基于申威 421 的视频解码的向量化并行

裴　航[1,2],王　磊[1,2],王　威[1,2],张书钦[1]
(1. 中原工学院 计算机学院,河南 郑州 451191;
2. 中原工学院 前沿信息技术研究院,河南 郑州 451191)

摘　要:H. 264 解码器在申威平台移植后遇到解码效率低,视频播放不流畅等问题。为提升视频解码性能,满足国产申威平台用户的多媒体需求,首先对 FFmpeg 开源编解码库中 H. 264 解码器进行了详细分析,使用性能分析工具找到视频解码的热点函数。然后充分利用申威处理器的向量扩展部件,对解码器运动补偿、DCT 反变换等关键模块代码使用手工嵌入式汇编进行向量指令替换来缩短指令周期,实现向量化并行。最后对环路滤波代码中不能直接向量化的循环通过数组重组等方式满足向量化分析,然后进行向量化计算,更深层次挖掘多媒体并行能力,从而提升多媒体程序运行速度。实验结果表明,向量化后的视频解码性能最高提升了 35. 3%,释放了 CPU 资源,解决了视频播放不流畅的问题,有效推动了申威处理器市场化发展。

关键词:H. 264 解码器;FFmpeg 编解码库;申威处理器;单指令多数据流;并行计算
中图分类号:TP302　　**文献标识码**:A　　**文章编号**:1673-629X(2021)10-0081-06
doi:10. 3969/j. issn. 1673-629X. 2021. 10. 014

Vectorization Parallelism of Video Decoding Based on Shenwei 421

PEI Hang[1,2], WANG Lei[1,2], WANG Wei[1,2], ZHANG Shu-qin[1]
(1. School of Computer Science, Zhongyuan University of Technology, Zhengzhou 451191, China;
2. Research Institute of Frontier Information Technology, Zhongyuan University of Technology, Zhengzhou 451191, China)

Abstract: The H. 264 decoder encountered problems such as low decoding efficiency and unsmooth video playback after being transplanted on the Shenwei platform. To promote the video decoding performance and meet the multimedia needs of domestic Shenwei platform users, firstly the H. 264 decoder in the FFmpeg open source codec library is analyzed in detail, and the performance analysis tool is used to find the hot functions of video decoding. Then making full use of the vector expansion components of the Shenwei processor, we use manual embedded assembly for vector instruction replacement for key module codes such as decoder motion compensation and DCT inverse transformation to shorten the instruction cycle and achieve vectorization parallelism. Finally, in the loop filter code that cannot be directly vectorized, the vectorized analysis is satisfied by means of array reorganization, and then vectorized calculation is carried out to dig deeper into the multimedia parallel capabilities, thereby improving the running speed of the multimedia program. The experiment shows that the video decoding performance after vectorization is improved by up to 35. 3%, which frees up CPU resources, solves the problem of unsmooth video playback, and effectively promotes the market development of Shenwei processors.

Key words: H. 264 decoder; FFmpeg codec library; Shenwei instructions; single instruction multi-data stream; parallel computing

0　引　言

随着互联网的高速发展以及第五代移动通信技术的到来,人们对多媒体品质的需求日益提升,高清视频的高效编解码成为研究热点。很多处理器平台都研发出相应的扩展部件来增强计算机多媒体数据处理能力。单指令流多数据流(SIMD)[1]结构可以高效地对大规模数据进行并行处理,有效实现应用程序中规整的数据级并行(data level parallelism, DLP),从而提升了处理器的吞吐率,在一定程度上满足了多媒体计算能力需求[2]。Intel 公司在 Pentium 处理器中最早采用了支持 MMX[3]的 SIMD 技术,一直不断升级至现在的 AVX2 技术。ARM 公司推出的 Cortex-A 系列处理器

收稿日期:2020-11-17　　**修回日期**:2021-03-17
基金项目:河南省高校重点科研项目(18B520044);河南省科技攻关项目(182102210526)
作者简介:裴　航(1994-),男,硕士研究生,CCF 会员(C6556G),研究方向为高性能计算;王　磊,博士,教授,CCF 会员(12516M),研究方向为高性能计算。

