# TOPPERS ASP3 カーネルにおける ARMv7M, ARMv8M 向けのディスパッチャ実装

小森 工

名古屋大学 情報学部 コンピュータ科学科

April 10, 2020

# 本資料の位置づけ

- ASP3 の ARM_M 依存部にはディスパッチの実行速度が低い問題があった
- 本資料ではパフォーマンスを改善した新しいディスパッチャの設計について記述する
- また旧実装から踏襲した点や ASP カーネルでの実装についても参考として付け加えた

# ASP, ASP3 における共通事項

- タスクコンテキストは Thread モード，非タスクコンテキストは Handler モードで実行
  - 割り込み発生時にそのまま Handler モードで ISR を呼び出す方が効率が良い
- Thread モードでは PSP，Handler モードでは MSP を使用
  - 多重割り込み発生時のスタッキングを MSP に行わせることができる
  - スタックを切り替えない場合，全タスクのスタックが最大ネスト分の例外フレームを格納できる必要がある

# ASP と ASP3 の旧実装における相違点

- ASP
  - ディスパッチを Thread モードで行う

    利点　タスクコンテキスト間のディスパッチが高速
    欠点　割り込みの出入り口処理が複雑で，ディスパッチが低速
  - タスク例外を考慮する必要がある
- ASP3 の旧実装
  - ディスパッチを Handler モードで行う

    利点　割り込みによるディスパッチがシンプルで，高速
    欠点　タスクコンテキストからのディスパッチに実行モード切り替えが必要で，低速
  - タスク例外はサポートしない

# PendSV/SVC を使用する場合の前提（旧実装から踏襲）

- PendSV の割込み優先度は最低優先度に設定する
  - ISR からのディスパッチ要求を PendSV で処理し，1 段目の割込みを終了したときにディスパッチャを呼び出すため
- SVC の割込み優先度は最高優先度に設定する
  - CPU ロック状態で SVC を呼び出せるようにするため
  - CPU ロック状態の割込み優先度は（最高優先度-1）に設定する

# PendSV/SVC を使用するための準備（旧実装からほぼ踏襲）

- 割込み優先度の設定
  - PendSV の外部優先度は-1，内部優先度は最低優先度（`INT_IPM(-1)`）
    - `core_initialize()` で `set_exc_int_priority(EXCNO_PENDSV, INT_IPM(-1))`
  - SVC の外部優先度は最高優先度，内部優先度は 0
    - `core_initialize()` で `set_exc_int_priority(EXCNO_SVCALL, 0)`
- 割込み要求ラインの最低外部優先度を-1 とする（旧実装から変更）
  - core_kernel.trb で`$INTPRI_CFGINT_VALID` を-1 まで
  - 優先度が-1 の割り込みは PendSV に待たされるが，まだ受け付けていない扱いであるため仕様を満たす．
- 例外ベクタテーブルに登録
  - core_kernel.trb で `excno == 11` に `svc_handler` を登録
  - core_kernel.trb で `excno == 14` に `pendsv_handler` を登録

# ASP3 の新実装コンセプト

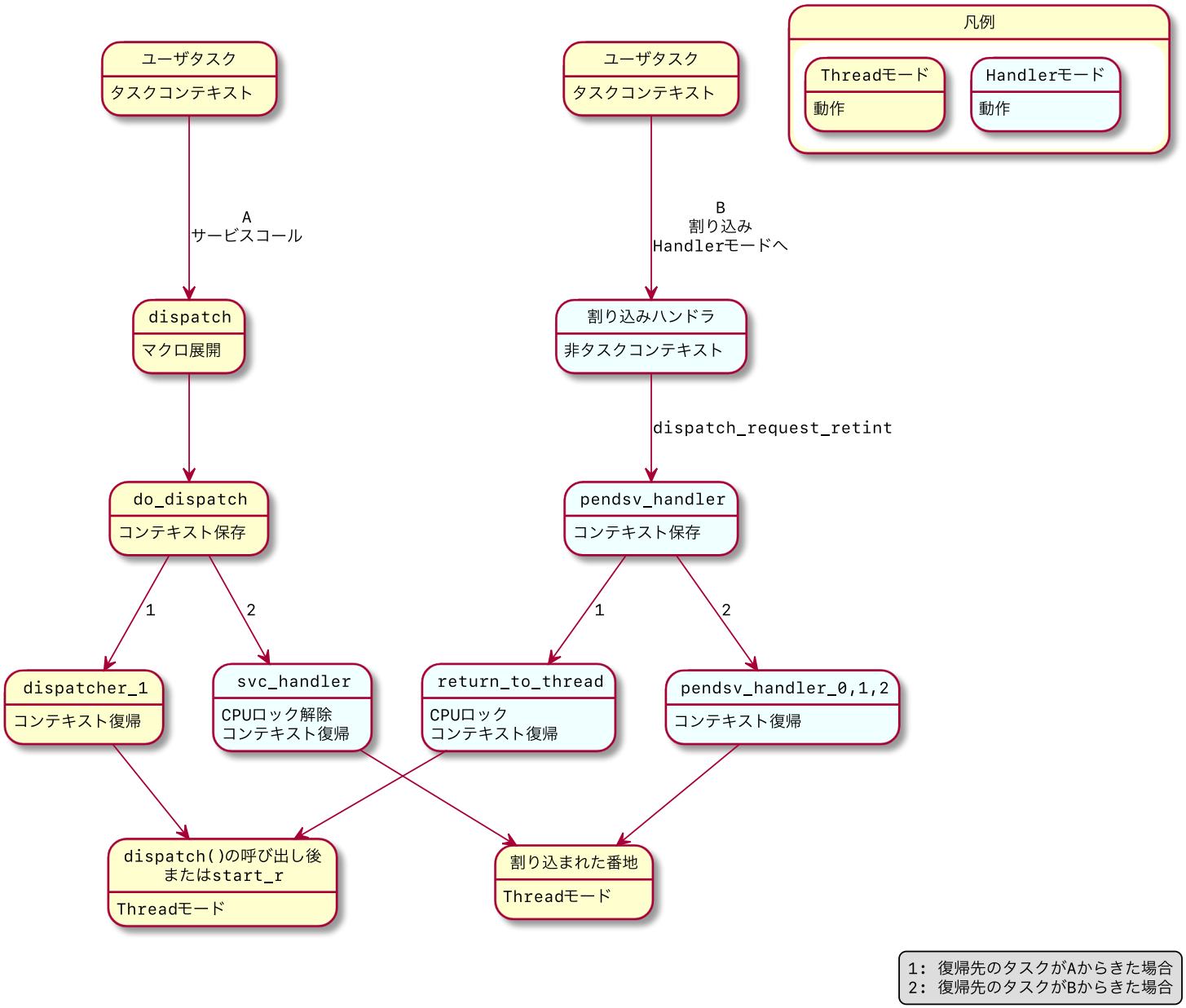
- ASP3 の新実装では，ディスパッチを Thread, Handler の両モードで行う
- 利点
  - 実行モードの切り替えが最小限で済むため，高速化できる（実証済み）
  - `activate_context()` でスタックを使用しない
- 欠点
  - 実装が複雑になるため，シンプルな実装より検証に難がある
  - HRP3 を実装する際に考慮を必要とする

