多核处理器构架的高速 JPEG 解码算法

关键词：多核处理器，JPEG，解码，多核，嵌入式处理器

JPEG(Joint Photographlc Experts Group)是一个适用范围很广的静态图像数据压缩标准，目前广泛应用于照相机、打印机等方面的图像处理。在这些应用中，设计出一个高速高效的 JPEG 解码器已经成为一个重要的研究方向。随着对嵌入式系统实时性、高性能和可扩展性要求的提高，多核(multi—core)嵌入式处理器的应用场合日益增多。

1 JPEG 解码算法原理

JPEG 压缩是一种有损压缩。它利用人的视角系统特性，使用量化和无损压缩编码相结合的方式去掉视角的冗余信息和数据本身的冗余信息来达到压缩的目的。JPEG 算法可分为基本 JPEG(Baseline system)和扩展 JPEG(Extended system)。其中 Baseline system 应用尤其广泛。本文主要讨论 Baseline system 的解码。JPEG 解码算法框图如图 1 所示。

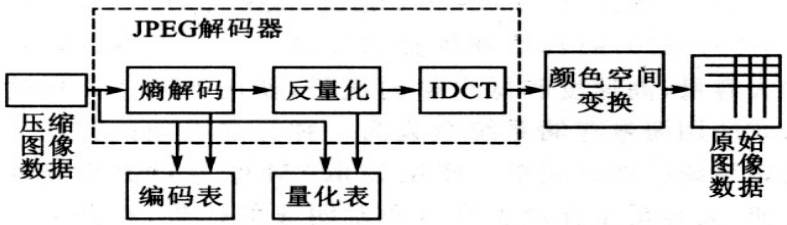

图 1　JPEG 解码算法框图

(1)颜色空间变换

JPEG 算法本身与颜色空间无关，因此“RGB 到 YUV 变换”和“YUV 到 RGB 变换”不包含在 JPEG 算法中。但由于作为输出的位图数据一般要求 RGB 的表示，所以将颜色空间变换也表示在算法框图中。

(2)JPEG 的编解码单元

在 JPEG 中，对于图像的编解码是分块进行的。整个图像被划分为若干个 8×8 的数据块，称为最小编码单元(MCU)，每一个块对应于原图像的一个 8×8 的像素阵列；各行的编解码顺序是从上到下，行内的编解码顺序是从左到右。

值得注意的是，由于一幅图像的高和宽不一定是 MCU 尺寸的整数倍，因此需要对图像的最右边一列或其最下边一行进行填充，扩展其高或宽，使得可以将整个图像划分为整数个 MCU；而在解码输出时，这些复制的行列是要被抛弃的。

(3)熵解码器

在 JPEG 的熵编码时，首先利用空间相关性对各块的直流值采用差分编码，即对相邻块之间的直流差值编码，以达到压缩码长的目的。然后对于交流部分以 ZigZag 方式扫描块中的元素，对块内元素采用先游程编码后哈夫曼编码的混合编码方式，得到一维二进制块码流。熵编码过程是由直流部分的差分编码和交流部分的 ZigZag 扫描、游程编码、哈夫曼编码组

成。而相应的熵解码过程是编码的逆过程，在解码端接收到的是由变长码(VLC)和变长整数(VLI)组成的数据流。为了从此数据流中恢复编码前的 DCT 系数，必须根据哈夫曼编码的原理及其各级码表生成的细节，生成哈夫曼解码表，再根据解码算法来恢复 DCT 的直流和交流系数。

(4)反量化
在 JPEG 解码端要利用发送过来的量化表对量化值进行译码。JPEG 文件里一般含有两个量化表：一个亮度分量的量化表，一个色度分量的量化表。反量化就是对熵解码出来的系数矩阵乘上相应的量化矩阵：

$$F(u,v) = C(u,v) \times Q(u,v)$$

其中，C(u，v)代表熵解码输出，Q(u,v)代表相应的量化矩阵。

(5)IDCT。变换
JPEG 解码算法能否满足实时应用，关键在于 8×8 的二维 IDCT 的计算速度。在编码阶段，正向离散余弦变换(FDCT)把空间域表示的图变换成频率域表示的图；相应
地在解码阶段，逆向离散余弦变换(IDCT)将频率域表示的图变换为空间域表示的图。

在 IDCT 的实现上，目前有多种算法。传统的方法是行-列法，即先对每行(列)进行一维 IDCT 计算，再对每列(行)进行一维 IDCT 计算。还有多项式变换法和三角函数公式法，这两种方法的加法次数与行-列法相当，乘法次数仅为行-列法的一半。但这两种方法的问题在于实现方法复杂，对于目标平台 (VLIW)来说，这样的结构难以提高指令的并行性，并且对于目标平台来说，乘法指令的执行时间与加法指令相当，因此减少了行-列法实现的代价。

