基于 GPU 的瑕疵检测

徐志鹏，须文波

(江南大学信息工程学院，无锡 214036)

摘　要：提出了使用廉价的图形卡来实现基于 SOM 的图像处理算法，达到检测产品瑕疵的目的。算法基于 GPU 的 fragment shader 程序，并使用了最新出现的浮点缓冲区技术，使算法达到了与 CPU 运算一致的精度，并对检测算法进行了优化。在廉价的硬件上实现了 5.6 帧/s 的处理速度。

关键词：瑕疵检测；自组织映射；GPU 计算；共生矩阵

Defect Detection Based on GPU

XU Zhipeng, XU Wenbo

(School of Information Technology , Southern Yangtze University, Wuxi 214036)

【Abstract】This paper proposes a new technique for realizing defect detection with the low cost graphics hardware.The algorithm is based on the fragment shader, it adopts the floating point buffer which makes the the same precision as CPU does ,and optimizes the algorithm. It gets a speed of 5.6 fps with cheap hardware.

【Key words】Defect detection; SOM; GPU computing; Co-occurrence matrix

在线瑕疵检测是现代造纸业、纺织业等一个重要的组成部分。检测的目的是对于影响产品质量的瑕疵进行检测和分类。在线检测系统不仅用作质量控制设备，而且可以用作生产设备的工况诊断工具。造纸业的特点是幅值宽、速度快。现代造纸机的幅宽可以超过 9m，速度达到 30m/s。这样的机器每秒钟生产大约 3×10^8 mm^2 的纸张，而且所有的产品必须被 100%地进行检测。

历史上，纸病的检测是利用硬件通过阈值和匹配滤波来实现的。这些技术可以检测最基本的纸病类型。更为复杂但却是严重的纸病的检测仍然不可靠。Jukka[1] 等提出使用统计自组织映射（SOM）来估计特征向量的分布，并且使用 VHDL 实现了纸病的在线实时检测。

本文提出使用非专业市场的图形硬件，来实现 SOM 的在线检测，使造价昂贵的纸病检测系统有可能使用廉价的硬件加以实现。

1 GPU 的特性

高性能的图像处理单元（GPU），如 Geforce FX ，具有强大的浮点运算能力和灵活的编程接口。由于它们具有并行处理的特性，GPU 有望成为未来进行高性能浮点运算的有力工具。GPU 已经被扩展到很多的应用领域。包括矩阵的乘法[2]，3 维卷积[3]，形态学基本操作（膨胀和腐蚀）[4]，光线追踪算法[5]，机器人路径规划[6]，神经网络[7]，稀疏矩阵的乘法[8]等。GPU 的像素处理 可以被看作是一个流处理器[9]。对于像素组成的流，GPU 执行同样的内核程序，渲染出每个像素。原始数据可以通过顶点、纹理表达成为 GPU 可以存取的数据。而内核程序则提供了丰富的指令，使得纹理的存取、复杂计算成为可能。

浮点缓冲区的出现使在多遍渲染的过程中，中间数据不再受到[0，1]及 255 级分辨率的限制，使 GPU 达到了和 CPU 运算同样的精度和动态范围。

2 SOM 算法

基本思想是用一个小的窗口滚动扫描图像，对每一个窗口重复以下步骤：特征提取，计算给定窗口的一组纹理特征，组成特征向量 f；把 f 和 SOM 进行比较，找到最匹配的单元 c。把最小的误差 ec 作为输出值。如果 ec 大于等于指定的限制范围 T，窗口中心像素被认为是一个瑕疵。否则该像素是完好的。于是问题可以描述为：把给定图像块的中心像素或者标示为瑕疵，或者标示为合格（无瑕疵）。见图 1。

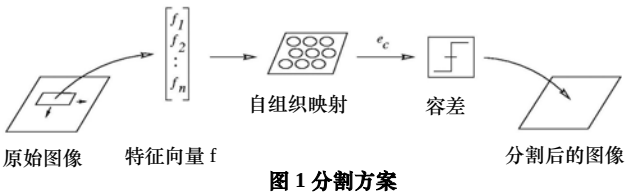

图 1 分割方案

2.1 特征提取

纸幅的图像通常可以看作是随机纹理，采用共生矩阵来描述纹理。在本文中，使用了共生矩阵的反差和均值两个特征值，均值与反差分别为

$$m = \frac{1}{N}\sum_i \sum_j iC_{i,j} \quad (1)$$

$$c = \frac{1}{N}\sum_i \sum_j (i-j)^2 C_{i,j} \quad (2)$$

其中,N 是图像 g 中共生矩阵像素对的个数，C 为共生矩阵。

2.2 分割部分

分割步骤中的基本思想是用无瑕疵样本对分类器进行训练。如果一个未知样本与无纸病样本的测定模型差别足够大，那么它就被认为是瑕疵。自组织映射(SOM)用于测定这些模型，而且每一个 SOM 单元都代表了一个精确的函数。

