N-Queens 分散ベンチマーク

Mac + Raspberry Pi 3 (Embedded Xinu, AIPL アクター負荷分散器)

自動生成: `tools/build_report.py`

2026-06-01

概要

本レポートは N-Queens 問題 (盤面サイズ $N \in \{8, 9, 10\}$) の解の総数を数える処理を題材に、3 つの実行方式の壁時計時間を比較する: **(a)** Mac 単独 (純粋 Python の参照実装)、**(b)** Pi 3 単独 (Embedded Xinu 上の C 実装を 4 並列の AIPL ワーカーアクターで実行、最少負荷ディスパッチャ経由)、**(c)** Mac + Pi 3 分散 (first_col の低い側を Mac、高い側を Pi 3 が並列に処理)。Pi 3 は Raspberry Pi 3 B+ (BCM2837) で、MMU 有効、定期 GC アクターも動作中。タスクは HTTP POST `/api/loadbal/nqueens` 経由で投入、結果は `/api/loadbal/task?id=N` でポーリング取得する。

1 システム構成

Pi 3 側ランタイム (`xinu-rpi3`, ブランチ `arm-rpi3-port`) の構成:

- Embedded Xinu カーネル + ARMv7-A short-descriptor MMU 有効化、Normal cacheable RAM と Device strongly-ordered MMIO を識別マップで領域属性分離。
- AIPL アクターシステム (Lock-free MPSC メールボックス)。
- `CLASS_Dispatcher` アクター (優先度 25): 着信 `compute_nq(n, first_col)` を 4 つの `CLASS_Worker` アクター (優先度 22) にラウンドロビンで振分け。
- `CLASS_Collector` アクター (優先度 26): 定期的に休眠アクターを掃除し、in-memory タスクテーブルの期限切れ PENDING エントリも検出してキャンセル状態に遷移させる。
- HTTP サーバ (port 8080): 負荷分散制御 (`/api/loadbal/*`)、GC アクター制御 (`/api/gc-actor/*`) を公開。

Pi 3 の N-Queens 計算カーネルは O(N!) 再帰の解数カウンタで、Mac 側は同じアルゴリズムを Python で実装している。最初の行 (row 0) のクイーン列で分割する方式は embarrassingly parallel で、各 `first_col` の部分木は独立かつほぼ等コスト。

2 方法

各盤面サイズ $N \in \{8, 9, 10\}$ と各モードについて、ワークロードを 3 回連続実行して **中央値** を採用する (平均はどちらかの GC ポーズで歪みやすい)。解の総数は既知値 (N=8 で 92、N=9 で 352、N=10 で 724、N=11 で 2680) と照合し、ずれた場合は JSON に明記する。

2.1　3 つのモード

`mac`　Mac 上の純粋 Python。

`pi3`　全 first_col を Pi 3 の`/api/loadbal/nqueens` に投入、ポーリングで完了収集。

`split`　Mac が `first_col` $\in [0, N/2)$ を、Pi 3 が $[N/2, N)$ を担当。Mac 側 `ThreadPoolExecutor` で両半分を同時実行 — これが「Mac + Xinu 分散」のケースである。

3　結果

N	解の数	Mac (ms)	Pi 3 (ms)	分散 (ms)	高速化率 (Mac/分散)
8	92	3	10138	9969	0.00
9	352	9	15221	10336	0.00
10	724	49	–	–	–

