基于 Cortex 嵌入式多处理器系统的图像中值滤波算法并行化的研究①

廖文献[1], 黄兴利[2]

[1](浙江工贸职业技术学院 信息传媒学院, 温州 325003)
[2](西北工业大学 自动化学院, 西安 710072)

摘 要: 嵌入式系统在图像处理、空间计算等领域越来越广泛, 如何在功耗、成本和计算能力三个主要方面取得平衡, 利用多核和多处理器系统以并行计算方式提高嵌入式系统计算能力是一种有效的解决方案. 讨论了基于 Cortex 嵌入式多处理器系统的基本结构, 并在该系统上进行图像中值滤波算法的并行化研究. 实验结果分析表明, 在该嵌入式多处理器平台上配合并行算法能够成倍提高图像中值滤波的运行性能.

关键词: Cortex 架构; 多处理器系统; 中值滤波; 并行算法

Research on Parallel Image Median Filtering Algorithm for Multi Processor Embedded System Based on Cortex

LIAO Wen-Xian[1], HUANG Xing-Li[2]

[1](College of Information and Communications, Zhejiang Industry&Trade Vocational College, Wenzhou 325003, China)
[2](School of Automation, Northwestern Polytechnical University, Xi'an 710072, China)

Abstract: Embedded system has become more and more popular in image processing, spatial computing and other fields. In order to achieve a balance in three major aspects of power consumption, cost and computing power, it is an effective solution to improve the computing ability of embedded system, using the multi-core and multi processor system based on parallel computing method. This paper discusses the basic structure of embedded multi processor system based on Cortex, and has a research on parallel image median filtering algorithm on this system. Experimental results show that the parallel algorithm can improve the performance of image median filtering exponentially on the embedded multi processor platform.

Key words: Cortex; multiprocessor system; median filter; parallel algorithm

1 引言

计算机单机技术发展的有限性和各种计算需求的无限性, 决定了计算技术发展必然走上多机并行的道路. 计算机发展经历了数代的发展, 其中三、四代在体系结构上以 Intel、IBM 为代表, 他们推出了多个系列的多核处理器产品, 包括桌面应用、高性能计算等[1,2], 当前绝大部分高性能计算机系统、超级计算机系统大多以这两家的多核处理器为基础构建. 而在四代到五代计算机体系的发展中, 嵌入式多核、多处理器系统因其自身的特点在空间图像处理、空间计算等需要高性能、高可靠性、低功耗处理器领域日益受到关注. 特别是 SoC 技术的飞速发展为芯片级并行处理计算平台的研究提供了坚实的技术基础.

数据处理, 特别是图像及视频数据处理逐步从后端密集型处理及应用向前端实时处理发展, 如实时图像处理、行为分析、目标跟踪等民用和军事应用. 常规嵌入式计算单元计算能力有限, 为了保持较低功耗的前提下提高计算性能, 采用多核并联处理成为优先

① 基金项目:浙江省自然科学基金(LY14F020030)
收稿时间:2016-06-07;收到修改稿时间:2016-07-14 [doi:10.15888/j.cnki.csa.005624]

解决方案. 文献[3]提出了一种在多核 SOC 上的 MPEG4 解码算法; 文献[4]提出了一种面向硬件的分区方法, 解决嵌入式多处理器 FPGA 系统的分区问题, 并通过 JPEG 编码系统验证了方案的可行性; 文献[5]设计了一种基于嵌入式的图像和视频处理应用系统以替代基于 PC 的应用系统; 文献[6]讨论了嵌入式处理器 TTA 的效率和能耗问题, 并对异步函数及相关控制指令进行了优化; 文献[7]讨论并提出了一种新的嵌入式异构多核处理器通讯机制, 比经典模型提高 23%的效率; 文献[8]为低功耗嵌入式处理器提出双重数据缓存体系结构, 通过这种合作缓存系统, 在提升系统性能的同时奖励的系统功耗, 并在三星电子芯片上进行了验证.

