УДК 004.852

В. Е. Уваров, В. А. Уварова, А. И. Фомин

ИСПОЛЬЗОВАНИЕ ГРАФИЧЕСКИХ ПРОЦЕССОРОВ ДЛЯ ОПТИМИЗАЦИИ ПРОЦЕССА КЛАССИФИКАЦИИ МНОГОМЕРНЫХ ПОСЛЕДОВАТЕЛЬНОСТЕЙ, ОПИСЫВАЕМЫХ СКРЫТЫМИ МАРКОВСКИМИ МОДЕЛЯМИ

В настоящее время широко распространены задачи машинного обучения. Они применяются во многих сферах: распознавание образов, таких как лица на фотографиях, дорожные знаки, объекты на видеоряде, классификация различных объектов, распознавание речи и т. п.

Один из статистических методов машинного обучения – *скрытые марковские* модели [1]. Машинное обучение состоит из двух этапов.

На первом этапе модели обучают с помощью специально подготовленных данных.

На втором этапе с помощью обученных моделей осуществляется анализ уже новых данных [2].

Обычно для обучения и анализа приходится обрабатывать большие объёмы данных, что делает последовательную реализацию метода на процессоре неприменимой из-за малой производительности.

Применение мультикомпьютеров для ускорения возможно, но данный метод достаточно затратный и неприменим для обычного пользователя. Хорошей альтернативой является использование графических ускорителей в сочетании с процессором (гетерогенное программирование). Такой подход предполагает подготовку данных и синхронизацию работ на процессоре, а выполнение связанных с ними расчётов - на графическом ускорителе.

Таким образом, можно достичь значительного ускорения работы метода практически на любой машине, оснащённой современной видеокартой и процессором.

Разрабатываемая в данной работе программа как раз и предполагает использование такого подхода.

Существуют различные реализации скрытых марковских моделей на видеокартах [4,5].

Однако эти реализации предполагают работу только с библиотекой CUDA[1], что делает их зависимыми от видеокарт фирмы Nvidia[2]. Кроме того, данные реализации рассчитаны на классификацию одномерных последовательностей.

Настоящий проект предполагает использование фреймворка OpenCL[3], который позволяет использовать видеокарты как от Nvidia, так и от ATI. Также разрабатываемая программа рассчитана на классификацию многомерных последовательностей, что обусловливает её новизну и расширяет область применения.

Ранее был разработан прототип программы, реализующий данный метод. Однако он показал неудовлетворительные результаты: вычисления на видеокарте заняли больше времени, чем на процессоре [6].

Задача совершенствования метода классификации многомерных числовых последовательностей, описываемых СММ, путём использования видеокарт для выполнения трудоёмких вычислений состоит в доработке и модификации алгоритма программы с целью ускорения вычислений как на этапе оценивания параметров СММ, так и на этапе классификации последовательностей.

Скрытые марковские модели и их применение для машинного обучения

Параметры СММ

Марковская модель – это система, которая в любой момент времени находится в одном из N различных состояний: $S_1, S_2, \dots, S_N$. Через равные, дискретные промежутки времени система переходит в новое состояние в соответствии с матрицей переходных вероятностей A.

В случае СММ последовательность состояний, в которые переходит модель, скрыта от наблюдателя. Ему доступны лишь сигналы (наблюдения), которые генерирует система при переходе в новое состояние.

Таким образом, СММ характеризуется следующим набором параметров:

N – количество состояний в модели. Обозначим отдельные состояния как $S = \{S_1, S_2, \dots, S_N\}$, а состояние, в котором находится модель в момент времени t как q_t;

M – число различных сигналов, которые способны генерировать состояния. Отдельные сигналы обозначаются: $V = \{v_1, v_2, \dots, v_M\}$;

$A = \{a_{ij}\}$ - матрица переходных вероятно-

[1] CUDA (англ. Compute Unified Device Architecture) — программно-аппаратная архитектура параллельных вычислений на видеоадаптерах

[2] NVIDIA - один из крупнейших разработчиков графических ускорителей и процессоров для них