采用了 NEON 技术[4]。如今越来越多的厂商都在处理器内集成、升级 SIMD 扩展部件,很多应用程序都在寻求 SIMD 并行,以增强移动终端的多媒体处理能力。

申威[5]是国内自主研发的通用处理器,提供 256 位向量寄存器且包括支持 256 位向量访存及运算的 SIMD 扩展指令。对于某些处理器体系结构,SIMD 扩展指令允许将原来需要多次装载的内存中地址连续的数据一次性装载到向量寄存器中。使用一条 SIMD 扩展指令实现对 SIMD 向量寄存器中所有数据元素的并行处理,这种执行方式非常适合于处理计算密集、数据相关性少的多媒体程序,如音视频编解码[6]。

H. 264[7]视频编码格式是目前发展比较成熟的编码格式。常用的开源解码器有 JM[8]、T264[9]和 FFmpeg[10],目前很多国产平台都对 FFmpeg 视频解码器针对各个平台特点做了相应的优化。文献[11]针对视频解码时 CPU 占用率高,专用硬件解码耗时长等问题,研究了飞腾平台上 GPU 硬件加速视频解码技术,设计了基于 VDPAU 硬件解码接口的 GPU 硬件视频解码方案。文献[12]在实现 FFmpeg 解码器到国产龙芯 3B 平台的移植的基础上,并通过龙芯 3B 所支持的向量扩展指令,对 FFmpeg 解码器的各个关键模块代码进行了向量优化。文献[13]结合飞腾多核 DSP 同构多核架构以及共享存储的特点,采用面向共享存储系统结构的工业标准 Open MP,通过设立并行域以及设置私有变量对解码程序有针对性地进行了并行优化。

文中基于国产自主可控平台申威 421,在对 H. 264 解码器的开源程序 FFmpeg 移植后,H. 264 视频解码效率低,高分辨率的视频播放不流畅甚至卡顿,严重影响国产平台桌面用户体验,阻碍了申威处理器的市场化进度。针对此现象,文中充分利用了申威 SIMD 扩展部件对多媒体程序进行向量化并行。H. 264 解码器性能显著提升,解决了视频播放不流畅问题,推动了国产申威处理器市场化的发展。

1 基于 FFmpeg 的 H. 264 解码器

H. 264 是国际标准化组织(ISO)和国际电信联盟(ITU)共同提出的继 MPEG4 之后的新一代数字视频编解码标准。最大的优势是具有比较高的数据压缩比率,在相同图像质量的条件下,H. 264 的压缩比是 MPEG-2 的 2 倍以上,是 MPEG-4 的 1. 5 ~ 2 倍。H. 264 视频编解码标准没有明确规定如何实现编解码器,但规定了编码视频比特流的语法和比特流的解码步骤方法[14]。

本研究主要基于 FFmpeg 的 H. 264 解码器进行优化。FFmpeg 是一套基于 Linux 操作系统的开源编解码解决方案,它集成了音视频录制、格式转换和编解码器,也可在大多数操作系统中编译和使用[15]。它不仅包含各种音频和视频编解码库,还支持基于流媒体的实时流传输。

FFmpeg 中 H. 264 解码器的解码流程如图 1 所示。首先通过函数 h264_decode_frame 解码一帧图像数据,其中 decode_nal_units 是用于解码 NALU 的函数。decode_nal_units 首先调用 ff_h264_decode_nal 判断 NALU 的类型,然后根据 NALU 类型的不同调用不同的解析函数和解码函数,解析函数包括解析 SPS/PPS/SEI 以及 slice_head。ff_h264_execute_decode_slices 用于解码获取图像信息,ff_h264_execute_decode_slices 调用了 decode_slice 函数,在 decode_slice 函数中完成了熵解码、变换与量化、宏块解码、环路滤波等解码的细节工作。