本文在 ARM 公司 CortexM3 架构的系统上设计并实现了一种主从式多处理器并行处理系统[9], 在该平台针对图像处理中广泛应用的中值滤波算法进行了并行化研究与实现, 通过测试验证了系统的有效性, 该系统基本能够满足动态图像处理的及时性要求.

2 嵌入式多处理器系统与滤波算法

基于 Cortex 的嵌入式多处理器系统: 实现系统选用 ARM 公司 CortexM3 架构, 使用多片处理器组成主-从处理结构, 其中四个从机构成并行结构, 主机负责任务调度和必要的任务处理, 系统结构简图如图 1 所示.

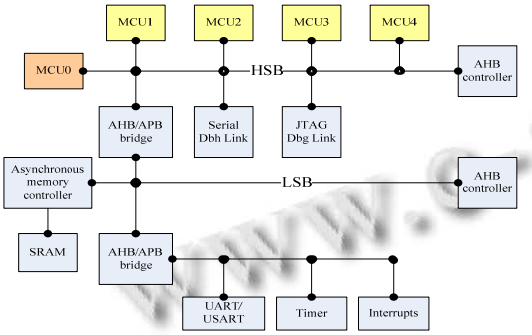


图 1 多处理器系统结构简化图

主机和从机通过 AHB 高速总线连接, 处理器之间切换通过互斥方法实现; 此外他们共享系统存储空间(这里主要是片内 SRAM 和外扩), 这样也可以节省数据传输的时间, 但需要注意的是片内存储空间属于处理器的私有空间, 不允许共享访问, 因此在执行后文中提到的并行算法时需要将共享存储区中的数据拷贝到私有存储区. 如果将多个主机通过网络连接起来就可以构成群集[10].

图像中值滤波算法: 图像经常被各种噪声污染, 一般使用平滑空间滤波器对其进行处理, 中值滤波算法是一种统计排序(非线性)滤波器算法, 能够有效抑制图像中的(椒盐)噪声[11], 可以有效抑制尖锐和长拖尾噪声同时能够保护信号中尖锐的跳变或边界. 该算法基本过程如下:

设图像由 $M\times N$ 个像素构成, $f[x,y]$ 是输入图像在坐标 $[x,y]$ 处的像素, 以该像素为中心设定一个 $m\times n$ 窗口 W, 按照从上向下, 从左至右逐一扫描每个象素, 按灰度值大小将窗口内的像素排序, 找到中间像素并利用该像素的值替换 $f[x,y]$, 用公式表示为:

$$G[x,y]=Median\{f[x-i,y-j]\},[i,j]\in W \quad (1)$$

基本的中值滤波所需的时间复杂度为 $O(M\times M\times m\times n)$, 由于主要涉及窗口排序和像素替换, 因此要减少中值滤波的总计算时间, 可采用两种方法: 静态分割算法和任务级并行处理算法.

3 并行算法设计

目前常见的中值滤波并行算法可以称为静态分割(或静态负载分割)算法[12], 通过将单个任务分解为多个子任务并行处理, 从而提高处理效率, 这里将简单说明该算法; 提出一种类似于处理器机流水线作业的任务级并行算法, 并非将单个任务简单分割, 而是将多个任务以流水线作业的方式分别调度到多个处理器上并行运转.

因此可以定义任务模型: 设定处理器数量为 n; 任务规模为 $F=M*N$; 中值滤波模板简化为正方形扫描窗口, 大小为 W.

3.1 静态分割算法

静态分割在主机端将待处理的图像 F 分割成若干份, 分别发送到从机进行处理, 待处理完成后将数据汇总为 F', 其基本处理流程图如图 2 所示.

静态分割算法详细处理过程为:

1) 第 $k^{\#}$ 进程($k=0,1,\cdots,n-2$): 得到矩阵 F', 根据中值滤波算法, 计算子矩阵 $G_y\left[kq\cdots((k+1)q-1)\right]$ 共 q 行结果.

2) 第 $(n-1)^{\#}$ 进程得到矩阵 F'' 后, 计算子矩阵 $G_y\left[(n-1)q\cdots(N-1)\right]$ 共 $(q+r)$ 行结果.

