

HyGTA2専用シミュレータ用の ログ解析ツールの使い方

コンピュータアーキテクチャ研究グループ B4

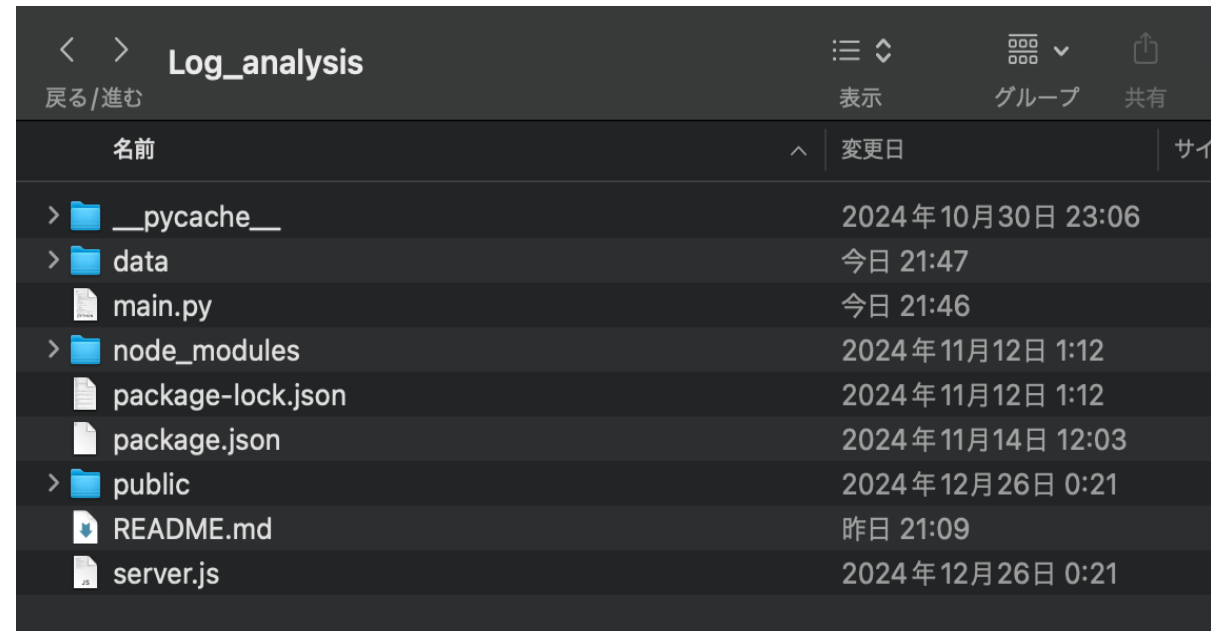
萱沼 颯

インストールが必要なもの

- Python (v3.12.2)
- Node.js (v22.9.0)