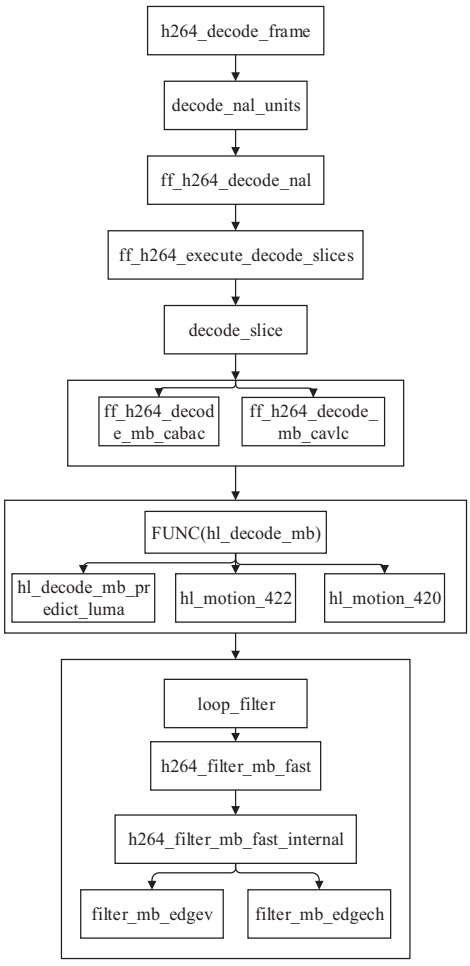

图 1 H. 264 解码器解码流程

熵解码部分的功能是读取数据(宏块类型、参考帧、运动矢量、残差等),并按照 H. 264 的语法和语义规则将其分配到 H. 264 解码器中相应的变量中。其中熵解码方面的函数有 ff_h264_decode_mb_cabac 和 ff_h264_decode_mb_cavlc。分别用于解码 CABAC 编码方式的 H. 264 数据和 CAVLC 编码方式的 H. 264

数据。

根据熵解码后各宏块的信息,函数 hl_decode_mb 会根据宏块类型的不同做处理,主要针对帧间或者帧内宏块。如果是帧间预测宏块,调用函数 hl_motion_422 或者函数 hl_motion_420 进行四分之一像素运动补偿;如果是帧内预测宏块,就会调用 hl_decode_mb_predict_luma 进行帧内预测。经过帧内预测或者帧间预测步骤之后,就得到了预测数据。最后 hl_decode_mb 会调用 hl_decode_mb_idct_luma 等几个函数对残差数据进行 DCT 反变换,并将变换后的数据叠加到预测数据上,形成解码后的图像数据。

Decode_slice 在完成宏块解码后,调用 loop_filter 函数进行环路滤波工作。环路滤波主要是为了去除图像变换量化后产生的块状视觉效应。loop_filter 函数遍历该行宏块中的每个宏块,并且针对每一个宏块调用 ff_h264_filter_mb_fast。函数 ff_h264_filter_mb_fast 中 h264_filter_mb_fast_internal 完成一个宏块的环路滤波工作。该函数分别调用 filter_mb_edgev 或 filter_mb_edgeh 对亮度垂直或边界进行滤波,和调用 filter_mb_edgecv 或 filter_mb_edgech 对色度的垂直或水平边界进行滤波。

2　H.264 关键模块的向量化

文中在对 H.264 解码器关键模块进行向量化并行时首先对热点函数中的循环进行向量化分析,分析循环内是否具有数据相关性、对齐、连续等条件。如果满足向量化分析,使用 SIMD 指令进行替换。如果不满足,分析是否可以通过循环变换数组重组等方法来满足向量化分析,然后通过内嵌汇编进行向量指令替换。最后对向量化后的程序性能进行测试,如果没有性能提升,继续对下一个循环进行分析。

2.1　热点函数分析

perf 是内置于 Linux 内核源码中的性能剖析工具。基于事件采样和性能事件的原则,它支持处理器和操作系统相关性能指标的性能配置文件,并可用于查找瓶颈和定位热点代码。本研究针对视频序列 foreman,使用 perf 工具列出了 H.264 解码器在在整个视频解码过程中所占比重较大的热点函数,如表 1 所示。

