组会报告

徐益

2018 年 12 月 17 日

1 工作内容

1. 学习并总结线程池；
2. 尝试两种新的调度方案；
3. 尝试改进线程池程序。

2 线程池

2.1 线程池的接口

```
1 void pool_init(int coreId_start, int _threadNum, int pool_index);
2 void pool_add_task(Fun myfun, void *arg, int pool_index);
3 void pool_destroy(int pool_index);
```

2.2 线程池的结构

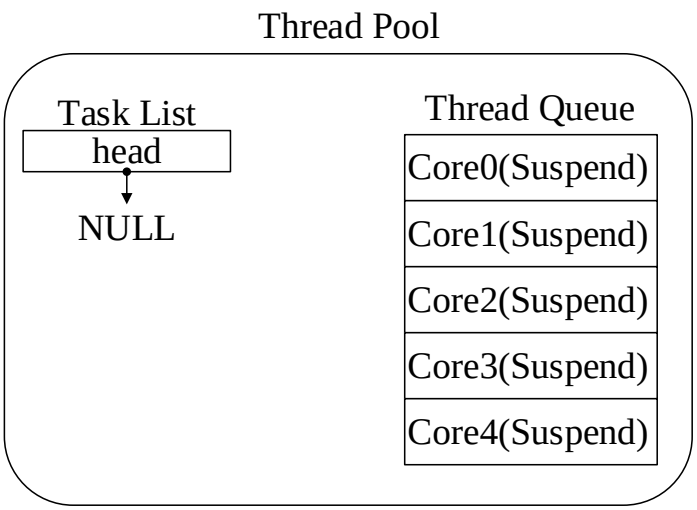

图 1: 初始化线程池

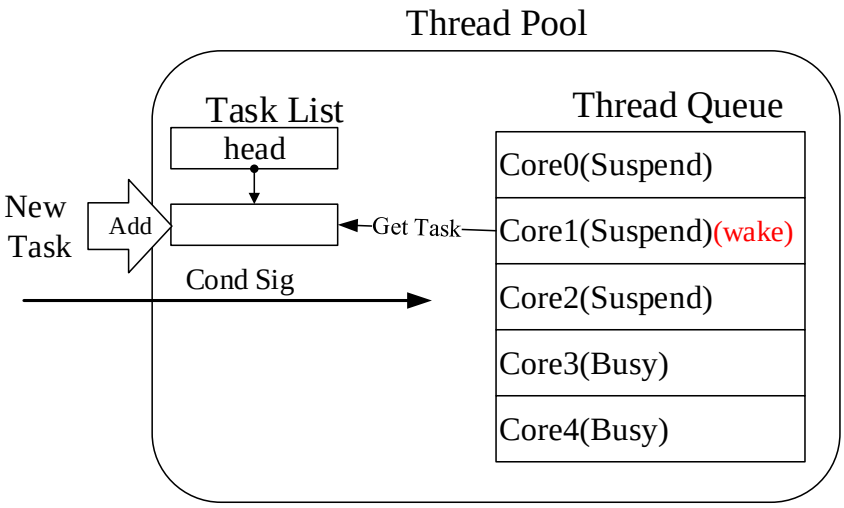

图 2: 有线程挂起时

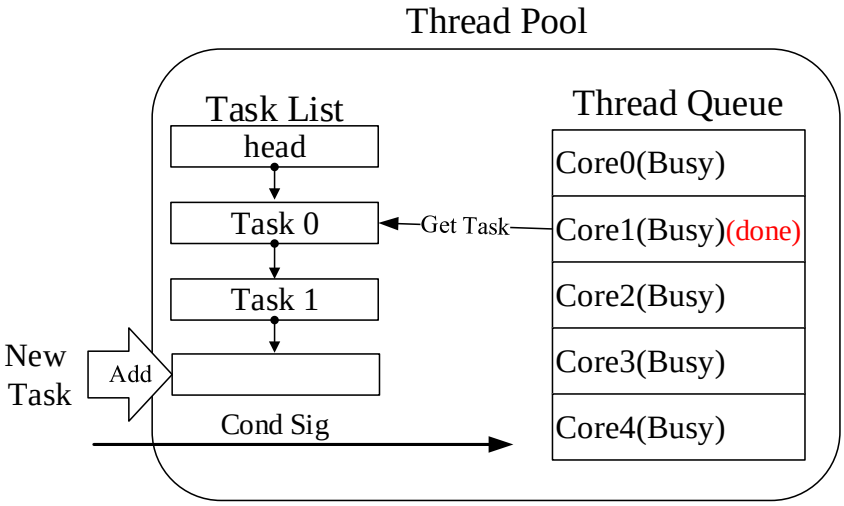

图 3: 线程全忙时

2.3 线程池的优势

1. 动态分配 CPU 核心；
2. 限制线程运行核心位置；
3. 避免重复创建和销毁线程。

3 两种新的调度方案

3.1 轮询方案 (不使用信号量)

```
534     pthread_mutex_lock(&rx_sche_mutex2[indx_tb][indx_vtb]);
535     rx_sche_flag2[indx_tb][indx_vtb]++;
536     pthread_mutex_unlock(&rx_sche_mutex2[indx_tb][indx_vtb]);
```