下準備（完全に初めて使う場合）

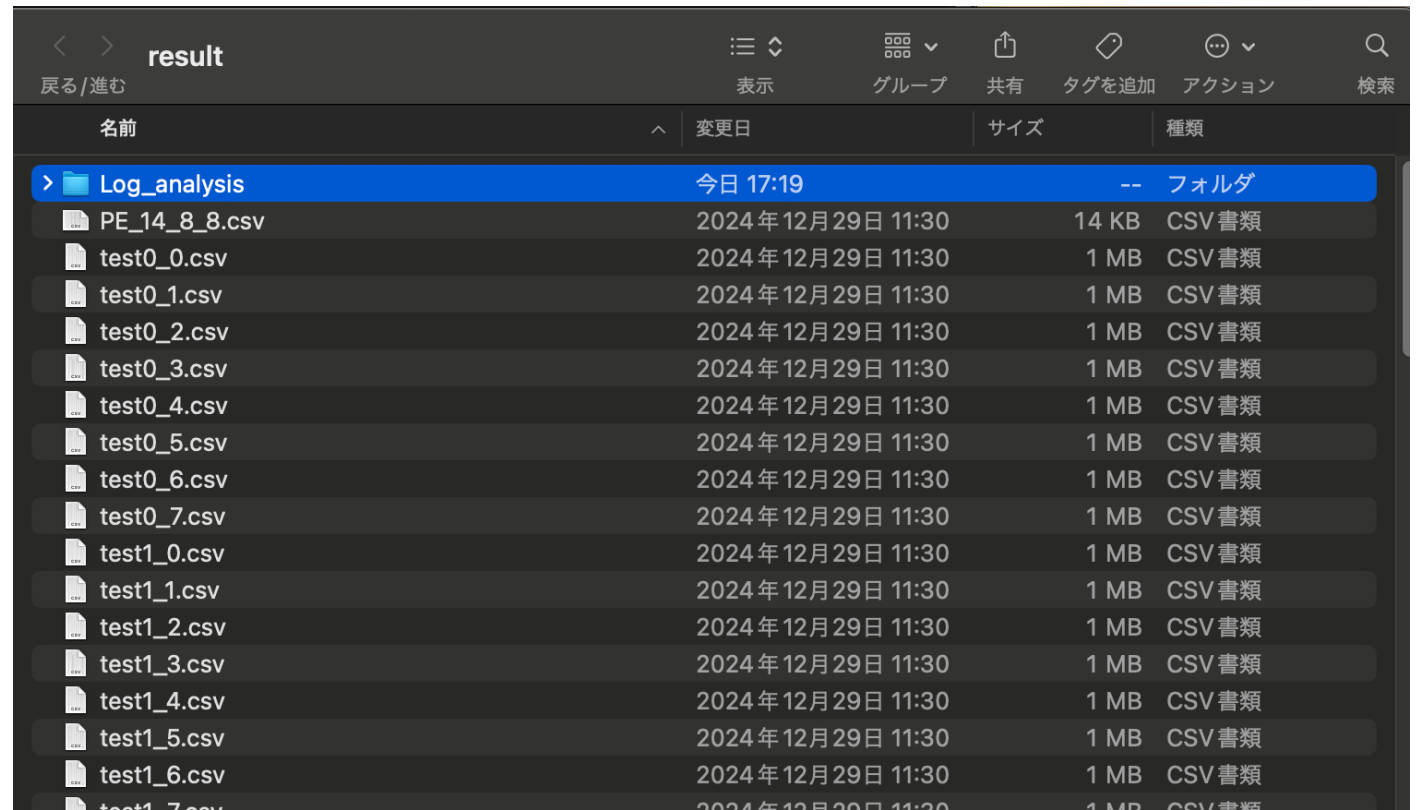
- Node.jsをインストール後,
GitHub(https://github.com/So-213/Log_analysis)
からログ解析ツール (Log_analysis)をダウンロード
- 空の“data”ディレクトリを作成
- Log_analysisルートディレクトリにて,
ターミナルからnpm installを実行
 - 依存パッケージを一括
インストールする（必須）
 - 実行後に“npm_modules”
フォルダが生成される



Log_analysis			表示	グループ	共有
戻る/進む					
名前	変更日	サイ			
> __pycache__	2024年10月30日 23:06				
> data	今日 21:47				
main.py	今日 21:46				
> node_modules	2024年11月12日 1:12				
package-lock.json	2024年11月12日 1:12				
package.json	2024年11月14日 12:03				
> public	2024年12月26日 0:21				
README.md	昨日 21:09				
server.js	2024年12月26日 0:21				

使い方1-1（解析から行う場合はここから）

- ログ解析ツールをログファイル群と同ディレクトリに置く



使い方1-2

- ターミナルからpython3 main.pyを実行
 - ログファイル群の解析をするPythonスクリプト
 - 各標準入力項目については次ページ参照

```
kayanuma@makku ~ % cd /Users/kayanuma/Desktop/result/Log_analysis
kayanuma@makku Log_analysis % python3 main.py
scale:14
PE行列の行(列)数:8
モジュール数:9
アニメーションのためのファイルを作成しますか？(T/F):T
データをとるモジュールindex(1~):4
各深さのサイクル数をカンマ区切りで入力してください(例: 25,389,2668,841,34) (わからない場合はそのままEnter): 63,2803,11242,1532,32
分析中...
完了しました
kayanuma@makku Log_analysis %
```

Python3 main.py時の標準入力について1

- モジュール数 . . . PE内部のモジュール数のこと .
csvファイルでは列数のこと (現状9) .
 - データをとるモジュールindex . . . 稼働率平均値/中央値/分散
をどのモジュールでとるか (迷ったら4) .
 - 1 : local frontier bitmap
 - 2 : working frontier
 - 3 : extract vertices issue receiver
 - 4 : traversal
 - 5 : filter pred data issue
 - 6 : unvisit
 - 7 : filter pred data issue receiver
 - 8 : Allgather
 - 9 : AlltoAll
- | 0 | Send | Wait | Wait | Wait | Wait | Wait | Wait | Wait | Wait |
|---|------|-------------------|---------|----------|------|------|------|---------------|-------------|
| 1 | Wait | Receive | Wait | Wait | Wait | Wait | Wait | Wait | Wait |
| 2 | Wait | Send Allgather | Wait | Wait | Wait | Wait | Wait | Receive | Wait |
| 3 | Wait | Wait | Wait | Wait | Wait | Wait | Wait | Send Frontier | Wait |
| 4 | Wait | Receive Alltather | Wait | Wait | Wait | Wait | Wait | Wait | Wait |
| 5 | Wait | Send -1 | Wait | Wait | Wait | Wait | Wait | Wait | Wait |
| 6 | Wait | Wait | Receive | Wait | Wait | Wait | Wait | Wait | Wait |
| 7 | Wait | Wait | Wait | Send End | Wait | Wait | Wait | Wait | Recieve End |