# ハードウェア的制約

ディスパッチャの実装にあたり，以下の点に留意が必要

- 割り込みにより実行権限を奪われたタスクは，Handler モードからの EXC_RETURN 以外で復帰できない
  - 複数ロード/ストア命令や IT ブロックの中途で割り込まれた場合，その復帰が EXC_RETURN 以外で行えないため
  - xPSR を確認して Thread モードから戻ることも可能だが，あまりに実装が複雑になるので考慮しない
- スクラッチレジスタを Thread モードから Handler モードに持ち越すことはできない
  - Tail-chaining により例外ハンドラが呼び出された場合，スクラッチレジスタの内容は不定となるため
- Handler モードからの復帰時にあらかじめ用意された例外フレームを使用することはできない
  - 復帰直後に NMI が発生した場合，例外フレームが積まれてしまうため

# ディスパッチの流れ


凡例
Threadモード
動作
Handlerモード
動作
ユーザタスク
タスクコンテキスト
A
サービスコール
dispatch
マクロ展開
do_dispatch
コンテキスト保存
1
2
dispatcher_1
コンテキスト復帰
svc_handler
CPUロック解除
コンテキスト復帰
dispatch()の呼び出し後
またはstart_r
Threadモード
ユーザタスク
タスクコンテキスト
B
割り込み
Handlerモードへ
割り込みハンドラ
非タスクコンテキスト
dispatch_request_retint
pendsv_handler
コンテキスト保存
1
2
return_to_thread
CPUロック
コンテキスト復帰
pendsv_handler_0,1,2
コンテキスト復帰
割り込まれた番地
Threadモード
1: 復帰先のタスクがAからきた場合
2: 復帰先のタスクがBからきた場合

# ディスパッチの流れ（その他）

- `start_dispatch` からは `do_dispatch` 内のコンテキスト保存以降にジャンプ
- `exit_and_dispatch` は `do_dispatch` 内のコンテキスト保存以降にジャンプ
- アイドルタスクは Thread モードで実行
  - コンテキスト復帰の際にアイドルかを判断し，ジャンプ

# SVC と PendSV の使用

- `do_dispatch` から Handler モードへ移行するために SVC を使用
  - CPU ロック状態で呼び出すため，最高優先度に設定する
- 割り込みの出口でディスパッチを行うため，PendSV を使用
  - 全ての割り込みの終了後に呼び出されるよう，最低優先度に設定する

# その他追加した機能

- Stack point limit register の設定（ARMv8-M 限定）
- 浮動小数点レジスタの Lazy stacking に対応
- `x_get_ipm` はないため，関連する処理を削除

# 実行再開番地をスタックに保存しない理由

- 実行再開番地をスタックに保存し，非スクラッチレジスタとまとめて復帰した方が効率良く思える
- しかし，`activate_context` で `start_r` の番地をスタックに積む必要があるため採用しない

# pendsv_handler で割り込みを禁止するか

pendsv_handler 内では割り込みを禁止した方が良いように思える

- 割り込みを禁止した場合
  - コンテキストを復帰し，タスクを切り替えたと同時にペンディングされた割り込みが入る
  - その割り込みハンドラ内でディスパッチが発生した場合，もう一度 PendSV が発生する
- 割り込みを禁止しない場合
  - 切り替えの途中でも割り込みが発生し，割り込みハンドラが実行される
  - PendSV ハンドラが p_schedtsk を取得した後に割り込みが発生すると，p_schedtsk が変更される場合がある
  - この場合，PendSV から復帰する先のタスクは本来切り替わるべきタスクではない
  - しかし，切り替え直後にもう一度 PendSV が発生するため問題ない