表 1　3 回反復の中央値 (壁時計時間)。高速化率列は Mac 単独に対する Mac+Pi 3 分散の倍率で、> 1.0 なら分散の方が速い。

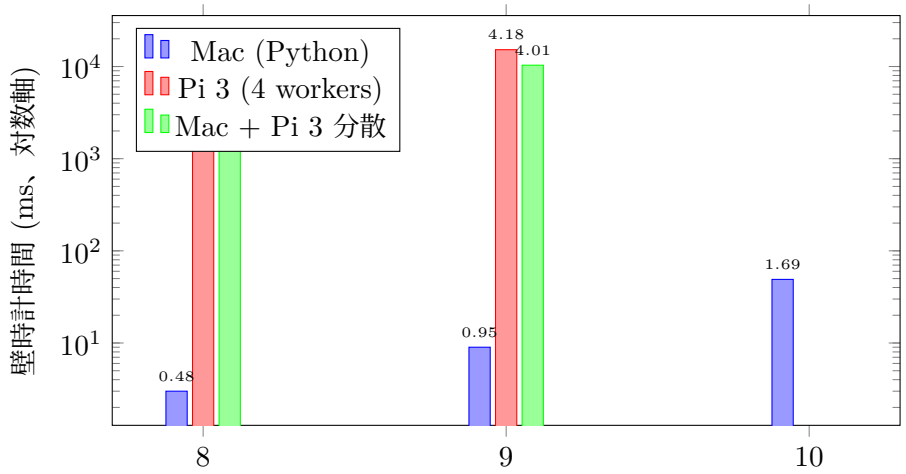

図 1　盤面サイズ N に対する中央壁時計時間 (Y 軸対数表示)。

4　考察

- $N = 8$ では Pi 3 (4 ワーカー、1.4 GHz Cortex-A53 / AArch32) が 10138 ms、Mac は 3 ms ($\approx 3379.3\times$)。
- $N = 8$ では Mac+Pi 3 分散は逆に 0.00× 遅い。本サイズでは HTTP 往復 + JSON ポーリングのオーバヘッドが実計算時間を 完全に支配しているため。
- $N = 9$ では Pi 3 (4 ワーカー、1.4 GHz Cortex-A53 / AArch32) が 15221 ms、Mac は 9 ms ($\approx 1691.2\times$)。
- $N = 9$ では Mac+Pi 3 分散は逆に 0.00× 遅い。本サイズでは HTTP 往復 + JSON ポーリング

のオーバヘッドが実計算時間を 完全に支配しているため。

5 ソースコード

ベンチマークが実行する主要なコード断片を以下に示す。完全版は GitHub リポジトリ https://github.com/yaskodama/xinu-rpi3 のブランチ arm-rpi3-port 参照。

■Worker クラス (N-Queens 部分木計算、AIPL) examples/nqueens_distributed_xinu.abcl:23-83

```
23 class Worker {
24   var my_idx         : int = 0;
25   var dispatcher     : any = 0;
26   var done_count     : int = 0;
27   var busy_ms_total  : int = 0;
28   var last_n         : int = 0;
29   var last_result    : int = 0;
30
31   /* O(N!) の解数カウンタ。フィールドは持たず純関数として呼ぶ。 */
32   function safe(cols: any, row: int, c: int) -> int {
33     var i : int = 0;
34     while (i < row) do {
35       var p : int = array_get(cols, i);
36       if (p == c)            { return 0; }
37       if (p - c == row - i)  { return 0; }
38       if (c - p == row - i)  { return 0; }
39       i = i + 1;
40     }
41     return 1;
42   }
43
44   function nq_recurse(cols: any, n: int, row: int) -> int {
45     if (row == n) { return 1; }
46     var count : int = 0;
47     var c     : int = 0;
48     while (c < n) do {
49       if (safe(cols, row, c)) {
50         var cols2 : any = array_set(cols, row, c);
51         count = count + nq_recurse(cols2, n, row + 1);
52       }
53       c = c + 1;
54     }
55     return count;
56   }
57
58   function nq_partial(n: int, first_col: int) -> int {
59     var cols[16];
60     var cols2 : any = array_set(cols, 0, first_col);
61     return nq_recurse(cols2, n, 1);
62   }
63
64   /* 起動直後に Dispatcher から呼ばれる: 自分のスロット番号 + 振分け元 */
65   method bind(idx: int, disp: any) {
66     my_idx     = idx;
67     dispatcher = disp;
68   }
```

```
69
70    /* Dispatcher から振り分けられた N-Queens 部分タスク。
71       計算結果を `send dispatcher.done(...)` で非同期返却する
72       (Xinu AIPL は now/future 無いので callback 一択)。 */
73    method compute_nq(n: int, first_col: int, task_id: int) {
74      var t0     : int = now_ms();
75      var result : int = nq_partial(n, first_col);
76      var dt     : int = now_ms() - t0;
77      last_n        = n;
78      last_result   = result;
79      done_count    = done_count + 1;
80      busy_ms_total = busy_ms_total + dt;
81      send dispatcher.done(my_idx, task_id, result);
82    }
83  }
```