这一对原始 SOM 算法的扩充被称为统计 SOM[10]。

(1)经典 SOM 自组织映射。自组织映射是一个无监督的自组织神经网络。它把高维特性空间映射到二维空间，同时拓扑特性得到保留。SOM 是由按照拓扑特性分布在通常是二维栅格上的一组映射单元组成。SOM 是一种竞争型网络。对于每一个特征向量 f(t),最匹配的映射单元被当作网络的输出。最匹配映射单元 c 是通过比较输入向量 f(t)和每个映射单元的权向量 mi 来确定的。

$$\| f(t) - m_c \| = \min_i \{ \| f(t) - m_i \| \} \quad (3)$$

通常采用欧式距离

$$\| f(t) - m_i \| = \sqrt{\sum_j (f_j(t) - m_{i,j})^2} \quad (4)$$

用大量的特征向量 f(t)对 SOM 进行基本学习规则下的训练之后，映射单元就把特征空间划分为称作 Voronoi 区的小区域。最后形成的分区被称作 Voronoi 棋盘。对每一个映射单元 i，它的 Voronoi 区域所对应的输入向量 f(t)的集合定义为 Voronoi 集合 F_i。也就是说，Voronoi 集合 F_i 是由输入向量所组成，其中单元 i 是为最匹配单元。

$$F_i = \{ f(t) \mid i = \arg\min_k \{ \| f(t) - m_k \| \} \forall t \} \quad (5)$$

(2)统计 SOM 算法。统计 SOM 的基本思想是把每一个 SOM 单元对应的 Voronoi 集合中的每一个成员平面拟合为一个一维密度函数，对每一个一维密度函数确定一个置信区间。一般采用均匀密度函数。

假设一个 SOM 网络经过大量的特征向量 f(t)的训练完成后，得到映射单元 m_i。每一个映射单元 m_i 都有一个 Voronoi 区域和一个 Voronoi 集合 Fi。映射单元 m_i 用一个新的权向量 m_i'和一个置信区间向量 d_i'来代替。这些就确定了映射单元 i 的一个超立方体的中心点 mi'和尺寸 d_i'。这些立方体内的所有点服从估计分布。根据以上所述，Voronoi 集合 F_i 中的第 j 个成员平面的置信区间就可以表示为

$$\left[m'_{i,j} - \frac{d'_{i,j}}{2}, m'_{i,j} + \frac{d'_{i,j}}{2} \right] \quad (6)$$

这样置信区间向量 d'_i 就定义了单元 i 的置信区间宽度。如果假设 Voronoi 集合 F_i 中的第 j 个成员平面符合均匀分布，那么置信区间向量 d'_i 由

$$d'_{i,j} = d(\max\{f_j(t)\} - \min\{f_j(t)\}), \forall f(t) \in F_i \quad (7)$$

给出。其中，d 是置信度(典型的值为 90%-100%)。新的权向量 m'_i 即为置信区间的中心

$$m'_{i,j} = \frac{\min\{f_j(t)\} + \max\{f_j(t)\}}{2}, \forall f(t) \in F_i \quad (8)$$

一个特征向量 f(t)和映射单元 i 的匹配无差 e_i 的定义与海明距离类似:

$$e_i = \sum_j e_{i,j}, \text{其中 } e_{i,j} = \begin{cases} 0 \text{如果 } (f_j(t) - m'_{i,j})^2 \le (d'_{i,j})^2 \\ 1 \text{其他} \end{cases} \quad (9)$$

其中，$m'_{i,j}$ 是新的映射单元 m'_i 的第 j 个成员，$d'_{i,j}$ 是映射单元的 i 置信区间向量的第 j 个成员。一种更为有效的比较方法是检测特征向量 f(t)是否位于置信区间以内。

$$e_i = \sum_{j=1}^{n} e_{i,j}, \text{其中 } e_{i,j} = \begin{cases} 0 \text{若 } m'_{i,j} - \frac{d'_{i,j}}{2} \le f_j(t) \le m'_{i,j} + \frac{d'_{i,j}}{2} \\ 1 \text{其他} \end{cases} \quad (10)$$

最短的海明距离给出了特征向量和最匹配单元之间的误差。

3 特征向量的快速计算

从上述的 SOM 算法来看，如何快速计算每个窗口的特征向量是关键。K. Heikkinen 和 P. Vuorimaa 提出一种快速计算特征向量的方法[11]。使这些特征向量可以在不计算共生矩阵的情况下，通过更新前一步的保存结果来进行计算。

在更新过程中，只考虑离开和进入窗口的像素对（见图 2)。在下面的表述中，假设置换向量 d 只有水平分量，即 d=(d_x,0)。然而，该结构可以很容易地扩展到一般情况。

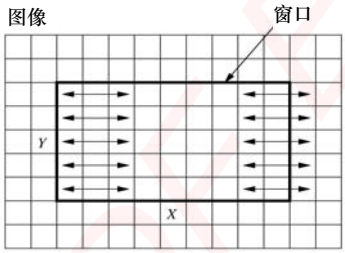

图 2 当窗口从左到右移动时,离开和进入其中的像素对点 dx=2

3.1 均值的快速计算

均值的计算公式为

$$m(t+1) = m(t) + (m(t+1) - m(t)) \quad (11)$$

其中，t 是离散的时间坐标。根据等式（2）差为

$$(m(t+1) - m(t)) = \sum_i \sum_j iC_{ij}(t+1) - \sum_i \sum_j iC_{ij}(t) = \sum_i \sum_j i(C_{ij}(t+1) - C_{ij}(t)) \quad (12)$$

如果移动窗口的尺寸是(X,Y)，那么差值就可以简单表示为

$$m(t+1) - m(t) = \sum_{l=0}^{Y-1} (g_{X-d_x,l(t)} - g_{0,l(t)}) \quad (13)$$

于是 $$m(t+1) = m(t) + \sum_{l=0}^{Y-1} (g_{X-d_x,l(t)} - g_{0,l(t)}) \quad (14)$$

从而不需要进行灰度矩阵的计算，提高了计算效率。

3.2 反差的快速计算

与上面类似，反差的计算如下:

$$c(t+1) = c(t) + (c(t+1) - c(t)) \quad (15)$$

根据反差的定义式（3)，差值为

$$c(t+1) - c(t) = \sum_i \sum_j (i-j)^2 C_{ij}(t+1) - \sum_i \sum_j (i-j)^2 C_{ij}(t) = \sum_i \sum_j (i-j)^2 (C_{ij}(t+1) - C_{ij}(t)) \quad (16)$$