どちらにしても複数回ディスパッチを行うことになるため，割り込み応答性を下げないためには割り込みを禁止しない方が良い．しかし検証性を確保するためカーネル管理の割り込みは禁止することとした．

# return_to_thread 内での r1 の保存

- return_to_thread において，start_r に分岐する場合を考えて r1 をスタックに保存してもよい
- 初期の実装では保存していたが，実装をシンプルにするため，またディスパッチの方が回数が多いと予想できることから保存しないこととした.
- 他の実装案
  - return_to_thread 内で毎回 p_runtsk を保存する
    - この場合，現在の ldr 命令 2 つに対して str 命令が 1 つで済む
    - タスクの起動頻度が高い場合はこちらの方が有利
  - return_to_thread から dispatcher_1 へ分岐する
    - この場合，スタックへ積む変数が多くなる等でパフォーマンスが低下すると考えられる

結局 start_r は特別扱いしないといけないため，オーバーヘッドのない実装は不可能

# Thread-Metric によるパフォーマンス評価結果

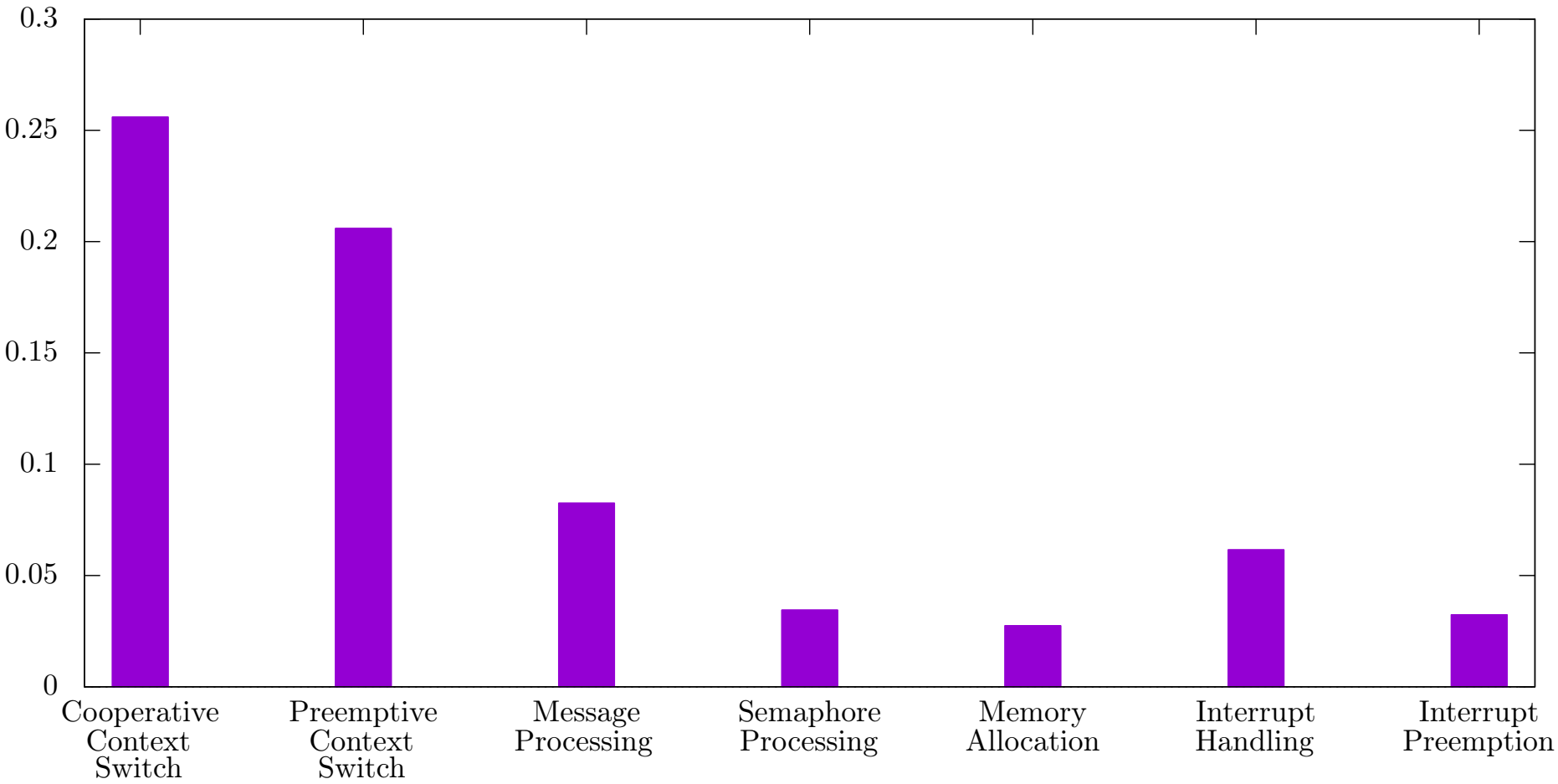
- Rev. 1310 と Rev. 1319 の比較
- ターゲットは nucleo_f401re_gcc
- GCC のバージョンは 8-2019-q3-update
- Thred-Metric[1]の各テストのスコアにより評価した
- 次ページのグラフは性能向上比を以下で計算し，プロットしたもの

