凌风探梅的专栏
RSS订阅
图像处理，图像分割，特征提取，机器学习，模式识别，深度学习等

转 嵌入式ARM多核处理器并行化方法

2016年06月01日 15:26:32 阅读数：

from:http://ee.ofweek.com/2014-11/ART-11001-2808-28902672.html

目前，嵌入式多核处理器已经在嵌入式设备领域得到广泛运用，但嵌人式系统软件开发技术还停留在传统单核模式，并没有充分发挥多核处理器的性能。程序并行化优化目前在PC平台上有一定运用，但在嵌入式平台上还很少，另外，嵌入式多核处理器与PC平台多核处理器有很大不同，因此不能直接将PC平台的并行化优化方法应用到嵌人式平台。本文分别从任务并行和缓存优化[illegible]进行并行化优化的研究，探索在嵌人式多核处理器上对程序进行并行化优化的方法。

1 嵌入式多核处理器结构

嵌人式多核处理器的结构包括同构（Symmetric）和异构（Asymmetric）两种。同构是指内部核的结构是相同的，这种结构目前广泛应用在PC多核处理器；而异构是指内部核的结构是不同的，这种结构常常在嵌入式领域使用，常见的是通用嵌入式处理器+DSP核。本文探究的嵌入式多核处理器采用同构结构，实现同一段代码在不同处理器上的并行执行。

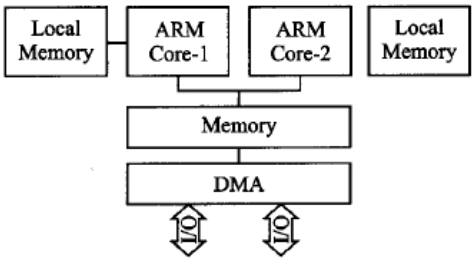

图1 ARM SMP处理器结构

在目前嵌入式领域中，使用最为广泛的为ARM 处理器，因此以ARM 双核处理器OMAP4430作为研究对象。ARM 对称多处理（Symmetric Multi-Processing，SMP）结构如图1所示，根据程序的局部性原理，每一个处理器都具有私有的内存（Local Memory），常见的是一级缓存（L1Cache）。然而，多个处理器之间又涉及到相互通信问题，因此在常见的ARM 处理器中使用二级缓存（L2 Cache）来解决这一问题。基于对称多处理器结构，所有的处理器（通常为2的倍数）在硬件结构上都是相同的，在使用系统资源上也是平等的。更重要的是，由于所有的处理器都有权利去访问相同的内存空间，在共享内存区域中，任何一个进程或者线程都可以运行在任意一个处理器之上，这样就使得程序的并行化成为可能。2在嵌入式多核平台上进行并行化优化，需要考虑以下问题：

① 并行化程序的性能取决于程序中串行化部分，程序性能不会随着并行线程数目的提升而不断提升；

② 嵌入式多核处理器相对于PC处理器而言，其总线速度较慢，并且缓存（Cache）更小，会造成大量数据在内存（Memory）和缓存（Cache）间不断拷贝，因此在进行并行化优化的过程中，应考虑缓存友好性（Cache friendly）；

③ 程序并行化执行线程数目应当小于或等于物理处理器的数目，线程过多会造成线程间抢占处理器资源，致使并行化性能下降。

2 OpenMP并行化优化

2.1 0penMP工作原理简介

OpenMP是一个基于共享内存模式的跨平台多线程并行[illegible]程接口。主线程生成一系列的子线程，并将任务映射到子线程进行

以使用work-sharingconstructs来划分任务，使每个线程执行其分配部分的代码。通过这种方式，使用OpenMP可以实现任务并行和数据并行。

表 1　算法加速比分析

项　目	数　值					平均值
数据规模	2^20	2^21	2^22	2^23	2^24	—
并行 2 线程	x1.25	x1.29	x1.38	x1.3	x1.29	x1.30
并行 4 线程	x1.35	x1.22	x1.33	x1.29	x1.27	x1.29
缓存优化	x1.19	x1.18	x1.26	x1.19	x1.21	x1.21

表 2　算法执行时间标准差

数据规模	2^20	2^21	2^22	2^23	2^24
并行 2 线程	62	66	201	529	477
并行 4 线程	54	72	223	489	603
缓存优化	12	23	77	94	130

