TTƯD.14:

CÀI ĐẶT THUẬT TOÁN BACKRGOUND SUBTRACTION SỬ DỤNG MÔ HÌNH TRỘN GAUSSIAN CỦA ZIVKOVIC TRÊN GPU

Sinh viên: **Phạm Duy Tùng** – Toán tin 2 – K52

Giáo viên hướng dẫn: **Ths. Đoàn Duy Trung**

Viện Toán ứng dụng và tin học

Tóm tắt:

Thuật toán Background Subtraction là một trong những thuật toán được sử dụng phổ biến nhất trong lĩnh vực Thị giác máy tính (Computer Vision). Thuật toán được sử dụng nhằm xác định những pixel thuộc những vật chuyển động trong video hay còn được gọi là Foreground (FG), còn những vật không chuyển động được gọi là Background (BG). Thuật toán Background Subtraction được sử dụng như là bước tiền xử lý trong rất nhiều bài toán Thị giác máy tính, kết quả của bước này có thể ảnh hưởng lớn đến kết quả của các bước tiếp theo (nhận dạng, theo dõi, …). Chính vì vậy trong vòng hơn 10 năm qua đã có một lượng lớn các bài báo nhằm giải quyết bài toán này. Trong bài viết này tôi xin trình bày một số kỹ thuật cài đặt thuật toán Background Subtraction của Zivkovic [1] trên GPU nhằm nâng cao hiệu năng tính toán. Tốc độ thực thi của thuật toán đạt được 720 fps với ảnh cỡ 960x540, gấp 60 lần bản cài đặt trên CPU và nhanh hơn bản cài đặt của Phạm Vũ [3] trên GPU 23.3% (Thí nghiệm được thực hiện trên hệ thống: Intel P4 3.0GHz CPU, 1G RAM, GeForce GT440 GPU).

Nội dung của bài viết như sau: trong phần 1 tôi giới thiệu sơ lược về thuật toán và mô hình tính toán song song trên GPU. Phần 2 tôi trình bày về mô hình GMM và phương pháp ước lượng tham số cho mô hình. Tôi trình bày thuật toán Background Subtraction của Zivkovic sử dụng GMM trong phần 3. Phần 4 tôi trình bày chi tiết kiến trúc, mô hình tính toán song song trên GPU, và các kỹ thuật cài đặt thuật toán BG subtraction trên GPU. Phần 5 tôi đưa ra một số kết quả thu được, cuối cùng là một số kết luận trong phần 6.

1. Giới thiệu.

Với sự gia tăng không ngừng về số lượng cũng như chất lượng của các thiết bị ghi hình kỹ thuật số mà giá cả ngày càng giảm, việc sở hữu một chiếc máy ảnh hay máy quay không còn xa vời đối với mỗi người, dẫn đến sự bùng nổ các dữ liệu ảnh, video, do đó chúng ta cần các ứng dụng thông minh hơn, mạnh mẽ hơn giúp ta tổ chức, quản lý dữ liệu một cách hiệu quả. Thuật toán Background Subtraction hay Foreground Detection đóng một vai trò quan trọng trong các ứng dụng này, nó giúp giảm bớt một lượng lớn các pixel cần xử lý (vì chỉ cần quan tâm đến các pixel thuộc Foreground). Một trong những phương pháp phổ biến nhất để giải quyết bài toán này là sử dụng mô hình trộn Gaussian để mô hình hóa Background. Mỗi pixel trong một frame được giả thiết là độc lập (giá trị của một pixel quan sát được không ảnh hưởng đến giá trị của các pixel khác) và các giá trị của một pixel bất kỳ quan sát được theo thời gian là một chuỗi thời gian. Ta sử dụng mô hình trộn Gaussian (Gaussian Mixture Model, để cho tiện tôi sẽ ký hiệu là GMM) để mô hình hóa chuỗi thời gian này. Tại mỗi thời điểm ta quan sát được một giá trị cho mỗi pixel, sau đó tham số của mỗi mô hình trộn Gaussian của mỗi pixel sẽ được cập nhật lại theo pixel vừa quan sát được. Sử dụng mô hình trộn Gaussian để mô hình hóa Background lần đầu tiên được đề xuất bởi Nir Friedman và Stuart Russell [4] vào năm 1997, sau đó là đề xuất của Chris Stauffer và W.E.L Grimson [5] vào năm 1999 sử dụng phương pháp ước lượng tham số online trong đó số thành phần Gaussian được chọn cố định, thuật toán được sử dụng rộng rãi trong các ứng dụng thời gian thực. Năm 2004, Zivkovic [1][2] sử dụng phương pháp Bayes để ước lượng số thành phần Gassian của mô hình sau khi quan sát được một giá trị pixel. Chi tiết về mô hình trộn Gaussian và phương pháp ước lượng tham số cho mô hình sẽ được trình bày chi tiết trong phần tiếp theo.

Hiện nay, GPU không còn đơn thuần là một thiết bị hỗ trợ xử lý đồ họa, nó là một thiết bị tính toán hỗ trợ cho CPU tạo nên một *"hệ thông tính toán lai"* (Heterogeneous System). Những thế hệ GPU hiện đại được có khả thực thi song song ở mức độ cao, được tối ưu cho các bài toán Visual Computing [12] và có thể lập trình được bằng ngôn ngữ lập trình C\C++ (NVIDIA GPU và AMD ATI GPU). Không giống như CPU, GPU có khả năng thực thi hàng nghìn *thread* cùng một lúc trên các tập dữ liệu khác nhau theo mô hình tính toán *data-parallel* (một chương trình được thực thi song song trên các tập dữ liệu khác nhau), do đó GPU rất thích hợp cho các bài toán trong xử lý ảnh (image processing), thị giác máy tính (computer vision),…Chi tiết xin xem trong 4.1.

Thuật toán Background Subtraction của Stauffer và Grimson [5] được cài đặt trên GPU lần đầu tiên vào năm 2006 bởi Lee và Jeong sử dụng thư viện Cg của Nvidia [6]. Sau đó vào năm 2008, Peter Carr đề xuất phương pháp cài đặt thuật toán Background Subtraction của Stauffer và Grimson trên GPU sử dụng thư viện Core Image Kernel Library của Apple [7]. Gần đây nhất là kết quả của cài đặt thuật toán Background Subtraction của Zivkovic [2] trên GPU của Vu Pham [3] sử dụng CUDA của NVIDIA.