$$\text{性能向上比} = \frac{\text{新実装のスコア} - \text{旧実装のスコア}}{\text{旧実装のスコア}}$$

- 全てのテストで性能向上が認められた
- ディスパッチを伴わないテストでも，インライン展開の見直し等による若干の性能向上がある

[1] https://rtos.com/wp-content/uploads/2017/10/Thread_Metric_Article-1.pdf

# Thread-Metricによるパフォーマンス評価結果



# Thread-Metricによるパフォーマンス評価結果

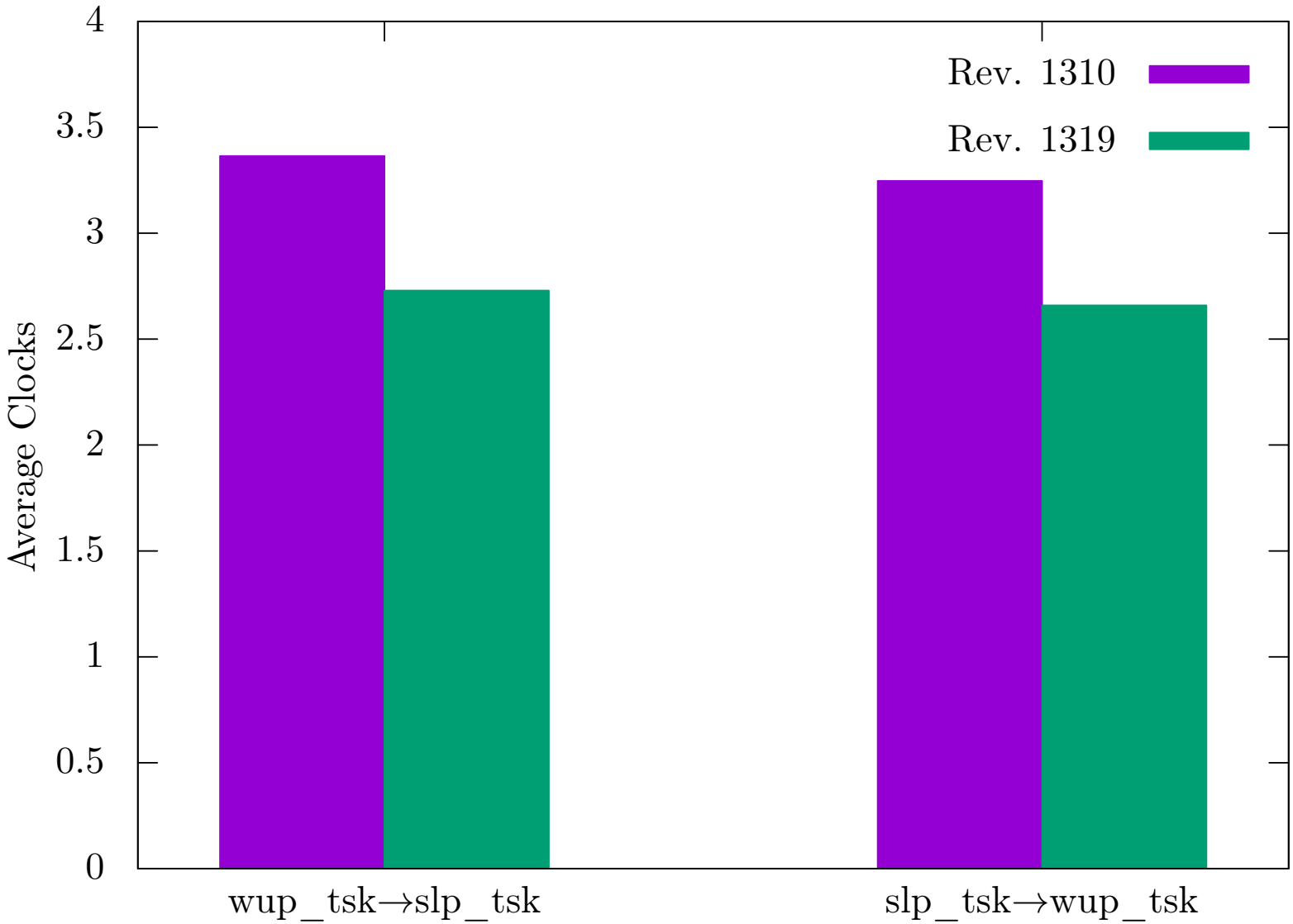
スコアの測定結果

| テスト項目 | 新実装 | 旧実装 |
|---|---|---|
| Cooperative Context Switch | 15,178,571 | 12,085,308 |
| Preemptive Context Switch | 7,374,205 | 6,115,107 |
| Message Processing | 8,415,841 | 7,774,390 |
| Semaphore Processing | 14,655,172 | 14,166,666 |
| Memory Allocation | 10,000,000 | 9,732,824 |
| Interrupt Handling | 9,807,692 | 9,239,131 |
| Interrupt Preemption | 4,344,123 | 4,207,921 |

# perf テストによるパフォーマンス評価結果

- Rev. 1310 と Rev. 1319 の比較
- ターゲットは nucleo_f401re_gcc
- GCC のバージョンは 8-2019-q3-update
- ASP3 付属の perf テストによる評価
- グラフはテスト結果から算出した平均クロック数から perf0 の平均クロック数を減じたもの

# perf1


Average Clocks
4
3.5
3
2.5
2
1.5
1
0.5
0
Rev. 1310
Rev. 1319
wup_tsk→slp_tsk
slp_tsk→wup_tsk

