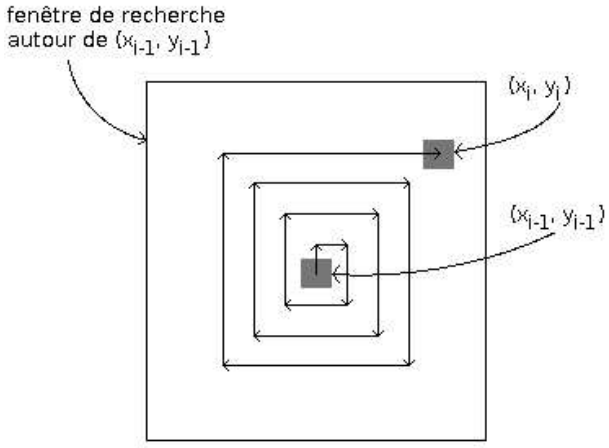

FIG. 3 – Recherche en spirale autour d'un point.

4.3 Sauvegarde des trajectoires

Le programme de suivi enregistre les résultats du calcul dans un fichier *.suivi* similaire au fichier de marquage des positions initiales vu précédemment. Chaque ligne du fichier correspond à un nœud, et représente les coordonnées successives de celui-ci sous la forme :

$$(x_0, y_0), (x_1, y_1), ..., (x_N, y_N)$$

4.4 Utilisation du programme

La ligne de commande pour effectuer le suivi séquentiel des marqueurs est de la forme suivante :

test -p <prefixe> -n <nb clichés>

- *prefixe* est l'emplacement où se trouvent les clichés à analyser (fichiers .pos et .PBM).
- *nb clichés* est le nombre de clichés à analyser.

5 Environnement de calcul parallèle

Dans cet partie, nous allons décrire la plate-forme matérielle et logicielle employée pour résoudre le même problème en parallèle sur p processeurs.

5.1 Organisation de la mémoire

L'architecture d'une plate-forme de calcul parallèle peut employer une mémoire partagée, une mémoire distribuée ou un modèle hybride des deux.

Dans une organisation en mémoire partagée, les p processeurs agissent indépendemment mais accèdent à la même ressource mémoire. Les avantages sont une facilité de programmation et un partage rapide des données entre les processus. L'inconvénient est un coût prohibitif lorsque le nombre de processeurs augmente.

Dans une organisation en mémoire distribuée (*cf.* figure 4), les p processeurs possèdent chacun leur propre mémoire et communiquent par échange de signaux à travers un réseau d'interconnection. L'avantage de ce modèle par rapport au précédent est la *scalabilité* de la mémoire : celle-ci augmente proportionnellement avec le nombre de processeurs. De plus, ce type d'architecture est facile à mettre en oeuvre. En effet, un réseau Ethernet classique constitue une architecture à mémoire distribuée. La difficulté vient cependant de la nécessité d'adapter les structures de données du programme à cette organisation particulière, et de gérer la communication et la synchronisation entre les processeurs.

Le choix de l'organisation de la mémoire pour notre travail s'est porté naturellement sur une mémoire partagée. Les architectures de calcul haute performance utilisent généralement un modèle hybride de mémoire partagée et distribuée.

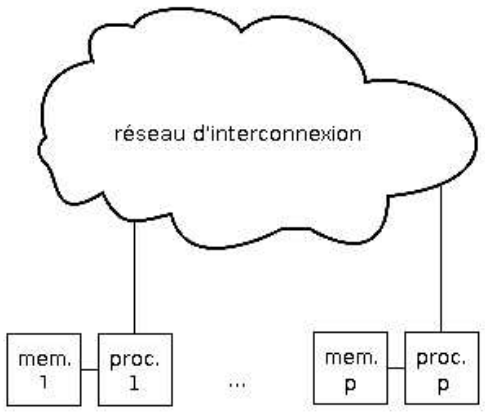

Fig. 4 – Organisation de mémoire distribuée.

5.2 Modèle de programmation

Nous nous intéressons au modèle de programmation **SPMD**. Dans celui-ci, un même programme est exécuté simultanément sur les p processeurs. À un instant donné, chaque processus peut exécuter des instructions identiques ou non au sein du même programme. Pour effectuer des traitements différents, le programme peut contenir des branchement ou des blocs conditionnels d'instructions. Chaque processus peut travailler sur des données différentes.