Trong bài viết này tôi xin đề xuất thêm một phương pháp cài đặt thuật toán Background Subtraction của Zivkovic trên GPU sử dụng ngôn ngữ lập trình tính toán mở OpenCL [8][9][10]. OpenCL(Open Computing Language) là ngôn ngữ lập trình tính toán mở cho các hệ thông tính toán lai (*heterogenerous computing system*) gồm CPU và GPU. Nó là một ngôn ngữ lập trình song song theo mô hình *data-based parallel* và *task-based parallel.* OpenCL là một chuẩn mở được phát triển bởi Khronos Group và được hỗ trợ bởi rất nhiều các hãng sản xuất chíp lớn như: Intel, AMD, NVIDIA, ARM….Do đó chương trình viết bằng OpenCL có thể chạy trên rất nhiều các nên tảng khác nhau. Bởi vì OpenCL được phát triển với mục đích đa nên tảng, nếu so sánh với CUDA ngôn ngữ được phát triển riêng cho vi xử lý của NVIDIA thì hiệu năng tính toán của chương trình viết bằng OpenCL không thể vượt qua được chương trình được viết bằng CUDA (chương trình), nhưng bù lại chương trình viết bằng OpenCL có thể chạy tốt trên các bộ vi xử lý của cả NVIDIA và AMD mà không cần phải viết lại chương trình. Chi tiết về hiệu năng tính toán của CUDA và NVIDA xin xem trong phần 5 của bài viết này hoặc có thể tham khảo thêm trong [13][14].

2. Mô hình trộn Gaussian hữu hạn.

Biến ngẫu nhiên $\boldsymbol{x} = (x_1, \dots, x_d)^T$ có hàm mật độ xác suất cho bởi:

$$p(\boldsymbol{x}|\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \quad (1)$$

với $\mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) = \frac{1}{(2\pi)^{\frac{d}{2}}.|\boldsymbol{\Sigma}_k|^{\frac{1}{2}}} e^{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}_k{}^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_k)}$, ta nói biến ngẫu nhiên $\boldsymbol{x}$ tuân theo mô hình trôn Gaussian.Trong đó $\boldsymbol{\pi} \equiv \{\pi_1, \dots, \pi_K\}$, $\boldsymbol{\mu} \equiv \{\boldsymbol{\mu_1}, \dots, \boldsymbol{\mu}_K\}$, $\boldsymbol{\Sigma} \equiv \{\boldsymbol{\Sigma}_1, \dots, \boldsymbol{\Sigma}_K\}$ là các tham số của mô hình, với $\pi_i \in [0; 1]$ và $\sum_{k=1}^{K} \pi_k = 1$ (Với các đại lượng vector sẽ được viết đậm, còn các đại lượng vô hướng sẽ được viết thường).

Cho tập dữ liệu $\mathbf{X} \equiv \{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(t)}\}$, để ước lượng các tham số cho mô hình ta giải bài toán cực trị sau:

$$\widehat{\boldsymbol{\theta}} = argmax_{\boldsymbol{\theta}} \ln p(\boldsymbol{X}|\boldsymbol{\theta}), \text{ trong đó } \boldsymbol{\theta} \equiv \{\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}\}. \quad (2)$$

Bài toán trên được biết với tên gọi *cực đại hàm lòng tin* (Maximize Likelihood function). Bài toán trở lên phức tạp hơn nhiều bài toán cực đại hàm lòng tin với $p(\boldsymbol{X}|\boldsymbol{\theta})$ là hàm phân phối Gaussian (K=1) vì xuất hiện *ln* của tổng các phân phối Gaussian. Để giải bài toán ta đưa thêm khái niệm biến ngẫu nhiên ẩn $\boldsymbol{z} = (z_1, \dots, z_K)^T$ (*laten random variable* hay *unobserved random variable*) với $z_k \in \{0, 1\}$ và $\sum_{k=1}^{K} z_k = 1$. Hàm phân phối của **z** được định nghĩa như sau: $p(\boldsymbol{z}) = \prod_{k=1}^{K} \pi_k{}^{z_k}$, và phân phối có điều kiện của biến ngẫu nhiên **x** trong điều kiện **z** được định nghĩa như sau: $p(\boldsymbol{x}|\boldsymbol{z}) = \prod_{k=1}^{K} \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)^{z_k}$. Phân phối đồng thời của **x, z** là :

$$p(\boldsymbol{x}, \boldsymbol{z}|\boldsymbol{\theta}) = p(\boldsymbol{z}|\boldsymbol{\theta})\, p(\boldsymbol{x}|\boldsymbol{z}, \boldsymbol{\theta}) = \prod_{k=1}^{K} \pi_k{}^{z_k} \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)^{z_k} \quad (3)$$

và $$p(\boldsymbol{x}|\boldsymbol{\theta}) = \sum_{\boldsymbol{z}} p(\boldsymbol{z}|\boldsymbol{\theta}) p(\boldsymbol{x}|\boldsymbol{z}, \boldsymbol{\theta}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k). \quad (4)$$

Thuật toán EM (*Expectation Maximization*) được sử dụng phổ biến để ước lượng tham số cho mô hình trộn Gaussian là một thủ tục lặp để tìm kiếm nghiệm cho bài toán tối ưu (2). Bắt đầu từ một giá trị tham số khởi tạo bất kỳ $\boldsymbol{\theta}^{(0)}$, thuật toán được chạy cho đến khi điều kiện dừng được thỏa mãn (khi tham $\boldsymbol{\theta}$ hoặc $ln\, p(\boldsymbol{X}|\boldsymbol{\theta})$ không còn thay đổi nhiều). Mỗi vòng lặp thuật toán thực hiện 2 bước:

E step: $\mathcal{Q}(\boldsymbol{\theta}, \boldsymbol{\theta}^{(k-1)}) = \sum_{\boldsymbol{z}} p(\boldsymbol{Z}|\boldsymbol{X}, \boldsymbol{\theta}^{(k-1)})\, ln\, p(\boldsymbol{X}, \boldsymbol{Z}|\boldsymbol{\theta})$ *(5)*

M step: $\boldsymbol{\theta}^{(k)} = arg\, max_{\boldsymbol{\theta}}\, \mathcal{Q}(\boldsymbol{\theta}, \boldsymbol{\theta}^{(k-1)})$ *(6)*

Chi tiết về mô hình trộn Gaussian và thuật toán EM các bạn có thể đọc thêm trong [15], [16], [17], [18].

