



hwlocality : Un binding Rust pour hwloc

Hadrien Grasland

2023-11-15

Parlons cache

- Pour des raisons de perf, on veut souvent **tenir en cache**
 - On ajustera donc nos volumes de données, boucles...
 - Mais pour **quel cache ?** L1 ? L2 ? L3 ? L4 ? LLC ?
- Amortissement des coûts fixe → **L1 pas toujours le top !**
 - Dépend du compromis coût fixe vs *throughput*
 - Dépend de l'intensité arithmétique du programme
 - Dépend du niveau de parallélisme, de l'affinité...
- **Autotuning ?** Complicqué et **risqué** (tâches de fond !)

Connaître la taille des caches ?

- Commençons par essayer **sous Linux**
 - Envie de parser quelques Ko de texte @ /proc/cpuinfo ?
 - La sortie de lscpu dépend beaucoup de la version utilisée !
- Et un jour, fatalement, on doit supporter **d'autres OS**
 - Utilisateurs macOS ? Une API qui n'a rien à voir
 - Utilisateur Windows ? Une autre API qui n'a rien à voir
- ...bref, vous voyez l'idée. POSIX ne vous sauvera pas ici.

Accès non uniforme aux ressources

- Certains croient encore au père Noël, d'autres au SMP*
 - La plupart des serveurs ont des **nœuds NUMA**
 - Beaucoup de CPUs clients ont des **caches asymétriques**
 - **Hyperthreading** → L1/L2 partagé entre paires de « coeurs »
- Pour maximiser la perf, il faut oublier les nombres agrégés
 - Accepter que la hiérarchie mémoire/calcul est un arbre
 - Privilégier la **communication entre branches voisines**

* Symmetrical Multi-Processing, ce compte de fées où tous les CPUs ont une perf identique et un accès identique aux ressources système. Très rare côté hardware depuis les années 2010.

hwloc

- Une bibliothèque de l'INRIA Bordeaux + quelques autres
- Donne un accès abstrait par rapport à l'OS...
 - A la **hiérarchie** du matériel (« topologie »)
 - A ses **propriétés** (ex : taille cache, taille ligne, associativité...)
 - A son **hétérogénéité** (ex : coeurs P/E, mémoire DDR/HBM...)
 - Au **contrôle** du placement des tâches/allocations
- Mais si c'est si bien, pourquoi tout le monde ne l'utilise pas ?

Observons deux points d'entrée...

```
int  
hwloc_set_membind(  
    hwloc_topology_t topology,  
    hwloc_const_bitmap_t set,  
    hwloc_membind_policy_t policy,  
    int flags  
)
```

- Types imprécis
- Pointeurs bruts dans tous les sens
- Noms pourris
- Paramètres interdépendants

```
static void *  
hwloc_alloc_membind_policy(  
    hwloc_topology_t topology,  
    size_t len,  
    hwloc_const_bitmap_t set,  
    hwloc_membind_policy_t policy,  
    int flags  
)
```

...bref, hwloc est une bibliothèque C.
Un binding Rust peut-il faire mieux ?

Comment améliorer l'ergonomie ?

- Sans aller dans le subjectif, il y a des principes consensuels
 - **Typage fort** → Moins de valeurs/opérations interdites
 - Rester familier par la surcharge *raisonnée** d'**opérateurs**
 - Détecter un maximum d'**erreurs à la compilation**
 - Erreurs d'exécution non ignorables avec rapports précis
 - **Eviter les APIs piégeuses** (paramètres interdépendants, pointeurs, libération manuelle de ressource...)
 - Avoir une **documentation** précise et complète
 - Rester accessible pour ceux qui connaissent l'API C

* Le chemin qui commence par l'utilisation de + pour la concaténation de chaînes et << pour l'affichage de texte dans la console se termine par JavaScript et PHP.

Exemple : Les bitmaps de hwloc

- Les bitmaps sont omniprésents dans l'API hwloc
 - Liste de CPUs logiques (« cpuset »)
 - Liste de nœuds NUMA (« nodeset »)
- **Viennent de la topologie** ou **alloués par l'utilisateur**
- Ils ont des opérations booléennes, des outils d'itération...
- **Indices** compris entre 0 et INT_MAX. **-1 = pas d'indice.**
- Certaines fonctions n'acceptent que des cpuset
 - D'autres peuvent accepter des nodesets **avec le bon flag**

Les bitmaps version hwlocality

- Types **CpuSet** et **NodeSet** distincts (donc cpu | node illégal)
 - On peut accepter les deux via le **trait SpecializedBitmap**
- **Libération auto** RAll + références **BitmapRef<'topology, B>**
 - Le compilateur assure la validité des BitmapRef
- Surcharge raisonnée des **opérateurs**
 - Feeling similaire au HashSet de la bibliothèque standard
- **Type BitmapIndex dédié***, Option<BitmapIndex> si besoin

* Comme un type custom n'a pas tout à fait l'ergonomie des entiers standard (ex : pas de littérales), l'utilisation de usize est aussi autorisée, avec panic lors des accès hors borne.

