УДК 004.9

Б.А. Князев

Алгоритм параллельной агрегации данных для визуализации данных о вербальном и невербальном поведении человека

Аннотация. В статье рассмотрен метод визуализации форм вербального и невербального поведений человека, представляющие собой данные большого объема. Приведены модель и алгоритм визуализации этих данных с использованием метода параллельной агрегации. Предложена агрегирующая функция, выполняющая поиск экстремумов блоков данных с помощью модернизированного алгоритма «reduction tree», что позволяет приблизить сложность алгоритма к минимальной. Оптимизация осуществлена за счет отображения данных в глобальную память видеопроцессора, большей нагрузке каждого потока и использования меньшего количества потоков в одном блоке. Предоставлены результаты сравнительного анализа пропускной способности центрального процессора и двух типов графического процессора, исполняемых предложенный алгоритм. Продемонстрировано значительное увеличение производительности только для второго, более мощного видеопроцессора. Однако она резко снижается при уменьшении объема данных и становится сравнимой с производительностью центрального процессора.

Ключевые слова: визуализация данных, биометрические данные, данные большой размерности, графический процессор, снижение размерности

Parallel aggregation algorithm for the visualization of human verbal and non-verbal data

Abstract. In this article the method for visualization of human verbal and nonverbal behavioural features which represent high-dimensional data is examined. The model and the algorithm for the visualization of these data using the parallel aggregation method are presented. The aggregation function calculating the extremums of data chunks based on the optimized reduction tree algorithm is suggested. This allows approaching the complexity of the overall algorithm to its minimum. Optimization is achieved by the mapping of data to the video processor global memory, processing more data per thread and using fewer threads per block. A comparative study of the throughput of a CPU and two series of a GPU, executing the developed algorithm, is conducted and its results are presented. Significant performance increase is demonstrated only for the second more powerful GPU. However, it plummets when data dimension lowers and becomes comparable with the one of the first GPU and the CPU.

Keywords: data visualization, biometric data, high-dimensional data, GPU, reduction tree

E-mail: bknyazev@bmstu.ru

1. Введение

Вербальное и невербальное поведения могут рассматриваться как процессы, изменяющиеся во времени. Для решения задач безопасности, медицинской и психологической диагностики, робототехники и других задач необходима объективная оценка параметров данных процессов [1]. Оценка может осуществляться с помощью интерпретации этих параметров в виде временных и частотных графиков.

При том частота движений частей тела и элементов лица не превышает 10-12 Гц (≤ 12 Гц для пальцев рук [2,3], ≤ 10 Гц для жестов рук и движений тела в целом [3,4] и ≤ 4 Гц для изменения мимики лица [5]); около 90% энергетической составляющей речевого сигнала лежит в диапазоне 100-5000 Гц [6]. Таким образом, из теоремы Котельникова следует, что для исключения значительных потерь исходного сигнала частота дискретизации исследуемых в данной работе невербальных и вербальных сигналов должна быть ≥ 25 кадров/сек и ≥ 10 КГц соответственно.

Длительность исследований, записанных на видео и/или аудио материалы, может достигать нескольких часов. Таким образом, количество отсчетов данных для визуализации N' равно размеру данных N:

$$N' = N = 3600 \cdot F \cdot L \text{ точек}, \quad (1)$$

где L – длительность исследования в часах, F – частота отсчетов в секунду.

Так, например, при стандартной частоте видеокадров F=25 кадров/сек (T_{fps}=0.4 сек/кадр) и длительности видеоматериала L=5 часов количество точек для отображения:

$$N = 3600 \cdot 25 \cdot 5 = 4.5 \cdot 10^5 \text{ точек}.$$

Количество точек для отображения при стандартной частоте дискретизации аудио сигнала $F_{Дискр}$=11025 Гц и длительности аудиоматериала L=5 часов:

$$N = 3600 \cdot 11025 \cdot 5 = 2 \cdot 10^8 \text{ точек}.$$

При этом объем данных для визуализации:

$$V = N \cdot B/8 \text{ байт}, \quad (2)$$

где B – количество бит на один отсчет (*bits per sample*).