表 1　H.264 解码器热点函数

模块	函数	比重
帧内预测	pred 系列	5.4
运动补偿	h264_qpel 系列	26.6
DCT 变换	h264_idct 系列	15.5
环路滤波	h264_loop_filter 系列	8.22

热点函数为应用程序中耗时较多,执行次数较多的函数,对热点函数进行优化相对来讲性能提升比较明显。通过性能分析工具可以发现热点函数主要集中在宏块解码的帧内预测、运动补偿、环路滤波等模块。上述函数几乎占据整个解码过程的 55%,因此对上述函数中的循环做向量化,解码器性能理论上可以得到显著提升。

2.2　热点函数向量化并行

热点函数中 h264_qpel 系列函数用于对 1/4 像素内插做运动补偿,其中 h264_qpel_lowpass 类函数完成了半像素内插函数的初始化工作,主要对 2×2、4×4、8×8 大小的图像方块通过水平或垂直方向滤波计算半像素。H.264 标准采用了一个 6 抽头的有限脉冲响应滤波器(finite impu1se response,FIR)进行计算,滤波器系数为(1/32,-5/32,5/8,5/8,-5/32,1/32),虽然 6 抽头滤波器比较复杂,但可以明显改善运动补偿性能[16]。半像素插值计算如图 2 所示,半像素点(如 b,h,m)通过对相应整像素点(图中灰色方块)进行 6 抽头滤波得出。半像素 b 的计算可由同一行中的整像素 E、F、G、H、I、J 得出。计算公式见式(1),其中 CLIP 用于将取值限幅在 0 ~ 255。

$$b = \mathrm{CLIP}[(E - 5F + 20G - 20H + 5I + J + 16) >> 5] \quad (1)$$

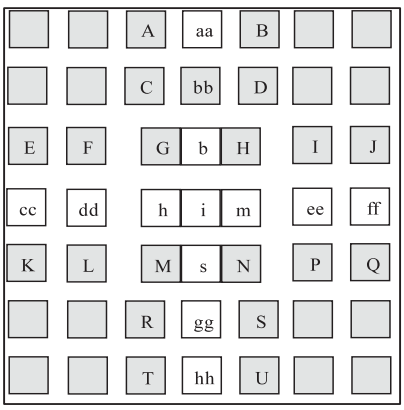

图 2　1/2 像素插值

对于 1/2 像素插值计算,由于计算需要像素每行的存储规则,且像素间不存在数据相关性,这种计算密集,复杂度较高的运算十分适合利用 SIMD 扩展部件对其进行向量计算,使用申威 SIMD 扩展指令集可以

一次性装入一行整像素点。通过向量计算得到每行的 1/2 像素点,循环 8 次就可以计算出 8×8 块的 64 个 1/2 像素点。8×8 图像块的每行的 1/2 像素点的 SIMD 计算如图 3 所示。

