Slide 1

EARTH et Threaded-C:

Éléments clés du manuel de références de Threaded-C

Bref historique de EARTH et Threaded-C

Ancêtres de l'architecture EARTH:

- Machine à flux de données *statique* de J.B. Dennis (le père du *dataflow*, MIT, 1975) = pas de procédures récursives mais *software pipelining*
- *Argument-fetching dataflow machine* de G.R. Gao et J.B. Dennis (McGill, 1988) = machine *dataflow sans* flux de données
- *Multi-Threaded Architecture* de G.R. Gao et al. (McGill, 1994) = Processeur ordinaire + Unité indépendante de synchronisation

$\Rightarrow$ Architecture EARTH en 1995 (*University of Delaware*)

= *Efficient Architecture for Running THreads*

Slide 2

Initialement, deux langages pour programmer la machine EARTH:

1. Threaded-C $\approx$ langage *machine* $\Rightarrow$ langage cible pour des compilateurs, pas pour des programmeurs
2. EARTH-C = langage parallèle de plus haut niveau: boucles `forall`, procédures parallèles, mémoire partagée, etc.

Faiblesses de EARTH-C: fossé sémantique important $\Rightarrow$ tâche trop complexe pour le compilateur (dépendances entre pointeurs) $\Rightarrow$ Threaded-C devient *le* langage de développement

Évolution de Threaded-C:

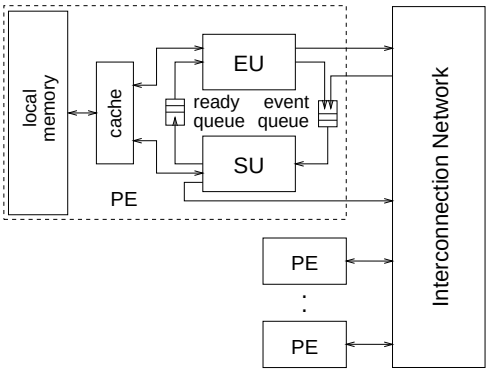
- Threaded-C 1.0 (1995): Langage de base avec *slots* et *thread*s numériques seulement $\approx$ langage machine
- Threaded-C 1.1 (1998): *Slots* et *thread*s symboliques $\approx$ langage d'assemblage
- Threaded-C 2.0 (2000): uniformisation des noms, déclaration plus simple des *slots*, surcharge des primitives, primitives pour exclusion mutuelle

Slide 3

3.1 L'architecture EARTH

EU = *Execution Unit* — *Ready queue* = Fibres prêtes à s'exécuter

SU = *Synchronization Unit* — *Event queue* = Requêtes de synchronisation



Fibre = séquence d'instructions (d'une même activation de fonction)

Slide 4

3.2 Fibres

- Une fibre est séquence d'instructions exécutée selon la sémantique ordinaire du langage C
- Une fibre est associée à une adresse dans le code du programme
  (`FP + IP` = `Frame Pointer + Instruction Pointer`)
- Une fibre est activée dans un style *dataflow*: lorsqu'une fibre a reçu suffisamment de signaux, son exécution peut être amorcée
- Une fibre est exécutée de façon non-préemptive et non-bloquante: lorsqu'une fibre amorce son exécution (EU), elle la poursuit jusqu'à la fin
- Conduit à un style de programmation *split-phase* avec des petites fibres: une fibre qui désire effectuer une opération avec un long temps de latence fait la demande ... puis la réponse est traitée par *une autre* fibre

Slide 5

3.2.1 *Slots* de synchronisation (*sync slots*)

- Une *sync slot* sert à recevoir des signaux de synchronisation
- Une *sync slot* est caractérisée par 3 attributs:
  1. Valeur courante = nombre de signaux requis avant que la *slot* ne devienne 0 et que la fibre associée ne soit activée
  2. Compte de ré-initialisation (*reset count*) = valeur à utiliser comme nouvelle valeur courante lorsque le compte tombe à 0
  3. Fibre associée = la fibre à activer lorsque le compte devient 0
- À chaque fois qu'un signal est reçu par une *sync slot*, sa valeur courante est diminuée de 1

Slide 6

3.2.2 Ordre d'exécution des fibres

Seule contrainte du modèle EARTH:

- Une fibre devient prête à s'exécuter aussitôt que la valeur courante de la *sync slot* qui la contrôle tombe à 0

$\Rightarrow$ modèle *dataflow* = les fibres s'envoient des signaux, par l'intermédiaire de *sync slots*, pour indiquer que les données requises sont disponibles

Mise en oeuvre typique du *pool* de fibres = queue FIFO

$\Rightarrow$ les fibres prêtes à s'exécuter sont mises dans une file d'attente

Mise en oeuvre sur machine SMP:

$\Rightarrow$ Plusieurs fibres peuvent, sur un même processeur, être en train de s'exécuter en même temps