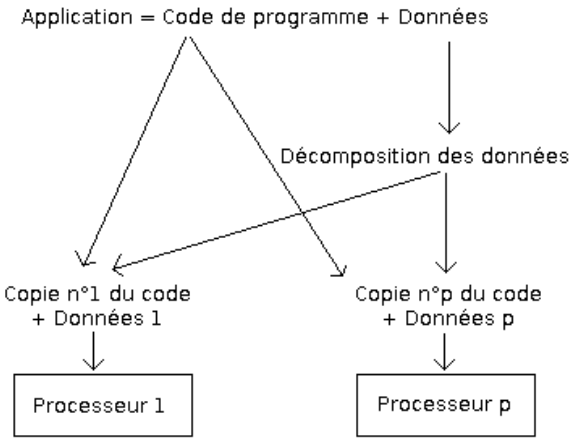

Fig. 5 – Programme SPMD (Single Programe Multiple Data).

5.3 MPI

La bibliothèque MPI [3] (Message Passing Interface) est le standard de communication pour les programmes parallèles s'exécutant sur des architectures de type mémoire distribuée. Elle fournit des primitives de haut niveau, permettant de faire abstraction de l'architecture réseau sous-jacente et ainsi d'écrire des programmes simples et portables.

La phase de développement d'un programme MPI se fait pour un nombre quelconque de processeurs. À l'exécution, on choisit de déployer p processus sur q processeurs par une commande du type :

mpirun -np p -machinefile mf <programme>

Le fichier mf contient la liste des processeurs (identifiés par leur nom ou leur adresse IP) sur lesquels on veut déployer le programme. On peut ainsi, lors du développement et du test du programme, utiliser un unique processeur (la machine de développement), sur lequel ont déploie p processus. Lors du passage en production, il suffit d'adapter la ligne de commande pour déployer le programme parallèle sur un plus grand nombre de processeurs.

Chaque processus est identifié par son rang, unique, obtenu par la primitive *MPI_Comm_rank()*. La primitive *MPI_Comm_size()* permet de connaître le nombre total de processus. À l'aide de ces deux primitives, les différents processus d'un programme SPMD peuvent se différencier et effectuer chacun le traitement qui leur incombe.

5.4 Communications

Les communications se font au choix sur un mode synchrone ou asynchrone.

Une communication binaire synchrone se fait sur le principe du rendez-vous. Lors de l'appel à une primitive de communication synchrone, l'exécution est bloquée dans le processus qui effectue un envoi (respectivement une réception), en attendant l'exécution d'une reception (respectivement un envoi) de son partenaire (*cf.* figure 6).

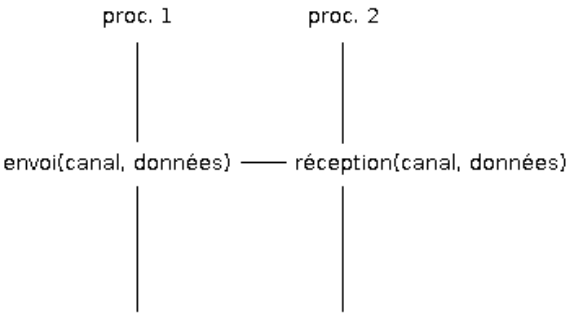

FIG. 6 – Principe du rendez-vous entre deux processus.

Le rendez-vous peut se faire avec ou sans échange de données. Dans ce dernier cas il s'agit d'une synchronisation pure. Les primitives de MPI permettant de mettre en oeuvre ces communications sont *MPI_Send()* et *MPI_Recv()*.

Les communications asynchrones se font de façon analogue grace aux primitives non-bloquantes suivantes : *MPI_ISend()* et *MPI_IRecv()*.

On peut également effectuer une synchronisation collective de l'ensemble des processus grace à la primitive *MPI_Barrier()*.

6 Suivi parallèle des marqueurs

6.1 Identification du code à paralléliser

Le calcul du déplacement d'un nœud d'une image sur l'autre constitue le goulot d'étranglement de notre programme séquentiel. C'est en effet la boucle la plus imbriquée de notre algorithme, et donc la fonction à paralléliser en premier lieu.

Par chance, le calcul se fait sur des fenêtres de recherche disjointes et l'information nécessaire est entièrement contenue dans celles-ci. Nous faisons donc face à un problème parallélisable par excellence. En effet, le découpage en sous-problème est trivial, ceux-ci sont indépendants et ne donc requièrent aucune communication entre eux. Dans ce cas de figure, on parle de parallélisme *gros grain* : une grosse quantité de calcul est effectuée entre chaque opération de communication.