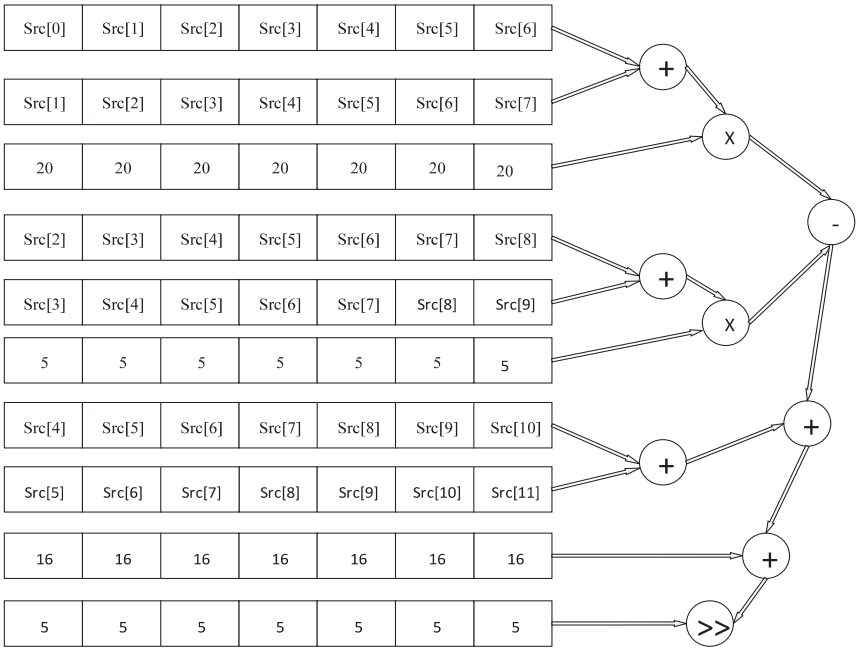

图 3 1/2 像素插值 SIMD 计算

申威扩展指令集提供 VLDW_U/VSTW_U 非对齐装入/存储指令,可以从存储器中取出 8 个 32 bit 或 32 个 8 bit 元素到 256 位向量寄存器,并支持不同长度的加/减移位等运算。8 bit 的像素可以一次装入 32 个元素到 256 位向量寄存器/存储器中,但对于 8×8 的图像块的 1/2 像素点的计算,只需使用向量寄存器的 64 位,每次进行 8 个元素的运算。函数 put_h264_qpel8_h_lowpass 优化前和优化后的代码如下所示:

优化前:

```
for(i=0; i<h; i++)
{
dst[0]=CLIP((((src[0]+src[1])*20-(src[-1]+src[2])*5+(src[-2]+src[3]))+16)>>5);
dst[1]=CLIP((((src[1]+src[2])*20-(src[0]+src[3])*5+(src[-1]+src[4]))+16)>>5);
......
}
```

优化后:

```
for(i=0; i<h; i++)
{
vldw_u $f1,0(a0)
vldw_u $f2,0(a1)
vldw_u $f3,1(a1)
vldw_u $f4,-1(a1)
vldw_u $f5,2(a1)
vldw_u $f6,-2(a1)
vldw_u $f7,3(a1)
ldwe $f10,0,(a2)
vaddw $f2, $f3, $f2
vsllw $f2,4, $f8;
vsllw $f2,2, $f9
vaddw $f8, $f9, $f8
vaddw $f4, $f5, $f4
vsllw $f4,2, $f9
vaddw $f9, $f4, $f9
vsubw $f8, $f9, $f8
vaddw $f6, $f7, $f6
vaddw $f8, $f6, $f8
vaddw $f8, $10, $f8
vsrlw $f8,5, $f8
vstw_u $f8,0(a0)
}
```

首先从存储器中取出以原像素点 src[0]为起始的 8 个整像素点放置向量寄存器中,然后分别与依次扩展 1 位的连续 8 个整像素点进行向量运算。其中乘法直接用加法和移位来代替,最后将计算的 8 个结果分别放置 8 个目标分像素点,完成一行的分像素点计算。接着增加步长进行下一行的分像素点计算,循环 8 次完成 8×8 块的分像素计算。使用申威 SIMD 指令进行替换,一条 SIMD 指令可以同时进行 8 次相同运算,计算出每行的分像素点,有效提升了分像素插值的效率。

对于 4×4、2×2 块的水平或垂直方向的 1/2 像素插值计算向量化并行同 8×8 块计算方法相似,只是 4×4 块对于每行 1/2 像素点的计算只能一次进行 4 个像素点计算。而对于双向预测的 1/2 像素点的计算更加复杂一些,同样适合使用 SIMD 指令来进行向量化并行计算 1/2 插值点。

