Методы ускорения алгоритмов распознавания символов

О. А. Славин

Институт системного анализа Российской академии наук,
Россия, 117312 Москва, пр. 60-летия Октября, 9

В статье рассматриваются различные методы ускорения работы программных модулей, реализующих алгоритмы распознавания образов символов для платформы Windows и процессоров Intel или AMD. Показаны способы использования алгоритмической оптимизации, потоковых инструкций SSE, инструкций GPU и распараллеливания процессов.

Введение

Быстродействие является одной из важнейших характеристик качества систем распознавания текстов (OCR). Время распознавания документа в OCR складывается из суммы времен работы использованных алгоритмов, в том числе алгоритмов распознавания образов символов. В настоящей работе мы будем рассматривать только базовые алгоритмы распознавания символов (АРС), каждый из которых реализует функцию классификации образа отдельного символа без возможностей комбинирования результатов нескольких функций. Результатом работы АРС является набор оценок, отражающих степень близости распознаваемого символа с символами из обучающей последовательности символов.

Необходимость рассмотрения характеристики быстродействия обусловлена различными причинами, такими как технические требования к скорости ввода документов, предъявляемые потребителем OCR, и конкуренция с аналогичными OCR других разработчиков. Собственно говоря, конкуренция по скорости возможна и с другими версиями собственных продуктов одного разработчика. Например, если использование нового АРС одновременно повышает точность и замедляет работу всей OCR в целом, то вопрос о возможности применения этого АРС нуждается в изучении: возможно, что более точный, но более медленный АРС может быть отвергнут, если он не удовлетворяет некоторому многокрите-

риальному условию оценки качества работы системы в целом. Иными словами, оптимизация быстродействия АРС необходима как из-за конкуренции методов и OCR, так и для оценки принципиальной возможности применения конкретного АРС.

Ниже мы будем рассматривать программные реализации АРС, при этом не будут делаться оговорки о предварительной подготовке этих реализаций, а именно о том, что в реализации удалены части программного кода, ненужного для достижения результата, например отладочные средства, и что реализация скомпилирована с оптимизацией по скорости.

В статье будут рассматриваться реализации АРС для операционной системы Windows и процессоров Intel или AMD. Для компиляции исходного кода методов применялись продукты Microsoft Visual C++® 6.0 и Microsoft Visual C++ 2008.

Критерием быстродействия служит *скорость* распознавания символов, вычисляемая как среднее время распознавания одного символа. Во время распознавания символа не включаются расходы времени на извлечение образа из хранилища тестовых символов и другие накладные расходы. Во всех вычислительных экспериментах использовались две тестовые последовательности TS_1 (154 000 образов) и TS_2 (8000 образов), содержащие печатные символы различного качества.

Под *ускорением* понимается отношение скорости распознавания символов АРС к скорости неоптимизированной версии АРС.

1. Примеры алгоритмов распознавания символов, подвергаемых оптимизации

Мы будем рассматривать три АРС:

- $АРС_1$ — метод сравнения с эталонами, которые образуются в результате самообучения [1];
- $АРС_2$ — метод сравнения с фиксированным набором эталонов, который был подготовлен заранее;
- $АРС_3$ — нейронная сеть, типа многослойный перцептрон, описанная в [2].

Эти методы выбраны из-за того, что их реализации включают в себя небольшие по объему функции, которые выполняют большую часть вычислительной работы.

Реализация $АРС_1$ базируется на следующих понятиях:

- представление распознаваемого образа $Rep(I)$, являющееся результатом центрирования бинарного растра I. Пример центрированного растра, нормализованного до размеров 128×64, приведен на рис. 1;