Так, при количестве бит на отсчет B=24, которым соответствует диапазон $20 \cdot log_{10}(2^{24}) = 144$ дБ, необходимо отобразить $V = 2 \cdot 10^8 \cdot 24/8 = 6 \cdot 10^8$ байт ≈ 0.56 Гб информации. При этом оценка использования ресурсов, используемых ЭВМ для визуализации, показала, что при размере данных всего 100 Кб, их отображение занимает около 100 секунд, размер выделенной памяти ОЗУ около 40 Мб.

Существует следующая парадигма для визуализации данных, размер которых превышает размер отображаемой области (например, дисплея монитора) [7]:

обзор данных→масштабирование и фильтрация→детализация по требованию.

Для эффективного (с частотой ≥ 10 операций в секунду) обзора данных необходимо снижение их размерности либо с помощью методов аппроксимации кривой, либо с помощью агрегирования данных. При этом под агрегированием в общем случае понимается объединение нескольких элементов в единое целое. При размере данных, достигающих 10^9 отсчетов, для выполнения быстрого масштабирования и фильтрации данных также необходимы высокоэффективные агрегирующие функции или специальные масштабируемые типы данных [8].

Методы параллельных вычислений также могут быть использованы для выполнения задачи агрегирования, так как [9]:

– они предназначены для вычисления независимых блоков данных;

– графические процессоры (*GPU*) имеют необходимую для данной задачи пропускную способность (до 63.4 Гб/сек для процессора *G80*);

– при корректном использовании ресурсов *GPU* значительно (до 100 раз) превосходят производительность центрального процессора (*CPU*).

Так, пиковая пропускная способность процессора *G80* достигает 346 GFLOPS (гигафлопс, количество операций с плавающей запятой в секунду, умноженных на 10^9,

выполняемых процессором, прим. автора: GFLOPS, как и другие размерности, специально не выделена *курсивом*), в то время как для процессора *CPU Intel E7500* она достигает 23 GFLOPS (рис. 1).

