Mise à jour des tableaux dans Trio_U v1.4.9
ArrOfInt / Double
Int / DoubleVect
Int / DoubleTab

Benoît Mathieu, juin 2005

Gestion de configuration

Les fichiers ArrOfInt et ArrOfDouble sont maintenant générés à partir du script check.sh. A la compilation de Trio_U, chaque répertoire peut contenir un tel script qui est exécuté avant la compilation des fichiers du répertoire. Dans le répertoire Math, le script check.sh reconstruit si nécessaire les fichiers ArrOfDouble.h, ArrOfDouble.cpp, ArrOfInt.cpp, ArrOfInt.h à partir du squelette ArrOf_Scalar_Prototype.cpp.h.

Modifications des spécifications

Etat du tableau:

Un tableau peut être dans l'un des trois états suivants:

- détaché (taille nulle, aucune zone de mémoire associée)
- normal (la mémoire est gérée par le tableau, elle est libérée à la fin)
- ref_data (la mémoire est n'est pas gérée par le tableau, resize est interdit)

Un cas particulier de tableau normal est « ref_array ». Dans ce cas, plusieurs objets Array partagent la même zone de mémoire. Dans ce cas, il est interdit de faire un « resize » du tableau.

Voir les fichiers .h et .cpp pour les spécifications détaillées.

ATTENTION: le comportement des méthodes qui agissent sur la **structure** du tableau dépend de l'état du tableau. Exemple: pour un tableau ref_data, il est interdit de faire un resize car le tableau n'est pas propriétaire du bloc de mémoire.. Donc il est interdit de faire x=y si x est un tableau ref_array et y est un tableau de taille différente car cela obligerait à changer la taille de x.

Politique de réallocation lors des resize:

Le tableau peut avoir deux modes de « resize » possibles (méthode set_smart_resize())

- normal: le bloc de mémoire alloué a toujours la même taille que le tableau. Tout changement de taille conduit à une réallocation et à une copie des données existantes.
- smart_resize: un bloc de mémoire n'est réalloué que si le bloc précédent n'est pas assez grand. Les blocs sont alloués en doublant au moins la taille du bloc précédent. Pour libérer la mémoire, on peut faire « reset() », ce qui ramène la taille du tableau à zéro.

Valeur par défaut du tableau:

Précédemment, un tableau construit par la ligne « ArrOfDouble t(n) » était initialisé avec

une valeur nulle. De plus, lors d'une augmentation de la taille du tableau par « resize », les nouvelles cases étaient remplies avec la valeur zéro.

Cette spécification a été conservée pour les tableaux **qui ne sont pas smart_resize.** Les tableaux smart resize n'ont aucune valeur par défaut (en mode débug, ils sont initialisés avec une valeur invalide).

Sélection du pool de mémoire utilisé:

Les fonctionnalités de la classe DoubleTrav ont été déplacée dans ArrOf__. Les tableaux « normaux » (par opposition aux tableaux ref_data) ont deux modes d'allocation accessibles par la méthode set_mem_storage :

- allocation sur le « heap » standard avec new
- allocation dans un pool de mémoire géré par l'objet Memoire (add_trav_int ou add_trav_double). Le gain apporté par ce type d'allocation est vraisemblablement négligeable.

Modifications apportées à l'interface des ArrOf_

Méthodes supprimées

Tous les constructeurs qui n'utilisent pas « entier »

Il ne peut y avoir un problème que dans le cas où « entier » est défini comme un « short » sur une machine 64 bits. Dans ce cas, il faut faire très attention aux warnings: la conversion d'un « int » en « entier » peut être fatale si le nombre est supérieur à 2^31. Il est fortement recommandé de faire :

```
int i;
entier i_entier = (entier) i; // conversion de type.
assert(i_entier == j); // Verification qu'on n'a pas perdu d'info.
```

Constructeur par pointeur

```
ArrayOf__::ArrOfDouble(const double* ptr, entier n, double x=0);
```

Désormais, il faut construire un tableau vide, puis associer l'adresse comme ceci:

```
ArrOfDouble a;
a.ref_data(ptr, size);
```

*L'operateur de conversion implicite en (*double*)*

Considérée comme dangereuse car elle permet de contourner silencieusement l'attribut const. Les operateurs de conversion implicite ne sont pas les bienvenus en général.

Il faut maintenant utiliser t.addr() pour obtenir l'adresse du bloc de mémoire.

ArrayOf__::Maxarray() et ArrayOf__::minarray(), ArrayOf__::max_abs_array, ArrayOf__::min_abs_array

Retirés car doublons avec friend ____ maxarray(const ArrOf____);

ArrayOf__::Carre_array, ArrayOf__::ajoute_array, ArrayOf__::ajoute_carre_array, ArrayOf__::racine_carre_array

Retirés car très peu utilisés

ArrayOf__::insert_array(), ArrayOf__::est_egal_a_array()

Retiré car très peu utilisé

imin, imax, norme

Renomés imin_array imax_array norme_array pour mettre en évidence qu'il s'agit d'opérations sur des vecteurs **non distribués.**

Ordonne

Supprimé (doublon avec ArrayOf___::ordonne_array()

carre, racine_carree

Retiré car pas utilisé (carre peut etre obtenu avec tab *= tab)

ArrOfDouble operator+(const ArrOfDouble&, const double)
ArrOfDouble operator-(const ArrOfDouble&, const double)
ArrOfDouble operator(const ArrOfDouble&, const double)*
ArrOfDouble operator/(const ArrOfDouble&, const double)

Retiré car mauvaises performances (génère deux copies des données)

```
ArrayOfDouble x,y,z;
//  x = y + z; Cette syntaxe ne fonctionne plus, utiliser ceci:
x = y;
x += z;
```

double operator(const ArrOfDouble&, const ArrOfDouble&)*

Attention, operator*(const DoubleVect &, const DoubleVect &) avait un comportement non trivial pour les vecteurs distribués. La fonction qui le remplace est « mp_prodscal_local »

Remplacé par dotproduct_array(const ArrOfDouble&, const ArrOfDouble&) (à cause de l'ambiguité de comportement d'operator* si l'un des tableaux est un tableau distribué).

Dotproduct_array calcule un produit scalaire non distribué.

mp_prodscal calcul un produit scalaire distribué

norme(const ArrOfDouble&)

Remplacé par norme_array() (idem: ambiguité entre opération sur un vecteur distribué ou non)

Equivalent pour un vecteur distribué: mp_norme_vect(...)

Fonctionnalités ajoutée

Smart_resize + append_array:

Le mécanisme smart_resize a été mis en place pour gérer des tableaux dynamiques qui changent souvent de taille ou pour construire efficacement des tableaux dont la taille initiale est inconnue sans passer par une liste. La méthode append_array(x) ajoute un élément au tableau et y stocke la valeur x. Cette méthode n'est accessible que si le tableau est de type « smart_resize ».

L'état interne du tableau est maintenant contrôlé de façon plus rigoureuse. Il est interdit de faire un resize d'un tableau de type ref_data, ou d'un tableau normal si un autre tableau utilise la même zone de mémoire (tableau construit par ref_array(...)). Le comportement de operator= s'en trouve affecté.

Il est déconseillé d'utiliser le mécanisme ref_array a cause des effets de bord sur le tableau qui sert de source, sauf si on cherche explicitement à empêcher tout resize du tableau source. Une REF(ArrayOf____) est généralement suffisament efficace.