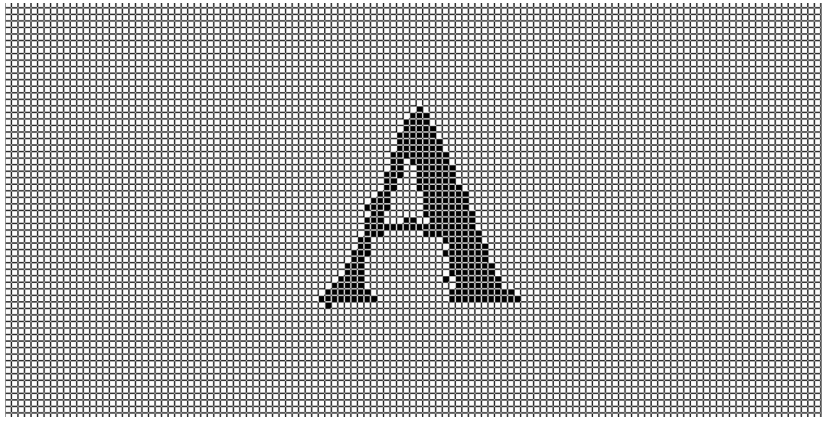

Рис. 1. Нормализованный образ

- набор эталонов $E = \{E_1(S_1), \ldots, E_m(S_m)\}$, каждый из которых представляет собой сумму нескольких центрированных образов, соответствующий одному типу начертания (*графеме*) некоторого символа S_i;
- коллекция альтернатив распознавания $Res(I) = \{(C_1, w_1), \ldots, (C_n, w_n)\}$, где C_i — код символа альтернативы, w_i — оценка надежности распознавания альтернативы;
- функция сравнения центрированного образа с эталоном.

В общем виде считаем, что представление $Rep(I)$ и каждый из эталонов E_i являются элементами пространства $\mathbf{R}^N$, где $N = 128 \times 64$, $\mathbf{R}$ — поле вещественных чисел.

Функция сравнения F двух векторов должна обладать следующими свойствами:

$$F(\mathbf{x}, \mathbf{y}) \geq 0 \text{ и } \forall x \; F(\mathbf{x}, \mathbf{x}) = 0.$$

На рис. 2 приведено распределение времени работы функций метода $АРС_1$. Из графика, построенного с помощью профайлера Intel VTune™ Performance Analyzer 5.0 [3], следует, что большая часть времени (95 %) расходуется всего двумя функциями, реализующими сравнение двух графических образов. Это обстоятельство позволит проводить формальную оптимизацию $АРС_1$ с точки зрения быстродействия.

Метод $АРС_2$ является более простым, чем $АРС_1$, поскольку эталоны и представление являются векторами пространства $\mathbf{R}^n$. Каждый из векторов отнормирован на длину 1: $\forall x \; ||\mathbf{x}|| = 1$. Функция сравнения в $АРС_2$ определяется как скалярное произведение $F(\mathbf{x}, \mathbf{y}) = (\mathbf{x}, \mathbf{y})$.

На рис. 3 приведено распределение времени работы функций метода $АРС_2$, в котором используется 400 эталонов, каждый из которых имеет дли-