Notre problème se résume donc à distribuer les données entre les processus, effectuer le calcul sur chacun d'eux puis rassembler et sauvegarder les résultats.

6.2 Partitionnement des données

Une première idée de découpage consiste à attribuer un nœud à chaque processus. Cette répartition est peu souple car elle requiert d'avoir autant de processeurs que de marqueurs à suivre. Si l'on augmente le nombre de nœuds de la grille, on ne peut pas espérer que le nombre de processeurs suive et cette solution n'est donc pas *scalable*.

Pour une grille de n nœuds et p processus, on convient donc d'attribuer à chaque processus un ensemble de nœuds de taille $n/(p-1)$. Le processus de rang 0 a pour rôle d'effectuer le partitionnement, la dispersion des données et le rassemblement des résultats. Si n n'est pas divisible par p, le dernier processus se voit attribuer un nombre inférieur de nœuds, ce qui ne pose pas de problème. Les nœuds sont simplement affectés dans l'ordre où ils sont lus dans le fichier *.pos* (*cf.* figure 7).

6.3 Répartition de charge

Idéalement, il faudrait tenir tous les processeurs occupés tout le temps. Cela revient à minimiser le temps d'attente à l'intérieur de chaque processus. Pour cela, il y a deux approches possibles.

La première consiste à utiliser une répartition *à priori* de la charge, si l'on suppose que le temps de calcul et le temps de communication est identique sur chacun des processus. Cette hypothèse peut être faite si les machines sont de même puissance et si le réseau d'interconnexion le permet. Il faut aussi veiller à ce que le temps de calcul sur chacune des données d'entrée soit identique. Dans le cas où l'une de ces conditions n'est

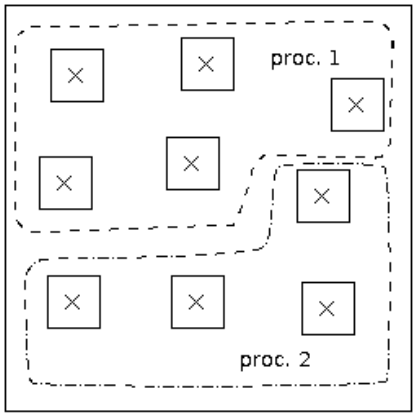

FIG. 7 – Partitionnement d'un cliché en deux ensembles de blocs ($p = 3$, le processus de rang 0 effectue le partitionnement).

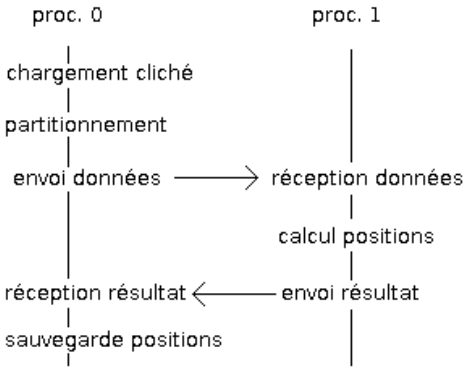

FIG. 8 – Algorithme parallèle d'analyse d'un cliché (communications entre les processus de rang 0 et 1).

pas vérifiée, un processeur risque d'être plus lent que les autres et donc de brider les performances de l'ensemble du réseau. D'un point de vue implémentation, les processus effectuent leurs calculs en même temps, et nous utilisons donc des communications synchrones.

La seconde répartition possible est de stocker les tâches à traiter dans une file d'attente. Chaque processeur prélève un élément de cette file lorsqu'il a du temps disponible. Cette solution est enviseagable si l'on dispose d'un réseau de machines plus hétérogène.

Dans le cadre de notre travail, nous avons opté pour une répartition statique de la charge.

6.4 Algorithme de suivi

Le programme parallèle se déroule comme suit. À l'initialisation, le processus de rang 0 se différencie. Il prend le rôle du chargement des données à partir du système de fichiers, du partitionnement, de la distribution et du rassemblement des résultats. Il sera appelé par la suite *processus père*. Les processus de rang supérieur à 0, appelés par la suite *processus fils*, effectueront la réception des données provenant du père, le calcul et enfin l'envoi au père des nouvelles positions trouvées.