3. Thuật toán Background Subtraction sử dụng mô hình trộn Gaussian của Zivkovic.

Trong thuật toán này, mỗi pixel được mô hình hóa như là một vector ngẫu nhiên 3 chiều $\mathbf{x}(\mathrm{m}, \mathrm{n}) = (R_{m,n}, G_{m,n}, B_{m,n})^T$ tương ứng với các kênh (*chanel*) của ảnh RGB, mỗi bức ảnh là một ma trận hai chiều cỡ *MxN*. Để tránh rườm rà tôi bỏ ký hiệu *m*, *n* và **x** = $(R, G, B)^T$ được hiểu là pixel tại vị trí *(m, n)*. Tại mỗi thời điểm ta chỉ quan sát được thêm một giá trị của pixel **x** ký hiệu là $\boldsymbol{x}^{(t)}$, các giá trị của pixel được giả thiết tuân theo phân phối trộn gaussian tức là $p(\boldsymbol{x}|\boldsymbol{\theta}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$ với là tham số của mô hình như trong phần trên. Để thuật toán có thể thích nghi được với những thay đổi của *Background* (quan sát trong điều kiện ánh sáng thay đổi, những vật đang là *foreground* sau đó không còn chuyển động nữa và trở thành một phần của *Background, ...*), sau mỗi lần quan sát được thêm một giá trị của pixel các tham số của mô hình cần được update lại, và mẫu ngẫu nhiên $\boldsymbol{X} = \{\boldsymbol{x}^{(t)}, \dots, \boldsymbol{x}^{(t-T)}\}$ dùng để ước lượng tham số cho mô hình được xây dựng như sau: ta chỉ lấy T quan sát gần nhất, mỗi khi quan sát được một giá trị mới ta đưa vào mẫu và loại bỏ quan sát cũ nhất. Bằng cách này những quan sát trong quá khứ quá xa sẽ không còn có giá trị do đó thuật toán có thể xấp xỉ tốt *background* ngay cả khi *background* thay đổi theo thời gian. Ngoài ra, để mô hình có thể xấp xỉ tốt *background* số lượng thành phần của mô hình đóng vai trò quan trọng, nếu số thành phần quá ít thì dẫn đến vấn đề *underfitting*, nếu số lượng thành phần quá nhiều thì dẫn đến vấn đề *overfitting*. Để có thể ước lượng được số thành phần phù hợp cho mô hình trộn Gaussian ta sử dụng phương pháp cực đại phân phối sau (*Maximize Posteriori*) thay cho phương pháp cực đại hàm lòng tin (*Maximize Likelihood function*).

$$\widehat{\boldsymbol{\theta}} = arg\, max_{\boldsymbol{\theta}}\{ln\, p(\boldsymbol{X}|\boldsymbol{\theta}(K)) + ln\, p(\boldsymbol{\theta}(K))\} \quad (7)$$

với K là số thành phần của mô hình trộn, là một tham số cần ước lượng, $p(\boldsymbol{\theta}(K))$ là hàm phân phối trước (*prior distribution*) của tham số $\boldsymbol{\pi}(K) \equiv \{\pi_1, \dots, \pi_K\}$ (Theo quan điểm của Bayes tham số cần ước lượng cũng được coi là biến ngẫu nhiên). Để ước lượng số thành phần cho mô hình ta xuất phát từ một giá trị K đủ lớn, trong quá trình update các tham số

ta loại bỏ dần các thành phần không hợp lệ, những thành phần có quá ít quan sát được xem như không hợp lệ và bị loại bỏ dần cho đến khi mô hình đạt trạng thái cân bằng. Để bài toán trở lên đơn giản ta chọn $p(\boldsymbol{\theta}(K)) \propto \prod_{k=1}^{K} \pi_k{}^{c_k}$ là hàm phân phối Dirichlet. Với cách chọn này ta có công thức ước lượng đúng cho các tham số (vì $p(\boldsymbol{z}|\,\boldsymbol{\theta}(K)) = \prod_{k=1}^{K} \pi_k{}^{z_k}$ có phân phối đa thức (*Multinominal distribution*) nên $p(\boldsymbol{\theta}(K))$ là một *conjugate prior distribution* cho phân phối đa thức). Ta chọn $c_k = -\frac{N}{2}$ với $k=1, \ldots, K$ theo như [17]. Trong đó N là số lượng tham số cần ước lượng cho mỗi thành phần của mô hình.

Nghiệm của (7) là nghiệm của phương trình:

$$\frac{\partial}{\partial \boldsymbol{\theta}}\{ln\, p(\boldsymbol{X}|\boldsymbol{\theta}(K)) + ln\, p(\boldsymbol{\theta}(K)) + \lambda(\textstyle\sum_{k=1}^{K} \pi_k - 1)\} = 0 \quad (8)$$

hay: $\frac{\partial}{\partial \boldsymbol{\theta}}\left\{\sum_{n=1}^{t} ln\{\sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x}^{(n)}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)\} - \frac{c}{2}\sum_{k=1}^{K} \ln \pi_k + \lambda(\sum_{k=1}^{K} \pi_k - 1)\right\} = 0$ (9)

Trong đó c_k= -c, và c = $\frac{N}{2}$. Giải (9) ta có:

$$\hat{\pi}_k^{(t)} = \frac{\hat{\Pi}_k^{(t)} - \frac{c}{t}}{1 - K\frac{c}{t}} \quad (10)$$

$$\boldsymbol{\mu}_k^{(t)} = \frac{1}{N_k}\sum_{n=1}^{t} \gamma^{(t)}(z_k^{(n)})\, \boldsymbol{x}^{(n)} \quad (11)$$

$$\boldsymbol{\Sigma}_k^{(t)} = \frac{1}{N_k}\sum_{n=1}^{t} \gamma^{(t)}(z_k^{(n)})\, (\boldsymbol{x}^{(n)} - \boldsymbol{\mu}_k^{(n)})(\boldsymbol{x}^{(n)} - \boldsymbol{\mu}_k^{(n)})^T \quad (12)$$

Trong đó $\hat{\Pi}_k^{(t)} = \frac{1}{t}\sum_{n=1}^{t} \gamma^{(t)}(z_k^{(n)})$, $N_k = \sum_{n=1}^{t} \gamma^{(t)}(z_k^{(n)})$

và $\gamma^{(t)}\left(z_k^{(n)}\right) \equiv p\left(z_k^{(n)} = 1 \middle| \boldsymbol{x}^{(n)}\right) = \frac{\pi_k \mathcal{N}(\boldsymbol{x}^{(n)}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{l=1}^{K} \pi_l \mathcal{N}(\boldsymbol{x}^{(n)}|\boldsymbol{\mu}_l, \boldsymbol{\Sigma}_l)}$

Chi tiết giải phương trình (9) xem trong [15] chapter 9 và [16].

