представлен на рисунке 1:

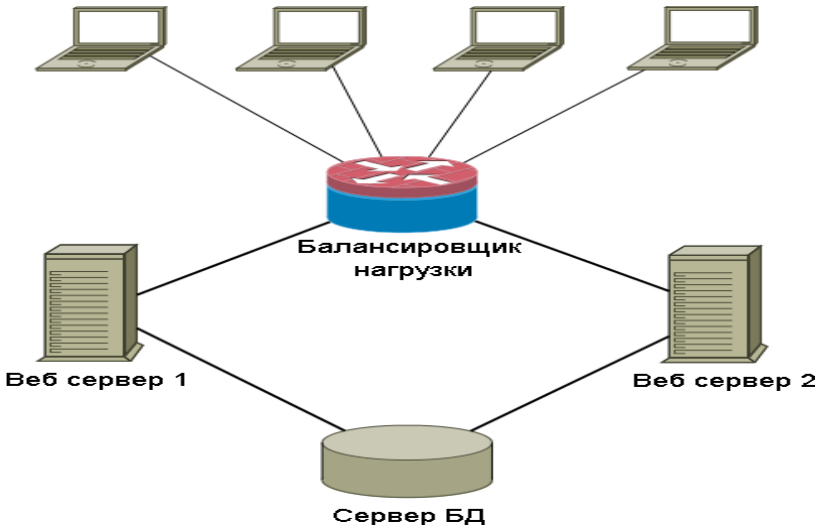

Рис. 1 – использование балансировщика нагрузки.

При такой топологии вводится понятие Прокси, который позволяет распределять нагрузку между серверами. Прокси – сервис, который принимает все входящие запросы и распределяет их между серверами в ферме в соответствии с их загруженностью. Как правило, прокси использует очереди для хранения входящих запросов.

Также использование балансировщиков нагрузки позволяет справиться с проблемой отказа одного из серверов. В таком случае вся нагрузки ложится на остальные сервера, но приложение продолжает свою работу, когда как в случае с единственным сервером, приложение бы прекратило свое корректное функционирование.

Также к положительным сторонам использования балансировщиков агрузки можно отнести простоту горизонтального масштабирования, это значит, что для сокращения нагрузки на существующие сервера достаточно добавить новый сервер, который также будет обрабатывать входящие запросы.

Данный подход является актуальным в наши дни, и большинство крупных систем используют балансирование нагрузки для поддержания работоспособности.

ЭМПИРИЧЕСКАЯ МОДОВАЯ ДЕКОМПОЗИЦИЯ С ИСПОЛЬЗОВАНИЕМ ГРАФИЧЕСКИХ ПРОЦЕССОРОВ NVIDIA, ПОДДЕРЖИВАЮЩИХ ПРОГРАММНО-АППАРАТНУЮ АРХИТЕКТУРУ CUDA

Белорусский государственный университет информатики и радиоэлектроники
г. Минск, Республика Беларусь

Мелещеня Д.В.

Бранцевич П.Ю. – к.т.н., доцент

Эмпирическая модовая декомпозиция является ключевой частью преобразования Гильберта-Хуанга. В силу того, что этапы преобразования включают построение огибающих с помощью интерполяционных полиномов, вся процедура представляет достаточно сложную вычислительную задачу. Для сокращения времени декомпозиции предлагается использовать графические процессоры, реализующих CUDA-совместимую архитектуру, а также, предлагаются подходы по работе с памятью, методы глобальной синхронизации и полной редукции на графическом процессоре, позволяющие ускорить декомпозицию сигнала на модовые функции.

Эмпирическая модовая декомпозиция (EMD – Empirical Mode Decomposition) – итеративная процедура, ставящая в соответствие исходному сигналу набор эмпирических мод (IMF – Intrinsic Mode Functions). Модовая декомпозиция, или просеивание, сводится к последовательности следующих этапов. По локальным экстремумам строятся верхняя и нижняя огибающие. После этого, вычисляется разность между средним значением огибающих и исходным сигналом. Далее, если остаток удовлетворяет критерию остановки, он считается очередной модовой функцией, в противном случае разность принимают за исходный сигнал и алгоритм повторяется. После нахождения IMF, ее вычитают из исходного сигнала и, если разность не является монотонной функцией либо меньше некоторого порогового значения, то алгоритм просеивания

повторяется.

Ключевой этап декомпозиции – построение огибающих на основе локальных экстремумов. Для интерполяции на участках между экстремумами используются кубические сплайны. Для того чтобы найти одну модовую функцию, в среднем, необходимо 6-8 операций просеивания, или 12-16 построений огибающих. Таким образом, процесс просеивания сам по себе может выполняться достаточно долго. При этом, в зависимости от исходного сигнала, для нахождения всех внутренних модовых функций необходимо выполнить несколько итераций просеивания. Экспериментальные исследования показали, что в случае разложения реальных сигналов количество получаемых модовых функций растет с увеличением длины исходного сигнала. Подытоживая данные факты можно сделать вывод, что в случае длинных сигнальных реализация, эмпирическая модовая декомпозиция требует много времени и вычислительных ресурсов, количество в лучшем случае зависит линейно от количества точек исходного сигнала.

