Description du code

Le démonstrateur est développé en C++. Il utilise exclusivement des fonctions de la librairie standard, et à ma connaissance aucune fonction spécifique au C++11. Aucun test unitaire n’a été développé ; il n’y a pas non plus d’exception (seul le flux de console d’erreur cerr est utilisé à cette fin). Toutes les fonctions « importantes » et qui sont utilisées dans la version finale sont documentées *a minima* dans les headers (principe de la fonction, entrées/modifications des objets/sortie).

Le démonstrateur a été développé sur Windows pour une utilisation sur Windows ; pour un autre système il faudrait prêter attention au format des fichiers de données (en particulier le caractère de fin de ligne) et à la création du dossier de sortie qui est fait par la librairie windows.h (il n’y a pas de gestion de l’OS).

Le développement de la première version a été fait en utilisant MinGW et son compilateur g++ (un makefile accompagne le code). Le développement de la deuxième version a été fait en utilisant Code ::Blocks, par défaut avec MinGW et son compilateur g++ .

# Liste des fichiers

Le code est composé de 27 fichiers, dont :

* 7 classes :
  + Calc : représente un calculateur ou une passerelle pure
  + SubNet : représente un sous-réseau (un bus)
  + Element : représente un élément physique du réseau (calculateur ou sous-réseau). Calc et SubNet en héritent.
  + Frame : représente une trame
  + Parameter : représente un paramètre
  + PathFrame : représente un chemin de trame (il y a un chemin par couple (trame ; 1 cible de la trame)
  + Criteria : représente le critère retenu pour l’optimisation (en particulier les opérations < et > sont définies sur cet objet)
* 6 fichiers (auxquels s’ajoutent leur headers) de fonctions :
  + IOManagement : fonctions de lecture des fichiers input et d’écriture des fichiers de sortie
  + Constraint : regroupe les fonctions destinées à vérifier les contraintes. Attention : toutes les contraintes n’y sont pas évaluées (par ex. la contrainte de charge maximale n’est pas évaluée ici).
  + Architecture : regroupe des opérations faites sur le réseau (copies, génération de topologie, lancement du routage…)
  + Solver : fonctions qui implémentent les heuristiques/méta-heuristiques d’optimisation
  + main : le main, qui permet essentiellement de gérer les 2 modes de résolution et lancer les fonctions adéquates
  + AddCalc : fonctions qui implémentent le mode 3 et 4 de résolution
* 2 headers globaux :
  + include.h : les include de la librairie standard que la majorité des classes/fonctions utilisent
  + const\_tool.h : définit les variables globales

Le code a été développé avec l’IDE CodeBlocks et est accompagné d’un exécutable sur le dossier bin\Release.

# Description des objets

### Element

Un élément est représenté par un identifiant (chaîne de caractère) qui doit être unique ; sa description est complétée par un booléen indiquant s’il s’agit d’un sous-réseau ou d’un calculateur.

### Calc

Un calculateur est un élément caractérisé par des paramètres (qu’il émet ou reçoit ; la liste des paramètres émis n’est pas stockée), qui doivent être reçu à des fréquences données, des connexions à des sous-réseaux CAN et des connexions à des sous-réseaux CAN-FD.

### SubNet

Un calculateur est un élément caractérisé par des connexions avec des calculateurs et une charge réseau (débit d’information qui circule dessus). Il est soit CAN, soit CAN-FD.

### Frame

Un trame est caractérisée par un identifiant (chaîne de caractère) unique, un calculateur émetteur (source), une fréquence d’émission, des calculateurs récepteurs (cibles), un contenu (paramètres, leur taille, la fréquence à laquelle ils doivent être reçu par les cibles), une charge CAN (calculée à partir de la fréquence d’émission et de la taille de la trame),une charge CAN-FD, les sous-réseaux qu’elle emprunte (routage), une latence (calculée à partir de la taille de la trame et des sous-réseaux qu’elle emprunte), si son contenu est fixé et si son routage est fixé.

##### void Frame::updateLoad()

Récupère la taille du champ de données à partir du placement des paramètres dans les octets (on ne peut envoyer qu’un nombre entier d’octets) et l’utilise pour calculer la taille totale de la trame puis sa charge CAN et sa charge CAN-FD.