3) 步骤 1)和 2)中对于模板扫描到的数据使用处

理器缓存存放，使用冒泡法排序计算中值，因此每台处理机平均需要 $\frac{NM}{n}w^2$ 个赋值运算步和 $\frac{NM^2}{n}$ 个排序(w^2 个数值的排序)运算.

4) 为了将任务分派到所有的从机并回收结果，因此需要 $2n$ 个通讯步；通信复杂度为 $O(nNM)$.

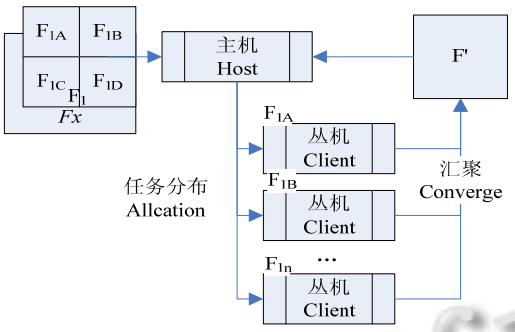


图 2　静态分割算法流程图

根据上式可计算整个处理过程总通信量为:

$$C_{sta}(n)=nNM+(n+1)qM+(q+r)M \\ =nNM+NM=(n+1)NM \tag{2}$$

3.2 任务级并行处理算法

任务级并行算法与静态分割算法类似，整个处理过程也主要分为三个部分，其最大区别在于任务级并行算法并不需要对图像本身进行分割，而是将整个待处理帧发送给空闲处理机处理，这与流水线作业处理方式类似，该处理过程如图 3 所示.

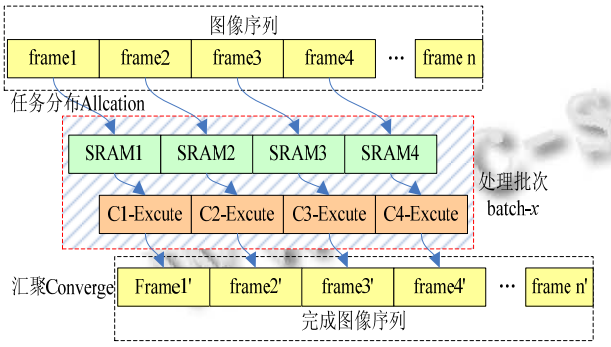


图 3　类流水线作业的任务级并行处理过程示意图

主机会根据从机书否空闲决定将待处理图像序列 *frame*l-*n* 中的帧分派对应的处理机，此时将该图像帧写入处理机对应的数据缓存中 SRAAM*x*，然后通知该处理机处理任务，按照此流程依次将剩余图像帧分派给其它空闲的从机，如果从机处理单个任务的耗时刚好与任务到达并写入缓存的延迟一致则可以形成一次批处理(batch)，然后按照这个处理过程形成下一次批处理. 该处理流程伪代码:

```
ServerProcess(){
Create_ClientProcess();
if(TasksCount <= 0){
   Sleep();// or return      }
 for(ClientId=0;ClientId<ClientCount;ClientId++)
{
   ClientsState=CheckClientsState(ClientId);
   if(ClientsState==0){
        AssignTask(ClientId,TaskId);
   } }
Statistics(GetCompleteInterrupt());   }
ClientProcess(){
   Interrupt = CheckInterrupt();
   if(Interrupt==1)
   {
        Excute(Task);
   }
   SendCompleteInterrupt();
}
```

相比于静态分割算法，在任务级并行算法中任务是作为一个完成的个体传送给处理器运行的，因此减少了数据分割产生的运算部和额外的数据，其平均执行 $NM\times w^2$ 个赋值运算步和 NM^2 个排序运算.

由于需要计算的任务 F 已经位于从机，因此无须再产生调度通讯的开销，仅需要取回结果执行 n 次通讯. 因此通信复杂度为 $O(N)$，总通信量为:

$$C_{dyn}(n)=N \tag{3}$$

相比较而言任务级并行算法较大幅度减少了总通讯量，其性能要优于静态分割算法.