和均值特性类似，反差特性也可以只使用图像窗口计算得到，不需要共生矩阵的计算。

$$c(t+1) - c(t) = \sum_{l=0}^{Y-1} \left((g_{X-d_x,l(t)} - g_{X,l(t)})^2 - (g_{0,l(t)} - g_{d_x,l(t)})^2 \right) \quad (17)$$

于是 $$c(t+1) = c(t) + \sum_{l=0}^{Y-1} \left((g_{X-d_x,l(t)} - g_{X,l(t)})^2 - (g_{0,l(t)} - g_{d_x,l(t)})^2 \right) \quad (18)$$

对上面的计算进行垂直方向的扩展，可以得到 mean 值为

$$m(t+1) = m(t) + \sum_{k=0}^{X-d_x-1} (g_{k,Y} - g_{k,0}) \quad (19)$$

contrast 值为

$$c(t+1) = c(t) + \sum_{k=0}^{X-d_x-1} \left((g_{k,Y}(t) - g_{k+d_x,Y}(t))^2 - (g_{k,0}(t) - g_{k+d_x,0}(t))^2 \right) \quad (20)$$

所以，对于一幅图像的特征向量的计算来说，只要计算左下角的第一个窗口的特征向量，然后计算出其他窗口特征向量的差值，最后迭代计算其它窗口的特征向量。大大加快了图像的处理速度。

4 基于 GPU 的硬件实现

由于本文中每个窗口的特征向量是 4 个，正好匹配像素的 RGBA 4 个分量,因此,计算结果保存在每个像素的 RGBA 分量中。为了保证图像纹理与屏幕像素的严格对应关系，我们采用了正投影。浮点缓冲区的使用使得中间结果不加任何的限幅和偏置，既保证了精度，又减少了指令。

4.1 灰度化及量化灰度级

图像首先经过灰度化及量化。按照 0.3*R + 0.59*G + 0.11* B，利用 Fragment 程序对 RGB 分量加权平均，然后直接表达成为[0-255]的整数值，以便后面的操作。

4.2 计算每个窗口的特征值增量

根据公式(14),(18),(19),(20)，可分别计算出水平方向和垂直方向窗口的特征值增量。下面以水平方向为例说明：

(1) 1-pass 算法

为清楚起见，假设窗口的大小为 15×5，特征向量的大小为(2,0)和(4,0)，以(2,0)为例。

假设任一窗口的一行像素分布为

p0 p1 p2 p3 p4 p5 p6 p7 p8 p9 p10 p11 p12 p13 p14 p15

其中 p15 属于下一窗口。根据公式(14)，mean 值的计算实际上包括两部分：计算每行的 p13-p0；叠加 5 行的结果。同样，对于 Contrast 值，也包括两部分：计算$(p15-p13)^2$-$(p2-p0)^2$；叠加 5 行的结果。

显而易见，这一过程很容易通过使用 Fragment 程序加以实现。程序的流程如图 3。

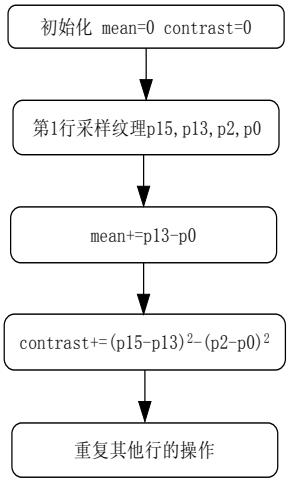

图 3 Fragment 程序流程

可见流程比较清晰，编程也相对容易。但是，该流程存在采样纹理过多，运行速度慢的问题。如果计算 2 个置换向量，需要执行采样纹理指令 TEX 30 次，减法 SUB 30 次，加法 ADD 和乘加 MAD 指令各 18 次，总指令条数超过 100 条。不能达到实时的处理。

(2) multi-pass 算法

仔细分析 Fragment 程序流程，发现速度慢的原因在于重复采样过多。对于 512×512 的图像，每个像素都要进行上下各 2 行的纹理采样，总的指令超过 512×512×100。而在这些指令中，对于相邻的上下两个窗口来说，有 4 行的像素被重复采样。同样，对于左右相距一定宽度的窗口，也有重复发生。因此如何避免重复采样是提高速度的关键。我们采取多遍渲染算法，避免了重复采样。

可以看出，如果把整个图像进行一次纹理的操作，其效果等同于每个窗口的计算，但是由于 GPU 对于纹理操作的高性能，提高了处理速度。对于每个像素，计算 2 个置换向量下的特征向量增量。

1) 减法和平方。对于每个像素，采样 p15,p13,p11,p0 纹理单元。然后执行 p13-p0,p11-p0,$(p15-p13)^2$,$(p15-p11)^2$,得到的 4 个结果保存在 p15 的 RGBA 分量中。需要说明的是，$(p2-p0)^2$ 和$(p4-p0)^2$ 的计算自动被$(p15-p13)^2$,$(p15-p11)^2$ 的计算所包含。如图 4 所示，其中 O 代表输出像素的颜色。本程序使用 TEX 4 个，SUB4 个，ADD 和 MUL 各 2 个。