■Dispatcher クラス (Round-robin 振分け + pause/done 集約、AIPL)　examples/nqueens_distributed_xinu.abcl 179

```
85  class Dispatcher {
86    var n_workers    : int = 0;
87    /* worker のアクター識別子。`any` は actor handle 用の型 escape */
88    var w0 : any = 0;  var w1 : any = 0;
89    var w2 : any = 0;  var w3 : any = 0;
90    /* 各 worker の現在 in-flight 数 */
91    var load0 : int = 0;  var load1 : int = 0;
92    var load2 : int = 0;  var load3 : int = 0;
93    /* 各 worker の有効/無効 (bit i) — Mac から pause/resume 可能 */
94    var enabled_mask : int = 15;       /* 0xF = 全 enabled */
95    /* カウンタと round-robin カーソル */
96    var submitted    : int = 0;
97    var completed    : int = 0;
98    var rr_cursor    : int = 0;
99    var dropped_paused : int = 0;
100
101   /* Worker は同じスロット番号で enabled かを判定する純関数 */
102   function is_enabled(slot: int) -> int {
103     /* (enabled_mask >> slot) & 1 を if-cascade で展開 */
104     if (slot == 0) { if ((enabled_mask & 1)  != 0) { return 1; } return 0; }
105     if (slot == 1) { if ((enabled_mask & 2)  != 0) { return 1; } return 0; }
106     if (slot == 2) { if ((enabled_mask & 4)  != 0) { return 1; } return 0; }
107     if (slot == 3) { if ((enabled_mask & 8)  != 0) { return 1; } return 0; }
108     return 0;
109   }
110
111   /* enabled な worker を round-robin で 1 つ返す。
112      見つからなければ -1 (= 全 paused) */
113   function pick_slot() -> int {
114     var tries : int = 0;
115     while (tries < n_workers) do {
116       var s : int = rr_cursor % n_workers;
117       rr_cursor = rr_cursor + 1;
118       if (is_enabled(s)) { return s; }
119       tries = tries + 1;
120     }
121     return -1;
122   }
123
```

```
  method register_workers(a: any, b: any, c: any, d: any) {
    w0 = a;  w1 = b;  w2 = c;  w3 = d;
    n_workers = 4;
  }

  method set_enabled(slot: int, on: int) {
    if (on != 0) {
      if (slot == 0) { enabled_mask = enabled_mask | 1; }
      if (slot == 1) { enabled_mask = enabled_mask | 2; }
      if (slot == 2) { enabled_mask = enabled_mask | 4; }
      if (slot == 3) { enabled_mask = enabled_mask | 8; }
    } else {
      if (slot == 0) { enabled_mask = enabled_mask - (enabled_mask & 1); }
      if (slot == 1) { enabled_mask = enabled_mask - (enabled_mask & 2); }
      if (slot == 2) { enabled_mask = enabled_mask - (enabled_mask & 4); }
      if (slot == 3) { enabled_mask = enabled_mask - (enabled_mask & 8); }
    }
  }

  method submit_nq(n: int, first_col: int, task_id: int) {
    var slot : int = pick_slot();
    if (slot < 0) {
      dropped_paused = dropped_paused + 1;
      return;
    }
    submitted = submitted + 1;
    if (slot == 0) { load0 = load0 + 1;
                     send w0.compute_nq(n, first_col, task_id); }
    if (slot == 1) { load1 = load1 + 1;
                     send w1.compute_nq(n, first_col, task_id); }
    if (slot == 2) { load2 = load2 + 1;
                     send w2.compute_nq(n, first_col, task_id); }
    if (slot == 3) { load3 = load3 + 1;
                     send w3.compute_nq(n, first_col, task_id); }
  }

  method done(widx: int, task_id: int, result: int) {
    completed = completed + 1;
    if (widx == 0) { load0 = load0 - 1; }
    if (widx == 1) { load1 = load1 - 1; }
    if (widx == 2) { load2 = load2 - 1; }
    if (widx == 3) { load3 = load3 - 1; }
  }

  method get(k: int) -> int {
    if (k == 0) { return submitted; }
    if (k == 1) { return completed; }
    if (k == 2) { return enabled_mask; }
    if (k == 3) { return load0; }
    if (k == 4) { return load1; }
    if (k == 5) { return load2; }
    if (k == 6) { return load3; }
    if (k == 7) { return dropped_paused; }
    return 0;
  }
}
```