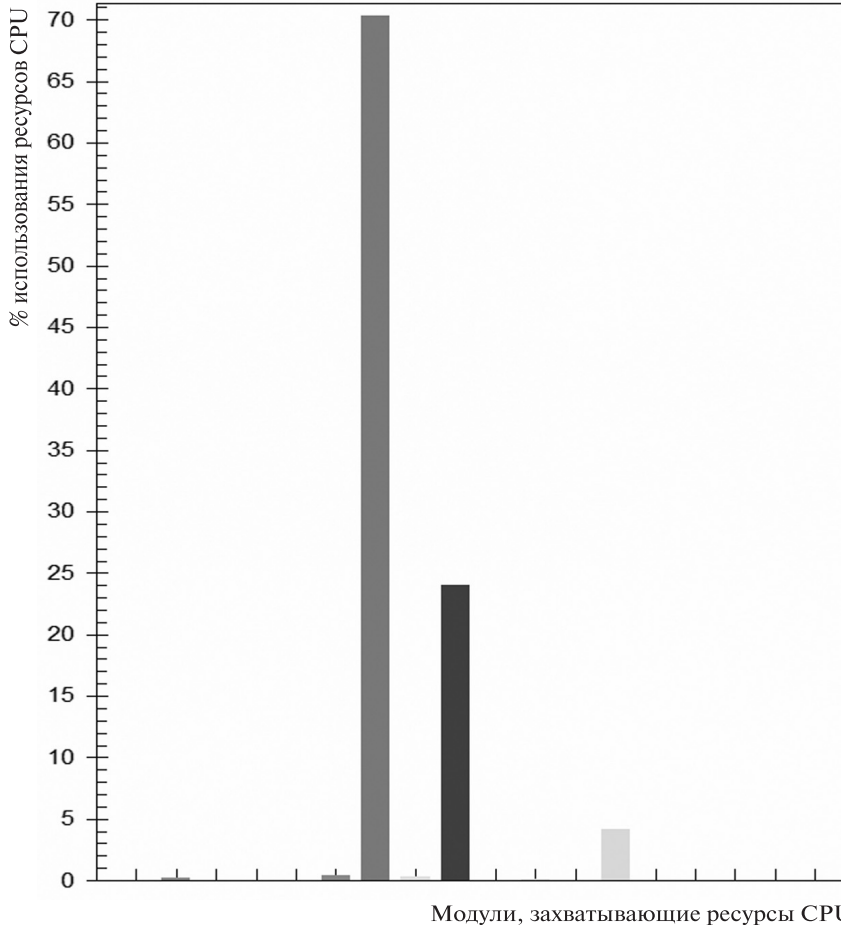

Рис. 2. Профиль работы APC_1

ну 430. Из графика следует, что большая часть времени (90 %) расходуется одной функцией, реализующей вычисление скалярного произведения.

Реализация нейронной сети APC_3 использует представление, являющееся полутоновым образом с размерами 16×16, из которого формируется 2312 признаков. Нейронная сеть APC_3 имеет один внутренний уровень, содержащий 100 узлов, и является полносвязанной, т. е. каждый узел внутреннего уровня соединен со всеми входными узлами, а каждый узел верхнего уровня соединен со всеми узлами внутреннего уровня.

