HPC – Projet final

Optimisation de DME (Rust)

Aubry Mangold et Samuel Roland

11 juin 2025

Sommaire

Contexte et architecture

- DME : « Delightful Markdown Experience » (projet scolaire)
- Conversion de Markdown vers HTML/CSS via Comrak
- Recherche de fichiers Markdown dans le système
- Architecture :
  - `dme-core` (Rust) + front VueJS/Tauri

Rust et Tree-Sitter

- Rust : modèle mémoire strict, ownership & lifetimes
- Comrak pour parser Markdown
- Tree-Sitter pour syntax highlighting
  - CST (Concrete Syntax Tree)
  - Queries & HighlightConfiguration
  - Difficultés avec le modèle mémoire

Tests et infrastructure de benchmark

- Tests unitaires et d'intégration
- Benchmarks intégrés avec le binaire `bench`
- Exécution systématique en `--release` (+ debug symbols pour Perf)

Problème initial de colorisation

- `HighlightConfig` recréé pour chaque snippet
- 117 snippets mènent à 117 initialisations coûteuses
- Perf montre un pic dans `HighlightConfig::new`

Résultats: TODO graph v1

Optimisation de la colorisation

- Cache global `TSH_CACHE: Lazy<RwLock<HashMap>>`
- Lecture rapide des highlighters existants
- Écriture (creation) uniquement au premier usage par langue

Résultats: TODO grpah v2

Optimisation de l'installation de grammaires

- Poids élevé dû à l'historique Git (.git 49 M)
- Passage à `git clone --depth 1 --single-branch`
  - Ajout de paramètres `only_latest_commits` et `single_branch`

TODO graph v3

Optimisation de la recherche

- Indexation rapide du dépôt MDN
- Fuzzy matching sur titres et chemins
- Benchmark général_keyword : 159 ms (index + recherche)
- Streaming des résultats pour réactivité
- Pas eu le temps d'optimiser

Conclusion et perspectives

- Transposition C ⟶ Rust présente des défis autour du modèle mémoire
- Importance des tests et bench intégrés
- Gains majeurs avec mise en cache et clone léger
- Pistes futures :
  - Parallélisation du cache (lecture concurrente)
  - Optimisation de la recherche fuzzy