2)减法。本模块实现$(p15-p13)^2$-$(p2-p0)^2$ 和$(p15-p11)^2$-$(p4-p0)^2$ 中的中间减法。其中$(p2-p0)^2$ 保存在 p2 的 R 分量中，$(p4-p0)^2$ 保存在 p11 的 G 分量中。如图 5 所示，其中 O 代表输出像素的颜色。本程序使用 TEX3 个，ADD2 个。

3)求和。这一步骤完成 5 个像素的叠加。进入此程序的像素 p 被认为是窗口的中心像素，设 p 的坐标是(x,y),则需要采集(x+7,y±1), (x+7,y±2), (x+7,y)共 5 个像素。运算结果保存在 RGBA 分量中，使用 TEX5 个，ADD6 个。

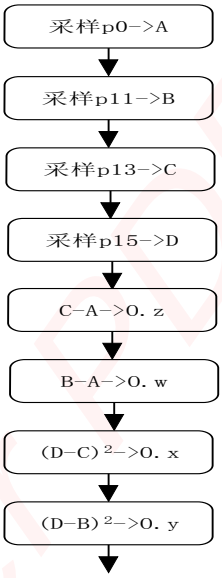

图 4 减法和平方图

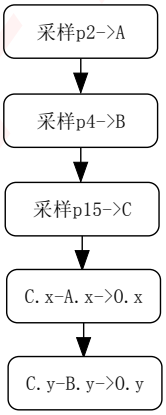

图 5 平方后的减法

至此，已经完成了每个窗口的特征向量的增量计算。可以看出，指令和为 TEX12 个，ADD 10 个，SUB4 个，MUL2 个，总数 28 个，不足优化前的 1/3。大大提高了运行速度。

4.3 叠代计算

需要说明，上面所进行的计算，是基于每行的第一个窗口的特征向量已经被计算出来。这一过程实际包括两部分：左下角第一个窗口的特征向量的计算是由 CPU 按照常规计算得来；另外，根据第一行第一列的特征值计算第一列其他行的特征值。后者实际也是可以通过 GPU 来计算，但是，由于计算量不是很大(498)，可以使用 CPU 来完成。实际上，这一过程的耗时约为 3ms。

接下来如何迭代计算每行的特征值，是一个典型的区域求和问题[12]，也就是说，对于一行中的每一个像素来说，它的值等于左面所有像素值的和。在 GDC 2003 会议上，nVidia 公司的 Simon Green 提出了用 GPU 计算 SAT 的一种方法[13]，但是文中也提到该方法可能产生不可预料的结果。

我们使用 CPU 完成迭代计算，这样就产生了 CPU 和 GPU 之间的数据传输，增加了运行时间。程序流程比较简单：首先，把浮点缓冲区的值读到主存(GPU->CPU)，然后左到右迭代计算每行每个像素的特征值，然后把特征值生成纹理(CPU->GPU)进行下一步的判断操作。

4.4 判断

实现对每个像素的特征值与已经训练好的统计 SOM 中的各个 SOM 单元的置信区间进行比较，得到最小海明距离，如果该距离小于设定的阈值，则该像素是完好的，否则，是一个瑕疵。每个 SOM 单元的置信区间作为参数传递给 Fragment 程序。虽然 SOM 单元的数目不多(假设为 9)，但是

(下转第 218 页)

点，CheckIn 和 CheckOut Web 服务；单击服务可查看说明，选择服务前的复选框则可调用该服务并自动生成 Web 服务代理。完成上述步骤后，单击“添加到应用”可正式开始使用该 Web 服务。图 6 为 CheckIn Web 服务的调用窗口。用户可以调用 ST-Developer 转换 STEP 物理文件（调用结果如文本框内容所示，其为 XML 有效内容绑定的 SOAP 消息）；窗体要求用户填写对象相应属性。单击 Check In 按钮，调用该 Web 服务。PDM 的 XML 处理器对返回的有效数据绑定的 SOAP 消息进行内容提取，以 BLOB 或关系数据的形式存入 PDM 系统以提供给其它 CAx 系统使用。

图 6　CheckIn Web 服务调用窗口

6 结论

对大量、分布的产品数据实施有效的管理直接影响着 VE 的运作效率，本文在给出基于 XML 的集成框架之后，对 CAx 系统和 PDM 系统之间的集成进行了新的探讨：将 Web 服务应用于两者的集成，并通过实例说明该方法的可行性，为跨企业应用集成开辟了新的路径。

但是，我们开发的 PDM 系统 Web 服务还缺少用户验证等安全保障功能，目前该模块尚在开发过程中。

参考文献

1 NIIIP Inc. NIIIP Reference Architecture.http://www.niiip.org, 1998
2 http://www.w3.org/TR/2000/REC-xml-20001006.htm
3 http://www.w3.org/TR/SOAP
4 http://www.w3.org/TR/DTD
5 http://www.w3.org/TR/Schema
6 Bhandarkar M, Downie B, Hardwick M, et al. IGES to STEP Translation and Visualization. The 6th Industrial Engineering Research Conference Proceedings, 1997-05: 656–661
7 http://www.w3.org/TR/ws-arch/#whatis

(上接第 27 页)

每个单元有 4 个特征向量，而每个特征向量又有 2 个值对应于置信区间的最大和最小值。因此总的判断程序比较冗长，其中 TEX1 个，SLT、ADD、DP4 各 9 个，SGT10 个，MIN8 个，MOV1 个。指令总数 39 个。