3.3 算法分析

进一步的分别从并行度、加速比和效率三个方面对两种算法进行对比分析:

① 并行度

并行度计算不考虑传输和调度延迟的情况下，处理器的数量为 n，因此理论并行度为 n.

② 加速比和效率

在 n 台处理机可用的情况下，设 t_f 为单个赋值运算的时间、t_{sort} 为 w^2 个数的排序时间，加速比为:

$$S_\infty=\frac{NM\left(w^2t_f+t_{sort}\right)}{(NM/n)\left(w^2t_f+t_{sort}\right)}=n \tag{4}$$

$$E=\frac{S_{\infty}}{n}=1 \tag{5}$$

所以, 两种算法理论上加速比均可达到 n , 效率可达到 1.

③ 系统加速比和系统整体效率

根据前文分析的系统在理想状况下的加速比和系统效率, LogGP 模型分析可得:

$$\begin{aligned} t_i^{(comm)} &= (T_{SR}+7T_w)+n(T_{SR}+NMT_w)+(n-1)[T_{SR}+qMT_w]+[T_{SR}+(q+r)MT_w] \\ &= (2n+1)T_{SR}+[(nN+nq+r)M+7]T_w \\ &= (2n+1)T_{SR}+[(n+1)NM+7]T_w \end{aligned} \tag{6}$$

$$\begin{aligned} S_{opt}^{(sta)} &= \frac{NMw^2t_f+NMt_{sort}}{\max_i\{(\frac{NM}{n}w^2t_f+\frac{NM}{n}t_{sort})+(2n+1)T_{sr}+[(n+1)NM+7]T_w\}} \\ &= \frac{NM(w^2t_f+t_{sort})}{(q+r)M(w^2t_f+t_{sort})+(2n+1)T_{sr}+[(n+1)NM+7]T_w} \end{aligned} \tag{7}$$

表 1 两种算法在不同任务规模下的测试效率表

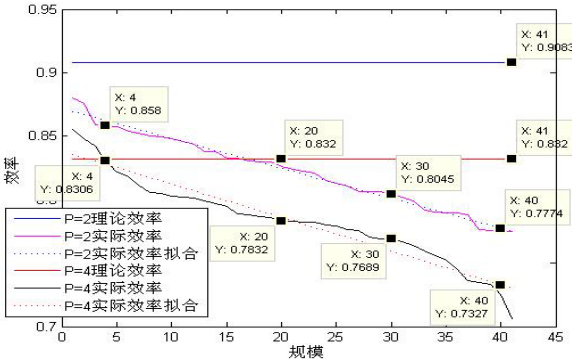
序号	规模 N	处理器数量 P=2				处理器数量 P=4			
		并行		静态		并行		静态	
		理论效率	实际效率	理论效率	实际效率	理论效率	实际效率	理论效率	实际效率
1	64*64	0.975	0.985	0.908	0.880	0.952	1.008	0.832	0.855
3	96*96	0.975	0.978	0.908	0.859	0.952	0.996	0.832	0.842
5	128*128	0.975	0.970	0.908	0.858	0.952	0.963	0.832	0.821
7	160*160	0.975	0.967	0.908	0.853	0.952	0.957	0.832	0.811
9	192*192	0.975	0.964	0.908	0.849	0.952	0.954	0.832	0.805
11	224*224	0.975	0.960	0.908	0.846	0.952	0.939	0.832	0.802
13	256*256	0.975	0.954	0.908	0.838	0.952	0.929	0.832	0.799
15	288*288	0.975	0.946	0.908	0.833	0.952	0.914	0.832	0.795
17	320*320	0.975	0.944	0.908	0.830	0.952	0.903	0.832	0.788
19	352*352	0.975	0.940	0.908	0.830	0.952	0.885	0.832	0.786
21	384*384	0.975	0.938	0.908	0.824	0.952	0.876	0.832	0.783
23	416*416	0.975	0.930	0.908	0.821	0.952	0.870	0.832	0.782
25	448*448	0.975	0.930	0.908	0.813	0.952	0.861	0.832	0.779
27	480*480	0.975	0.929	0.908	0.806	0.952	0.860	0.832	0.776
29	512*512	0.975	0.927	0.908	0.806	0.952	0.856	0.832	0.769
31	544*544	0.975	0.923	0.908	0.802	0.952	0.851	0.832	0.767
33	576*576	0.975	0.919	0.908	0.792	0.952	0.848	0.832	0.759
35	608*608	0.975	0.909	0.908	0.790	0.952	0.836	0.832	0.752
37	640*640	0.975	0.900	0.908	0.788	0.952	0.832	0.832	0.736
39	672*672	0.975	0.886	0.908	0.776	0.952	0.829	0.832	0.733
41	704*704	0.975	0.867	0.908	0.775	0.952	0.789	0.832	0.706