[3] OpenCL - (англ. Open Computing Language) — фреймворк для написания компьютерных программ, связанных с параллельными вычислениями на различных графических и центральных процессорах

стей, где $a_{ij} = P[q_t = S_j | q_{t-1} = S_i], 1 \le i,j \le N$;

распределение вероятности появления символа k в состоянии j, обозначается $B = \{b_j(k)\}$, где $b_j(k) = P[v_k$ в момент $t | q_t = S_j], 1 \le j \le N$;

начальное распределение $\Pi = \{\pi_i\}$, где $\pi_i = P[q_1 = S_i], 1 \le i \le N$;

Введём краткую запись $\lambda = (A, B, \Pi)$ для обозначения параметров модели [1].

Метод машинного обучения, применительно к СММ, предполагает два этапа: обучения и классификации новых данных.

Этап обучения

Этап обучения сводится к решению одной из основных задач СММ, т. е. к подбору параметров модели $\lambda = (A, B, \Pi)$ таким образом, чтобы максимизировать сумму вероятностей появления последовательностей наблюдений $O^k = \{O_1^k, O_2^k, \dots, O_T^k\}$ для модели λ: $\sum_{k=1}^{K} P(O^k|\lambda)$.

Для решения этой задачи используется алгоритм Баума-Велша [1].

Введём следующее обозначение.

Прямая вероятность – вероятность того, что модель находится в состоянии S_i в момент времени *t*, при заданных наблюдениях $O_1, O_2, \dots, O_t$:

$$\alpha_t(i) = P(O_1, O_2, \dots, O_t, q_t = S_i|\lambda).$$

Чтобы избежать ошибок, связанных с ограниченной точностью представления малых вещественных чисел в компьютере, будем использовать процедуру масштабирования.

Введём коэффициенты масштабирования:

$$c_t = [\ \sum_{i=1}^{N} \alpha_t(i)\ ]^{-1} \qquad (1)$$

Теперь для вычисления вероятностей следует воспользоваться формулой:

$$\ln[P(O|\lambda)] = -\sum_{t=1}^{T} \ln c_t \qquad (2)$$

Здесь вычисляется логарифм от вероятности, так как сами значения вероятностей слишком малы для машинного представления.

С целью повышения качества обучения будем использовать для обучения несколько последовательностей наблюдений (K последовательностей).

Принцип работы алгоритма Баума-Велша заключается в проведении процедуры повторного оценивания, т. е. получения новых, улучшенных значений параметров на основе старых параметров. Данная процедура повторного оценивания ведёт к локальному максимуму [1]. Для поиска глобального максимума следует применять алгоритм для нескольких ($NumInit$ единиц) различных начальных приближений параметров. Повторное оценивание следует проводить несколько раз ($Iter$ итераций), пока изменение параметров не станет незначительным. После получения $NumInit$ наборов параметров из разных приближений, следует выбрать тот набор λ^*, для которого величина $\ln[P(O, \lambda^*)]$ является наибольшей среди других наборов. Логарифм вероятности $\ln[P(O, \lambda^*)]$ вычисляется по формуле (2).

Этап классификации

Имеется несколько конкурирующих моделей $\lambda_1, \lambda_2 \dots \lambda_{NumModels}$, среди которых нужно выбрать ту, для которой $P(O|\lambda)$ является максимальной, т. е. лучшую модель.

Так как логарифмы вероятностей можно получить, посчитав значения масштабирующих коэффициентов (c_t, $1 \le t \le T$), произведём их вычисления так же, как в предыдущем пункте, т. е. по формуле (1). Затем, по формуле (2) посчитаем логарифмы вероятностей для всех моделей и выберем ту модель, для которой это значение будет наибольшим.

Технология OpenCL для программирования вычислений на графических процессорах

OpenCL (от англ. Open Computing Language – открытый язык вычислений) – это фреймворк для написания компьютерных программ, связанных с параллельными вычислениями на различных графических и центральных процессорах [3]. Данный стандарт создаёт конкуренцию технологии CUDA от NVidia и, возможно, в ближайшем будущем потеснит последнюю благодаря своей универсальности и кроссплатформенности.