5 仿真结果

试验平台是 Windows 2000,AMD Duron 900MHz,256MB 内存,Geforce FX5200 128MB 显存，其中 FX5200 的时钟频率 250MHz。为利用 Visual C++进行了仿真试验。GPU 的执行时间约为 235ms。如果不考虑 CPU 与 GPU 之间的数据传输所消耗的时间，我们的结果还是令人鼓舞的。见表 1。

表 1 CPU 与 GPU 执行效率比较

	灰度化及量化	增量计算	迭代	判断
CPU(ms)	20	100	-	90
GPU(ms)	6	60	114	47

(1)灰度化及量化灰度级。GPU 所用时间为 6ms，而 CPU 用来执行灰度化的时间约为 27ms，其中加权运算的浮点操作占用了 1/2 的时间。

(2)窗口特征值增量的计算。包括 multi-pass 中的浮点缓冲区的切换，以及纹理的拷贝等在内，GPU 的执行时间约为 40ms。

(3)迭代。由于 GPU 无法完成迭代操作，因此需要读写浮点缓冲区，其中读出缓冲区所用时间为 36ms。迭代计算约为 30ms，重新生成纹理的时间约为 47ms。总时间 114ms。大大超出前面所有的 GPU 操作。

如果不考虑迭代所产生的时耗，总的处理速度 GPU 大约为 CPU 的 2 倍。

6 讨论

决定 CPU 运算速度的因素不只是时钟频率，程序中要进行频繁的内存读写，一般 PC3200 标识的计算机，配有 DDR400 内存，数据总线只有 64 位。理论的带宽为 3.2GBps。内存的带宽满足不了 CPU 的全速运行，大部分时间 CPU 都在等待内存读写操作的完成。而在 GPU 中，图像子系统的带宽为 256 位，系统的显存工作频率为 200MHz，因此，带宽为 6.4GBps。因此，理论上 GPU 的带宽远远大于 CPU。

鉴于本文中所采用的 Geforce FX5200 的硬件配置，相信一些高端产品(如 nVidia FX5800，显存工作频率为 500MHz)，可以明显地提高处理速度,有可能达到实时处理。

由于 GPU 本身的限制，无法完成迭代操作，是处理过程的瓶颈所在。我们希望在下一版本的 OpenGl 中能够提供类似的扩展。

参考文献

1 Iivarinen J, Heikkinen K, Rauhamaa J, et al. A Defect Detection Scheme for Web Surface Inspection. International Journal of Pattern Recognition and Artificial Intelligence, 2000,14(6):735-755
2 Larsen E S,McAllister D.Fast Matrix Multiplies Using Graphics Hardware. The International Conference for High Performance Computing and Communications, 2001
3 Hopf M，Ertl T.Accelerating 3d Convolution Using Graphics Hardware. In IEEE Visualization ’99,1999-10：47-474
4 Hopf M，Ertl T.Accelerating Morphological Analysis with Graphics Hardware. In Workshop on Vision, Modelling ,and Visualization VMV ’00, 2000：337-345
5 Purcell T, Buck I, Mark W,et al.Ray Tracing on Programmable Graphics Hardware. In Proceedings of SIGGRAPH 2002. ACM, 2002
6 Lengyel J,Reichert M,Donald B,et al.Real-time Robot Motion Planning Using Rasterizing Computer Graphics Hardware. In Proceedings of SIGGRAPH 1990, 1990-327-335
7 Bohn C A.Kohonen Feature Mapping Through Graphics Hardware. In Proceedings of Int. Conf. on Compu. Intelligence and Neurosciences 1998
8 Bolz J, Farmer I, Grinspun E, et al.Sparse Matrix Solvers on the Gpu: Conjugate Gradients and Multigrid.In SIGGRAPH 2003,Computer Graphics Proceedings, 2003
9 Khailany B, Dally W J, Rixner S, et al.Imagine: Media Processing with Streams. IEEE Micro 21, 2001-02:35-46
10 Iivarinen J, Rauhamaa J,Visa A.Unsupervised Segmentation of Surface Defects.Proc. 13th Int. Conf. Pattern Recognition, Vol. IV,Wien, Austria,1996-08-25:356-360

嵌入式资源免费下载

总线协议：

1. 基于 PCIe 驱动程序的数据传输卡 DMA 传输
2. 基于 PCIe 总线协议的设备驱动开发
3. CANopen 协议介绍
4. 基于 PXI 总线 RS422 数据通信卡 WDM 驱动程序设计
5. FPGA 实现 PCIe 总线 DMA 设计
6. PCI Express 协议实现与验证
7. VPX 总线技术及其实现
8. 基于 Xilinx FPGA 的 PCIE 接口实现
9. 基于 PCI 总线的 GPS 授时卡设计
10. 基于 CPCI 标准的 6U 信号处理平台的设计
11. USB30 电路保护
12. USB30 协议分析与框架设计
13. USB 30 中的 CRC 校验原理及实现
14. 基于 CPLD 的 UART 设计
15. IPMI 在 VPX 系统中的应用与设计
16. 基于 CPCI 总线的 PMC 载板设计
17. 基于 VPX 总线的工件台运动控制系统研究与开发
18. PCI Express 流控机制的研究与实现
19. UART16C554 的设计
20. 基于 VPX 的高性能计算机设计
21. 基于 CAN 总线技术的嵌入式网关设计
22. Visual C 串行通讯控件使用方法与技巧的研究
23. IEEE1588 精密时钟同步关键技术研究
24. GPS 信号发生器射频模块的一种实现方案
25. 基于 CPCI 接口的视频采集卡的设计
26. 基于 VPX 的 3U 信号处理平台的设计
27. 基于 PCI Express 总线 1394b 网络传输系统 WDM 驱动设计
28. AT89C52 单片机与 ARINC429 航空总线接口设计
29. 基于 CPCI 总线多 DSP 系统的高速主机接口设计
30. 总线协议中的 CRC 及其在 SATA 通信技术中的应用
31. 基于 FPGA 的 SATA 硬盘加解密控制器设计
32. Modbus 协议在串口通讯中的研究及应用
33. 高可用的磁盘阵列 Cache 的设计和实现
34. RAID 阵列中高速 Cache 管理的优化

