УДК 004.891.2

В.В. Симонов

Оценка эффективности параллельных алгоритмов для моделирования многослойного персептрона

Дано краткое описание параллельного алгоритма моделирования многослойного персептрона и параллельного алгоритма обучения iRprop. Алгоритмы используют параллельные вычисления с разделяемой памятью и параллельность на уровне инструкций процессора. Приведено сравнение быстродействия последовательного и параллельного алгоритмов, а также динамика изменения их быстродействия за последние 6 лет.
Ключевые слова: многослойный персептрон, обучение многослойного персептрона, параллельные вычисления с разделяемой памятью, SIMD инструкции.

Введение

Алгоритмы моделирования и обучения нейронных сетей имеют высокую вычислительную сложность. Во многих приложениях ИНС, вычислительная сложность пропорциональна квадрату количества настраиваемых связей ИНС. Обучение больших ИНС может занимать несколько дней. Ускорение процесса их обучения поможет исследователям быстрее находить оптимальные топологии и добиваться более точных результатов прогнозирования и распознавания. Также быстродействие моделей ИНС очень важно для систем принятия решений в реальном времени, например систем, использующих машинное зрение.

Объем задач, решаемых с помощью ИНС, непрерывно растет. Увеличивается сложность моделей, ужесточаются бизнес-требования, предъявляемые к моделям. Все это приводит к увеличению вычислительной сложности.

Ранее созданные программы не могут использовать даже половину вычислительной мощности современных процессоров, так как не используют параллельные вычисления, а потенциал увеличения производительности отдельно взятого процессорного ядра практически исчерпан. Быстродействие ранее созданных алгоритмов на современных компьютерах будет повышаться медленнее роста сложности задач, что приведет к неэффективности систем, использующих эти алгоритмы. Поэтому сейчас очень важно создавать новые, параллельные реализации моделей ИНС, чтобы они могли использовать возрастающую вычислительную мощь современных ЭВМ и легко масштабировались для решения больших и сложных практических задач.

Проблема разработки параллельных алгоритмов встала относительно недавно, но ее актуальность непрерывно растет. На данный момент имеется совсем немного теоретических разработок и методов организации параллельных вычислений. Поэтому необходимо искать новые способы распараллеливания алгоритмов, разрабатывать методы выделения шагов, способных выполняться параллельно.

Использование параллельных вычислений позволит повысить энергоэффективность процесса моделирования и обучения, измеренную в миллионах операций с плавающей точкой (mflops) на затраченный ватт электроэнергии.

1. Постановка задачи

Для популярных в настоящее время математических моделей ИНС и моделей их обучения необходимо разработать параллельные алгоритмы, позволяющие в полной мере использовать вычислительную мощь современных многоядерных процессоров. Параллельные алгоритмы должны хорошо масштабироваться. Это означает, что со временем количество вычислительных потоков, способных выполняться одновременно, будет расти, и алгоритмы должны быть построены таким образом, чтобы обеспечить работой как можно большее количество параллельных потоков.

2. Выбор метода исследования

Быстродействие реализации сравнивается с быстродействием алгоритмов, реализованных в библиотеке Fast Artificial Neural Network Library (FANN) версии 1.2.0 [1]. Реализация ИНС из библиотеки FANN обладает отличным быстродействием и использует оптимизации, такие как оптимизация доступа к данным, разворачивание циклов (loop unrolling).

Быстродействие измерялось в секундах, затраченных реализациями алгоритмов на выполнение одних и тех же вычислений. Высчитывалось среднее время из десяти проходов для каждой реализации.

В ходе тестирования было установлено, что реализации многослойного персептрона и iRprop [2] не требовательны к пропускной способности памяти. Их быстродействие ограничено только быстродействием процессора. Было проведено тестирование реализации на различных процессорах Intel от 2004 до 2009 года выпуска, от Pentium 4 до Core i7, что позволит сделать некоторые выводы относительно масштабируемости реализаций и спрогнозировать будущие изменения быстродействия.