在 H.264 关键模块算法中都存在满足向量化分析的循环,对此可以直接进行向量指令替换。经过 1/4 像素运动补偿的帧间或者帧内预测后,就需要将残差数据进行 DCT 反变换,DCT 系列函数中 ff_h264_idct8_dc_add_8c 对只有 DC 系数的 8×8 矩阵进行整数 DCT 反变换。最内层循环通过一条向量指令 vaddw 就可以实现对 8×8 像素每行与 dc 系数的累加来代替最内层的 8 次循环,降低了循环开销。相对比较复杂的 4×4 整数 DCT 反变换函数 ff_h264_idct_add_8_c,先对每列进行一维整数变换,然后对经过列变换块的每行进行一维整数变换,最后将变换后的残差数据叠加来实现传统整数变换的矩阵乘法。由于数据是以行来存储的,所以使用 SIMD 扩展指令可以先实现一行元素的整数变换,经过四次向量计算完成整个块的纵向一维整数变换。再对经过纵向一维整数变换的块进行另一维的向量整数变换,最后通过向量指令进行残差数据叠加。而对于环路滤波中 h264_loop_filter 类函数由于循环内数组访问不连续以及存在控制流,不能直接向量化,需要对数组进行重组,然后进行向量化。

3　实　验

3.1　实验环境

本次实验平台基于国产自主可控平台申威 421,操作系统为 deepin15.5,编译器为申威自主研发编译器 swgcc-5.3.0。实验环境如表 2 所示,为使测试结果具有代表性,实验分别选取了 3 组不同分辨率 720×486、1 600×960、2 048×1 024 的视频序列,视频播放采用 FFmpeg 组件 ffplay。

表 2　实验环境

处理器	操作系统	编译器	核心	处理器频率/GHz	内存/G	显卡
sw421	deepin15.5	swgcc-5.3.0	4	1.8	8	Ich2

3.2　实验结果分析

基于移植到申威平台的 H.264 解码器有很大性能提升。表 3 表示对各个热点函数向量化与向量化后的性能对比,表 4 表示 3 个视频序列在对解码器中关键模块向量化后的性能对比。

表 3　各函数向量化前后性能对比

函数	Football			Mit			Surfside		
	优化前	优化后	加速比	优化前	优化后	加速比	优化前	优化后	加速比
avg_h264_qpel8_v_lowpass	3.68	1.47	2.5	4.11	1.79	2.3	4.56	2.06	2.21
put_h264_qpel8_h_lowpass	3.92	1.15	3.4	4.19	1.34	3.12	4.88	1.57	3.1
ff_h264_idct_dc_add_8_c	2.36	1.31	1.8	2.65	1.46	1.82	2.81	1.54	1.83
put_h264_qpel4_hv_lowpass	3.17	2.44	1.3	3.65	2.92	1.25	3.99	2.31	1.73
h264_v_loop_filter_chroma_8_c	3.55	2.96	1.2	4.20	3.41	1.23	4.73	3.64	1.3
put_h264_chroma_mcXY_8_c	0.63	0.57	1.1	0.62	0.59	1.05	0.59	0.49	1.2

表 4　测试视频向量化前后性能对比

视频序列	分辨率	帧数	优化前		优化后		性能提升/%
			CPU 占用率/%	解码时间/s	CPU 占用率/%	解码时间/s	
Football	720×486	300	35	278	31	189	35.3
Mit	1 600×960	300	41	312	37	226	27.6
Surfside	2 048×1 024	300	57	465	46	334	28.2

通过实验对比,对热点函数中满足向量化分析的循环进行向量指令替换,增加了指令吞吐率,在保证原视频清晰度的基础上,视频解码性能有大幅度提升。其中函数 put_h264_qpel8_h_lowpass 向量化后加速比达到 3.4,而函数 put_h264_chroma_mcXY_8_c 中由于对未知变量的乘法,需要对系数进行判断再执行移位运算来代替乘法继续向量运算,向量化指令更加复杂,加速比只有 1.1。测试的 3 个视频序列中 football 的解码性能最高有 35.3% 的提升。并且 CPU 占用率明显降低,释放了 CPU 资源,2 048×1 024 分辨率高的视频播放流畅度更高,提高了国产申威平台用户的多媒体视觉体验。