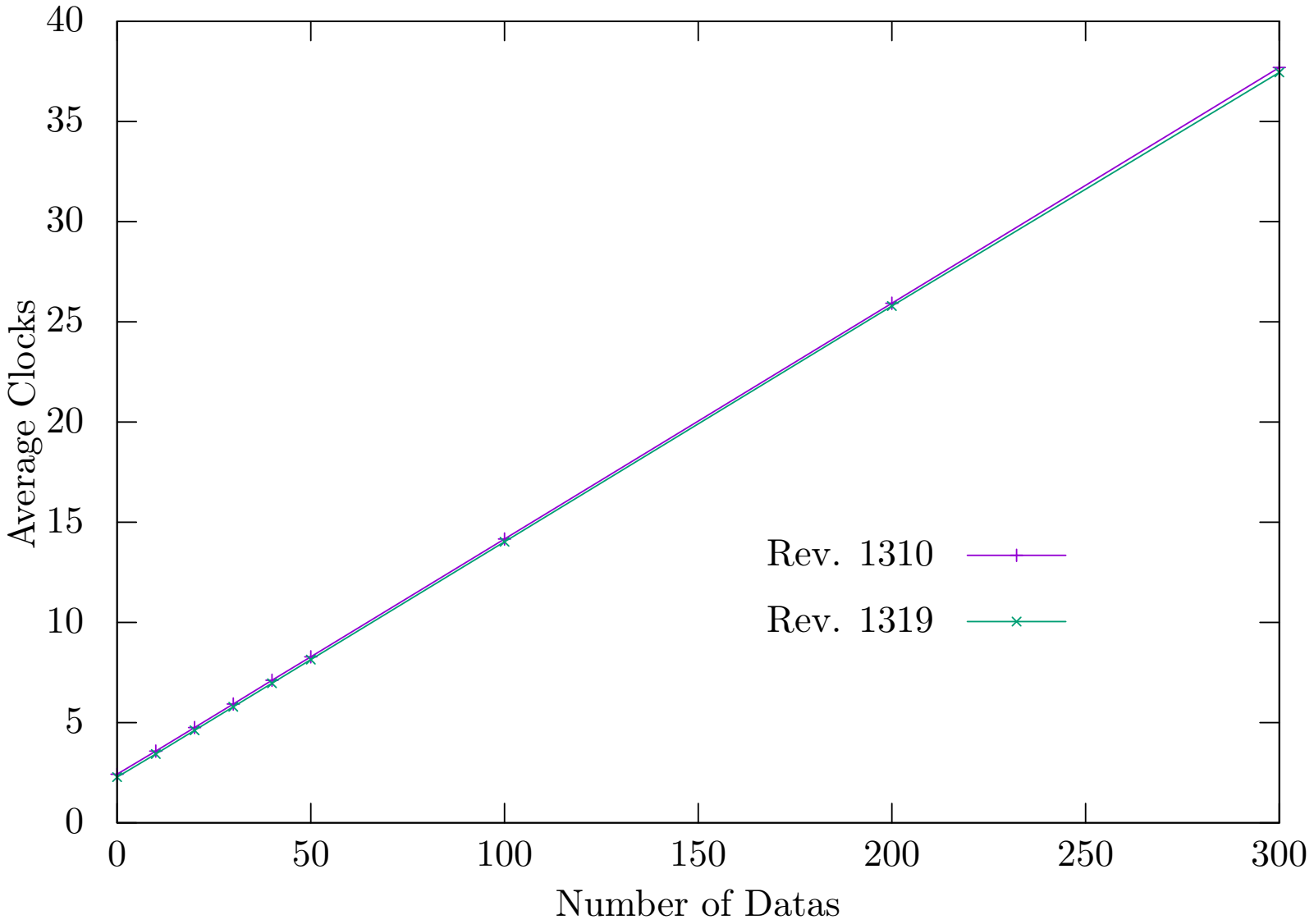
## Rev. 1310

```
1 Execution times of wup_tsk -> slp_tsk
2 4 : 8824
3 5 : 1176
4 Execution times of slp_tsk -> wup_tsk
5 4 : 9999
6 5 : 1
```

## Rev. 1319

```
1 Execution times of wup_tsk -> slp_tsk
2 3 : 5176
3 4 : 4824
4 Execution times of slp_tsk -> wup_tsk
5 3 : 5881
6 4 : 4119
```

# perf2


Average Clocks
Number of Datas
0
5
10
15
20
25
30
35
40
0
50
100
150
200
250
300
Rev. 1310
Rev. 1319