Для того чтобы сократить время преобразования сигнала предлагается использовать графический процессор (GPU – Graphics Processing Unit). OpenCL и CUDA являются наиболее популярными фрэймворками для вычислений общего назначения на GPU (GPGPU – General-Purpose Computing For GPU), поскольку предоставляют возможность писать программы для графического процессора основываясь на стандарте языка Си и поддерживаются крупными производителями оборудования. В общем оба фреймворка реализуют схожую модель памяти и вычислений, однако в силу того, что адаптеры NVIDIA для GPGPU доступны как облачные сервисы, использование CUDA является более удобным с точки зрения разработки и дальнейшего использования.

В реализации преобразования Гильберта-Хуанга первый этап – нахождение локальных экстремумов – выполняется на центральном процессоре (CPU – CentralProcessingUnit). Далее полученный массив точек копируется в глобальную память GPU. При этом выделяется только один участок памяти для хранения минимумов и максимумов, а также сервисных структур данных, идентифицирующих принадлежность точек исходной кривой тому или иному интервалу интерполяции. В дальнейшем, для доступа к массивам используется указатели – каждый с соответствующим смещением. При частом выделении памяти, которое характерно для HHT с большим количеством итераций просеивания, такой подход позволяет сократить суммарное время расчета на 20 - 25%, поскольку каждая операция выделения памяти ведет к дополнительным накладным расходам.

Первым этапом после того, как в память графического устройства скопированы все данные, является экстраполяция на интервалах между экстремумами. Для этого используется сплайны Катмула-Рома. Суть метода заключается в том, что для каждого интервала исходной функции находится свой полином, описывающий кривую на данном участке. При этом значение в некоторой точке произвольного интервала определяется по следующей формуле:

$$p(x) = h_{00}(t)p_k + h_{01}(t)p(x_{k+1} - x_k)m_k + h_{01}(t)p_{k+1} + h_{11}(t)(x_{k+1} - x_k)m_{k+1},$$

где

$$\mathrm{t} = (\mathrm{x} - x_k)/(x_{k+1} - x_k),$$

$$m_n = (p_{n+1} - p_{n-1})/(t_{n+1} - t_{n-1}),$$

$h_{00}, h_{01}, h_{01}, h_{11}$ – базисные функции Эрмита.

За первый проход на основе локальных экстремумов вычисляются коэффициенты интерполирующих полиномов. Полученные коэффициенты сохраняются в глобальную память устройства. Вместо того, чтобы хранить коэффициенты в виде массива структур, для коэффициентов при одинаковой степени неизвестного всех интервалов выделяется свой массив. Таким образом, обеспечивается выравнивание данных в памяти. А поскольку особенностью CUDA-совместимой архитектуры является объединение доступа к локальной памяти, это позволяет сократить задержки при чтении и наиболее эффективно использовать кэш первого уровня L1.

За второй проход интерполяции на основе полученных на предыдущем этапе коэффициентов вычисляется значения функции в точках между интервалами. При нахождении значения функции в заданной точке используется схема Горнера. В соответствии с этим правилом многочлен n-й степени:

$$P_n(x) = a_0x^n + a_1x^{n-1} + \cdots + a_{n-1}x + a_n$$

представляется в виде:

$$P_n(x) = \big(\ldots\big((a_0x + a_1)x + a_2\big)x + \cdots + a_{n-1}\big)x + a_n.$$

Это позволяет сократить количество операций умножения с 7 до 3, а с учетом того, что необходимо строить верхнюю и нижнюю огибающие – с 14 до 6. После этого находится среднее значение огибающих, которое вычитается из исходного сигнала.

Последним этапом в итерации просеивания является определение, найдена ли очередная модовая функция. В качестве критерия останова используется среднеквадратичная разность между исходным сигналом и возможной модой. Если она не превышает заданного значения, очередная внутренняя модовая функция считается найденной. Для нахождения среднеквадратичной разницы используется достаточно часто применяемый в параллельной обработке данных подход – редукция. Однако в классическом варианте

заключительный этап производится на центральном процессоре, что может быть не совсем эффективным в случае с длинными сигнальными реализациями. Поэтому в предложенном алгоритме модовой декомпозиции все итерации редукции производятся на графическом адаптере, а для синхронизации всех потоков используется перезапуск ядра, так как CUDA поддерживает только синхронизацию на уровне группы потоков, но не содержит примитивов синхронизации всех потоков. Еще одним архитектурным решением, позволяющим сократить время вычисления, является использование разделяемой памяти внутри блоков. Доступ к разделяемой памяти осуществляется гораздо быстрее чем к глобальной, поэтому каждый блок работает с копией данных в разделяемой памяти.

