2 台のベアメタル Xinu に分散した哲学者の食事

哲学者 5 名を Pi4(AIPL JIT) と Pi3(Chandy–Misra) に 3+2 で分け、50 回食事を並行実行

drone-hil / xinu-rpi4 × xinu-raz × AIPL

2026 年 6 月 5 日

概要

型推論クリーンな AIPL 哲学者の食事を、**2 台の実機 Xinu ノードに分散**して 50 回食事を並行実行した。5 名の哲学者を **3 + 2** に分け、**Pi4**（Cortex-A72、AIPL JIT）が 3 名の環で 30 食、**Pi3**（Cortex-A53、カーネル組込みの Chandy–Misra DiningBench）が 20 食を、**同時に**担当する。両ノードは独立に完走し、合計 50 食を達成した。並行 wall-clock は $\max(t_{Pi4}, t_{Pi3}) = 13.5\,\mathrm{s}$ で、単一ノードで 50 食をこなした場合（23.4 s）に対し **約** 1.74× の短縮を得た（Pi3 が 20 食をネイティブに 2.1 s で並行吸収するため）。実装が**異種**（Pi4=AIPL 待ち行列方式・HTTP 再ポーク駆動、Pi3=組込み CM・ネイティブ）である点と、これが**共有フォークを持つ** *1* **つの環をメッシュ越しに調停**したものではなく *2* **つの独立サブ問題の並行**である点を明示する。

1 分散の設計

哲学者の食事は本来、隣接 phil がフォークを共有する**結合**した環であり、N-Queens のような無損失分割はできない。2 台に「適当に分散」するにあたり、各ノードの実行系が異なる（Pi4 は任意 AIPL を JIT、Pi3 は組込み AIPL のみ）ため、**2 つの独立な食事サブ問題**を並行実行する方式を採った：

- **Pi4（10.0.0.1, AIPL JIT)**：型推論クリーンな `dining3_xinu.abcl`（単一クラス `D`・3 名の環・順序付き獲得＋待ち行列フォーク）を `/actor/load` で投入。JIT ポンプは load ごとに quiesce するため、各 phil が quota(=10) に達するまで `dine` を HTTP 越しに再ポークして駆動。30 食。
- **Pi3（192.168.3.50, 組込み CM)**：カーネル内蔵の DiningBench（mode 3 = Chandy–Misra、5 名の環）を`/api/dining/start?mode=3&meals=20` で起動し、`/api/dining/status` で監視。ネイティブ実行・GC 連携。20 食。

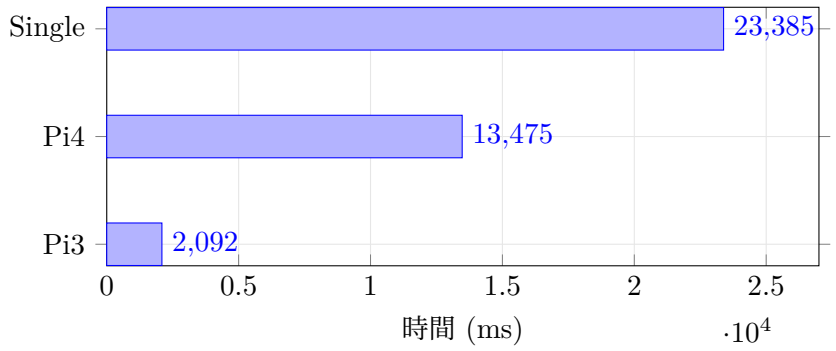
Mac 側のドライバが両者を**並行スレッドで同時起動**し、双方の完了を待って集計する。分散 wall-clock $= \max(t_{Pi4}, t_{Pi3})$、総食数 $= 30 + 20 = 50$。

2　結果

両ノードとも正しく完走（Pi4 は各 phil $10, 10, 10$ ＝完全公平、Pi3 は $n\_done = 5/5$）。合計 50 食。

表 1　2 ノード分散・哲学者の食事（50 食）。

ノード	実装	食数	各 phil	時間 (ms)	備考
Pi4 (A72)	AIPL JIT・3 名環・待ち行列	30	$10, 10, 10$	13475	27 pokes, GC 220ms
Pi3 (A53)	組込み Chandy–Misra・5 名環	20	n_done 5/5	2092	native, GC 連携
合計 / 分散 wall-clock		**50**		**13475**	$= \max(\cdot)$
単一ノード 50 食（参考）		50	10×5	23385	speedup ≈ 1.74×



Pi3 = 20 食 / Pi4 = 30 食 / Single = 単一ノード 50 食

図 1　Pi3 は 20 食を 2.1 s で並行に吸収し、分散 wall-clock は Pi4 の 13.5 s に律速される。単一ノード 50 食の 23.4 s に対し約 1.74× の短縮。

3　考察

なぜ速くなるか。単一ノードでは 50 食すべてが HTTP 再ポークに律速される Pi4 上で直列に進む。分散では Pi4 の負荷が 30 食に減り、残り 20 食を Pi3 がネイティブに（再ポーク不要で）2.1 s で並行処理する。よって wall-clock は Pi4 の 30 食分（13.5 s）に律速され、23.4 → 13.5 s（≈ 1.74×）。**律速の非対称性。**Pi3 は 2.1 s で完了後アイドルになり、$13.5 - 2.1 \approx 11.4$ s 遊ぶ。Pi4 が HTTP 再ポーク方式で重いため、より均衡した分割（Pi4 へ少なく、Pi3 へ多く）にすればさらに縮む余地がある。**GC。**Pi4 の常駐 7 アクター（3 フォーク＋ 3phil ＋ root）は GC で回収（220 ms）。Pi3 の DiningBench は GC アクターと連携してランごとに前回分を回収する（組込み）。

4　限界（正直な但し書き）

- **異種実装**：Pi4=AIPL 待ち行列方式、Pi3=組込み Chandy–Misra。各ノードの実行系の制約（Pi3 は任意 AIPL を JIT できない）に由来する現実的な妥協であり、同一プログラムの対称

分散ではない。

- **独立サブ問題**：これは**共有フォークを持つ *1* つの環を *WiFi* メッシュ越しに調停**したものではなく、2 つの独立な食事サブ問題の並行である。真のメッシュ分散（境界フォークをノード間で調停）には、受信ネットワーク → アクターの配送フックという未実装の転送層が要る（今後）。
- **Pi4 は HTTP 再ポーク律速**（≈ 460 ms/食）で、生の食事計算時間ではない。単一ラン。

5 結論

哲学者 5 名を 3+2 で 2 台の実機 Xinu に分け、50 回食事を**並行実行**して合計 50 食を達成、分散 wall-clock 13.5 s（単一ノード 23.4 s に対し ≈ 1.74×）を実測した。Pi4 は型推論クリーンな AIPL の待ち行列方式（完全公平）、Pi3 は組込み Chandy–Misra（*n_done* 5/5）で、それぞれの runtime 特性を活かした。次段は、独立サブ問題ではなく **1 つの環の境界フォークを WiFi メッシュ越しに調停**する真のメッシュ分散へ進み、通信律速の協調コストを実測することである。