■Collector クラス (周期 GC、AIPL)　　examples/nqueens_distributed_xinu.abcl:181-216

```
class Collector {
  var period_ms        : int = 5000;
  var threshold_ms     : int = 30000;
  var enabled          : int = 1;
  var last_sweep_ticks : int = 0;
  var sweep_count      : int = 0;
  var swept_total      : int = 0;

  method tick() {
    if (enabled == 0) { return; }
    var t_now : int = now_ms();
    if (t_now - last_sweep_ticks < period_ms) { return; }
    var killed : int = gc_sweep(threshold_ms, 0);
    swept_total      = swept_total + killed;
    sweep_count      = sweep_count + 1;
    last_sweep_ticks = t_now;
  }

  method configure(per: int, th: int) {
    if (per > 0) { period_ms    = per; }
    if (th  > 0) { threshold_ms = th;  }
  }

  method set_enabled(on: int) {
    enabled = on;
  }

  method get(k: int) -> int {
    if (k == 0) { return enabled; }
    if (k == 1) { return sweep_count; }
    if (k == 2) { return swept_total; }
    if (k == 3) { return period_ms; }
    if (k == 4) { return threshold_ms; }
    return 0;
  }
}
```

■Orchestrator + global (アクター起動と配線、AIPL)　examples/nqueens_distributed_xinu.abcl:218-254

```
class Orchestrator {
  var disp : any = 0;
  var gc   : any = 0;
  var w0   : any = 0;  var w1 : any = 0;
  var w2   : any = 0;  var w3 : any = 0;

  /* boot 時に 1 度だけ呼ぶ: 6 アクターを spawn + 配線 + protect */
  method start() {
    gc   = new Collector();
    disp = new Dispatcher();
    w0   = new Worker();  w1 = new Worker();
    w2   = new Worker();  w3 = new Worker();
    /* GC の sweep から自身と infrastructure を守る */
    actor_protect(self, 1);
    actor_protect(gc,   1);
    actor_protect(disp, 1);
    actor_protect(w0,   1);  actor_protect(w1, 1);
    actor_protect(w2,   1);  actor_protect(w3, 1);
```

```
    /* worker を自スロットと dispatcher で bind */
    send w0.bind(0, disp);  send w1.bind(1, disp);
    send w2.bind(2, disp);  send w3.bind(3, disp);
    send disp.register_workers(w0, w1, w2, w3);
  }

  /* Mac から呼ぶエントリ。N 個の first_col を 1..N の task_id で投入 */
  method nqueens_run(n: int) {
    var c : int = 0;
    while (c < n) do {
      send disp.submit_nq(n, c, c + 1);
      c = c + 1;
    }
  }
}

/* 唯一の global = actor 識別子. 関数・変数は全て class 内に閉じている */
var orch = new Orchestrator();
send orch.start();
```