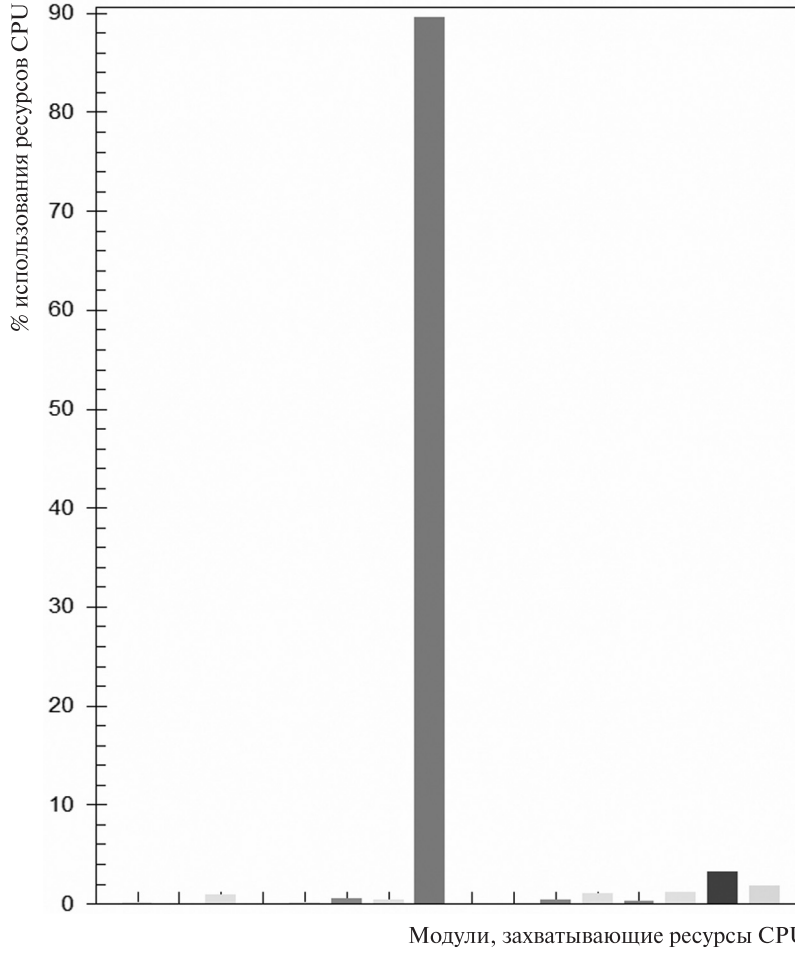

Рис. 3. Профиль работы APC_2

Всего нейронная сеть имеет значительное (более 200 тысяч) число весов, отличающее структуру сети от описанных в работе [4] различных нейронных классификаторов.

На рис. 4 приведено распределение времени работы функций метода APC_3. Из графика следует, что значительная часть времени (50 %) расходуется двумя функциями, реализующими два процесса: скалярное произведение двух векторов $d = \mathbf{x} \cdot \mathbf{y}$, где $\mathbf{x}, \mathbf{y} \in \mathbf{R}^n$, и операцию $\mathbf{z} = \mathbf{x} + \alpha \cdot \mathbf{y}$, где $\mathbf{x}, \mathbf{y} \in \mathbf{N}^m$.

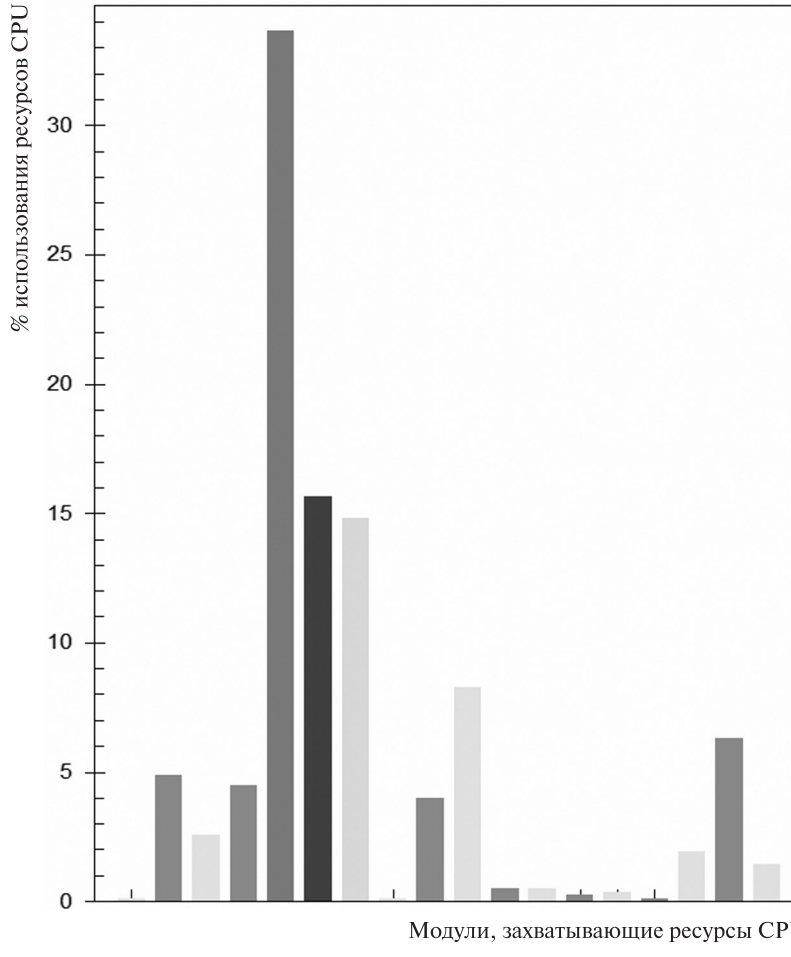

Рис. 4. Профиль работы $АРС_3$

2. Алгоритмическая оптимизация

Ускорение метода $АРС_1$ может осуществляться за счет функции сравнения. Согласно теории, приведенной в [1], из суммы центрированных растров удалением всех точек, значения в которых меньше некоторого порога. Из таким образом построенного эталона извлекаются:

- скелетный образ $SKEL(S, \alpha)$, содержащий черные точки там, где значения в соответствующих точках эталона выше α;

- расширенный образ $COVER(S, \beta)$, содержащий черные точки там, где расстояние до точек эталона меньше заданного β. $COVER(S, \beta) \supseteq \supseteq SKEL(S, \alpha)$.