[illegible]

Python3 main.py時の標準入力について2

- 各深さのサイクル数・・・
専用シミュレータのターミナルから情報取得.

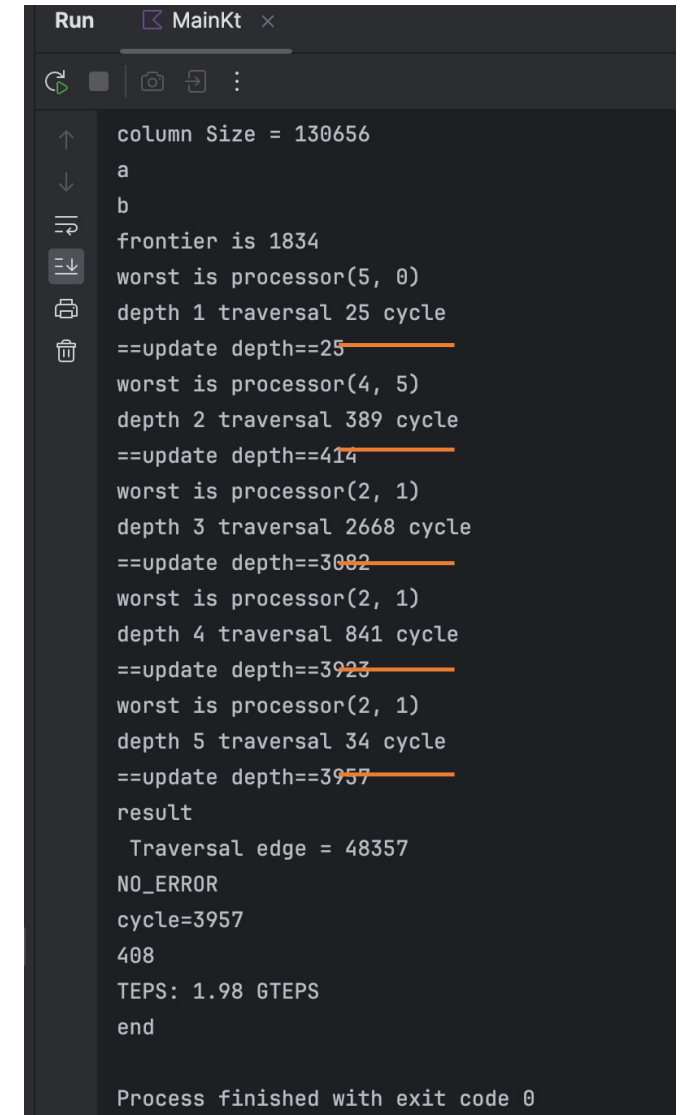
例えばPEアレイ8×8の64の場合
(PE数とグラフスケールによって変わる)

Scale10
18,97,706,262,23

Scale12
25,389,2668,841,34

Scale14
63,2803,11242,1532,32

HyGTA2専用
シミュレータ→






```
Run MainKt x
column Size = 130656
a
b
frontier is 1834
worst is processor(5, 0)
depth 1 traversal 25 cycle
==update depth==25
worst is processor(4, 5)
depth 2 traversal 389 cycle
==update depth==414
worst is processor(2, 1)
depth 3 traversal 2668 cycle
==update depth==3082
worst is processor(2, 1)
depth 4 traversal 841 cycle
==update depth==3923
worst is processor(2, 1)
depth 5 traversal 34 cycle
==update depth==3957
result
  Traversal edge = 48357
NO_ERROR
cycle=3957
408
TEPS: 1.98 6TEPS
end

Process finished with exit code 0
```