4　结束语

本次研究针对 H.264 解码器中的热点函数使用 SIMD 扩展部件来进行向量化计算,提升解码器性能。但是由于并没有充分利用到 256 位向量寄存器,向量重组等技术存在一定开销,向量化效果并没有达到理

想的提升。接下来将会针对热点函数中由于数组非对齐及存在数据依赖等程序进行重点分析,充分利用 256 位向量寄存器,更深层次地挖掘数据的并行性。随着高性能计算的发展,SIMD 扩展部件仍然是程序加速的重要手段,SIMD 扩展部件结构简单、功耗低、加速效果明显,不需要增加通信以及 cache 和内存的开销,在多媒体领域,数字信号处理中都起着关键作用。而如何充分利用 SIMD 扩展部件更深层次地挖掘数据并行性,提升程序性能,是长久的工作重心。

参考文献:

[1] ROBERTSÉN F, MATTILA K, WESTERHOLM J. High-performance SIMD implementation of the lattice-Boltzmann method on the Xeon Phi processor[J]. Concurrency, Practice and Experience, 2019, 31(13): e5072.1-e5072.16.

[2] 周 锋,彭元喜,王耀华. SIMD 结构中多指令流扩展的设计与实现[C]//第十九届计算机工程与工艺年会暨第五届微处理器技术论坛论文集. 哈尔滨:中国计算机学会计算机工程与工艺专业委员会,2015:7.

[3] AMIRI H, SHAHBAHRAMI A. SIMD programming using Intel vector extensions[J]. Journal of Parallel and Distributed Computing, 2019, 135(2): 134-135.

[4] LINDOSO A, GARCIA-VALDERAS M, ENTRENA L. Analysis of neutron sensitivity and data-flow error detection in ARM microprocessors using NEON SIMD extensions[J]. Microelectronics Reliability, 2019, 100-101: 113346.

[5] 施光源. 基于国产申威处理器的自主可控产品实践之路[J]. 网络空间安全,2018,9(7):78-81.

[6] 高 伟,赵荣彩,韩 林,等. SIMD 自动向量化编译优化概述[J]. 软件学报,2015,26(6):1265-1284.

[7] WANG Ping, CHENG Hao. Early termination of intra mode decision based on most probable mode and SATD for H.264/AVC encoding[J]. Journal of Signal Processing Systems, 2020, 92(2): 173-186.

[8] 王彩霞. 基于 JM 模型的 H.264 解码器的优化研究[J]. 计算机与信息技术,2012,20(3):34-37.

[9] 王 娇. 基于 T264 的编码器在多核异构处理器上的设计与实现[D]. 成都:西南交通大学,2015.

[10] 阿不都克里木·玉素甫,王亮亮. 基于自主可控平台的 FFMPEG 在线视频转换系统[J]. 计算机与现代化,2020(1):81-84.

[11] 刘 敏. 国产飞腾处理器的视频解码技术研究[D]. 长沙:国防科学技术大学,2014.

[12] 裴晓航,何颂颂. 基于龙芯 3B 的 H.264 解码器的向量化[J]. 电子技术,2010,37(10):88-90.

[13] 王子聪,扈 啸,郭 阳. 面向飞腾多核 DSP 的 H.264 视频解码的 OpenMP 并行优化[C]//第十八届计算机工程与工艺年会暨第四届微处理器技术论坛论文集. 贵阳:中国计算机学会计算机工程与工艺专业委员会,2014:5.

[14] 毕厚杰. 新一代视频压缩编码标准—H.264/AVC[M]. 北京:人民邮电出版社,2005.

[15] 胡 聪,周 甜,唐璐丹. 基于 FFMPEG 的跨平台视频编解码研究[J]. 武汉理工大学学报,2011,33(11):139-142.

[16] 梁 彬. 基于 ARM Cortex-A8 平台的 H.264 解码器移植与优化[D]. 武汉:华中科技大学,2012.