2 多核处理器构架
FRl000 是 Fujitsu 公司生产的主要应用于嵌入式系统的多核处理器。FRl000 将 4 个处理器核(processorelement)集成在 1 枚芯片上，各个处理器核之间共享内存和其他外部设备。4 个处理器核分别叫作 PM(PE0)、PE1、PE2、PE3。其中，每个处理器内核均为一个独立的 VLIW(超长指令集)型架构的处理器，每个处理器核上都设置有独立的高速二级缓存，以减小多核处理器在并行访问内存时的瓶颈效应。FRl000 多核处理器的硬件结构如图 2 所示。

FR1000 在每一个处理器核上都运行一个独立的实时操作系统(RTOS)，而各个处理器核之间通过并行扩展库 (MP extended library)进行通信。通过并行扩展库的扩展，运行在一个处理器核上的任务不仅能和运行在同一处理器核上的任务通信，还能和运行在不同处理器核上的任务通信。这样，任务间就可以通过协同通信来完成特定的应用。而通过将应用划分为不同的可以并行运行的任务而运行在不同的处理器核上，就可以并行地处理数据，从而达到提高系统性能的目的。

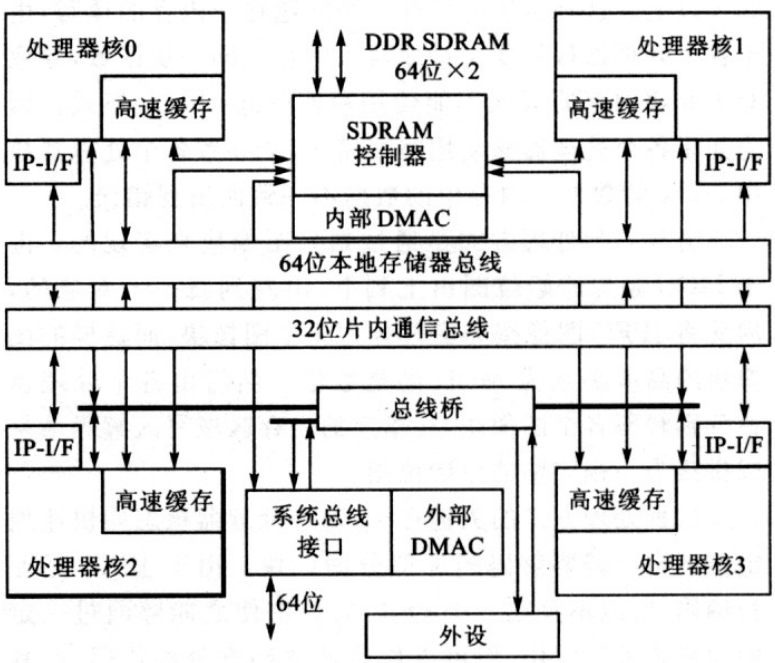

图 2　FR1000 硬件模块结构图

FRl000 系统的结构框图如图 3 所示。

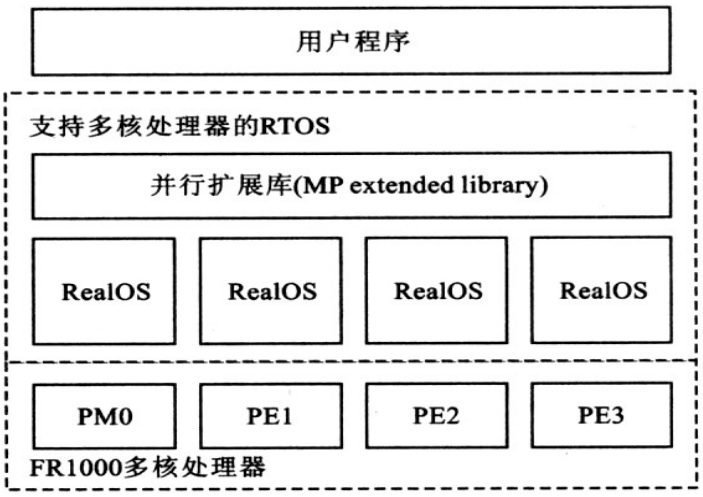

图 3　FR1000 系统结构框图

从 FRl000 的构架可以看出，为了提升对于图形和多媒体数据的处理速度，处理器着重于扩展其并行处理数据的能力。这样的扩展主要表现在两个方面：一方面使用 VLIW 构架的处理器核(这样的处理器核一次最多能够并行执行 8 条指令，这样的并行性主要由编译器支持，这是一种指令的并行性)；另一方面使用多核处理器(CMP)的构架，使针对应用划分的任务能够真正并行地运行在多个处理器核上。(这样的并行性需要由应用支持，通过恰当的划分任务来实现)