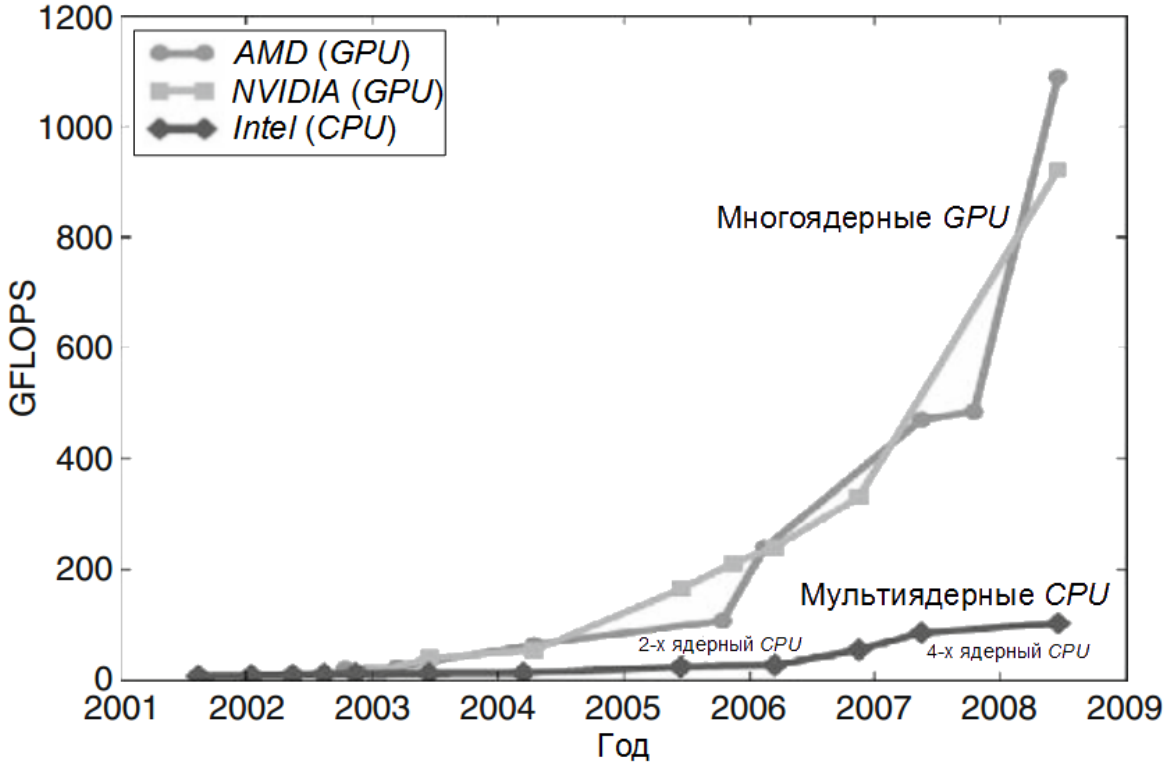


Рис. 1. Сравнение производительности процессоров *GPU* и *CPU* (перевод рисунка из [9]). *AMD*, *NVIDIA* и *Intel* – фирмы-производители процессоров.

Таким образом, повышение эффективности визуализации данных может осуществляться на двух уровнях:

1. *агрегирование данных → размерность ↓→ быстрый обзор данных;* (текст Times New Roman, стрелки и формулы Cambria Math)
2. *параллельное агрегирование → скорость ↑→ быстрое масштабирование и навигация.*

В данной работе предлагается модель и алгоритм параллельной агрегации данных, позволяющие эффективно отображать отсчеты процессов, изменяющихся во времени, а также производить быстрое масштабирование и перемещение по отсчетам. Новациями данной работы являются: исследование и сравнение характеристик процессоров *CPU* и

GPU для алгоритма дерева редукций (*reduction tree*) и его оптимизированной версии; исследование возможности и целесообразности использования *GPU* для визуализации данных большого размера.

Решение данной задачи основывается на следующих принципах:

– на одной точке (пикселе) монитора может отображаться не более одного отсчета;

– высота графика без масштабирования $H \leq 2^B$.

В рамках следующих условий и ограничений:

– обеспечен постоянный доступ к изначальным данным;

– промежуточные данные могут храниться только в ОЗУ;

– требуемая разрешающая способность визуализации данных $Q \leq Min(T_{fps}, 1/F_{Дискр})$;

– сложность алгоритма параллельной агрегации → *O(log(N))*.

2. Модель визуализация данных M_S

2.1. Общий случай

Модель визуализации данных может быть описана в виде следующего теоретико-множественного представления:

$$M_S = < S, F, R, P >, \text{(выравнивание по центру)}, \quad (3)$$

где S – исходные данные о невербальном и вербальном поведении (рис. 2), F – частота отсчетов данных, соответствующая требуемой разрешающей способности Q. $R = \{r_k\}, k = 1:n$ – набор некоторых правил агрегирования данных, n – количество правил, P – набор параметров визуализации:

$$P =< W, H, x_0, s_X, s_Y >, \quad (4)$$

где W, H – соответственно ширина и высота области, в пределах которой необходимо визуализировать данные; x_0 – отступ по оси X, начиная с которого

необходимо отображать данные (параметр, необходимый для перемещения по графику); s_X, s_Y – масштабы визуализации по осям X и Y соответственно.

Таким образом, модель визуализации данных S может быть представлена следующим образом:

$$\forall\, i,j \in 1{:}\,W \;\; \exists\, \{P,R,F\}, \Delta s_i \subseteq S{:} \begin{cases} \forall \Delta s_j \subseteq S, \Delta s_i \wedge \Delta s_j = \emptyset, i \neq j, \\ \Delta s_i = S[s_0 + (i-1)\cdot L(\Delta s_i){:}\, s_0 + i \cdot L(\Delta s_i)], \\ \forall s_{i,m} \in \Delta s_i \; \exists r_k(s_{i,m}){:}\, R(\Delta s_i) \rightarrow \{v_{i,k}\}, \end{cases} \quad (5)$$