Trong thực tế, có nhiều bài toán mà tại mỗi thời điểm ta chỉ có thể quan sát được một giá trị của **x**, nếu sử dụng (10), (11), (12) sẽ rất tốn kém bộ nhớ và chi phí tính toán rất cao vì mỗi lần quan sát được một giá trị của **x** ta phải sử dụng lại tất cả các quan sát cũ để update các tham số. Vì vậy cần phải xây dựng một thuật toán update hồi quy (*Recursive update*) sao cho mỗi lần update ta chỉ cần sử dụng giá trị vừa quan sát được và kết quả update tại thời điểm trước đó. Phương trình update hồi quy cho mô hình như sau:

$$\hat{\pi}_k^{(t+1)} = \hat{\pi}_k^{(t)} + (t-1)^{-1}\left(\frac{\gamma(z_k^{(t)})}{1 - K\frac{c}{T}} - \hat{\pi}_k^{(t)}\right) - (1-t)^{-1}\frac{\frac{c}{T}}{1 - K\frac{c}{T}} \quad (13)$$

$$\hat{\boldsymbol{\mu}}_k^{(t+1)} = \hat{\boldsymbol{\mu}}_k^{(t)} + (t+1)^{-1}\frac{\gamma^{(t)}\left(z_k^{(t+1)}\right)}{\hat{\pi}_k^{(t)}}(\boldsymbol{x}^{(t+1)} - \hat{\boldsymbol{\mu}}_k^{(t)}) \quad (14)$$

$$\widehat{\boldsymbol{\Sigma}}_k^{(t+1)} = \widehat{\boldsymbol{\Sigma}}_k^{(t)} + (t+1)^{-1}\frac{\gamma^{(t)}\left(z_k^{(t+1)}\right)}{\hat{\pi}_k^{(t)}}\left(\left(\boldsymbol{x}^{(t+1)} - \widehat{\boldsymbol{\mu}}_k^{(t)}\right)\left(\boldsymbol{x}^{(t+1)} - \widehat{\boldsymbol{\mu}}_k^{(t)}\right)^T - \widehat{\boldsymbol{\Sigma}}_k^{(t)}\right) \quad (15)$$

Phương trình (13) được xây dựng từ (10) với việc giữ cố định $c/t = c/T$ (khi đó $\hat{\pi}_k$ là một ước lượng chệch của π_k, tuy nhiên với T lớn thì độ chệch này rất nhỏ có thể chấp nhận được). Với thuật toán này chi phi cho mỗi lần update tham số giảm đi rất nhiều. Chi tiết về thuật toán update hồi quy cho mô hình trộn Gaussian hữu hạn các bạn có thể đọc thêm trong [2], [16], [18]. Tuy nhiên để có thể sử dụng thuật toán trong các ứng dụng thời gian thực đòi thời gian tính toán rất thấp, các phương trình update tham số (13), (14), (15) chi phí tính toán vẫn còn khá cao do phải tính hàm $\gamma(z_k^{(t)})$. Vì $\gamma(z_k^{(t)})$ là xác suất để $\boldsymbol{x}^{(t)}$ thuộc thành phần thứ *k* trong điều kiện $\boldsymbol{x}^{(t)}$ đã được quan sát, nên ta có thể xấp xỉ $\gamma(z_k^{(t)})$ như sau nếu $\boldsymbol{x}^{(t)}$ đủ gần thành phần *k* thì $\gamma(z_k^{(t)})$ = 1, các hàm còn lại được coi bằng 0. Quan sát $\boldsymbol{x}^{(t)}$ đủ gần thành phần *k* được định nghĩa như sau:

$$D_k^2\left(\boldsymbol{x}^{(t+1)}\right) = \left(\boldsymbol{x}^{(t+1)} - \widehat{\boldsymbol{\mu}}_k^{(t)}\right)^T \widehat{\boldsymbol{\Sigma}}_k^{(t)^{-1}}\left(\boldsymbol{x}^{(t+1)} - \widehat{\boldsymbol{\mu}}_k^{(t)}\right) < c_{close} \quad (16)$$

Với c_{close} là một hằng số.

Ngoài ra ta cũng giả thiết rằng các pixel trong cùng một frame độc lập và các kênh *(Red, Green, Blue)* của frame biến đổi đồng đều, khi đo $\boldsymbol{\Sigma} = \sigma^2\boldsymbol{I}$. Và $1 - K\frac{c}{t} \approx 1$ khi t lớn. Ta có các phương trình update tham số cho mô hình trộn Gaussian của mỗi pixel như sau:

$$\hat{\pi}_k^{(t)} = \hat{\pi}_k^{(t-1)} + \alpha\left(\gamma^{(t)}\left(z_k^{(t+1)}\right) - \hat{\pi}_k^{(t-1)}\right) - \alpha c_T \quad (17)$$

$$\widehat{\boldsymbol{\mu}}_k^{(t)} = \widehat{\boldsymbol{\mu}}_k^{(t-1)} + \gamma^{(t)}\left(z_k^{(t+1)}\right)\left(\frac{\alpha}{\hat{\pi}_k^{(t)}}\right)\left(\boldsymbol{x}^{(t)} - \widehat{\boldsymbol{\mu}}_k^{(t-1)}\right) \quad (18)$$

$$\hat{\sigma}_k^{2(t)} = \hat{\sigma}_k^{2(t-1)} + \gamma^{(t)}\left(z_k^{(t+1)}\right)\left(\frac{\alpha}{\hat{\pi}_k^{(t)}}\right)\left(\left(\boldsymbol{x}^{(t)} - \widehat{\boldsymbol{\mu}}_k^{(t-1)}\right)^T\left(\boldsymbol{x}^{(t)} - \widehat{\boldsymbol{\mu}}_k^{(t-1)}\right) - \hat{\sigma}_k^{2(t-1)}\right) \quad (19)$$

Trong đó $c_T = \frac{c}{T}$ và $\alpha = \frac{1}{T}$. Thủ tục update tham số cho mô hình được thực hiện như sau: Khi quan sát được $\boldsymbol{x}^{(t)}$ ta update tham số của mô hình theo (17), (18), (19), loại bỏ những thành phần có $\hat{\pi}_k^{(t)} < 0$, nếu $\boldsymbol{x}^{(t)}$ không đủ gần bất cứ thành phần nào của mô hình ta thêm vào mô hình một thành phần mới có $\hat{\pi}_{K+1} = \alpha$, $\widehat{\boldsymbol{\mu}}_{K+1} = \boldsymbol{x}^{(t)}$, $\hat{\sigma}_{K+1}^2 = \sigma_0^2$ nếu K nhỏ hơn số thành phần cực đại của mô hình, nếu K đạt cực đại thì ta thay thế thành phần có $\hat{\pi}_k$ nhỏ nhất. Các tham sô $\hat{\pi}_k^{(t)}$ của mô hình được sắp xếp giảm dần và chuẩn hóa sau mỗi lần update.