##### bool Frame::arrangeParamPlacement()

##### bool Frame::arrangeParamPlacement(vector< vector<Parameter\*> >& parameterArrangement)

Décide d’un placement des paramètres dans les octets du champ de données.

Si au moins un paramètre a une taille strictement supérieure à 8 bits ou bien si l’on autorise les paramètres à être à cheval sur plusieurs octets, les paramètres sont mis bout-à-bout quitte à les placer à cheval sur plusieurs octets. Le placement est indiqué dans l’argument parameterArrangement où seul le premier élément est utilisé.

Le cas contraire et si on a spécifié qu’on voulait empêcher les paramètres d’être à cheval sur 2 octets, une heuristique non-exacte les place dans l’argument parameterArrangement où chaque élément est un vector<Parameter\*> représentant un octet.

Dans tous les cas, un booléen indiquant si un placement correct (c’est-à-dire, qui respecte le nombre maximum d’octets disponibles) a pu être trouvé est renvoyé. L’utilisation de l’heuristique approchée n’assure pas de trouver une solution, même s’il en existe.

La version sans argument fait appel à la version avec.

##### bool Frame::addParameter(Parameter\* newpar, bool force=false)

Essaie d’ajouter le paramètre newpar au contenu de la trame (s’il loge). Met à jour le paramètre et la trame (contenu, charges et fréquence d’émission) en conséquence.

##### bool Frame::remParameter(Parameter\* par)

Essaie de retirer le paramètre newpar au contenu de la trame (s’il y est déjà). Met à jour le paramètre et la trame (contenu, charges et fréquence d’émission) en conséquence.

##### bool Frame::spreadFrame(vector<SubNet> const\* snList, vector< vector<bool> > const& curNetworkMatrix)

##### bool Frame::spreadFrame(vector<SubNet> const\* snList, vector< vector<int> > const& curNetworkMatrix)

Effectue le routage de la trame de manière à minimiser la charge du sous-réseau le plus chargé. Si un routage existe pour relier la source de la trame à toutes les cibles, il sera trouvé et la fonction renverra le booléen true, la latence (qui dépend des réseaux traversés) sera mise à jour. Le cas contraire, le booléen false sera renvoyé et le champ Frame ::snFilled ne sera pas modifié.

Si le routage est fixé, la fonction renvoie true sans faire les calculs et sans modifier Frame ::snFilled.

### Parameter

Un paramètre est caractérisé par un identifiant (chaîne de caractères) unique, un calculateur émetteur (source), des calculateurs cibles et les fréquences de réception associées, les trames dans lesquels le paramètre est encapsulé et si on peut changer ces trames hôtes.

### PathFrame

Un chemin de trame est caractérisé par la trame à laquelle il se réfère, le calculateur cible de la trame qui est concerné par ce chemin, la longueur (nombre d’éléments traversés, y compris la source et la cible) et la liste des éléments traversés.

##### bool PathFrame::findShortestPath(vector<Element\*>\* elems\_) const

Trouve le chemin reliant la source et la cible du chemin respectant la topologie du réseau (les connexions calculateur – sous-réseau) et qui minimise le nombre d’éléments traversés en utilisant une recherche en largeur d’abord dans le graphe de connexions. S’il existe un tel chemin, il sera trouvé.

##### bool PathFrame::generatePath()

Essaie de génère un chemin reliant la source et la cible du chemin (en utilisant l’algorithme de plus court chemin), renvoie si il a réussi et met à jour les champs le cas échéant.

### Criteria

Définit le critère à minimiser par les méta-heuristiques d’optimisation de l’architecture. Le critère implémenté est composé de :

|  |  |
| --- | --- |
| Coût | nombre de connecteurs utilisés \* prix d'un connecteur  + nombre de bus utilisés \* prix d'un bus  + nombre de passerelles pures utilisées \* prix d'une passerelle pure |
| Charge max | Charge du bus le plus chargé |

L’important est de définir une relation d’ordre qui permet de comparer tout couple d’architecture ; c’est pourquoi la relation < est définie entre deux Criteria-s.

