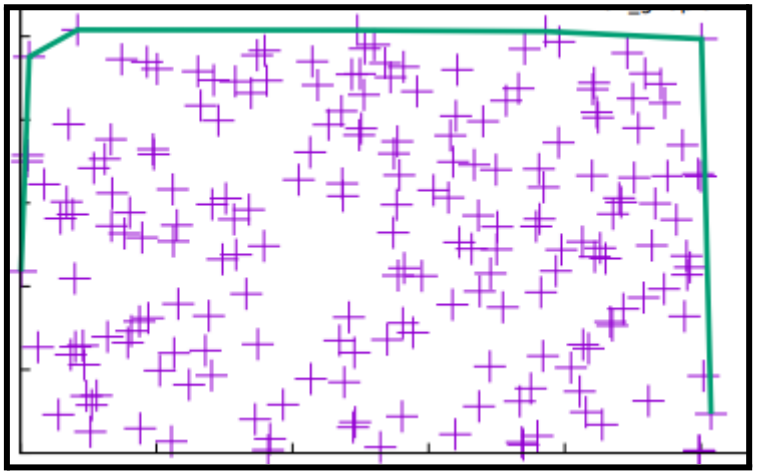
Rapport de projet Programmation Parallèle Haute Performance

Calcul en parallèle d'enveloppe convexe supérieure



Lucas Bernin - Maxime Yonnet - SIIA 1

I - Descriptif du problème **3**

II - Structure de donnée **4**

III - Explication **5**

IV - Résultat et état de l'avancement du projet **6**

V - Annexe **9**

I - Descriptif du problème

L'objectif du programme est de calculer l'enveloppe convexe supérieure d'un ensemble de points dans un plan à deux dimensions. La méthode utilisée pour ce calcul est une stratégie de "diviser pour régner".

Pour l'ensemble des points constituant le problème, on divise le problème en deux problèmes de même taille, avec une moitié des points dans chaque problème. On recommence cette division jusqu'à ce que tous les problèmes aient une taille inférieure ou égale à 4 points. Ainsi, dans chaque problème, il est possible de calculer une sous-enveloppe convexe supérieure. Lorsque toutes les enveloppes sont calculées, on fusionne les enveloppes séquentiellement. Le résultat obtenu est l'enveloppe convexe supérieure de l'ensemble des points.

Une résolution de ce problème en séquentielle est tout à fait possible, au prix d'une augmentation importante du temps de calcul, mais chaque point étant indépendant les uns des autres, il est possible de travailler sur plusieurs calculs en parallèle. Ainsi, on souhaite appliquer la parallélisation dans ce problème. Grâce à cela, on sera en mesure de gagner du temps de calcul.

Afin d'atteindre cet objectif, nous utiliserons PVM pour déléguer les tâches parallèles à des processus esclaves.

II - Structure de donnée

Nous avons décidé de créer des structures de données aptes à recevoir l'essentiel des informations afin de résoudre ce problème. Il nous a fallu d'abord une première structure de données capable de mémoriser les informations d'une liste de points, à savoir ses coordonnées X et Y et son successeur éventuel. Une seconde structure de données, regroupant la composition d'un problème, garde le type du problème, qu'il soit un problème de calcul d'enveloppe convexe ou de fusion d'enveloppe convexe, et les points concernés ainsi que leur taille.

L'utilisation de PVM implique que les envois et réceptions des données à paralléliser nécessite des types primitifs. Hors, dans le cas de ce problème, nous devons envoyer des structures aux esclaves, pour qu'ils puissent les calculer. Nous devons donc récupérer l'information primaire contenue dans ces structures pour les envoyer, et reconstituer les structures après réception.

Pour conserver la liste des structures des problèmes, nous avons créé une liste "pb_t" qui contient tous les problèmes à traiter. Si cette liste possède plus de 1 problème, alors c'est qu'il reste des problèmes à traiter. Si elle n'en possède qu'un, alors elle contient la solution au problème posé.

Nous avons aussi découpé en fonction les envois et réceptions entre maîtres et esclaves. Les fonctions sont "send_problem" et "receive_problem".

III - Explication

Au lancement du programme, on effectue une démarche de vérification du nombre de processus disponibles, et s' il y a besoin de processus supplémentaires pour régler le problème. Si le nombre de processus n'est pas suffisant, il les crée avant de continuer.