Sau khi update các tham số cho mô hình ta cần xác định $\boldsymbol{x}^{(t)}$ là *Foreground (FG)* hay *Background (BG)*, để làm được điều này ta gọi $p(\boldsymbol{x}^{(t)}|BG)$ là xấp xỉ cho *Background*, $p(\boldsymbol{x}^{(t)}|FG)$ là xấp xỉ cho *Foreground*, còn $p(\boldsymbol{x}^{(t)}|BG + FG)$ là xấp xỉ cho $\boldsymbol{X} = \{\boldsymbol{x}^{(t)}, \dots, \boldsymbol{x}^{(t-T)}\}$ là mô hình trộn Gaussian mà ta vừa update tham số cho mô hình. $\boldsymbol{x}^{(t)}$ được xác định là thuộc BG nếu

$$\frac{p(BG|\boldsymbol{x}^{(t)})}{p(FG|\boldsymbol{x}^{(t)})} = \frac{p(\boldsymbol{x}^{(t)}|BG)p(BG)}{p(\boldsymbol{x}^{(t)}|FG)p(FG)} > 1 \quad (20)$$

Vì ta không có thông tìn gì về $\boldsymbol{x}^{(t)}$ có thể là BG hay FG nên ta giả sử $p(BG) = p(FG) = 0.5$, và giả sử sự xuất hiện của FG tuân theo luật phân phối đều $p(\boldsymbol{x}^{(t)}|FG) = c_{FG}$, từ đó ta có $\boldsymbol{x}^{(t)}$ là BG nếu $p(\boldsymbol{x}^{(t)}|BG) > c_{thr}(=c_{FG})$. Và $p(\boldsymbol{x}^{(t)}|BG)$ được xác định như sau:

$$p(\boldsymbol{x}^{(t)}|BG) \propto \sum_{k=1}^{B} \pi_k \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \quad (21)$$

Trong đó $B = \text{argmin}_b\left(\sum_k^b \hat{\pi}_k > (1 - c_f)\right)$ (22)

Với c_f là tỉ lệ dữ liệu lớn nhất thuộc về FG, ví dụ nếu một vật đang chuyển động (FG) sau đó dừng lại thì $\hat{\pi}_{B+1}$ sẽ tăng dần cho đến khi $\hat{\pi}_{B+1} > c_f$ vật đó sẽ trở thành FG. Từ phương trình (17) ta có thể tính được số frame lớn nhất để một vật đang chuyển động sau đo dừng lại trở thành BG là $\frac{\ln(1-c_f)}{\ln(1-\alpha)}$, với $c_f = 0.1$ và α = 0.001 thì ta cần 105 frame. (chú ý $\hat{\pi}_k^{(0)} = \alpha$, $\hat{\pi}_k^{(t)} \approx (1-\alpha)\hat{\pi}_k^{(t-1)} + \alpha$. Tính được $\hat{\pi}_k^{(t)} \approx \alpha \sum_{n=0}^{t}(1-\alpha)^n = 1 - (1-\alpha)^{t+1}$).

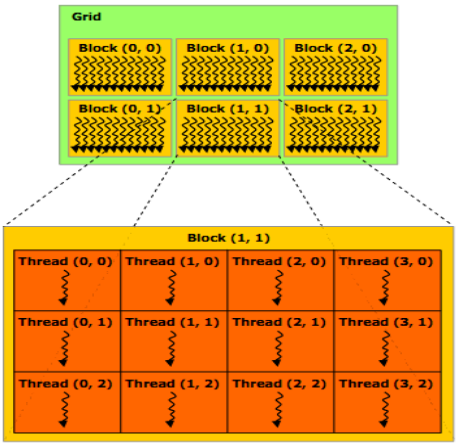
4. Cài đặt thuật toán trên GPU.

GPU là thiết bị tính toán được thiết kế riêng cho xử lý đồ họa, có khả năng thực thi đồng thời hàng nghìn *thread*. Với khả năng này GPU cũng rất phù hợp cho các bài toán mô phỏng, computer vision, machine learning, data mining,… Để lập trình trên GPU ta có thể dùng các ngôn ngữ lập trình như CUDA, OpenCL, DirectCompute, Core Image Kernel Language (CIKL), Cg (C for graphic) … Hiện nay, hai ngôn ngữ lập trình OpenCL và CUDA được sử dụng phổ biến hơn cả cho các bài toán đòi hỏi chi phí tính toán cao. OpenCL với ưu thế là một chuẩn mở, được hỗ trợ bởi nhiều nhà sản xuất vi xử lý lớn như Intel, NVIDIA, AMD, ARM, IBM,….CUDA ngôn ngữ lập trình được thiết kế riêng cho NVIDA GPU nên có được hiệu năng tính toán vượt trội so với các ngôn ngữ khác.

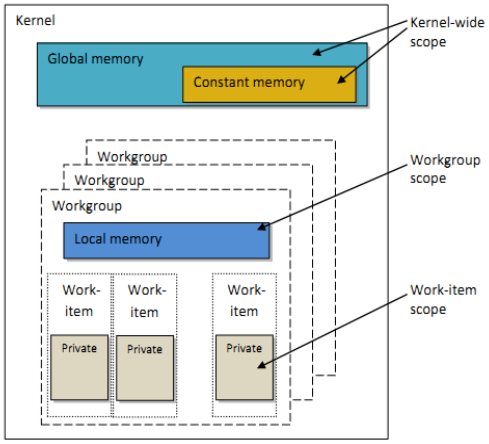
4.1. Mô hình tính toán song song của OpenCL.