Il existe trois critères de choix entre deux architectures, on choisit toujours le cout minimum en 1er, puis à coût égal :

* l’architecture qui minimise le réseau le plus chargé
* l’architecture qui minimise la latence total
* l’architecture qui minimise le réseau le plus chargé et à charge égal, choisi l’architecture qui minimise la latence

# Description des principales fonctions

### Main2.cpp

##### main()

Lit le fichier de configuration, gère le type de résolution demandée et redirige la sortie standard vers un fichier texte si nécessaire.

##### optimTopologyWrapper(char\* date)

Lance la lecture des fichiers d’entrée pour le mode de résolution 1 (trames fixées), l’optimisation de la topologie à trames fixées, la génération automatique de topologies initiales et leur optimisation en cas de multistart, l’écriture des fichiers de sorties.

##### optimTopologyLauncher(…)

Lance l’optimisation de la topologie fournie en paramètre.

##### optimMessageSetWrapper(char\* date)

Lance la lecture des fichiers d’entrée pour le mode de résolution 2 (messagerie + topologie), l’optimisation conjointe de la topologie et de la mise en trames, l’écriture des fichiers de sorties.

##### optimMessageSetLauncher(…)

Lance l’optimisation de l’architecture (topologie+messagerie) passée en paramètres.

### Architecture.cpp

##### initFrameFromBFDHeuristic(vector<Parameter>\* par, vector<Calc> const \* const calcList, vector<Frame>\* fr, vector<Frame> const \* const frIni )

A partir des données (par et frIni, respectivement la liste des paramètres et celle des trames lues dans les données) et de la liste des calculateur, crée une messagerie en utilisant une heuristique inspirée du *Best Fit Decreasing*. La messagerie créée est stockée dans fr, la liste des paramètres est mise à jour de manière à pointer vers ces nouvelles trames.

##### initPath(…)

Réinitialise les bus utilisés par les trames afin d’initialiser des chemins entre les sources et les cibles de toutes les trames ; retourne le nombre de chemin non trouvés.

##### drawPath(…)

Lance la génération d’un chemin pour chaque couple (trame ; cible de la trame) et retourne le nombre de chemin non trouvés.

##### spreadPathperFrame()

Lance le routage complet : faire un certain nombre de fois le routage de chacune des trames (triées par charge décroissante).

##### copyNetwork(vector<Calc> const \* const calcData, vector<SubNet> const \* const subNetData, vector<Frame> const \* const frameData, vector<Calc>\* calcCopy, vector<SubNet>\* subNetCopy, vector<Frame>\* frameCopy)

Copie les calculateurs, sous-réseaux et trames de manière à ce que les objets “copies” aient les mêmes liens entre eux que les objets « originaux » avaient. Noter que les objets originaux comme les objets copies pointent vers les mêmes objets paramètres originaux.

##### copyData(vector<Calc> const \* const calcData, vector<Frame> const \* const frameData, vector<Calc>\* calcCopy, vector<Frame>\* frameCopy)

Copie les calculateurs et les trames, tout en gardant les liens entre eux. Noter que les objets originaux comme les objets copies pointent vers les mêmes objets paramètres originaux. S’il y avait des connexions des calculateurs originaux à des sous-réseaux, celles-ci ne sont pas conservées pour les calculateurs copies.

##### copyNetworkFull(vector<Calc> const \* const calcData, vector<SubNet> const \* const subNetData, vector<Frame> const \* const frameData, vector<Parameter> const \* const paramData, vector<Calc>\* calcCopy, vector<SubNet>\* subNetCopy, vector<Frame>\* frameCopy, vector<Parameter>\* paramCopy)

Copie l’ensemble des objets (calculateurs, sous-réseaux, paramètres et trames) de manière à ce que les objets “copies” aient les mêmes liens entre eux que les objets « originaux » avaient.

##### adaptParamToCopiedNetwork(vector<Parameter> \* const par, vector<Calc> const \* const calcCopy, vector<Frame> const \* const frameCopy)

Utilise les identifiants des sources/cibles/trames hôtes des paramètres pour les faire pointer vers les objets contenus dans calcCopy et frameCopy. Noter que les objets calcCopy et frameCopy ne sont pas modifiés et doivent déjà pointer vers les opbjets contenus dans par.

##### constructValidNetwork(…)

Tente de construire une topologie respectant les contraintes de topologie et utilisant un maximum de connexions.

##### computeNetworkConMatrix(…)

Représente une topologie par une matrice de connexions.