Une fois cela fait, on transforme la liste de point en un problème de calcul. On crée autant de processus esclave que définis dans le fichier de paramètre point.h. Tant qu'il y a plus d'un problème dans la liste et que ce n'est pas un problème de fusion, mais bien un problème de calcul, le processus maître s'occupe des envois et réceptions des problèmes, en deux parties bien distinctes.

Pour la première partie, pour chaque processus esclave, on sélectionne un problème, et si c'est un problème de calcul on l'envoie directement à l'esclave pour qu'il le traite. Si le problème pioché est de type fusion on pioche un deuxième problème. Si le deuxième est aussi de type fusion, on transforme les deux problèmes en un seul et on l'envoie en traitement à un esclave. Sinon, on envoie le second problème de type calcul et on remet le premier problème de type fusion pioché. Pendant l'envoie on peut envoyer autant de calcul que l'on veut mais on ne peut envoyer qu'une seule fusion.

Dans la partie du code effectuée par l'esclave, on fait une réception et une vérification du type de problème reçu du maître. Si ce problème est un problème de calcul, on effectue le calcul avec la fonction intégrée "point_UH" avec la liste de point envoyée. Si c'est un problème de fusion, on utilise la fonction intégrée "point_merge_UH" avec les deux listes de point envoyées. Dans les deux cas, l'esclave renvoie le résultat obtenu au maître.

Pour la partie réception, on récupère toutes les valeurs envoyées, et si c'est un calcul, on le transforme en fusion on le met dans la pile et si c'est une fusion on le met directement dans la pile. Comme dans le programme de base, on dessine les points en début de programme et les lignes en fin de programme.

IV - Résultat et état de l’avancement du projet

Actuellement le programme marche avec n’importe quel nombre d'esclaves et n’importe quel nombre de points, cependant seuls les calculs sont en en parallèle, les fusions sont faites sur un processus esclave mais une fusion à la fois.

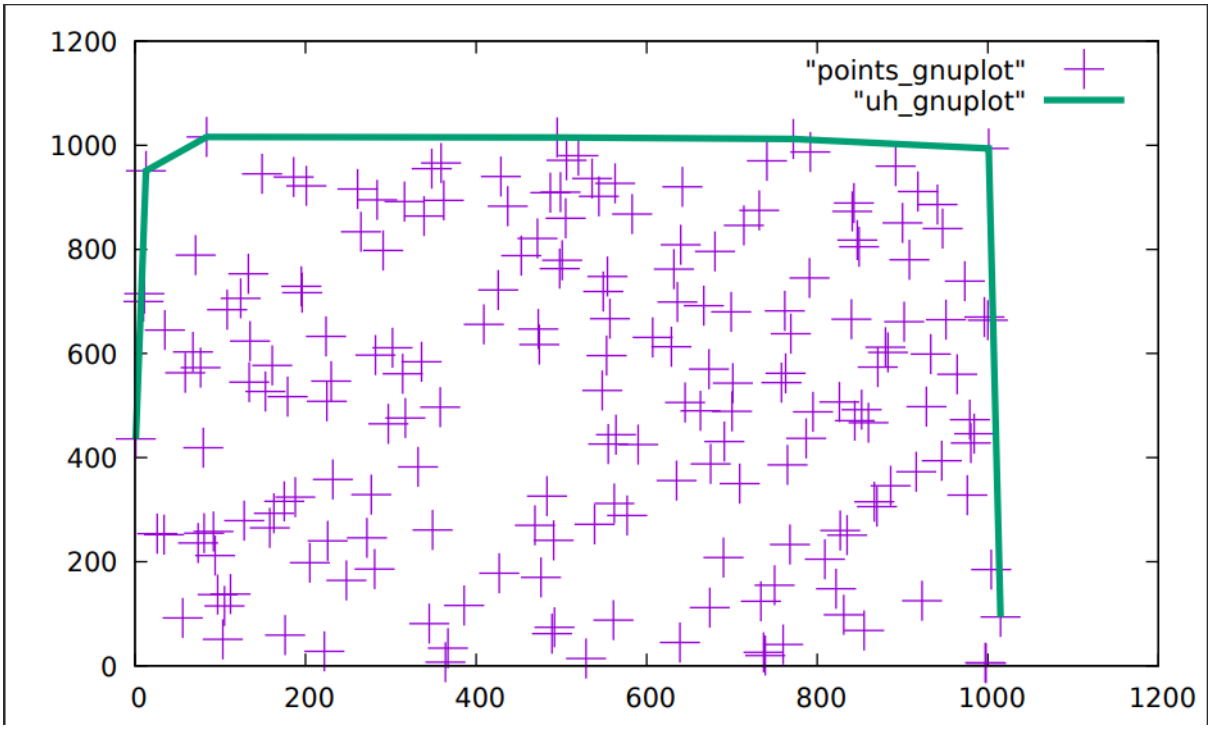
Si la fusion des résultats n’a pas pu être parallélisée aussi facilement que celle des calculs, c’est car la fusion fait intervenir un autre problème; la fusion nécessite d’être ordonnée pour rester correcte, sinon le résultat devient incohérent.