3. Описание параллельных алгоритмов

3.1. *Параллельная реализация многослойного персептрона*

Многослойный персептрон реализован в виде класса Perceptron. Этот класс отвечает за прямое распространение сигнала в нейронной сети и информацию о топологии сети.

Perceptron содержит данные двух видов: данные для чтения (Structure) и данные для чтения и записи (Data). Во время функционирования нейронной сети ее структура не меняется, т.е. описание структуры сети используется только для чтения, в то время как Data полностью перезаписывается при прямом распространении сигнала в сети. Чтобы можно было выполнять прямое распространение сигнала параллельно, необходимо создать копию экземпляра Perceptron для каждого потока, но так как структура персептрона при этом не меняется, ее можно не копировать. Для этих целей был создан метод Perceptron::ParallelCopy. Таким образом, объект Structure может использоваться несколькими объектами Perceptron. Для разделяемых объектов Structure реализован подсчет ссылок, это позволяет удалить класс Structure, если на него не ссылается ни один из Perceptron.

При прямом распространении сигнала выходной сигнал для каждого нейрона слоя рассчитывается параллельно. Это реализовано с помощью стандарта OpenMP [3].

Число синапсов каждого нейрона сети сделано кратным четырем. Например, если число синапсов нейрона – 13, оно будет увеличено до 16. Однако пользователь попрежнему может работать только с 13 синапсами, веса добавленных 3 синапсов всегда равны нулю. Добавление дополнительных синапсов делается для того, чтобы можно было применить векторные инструкции для SSE (Streaming SIMD Extension) [4, 5] для вычисления взвешенной суммы входов нейронов. Инструкции SSE позволяют выполнить одну и ту же арифметическую операцию над 4 вещественными числами одинарной точности одной командой. Использование SSE позволило увеличить быстродействие сети в 2,5 раза.

3.2. *Параллельная реализация алгоритма iRprop*

При обучении ИНС с помощью алгоритма iRprop ей сначала предъявляются все примеры обучающей выборки, для каждого примера вычисляется ошибка аппроксимации (разность между выходом нейронной сети и полученным значением). Происходит обратное распространение ошибки, т.е. распространение сигнала от выхода нейронной сети ко входу. Вычисляется оценка частной производной $\frac{\partial E}{\partial w_{ij}}$, где $E(W)$ – функция ошибки нейронной сети от всех весов сети W; w_{ij} – вес нейронной сети, связывающий i нейрон с j, $w_{ij} \in W$.

Эти шаги можно выполнить параллельно. Для этого каждый вычислительный поток должен иметь параллельную копию многослойного персептрона (можно использовать Perceptron::CloneParallel), копию массива отклонений (ошибок) каждого нейрона в сети и копию массива оценок частной производной функции ошибки по весу для каждого настраиваемого веса в сети.

4. Постановка эксперимента

В результате выполнения данной работы была написана библиотека классов, содержащая параллельную реализацию многослойного персептрона и параллельную реализацию алгоритма обучения iRprop.

Было произведено тестирование библиотеки. Для этого была взята большая нейронная сеть с 200 входами, 100 нейронами в скрытом слое, 1 нейроном в выходном слое. Сеть представляла собой многослойный персептрон, в котором каждый нейрон последующего слоя был связан со всеми нейронами предыдущего слоя. Для сравнения быстродействия использовалась библиотека FANN [1]. Для компиляции параллельной и последовательной (FANN) реализаций использовался компилятор Microsoft C++ Compiler 10.

Веса нейронной сети инициализировались каждый раз одними и теми же значениями, одинаковыми для FANN сети и для описываемой реализации. Далее на каждой тестовой машине проводились 100 эпох обучения нейронной сети и измерялось среднее время для 10 попыток обучения. Это позволило сразу задействовать алгоритмы прямого распространения сигнала в нейронной сети и алгоритм iRprop, сравнивать разные реализации алго-

ритмов в одинаковых условиях и увеличить точность оценок за счет повторения экспериментов.