根据前文分析, 静态并行算法的在处理器数量为 2 的时候理论效率为 90.8%, 处理器为 4 的时候效率为 83.2%; 任务级并行算法在处理器数量为 2 的时候理论效率为 97.5%, 处理器为 4 的时候效率为 95.2%.

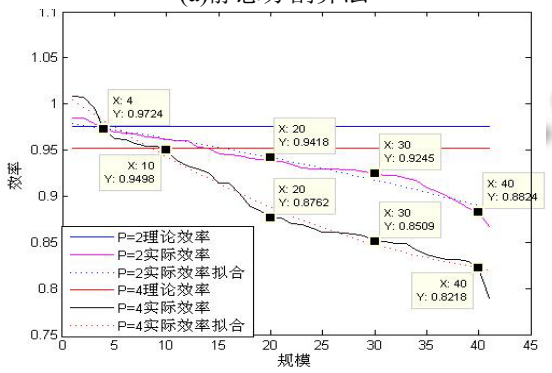
4 算法测试

测试中使用不同尺寸的方形灰度图像, 规模从 64*64 开始, 间隔 16 个单位新增一个规模, 最大测试规模为 704*704 像素灰度图, 两种并行算法和不同处理机数量的测试如表 1 所示. 从以上实验数据中, 可

以得到以下结论:



(a)静态分割算法



(b)任务级并行算法

图 4　并行度 2 和 4 下的效率对比图

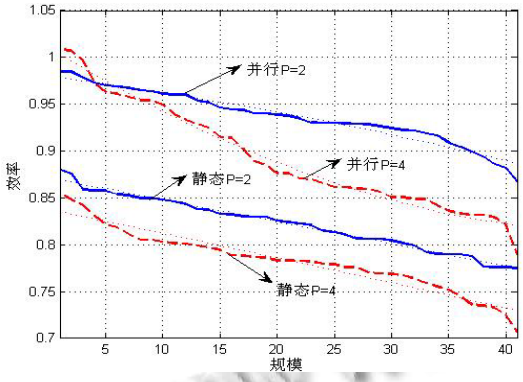


图 5　两种算法分别在并行度 2 和 4 下的效率对比图

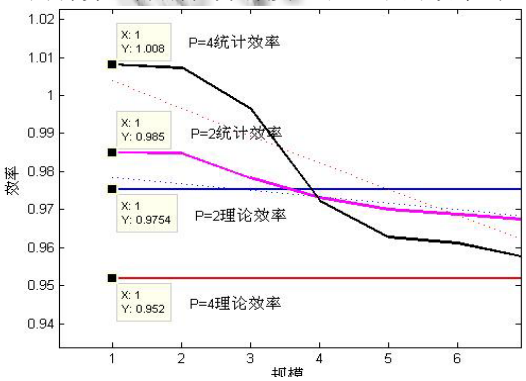


图 6　实际统计的效率局部放大图

① 在任务规模较小时, 4 个处理器的效率高于 2 个处理器;

② 在任务规模较小时, 处理器的效率甚至超过了理论效率——主要是因为统计时间不精确造成的, 这一点与静态分割算法是相同的, 如图 6 的局部放大所示;