Пороги L_E, α, β подбираются экспериментально.

Обозначим: $d(x, Y)$ — расстояние от точки x до множества Y, $\mu(Y)$ — мощность множества Y (количество элементов множества). Например, функция d может быть расстоянием Хаусдорфа от точки x до множества. Тогда расстояние $Dist(R, E)$ между распознаваемым растром R и эталоном E вычисляем как

$$Dist(R, E) = \mu(r_{ij} \mid r_{ij} > 0, d(r_{ij}, COVER(E)) = 1, i \in \{1, \ldots, N\}, j \in \{1, \ldots, M\}) + \\ + 2\mu(r_{ij} \mid r_{ij} > 0, d(r_{ij}, COVER(E)) > 1, i \in \{1, \ldots, N\}, j \in \{1, \ldots, M\}) + \\ + \mu(s_{ij} \mid s_{ij} > 0, d(s_{ij}, R) = 1, i \in \{1, \ldots, N\}, j \in \{1, \ldots, M\}). \quad (1)$$

Степень сходства рассматриваемого растра R и эталона E вычисляется следующим образом:

$$Conf(R, E) = \max(0, 255 - Dist(R, E)).$$

Рассмотрим несколько способов оптимизации быстродействия. Прежде всего, отметим прием, состоящий в вычислении расстояния d от всех внешних точек r_{ij} по отношению к множеству расширенного образа $COVER$: $d_{ij} = d(r_{ij}, COVER)$. Эти расстояния сохраняются в компонентах представления растра и многократно используются при вычислении расстояния (1). Этот способ, описанный в [5], позволяет производить вычисление расстояний Хаусдорофа только один раз при формировании набора эталонов. Собственно говоря, без заранее вычисленных расстояний d_{ij} алгоритм $АРС_1$ не работает, по косвенным данным, распознавание ускоряется не менее чем в 12 раз.

Другой способ состоит в использовании массива функций $F[256]$, каждая из которых вычисляет часть расстояния по формуле (1) для восьми

Таблица 1

Ускорение $АРС_1$

Компьютер / Стенд/ускорение	AMD Athlon 64 X2, 4,2 ГГц, DDR2 800	Intel PIV, 3 ГГц, DDR 400	Intel® Core™ 2 Quad, 2,4 ГГц, DDR2 800
TS_1 без модификации	220	180	270
TS_1 с модификацией	380	340	610
TS_2 без модификации	270	160	320
TS_2 с модификацией	440	290	730
Максимальное ускорение	1,73	1,89	2,28

точек r_{ij}, хранящихся в одном байте бинарного растра. Например, для $F[0]$ никаких вычислений не производится, а для $F[255]$ необходимо обработать все восемь точек. Отметим, что на практике используются две более сложные модификации расстояния (1), в каждой из которых одновременная обработка восьми точек производится не всегда.

В табл. 1 приведены результаты вычислительных экспериментов по оценке ускорения вторым из описанных способов.

Таким образом, относительное увеличение скорости для рассмотренного способа составляет 1,8–2,3 раза.

3. Использование инструкций CPU (центрального процессора)

Рассмотрим ускорение метода APC_2, использующее наборы потоковых инструкций SIMD процессоров Intel. Речь идет о расширениях SSE (Streaming SIMD Extensions), существующих в процессорах Intel начиная с 1999 года. В настоящее время процессоры AMD также поддерживают расширения SSE. В функции, реализующей вычисление скалярного произведения векторов с вещественными компонентами (float), для ускорения могут быть применены инструкции SSE.