Были проведены тесты параллельной реализации с различным количеством вычислительных потоков. На каждой системе были проведены тесты с использованием одного вычислительного потока (то же самое, что использовать последовательные вычисления), двух, и т.д. до n, где n – максимальное количество параллельных вычислительных потоков в системе. Например, для процессора Intel Core i7 920 с 4 физическими ядрами, каждое из которых содержит два логических, этот показатель будет равен $2 \cdot 4 = 8$. Была вычислена оценка эффективности параллельных вычислений. Эффективность распараллеливания показывает, какой процент времени вычислительные потоки тратят на эффективную работу, на вычисления. Значение 60% означает, что вычислительные потоки 60% времени выполняют полезную работу и 40% времени простаивают или тратят на синхронизацию. Эффективность вычислялась по формуле

$$\delta = \frac{T_{\text{пос}}}{n \cdot T_{\text{пар}}} \cdot 100\%, \qquad (1)$$

где $T_{\text{пос}}$ – время, затраченное на последовательное вычисление; $T_{\text{пар}}$ – время, затраченное на параллельное вычисление; n – количество вычислительных потоков.

5. Результаты эксперимента

Результаты измерения времени обучения многослойного персептрона для описываемой реализации и FANN приведены в табл. 1.

Таблица 1

Результаты измерения времени, затрачиваемого различными реализациями алгоритма

N п/п	Процессор	Дата начала выпуска	Время FANN, с	Описываемая реализация последовательно, время, с	Описываемая реализация параллельно, время, с	Эффективность распараллеливания, %
1	Pentium 4 HT@3.0	01.02.2004	24,26	13,07	11,35	57,6
2	Pentium D 805 @2.66	01.03.2006	24,42	13,17	7,52	87,5
3	Core 2 Duo E6400 @2.14	27.07.2006	11,82	6,28	3,58	87,8
4	Xeon E5345 (x2) @2.33	01.10.2006	11,85	6,50	1,89	81,4
5	Core 2 Quad Q6600 @2.4	07.01.2007	12,74	6,34	1,57	100,8
6	Core 2 Duo E7200 @2.53	20.04.2008	10,14	5,33	3,06	87,1
7	Core 2 Duo E8400 @3.0	20.01.2008	8,58	4,46	2,48	90,1
8	Core 2 Quad Q8300 @2.53	30.11.2008	10,16	5,34	1,59	84,1
9	Core i7 920 (w/o HT) @2.66	01.11.2008	9,16	4,78	1,31	90,9
10	Core i7 920 (HT) @2.66	01.11.2008	9,16	4,77	1,06	56,3
11	Core i7 860 @2.8	01.09.2009	8,75	4,56	1,12	51,0
12	Core i5 750 @2.66	08.09.2009	10,59	4,80	1,31	93,6

В колонке «Процессор» приведено наименование процессоров Intel и тактовая частота в гигагерцах, на которой работал процессор во время тестирования. Тактовая частота следует после символа «@».

В табл. 1 видно, что в последовательном режиме описываемая реализация работает примерно в 2 раза быстрее, чем реализация FANN. Достигается это за счет использования SSE инструкций процессора. Эти инструкции позволяют выполнять 4 одинаковых операции за одну инструкцию или за два такта процессора [5].

Для каждого процессора по формуле (1) была вычислена эффективность распараллеливания. В среднем эффективность распараллеливания составила около 90% при использовании нескольких физических ядер и 60% при использовании технологии Intel Hyper-Threading. Intel Hyper-Threading (HT) – технология, позволяющая организовать два вычислительных потока на одном процессорном ядре путем удвоения количества регистров и дублирования очереди команд. Эта технология позволяет оптимально загружать устройства процессора работой. В данном тестировании процессоры Pentium 4 HT, Core i7 920 и Core i7 860 используют HT. Стоит отметить, что для Core i7 920 приведены результаты

как с включенной HT (строка №10), так и без нее (строка №9). Эффективность распараллеливания без HT выше, чем с HT. Время вычислений с включенной HT меньше, чем без нее. HT позволяет получить более высокое быстродействие при том же самом количестве физических ядер, но снижает эффективность распараллеливания программ.