Pour pallier ce problème, une piste de solution aurait été de tracer chaque problème de fusion à l’aide d’un identifiant, et de comparer avec cet identifiant avant d’effectuer les fusions. Pour qu’une fusion soit valide, il faut que les identifiants de deux problèmes de fusion soient voisins direct. Ainsi, on accepte une fusion entre les problèmes 2 et 3, mais on décide d’ignorer une fusion entre les problèmes 2 et 4.

La fusion effectué, on a plus qu’à choisir l’identifiant le plus grand des deux fusions, et de continuer tant qu’il ne reste pas qu’un seul problème.

```
tp@tp:/media/sf_virtualbox/UH/Final_V1$ ./upper 15
----- N. Problème : 1
### Premier problème CALCUL N. 1, Nb problème : 0
### Création du problème N. 1
### Création du problème N. 2

MAITRE >>>>> Problème : 2 type CALCUL, Taille : 3 au thread : 0
### Premier problème CALCUL N. 2, Nb problème : 1

MAITRE >>>>> Problème : 1 type CALCUL, Taille : 4 au thread : 1
### Premier problème CALCUL N. 1, Nb problème : 0
### Création du problème N. 1

MAITRE >>>>> Problème : 1 type CALCUL, Taille : 4 au thread : 2
### Premier problème CALCUL N. 1, Nb problème : 0

MAITRE >>>>> Problème : 0 type CALCUL, Taille : 4 au thread : 3
--- Fin des envois, Pb restant : 0 et Pb envoyé : 4

MAITRE <<<<< Problème CALCUL avec 2 points au tour 0
### On réempile le problème N. 1,

MAITRE <<<<< Problème CALCUL avec 4 points au tour 1
### On réempile le problème N. 2,

MAITRE <<<<< Problème CALCUL avec 3 points au tour 2
### On réempile le problème N. 3,

MAITRE <<<<< Problème CALCUL avec 2 points au tour 3
### On réempile le problème N. 4,
--- Reception de problème terminée, il reste 4 problèmes
### On récupère le premier problème FUSION numéro 4, le nombre de probleme est égal à 3
### On récupère le deuxieme problème FUSION numéro 3, le nombre de probleme est égal à 2

MAITRE >>>>> Problème : 2 type FUSION, taille 2 et 3 au thread : 0
--- Fin des envois, Pb restant : 2 et Pb envoyé : 1

MAITRE <<<<< Problème FUSION avec 3 points au tour 0
### On réempile le problème N. 3,
--- Reception de problème terminée, il reste 3 problèmes
### On récupère le premier problème FUSION numéro 3, le nombre de probleme est égal à 2
### On récupère le deuxieme problème FUSION numéro 2, le nombre de probleme est égal à 1

MAITRE >>>>> Problème : 1 type FUSION, taille 3 et 4 au thread : 0
--- Fin des envois, Pb restant : 1 et Pb envoyé : 1

MAITRE <<<<< Problème FUSION avec 6 points au tour 0
### On réempile le problème N. 2,
--- Reception de problème terminée, il reste 2 problèmes
### On récupère le premier problème FUSION numéro 2, le nombre de probleme est égal à 1
### On récupère le deuxieme problème FUSION numéro 1, le nombre de probleme est égal à 0

MAITRE >>>>> Problème : 0 type FUSION, taille 6 et 2 au thread : 0
--- Fin des envois, Pb restant : 0 et Pb envoyé : 1

MAITRE <<<<< Problème FUSION avec 5 points au tour 0
### On réempile le problème N. 1,
--- Reception de problème terminée, il reste 1 problèmes
--- Un seul problème, fin des processus ---
newgraph
legend top
newcurve pts
74 236
84 1016
359 966
563 927
886 346
courbe dans upper_hull.pdf
evince upper_hull.pdf
tp@tp:/media/sf_virtualbox/UH/Final_V1$
```

