Embedded Xinu (Raspberry Pi 3) 上の GC 連動・分散 N-Queens ベンチマーク報告

Yasushi Kodama

2026 年 6 月 2 日

1　概要

ベアメタル Embedded Xinu (arm-rpi3 / BCM2837 / Cortex-A53) 上で動作する AIPL アクター・ランタイムの負荷分散フレームワーク (Dispatcher + Worker 群) を用い、**周期 GC アクターを有効にした状態で分散 N-Queens 問題のベンチマーク**を実施した。解の正しさ・スループット・GC の動作を確認し、計測中に判明した制御プレーンのボトルネックと、その改善 (Collector への push 集約) を報告する。ビルド ID は `xinu-gwm-b69`。

2　システム構成

- **Pi 3 (arm-rpi3)**: 有線 Ethernet 経由の HTTP サーバ `webactor` (192.168.3.50:8080) が AIPL ランタイムを制御。
- **負荷分散**: `Dispatcher` 1 体 + `Worker` 4 体 (`/api/loadbal/init`)。Dispatcher は最小負荷のワーカーへタスクを振り分ける。
- **GC**: `Collector` アクター (`/api/gc-actor/init`, `enable`) を起動。周期 5 s でアクター表をスイープし、アイドル閾値 30 s を超えた使い捨てアクターを回収する。
- **計測ドライバ**: Mac 側 `tools/nqueens_bench.py` (参照解は Mac で計算し照合)。

3　ベンチマーク手法

N-Queens を**第 1 列ごとに分割**し、各列を 1 個の `compute_nq` タスクとして Dispatcher へ投入する (`/api/loadbal/nqueens?n=N`)。各 Worker が担当列の部分解数を数え、Dispatcher へ `done(idx, task_id, result)` を返す。全列の部分解数の総和が N-Queens の総解数となる。GC を有効にしたまま実行し、計算アクターの生成・回収が並行して行われる状況を再現した。

N	分散実行 (Pi 3) 総解数	既知の正解	判定
8	**92**	92	一致
9	**352**	352	一致

表 1　分散 N-Queens の総解数 (Pi 3、列分散・GC 有効)

4　結果

4.1　解の正しさとスループット

$N = 8$ の第 1 列ごとの内訳は左右対称で $\{4, 8, 16, 18, 18, 16, 8, 4\}$、総和 92 と一致した。各 `compute_nq` タスクの実計算時間は Pi 3 上で約 **20 ms/タスク**。照合用の Mac 参照計算は $N = 8$ で 92 解を 2 ms。

4.2　GC の動作

ベンチ実行中の GC アクター統計を以下に示す。

```
gc-actor obj=6 enabled=1 period_ms=5000 threshold_ms=30000
sweep_count=1 swept_total=0 last_scanned=7
```

周期スイープが実行され (`sweep_count`≥1)、アクター 7 体をスキャンした。`swept_total=0` は**回収対象なし**を意味する。Dispatcher・Worker・Collector は常駐であり、N-Queens タスクは部分解数を返すのみで使い捨てアクターを残さないため、回収すべきゴミが発生しないのは正しい挙動である。

5　考察: 制御プレーンのボトルネック

分散計算そのものは正しく高速 (約 20 ms/タスク) であったが、**Pi 3 の `webactor` は単一スレッド**であり、Worker の N-Queens 計算が走ると HTTP 制御・集計が一時的に CPU 飢餓になる。当初の集計方式は Mac が各タスクを `/api/loadbal/task?id=K` で個別ポーリングして合計する方式であったため、計算負荷下で HTTP 応答が滞り、全自動ベンチ (`nqueens_bench.py`) が 300 s の予算内に集計を完了できずタイムアウトした。すなわちボトルネックは**分散計算ではなく、計算と同一コアを奪い合う制御プレーン (HTTP ポーリング)** にある。

6　改善: Collector への push 集約

上記を受け、集計を**ポーリングから push 集約へ**変更した (ビルド `b69`)。

- Worker は従来どおり完了時に Dispatcher へ `done` を **push** する。
- Dispatcher の `done` ハンドラで N-Queens タスク (kind=`'n'`) の部分解数をその場で**積算**する (`g_nq_sum`, `g_nq_received`)。
- Mac は新設の `/api/loadbal/nqueens-result` を **1 回**問い合わせるだけで、集約済みの総和と完了状況 (`done/received/expected/total`) を得る。

これにより、N 個のタスクに対する HTTP 集計リクエスト数が $O(N)$ **回** $\to$ **1 回**へ削減され、計算と制御プレーンの競合が大幅に緩和される。集約は Pi 3 側 (Dispatcher) で完結し、Mac 側は受動的に総和を読むだけとなる。

7 結論

GC を有効にした状態で、分散 N-Queens は $N = 8 \to 92$、$N = 9 \to 352$ と**正答**を得た。GC は周期スイープを実行し (常駐アクターのみのため回収 0)、計算と並行して正常に動作した。スループット上の制約は分散計算ではなく単一スレッド HTTP 制御プレーンにあることを特定し、**Collector への push 集約** ($O(N) \to 1$ の集計リクエスト削減) として改善を実装した。