Основная идея состоит в замене четырех операций произведения компонент $x_1 \cdot y_1 + x_2 \cdot y_2 + x_3 \cdot y_3 + x_4 \cdot y_4$ на одну инструкцию MULPS, обеспечивающую умножение вещественных операндов (x_1, x_2, x_3, x_4) и (y_1, y_2, y_3, y_4) в форме 128-битных пакетов. Для наивысшего быстродействия адреса векторов должны быть выровнены на границу 16 байт, такие данные считываются по четыре числа одной инструкцией MOVAPS.

Использование инструкций во встроенном ассемблере Microsoft Visual C++® 6.0 возможно после установки патча Visual C++ 6.0 Processor Pack.

В табл. 2 приведены результаты вычислительных экспериментов по оценке ускорения метода APC_2 с помощью инструкций SSE.

Относительное увеличение скорости для рассмотренного способа составляет 1,14–1,88 раза.

Оптимизация APC_3 производилась за счет использования инструкций SSE для реализации двух функций:

- скалярного произведения векторов с вещественными компонентами (float);
- сложения со сдвигом по формулам $x_i = x_i + \alpha \cdot y_i$ для целых компонент.

Первая функция была рассмотрена выше, вторая реализуется с помощью следующих инструкций MMX:

- PMADDWD (умножение целых чисел в 64-битном пакете);
- MOVQ (обмен 64-битными операндами).

Таблица 2

Ускорение APC_2 при использовании SSE

Компьютер / Стенд/ускорение	AMD Athlon 64 X2, 4,2 ГГц, DDR2 800	Intel PIV, 3 ГГц, DDR 400	Intel ® Core ™ 2 Quad, 2,4 ГГц, DDR2 800
TS_1 без SSE	2450	2290	4480
TS_1 с SSE	2790	4230	8340
TS_2 без SSE	2470	2310	4490
TS_2 с SSE	2780	4220	8430
Максимальное ускорение	1,14	1,85	1,88

Таблица 3

Ускорение APC_3 при использовании SSE и MMX

Компьютер / Стенд/ускорение	AMD Athlon 64 X2, 4,2 ГГц, DDR2 800	Intel PIV, 3 ГГц, DDR 400	Intel ® Core ™ 2 Quad, 2,4 ГГц, DDR2 800
TS_1 без SSE и MMX	10 570	8760	12 800
TS_1 с SSE и MMX	12 720	11 440	15 600
TS_2 без SSE и MMX	10 780	9190	13 300
TS_2 с SSE и MMX	13 000	11 990	16 200
Максимальное ускорение	1,2	1,3	1,22

В табл. 3 приведены результаты вычислительных экспериментов по оценке ускорения метода APC_3 с помощью инструкций SSE и MMX на различных компьютерах.

Относительное увеличение скорости для рассмотренного способа составляет 1,2–1,3 раза.

Отметим, что выигрыш в быстродействии достигается при обработке векторов с большой длиной.

4. Использование инструкций GPU (графического процессора)

В настоящее время возможности видеокарт, включающих графические процессоры, позволяют производить достаточно сложные вычисления над большими объемами данных.

Технология CUDA [6], предлагаемая компанией NVIDIA, состоит в использовании среды разработки, которая позволяет писать программное

Таблица 4

Ускорение APC_2 при использовании GPU

Компьютер / Кол-во эталонов	Intel® Core™ 2 Quad, 2,4 ГГц, видеокарта GeForce 8400 GS (частота пропускания 6,4 Гб/с)		Intel® Core™ 2 Quad, 2,4 ГГц, видеокарта GeForce GTX 285 (частота пропускания 159,0 Гб/с)	
	CPU	**GPU**	**CPU**	**GPU**
400	4533	5064	5844	8607
800	2394	4241	3111	8576
2000	914	2375	1293	7893
6400	541	1638	813	7970
4000	432	1320	633	7149
8000	218	738	308	6428
12 000	146	497	206	5313
16 000	109	380	154	4281
Максимальное ускорение	3,49		27,8	

обеспечение для решения сложных вычислительных задач с возможностью многоядерной вычислительной мощности графических процессоров. В частности, технология CUDA позволяет реализовать функцию скалярного произведения векторов $\mathbf{R}^N$, используемую в методе APC_2. Для компиляции требуется Microsoft Visual C++ 2008 и пакет NVIDIA CUDA Toolkit. Для работы ПО необходима видеокарта GeForce не ниже серии 8 или специализированные карты Tesla и Quadro, а также драйвер видеокарты, поддерживающий работу CUDA.