OpenCL sử dụng mô hình tính toán song song data-based parallel, bài toán lớn (*Grid*) được chia thành nhiều bài toán nhỏ (*Block*) thực thi song song đồng thời trên các tập dữ liệu khác nhau một cách độc lập. Mỗi bài toán nhỏ được chia thành các bài toán nhỏ hơn (*Thread*) thực thi song song trên mỗi phần tử của tập dữ liệu. Các *thread* trong

cùng một *block* có thể trao đổi thông tin với nhau thông qua *shared-memory*, và một số công cụ giúp đồng bộ hóa giữa các *thread* trong cùng một *block* (xem hình 1). Các *thread* thực thi cùng một chương trình gọi là *kernel*.



Hình 1: *Mô hình tính toán song song trên GPU.*



Hình 2: *Mô hình bộ nhớ trong OpenCL.*

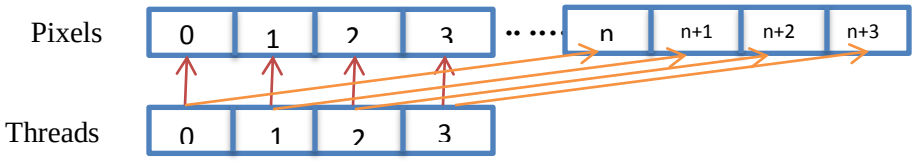
Mỗi *thread* được cấp phát riêng một số lượng thanh ghi cần thiết để thực thi, một bộ đếm chương trình riêng, do đó các *thread* được tự do rẽ nhánh khác với kiến trúc Single Instruction Multi-Data (SIMD) trên CPU. Mỗi *thread* cũng được cấp phát riêng một vùng nhớ gọi là *private memory space*, các *thread* trong cùng một *block* có thể truy cập đến cùng một vùng nhớ được gọi là *local memory (shared memory)*, và các *thread* trong cùng một *grid* có thể truy cập đến cùng một không gian nhớ được gọi là *global memory* (xem hình 2). Chi tiết xin xem trong [8][9][10][11].

4.2. Cài đặt thuật toán.

a. Tối ưu truy cập bộ nhớ.

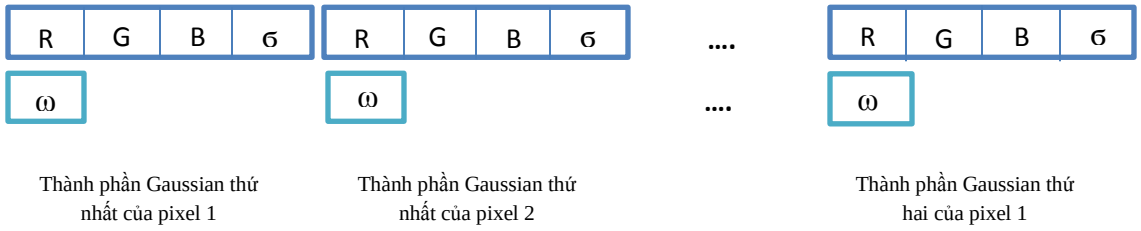
Chiến lược tối ưu quan trọng nhất khi lập trình trên GPU (NVIDIA GPU, AMD ATI GPU) có lẽ là tối ưu đọc ghi bộ nhớ, vì tốc độ đọc ghi bộ nhớ chậm hơn rất nhiều so với tốc độ tính toán của vi xử lý, hạn chế tối đa việc đọc ghi dữ liệu được đặt lên hàng đầu khi lập trình trên GPU. Yêu cầu ghi, đọc bộ nhớ của một *half-warp* (16 *threads* liên tiếp trong cùng một *block*, *half-warp* đầu tiên chứa *thread* đầu tiên của *block*) được gộp chung lại thành một giao dịch (*Transaction*) nếu thỏa mãn một số yêu cầu như các thread trong cùng một *half-warp* truy cập đến các vùng nhớ liên tiếp, và vùng nhớ được truy cập phải được sắp hàng (*aligned*) đến 16 lần cỡ của phần tử đang được truy cập. Xem thêm tại [8][11].

Để đạt được những yêu cầu trên, *input image* được được chuyển đổi thành *4-channel image* (RGBA), mỗi pixel chiếm 4 bytes. Bằng cách này mỗi *thread* sẽ truy cập đến các pixel theo bước có độ lớn bằng số *thread* trong *grid* là bội số của 32 (hình 3).



Hình 3: *Các thread truy cập đến các pixel theo bước n có độ lớn bằng số thread trong grid*

Ngoài ra, ta phải lưu trữ một lượng lớn các tham số cho mô hình trộn gaussian, việc thiết kế một cấu trúc dữ liệu phù hợp nhằm tối ưu truy xuất là rất quan trọng, ảnh hưởng lớn đến hiệu năng tính toán của chương trình. Thay vì sử dụng cấu trúc dữ liệu AoS (Array of Structures) như bản cài đặt trên CPU, tôi sử dụng cấu trúc dữ liệu SoA (Structure of Array). Thay vì đặt các tham số của mô hình của mỗi pixel vao cùng một *struct*, tôi lưu vector kỳ vọng (μ) và phương sai (σ) vào cùng một struct, còn trọng số (ω) được lưu trữ riêng, và các tham số của mỗi thành phần gaussian của một mô hình trộn Gaussian được lưu trữ theo bước có độ lớn bằng số pixel của *input image* (hình 4).



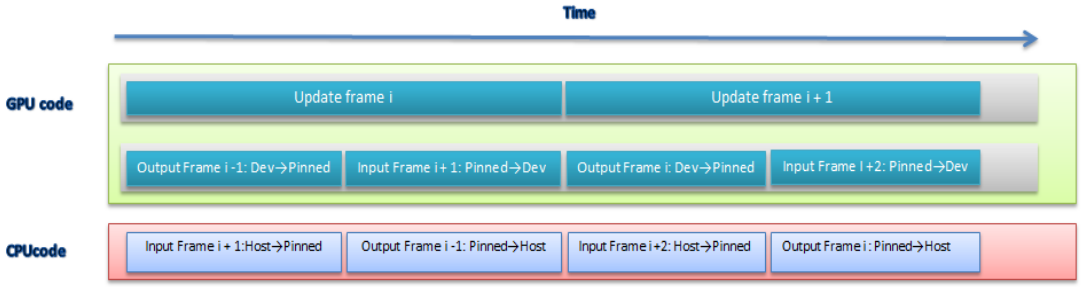
Hình 4: *Cấu trúc dữ liệu cho các tham số.*

b. Sử dụng Pinned memory.