使い方1-3

- うまく実行できるとdataフォルダに4ファイル（3ファイル）生成される
 - each_module_oc_rate.json・・・各PE内部の各モジュールの稼働率を格納
 - frames.json・・・PE挙動アニメーション用（作成しないモードでは生成されない）
 - info.json・・・色々な情報を格納. 特に指定したモジュールの稼働率平均値/中央値/分散が
average/median/varianceに記載.
 - layers.json・・・各PEの指定したモジュールに対して, 各深さでの稼働率を格納

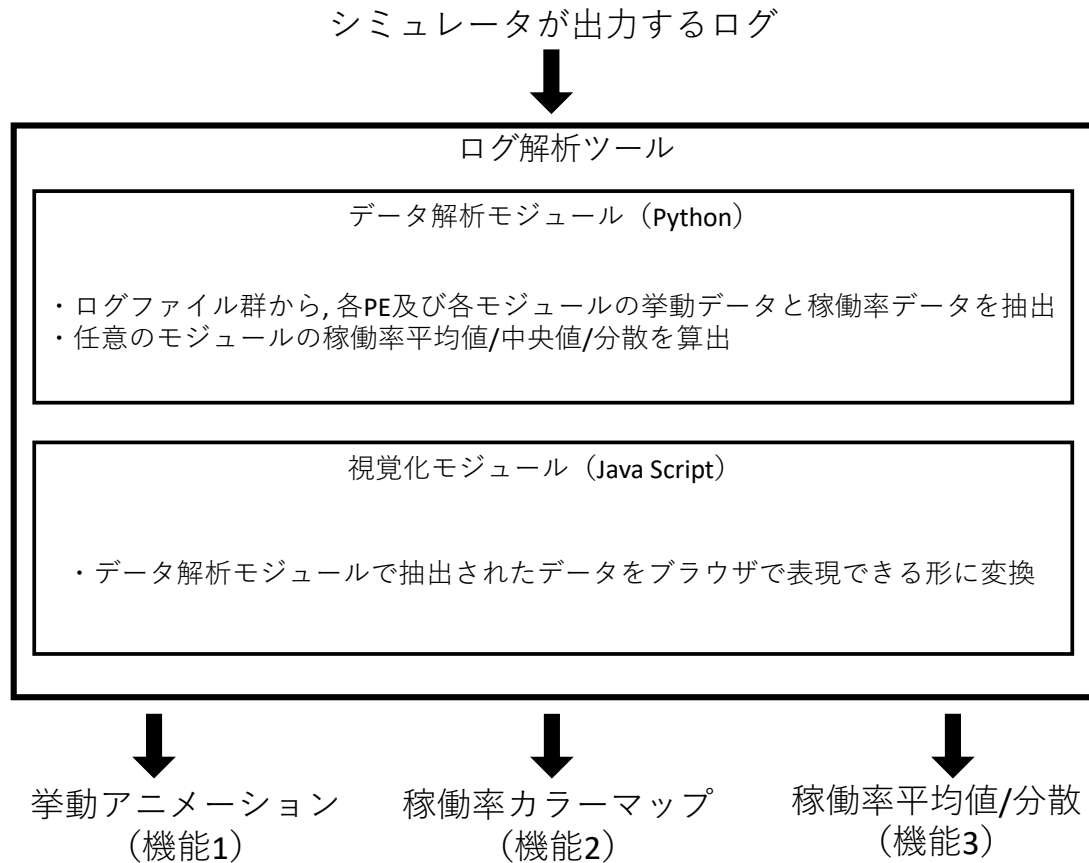
< > data		⋮ ⬆
戻る/進む		表示
名前	⌵	変更日
 each_module_oc_rate.json		今日 21:47
 info.json		今日 21:47
 layers.json		今日 21:47