где $\Delta s_i, \Delta s_j$ – непересекающиеся i-ый и j-ый блоки данных, принадлежащие исходным данным S; $L(\Delta s_i) = N/(s_X \cdot W)$ – размер блока данных Δs_i, отображаемых на i-ом пикселе, N – размер всех исходных данных S; $s_0 = x_0/(s_X \cdot W)$ – отсчет данных S, соответствующий позиции x_0; $s_{i,m}$ – отсчет данных из блока Δs_i, $m \in 1{:}\,L(\Delta s_i)$; $\{v_{i,k}\}$ – набор значений, возвращаемых правилом агрегации R для каждого блока данных Δs_i, $k \in 1{:}\,n$; n – количество значений в таком наборе, равное количеству правил.

Из данной модели следует, что по сравнению с (1), количество отображаемых отсчетов исходных данных S не зависит от размера самих данных (N) и равно:

$$N' = n \cdot W, N' \neq N. \quad (6)$$

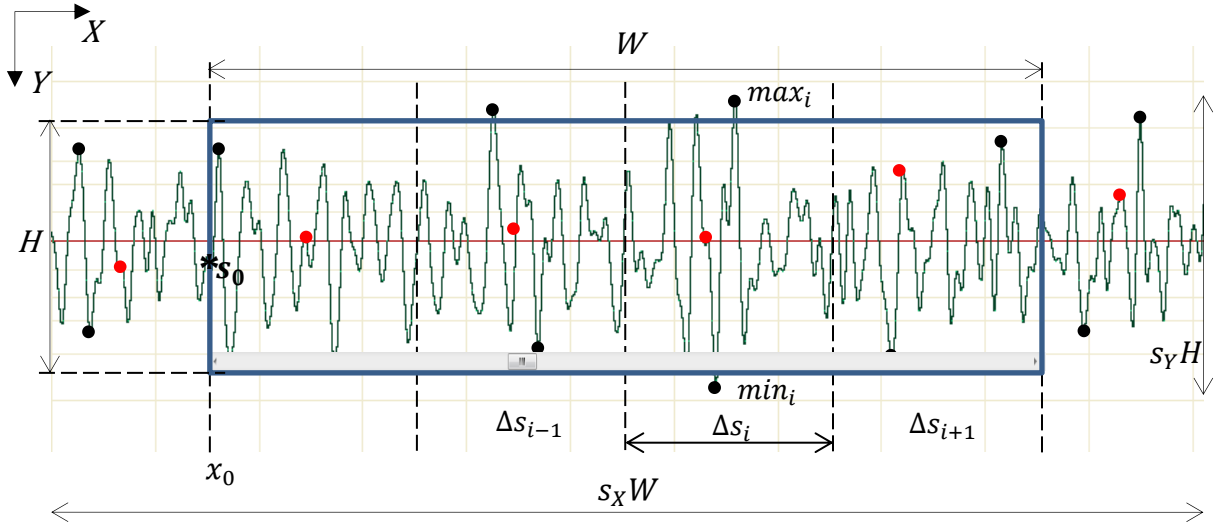


Рис. 2. График исходных данных S. $W \times H$ – видимая область отображения графика;

$s_X W \times s_X H$ – виртуальная область; черными точками показаны отсчеты, которые следует выбрать из соответствующих блоков данных; красными – неточно выбранные отсчеты.

Параметры s_X и x_0 должны быть заданы пользователем и управляют масштабом и величиной сдвигом по оси X соответственно.

Параметр s_Y можно задавать для управления масштабом по оси Y, либо вычислять значение по умолчанию для вписывания графика в область окна.

2.2. Случай для правила максимума и минимума

В качестве правила агрегации может быть использовано следующее правило:

$$R(\Delta s_i) = \{r_{max}, r_{min}\} = \begin{cases} r_{max}: \forall s_{i,m} \in \Delta s_i \;\; s_{i,m} \le max_i \\ r_{min}: \forall s_{i,m} \in \Delta s_i \;\; s_{i,m} \ge min_i \end{cases}, \quad (7)$$

где max_i, min_i – соответственно максимум и минимум блока данных Δs_i; $m \in 1: L(\Delta s_i)$.