Основным структурным элементом для выполнения параллельных вычислений на OpenCL является рабочий элемент (WorkItem) [3]. Каждый рабочий элемент выполняется в отдельном потоке. Рабочие элементы объединяются в рабочие группы. Синхронизация рабочих элементов во время вычислений на GPU возможна только в пределах группы.

OpenCL использует концепцию SPMD – единая программа, множество данных. Таким образом, каждый рабочий элемент выполняет одну программу, называемую кернелом (kernel). Параллельность достигается за счёт того, что каждый рабочий элемент может определить свой глобальный, локальный номера, а также номер группы, в которой он состоит и соответствующим образом спланировать свои действия. Рабочие группы выполняются на вычислительных единицах (compute units), встроенных в видеокарту. Современные видеокарты имеют от 5 до 32 вычислительных единиц. В свою очередь, рабочие элементы выполняются на обрабатывающих элементах (processing elements), встроенных в вычислительные единицы. Видеокарты AMD имеют 64 обрабатывающих элемента. Одновременно на одной вычислительной единице может выполняться более чем одна рабочая группа. Точно так же обрабатывающий элемент может быть использован несколькими рабочими элементами в конкурентном режиме.

При запуске кернела разработчик указывает n-мерный диапазон (NDRange), в котором он передаёт информацию о количестве глобальных рабочих элементов G, о размере рабочей группы S для каждого измерения. При этом можно определить (что?) вплоть до 3-х измерений.

Применение метода параллельных вычислений на графических процессорах для оптимизации процессов обучения и классификации с помощью скрытых марковских моделей

Для параллелизации большинства алгоритмов применяется следующий способ. Пусть имеется K последовательностей длиной T, сгенерированных СММ, имеющую N состояний. Формируется трёхмерная вычислительная сетка, как показано на рис. 1.

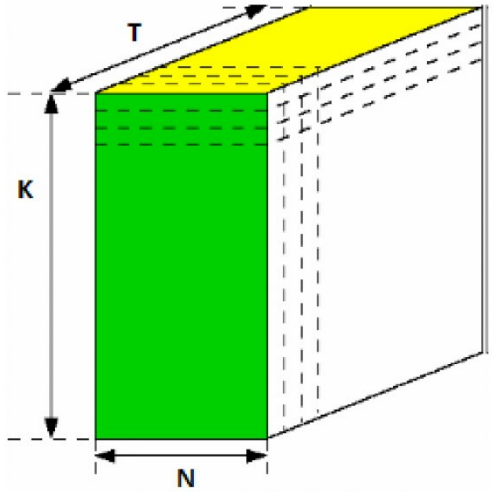

Рис. 1. Трёхмерная вычислительная сетка

В тех случаях, когда это возможно, вычисления разбиваются таким образом, что каждый поток обрабатывает один элемент из трёхмерной сетки. Если это невозможно, то каждый поток обрабатывает один из элементов двумерной сетки KxN. Группы потоков объединены в рабочие группы – подмножества разрезов сетки KxN.

Данные для всех вычислений собираются и организуются на процессоре, а уже затем передаются на видеокарту для трудоемких вычислений. Затем с видеокарты они передаются обратно на процессор для обработки результатов.

Исследования быстродействия проводились на компьютере под управлением операционной системы Windows 8.1 (x64). Характеристики компьютера представлены в табл. 1.

Таблица 1. Характеристики компьютера: процессора (cpu) и видеочипа (gpu)

Компонент	Спецификация
CPU	Intel Core i7-4790 Haswell (3600MHz)
GPU	Intel HD Graphics 4600 (1100MHz) (во время проведения исследования данный видеочип не использовался для вывода изображения на экран)

Для исследования были взяты следующие параметры СММ: $N = 3$, $M = 3$, $K = 100$, $T = 100$, $NumInit = 5$, $NumModels = 2$, $Z = 8$.

Результаты сравнение времени работы алгоритма на CPU и GPU представлены в табл. 2 и 3.