③ 随着任务规模的逐步增加, 处理器效率都呈下降趋势, 特别是在末端的时候下降较快, 这是因为任务的规模在“规模 41”的时候为 704*704 合计约 500KB 的数据, 远超过处理器内部缓存的大小, 这将导致更多的数据交换和复制过程, 同时也会消耗较多的调度时间, 因此其性能会较快下降;

④ 两类算法中处理器的增加并没有带来效率的提升, 从总体上来看处理器的增加导致调度开销较快增长, 反之效率下降明显;

⑤ 静态分割算法需要向从处理器拷贝待处理图像, 而任务级并行算法则利用类似于流水线处理机制无需额外的数据拷贝, 因而静态分割算法效率整体低于任务级并行处理算法;

⑥ 以上实验数据中规模为 512*512 的图像处理时间都超过了 1000ms, 其时效性很低, 这是因为默认采用非压缩格式的 BMP 图像, 同时滤波算法部分也未作任何优化. 针对中值滤波算法前人做了大量的研究包括基于快速排序的 FSMF 算法[13], 可以将时间复杂度降低到 O(NlnN); NSMF 可以将 3x3 模板的比较次数降低到 30 次[14]; 基于统计思想的 SMF 算法[15]以及 FMF 算法[16]仍然需要 21 次比较; 文献[17]仅需要 18 次比较, 则可以大幅度降低算法消耗的时间. 分别采用以上算法后优化, 对于规模为 640*480 的图像处理结果如表 2 所示.

表 2　多种优化算法并行处理结果

规模 N	串行时间 (ms)	并行(P=2)				并行(P=4)			
		总时间	计算时间	加速比	效率	总时间	计算时间	加速比	效率
冒泡法	4259	2261	2204	1.932	0.966	1189	1127	3.779	0.945
NSMF	3552	1890	1841	1.929	0.965	983	923	3.848	0.962
FMF	2490	1351	1285	1.938	0.969	699	640	3.891	0.973
快速计算	2131	1169	1103	1.932	0.966	606	547	3.896	0.974

可以看到, 经过算法优化后的计算时间大幅度降低, 其中快速计算在并行度为 4 的时候计算时间降低到 500ms, 相比串行算法已经快了 8 倍; 如果在再使用

BMP 压缩算法, 至少可以使得需要处理的数据量至少减少到 10~20%(典型压缩比远超 500%), 则该系统基本能够满足动态图像处理的及时性要求.

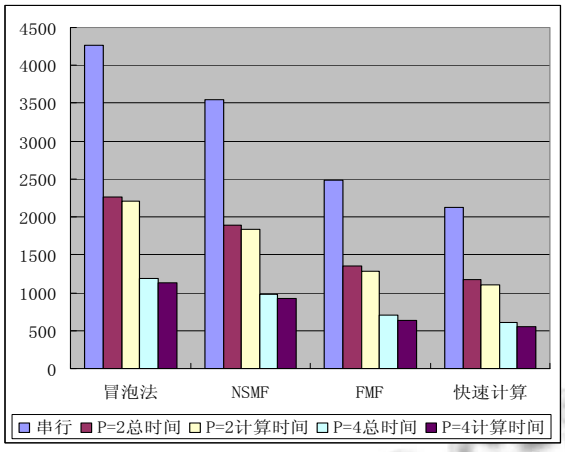


图 7 多种算法并行计算时间的柱状图

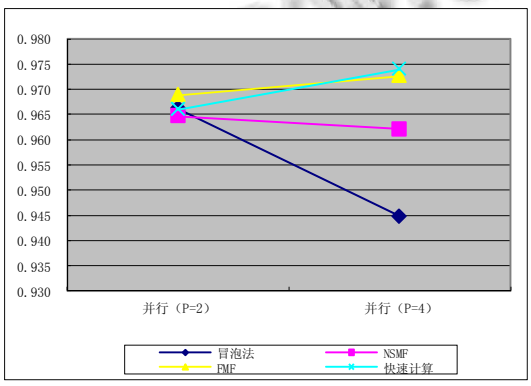


图 8 多种算法在不同并行度时的效率

5 总结与展望

设计了一种基于嵌入式系统的同构并行处理系统, 并选择图像中值滤波算法进行并行化处理, 在基础的静态任务分割基础上针对平台特性提出了任务级并行处理算法, 并针对这两种算法在不同任务规模和并行度下进行了测试, 验证了系统的可行性和实际性能.