35. 一种新的基于 RAID 的 CACHE 技术研究与实现
36. 基于 PCIE-104 总线的高速数据接口设计
37. 基于 VPX 标准的 RapidIO 交换和 Flash 存储模块设计
38. 北斗卫星系统在海洋工程中的应用
39. 北斗卫星系统在远洋船舶上应用的研究
40. 基于 CPCI 总线的红外实时信号处理系统
41. 硬件实现 RAID 与软件实现 RAID 的比较
42. 基于 PCI Express 总线系统的热插拔设计
43. 基于 RAID5 的磁盘阵列 Cache 的研究与实现
44. 基于 PCI 总线的 MPEG2 码流播放卡驱动程序开发
45. 基于磁盘异或引擎的 RAID5 小写性能优化
46. 基于 IEEE1588 的时钟同步技术研究
47. 基于 Davinci 平台的 SD 卡读写优化
48. 基于 PCI 总线的图像处理及传输系统的设计
49. 串口和以太网通信技术在油液在线监测系统中的应用
50. USB30 数据传输协议分析及实现
51. IEEE 1588 协议在工业以太网中的实现
52. 基于 USB30 的设备自定义请求实现方法
53. IEEE1588 协议在网络测控系统中的应用
54. USB30 物理层中弹性缓冲的设计与实现
55. USB30 的高速信息传输瓶颈研究
56. 基于 IPv6 的 UDP 通信的实现
57. 一种基于 IPv6 的流媒体传送方案研究与实现
58. 基于 IPv4-IPv6 双栈的 MODBUS-TCP 协议实现
59. RS485CAN 网关设计与实现
60. MVB 周期信息的实时调度
61. RS485 和 PROFINET 网关设计
62. 基于 IPv6 的 Socket 通信的实现
63. MVB 网络重复器的设计
64. 一种新型 MVB 通信板的探究

VxWorks:

1. 基于 VxWorks 的多任务程序设计
2. 基于 VxWorks 的数据采集存储装置设计
3. Flash 文件系统分析及其在 VxWorks 中的实现
4. VxWorks 多任务编程中的异常研究
5. VxWorks 应用技巧两例
6. 一种基于 VxWorks 的飞行仿真实时管理系统

7. 在 VxWorks 系统中使用 TrueType 字库
8. 基于 FreeType 的 VxWorks 中文显示方案
9. 基于 Tilcon 的 VxWorks 简单动画开发
10. 基于 Tilcon 的某武器显控系统界面设计
11. 基于 Tilcon 的综合导航信息处理装置界面设计
12. VxWorks 的内存配置和管理
13. 基于 VxWorks 系统的 PCI 配置与应用
14. 基于 MPC8270 的 VxWorks BSP 的移植
15. Bootrom 功能改进经验谈
16. 基于 VxWorks 嵌入式系统的中文平台研究与实现
17. VxBus 的 A429 接口驱动
18. 基于 VxBus 和 MPC8569E 千兆网驱动开发和实现
19. 一种基于 vxBus 的 PPC 与 FPGA 高速互联的驱动设计方法
20. 基于 VxBus 的设备驱动开发
21. 基于 VxBus 的驱动程序架构分析
22. 基于 VxBus 的高速数据采集卡驱动程序开发
23. Vxworks 下的冗余 CAN 通讯模块设计
24. WindML 工业平台下开发 S1d13506 驱动及显示功能的实现
25. WindML 中 Mesa 的应用
26. VxWorks 下图形用户界面开发中双缓冲技术应用
27. VxWorks 上的一种 GUI 系统的设计与实现
28. VxWorks 环境下 socket 的实现
29. VxWorks 的 WindML 图形界面程序的框架分析
30. VxWorks 实时操作系统及其在 PC104 下以太网编程的应用
31. 实时操作系统任务调度策略的研究与设计
32. 军事指挥系统中 VxWorks 下汉字显示技术
33. 基于 VxWorks 实时控制系统中文交互界面开发平台
34. 基于 VxWorks 操作系统的 WindML 图形操控界面实现方法
35. 基于 GPU FPGA 芯片原型的 VxWorks 下驱动软件开发
36. VxWorks 下的多串口卡设计
37. VxWorks 内存管理机制的研究
38. T9 输入法在 Tilcon 下的实现
39. 基于 VxWorks 的 WindML 图形界面开发方法
40. 基于 Tilcon 的 IO 控制板可视化测试软件的设计和实现
41. 基于 VxWorks 的通信服务器实时多任务软件设计
42. 基于 VXWORKS 的 RS485MVB 网关的设计与实现