Trace d'exécution pour un problème comportant 15 points dans le terminal



Graphe obtenu pour un problème de 201 points, la ligne verte est l'enveloppe convexe des points

V - Annexe

Les annexes déroulent les parties du code qui ont été développées lors de ce projet, le reste des fonctions ne sont pas modifiées de façon significative, et sont accessibles dans les sources du projet.

```c
#include "stdio.h"
#include "stdlib.h"
#include "point.h"

#include "pvm3.h"

static pb_t * Q;        /* la pile des problemes */
static int Q_nb = 1;

void init_queue(data, N)
point * data; int N;
{
    point * pts = data;
    Q = (pb_t *)malloc(N * sizeof(pb_t));
    Q[1].type = PB_CALC;
    Q[1].data1 = *pts;
}

void divide_problem(pb_t pb){
    while(point_nb(&pb.data1) > 4){
        printf("### Création du problème N. %d\n",Q_nb+1);
        Q_nb++;
        Q[Q_nb].data1 = *point_part(&pb.data1);
        Q[Q_nb].type = PB_CALC;
    }
}
```

Fichier upper.c, include, fonctions d'initialisation de la pile et de division des problèmes

```c
void send_problem(pb_t * pb, int destinataire, int msgtag){
    pb->taille1 = point_nb(&pb->data1);
    //Si problème de type fusion et que le second tableau est non null,
    // on définit la taille du second tableau.
    if(pb->type == PB_FUS && &pb->data2 != NULL) pb->taille2 = point_nb(&pb->data2);
    else pb->taille2 = 0;
    pvm_initsend(PvmDataDefault);
    // Récupère d'autres infos
    pvm_pkint(&pb->type,1,1);
    pvm_pkint(&pb->taille1,1,1);
    if(pb->taille1 > 0){
        point ptsTemps = pb->data1;
        // Pour tout les points du premier tableau
        for(int i = 0; i < pb->taille1; i++){
            //ajoutes les coordonnées
            pvm_pkint(&ptsTemps.x,1,1);
            pvm_pkint(&ptsTemps.y,1,1);
            //Si ce n'est pas le dernier point, on passe au point suivant
            if(i+1 < pb->taille1){
                ptsTemps = *ptsTemps.next;
            }
        }
    }
    //Si le type de problème à envoyer est fusion
    if(pb->type == PB_FUS){
        //Prépare la taille du deuxieme tableau à l'envoie
        pvm_pkint(&pb->taille2,1,1);
        //Si la taille est positive
        if(pb->taille2 > 0){
            point ptsTemps = pb->data2;
            //Pour tous les points du tableau
            for(int i = 0; i < pb->taille2; i++){
                //Prépare la coordonnée x à l'envoi
                pvm_pkint(&ptsTemps.x,1,1);
                //Prépare la coordonnée y à l'envoi
                pvm_pkint(&ptsTemps.y,1,1);
                //Si ce n'est pas le dernier point on passe au point suivant
                if(i+1 < pb->taille2){
                    ptsTemps = *ptsTemps.next;
                }
            }
        }
    }
    //Effectue l'envoi
    pvm_send(destinataire, msgtag);

}
```

Fichier upper.c, fonction d'envoi d'un problème

```c
pb_t * receive_problem(int sender){
    int msgtag;
    int tid;
    int bufid;
    //Effectue la reception
    bufid = pvm_recv(sender,-1);
    //Récupère les informations sur l'envoi
    pvm_bufinfo(bufid,NULL, &msgtag,&tid);
    //Si le tag est MSG_END on returne NULL
    if(msgtag == MSG_END) return NULL;
    //On initialise le problème
    pb_t * pb = (pb_t *) malloc(sizeof(pb_t));
    pb->taille1 = 0;
    pb->taille2 = 0;
    //On récupère les infos du problème
    pvm_upkint(&pb->type,1,1);
    pvm_upkint(&pb->taille1,1,1);
    if(pb->taille1 > 0){
        //On récupère le x et y
        pvm_upkint(&pb->data1.x,1,1);
        pvm_upkint(&pb->data1.y,1,1);
        point * ptsTemps = point_alloc();
        if(pb->taille1 > 1 ) pb->data1.next = ptsTemps;
        else pb->data1.next = NULL;
        for(int i = 0; i < pb->taille1-1; i++){
            //On récupère le x et y
            pvm_upkint(&ptsTemps->x,1,1);
            pvm_upkint(&ptsTemps->y,1,1);
            //Si il y a un point suivant
            if(i+1 < pb->taille1-1){
                ptsTemps->next = point_alloc();
                ptsTemps = ptsTemps->next;
            }else{
                ptsTemps->next = NULL;
            }
        }
    }
    //Si le problème est une fusion
    if(pb->type == PB_FUS){
        //Récupère la taille du deuxieme tableau
        pvm_upkint(&pb->taille2,1,1);
        if(pb->taille2 > 0){
            pvm_upkint(&pb->data2.x,1,1);
            pvm_upkint(&pb->data2.y,1,1);
            point * pts2Temps = point_alloc();
            if(pb->taille2 > 1 ) pb->data2.next = pts2Temps;
            else pb->data2.next = NULL;

            for(int i = 0; i < pb->taille2-1; i++){
                pvm_upkint(&pts2Temps->x,1,1);
                pvm_upkint(&pts2Temps->y,1,1);
                if(i+1 < pb->taille2-1){
                    pts2Temps->next = point_alloc();
                    pts2Temps = pts2Temps->next;
                }else{
                    pts2Temps->next = NULL;
                }
            }
        }
    }
    return pb;
}
```