CortexM3 架构的嵌入式系统其运算能力约为 1.25DMIPS/MHz, 一般出于功耗控制, 其主频大多在 100MHz 以内, 即运算能力在 125DMIPS 左右, 可以满足简单计算的需要, 当面对更复杂的任务如模式识别、深度学习时运算能力则有所欠缺; NVIDIA 于 2014 年第一季度发布 Jetson TK1 嵌入式超算平台功耗仅 10w, 而计算能力达到了 326GFLOPS, 配合 CUDA 能够更好的发挥并行计算系统的运算能力, 代表着未来低功耗高性能计算的一种发展趋势.

参考文献

1 McNairy C, Bhatia R. Montecito: A dual-core, dual-thread itanium processor. IEEE Micro, 2005, 25(2):10–20.

2 Kalla R. IBM power5 chip: A dual-core multithreaded processor. IEEE Micro, 2004, 24(20): 40–47.

3 Mladen B, Hans JS, Peter P. Multicore system-onchip architecture for MPEG-4 streaming video. IEEE Trans. on Circuits and Systems for Video Technology, 2002, 12(8): 688–699.

4 Lee TY, Fan YH, Cheng YM, et al. Hardware oriented partition for embedded multiprocessor FPGA system. Proc. of the 2nd International Conference on Innovative Computing, Information and Control (ICICIC 2007). Kumamoto, Japan. 2007.

5 Toledo F, Martinez J, Ferrandez J. FPGA-based platform for image and video processing embedded systems. Proc. of the 2007 3rd Southern Conference on Programmable Logic (SPL'07). 2007. 171–176.

6 Li Y, Wang ZY, Zhao XM, et al. Design of a low-power embedded processor architecture using asynchronous function units. Lecture Notes in Computer Science, 2008: 354–363.

7 Yan LK, Shi QS, Chen TZ, et al. An on-chip communication mechanism design in the embedded heterogeneous multi-core architecture. Proc. of the 2008 IEEE Internation Conference on Networking, Sensing and Control(ICNSC). 2008. 1842–1845.

8 Park GH, Lee KW, Han TD, et al. Cooperative cache system: A low power cache system for embedded processor. IEICE Trans. on Electronics, 2007, E90-C(4): 708–717.

9 李哲,慕德俊,郭蓝天,黄兴利,李刘涛.嵌入式多处理器系统混合调度机制的研究.西北工业大学学报,2015,01:50–56.

10 张天凡.基于 Cortex 嵌入式架构的智能储物管理系统[硕士学位论文].武汉:武汉大学,2012:10–12,33,56–60.

11 Gonzalez RC, Woods RE. Digital Image Processing. Prentice Hall, 2010: 178–179.

12 董付国,范辉,原达.一种新的图像中值滤波并行算法.计算机科学,2007,34(12):99–101.

13 张丽,陈志强,高文焕.均值加速的快速中值滤波算法.清华大学学报,2004,44(9):1157–1159.

14 朱捷,朱小娟,贺明.基于 FPGA 的实时性图像处理器中值滤波器设计实现.计算机测量与控制,2007,15(6):789–800.

15 李元帅,张勇,周国忠,等.图像中值滤波硬件算法及其在 FPGA 中的实现.计算机应用,2006,26(6):62–63.

16 董付国,原达,王金鹏.中值滤波快速算法的进一步思考.计算机工程与应用,2007,43(26):48–49.

17 王宇新,贺圆圆,郭禾,龙珠.基于 FPGA 的快速中值滤波算法.计算机应用研究,2009,26(1):224–226.