При использовании такого правила агрегирования на каждом интервале данных Δs_i будут вычислены max_i и min_i (рис. 3). Изменение масштаба по оси X при этом влияет только на размер блока данных Δs_i: $L(\Delta s_i) = f(1/s_X)$.

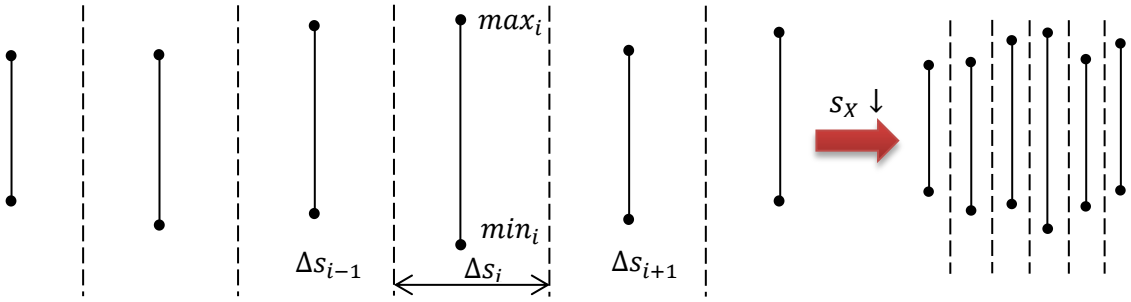


Рис. 3. Вычисление агрегирующей функции и последующее масштабирование; справа показан график в уменьшенном масштабе.

Модель визуализации данных M_S при этом по сравнению с (5) будет отличаться только нижней строкой:

$$\forall\, i,j \in 1{:}\,W\ \ \exists\,\{P,R,F\}, \Delta s_i \subseteq S{:}\begin{cases} \forall \Delta s_j \subseteq S, \Delta s_i \wedge \Delta s_j = \emptyset, i \neq j, \\ \Delta s_i = S[s_0 + (i-1)\cdot L(\Delta s_i){:}\, s_0 + i \cdot L(\Delta s_i)], \\ \forall \boldsymbol{s_{i,m}} \in \Delta \boldsymbol{s_i}\ \exists \boldsymbol{r_k}(\boldsymbol{s_{i,m}}){:}\, \boldsymbol{R}(\Delta \boldsymbol{s_i}) \rightarrow \{\boldsymbol{max_i}, \boldsymbol{min_i}\}, \end{cases} \quad (8)$$

где $k = 1{,}2; n = 2$. Тогда, в соответствии с (6):

$$N' = 2 \cdot W. \quad (9)$$

При разрешении дисплея $W \times H = 1920 \times 1080 \rightarrow N' = 2 \cdot 1920 = 3840$ точек.

Таким образом, при заданном B, например $B = 24$, имеем постоянный объем визуализируемых данных в соответствии с формулой (2): $V = N \cdot B/8 = 11.25$ Кб, который не зависит от масштаба s_X и количества отсчетов исходных данных *N*.

3. Реализация алгоритма Reduction Tree на GPU для поиска экстремумов

При исходном $N = 2 \cdot 10^8$ и количестве блоков Δs_i равном ширине окна в пикселях *W* размер блоков, для которых применяется правило *R*, может достигать значения 10^6 (см. (5)). При реализации последовательного алгоритма применения данного правила для всех блоков сложность операции $O(W \cdot Length(\Delta x_i)|_{s_X=1, x_0=0}) = O(N)$, что будет эффективно только для небольшого ($\sim 10^2$) количества элементов (рис. 4). Для реализации алгоритма параллельной агрегации данных предлагается использование алгоритма дерева редукций (*reduction tree*) [10], оптимизированного в данной работе под выполнение задачи поиска минимума и максимума массива данных.