Pinned memory hay còn gọi là page-locked memory, sử dụng pinned memory giúp giảm bớt chi phí chuyển dữ liệu *từ host memory* (RAM) qua *GPU memory* (tốc độ có thể đạt 5Gbps trên hệ thống PCIe x 16 Gen2). Ngoài ra việc sử dụng *pinned memory* có thể thực hiện chuyển dữ liệu *từ host memory* (RAM) qua *GPU memory* trong khi GPU đang thực hiện tính toán.

c. Thực hiện chuyển dữ liệu và tính toán song song.

Để giảm bớt thời gian trễ của việc chuyển dữ liệu từ *host memory* qua *GPU memory* tôi thực hiên việc chuyển dữ liệu song song với việc GPU đang update tham số cho mô hình (xem hình 4).



Hình 4: *Chuyển dữ liệu qua GPU trong khi GPU đang thực hiện update các tham số cho mô hình và CPU đang decode video và chuyển dữ liệu qua pinned memory*

và CPU đang thực hiên decode video và chuyển dữ liệu qua *pinned memory*. Để thực hiện được điều này tôi sử dụng 2 *command queue*, một cho thực thi *kernel* và một cho chuyển dữ liệu qua *GPU memory*. Và thực hiện đồng bộ hóa giữa thực thi *kernel*, chuyển dữ liệu giữa *CPU memory* và *GPU memory* một cách hợp lý.

d. Tối ưu hóa các lệnh (instruction).

Cũng giống như trên CPU, để thực hiện một phép nhân GPU phải thực thi qua nhiều nhịp đồng hồ (*clock cycle*) tùy theo các phiên bản GPU. Để giảm bớt chi phi tính toán, tôi thay thê hoàn toàn các phép nhân trong tính toán địa chỉ các tham số của mô hình trên *global memory* bằng phép cộng. Tuy việc này làm tăng số lượng thanh ghi cần sử dụng cho mỗi *thread* (mỗi *thread* sử dụng 30 thanh ghi) làm giảm số lượng *thread* có thể thực thi song song, nhưng thời gian tính toán của mỗi *thread* lại giảm, và cuối cùng tổng chi phí tính toán đã giảm đáng kể.

5. Kết quả.

Thí nghiệm được thực hiện trên hệ thống: Intel P4 3.0GHz CPU, 1G RAM, GeForce GT440 GPU, với dữ liệu đầu vào là 4 đoạn video dài 1m40s (2426 frames) với kích cỡ lần lượt là 960x540, 1024x576, 1280x720 (hd720), 1920x1080 (hd1080). Để có thể đo được chính xác hiệu năng tính toán của các bản cài đặt tôi thực hiện như sau: tôi đo tổng thời gian tính toán của các frame (2426 frames) trong đoạn video trên rồi chia cho tổng số frame được xử lý, kết quả thu được trong bảng 1. Ngoài ra tôi cũng đo thời gian tính toán một frame lớn nhất của hai bản cài đặt trên GPU sử dụng OPENCL và CUDA, kết quả được ghi lại trong bảng 2.

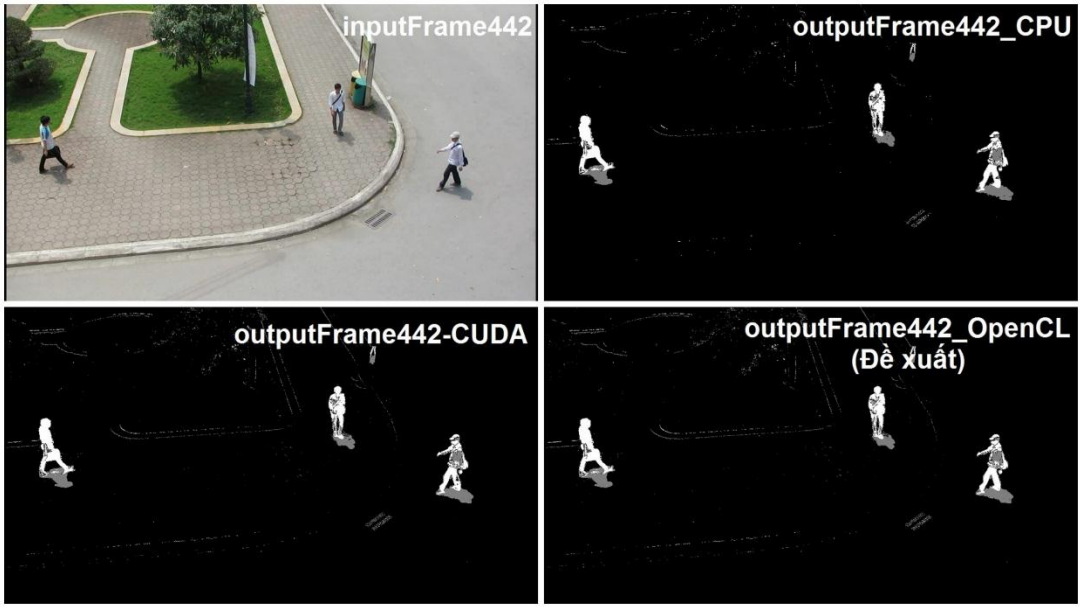
Theo như kết quả thí nghiệm, độ chính xác của các bản cài đặt không có sự khác biệt (hình 5) và hiệu năng của bản cài đặt trên GPU sử dụng OpenCL (đề xuất) vượt trội hơn hẳn so với các bản cài đặt khác (bảng 1, bảng 2), tuy nhiên khi tăng kích cỡ video (lớn hơn 1920x1080) hiệu năng của bản cài đặt của tôi xấp xỉ bản cài đặt của Phạm Vũ.

