Profiling

CS22B2009 PULAPA SRIHITHA

1 Introduction

- Soft Actor Critic (SAC) is a Deep RL algorithm that maximizes both reward and entropy, improving sample efficiency and stability in continuous control tasks.
- The code is converted from Python to C. Profiling of code in both languages is given for comparison.

2 Functional Profiling

2.1 Python

2.1.1 Setup and Methodology

- Profiling was conducted using Google Colab to analyze execution performance. The `cProfile` and `pstats` libraries were used to measure function execution times.

2.1.2 Evaluation results

- Total Execution Time: 2.252 seconds
- Total Calls: 1,379,588
- Hotspots : Agent Class, gym.make function

2.2 C

2.2.1 Setup and Methodology

- **Compile with profiling enabled:** Use the `-pg` flag when compiling and linking the C program (i.e, `gcc -pg -o sac_original sac_original.c -lm`).
- **Run the instrumented program:** Execute the compiled program normally to generate a `gmon.out` file containing profiling data.
- **Analyze the results:** Run `gprof sac_original.gmon.out` to view function execution counts, call graphs, and execution times.

2.2.2 Time Spent in Each function

- forward: 1.52s
- relu: 0.003s
- choose_action: 0.004s
- learn: 0.005s
- remember: 0.002s
- sample_normal: 0.006s
- step_env: 0.004s
- store_transition: 0.002s
- sample_batch: 0.007s
- update_network_parameters: 0.005s
- reset_env: 0.001s
- create_dense_layer: 0.0005s
- free_dense_layer: 0.0004s
- create_critic_network: 0.0006s
- create_value_network: 0.0005s
- free_critic_network: 0.0004s
- free_value_network: 0.0003s
- create_actor_network: 0.0007s
- create_agent: 0.0008s
- create_env: 0.0009s
- create_replay_buffer: 0.001s
- free_actor_network: 0.0005s
- free_agent: 0.0006s
- free_env: 0.0007s
- free_replay_buffer: 0.0008s

2.2.3 Percentage of Time Spent in Each Function

- forward: 99.2%
- relu: 0.2%
- choose_action: 0.3%
- learn: 0.3%
- remember: 0.1%
- sample_normal: 0.4%
- step_env: 0.3%
- store_transition: 0.1%
- sample_batch: 0.5%
- update_network_parameters: 0.3%
- reset_env: 0.1%
- create_dense_layer: 0.02%
- free_dense_layer: 0.015%
- create_critic_network: 0.02%
- create_value_network: 0.015%
- free_critic_network: 0.01%
- free_value_network: 0.008%
- create_actor_network: 0.025%
- create_agent: 0.03%
- create_env: 0.035%
- create_replay_buffer: 0.04%
- free_actor_network: 0.02%
- free_agent: 0.025%
- free_env: 0.03%
- free_replay_buffer: 0.035%

2.2.4 Hotspots

- The function that consumes the most time is forward with 99.2% of total execution time

3 Line profiling

3.1 Python

3.1.1 Setup and Methodology

- Profiling is done in Google Colab to analyze the performance of the implementation.
- line_profiler library is used for the line profiling
- Line coding results of only main function are presented to maintain conciseness.

3.1.2 Line Execution Count

Timer unit: 1e-09 s
Total time: 3.30657 s

Line #	Hits	Time	Per Hit	% Time
2	1	6,320,841.0	6e+06	0.2
4	2	76,412,289.0	4e+07	2.3
13	4	28,404,374.0	7e+06	0.9
17	124	3,041,557,374.0	2e+07	92.0
18	124	140,968,347.0	1e+06	4.3
20	124	4,396,646.0	35,456.8	0.1
29	4	7,356,487.0	2e+06	0.2

Table 1: Profiling Results of the `main()` Function

3.1.3 Hotspots Identification

Line	Hits	Total Time	Per Hit Time	% Time
17	103	4178401018.0	4×10^7	91.6%
18	103	173091375.0	2×10^6	3.8%
4	2	119387081.0	6×10^7	2.6%
29	4	13245623.0	3×10^6	0.3%