Une itération de l'algorithme de suivi se présente comme illustré par la figure 8. À chaque itération, le processus de rang 0 effectue deux opérations de communication (envoi des données puis réception du résultat) avec chacun des autres processus. Les processus de rang supérieur à 0 ne communiquent jamais entre eux.

Le message envoyé du père au fils comporte tout d'abord le nombre de nœuds à suivre (n/p), puis un ensemble de structures de la forme suivante :

- Position (x_{i-1}, y_{i-1}) du nœud suivi dans le cliché précédent.
- Region rectangulaire de l'image correspondant à la fenêtre de recherche.

Le message envoyé en retour du fils à son père ne contient simplement, pour chaque nœud, que sa position (x_i, y_i) dans l'image en cours d'analyse.

6.5 Utilisation du programme

La ligne de commande pour effectuer le suivi parallèle des marqueurs est de la forme suivante :

mpirun -np <nb processus> -machinefile <machines>
testP1 -p <prefixe> -n <nb clichés>

- *nb processus* est le nombre de processus à exécuter.
- *machines* est la listes des machines disponibles.
- *prefixe* est l'emplacement où se trouvent les clichés à analyser (fichiers .pos et .PBM).
- *nb clichés* est le nombre de clichés à analyser.

7 Conclusion et résultats

Le programme de génération a été développé en décembre, conjointement par les deux membres du binôme. Les programmes de suivi séquentiel puis parallèle ont été écrit en février après réflexion avec l'encadrant puis entre membres du binôme sur la marche à suivre. Le volume horaire total est d'environ 50 heures chacun, dont 5 heures de travail avec l'encadrant, 15 heures de programmation, 15 heures de rédaction du rapport et 15 heures de recherche et de documentation.

Le suivi parallèle a été testé dans une salle machine du bâtiment *Nautibus*, sur un ensemble de 9 machines (de *b710pjt* à *b710pkb*).

Cela nous a permis de verifier le fonctionnement du programme parallèle, c'est-à-dire s'il se terminait effectivement et s'il donnait les mêmes résultats que le programme séquentiel sur un jeu d'essai.

Cependant, nous n'avons pas pu mesurer une amélioration des performances. En effet, l'algorithme *boîte noire* de suivi employé est une version très simplifiée de

l'algorithme original, et ses temps de calculs sont donc trop réduits pour calculer sa durée d'exécution. De plus, le réseau informatique du bâtiment *Nautibus* n'est pas dédié au calcul à haute performance. Des technologies telles que Quadrics [4] ou Infiniband [5] permettent de garantir des communications de plus faible latence et de meilleur débit.

Une poursuite des travaux pourrait se faire en tentant d'interpréter le mouvement de chaque nœud, afin d'identifier des cycles et ainsi de prédire avec plus de précision la position des marqueurs dans le temps.

Une autre extension serait de tenter une méthode différente de suivi. La première technique envisagée est une subdivision récursive de l'image en quadrants, jusqu'à trouver le nœud recherché. La seconde est un apprentissage artificiel par réseau de neurones, celui-ci constituant un appoximateur de fonction. Ces deux types d'approches ont l'avantage d'être de nature intrinsèquement parallèle.

Références

[1] Reconstruction of breathing movements from videos, O. Govokhina, É. Desserée & J-M. Moreau, Proceedings of SURGETICA 2005, pp. 193-200, Chambéry, France, Février 2005.

[2] N. D'Apuzzo and R. Plänkers, Human body modeling from video sequences. International Archives of Photogrammetry and Remote Sensing 32(5-3W12), pp. 133-140, 1999.

[3] M.P.I Forum. MPI : A Message-Passing Interface Standard (version 1.1). Technical report, University of Tenessee, Knoxville, Tennessee, June 1995. Disponible sur http ://www.mpi-forum.org/.

[4] Performance Evaluation of the Quadrics Interconnection Network, Fabrizio Petrini, Salvador Coll, Eitan Frachtenberg and Adolfy Hoisie. Disponible sur http ://www.c3.lanl.gov/ fabrizio/papers/jcc.pdf

[5] High Performance RDMA-Based MPI Implementation over InfiniBand, Jiuxing Liu, Jiesheng Wu, Sushmitha P. Kini, Pete Wyckoff, Dhabaleswar K. Panda. Disponible sur http ://www.cse.ohio-state.edu/ liuj/pub/liu_ics03.pdf