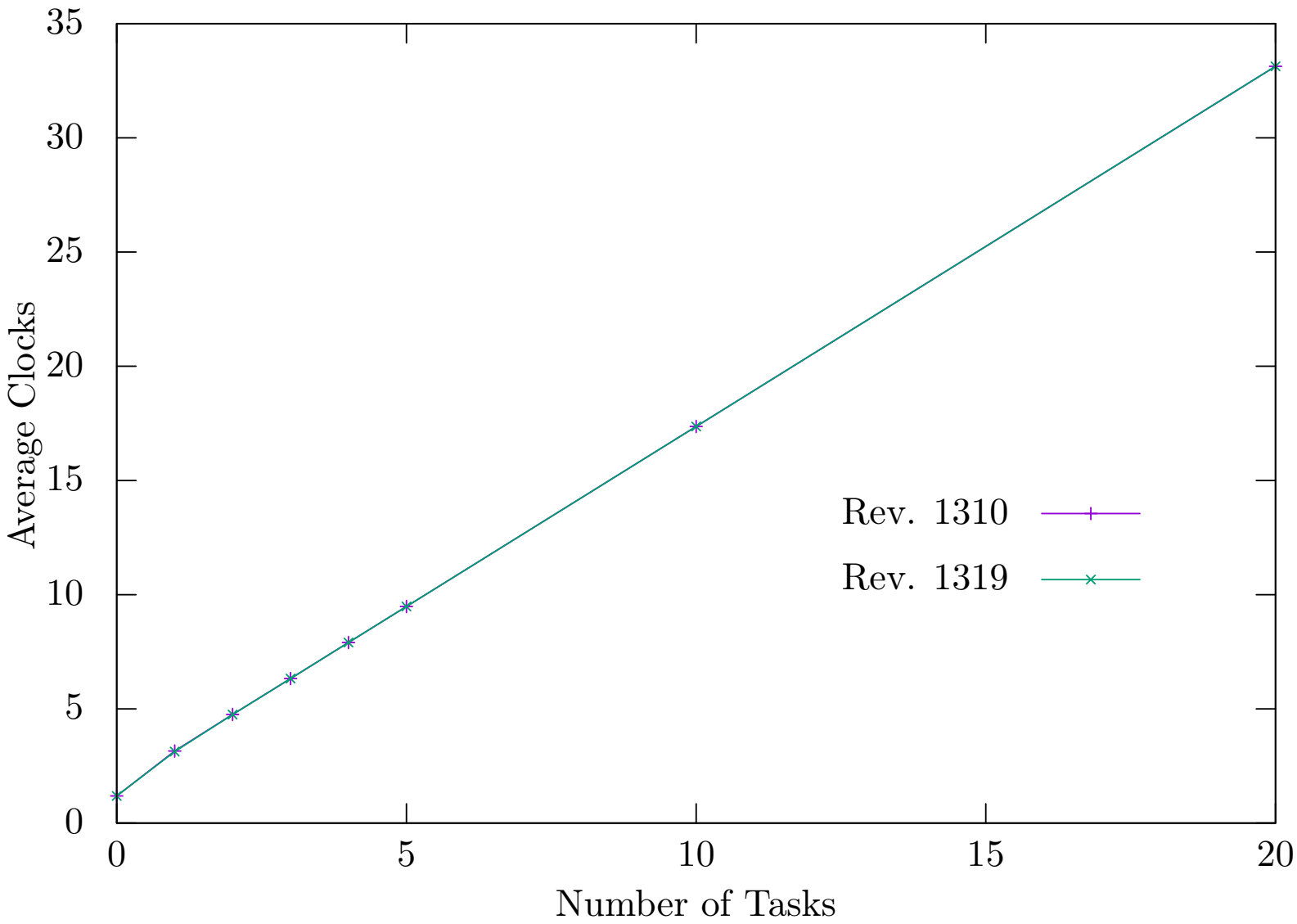
## Rev. 1310

```
1  Execution times of snd_pdq when 0 data are queued.
2  3 : 8236
3  4 : 1764
4  Execution times of snd_pdq when 10 data are queued.
5  4 : 6705
6  5 : 3295
7  Execution times of snd_pdq when 20 data are queued.
8  5 : 4940
9  6 : 5060
10 Execution times of snd_pdq when 30 data are queued.
11 6 : 3177
12 7 : 6823
13 Execution times of snd_pdq when 40 data are queued.
14 7 : 1411
15 8 : 8589
16 Execution times of snd_pdq when 50 data are queued.
17 9 : 9646
18 10 : 354
19 Execution times of snd_pdq when 100 data are queued.
20 14 : 823
21 15 : 9177
22 Execution times of snd_pdq when 200 data are queued.
23 26 : 3175
24 27 : 6825
25 Execution times of snd_pdq when 300 data are queued.
26 38 : 5529
27 39 : 4471
```

## Rev. 1319

```
1  Execution times of snd_pdq when 0 data are queued.
2  3 : 9647
3  4 : 353
4  Execution times of snd_pdq when 10 data are queued.
5  4 : 8119
6  5 : 1881
7  Execution times of snd_pdq when 20 data are queued.
8  5 : 6353
9  6 : 3647
10 Execution times of snd_pdq when 30 data are queued.
11 6 : 4587
12 7 : 5413
13 Execution times of snd_pdq when 40 data are queued.
14 7 : 2823
15 8 : 7177
16 Execution times of snd_pdq when 50 data are queued.
17 8 : 1059
18 9 : 8941
19 Execution times of snd_pdq when 100 data are queued.
20 14 : 2237
21 15 : 7763
22 Execution times of snd_pdq when 200 data are queued.
23 26 : 4587
24 27 : 5413
25 Execution times of snd_pdq when 300 data are queued.
26 38 : 8000
27 39 : 2000
```