При компиляции функций, содержащих инструкции CUDA, код, предназначенный для CPU, компилируется стандартным образом, а код, предназначенный для GPU, на лету конвертируется в объектный код PTX (с помощью драйвера). В заключение происходит трансляция PTX в исполнимый модуль, который будет использовать как инструкции CPU, так и GPU. Следует отметить возможность как внутреннего параллелизма вычислений в блоке GPU, так и возможности асинхронного выполнения инструкций CPU параллельно с работой GPU.

Программирование для CUDA требуют разбиения приложения между несколькими ядрами без разделения данных, которые хранятся в общей видеопамяти.

В табл. 4 приведены результаты вычислительных экспериментов по оценке ускорения метода APC_2 с помощью возможностей GPU на различ-

ных компьютерах. Была использована тестовая последовательность TS_1, в экспериментах количество загруженных эталонов варьировалось в диапазоне 400–16 000.

Результаты, приведенные в табл. 4, следует интерпретировать следующим образом. Увеличение мощности множества эталонов, используемых в методе APC_2, может использоваться для улучшения иных характеристик качества распознавания, таких как точность, полнота коллекции или монотонность оценок. Однако для данного метода рост числа эталонов приводит к линейному замедлению на CPU, в то время как для GPU время замедляется нелинейно. Так для GPU с полосой пропускания 159,0 Гб/с увеличение числа эталонов в 25 раз приводит к скорости, равной скорости распознаванию без GPU с исходным числом эталонов.

Относительное увеличение скорости для рассмотренного способа составляет 1,12–27,8 раза.

5. Распараллеливание алгоритмов

В настоящее время для распараллеливания процессов и потоков OCR пригодны два типа архитектур:

- многопроцессорные и многоядерные компьютеры с общей памятью (SMP);
- кластерные системы с распределенной памятью (MPP).

Простой и недорогой способ ускорения состоит в запуске нескольких копий распознающих процессов (*recounits* в терминологии Ai2A [7]), каждый из которых решает задачу распознавания одного образа и занимает значительные ресурсы процессора. В таком способе обязательным является наличие диспетчера, обеспечивающего распределение работ между процессами и сохранение результатов распознавания. Ускорение достигается за счет квантования времени между работающими процессами, причем указанное квантование происходит благодаря возможностям операционной системы, начиная с Windows XP. Возможен и другой способ, в котором параллельно выполняются части задачи распознавания одного образа [8], при этом используются системы программирования MPI [9].

Экспериментально устанавливается, что описанный способ эффективен, если распараллеливаемые процессы достаточно длительны. Например, для перечисленных выше многоядерных компьютеров эффективно распараллеливание процессов, время работы которых превышает 1 секунду. Однако для рассмотренных APC такой способ также применим в ситуации, когда каждый из процессов recounits выполняет распознавание множества образов символов, например, всех выделенных на образе странице образов символов. Для проведения экспериментов будем использовать тестовое приложение, функционирующее следующим образом:

Таблица 5

Скорость распознавания при распараллеливании

Компьютер / Кол-во процессов	AMD Athlon 62 X2, 4,2 ГГц, DDR2 800	Intel PIV, 3 ГГц, DDR 400 HyperTreading	Intel ® Core ™ 2 Quad, 2,4 ГГц, DDR2 800
1	10 500,00	8800,00	12800,00
2	20 999,99	13058,06	25599,99
3	24 499,98	12650,00	38399,99
4	22 615,38	12650,00	51199,99
5	28 269,22	12650,00	46222,20
6	22 049,99	–	49919,98
7	–	–	50643,46
8	–	–	53247,98
9	–	–	53485,70
10	–	–	55466,64
11	–	–	52297,13
12	–	–	49051,41
Максимальное ускорение	2,69	1,48	4,33

- запуск приложением-диспетчером нескольких процессов recounits;
- при запуске каждый recounit извлекает из хранилища графических образов тестовую последовательность большой длины (TS_1);
- по сигналу приложения-диспетчера все процессы recounit начинают распознавание;
- распознавание заканчивается в момент завершения обработки последней тестовой последовательности.