图2 任务并行模型

任务并行模式创建一系列独立的线程，每一个线程运行一个任务，线程之间相互独立，如图2所示。OpenMP使用编译原语session directive和task directive来实现任务分配，每个线程可以独立运行不同的代码区域，同时支持任务的嵌套和递归。一旦创建任务，该任务就可能会在线程池（其大小等于物理线程数目）中空闲的线程上执行。

数据并行也就是数据级并行，对任务中处理的数据进行分块并行执行，如图3所示。C语言中的for循环最适合使用数据并行。

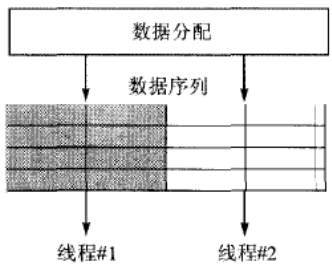

图3 数据并行模型

2.2 快速排序算法原理

快速排序算法是一种递归分治算法，算法中最为关键的就是确定哨兵元素（pivot data）。数据序列中小于哨兵的数据将会放在哨兵元素的左侧，序列中大于哨兵的数据将会被放在哨兵元素的右侧。当完成数据扫描后，哨兵元素分成的左右两个部分就会调用快速排序算法递归进行。

快速排序算法中涉及算法的递归调用，会产生大量任务，并且这些任务相互独立，非常适合OpenMP的任务并行模式；另外，就一次快速排序搜索算法而言，哨兵元素对于左右子区间数据容量大小具有决定性作用，考虑到嵌入式平台的缓存（Cache）空间较小，需要对哨兵元素筛选算法进行优化，尽量使得划分出来的左右子区间更均衡，满足负载均衡的要求。

2.3 任务并行化优化

通过对快速排序算法的分析，快速排序是一个递归调用算法，算法的执行过程中会产生大量重复函数调用，并且函数的执行相互独立。对于快速排序的一次扫描运算而言，算法首先确定哨兵元素（pivot），并对数据序列进行一次调整，然后对哨兵元素的左右区间再次进行递归调用算法。

如下所示，对任务并行化优化针对每次扫描调整后的左右子区间，将每个子区间的运算抽象为一个任务，并通过OpenMP中的任务并行化原语#pragma omp task实现任务的并行化执行，从而实现了快速排序的任务并行化优化。

```
void qsort_test(){
    # pragma omp parallel
    # pragma omp single
    qsort(0,size);
}
void qsort(int low,int high){
    int pivot;
    if(low<high){

pivot=median3_partions(low,high);
# pragma omp task
  qsort(low,pivot-1);
# pragma omp task
  qsort(pivot+1,high);
  }
}
```

任务空间中的数据大小取决于哨兵元素，因此，算法选取的划分算法（Partition Algorithm）应尽量将数据序列的划分均衡化，本文使用简单划分算法和三元中值法（Median-of-Three Method）进行测试。

2.4 缓存优化

缓存优化（Cache friendly）的目标是减少数据在内存和缓存之间的拷贝。对于220个整型数据而言，数据大小为4 MB，本文的测试平台（）MAP4430的二级缓存为1 MB，需要将数据划分为4个部分。

如下所示，算法将4部分数据分为4个快速排序任务，4部分任务并行执行，完成后每部分数据序列排序完成，需要将4部分数据进行合并形成完成数据序列，因此在并行任务结束后，需要对数据进行归并排序。

```
void qsort_test(int first,int last){
    # pragma omp parallel
    # pragma omp task
        qsort_qsort(first,((last+1)/divide_size)-1);
    # pragma omp task
        qsort_qsort(((last+1)/divide_size),2 * ((last+1)/
divide_size)-1);
    # pragma omp task
        qsort _qsort(2 * ((last+1)/divide_size),3 * ((last+
1)/divide_size)-1);
    # pragma omp task
        qsort _qsort(3 * ((last+1)/divide_size),4 * ((last+
1)/divide_size)-1);
    # pragma omp taskwait
    # pragma omp task

            MergeSort(inputs, mergeoutputs, 0, 2 * ((last+1)/
    divide_size)-1);
        # pragma omp taskwait
        # pragma omp task
            MergeSort(inputs, mergeoutputs, 0, 3 * ((last+1)/
    divide_size)-1);
        # pragma omp taskwait
        # pragma omp task
            MergeSort(inputs, mergeoutputs, 0, 4 * ((last+1)/
    divide_size)-1);
    }
```