# perf3


Average Clocks
Number of Tasks
0
5
10
15
20
25
30
35
Rev. 1310
Rev. 1319

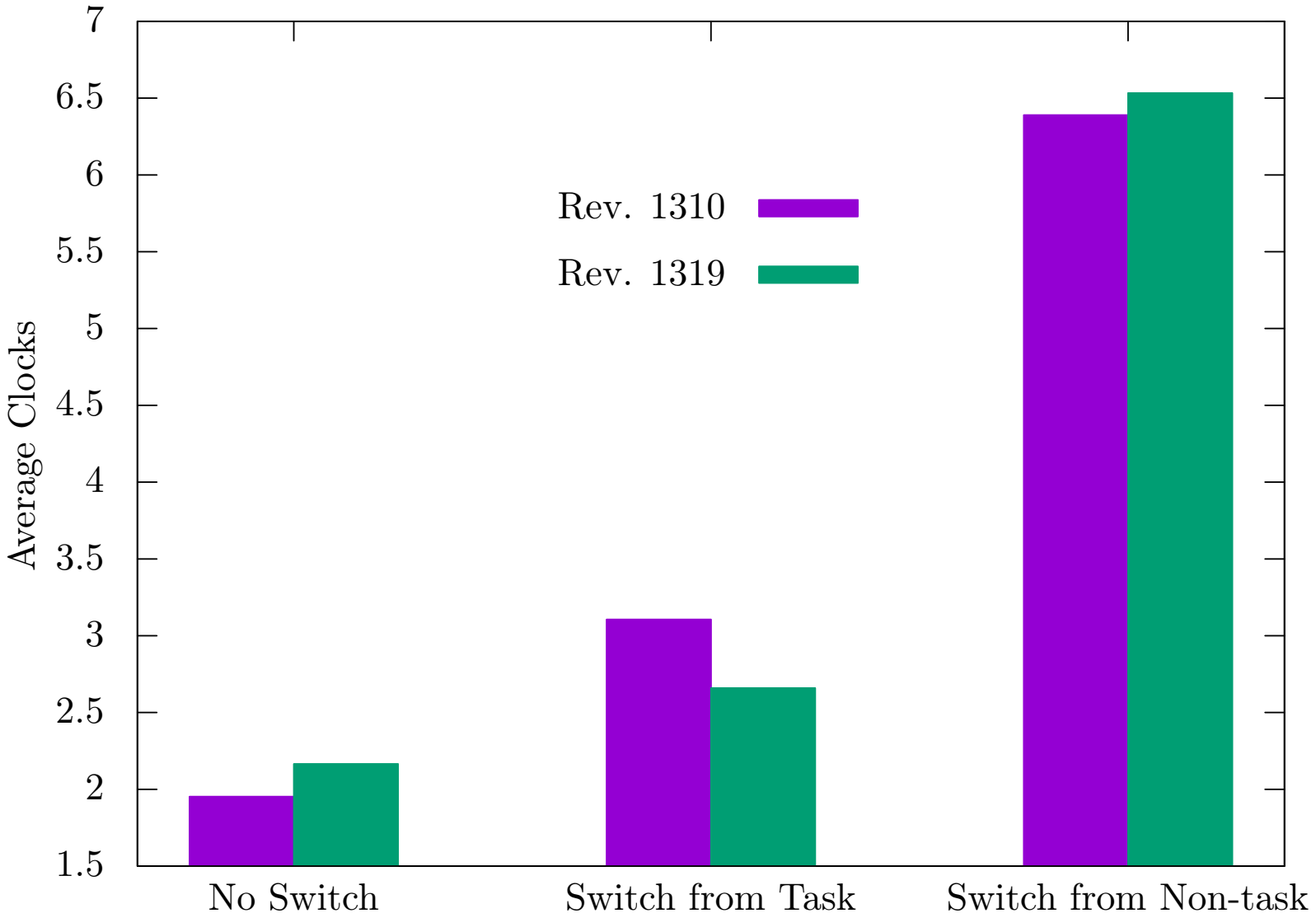
## Rev. 1310

```
Execution times of set_flg when 0 tasks are released from waiting.
1 : 588
2 : 9412
Execution times of set_flg when 1 tasks are released from waiting.
3 : 941
4 : 9059
Execution times of set_flg when 2 tasks are released from waiting.
5 : 4941
6 : 5059
Execution times of set_flg when 3 tasks are released from waiting.
7 : 9176
8 : 824
Execution times of set_flg when 4 tasks are released from waiting.
8 : 3409
9 : 6591
Execution times of set_flg when 5 tasks are released from waiting.
10 : 7646
11 : 2354
Execution times of set_flg when 10 tasks are released from waiting.
18 : 8823
19 : 1177
Execution times of set_flg when 20 tasks are released from waiting.
33 : 1175
34 : 8825
```

## Rev. 1319

```
Execution times of set_flg when 0 tasks are released from waiting.
1 : 588
2 : 9412
Execution times of set_flg when 1 tasks are released from waiting.
3 : 1177
4 : 8823
Execution times of set_flg when 2 tasks are released from waiting.
5 : 4942
6 : 5058
Execution times of set_flg when 3 tasks are released from waiting.
7 : 9177
8 : 823
Execution times of set_flg when 4 tasks are released from waiting.
8 : 3412
9 : 6588
Execution times of set_flg when 5 tasks are released from waiting.
10 : 7646
11 : 2354
Execution times of set_flg when 10 tasks are released from waiting.
18 : 8823
19 : 1177
Execution times of set_flg when 20 tasks are released from waiting.
33 : 1176
34 : 8824
```

# perf4


Average Clocks
7
6.5
6
5.5
5
4.5
4
3.5
3
2.5
2
1.5
Rev. 1310
Rev. 1319
No Switch
Switch from Task
Switch from Non-task