Эффективность распараллеливания могла быть и выше, если бы вычислительным потокам давались более крупные задания, что привело бы к сокращению накладных расходов на запуск и остановку потоков. Но в описываемой реализации этого намеренно не сделано. Кроме создания эффективной параллельной реализации, также стояла задача выявить как можно большее количество участков кода, способных выполняться параллельно с тем, чтобы потом можно было использовать эти наработки при переходе на другие стандарты параллельного программирования, такие как OpenCL.

Тестирование на процессоре Core 2 Quad Q6600 показало эффективность распараллеливания более 100%. Это можно объяснить только аномально низким быстродействием программы при последовательном режиме. Возможно, это было вызвано активностью фоновых программ или частыми промахами процессора в кэш память [5].

Также в табл. 1 приведены даты выпуска процессоров, что позволяет оценить скорости роста быстродействия последовательных и параллельных вычислений за 2004–2009 гг., применительно к задаче обучения искусственных нейронных сетей. На рис. 1 показана зависимость времени обучения нейронной сети от даты выпуска процессора для параллельных и последовательных вычислений.

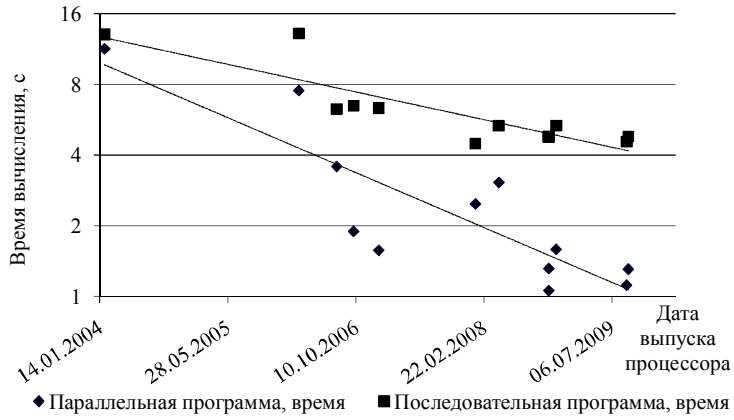

Рис. 1. Зависимость времени вычисления от даты выхода процессора

График построен с использованием логарифмической оси. Согласно закону Мура, быстродействие вычислительной техники растет экспоненциально, соответственно уменьшается время, необходимое для выполнения одной и той же вычислительной задачи. Тогда прирост производительности должен дать на графике с логарифмической шкалой прямую линию. По методу наименьших квадратов построим линии тренда для последовательных и параллельных вычислений. Из рис. 1 видно, что скорость параллельных вычислений со временем возрастает в 2 раза быстрее, чем скорость последовательных.

В табл. 2 приведена зависимость времени работы описываемого параллельного алгоритма от количества параллельных вычислительных потоков. Номера строк соответствуют номерам строк из табл. 1. Количество вычислительных потоков взято меньше или равным количеству параллельных потоков в системе. Поэтому для системы на Core 2 Duo – это только 2 потока, а для Core i7 – 8.

На рис. 2 показана зависимость времени обучения нейронной сети от количества вычислительных потоков.

Для процессоров, не использующих HT, график представляет собой гиперболу (с поправкой на эффективность распараллеливания, см. табл. 1).

Иначе дела обстоят с процессором Core i7 920. Для первых вычислительных потоков график представляет собой гиперболу, но затем уменьшение времени замедляется. Связано это с тем, что на 4 физических ядрах может параллельно выполняться только 4 потока, при увеличении количества потоков на одном ядре процессора начинает выполняться по 2 логических потока (благодаря HT). Производительность продолжает расти, но эффективность распараллеливания падает.