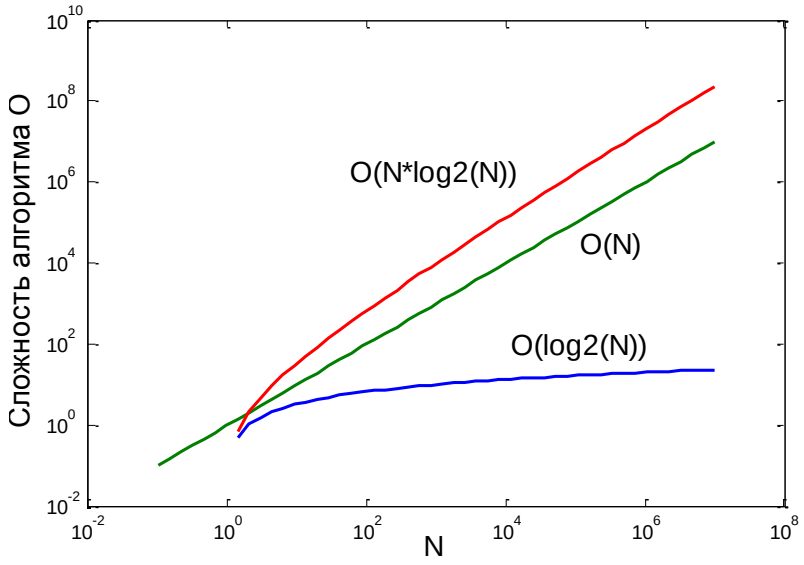


Рис. 4. Сложности алгоритмов при последовательной и параллельной реализации. Красным и синим показаны соответственно наихудший и наилучший случаи параллельной реализации; зеленым показан случай последовательной реализации.

Оценить сложность алгоритма дерева редукций при параллельной реализации несколько сложнее, чем при последовательной, так как заранее неизвестно, сколько потоков будет выполняться параллельно. Данный факт зависит от параметров и архитектуры конкретного графического процессора (*GPU*) [9] (в скобках указаны доступные на сегодняшний день значения):

- количество потоков на один мультипроцессор (768, 1024, 1536);
- количество блоков потоков (*thread block*) на один мультипроцессор (8,16);
- количество мультипроцессоров (16, 30);
- объем разделяемой памяти (*shared memory*) (16, 32 Кб);
- количество регистров на один мультипроцессор (8192, 16384, 32768);
- и другие параметры.

В худшем случае, т.е. при последовательном выполнении инструкций каждого блока, сложность алгоритма $O(N \cdot \log(N))$. В лучшем случае, т.е. при параллельном выполнении инструкции всех блоков, сложность алгоритма $O(\log(N))$ (рис. 4).

Одной из задач данной работы является стремление сложности к $O(\log(N))$. Для поиска минимума и максимума по алгоритму дерева редукций необходимо:

1. Инициализировать адресное пространство в глобальной памяти *GPU*.
2. Разделить входной массив на блоки данных Δs_i в соответствии с моделью $\boldsymbol{M_S}$, так чтобы количество блоков данных было равно ширине окна в пикселях *W*.
3. Вычислить локальный минимум и максимум на каждом блоке.

Базовый алгоритм дерева редукций не эффективен по следующим причинам. Максимальное количество обрабатываемых элементов ограничено размером сетки блоков (*grid*) (2^{16} для процессоров *G80* и *GF104*) и размером потоков на один блок (2^9 и 2^{10} для процессоров *G80* и *GF104* соответственно). Таким образом, максимальное размер входных данных 2^{25}-2^{26}<3.3-6.7·10^7, в то время как для данной задачи необходимо 2·10^8 элементов. Каждый поток обрабатывает максимум 2^9-2^{10} элементов и, соответственно, не полностью использует ресурсы регистров и разделяемой памяти и, как следствие, всего процессора в целом [11]. Таким образом, при такой реализации сложность алгоритма будет стремиться к наихудшей ($O(N \cdot \log(N))$), и проигрывать даже последовательной реализации (рис. 4).