3 JPEG 解码算法在多核处理器上的实现

针对 FRI000 处理器的特点，需要对 JPEG 图像的解码划分为适当的可以并行执行的任务进行处理。比较直观的想法是，将 JPEG 图像划分为 4 个部分，分别在 4 个处理器核上进行解码。但由于 JPEG 图像的数据流是变长编码，根据现有的数据流，难以将其划分为 4 个能并行解码的图像。(这样的划分付出的时间代价过大)

根据前面所叙述的 JPEG 图像解码原理可以看出，解码的基本单位是 MCU，因此在第一步熵解码之后生成的 MCU 是可以并行解码的最小单元。因此对一个 JPEG 图像在多核处理器上进行并行解码的关键在于，将此，JPEG 图像所包含的 MCU 负载均衡地分配到各个处理器核上进行并行解码处理。

由此，处理方法有两种：一种是以一个 MCU 作为任务分配的单位，由 PM 通过熵解码生成 MCU，然后将 MCU 均匀地分配到各个处理器核(PE)上，由各个处理器核在完成 MCU 的解码之后再写入到位图的相应位置。这样做的好处有两点：①可以做到很好的负载均衡，使每一个处理器核都承担几乎相同的负载。②可以使熵解码和 MCU 的解码并行进行。但这种做法的一个很大问题在于处理器核之间通信所消耗的时间代价过大。因为这可以抽象为一种生产者与消费者的模型，生产者在每次生产出一个 MCU 的时候都需要与消费者进行一次通信或者说更新消费者端的数据输入。经过实测以后发现，这种做法所带来的通信开销过大，占到解码程序运行时间的 20%以上。这种做法的另外一个问题在于内存的读写，由于各个处理器核需要交错地写入内存的同一块区域，导致对于此块内存的写入不能使用写回(copy back)模式。因为如果各个处理器核使用写回模式，会导致各个处理器核中 cache 的数据与内存中的数据不一致而出现错误。

另外一种处理方式是通过划分图像块来实现的。由于 MCU 是与原始位图由上到下、由左到右一一对应的，因此将 JPEG 图像按高度等分为 4 个图像块，而这样的图像块的高度必须为 MCU 的整数倍。然后由各个处理器核分别解码各个图像块，在指定的内存区域写入解码结果以拼接为一幅完整的原始位图。

这种处理方式的关键在于，每个处理器核怎样快速地定位到自己需要解码的那部分图像块。由于 JPEG 是变长编码，所以不存在一个 O(1)的算法使之能够通过一定的偏移量进行定位，但可以修改熵解码部分的代码，使其能够跳过不必要的解码，快速定位到需要处理的区域。具体来说，定位的过程实际上就是对 MCU 进行计数的过程，定位时没有必要保存 MCU 的内容，只需要对解出来的 MCU 进行计数。由于 MCU 与原始位图的一一对应关系，所以可以通过对 MCU 的计数来定位到需要处理的区域。具体换算公式如下：

$$N_{mcu} = \lfloor \mathrm{Height} \rfloor / 8 \rfloor * (\mathrm{PEID}/4) * \lfloor \mathrm{Width}/8 \rfloor$$