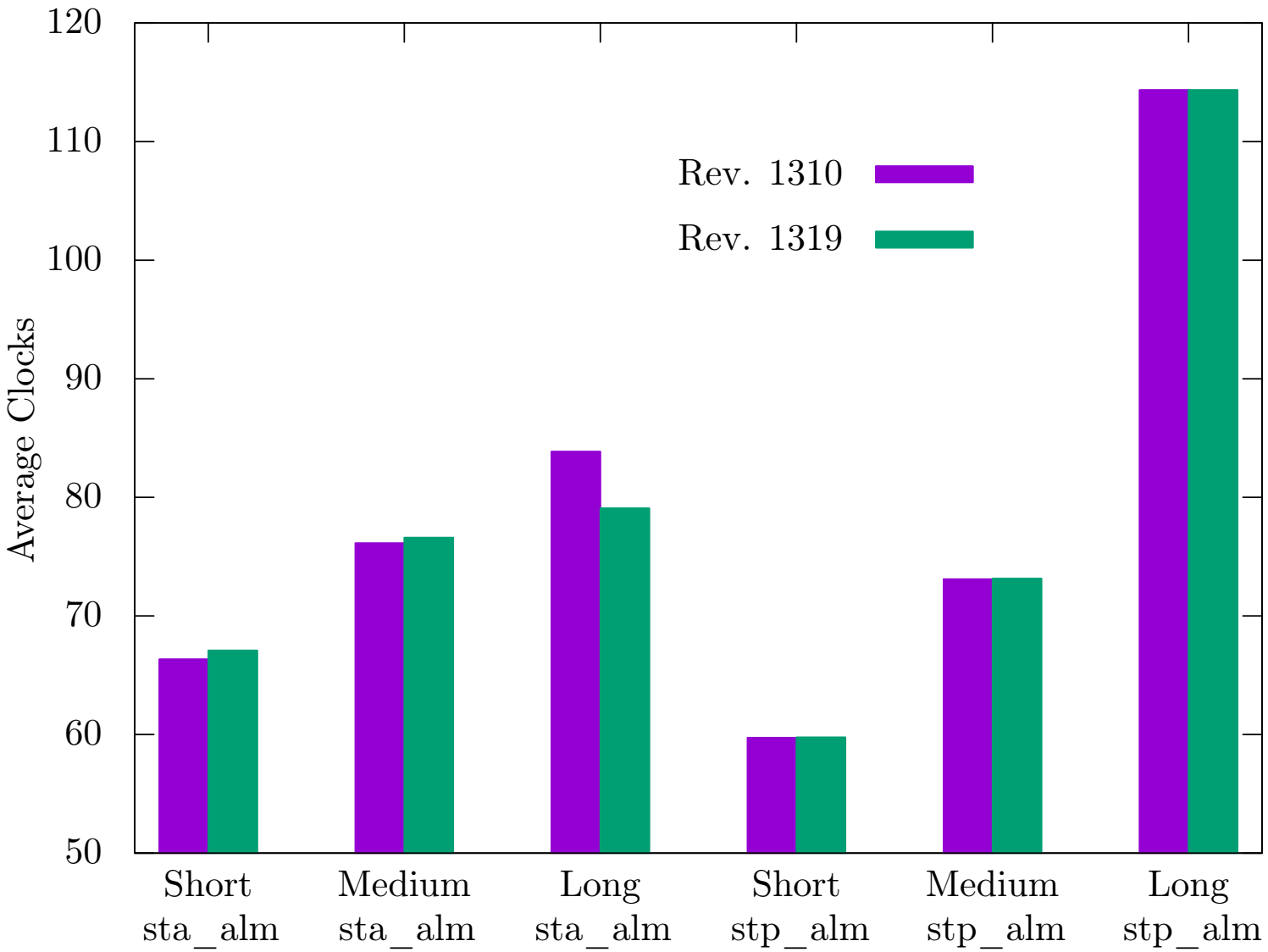
## Rev. 1310

```
1  Execution times of act_tsk from task context without task switch
2  3 : 7059
3  4 : 2941
4  Execution times of act_tsk from task context with task switch
5  3 : 1412
6  4 : 8587
7  5 : 1
8  Execution times of act_tsk from non-task context with task switch
9  7 : 857
10 8 : 143
```

## Rev. 1319

```
1  Execution times of act_tsk from task context without task switch
2  2 : 824
3  3 : 9176
4  Execution times of act_tsk from task context with task switch
5  3 : 5882
6  4 : 4118
7  Execution times of act_tsk from non-task context with task switch
8  7 : 714
9  8 : 286
```

# perf5


120
110
100
90
80
70
60
50
Average Clocks
Rev. 1310
Rev. 1319
Short sta_alm
Medium sta_alm
Long sta_alm
Short stp_alm
Medium stp_alm
Long stp_alm

## Rev. 1310

```
Execution times of 30 short sta_alm
67 : 894
68 : 106
Execution times of 30 medium sta_alm
76 : 104
77 : 896
Execution times of 30 long sta_alm
84 : 388
85 : 612
Execution times of 30 short stp_alm
60 : 505
61 : 495
Execution times of 30 medium stp_alm
73 : 141
74 : 859
Execution times of 30 long stp_alm
115 : 893
116 : 107
```

## Rev. 1319

```
Execution times of 30 short sta_alm
67 : 165
68 : 835
Execution times of 30 medium sta_alm
77 : 637
78 : 363
Execution times of 30 long sta_alm
79 : 164
80 : 836
Execution times of 30 short stp_alm
60 : 482
61 : 518
Execution times of 30 medium stp_alm
73 : 93
74 : 907
Execution times of 30 long stp_alm
115 : 895
116 : 105
```