Таблица 2. Время работы алгоритма на этапе обучения

Компонент	Время выполнения, с
CPU	7,62
GPU	5,54

Таблица 3. Время работы алгоритма на этапе классификации

Компонент	Время выполнения, с
CPU	0,079
GPU	0,021

Как видно из таблиц, видеокарта показывает большее быстродействие, чем процессор. Таким образом, можно констатировать, что разработанная параллельная реализация значительно эффективнее последовательной: на этапе обучения производительность возросла в 1,37 раз, а на этапе классификации – в 3,76 раз. Ускорение на этапе классификации выше, так как вычисления на данном этапе легче поддаются параллельному исполнению. Так, на многих этапах можно параллельно исполнять $N * K * T = 3 * 100 * 100 = 30000$ потоков, в то время как на этапе обучения чаще всего можно исполнять параллельно лишь $N * M = 9$ или $N * N = 9$ или $N * Z = 24$ потоков.

На данном этапе исследования полученные результаты можно считать удовлетворительными, так как процесс классификации в плане быстродействия более важен, чем процесс обучения. Например, систему для распознавания дорожных знаков достаточно обучить один раз и время от времени обучать повторно, что не столь критично по времени. Однако в условиях реального дорожного движения такой системе необходимо максимальное быстродействие, чтобы распознать тот или иной знак и подать сообщение водителю или системе управления автомобилем.

Таким образом, усовершенствованный прототип программы применим для классификации многомерных числовых последовательностей с помощью параллельных вычислений на видеокартах. По сравнению с предыдущим результатом [6] наблюдается значительный прогресс, выраженный ускорением процесса вычислений, однако остается очевидным тот факт, что имеются большие ресурсы для доработки данного алгоритма и значительного ускорения вычислений на этапе обучения. Этого можно добиться путем видоизменения алгоритма, дальнейшей его адаптации к параллельному исполнению, а также с помощью использования двухмерной редукции и прочими приемами параллельного программирования. Дальнейшая оптимизация работы алгоритма позволит задействовать больший процент ресурсов видеокарты, что должно повлечь за собой увеличение скорости как при оценивании параметров СММ, так и при классификации последовательно-

стей.

СПИСОК ЛИТЕРАТУРЫ

1. *Rabiner L. R. A* Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition// Proceedings of the IEEE, 1989, vol. 77, no. 2, pp 257-285.

2. Ryszard Stanislaw Michalski Machine Learning: An Artificial Intelligence Approach, vol. 1 / Ryszard Stanislaw Michalski, Jaime Guillermo Carbonell, Tom Michael Mitchell. – San Francisco, CA, USA : M. Kaufmann, 1983. – 572 p.

3. *Aaftab Munshi* OpenCL Programming Guide / Aaftab Munshi, Benedict R. Gaster, Timothy G. Mattson. – Boston, MA, USA : Addison-Wesley Professional, 2011. – 648 p.

4. cuHMM: a CUDA Implementation of Hidden Markov Model: Training and Classication [Электронный ресурс] / Chuan Liu, 2009 — Режим доступа: https://liuchuan.org/pub/cuHMM.pdf.

5. Final Project: Parallel Viterbi on a GPU [Электронный ресурс] : University Of Washington — Seong Jae Lee, Miro Enev — Режим доступа: http://homes.cs.washington.edu/~miro/docs/HMM_on_GPU.pdf.

6. Использование графических процессоров для оптимизации работы скрытых марковских моделей // Гультяева Т. А., Саутин А. С., Уваров В. Е. // Труды XII Международной научно-технической конференции «Актуальные проблемы электронного приборостроения» (АПЭП-2014).

Авторы статьи:

Уваров Вадим Евгеньевич,
студент 5 курса ФПМИ Новосибирского НГТУ. E-mail: uvarov.vadim42@gmail.com.

Уварова Варвара Александровна,
канд. техн. наук, ведущий науч. сотр. ОАО «НЦ ВостНИИ». E-mail: uvarova.v.a@mail.ru

Фомин Анатолий Иосифович,
докт. техн. наук, проф. каф. аэрологии, охраны труда и природы КузГТУ. E-mail: aotp2012@yandex.ru

Поступило в редакцию 7.01.2015