На рисунке 1 представлено сравнение трех реализаций преобразования Гильберта-Хуанга: для CPU, GPU и вариант для GPU с оптимизацией чтения из глобальной памяти, редукции и использованием разделяемой памяти.

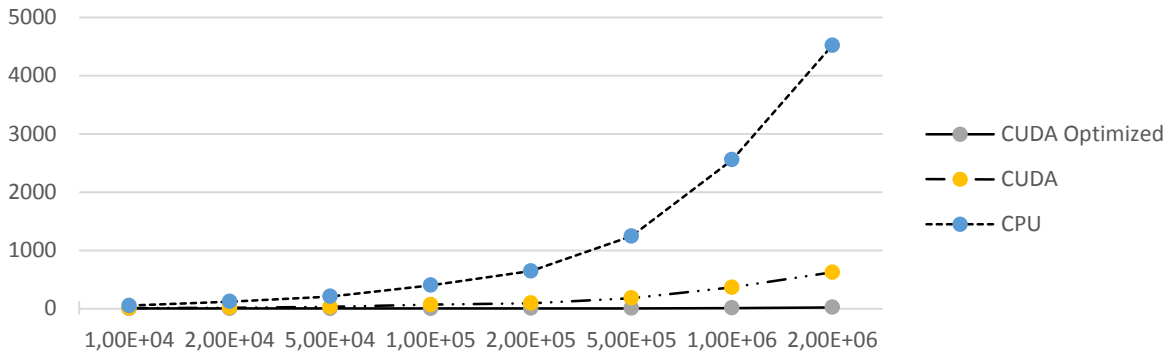

Рис. 1 – Сравнение зависимости времени декомпозиции сигнала от его длины при запуске на графическом и центральном процессорах (по оси ординат время в миллисекундах, по оси абсцисс количество точек)

Экспериментальные исследования проводились с использованием процессора IntelXeonE5450 и графического адаптера NvidiaTeslaK80.

Список использованных источников:

1. Роджерс, Д. Математические основы машинной графики / Д. Роджерс, Дж. Адамс. – М.: Мир. – 2001. – 604 с.
2. Huang, N.E. The empirical mode decomposition and the Hilbert spectrum for nonlinear and non-stationary time series analysis / N.E. Huang, Z. Shen, S.R. Long, M.C. Wu, H.H. Shih and other. – 1998. – T.454. – c.903 – 995.
3. Huang E., An Introduction to Hilbert-Huang Transform: A Plea for Adaptive Data Analysis. Research Center for Adaptive Data Analysis.
4. Sanders, J. CUDA by Example: An Introduction to General-Purpose GPU Programming / J. Sanders, E. Kandrot. – Paperback, – 2010 – 279 p.
5. Документация CUDA Toolkit // NVIDIA [Электронный ресурс]. – 2018. – Режим доступа: http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html Дата доступа: 18.01.05

МОДЕЛЬ ПОСТРОЕНИЯ ОБЩИХ ТРЕБОВАНИЙ ПО ОБРАБОТКЕ БАНКОВ ДАННЫХ В ЗАДАЧАХ СОПРОВОЖДЕНИЯ СИСТЕМ

Белорусский государственный университет информатики и радиоэлектроники
г. Минск, Республика Беларусь

Моженкова Е.В.

Парамонов А.И. – к.т.н., доцент

Изменения программного обеспечения необходимы для адаптации к повышенным функциональным требованиям и различным системным конфигурациям, вызванным этими изменениями. Корпоративная информационная система (КИС) становятся все более сложной по мере роста и развития, поэтому поддержание такой системы является основной задачей для отрасли. Одной из наиболее актуальных проблем в компьютерной индустрии является необходимость поддерживать и улучшать программный продукт с более высокими темпами и с меньшими затратами.

На основе результатов, которые получены в исследованиях [1-2], можно сформулировать одну из главных целей обработки банков данных в задачах сопровождения КИС – определение области расширения автоматизации в задачах сбора и анализа данных клиента по выявленной проблеме с целью дальнейшей локализации бизнес-процесса системы на стороне разработчика. Эта проблема особенно актуальна в свете разнородности структур баз данных КИС. На рисунке 1 представлена контекстная диаграмма модели сбора общих требований к программному средству обработки банков данных в нотации IDEF0.

Модель включает внешние и внутренние факторы, влияющие на требования к программному средству обработки банков данных. Контекстная диаграмма состоит из четырех процессов:

– «Определение логики построения структуры данных» – информационная структура предприятия определяет эксплуатационные характеристики КИС и уровень безопасности обрабатываемых в системе