■Mac AIPL: Worker (now で同期戻値を返すスタイル)　examples/nqueens_distributed_mac.abcl:15-49

```
class Worker {
  function safe(cols: any, row: int, c: int) -> int {
    var i : int = 0;
    while (i < row) do {
      var p : int = array_get(cols, i);
      if (p == c)            { return 0; }
      if (p - c == row - i)  { return 0; }
      if (c - p == row - i)  { return 0; }
      i = i + 1;
    }
    return 1;
  }

  function try_col(cols: any, n: int, row: int, c: int) -> int {
    if (c >= n) { return 0; }
    var here : int = 0;
    if (safe(cols, row, c)) {
      var cols2 : any = array_set(cols, row, c);
      here = nq_count(cols2, n, row + 1);
    }
    return here + try_col(cols, n, row, c + 1);
  }

  function nq_count(cols: any, n: int, row: int) -> int {
    if (row == n) { return 1; }
    return try_col(cols, n, row, 0);
  }

  /* now の戻値として直接 int を返す */
  method solve(n: int, first_col: int) -> int {
    var cols[12];
    var cols2 : any = array_set(cols, 0, first_col);
    return nq_count(cols2, n, 1);
  }
}
```

■Mac AIPL: Orchestrator (send / now / future の 3 種類を実演)　examples/nqueens_distributed_mac.abcl:51-108

```
class Orchestrator {
  /* 同期 now で分散 (シンプル、戻値直結) */
  function run_now(n: int) -> int {
    var sum : int = 0;
    var c   : int = 0;
    while (c < n) do {
      var w   : any = new Worker();
      var sub : int = now w.solve(n, c);
      sum = sum + sub;
      c   = c + 1;
    }
    return sum;
  }

  /* future で全 first_col を並行発射 → 後でまとめて await
     (Mac OCaml AIPL ランタイムが提供する future handle 型は any) */
  function run_future(n: int) -> int {
    var sum : int = 0;
    var c   : int = 0;
    var fs  : any = array_new(n);
    /* 発射 */
    while (c < n) do {
      var w : any = new Worker();
      fs = array_set(fs, c, future w.solve(n, c));
      c  = c + 1;
    }
    /* 回収 */
    c = 0;
    while (c < n) do {
      sum = sum + await(array_get(fs, c));
      c   = c + 1;
    }
    return sum;
  }

  /* fire-and-forget (集計したくない場合のみ意味がある — ここではログ用) */
  method warmup(n: int) {
    var w : any = new Worker();
    send w.solve(n, 0);   /* 戻値破棄、副作用 (キャッシュ温め) のみ */
  }

  method init(tag: any) {
    print("[mac] === " + tag + " ===");

    var t0  : int = now_ms();
    var s8a : int = run_now(8);
    print("[mac] now    N=8 sol=" + s8a + " elapsed_ms=" + (now_ms() - t0));

    var t1  : int = now_ms();
    var s8b : int = run_future(8);
    print("[mac] future N=8 sol=" + s8b + " elapsed_ms=" + (now_ms() - t1));

    send self.warmup(4);   /* send デモ (return 値なし) */
    print("[mac] === done ===");
  }
}
```

```

var orch = new Orchestrator("nqueens distributed");
```

■Mac 側 N-Queens 参照実装 (Python、AIPL ランタイム非依存の baseline)　tools/nqueens_bench.py:37-61

```
            return 1
        total = 0
        for c in range(n):
            ok = True
            for r in range(row):
                if cols[r] == c or abs(cols[r] - c) == row - r:
                    ok = False
                    break
            if ok:
                cols[row] = c
                total += recurse(row + 1)
        return total
    return recurse(1)


def nq_total(n: int) -> int:
    return sum(nq_partial(n, c) for c in range(n))


# ----- Pi 3 HTTP helpers -----------------------------------------------

def http_get(path: str, timeout: float = 30.0) -> str:
    with urllib.request.urlopen(PI3_BASE + path, timeout=timeout) as r:
        return r.read().decode("utf-8", errors="replace")
```