Скоростью распознавания считается количество графических образов, распознанных всеми процессами за единицу времени.

В табл. 5 приведены результаты вычислительных экспериментов по оценке ускорения метода $АРС_3$ с помощью распараллеливания на различных компьютерах с одним и несколькими ядрами.

Относительное увеличение скорости для рассмотренного способа составляет 1,48–4,33 раза.

Выводы

Рассмотренные способы ускорения АРС являются результативными, т. е. дают выигрыш в скорости на различных типах компьютеров от 10

до 80 % для способов, перечисленных в разделах 2 и 3, и выигрыш от 269 до 2780 % для способов, перечисленных в разделах 4 и 5. При этом отметим, что трудоемкость разработки, относящейся к оптимизации, невелика, прежде всего из-за больших затрат времени на исследовательскую работу, предшествующую созданию алгоритма распознавания символа.

Рассмотренные способы ускорения АРС с успехом применяются в OCR Cuneiform и Cognitive Forms.

Автор выражает благодарность А. Менделенко, написавшему функции работы с GPU NVIDIA, использованные при получении результатов табл. 4, а также А. Джораеву за обсуждения технологии CUDA NVIDIA.

Литература

1. *Мисюрев А. В.* Использование искусственных нейронных сетей для распознавания рукопечатных символов // Труды ИСА РАН «Интеллектуальные технологии ввода и обработки информации». 1998. С. 122–127
2. *Котович Н. В.* Алгоритмы кластеризации образов символов // Труды ИСА РАН «Обработка изображений и анализ данных». М.: URSS, 2008. С. 241–251.
3. Intel® VTune™ Performance Analyzer 9.1 for Windows* — Overview. [Электронный ресурс]: www.intel.com/cd/software/products/asmo-na/eng/219898.htm
4. *Alimoğlu F., Alpaydin E.* Combining Multiple representations and Classifiers for Pen-based Digit Recognition // The 4th International Conference on Document Analysis and Recognition (ICDAR 97). Ulm, Germany, August 1997. P. 637–640.
5. *Арлазаров В. Л., Троянкер В. В., Котович Н. В.* Адаптивное распознавание символов // Труды ИСА РАН «Интеллектуальные технологии ввода и обработки информации». 1998. С. 39–56.
6. CUDA Zone — Подробно о CUDA. [Электронный ресурс]: www.nvidia.ru/object/cuda_what_is_ru.html
7. *Gorski N., Anisimov V., Augustin E., Baret O., Price D., Simon J.-C.* A2iA Check Reader: a family of bank check recognition systems // Proceedings of the Fifth International Conference on Document Analysis and Recognition (ICDAR 99). Bangalore, India, September 1999. P. 523–526.
8. *Талалаев А. А.* Особенности архитектуры параллельной программной системы распознавания графических образов на основе искусственных нейронных сетей // Нейрокомпьютеры: разработка и применение. 2008. № 9. С. 43–51.
9. MPI: The Message Passing Interface — русскоязычная страница. [Электронный ресурс]: http://parallel.ru/tech/tech_dev/mpi.html
10. *Арлазаров В. Л., Славин О. А.* Алгоритмы распознавания и технологии ввода текстов в ЭВМ // Информационные технологии и вычислительные системы. 1996. Т. 6. № 1. С. 48–54.
11. *Арлазаров В. Л., Постников В. В., Шоломов Д. Л.* Cognitive Forms — система массового ввода документов // Труды ИСА РАН «Управление информационными потоками». М.: URSS, 2002. С. 35–47.