##### clearNetwork(…)

Supprime les sous-réseaux vides se trouvant à la fin de la liste des sous-réseaux.

##### updateLoads (…)

Met à jour les charges sur les sous-réseaux en y faisant passer les trames par les chemins donnés en paramètre.

##### updateLoadsFromFrame(…)

Met à jour les charges sur les sous-réseaux en utilisant les champs Frame::snFilled qui représentent les sous-réseaux utilisés par chaque trame.

##### evaluateCost(…)

Évalue le coût d’une topologie à partir du nombre de connecteurs, du nombre de passerelles pures et du nombre de sous-réseaux utilisés ainsi que de leur coût unitaire.

##### evaluateArchi(…)

Évalue le respect des contraintes par l’architecture passée en paramètre, lance son routage et renvoie le critère associé à l’architecture.

### IOManagement.cpp

##### cfgFromFile()

Lit le ficher de configuration et enregistre les paramètres globaux.

##### importFcioFileData(…)

Importe un fichier de messagerie-trames.

##### importPcioFileData(…)

Importe un fichier de messagerie-paramètres.

##### importInitSubNet(…)

Importe un fichier de topologie initiale.

##### importUserDefinedTopologyConstraints(…)

Importe un fichier de contraintes de topologies spécifiques fourni par l’utilisateur.

##### outputMessagingFile(…)

Écrit un fichier de messagerie-paramètres.

##### outputFrameSynthesisFile(…)

Écrit un fichier de messagerie-trames.

##### outputTopologyFile(…)

Écrit un fichier de topologie.

### Constraint.cpp

##### checkTopologyValidity(…)

Évalue le respect des contraintes concernant la topologie : nombre de calculateurs par bus, nombre de connecteurs utilizes pour chaque calculateur, nombre de passerelles pures utilisées, contraintes dites « additionnelles » qui sont définis dans le fichier fourni par l’utilisateur.

##### checkAdditionnalTopologyConstraint(…)

Vérifie les contraintes « additionnelles » qui sont définis dans le fichier fourni par l’utilisateur.

##### checkRoutingValidity(…)

Évalue le respect des contraintes de routage (sauf respect de la charge maximale) : existence de chemins afin de livrer tous les paramètres.

### Solver.cpp

##### optimTabu(…)

Optimisation par algorithme tabou de la topologie passé en paramètre. L’algorithme tabou fait évoluer la topologie courante en lui ajoutant ou lui retirant une connexion à chaque itération (la totalité de ce voisinage est visitée avant de choisir quelle connexion modifier). La topologie à l’itération suivante doit vérifier toutes les contraintes, exceptée la contrainte de charge maximale.

##### updateBestTopology(…)

Met à jour la file des meilleures topologies.

##### metaTabu(…)

Optimisation par métaheuristique tabou de l’architecture passée en paramètre. L’algorithme tabou fait évoluer la topologie courante en lui ajoutant ou lui retirant une connexion à chaque itération (la totalité de ce voisinage est visitée avant de choisir quelle connexion modifier, et à chaque voisin une optimisation de la mise en trames a lieu à partir d’une même messagerie initiale). La topologie à l’itération suivante doit vérifier toutes les contraintes, excepté la contrainte de charge maximale.

##### updateBestArchi(…)

Le pendant de updateBestTopology(…), mais pour des couples (topologie ; messagerie).

##### optimLoadsSimulatedAnnealing2(…)

Optimise la messagerie initiale passée en paramètre en utilisant une heuristique de recuit simulé qui permet à chaque itération de changer un paramètre de trame hôte, tout en conservant une mise en trames valide. Un routage partiel (uniquement sur l’ancienne trame hôte du paramètre et la nouvelle) est lancé à chaque itération, ainsi qu’un routage complet à la fin du recuit.

### AddCalc.cpp

##### optimNewCalc (…)

Lance la lecture des fichiers d’entrée pour le mode de résolution 3 (ajouter nouveaux calculateurs à topologie), l’optimisation de la topologie, l’écriture des fichiers de sorties.

##### addCalcWithMessage(…)

Lance la lecture des fichiers d’entrée pour le mode de résolution 4 (messagerie + ajouter nouveaux calculateurs à topologie), l’optimisation conjointe de la topologie et de la mise en trames, l’écriture des fichiers de sorties.