Binding de `hwloc_set_membind()`

```
[{-] pub fn bind_memory<Set: SpecializedBitmap>( source  
    &self, // Syntaxe méthode : topology.bind_memory()  
    set: &Set, // Accepte CpuSet et NodeSet sans triturer les flags  
    policy: MemoryBindingPolicy, // Erreurs mieux documentées et impossible à ignorer  
    flags: MemoryBindingFlags  
) -> Result<(), MemoryBindingError<Set::Owned>>
```

Set the default memory binding policy of the current process or thread to prefer the NUMA node(s) specified by `set`.

Memory can be bound by either `CpuSet` or `NodeSet`. Binding by `NodeSet` is preferred because some NUMA memory nodes are not attached to CPUs, and thus cannot be bound by `CpuSet`.

You must specify exactly one of the `ASSUME_SINGLE_THREAD`, `PROCESS` and `THREAD` binding target flags when using this method.

Pas de comportement surprenant quand on oublie un flag

Requires `MemoryBindingSupport::set_current_process()` or `MemoryBindingSupport::set_current_thread()` depending on flags.

Support OS requis mieux documenté

Errors

- `BadFlags` if the number of specified binding target flags is not exactly one
- `BadSet` if the system can't bind memory to that CPU/node set
- `Unsupported` if the system cannot bind the current thread/process with the requested policy

Autre exemple

```
hwloc_obj_t  
hwloc_get_obj_by_depth(  
    hwloc_topology_t topology,  
    int depth,  
    unsigned idx  
)
```

- Pointeurs de portées interdépendantes
- Indices positifs ordonnés, bornés par une autre fonction + indices négatifs spéciaux non ordonnés
- Indices bornés par une autre fonction

Version hwlocality

```
[~] pub fn objects_at_depth<DepthLike>(                                     source
    &self,                               Syntaxe méthode : topology.objects_at_depth()
    depth: DepthLike
) -> impl DoubleEndedIterator<Item = &TopologyObject> + Clone + ExactSizeIterator + FusedIterator
where
    DepthLike: TryInto<Depth>,        Retourne un itérateur, vide pour les profondeurs non habitées,
    <DepthLike as TryInto<Depth>>::Error: Debug,    pas d'indice d'objet en entrée
```

`TopologyObjects` at the given `depth` Utilisation des références validée par le compilateur

Accepts `Depth`, `NormalDepth` and `usize` operands. Use the former two for type-safety (they are guaranteed to be in range as a type invariant) or the latter for convenience (it is more tightly integrated with Rust's built-in integer support, for example it supports integer literals).

Distinction marquée entre profondeurs normales et spéciales (on garde `depth` en entrée pour gérer les profondeurs spéciales)

Et si je connais déjà hwloc ?

- On peut interroger la doc par nom de type ou fonction C

hwloc_alloc_mbind_policy ? 

Results in

	In Names (3)	In Parameters (0)	In Return Types (0)
method	hwloc_alloc_mbind_policy - see hwlocality::topology::Topology::binding_allocate_memory		Allocate some memory on NUMA nodes specified ...

- Pointe vers l'équivalent dans l'API Rust
- Et pour les cas *très* tordus, on peut aussi utiliser le backend hwlocality_sys qui suit fidèlement l'API C...

Statut du projet

- Code en ligne sur <https://github.com/HadrienG2/hwlocality>
- Déjà utilisable comme dépendance git, sans garanties
- Couvre presque tout hwloc, sauf **quelques fonctions d'interop***
- Pour ce qui est de la publication « propre » sur crates.io...
 - hwlocality_sys est déjà disponible
 - hwlocality le sera après le grand nettoyage de fin de projet
 - Amélioration de la couverture de tests
 - Dernière revue d'API avant stabilisation semver

* Les contributions sont bienvenues, c'est souvent une question d'accès au hardware.

En conclusion

- La localité des ressources de calcul, c'est important
 - ...et plus subtil qu'il n'y paraît
- hwloc l'expose avec portabilité entre OSes
 - ...mais c'est une bibliothèque C, donc difficile à utiliser
- hwlocality, c'est hwloc sans les problèmes du C
 - A la base un sous-produit de Gray-Scott Battle
 - Mais finalement, en bonne voie vers la publication

Et pourquoi pas en C++ ?

- Nuance CpuSet/NodeSet bien plus facile avec les traits Rust
 - Concepts C++20 = jouet cassé en comparaison
- C++ = Aucune vérification des erreurs courantes (références, concepts, itérateurs, indices, arithmétique, threads...)
- Itérateurs C++ déficients, ranges abonnés absents
 - Gestion de deps cauchemardesque → Pas juste pour ça !
- Les *newtypes* forts sont (encore plus) un cauchemar en C++
- Vous préférez Cargo et Rustdoc ou CMake et Doxygen ?

Merci de votre attention !