使い方2-1（閲覧のみ行う場合はここから）

- ターミナルからnpm startを実行
 - ローカルホストでサーバを設置し、ブラウザからアクセスしてくれる。



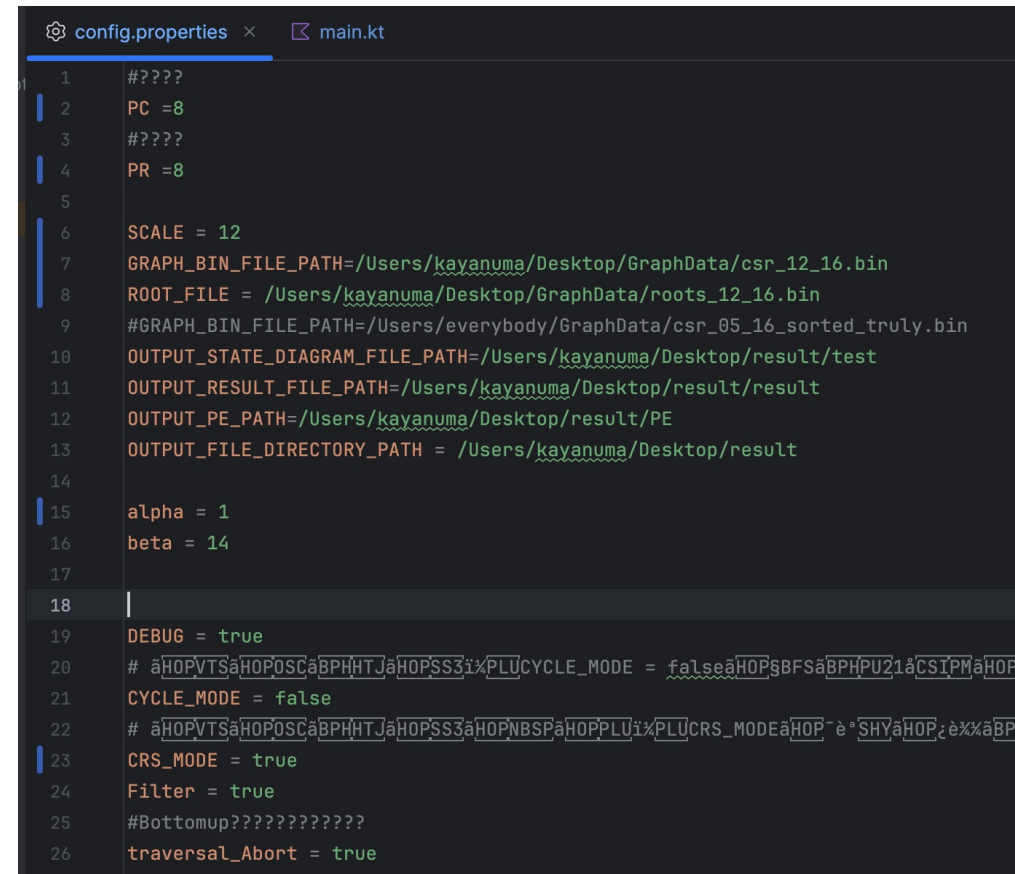
ツール構成



卒論

- Scale12の評価・・・性能分析.xlsxのシート3 (scale12_CSR_original)

- Teams上のOriginalグラフデータ (Graph500OriginalGraphData) のグラフスケール12 (csr_12_16.binとroots_12_16.bin) を専用シミュレータに入力させた.
- 専用シミュレータでは, config.propertiesのパラメータを調整して評価した.
 - CRS_MODE=true
 - CYCLE_MODE=falseなのでおそらくrootは最初の1つのみしか検証していない



```
config.properties x main.kt
1  #????
2  PC =8
3  #????
4  PR =8
5
6  SCALE = 12
7  GRAPH_BIN_FILE_PATH=/Users/kayanuma/Desktop/GraphData/csr_12_16.bin
8  ROOT_FILE = /Users/kayanuma/Desktop/GraphData/roots_12_16.bin
9  #GRAPH_BIN_FILE_PATH=/Users/everybody/GraphData/csr_05_16_sorted_truly.bin
10 OUTPUT_STATE_DIAGRAM_FILE_PATH=/Users/kayanuma/Desktop/result/test
11 OUTPUT_RESULT_FILE_PATH=/Users/kayanuma/Desktop/result/result
12 OUTPUT_PE_PATH=/Users/kayanuma/Desktop/result/PE
13 OUTPUT_FILE_DIRECTORY_PATH = /Users/kayanuma/Desktop/result
14
15 alpha = 1
16 beta = 14
17
18
19 DEBUG = true
20 # äHOPVTSäHOPOSCäBPHHTJäHOPSS3ixPLUCYCLE_MODE = falseäHOPSBFSäBPHPU21äCSIPMäHOPN
21 CYCLE_MODE = false
22 # äHOPVTSäHOPOSCäBPHHTJäHOPSS3äHOPNBSPaHOPPLÜixPLUCRS_MODEäHOP~è"SHYäHOPçè%äBPH
23 CRS_MODE = true
24 Filter = true
25 #Bottomup????????????
26 traversal_Abort = true
```

- Scale10~16の評価・・・性能分析.xlsxのシート4 (scales_CSR_original)
 - こちらもOriginalグラフデータかつCSRモードtrueで評価.

課題

- 性能分析.xlsxのシート4 (scales_CSR_original) 下部のこれ
 - こちらもOriginalグラフデータ (Graph500OriginalGraphData) かつ CSRモードtrueで評価