Данные ограничения предлагается обойти следующими способами. С помощью технологии отображения памяти (*memory mapping*), т.е. вызовом функций *cudaHostRegister* и *cudaHostGetDevicePointer* из библиотеки параллельных вычислений *CUDA* для графических процессоров *NVIDIA*, можно инициализировать значительно больше памяти, так как ее размер ограничен лишь размером глобальной памяти процессора (640 Мб и 768-2048 Мб для процессоров *G80* и *GF104* соответственно). Также, в соответствии с [10] необходимо больше нагружать каждый поток, что позволит повысить пропускную способность процессора в целом. В [11] также показано, что меньшее количество потоков в одном блоке ведет к увеличению производительности.

Таким образом, для решения данной задачи необходимо вычислять 1024-4096 элементов в потоке и от 64 до *W* блоков по 128-256 потоков (рис. 5).

Так, при *W*=1920, количество элементов, которые могут быть обработаны *GPU* $N = 1920 \cdot 128 \cdot 4096 = 10^9$. Пусть ShM – размер разделяемой памяти, доступной одному блоку потоков. Тогда, максимальное количество элементов *Npb*, обрабатываемых одним блоком, может быть вычислено как

$$Npb = ShM/\,(2 \cdot (B/8)) = 4 \cdot ShM/B. \quad (10)$$

При $ShM = 16$ Кб $Npb = 4 \cdot 16384/8 = 8129$ элементов. Таким образом, в цикле необходимо подгружать элементы из глобальной памяти (*global memory*) $\lfloor 128 \cdot 4096/8129 \rfloor = 65$ раз. Следует заметить, что доступ к глобальной памяти достигает 80 Гб/с, что позволяет быстро обращаться к входным данным из процессов *GPU* [12].