图 4: 关键步骤

```
Rx:
==============================
total bits: 49914368000
total time:  86.1977s
throughput: 579.0684Mbps
==============================
Rx:
==============================
total bits: 50571136000
total time:  87.3273s
throughput: 579.0989Mbps
==============================
Rx:
==============================
total bits: 51227904000
total time:  88.4603s
throughput: 579.1059Mbps
==============================
```

图 5: 轮询方案性能

```
Rx:
==============================
total bits: 49914368000
total time:  97.0158s
throughput: 514.4971Mbps
==============================
Rx:
==============================
total bits: 50571136000
total time:  98.2937s
throughput: 514.4902Mbps
==============================
Rx:
==============================
total bits: 51227904000
total time:  99.5709s
throughput: 514.4865Mbps
==============================
```

图 6: 信号量方案性能

3.2 不使用线程池

```
339     pthread_create(&decode_thread[indx_tb][indx_vtb][indx_cb], NULL,
340                    nr5g_phy_rx_decode_subthrd_nopool,
341                    (void *)&decode_arg[indx_tb][indx_vtb][indx_cb]);
```

图 7: 关键步骤

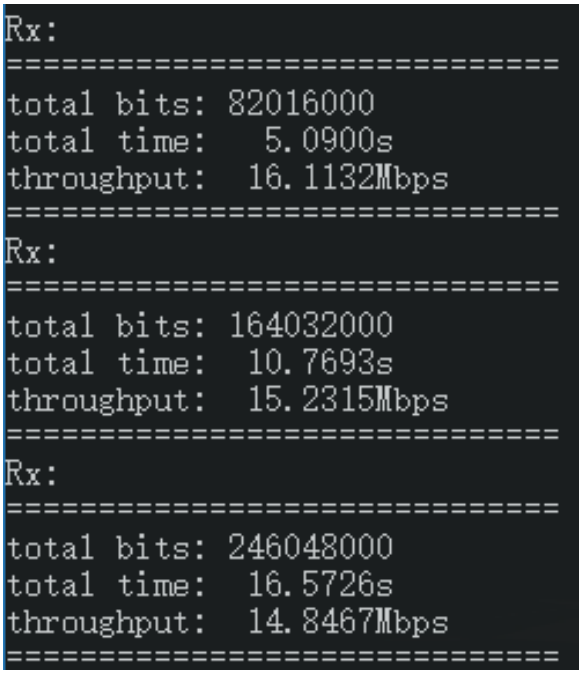

图 8: 性能

4 尝试修改线程池

4.1 线程池过大造成性能下降的问题

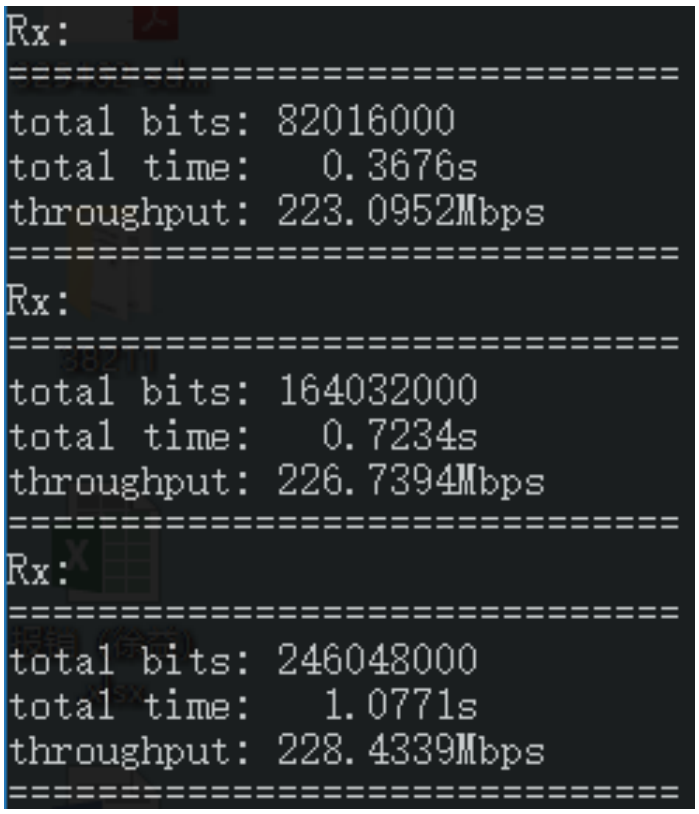

图 9: 单流 4CPU2CORE

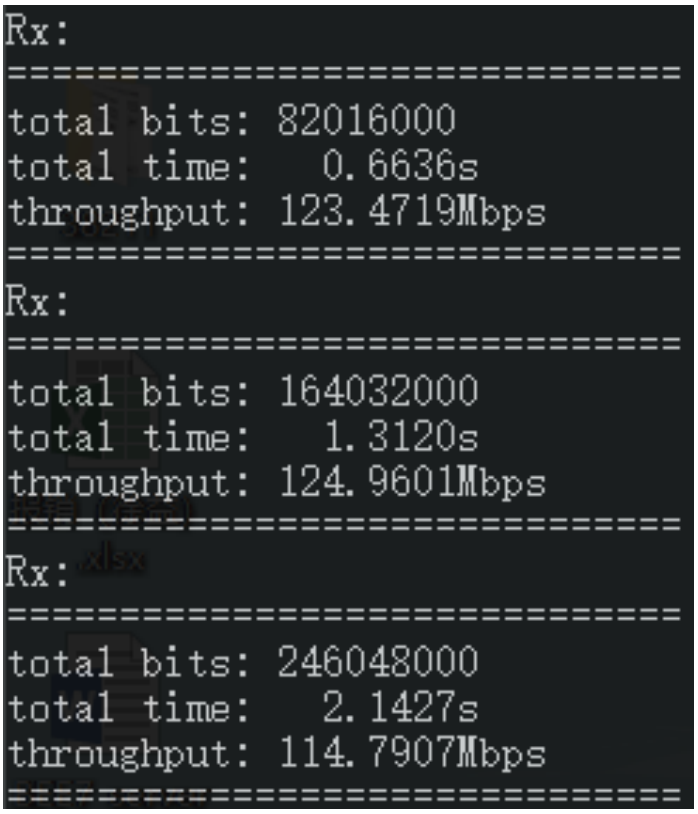

图 10: 单流 4CPU14CORE

```c
/*创建新线程 实际任务数量大于 最小正在等待的任务数量，存活线程数小于最大线程数*/