Linux：

1. Linux 程序设计第三版及源代码
2. NAND FLASH 文件系统的设计与实现
3. 多通道串行通信设备的 Linux 驱动程序实现
4. Zsh 开发指南-数组
5. 常用 GDB 命令中文速览
6. 嵌入式 C 进阶之道
7. Linux 串口编程实例
8. 基于 Yocto Project 的嵌入式应用设计
9. Android 应用的反编译
10. 基于 Android 行为的加密应用系统研究
11. 嵌入式 Linux 系统移植步步通
12. 嵌入式 CC++语言精华文章集锦
13. 基于 Linux 的高性能服务器端的设计与研究
14. S3C6410 移植 Android 内核
15. Android 开发指南中文版
16. 图解 Linux 操作系统架构设计与实现原理（第二版）
17. 如何在 Ubuntu 和 Linux Mint 下轻松升级 Linux 内核
18. Android 简单 mp3 播放器源码
19. 嵌入式 Linux 系统实时性的研究
20. Android 嵌入式系统架构及内核浅析
21. 基于嵌入式 Linux 操作系统内核实时性的改进方法研究
22. Linux TCP IP 协议详解
23. Linux 桌面环境下内存去重技术的研究与实现
24. 掌握 Android 7.0 新增特性 Quick Settings
25. Android 应用逆向分析方法研究
26. Android 操作系统的课程教学
27. Android 智能手机操作系统的研究
28. Android 英文朗读功能的实现
29. 基于 Yocto 订制嵌入式 Linux 发行版
30. 基于嵌入式 Linux 的网络设备驱动设计与实现
31. 如何高效学习嵌入式
32. 基于 Android 平台的 GPS 定位系统的设计与实现
33. LINUX ARM 下的 USB 驱动开发
34. Linux 下基于 I2C 协议的 RTC 驱动开发
35. 嵌入式下 Linux 系统设备驱动程序的开发
36. 基于嵌入式 Linux 的 SD 卡驱动程序的设计与实现
37. Linux 系统中进程调度策略
38. 嵌入式 Linux 实时性方法
39. 基于实时 Linux 计算机联锁系统实时性分析与改进
40. 基于嵌入式 Linux 下的 USB30 驱动程序开发方法研究
41. Android 手机应用开发之音乐资源播放器
42. Linux 下以太网的 IPv6 隧道技术的实现

43. Research and design of mobile learning platform based on Android
44. 基于 linux 和 Qt 的串口通信调试器调的设计及应用
45. 在 Linux 平台上基于 QT 的动态图像采集系统的设计
46. 基于 Android 平台的医护查房系统的研究与设计
47. 基于 Android 平台的软件自动化监控工具的设计开发
48. 基于 Android 的视频软硬解码及渲染的对比研究与实现
49. 基于 Android 移动设备的加速度传感器技术研究
50. 基于 Android 系统振动测试仪研究
51. 基于缓存竞争优化的 Linux 进程调度策略

Windows CE:

1. Windows CE.NET 下 YAFFS 文件系统 NAND Flash 驱动程序设计
2. Windows CE 的 CAN 总线驱动程序设计
3. 基于 Windows CE.NET 的 ADC 驱动程序实现与应用的研究
4. 基于 Windows CE.NET 平台的串行通信实现
5. 基于 Windows CE.NET 下的 GPRS 模块的研究与开发
6. win2k 下 NTFS 分区用 ntldr 加载进 dos 源代码
7. Windows 下的 USB 设备驱动程序开发
8. WinCE 的大容量程控数据传输解决方案设计
9. WinCE6.0 安装开发详解
10. DOS 下仿 Windows 的自带计算器程序 C 源码
11. G726 局域网语音通话程序和源代码
12. WinCE 主板加载第三方驱动程序的方法
13. WinCE 下的注册表编辑程序和源代码
14. WinCE 串口通信源代码
15. WINCE 的 SD 卡程序[可实现读写的源码]
16. 基于 WinCE 的 BootLoader 研究
17. Windows CE 环境下无线网卡的自动安装
18. 基于 Windows CE 的可视电话的研究与实现
19. 基于 WinCE 的嵌入式图像采集系统设计
20. 基于 ARM 与 WinCE 的掌纹鉴别系统
21. DCOM 协议在网络冗余环境下的应用
22. Windows XP Embedded 在变电站通信管理机中的应用
23. XPE 在多功能显控台上的开发与应用
24. 基于 Windows XP Embedded 的 LKJ2000 仿真系统设计与实现
25. 虚拟仪器的 Windows XP Embedded 操作系统开发
26. 基于 EVC 的嵌入式导航电子地图设计
27. 基于 XPEmbedded 的警务区 SMS 指挥平台的设计与实现

PowerPC：

1. Freescale MPC8536 开发板原理图
2. 基于 MPC8548E 的固件设计
3. 基于 MPC8548E 的嵌入式数据处理系统设计
4. 基于 PowerPC 嵌入式网络通信平台的实现
5. PowerPC 在车辆显控系统中的应用
6. 基于 PowerPC 的单板计算机的设计
7. 用 PowerPC860 实现 FPGA 配置
8. 基于 MPC8247 嵌入式电力交换系统的设计与实现
9. 基于设备树的 MPC8247 嵌入式 Linux 系统开发
10. 基于 MPC8313E 嵌入式系统 UBoot 的移植
11. 基于 PowerPC 处理器 SMP 系统的 UBoot 移植
12. 基于 PowerPC 双核处理器嵌入式系统 UBoot 移植
13. 基于 PowerPC 的雷达通用处理机设计
14. PowerPC 平台引导加载程序的移植
15. 基于 PowerPC 嵌入式内核的多串口通信扩展设计
16. 基于 PowerPC 的多网口系统抗干扰设计
17. 基于 MPC860T 与 VxWorks 的图形界面设计
18. 基于 MPC8260 处理器的 PPMC 系统
19. 基于 PowerPC 的控制器研究与设计
20. 基于 PowerPC 的模拟量输入接口扩展
21. 基于 PowerPC 的车载通信系统设计
22. 基于 PowerPC 的嵌入式系统中通用 IO 口的扩展方法
23. 基于 PowerPC440GP 型微控制器的嵌入式系统设计与研究
24. 基于双 PowerPC 7447A 处理器的嵌入式系统硬件设计
25. 基于 PowerPC603e 通用处理模块的设计与实现
26. 嵌入式微机 MPC555 驻留片内监控器的开发与实现

