HPC - Labo1 - Analyse ECG

BleuerRémy

22.02.2026

Objectif

Implémenter un algorithme de détection de pics R dans un signal ECG en C, en utilisant la méthode de Pan-Tompkins. J'ai également essayé de privilégier la performance, que ça soit l'utilisation de la mémoire ou la latence.

Méthode

En s'inspirant de l'algorithme de Pan-Tompkins, j'ai implémenté les étapes suivantes :

1. Filtre passe-haut avec `ecg_highpass_ma()`.
2. Dérivée du signal avec `ecg_derivative_1()`.
3. Mise au carré du signal avec `ecg_square()`.
4. Intégration fenêtre glissante avec `ecg_mwi()`.
5. Détection seuil adaptif + période réfractaire.

Choix d'implémentation

- **Pas d'alloc dynamique** : Les buffers sont alloués une seule fois dans `ecg_Create()`. Pas de malloc utilisés pendant l'analyse.
- **Somme gissante en O(n)** : Le filtre passe-haut et l'intégration utilisent des sommes gissantes pour éviter les recalculs redondants.
- **Seuil adaptatif en O(1)** : Le seuil de détection est ajusté dynamiquement à chaque pic san historique. Aucun tableau et recalcul nécessaire.
- **Accès séquentiel** : Passe linéaire sur tableau contigu.

Résultats

En lancant le programme avec `../80bpm0.csv ../resultat_analyse.json` comme paramètre pour spécifier le fichier d'entrée et de sortie, j'obtiens ce résultat dans le terminal :

```
CSV chargé avec 12 leads et 7500 échantillons.
[ECG] fs=500 Hz, low_pass_window=65 samples, mwi_window=65 samples, refractory_samples=135 samples
[ECG] Pics R détectés: 20
[ECG] Intervalles RR valides: 19
[ECG] Fréquence cardiaque moyenne: 79.9 BPM
[ECG] RR moyen: 0.751 s
20 pics R détectés.
Analyse terminée. Résultats sauvegardés dans ../resultat_analyse.json
```

Figure 1: resultat analyse

Le fichier `resultat_analyse.json` se trouve à la source du projet.

Et voici le résultat du scripte python pour afficher le plots avec les deux fichiers :

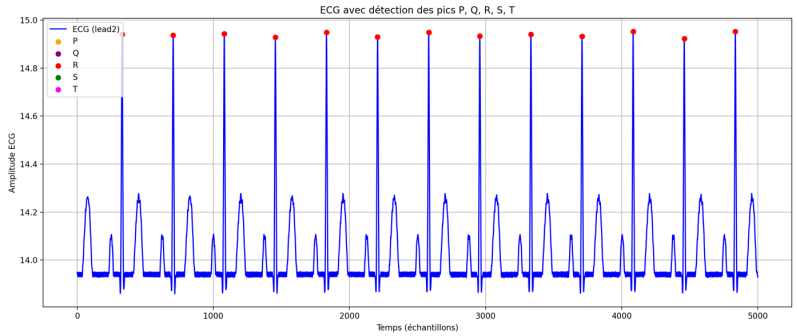

Figure 2: plot ex 7.1

Benchmark

Hyperfine

En utilisant `hyperfine` pour mesurer les performances de l'implémentation avec cette commande :

```
hyperfine './cmake-build-debug/ecg_dealination 80bpm0.csv resultat_analyse.json'
--warmup 100 -N --export-json=resultat_hyperfine.json
```

J'ai obtenu les résultats suivants :

```
Benchmark 1: ./cmake-build-debug/ecg_dealination 80bpm0.csv resultat_analyse.json
  Time (mean ± σ):       4.2 ms ±   0.2 ms    [User: 3.2 ms, System: 0.6 ms]
  Range (min … max):     3.9 ms …   6.4 ms    733 runs
```

Figure 3: resultat hyperfine

gtime

J'ai également utilisé `gtime` qui est un équivalent de `usr/bin/time` pour mac, afin de mesurer le temps d'exécution de l'implémentation avec cette commande :

```
gtime -v ./cmake-build-debug/ecg_dealination 80bpm0.csv resultat_analyse.json
```

Rien de très concluant, car temps d'exécution est très rapide, mais voici les résultats :

```
Command being timed: "./cmake-build-debug/ecg_dealination 80bpm0.csv resultat_analyse.json"
User time (seconds): 0.00
System time (seconds): 0.00
Percent of CPU this job got: 75%
Elapsed (wall clock) time (h:mm:ss or m:ss): 0:00.01
Average shared text size (kbytes): 0
Average unshared data size (kbytes): 0
Average stack size (kbytes): 0
Average total size (kbytes): 0
Maximum resident set size (kbytes): 3424
Average resident set size (kbytes): 0
Major (requiring I/O) page faults: 3
Minor (reclaiming a frame) page faults: 357
Voluntary context switches: 6
Involuntary context switches: 8
Swaps: 0
File system inputs: 0
File system outputs: 0
Socket messages sent: 0
Socket messages received: 0
Signals delivered: 0
Page size (bytes): 16384
Exit status: 0
```

Figure 4: resultat gtime

Conclusion

L'implémentation de l'algorithme de Pan-Tompkins en C a permis de détecter efficacement les pics R dans le signal ECG. En optimisant l'utilisation de la mémoire et en évitant les recalculs redondants, j'ai réussi à obtenir une solution performante. Les résultats obtenus sont cohérents avec les attentes, et les benchmarks montrent que l'implémentation est rapide.