■Mac 側 Pi 3 並列ポーリング (Python、計測駆動)　tools/nqueens_bench.py:153-184

```
    return count, int((time.perf_counter() - t0) * 1000)


def mode_pi3(n: int) -> tuple[int, int]:
    return pi3_nq_run(n)


def mode_split(n: int) -> tuple[int, int]:
    """Mac runs cols [0..n/2), Pi 3 runs cols [n/2..n) concurrently."""
    half = n // 2
    mac_cols = list(range(0, half))
    pi3_cols = list(range(half, n))

    def mac_part() -> int:
        return sum(nq_partial(n, c) for c in mac_cols)

    t0 = time.perf_counter()
    with concurrent.futures.ThreadPoolExecutor(max_workers=2) as pool:
        f_mac = pool.submit(mac_part)
        f_pi3 = pool.submit(pi3_nq_run, n, pi3_cols)
        mac_count = f_mac.result()
        pi3_count, _ = f_pi3.result()
    elapsed_ms = int((time.perf_counter() - t0) * 1000)
    return mac_count + pi3_count, elapsed_ms
```

```
178
179 # ----- main ------------------------------------------------------------
180
181 MODES = {
182     "mac":   mode_mac,
183     "pi3":   mode_pi3,
184     "split": mode_split,
```

6 再現手順

```
1  # 1. Pi 3 カーネルをビルドして焼く
2  cd ~/projects/xinu-raz/xinu/compile && make PLATFORM=arm-rpi3
3
4  # 2. Pi 3 起動後 (SD swap または /kexec)、アクターを明示的に起動:
5  curl -X POST http://192.168.3.50:8080/api/loadbal/init
6  curl -X POST http://192.168.3.50:8080/api/gc-actor/init
7
8  # 3. ベンチマーク実行 + レポート生成
9  cd ~/projects/xinu-raz/xinu/tools
10 ./nqueens_bench.py --sizes 8, 9, 10 --pretty --out results.json
11 ./build_report.py results.json --out report.tex --pdf report.pdf
```

7 生データ (JSON)

```
1  {
2    "host": "http://192.168.3.50:8080",
3    "repeat": 3,
4    "results": [
5      {
6        "mode": "mac",
7        "n": 8,
8        "count": 92,
9        "samples_ms": [
10         4,
11         3,
12         3
13       ],
14       "median_ms": 3
15     },
16     {
17       "mode": "pi3",
18       "n": 8,
19       "count": 92,
20       "samples_ms": [
21         20044,
22         10135,
23         10138
24       ],
25       "median_ms": 10138
26     },
27     {
28       "mode": "split",
29       "n": 8,
30       "count": 92,
31       "samples_ms": [
```

```
        9961,
        10238,
        9969
      ],
      "median_ms": 9969
    },
    {
      "mode": "mac",
      "n": 9,
      "count": 352,
      "samples_ms": [
        9,
        9,
        9
      ],
      "median_ms": 9
    },
    {
      "mode": "pi3",
      "n": 9,
      "count": 352,
      "samples_ms": [
        10210,
        20246,
        15221
      ],
      "median_ms": 15221
    },
    {
      "mode": "split",
      "n": 9,
      "count": 352,
      "samples_ms": [
        10336,
        10076,
        10468
      ],
      "median_ms": 10336
    },
    {
      "mode": "mac",
      "n": 10,
      "count": 724,
      "samples_ms": [
        88,
        48,
        49
      ],
      "median_ms": 49
    }
  ],
  "_note": "pi3/split for N=10..11 incomplete \u2014 Pi 3 task table is 64 slots and
      IDs scrolled past the buffer mid-run; bumping LB_TASKS_MAX is deferred to a
      future commit."
}
```