其中 Height 表示图像高度，Width 表示图像宽度，PEID 表示每个处理器核所对应的编号(1～4)。

各处理器核的解码流程如图 4 所示。

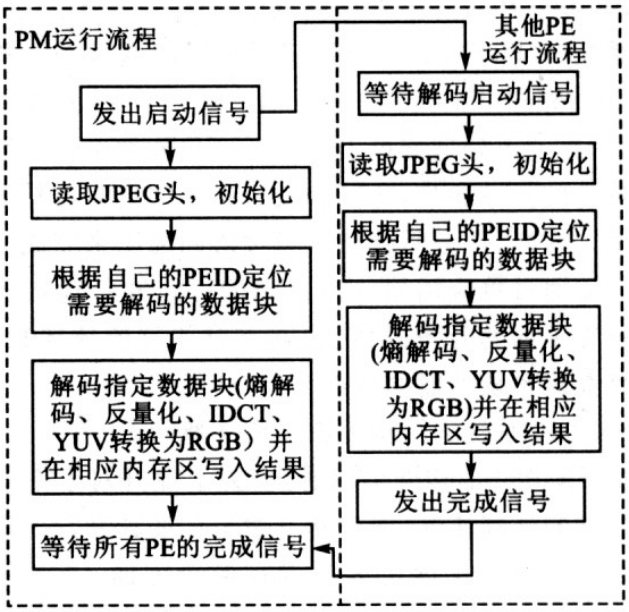

图 4　解码流程

这种处理方式的问题在于，每一个处理器核都需要花费额外的时间来定位需要解码的数据块；但实测以后发现，定位操作所消耗的时间只占 5%左右。因为在 FRI000 平台上，大量的解码时间消耗主要在于 IDCT 变换和 YUV 到 RGB 的颜色空间转换上面。这种处理方式降低了通信的时间消耗，在一幅 JPEG 的图像解码中只需要两次处理器核之间的通信。这种处理方式的另外一个好处在于，每一个 PE 在写入结果的位图数据时可以对内存的写入采用写回(copy back)模式，只需要在图像块交界的地方作刷新 cache 的操作就可以保证结果的正确性。在随后的关于优化的讨论中可以看到，这种方式对于提高解码的速度起着相当重要的作用。

4 优化

一般来说，一个程序在多核处理器上的运行时间除上在其中一个单核处理器上的运行时间称为多核并行度(MP)。在有 4 个处理器核的 FRI000 处理器上， MP 的极限值(存在必要的通信开销)应该为 25%。但根据图 3 的解码流程，实测的 MP 只有 43%左右。进一步分析后发现，由于多个处理器核沿相同的流程进行解码，从而在相同的时间里对内存有大量并发的读写操作，而这样的并发操作导致对于内存的读写成为系统的瓶颈。在单核上需要 16～20 个周期的 1 行 cache 读入操作，在多个处理器核同时运行时，需要 30～40 个周期才能完成。

优化主要从两个方面进行：

①尽量减少对内存的读写操作。一般的 JPEG 解码程序，会以行为单位保存熵解码后的中间结果，也就是使用存储 1 行 MCU 的空间作为临时缓冲区。这样的临时缓存区是随着图像行的宽度增大而增大的，当图像的宽度变大到一定程度的时候，这样的临时缓存将很可能大到没有办法驻留在 cache 中，cache 不命中从而导致大量的内存读写和对于 cache 的置换。优化后将其改为熵解码完一个 MCU 后，立即作反量化、IDCT 和颜色空间变换，直至写入位图。这样只需要一个 MCU 大小的临时缓存。可以保证这样的缓冲一直保存在

cache 中，从而避免大量的读写内存的操作。但这样的方式需要恰当的判断边界条件，如前所述，由于图像的长宽不一定是 MCU 的整数倍，所以在最下一行和最右一列有填充数据，需要在解码的时候丢弃掉。

②恰当地选择内存的读写模式。由于整个解码程序中，在最后写入位图时需要大量地写入内存的操作。如果使用写透(write through)模式，每次均同时写入 cache 和内存，这样必然会造成大量的内存读写操作。所以在写入位图的区域使用写回模式，这样只需要在每次 cache 行置换的时候需要写入内存，极大地减少了对于内存的读写操作。但需要注意的是，在多核处理器的环境下，必须保证该内存区域和各个处理器核上的 cache 数据之间的一致性。这需要恰当地划分各个处理器核的内存读写区域，并且在读写各个区域交界的地方时用指令刷新相应的 cache 行。

值得注意的是，在多核处理器的构架上，由于多个处理器会并行访问内存，所以内存很容易成为瓶颈，在涉及大量内存操作的图像处理程序方面表现得尤为突出。因此对于程序的优化应该着重将注意力放在对于内存的读写优化方面。

5 实验结果

选取 256×256、1024×1024、4096×4096 三个 JPEG 图像进行解码，其耗费的周期数如表 1 所列。

表 1

图像大小/像素数	单核运行周期数/个	多核运行周期数/个	MP/%
256×256	2 046 859	579 265	28.3
1024×1024	47 449 173	13 337 963	28.1
4096×4096	539 573 524	150 294 950	27.8

可见对于尺寸越大的图像，其 MP 越接近于 25%的极限值，因为此时通信所占的开销越小；同时随着内存块的增大，在每个处理器核处理的图像块的边界处刷新 cache 行的代价也越小，而平均的 MP 约为 28%左右。

6 结论

针对多核处理器构架的特点，在其上实现高速的 JPEG 解码算法，其多核的并行度(MP)接近于 25%的极限值。上述实现虽然只针对 FRl000 的多核处理器，但同样适应于其他具有多核构架的处理器。此外针对多核处理器构架方面的优化方法对于其他运行于多核处理器构架上的应用也有一定的借鉴价值。

来源：单片机及嵌入式系统应用