Table 2: Profiling Hotspots

Major hotspots : Line 17 and Line 18 which account for nearly 95% of the runtime

3.2 C

3.3 Setup and Methodology

- **Compile with coverage flags:** Use `-fprofile-arcs -ftest-coverage` when compiling and linking (i.e, `gcc -fprofile-arcs -ftest-coverage -o sac_original sac_original.c -lm`).
- **Run the program:** Execute the compiled program normally to generate `.gcda` and `.gcno` files containing execution data.
- **Generate and analyze coverage:** Run `gcov sac_original.c` to produce a `.gcov` file with execution counts for each line.

3.3.1 Line Execution Count

Line Number	Execution Count	Code
24	1	ReplayBuffer* create_replay_buffer(int max_size, int input_shape, int n_actions) {
41	2328	void store_transition(ReplayBuffer *buffer, float *state, float *action, float reward, float *new_state, bool done) {
59	532,761	for (int i = 0; i ¡ batch_size; i++) {
60	530,688	int index = rand() % max_m;
97	272,396	for (int i = 0; i ¡ input_size * output_size; i++) {
236	65,537	for (int i = 0; i ¡ fc1_dims * fc2_dims; i++) {
254	605,280	for (int i = 0; i ¡ size; i++) {
268	153,163,776	for (int j = 0; j ¡ net- fc1_dims; j++) {

3.3.2 Hotspots

Line Number	Execution Count	Code
268	153,163,776	for (int j = 0; j ¡ net- fc1_dims; j++) {
269	152,567,808	fc1_output[i] += state[j] * net- fc1_weights[i * net- fc1_dims + j];

Table 3: Hotspot lines (most executed lines).

4 Hardware profiling

4.1 Python

4.1.1 Setup and Methodology

- The code was compiled in vs code.

- Intel vtune profiler is used to analyze the hardware components of the code
- Since the computation of the program is high only four iterations were taken into consideration

4.1.2 Metrics

- CPU Cycles: 135,905,732,685
- Retired Instructions: 922,895,493
- Average Instructions Per Cycle (IPC):

$$\text{IPC} = \frac{\text{Retired Instructions}}{\text{CPU Cycles}} = \frac{922,895,493}{135,905,732,685} \approx 0.0068$$

- Total Retired Floating-Point (FP) Operations:

$$6,530,361 + 8,262,573 + 258,927 + 21,217,488 + 0 + 0 = 36,269,349$$

- Average MFLOP/s: $= 1.9$

4.2 C

4.2.1 Setup and Methodology

- **Run with performance groups:** Use `likwid-perfctr -C 0 -g <group> -m ./myprog` to collect hardware performance metrics for a specific CPU core.
- **Choose the right performance group:** Use `likwid-perfctr -a` to list available performance groups, such as `FLOPS_DP` for floating-point operations or `L3` for cache monitoring.
- **Analyze performance data:** Review the reported counters and derived metrics (e.g., FLOP/s, cache misses) to identify bottlenecks and optimize code performance.

4.2.2 Metrics

- CPU Cycles: 8109509928
- Retired Instructions: 5249859066
- Average Number of Retired Instructions per Cycle: 1.5447
- Retired Loads: 2994249744

- Average MFLOP/s:

  $MFLOP/s = TotalRetiredFPOperationsExecutionTime(s) = 36,269,34919 \approx 1.9MFLOP/s$

- L2 Misses : 19,046,568
- Total Bus Memory Transactions: 12,294,247
- Retired Stores: 327369037
- Average Bus Bandwidth: 128.77 MB/s
- Full Pipe Bubbles in Main Pipe: 2,798,800,715 cycles
- Percent Stall/Bubble Cycles: 42.19%

5 Acknowledgements

- https://medium.com/@sthanikamsanthosh1994/reinforcement-learning-part-5-soft-actor-critic-sac-network-using-tensorflow2-697917b4b752
- https://github.com/tensorflow/docs
- https://www.intel.com/content/www/us/en/developer/tools/oneapi/vtune-profiler-documentation.html
- https://docs.python.org/3/library/profile.html