Fichier upper.c, fonction de réception d'un problème

```c
int main(int argc, char **argv)
{

int mytid;      /* tid tache */
int parent;     /* tid du pere */
int tids[NB_SLAVE];     /* tids fils */
int nb_paquet;
point *pts;
int numeroSlave = 0;

parent = pvm_parent();



if(parent == PvmNoParent){  //MASTER
    if (argc != 2) {
        fprintf(stderr, "usage: %s <nb points>\n", *argv);
        exit(-1);
    }
    pts = point_random(atoi(argv[1]));
    point_print_gnuplot(pts, 0); /* affiche l'ensemble des points */
    init_queue(pts,atoi(argv[1]));
    //Si il y a besoin d'esclaves
    if(atoi(argv[1]) >= 1){
        pvm_spawn(EPATH "/upper", (char**)0,0, "", NB_SLAVE, tids);

        printf("----- N. Problème : %d \n",Q_nb);
        do{
            /*-------------PARTIE ENVOI -------------*/
            int nbEnvoie = 0;
            //Pour tout les threads
            for(int numeroThread = 0; numeroThread < NB_SLAVE; numeroThread++){
                //Si il n'y a plus de problème à piocher, on passe à la réception
                if(Q_nb <1){
                    break;
                }else if(Q_nb == 1 && Q[Q_nb].type == PB_FUS){
                    break;
                }
                nbEnvoie++;
                //On récupère un premier problème
                pb_t pb1 = Q[Q_nb];
                Q_nb--;
```

Fichier upper.c, fonction main première partie

```
//Si le premier problème récupéré est un problème de calcul
if(pb1.type == PB_CALC){
    printf("### Premier problème CALCUL N. %d, Nb problème : %d\n",Q_nb+1,Q_nb);
    //On divise les problèmes jusqu'a que le problème actuel
    //ai un nombre de point inferieur à 5
    divide_problem(pb1);
    printf("\nMAITRE >>>>> Problème : %d type CALCUL,
        Taille : %d au thread : %d \n",Q_nb,point_nb(&pb1.data1), numeroThread);
    send_problem(&pb1,tids[numeroThread],MSG_PB);
//Si le premier problème récupéré est un problème de fusion
}else{
    printf("### On récupère le premier problème FUSION numéro %d, le nombre de probleme est égal à %d\n",Q_nb+1,Q_nb);


    //On récupère un deuxième problème (pour esperer avoir une autre fusion et pouvoir fusionner les deux problèmes)
    pb_t pb2 = Q[Q_nb];
    Q_nb--;
    //Si le deuxieme problème n'est pas un problème de fusion
    if(pb2.type == PB_CALC){
        printf("### On récupère le deuxieme problème CALCUL numéro %d,
         le nombre de probleme est égal à %d\n",Q_nb+1,Q_nb);
        //On divise les problèmes jusqu'a que le problème actuel ai un nombre de point inferieur à 5
        divide_problem(pb2);
        printf("\nMAITRE >>>>> Problème : %d type CALCUL,
         taille : %d au thread : %d \n",Q_nb,point_nb(&pb2.data1), numeroThread);
        send_problem(&pb2,tids[numeroThread],MSG_PB);
        Q_nb++;
        // On remet le premier problème dans la pile
        Q[Q_nb] = pb1;
        printf("### On réempile le problème N. %d\n",Q_nb);

    //Si le deuxieme problème reçu est un problème de fusion
    }else{
        printf("### On récupère le deuxieme problème FUSION numéro %d,
         le nombre de probleme est égal à %d\n",Q_nb+1,Q_nb);
        //On transfère les points du second problème dans le premier et l'envoie à l'esclave
        pb1.data2 = pb2.data1;
        printf("\nMAITRE >>>>> Problème : %d type FUSION,
            taille %d et %d au thread : %d \n",Q_nb,point_nb(&pb1.data1), point_nb(&pb1.data2), numeroThread);

        send_problem(&pb1, tids[numeroThread],MSG_PB);
        break; // Si on a pioché une fusion on envoie plus rien pour pouvoir faire
        // les fusions dans l'ordre voir commentaire #1 en bas
    }
}
```