Slide 7

3.3 Les fonctions (procédures) *threaded*

- Une fibre s'exécute toujours dans le *contexte* d'une activation de fonction dite "fonction *threaded*"
- Fonction *threaded* = fonction qui doit être exécutée en parallèle et qui contient 1 ou plusieurs fibres

⇒ Les fibres d'une même activation de fonction peuvent communiquer entre elles par l'intermédiaire de variables partagées = paramètres et variables locales de la fonction, variables globales

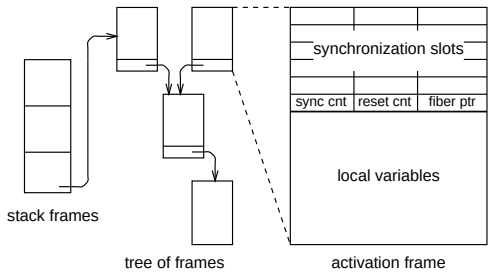
Remarque importante: En Threaded-C, lorsqu'on parle d'une *fonction threaded*, il s'agit plus précisément de *procédure threaded*: une fonction *threaded* ne peut pas retourner de résultat comme une fonction ordinaire mais doit plutôt utiliser des paramètres résultats (par référence)

Slide 8

Activation *parallèle* des fonctions

⇒ les fonctions ne se terminent pas nécessairement dans l'ordre dans lequel elles sont appelées

⇒ les blocs d'activation *ne peuvent pas* être conservés sur une pile



Activation frame partagé par les fibres d'une activation ⇒ toutes les fibres d'une même fonction doivent s'exécuter sur le même processeur

Slide 9

4 Introduction à Threaded-C

Exemples (réfèrent au manuel de référence du langage Threaded-C):

- `Hello world!` (pp. 11–12), `recfib` (p. 17), `nqueens` (pp. 21–22)

Synchronisation locale vs. non-locale (à distance = *remote*)

- `void XXX_SYNC( SLOT_JD ); /* Slot locale. */`
- `void XXX_SYNC( SPTR ); /* Slot non-locale. */`

Quelques primitives EARTH:

- `SPTR TO_SPTR( SLOT_JD ):` Convertit une slot locale en un pointeur non-local (de type `SPTR` = Slot PoinTeR)
- `SPAWN( FIBER_JD ):` Cédule une fibre directement, sans passer par sa *slot* de synchronisation
- `TOKEN( foo, arg_1, ..., arg_n );`: Crée une activation de la fonction `foo`, avec les arguments indiqués, sur un processeur choisi par le système (plus précisément, choisi par le *load balancer*)

Slide 10

Transferts de données

Modèle de mémoire sous-jacent à la machine EARTH

= mémoire distribuée

$\Rightarrow$ impossible d'accéder directement à la mémoire d'un autre processeur

Style C de manipulations de données = Utilisation de pointeurs

Accès mémoire non-local $\Rightarrow$ Utilisation de pointeurs spéciaux

```
int *GLOBAL pt;  /* pt refere a une memoire non-locale,
                    dans un autre processeur. */
```

`OWNER_OF(pt)` = Numéro du processeur où est située cette addresse

`TO_LOCAL(pt)` = Référence locale d'un pointeur
(valable seulement si on est dans le processeur propriétaire)

`TO_GLOBAL(pt)` = Convertit un pointeur local en adresse globale

Slide 11

Opérations de transfert de donneés

Écriture d'une donnée:

```
PUT_SYNC( T datum, T *GLOBAL dest, SLOT_ID s );
PUT_SYNC( T datum, T *GLOBAL dest, SPTR s );
```

Lecture (*split-phase*) d'une donnée:

```
GET_SYNC( T *GLOBAL src, T *GLOBAL dest, SLOT_ID s );
GET_SYNC( T *GLOBAL src, T *GLOBAL dest, SPTR s );
```

Note: PUT et GET s'utilisent pour des types T pouvant être passés *par valeur*

Transfert des blocs de données, e.g., tableaux:

```
BLKMOV_SYNC(void *GLOBAL src, void *GLOBAL dest, long length, SLOT_ID s);
BLKMOV_SYNC(void *GLOBAL src, void *GLOBAL dest, long length, SPTR s);
```

Slide 12

Façons typiques pour des fibres de communiquer entre elles:

- Fibres dans un même activation de fonction: par l'intermédiaire des variables locales à la fonction (variables partagées)
- Fibres dans des activations différentes sur un même processeur: par l'intermédiaire de variables de portée globale (déclarées à l'extérieur d'une déclaration de fonction) ou par l'intermédiaire du tas local (*heap*, i.e., allocation dynamique avec pointeurs)
- Fibres dans des activations différentes sur des processeurs différents: à l'aide d'opérations de transferts de données (PU/TGET_SYNC) portant sur des variables privées (à la fonction) ou des objets de leur tas