Frame size	CPU version *(Zivkovic)*	GPU OpenCL version *(đề xuất)* ***(1)***	GPU CUDA version *(Phạm Vũ)* ***(2)***	$\frac{(1)-(2)}{(2)}$ %
960x540	11.9fps(83.8 ms/frame)	720 fps(1.4 ms/frame)	584.3 fps(1.7 ms/frame)	23.3%
1024x576	10.4fps (95.8ms/frame)	625.7 fps(1.6 ms/frame)	517.4 fps(1.9 ms/frame)	20.9%
1280x720 (hd720)	6.7fps (149.2ms/frame)	407.6 fps(2.5 ms/frame)	356.4 fps(2.8 ms/frame)	14.3%
1920x1080 (hd1080)	3.0fps(330.3ms.frame)	188.3 fps(5.3 ms.frame)	179.2 fps(5.6 ms/frame)	0.1%

Bảng 1*: Số frame được xử lý trong một giây và thời gian xử lý một frame của các bản 3 bản cài đặt với các kích cỡ khác nhau. Cột cuối cùng là so sánh hiệu năng giữa bản cài đặt của tôi và bản cài đặt của Phạm Vũ.*

Lần đo thứ	1	2	3	4	5	6	7	8	9	10	TB
CUDA (fps)	409	404	433	411	421	415	417	387	405	424	412.6
OPENCL(fps)	417	468	476	474	476	478	466	470	464	473	466.2

Bảng 2*: Đo tốc độ tính toán nhỏ nhất của hai bản cài đăt CUDA(Phạm Vũ) và OPENCL (đề xuất), thực hiện trên video có cỡ 960x540, 2426 frames. Tốc độ thực thi nhỏ nhất của bản cài đăt bằng OPENCL luôn cao hơn bản cài đặt bằng CUDA, trung bình 13.12%.*



Hình 5: *Kết quả tách foreground của các bản cài đặt. Phía trên bên trái là input frame, phía trên bên phải là kết quả của bản cài đặt trên CPU, phía dưới bên trái là kết quả bản cài đặt bằng CUDA, phía dưới bên trái là kết quả bản cài đặt của tôi sử dụng OpenCL.*

6. Kết luận.

Trong bài viết này tôi đã giới thiệu với các bạn mô hình trộn gaussian hữu hạn và phương pháp ước lượng tham số cho mô hình. Mô hình trộn Gaussian là một trong những mô hình được sử dụng rộng rãi trong các thuật toán *unsupervised learning* nhất hiên nay. Tôi cũng giới thiệu với các bạn một thuật toán Background Subtraction được sử dụng phổ

biến trong lĩnh vực thị giác máy tính, kết quả của thuật toán là dữ liệu đầu vào cho các thuật toán khác. Độ chính xác của thuật toán ảnh hưởng lớn đến kết quả của các thuật toán tiếp sau. Thuật toán Background Substraction, trong nhiều hệ thống, có thời gian tính toán lớn nhất, do đó việc giảm bớt thời gian tính toán cũng rất quan trong, đặc biệt là trong các hệ thống thời gian thực (*real-time system*). Trong tương lai tôi sẽ nghiên cứu cài đặt thêm các thuật toán tracking motion (*particle filters*), body action analysis, face recognition,… sử dụng ngôn ngữ lập trình tính toán mở OpenCL. Các bạn có thể download mã nguồn của dự án tại: http://code.google.com/p/gpu-gmm-bg-sub/downloads/list

Tài liệu tham khảo:

[1]. **Z.Zivkovic**, "Improved adaptive Gausian mixture model for background subtraction", International Conference Pattern Recognition, Vol.2, pages: 28-31, 2004.

[2]. ***Zivkovic, Z., van der Heijden, F.***, Recursive unsupervised learning of finite mixture models. IEEE Trans. Pattern Anal. Mach. Intell. 26 (5), 651–656, 2004.

.

[3]. **Vu Pham, Phong Vo, Hung Vu Thanh, Bac Le Hoai**, "GPU Implementation of Extended Gaussian Mixture Model for Background Subtraction", IEEE-RIVF 2010 International Conference on Computing and Telecommunication Technologies, Vietnam National University, Nov. 01-04, 2010. DOI: 10.1109/RIVF.2010.5634007.

[4]. **N. Friedman and S. Russell**, "Image segmentation in video sequences: A probabilistic approach," in Thirteenth Conference on Uncertainty in Artificial Intelligence (UAI), 1997, pp. 175–181.

[5]. **C. Stauffer and W. E. L. Grimson**, "Adaptive Background Mixture Models for Real-Time Tracking," in IEEE Conference on Computer Vision and Pattern Recognition, 1999, pp. 246–252.

[6]. **S.-j. Lee and C.-s. Jeong,** "Real-time Object Segmentation based on GPU," in Proc. International Conference on Computational Intelligence and Security, November 2006, pp. 739–742.

[7]. **P. Carr**, "GPU Accelerated Multimodal Background Subtraction," in Proc. Digital Image Computing: Techniques and Applications, December 2008, pp. 279–286.

[8]. **Benedict Gaster, Lee Howes, David R. Kaeli, Perhaad Mistry, Dana Schaa**, "Heterogeneous Computing with OpenCL", Elsevier Inc, 2012.

[9]. **Aaftab Munshi, Benedict R. Gaster, Timothy G. Mattson, James Fung, Dan Ginsburg**, "OpenCL Programming Guide" Addison-Wesley, 2011.

[10]. Khronos OpenCL Working Group, " OpenCL 1.2 Specification".

[11]. NVIDIA corporation, "OpenCL Programming Guide for the CUDA Architecture"

[12]. Microsoft reseach, Visual computing - http://research.microsoft.com/en-us/groups/vc/.

[13]. **Jianbin Fang, Varbanescu, A.L., Sips,** H."A Comprehensive Performance Comparison of CUDA and OpenCL " , Parallel Processing (ICPP), 2011 International Conference on, p216 – 225.

[14]. **Kamran Karimi, Neil G. Dickson, Firas Hamze**, "A Performance Comparison of CUDA and OpenCL", http://arxiv.org/abs/1005.2581

[15]. **Christopher M. Bishop,** Pattern Recognition and Machine Learning, Springer Press 2006.

[16]. **Phạm Duy Tùng**, Ước lượng tham số cho mô hình trộn Gaussian hữu hạn và cài đặt thuật toán Background Subtraction sử dụng mô hình trộn Gaussian của Zivkovic trên GPU, Đồ án tốt nghiệp đại học, Viên Toán ứng dụng và Tin học, Đại học Bách Khoa Hà Nội, 2012.

[17]. **M. Figueiredo and A.K. Jain**, "Unsupervised Learning of Finite Mixture Models," IEEE Trans. Pattern Analysis and Machine Intelligence, vol. 24, no. 3, pp. 381-396, Mar. 2002.

[18]. **Titterington, D., A. Smith, and U. Makov** (1985) "Statistical Analysis of Finite Mixture Distributions," John Wiley & Sons. ISBN 0471907634

[19]. **McLachlan, G.J. and Peel, D.** (2000) *Finite Mixture Models*, Wiley. ISBN 0471006262