3 并行化性能分析

3.1 实验环境介绍

本文采用德州仪器（Texas Instruments）的OMAP4430嵌入式开发平台。OMAP443O为嵌入式多核处理器，拥有对称多处理双核ARM 处理器（Dual-core ARM Cortex-A、一级缓存32 KB、二级缓存1 MB，嵌入式操作系统采用Ubuntul2.O4内核，编

如下式所示，采用计算加速比的方式来分析并行优化的性能，加速比数值越大表示算法的并行程度越高，最低为1.性能测试采用4个算法版本，包括串行版本、并行2线程、并行4线程和缓存优化版，从不同角度来分析性能。

$$并行化加速比=\frac{算法串行执行时间}{算法并行执法时间}$$

如图4所示，从折线图可以看出，3种并行化优化算法相对于串行版本，算法的并行性能都有较大提升，如表1所列，其并行加速比分别为1.30、1.29和1.21.对任务并行优化方案而言，分别使用2线程和4线程版本进行测试，从加速比的分析结果看来，2线程版本较4线程版本略好。理论上并行线程的数目越多性能越好，但本文采用OMAP443O只有两个对称多处理核心，即使算法拥有4个并行线程，但实际执行的线程只有2个，同时4个线程在获取2个物理处理器时存在竞争关系，因而造成性能较之2线程版本有所下降。

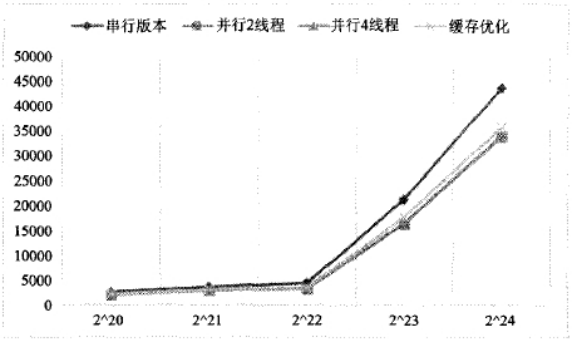

图4 算法执行时间

评价并行算法优劣还需考虑算法的负载均衡性，如表1、表2所列，缓存优化方案标准差远远小于任务并行化方案。究其原因，对于任务并行化方案而言，不同的测试数据以及划分算法（partition）对区间的划分有重要影响，从而造成任务执行时间变化范围很大；对于缓存优化方案而言，其实质是数据并行，其每一个任务都是根据缓存大小进行划分，因此每一个任务处理的数据规模基本一致，每一个任务执行的时间更确定，但由于并行任务执行完成后，需要对数据进行归并，造成一定的性能下降。

表 1　算法加速比分析

项　目	数　值					平均值
数据规模	2^20	2^21	2^22	2^23	2^24	—
并行 2 线程	x1.25	x1.29	x1.38	x1.3	x1.29	x1.30
并行 4 线程	x1.35	x1.22	x1.33	x1.29	x1.27	x1.29
缓存优化	x1.19	x1.18	x1.26	x1.19	x1.21	x1.21

表 2　算法执行时间标准差

数据规模	2^20	2^21	2^22	2^23	2^24
并行 2 线程	62	66	201	529	477
并行 4 线程	54	72	223	489	603
缓存优化	12	23	77	94	130

结语

本文通过对嵌入式多核处理器硬件结构的分析，从对称多处理角度对串行快速排序算法进行并行化优化，取得了很好的效果。

以ARM 双核处理器（OMAP4430）作为测试平台，从任务并行和缓存优化实现并行优化，从性能测试的结果看，任务并行具有良好的加速比，但负载均衡性差，并行线程数目不应超过物理处理器核的数目，过多的并行线程竞争处理器资源，造成性能下降。缓存优化具有良好的负载均衡性，但需要后续进行归并操作，造成性能有所下降。

总之，在嵌入式多核处理器上进行并行化优化，一方面要充分发掘嵌人式多核处理器的并行性能，提高程序的并行性；另一方面也要考虑程序算法的负载均衡性，确保在不同应用环境中程序性能一致。

个人分类： ARM　OpenMP