ARM：

1. 基于 DiskOnChip 2000 的驱动程序设计及应用
2. 基于 ARM 体系的 PC-104 总线设计
3. 基于 ARM 的嵌入式系统中断处理机制研究

4. 设计 ARM 的中断处理
5. 基于 ARM 的数据采集系统并行总线的驱动设计
6. S3C2410 下的 TFT LCD 驱动源码
7. STM32 SD 卡移植 FATFS 文件系统源码
8. STM32 ADC 多通道源码
9. ARM Linux 在 EP7312 上的移植
10. ARM 经典 300 问
11. 基于 S5PV210 的频谱监测设备嵌入式系统设计与实现
12. Uboot 中 start.S 源码的指令级的详尽解析
13. 基于 ARM9 的嵌入式 Zigbee 网关设计与实现
14. 基于 S3C6410 处理器的嵌入式 Linux 系统移植
15. CortexA8 平台的μC-OSⅡ及 LwIP 协议栈的移植与实现
16. 基于 ARM 的嵌入式 Linux 无线网卡设备驱动设计
17. ARM S3C2440 Linux ADC 驱动
18. ARM S3C2440 Linux 触摸屏驱动
19. Linux 和 Cortex-A8 的视频处理及数字微波传输系统设计
20. Nand Flash 启动模式下的 Uboot 移植
21. 基于 ARM 处理器的 UART 设计
22. ARM CortexM3 处理器故障的分析与处理
23. ARM 微处理器启动和调试浅析
24. 基于 ARM 系统下映像文件的执行与中断运行机制的实现
25. 中断调用方式的 ARM 二次开发接口设计
26. ARM11 嵌入式系统 Linux 下 LCD 的驱动设计
27. Uboot 在 S3C2440 上的移植
28. 基于 ARM11 的嵌入式无线视频终端的设计
29. 基于 S3C6410 的 Uboot 分析与移植
30. 基于 ARM 嵌入式系统的高保真无损音乐播放器设计
31. UBoot 在 Mini6410 上的移植
32. 基于 ARM11 的嵌入式 Linux NAND FLASH 模拟 U 盘挂载分析与实现
33. 基于 ARM11 的电源完整性分析
34. 基于 ARM S3C6410 的 uboot 分析与移植
35. 基于 S5PC100 移动视频监控终端的设计与实现

Hardware:

1. DSP 电源的典型设计
2. 高频脉冲电源设计
3. 电源的综合保护设计
4. 任意波形电源的设计

5. 高速 PCB 信号完整性分析及应用
6. DM642 高速图像采集系统的电磁干扰设计
7. 使用 COMExpress Nano 工控板实现 IP 调度设备
8. 基于 COM Express 架构的数据记录仪的设计与实现
9. 基于 COM Express 的信号系统逻辑运算单元设计
10. 基于 COM Express 的回波预处理模块设计
11. 基于 X86 平台的简单多任务内核的分析与实现
12. 基于 UEFI Shell 的 PreOS Application 的开发与研究
13. 基于 UEFI 固件的恶意代码防范技术研究
14. MIPS 架构计算机平台的支持固件研究
15. 基于 UEFI 固件的攻击验证技术研究
16. 基于 UEFI 的 Application 和 Driver 的分析与开发
17. 基于 UEFI 的可信 BIOS 研究与实现
18. 基于 UEFI 的国产计算机平台 BIOS 研究
19. 基于 UEFI 的安全模块设计分析
20. 基于 FPGA NiosⅡ的等精度频率计设计
21. 基于 FPGA 的 SOPC 设计
22. 基于 SOPC 基本信号产生器的设计与实现
23. 基于龙芯平台的 PMON 研究与开发
24. 基于 X86 平台的嵌入式 BIOS 可配置设计
25. 基于龙芯 2F 架构的 PMON 分析与优化
26. CPU 与 GPU 之间接口电路的设计与实现
27. 基于龙芯 1A 平台的 PMON 源码编译和启动分析
28. 基于 PC104 工控机的嵌入式直流监控装置的设计
29. GPGPU 技术研究与发展
30. GPU 实现的高速 FIR 数字滤波算法
31. 一种基于 CPUGPU 异构计算的混合编程模型
32. 面向 OpenCL 模型的 GPU 性能优化
33. 基于 GPU 的 FDTD 算法

Programming:

1. 计算机软件基础数据结构 - 算法
2. 高级数据结构对算法的优化
3. 零基础学算法
4. Linux 环境下基于 TCP 的 Socket 编程浅析
5. Linux 环境下基于 UDP 的 socket 编程浅析
6. 基于 Socket 的网络编程技术及其实现

7. 数据结构考题 - 第 1 章 绪论
8. 数据结构考题 - 第 2 章 线性表
9. 数据结构考题 - 第 2 章 线性表 - 答案