Таблица 2

Зависимость времени вычисления от количества параллельных вычислительных потоков

№ п/п	Зависимость времени обучения от кол-ва потоков							
	1	2	3	4	5	6	7	8
1	13,07	11,35						
2	13,17	7,52						
3	6,28	3,58						
4	6,50	3,54	2,47	1,89	1,60	1,43	1,23	1,00
5	6,34	3,15	2,07	1,57				
6	5,33	3,06						
7	4,46	2,48						
8	5,34	2,86	2,02	1,59				
9	4,78	2,56	1,75	1,31				
10	4,77	2,55	1,75	1,43	1,41	1,31	1,18	1,06
11	4,56	2,47	1,76	1,52	1,38	1,26	1,24	1,12
12	4,80	2,56	1,75	1,31				

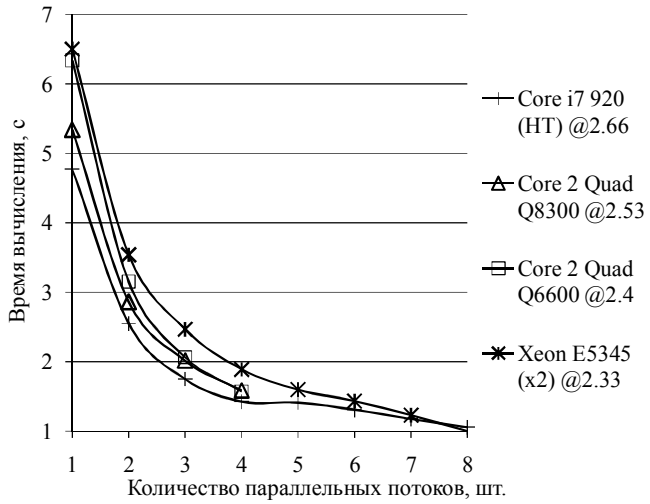

Рис. 2. Зависимость времени обучения нейронной сети от количества параллельных потоков

Пара процессоров Xeon E5345 (2 процессора по 4 ядра) позволяет нам взглянуть на будущее параллельных настольных вычислительных систем.

Заключение

Разработана, создана и протестирована эффективная параллельная реализация многослойного персептрона и алгоритма обучения iRprop. Алгоритмы оформлены в виде библиотеки классов на C++, что обеспечивает высокое быстродействие и позволяет повторно использовать эти классы в приложениях.

В ходе проведенных экспериментов установлено, что использование параллельных вычислений позволяет получить прирост скорости вычисления от 60 до 600%, по сравнению с последовательными вычислениями.

Литература

1. Сайт проекта Fast Artificial Neural Network Library (FANN) [Электронный ресурс]. – Режим доступа: http://leenissen.dk/fann/, свободный (дата обращения 21.04.2010).

2. Igel C. Improving the Rprop Learning Algorithm / C. Igel, M. Husken // Proceedings of the Second International Symposium on Neural Computation. – 2000. – С. 115-121.

3. Открытая спецификация OpenMP API для параллельного программирования [Электронный ресурс]. – Режим доступа: http://openmp.org/wp/, свободный (дата обращения 21.04.2010).

4. Свободная Интернет энциклопедия «Википедия». Статья «SSE» [Электронный ресурс]. – Режим доступа: http://ru.wikipedia.org/wiki/SSE, свободный (дата обращения 21.04.2010).

5. Intel® 64 and IA-32 Architectures Optimization Reference Manual November 2009 [Электронный ресурс]. – Режим доступа: http://www.intel.com/assets/pdf/manual/248966.pdf, свободный (дата обращения 21.04.2010).

6. Открытый стандарт параллельного программирования для гетерогенных сред OpenCL [Электронный ресурс]. – Режим доступа: http://www.khronos.org/opencl/, свободный (дата обращения 21.04.2010).

Симонов Василий Владимирович
Аспирант каф. автоматизированных систем управления ТУСУРа
Тел.: +7-960-971-77-53
Эл. почта: simonov.vasiliy@gmail.com

Simonov V.V.

Efficiency estimation of parallel algorithms for multilayer perceptron modeling

There is a brief description for parallel algorithm for modeling of multilayer perceptron and iRprop learning algorithm, introduced in this article. The algorithms has been developed using parallel calculations with shared memory model and parallel instructions for CPU. The performance comparison for serial and parallel programs and the analysis of their dynamics over the past 6 years have been done.
Keywords: multilayer perceptron, multilayer perceptron training, parallel calculations with shared memory, SIMD instructions.