if (queue_size >= MIN_WAIT_TASK_NUM && live_thr_num <= pool->max_thr_num)
{
    printf("admin add----------\n");
    pthread_mutex_lock(&(pool->lock));
    int add = 0;

    /*一次增加 DEFAULT_THREAD_NUM 个线程*/
    for (i = 0; i < pool->max_thr_num && add < DEFAULT_THREAD_NUM && pool->live_thr_num < pool->max_thr_num; i++)
    {
        if (pool->threads[i] == 0 || !is_thread_alive(pool->threads[i]))
        {
            pthread_create(&(pool->threads[i]), NULL, threadpool_thread, (void *)pool);
            add++;
            pool->live_thr_num++;
            printf("new thread ----------------------\n");
        }
    }

    pthread_mutex_unlock(&(pool->lock));
}

/*销毁多余的线程 忙线程x2 都小于 存活线程，并且存活的大于最小线程数*/
if ((busy_thr_num * 2) < live_thr_num && live_thr_num > pool->min_thr_num)
{
    // printf("admin busy --%d--%d----\n", busy_thr_num, live_thr_num);
    /*一次销毁DEFAULT_THREAD_NUM个线程*/
    pthread_mutex_lock(&(pool->lock));
    pool->wait_exit_thr_num = DEFAULT_THREAD_NUM;
    pthread_mutex_unlock(&(pool->lock));

    for (i = 0; i < DEFAULT_THREAD_NUM; i++)
    {
        //通知正在处于空闲的线程，自杀
        pthread_cond_signal(&(pool->queue_not_empty));
        printf("admin cler --\n");
    }
}
```

图 11: 动态调整线程池大小

4.2 问题

1. 每个线程池需要额外的管理线程；
2. 线程池大小调整过于频繁。

5 下阶段计划

1. 尝试从文献中寻找新的线程池方案；
2. 完善 PHY-MAC 接口（包括链路自适应和分块重传部分）。