Fichier upper.c fonction main deuxième partie

```
            /*-------------PARTIE RECEPTION -------------*/
            printf("--- Fin des envois, Pb restant : %d et Pb envoyé : %d\n", Q_nb,nbEnvoie );
            for(int numeroThread = 0; numeroThread < nbEnvoie ; numeroThread++){    //Pour tous les envois effectués
                pb_t * pbTest = receive_problem(tids[numeroThread]);
                pb_t pb = *pbTest;
                Q_nb++;
                //Si le problème récupéré est une fusion on l'ajoute à la pile
                if(pb.type == PB_FUS){
                    printf("\nMAITRE <<<<< Problème FUSION avec %d points au tour %d \n", point_nb(&pb.data1),numeroThread);
                //Si le problème récupéré est un calcul alors on le transforme en fusion et on l'ajoute dans la pile
                }else{
                    printf("\nMAITRE <<<<< Problème CALCUL avec %d points au tour %d \n", point_nb(&pb.data1),numeroThread);
                    pb.type = PB_FUS;
                }
                printf("### On réempile le problème N. %d,\n",Q_nb);
                Q[Q_nb] = pb;
            }
            printf("--- Reception de problème terminée, il reste %d problèmes\n", Q_nb);
        }while( Q_nb > 1);  //On fait ça jusqu'a qu'il n'y ai plus qu'un problème
        printf("--- Un seul problème, fin des processus ---\n");
            //Pour tous les esclaves, on leur envoie un message de fin
        for(int i = 0; i < NB_SLAVE; i++){
            pb_t pb;
            pb.data1 = *point_alloc();
            pb.type = PB_CALC;
            send_problem(&pb, tids[i], MSG_END);
        }
        point_print(&Q[1].data1);
        point_print_gnuplot(&Q[1].data1, 1);
        point_free(pts);
    }
}else{  //SLAVE
    while(1){
        pb_t * pbTest = receive_problem(parent);
        if(!pbTest) break;  //Si le problème reçu est un message de fin
        pb_t pb = *pbTest;
        if(pb.type == PB_CALC){ //Si c'est un problème de type calcul
            point_UH(&pb.data1);
    }else{
        point * pts1 = point_alloc();
        *pts1 = pbTest->data1;
        pb.data1 = *point_merge_UH(&pbTest->data2, pts1);
    }
    send_problem(&pb,parent,MSG_PB);
    free(pbTest);
}
}
```

Fichier upper.c, fonction main dernière partie

```c
#define XMAX (1024)
#define YMAX (1024)

#define TAILLEMAXPAQUET 4

#define NB_SLAVE 4
#define NAME_SLAVE "upper"

#define MSG_PB 0
#define MSG_END 1

#define PB_CALC 0
#define PB_FUS 1


/*
 * structure associee a chaque point
 */
typedef struct st_point point;
struct st_point {
    int x, y;
    point *next;    /* liste chainee des points de l'enveloppe */
};

struct st_pb {
    int type;
    point data1;
    int taille1;
    point data2;
    int taille2;
};

typedef struct st_pb pb_t;

/*
 * dans point.c
 * utilitaire de calcul pour le TAD point
 */
extern point *point_alloc();
extern void point_free();
extern void point_print();
extern point *point_random();
extern point *point_UH();
extern int point_nb();
extern point *point_part();
extern point *point_merge_UH();
extern void point_print_gnuplot();
extern void upper_hull();

extern void send_pb();
extern pb_t *receive_pb();
```

Fichier point.h, structures de données