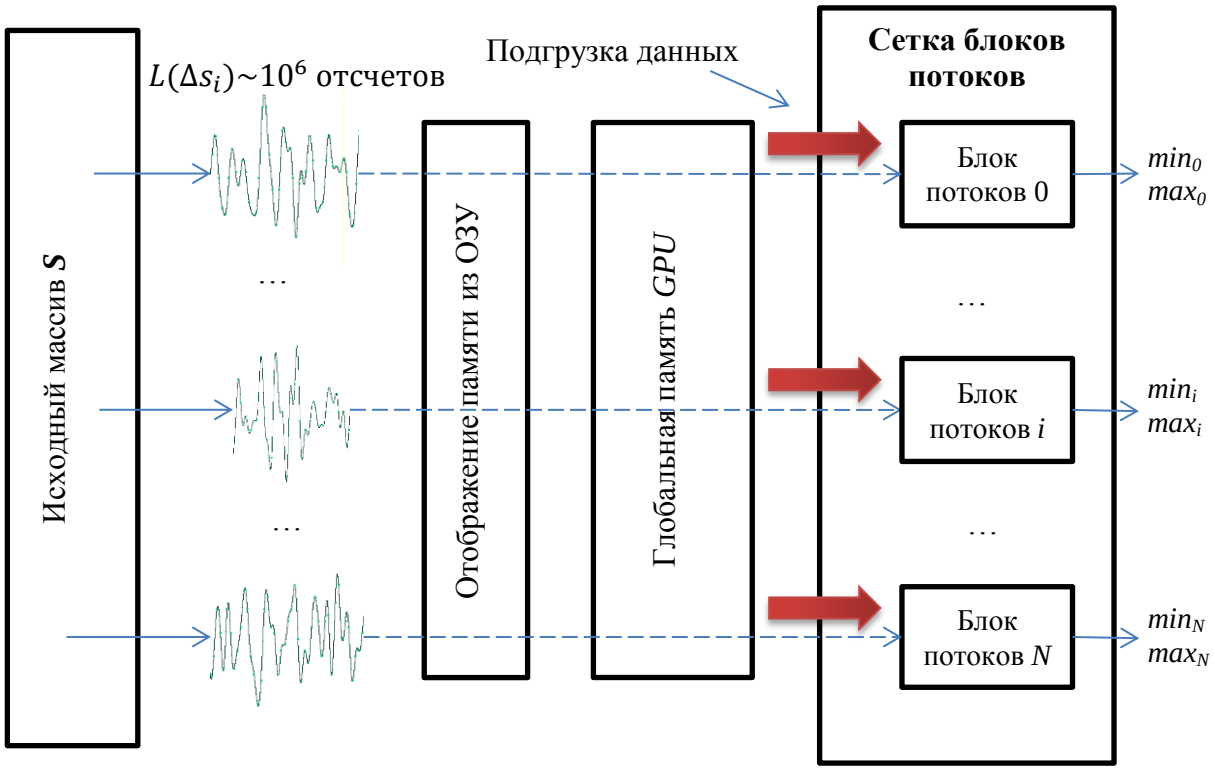


Рис. 5. Алгоритм параллельной агрегации данных.

Ниже приведен листинг функции вычисления экстремумов для блока данных Δs_i. При правильном подборе размера блоков (*blockSize*) и размера сетки блоков (*blockDim*), инструкции для всех блоков выполняются одновременно.

```
// шаблонная функция вычисления минимума и максимума блока данных
```

```
// в качестве типа T может быть int, byte, float и другие
template <class T, unsigned int blockSize>
__global__ void minmaxReduction(T * input, T * out_min, T * out_max, int len)
{

    T* sMin = SharedMemory<T>();  // объявляем extern __shared__ переменные
    T* sMax = &sMin[blockSize];

    … // инициализация параметров

    while (i < len)
    {
      // загружаем минимумы/максимумы в переменную sMin (sMax)
      // проверяем граничные условия
      if ( i + blockDim.x < len)
        sMin[t] = min(sMin[t], min(input[i], input[i + blockDim.x]));
      else
        sMin[t] = min(sMin[t], input[i]);
      i += gridSize; // подгружаем следующий блок данных
    }
    __syncthreads(); // синхронизируем потоки внутри блока

    // цикл модифицированного алгоритма reduction tree
    for (unsigned int stride = blockDim.x/2; stride > 32; stride >>= 1)
    {
      if (t < stride)
        sMin[t] = min(sMin[t], sMin[t + stride]);
      __syncthreads();
    }
// раскрываем цикл для потоков в пределах порций потоков (t < 32), избавляясь
от необходимости синхронизировать потоки в пределах порций потоков (warp)
    if (t < 32)
    {
      // объявляем временные переменные volatile, чтобы на уровне компиляции
избежать некорректного поведения
        volatile T *minV = sMin;

          …  // аналогично коду в цикле for
    }
  if (t == 0) // в первом потоке блока вычисляются конечные величины
      out_min[blockIdx.x] = sMin[0];
}
```

4. Эксперимент и результаты

Для оценки эффективности обзора данных, масштабирования и навигации данных проводилось два исследования:

1. Оценка скорости алгоритма агрегирования и пропускной способности *CPU* и *GPU*.
2. Оценка загруженности ресурсов *CPU* и *GPU* при построении данных большого размера.

Для исследования скорости работы алгоритма и пропускной способности графического процессора использовались шумовые сигналы длительностью 2^{10}-2^{30} отсчетов по 1-4 байтов на отсчет. Сравнение производительности алгоритма проводилось между центральным процессором *CPU E7500*, графическим процессором *G86* и *GF104* [12] (табл. 1).

Таблица 1

Характеристики используемых для исследования процессоров

Процессор	Пропускная способность *C*, Гб/сек	Производительность, GFLOPS	Размер глобальной памяти *GlobalMemSize*, байт	Размер сетки блоков *GridSize*, кол-во потоков	Размер блока потоков *BlockSize*, кол-во потоков
CPU E7500	5.3	23.464	-	-	-
G86	6,4	22	2^{29}	2^{16}	2^{9}
GF104	108.8	748.8	2^{30}	2^{16}	2^{10}

Исследовались четыре типа данных: размером 1 байт (тип *byte*), 2 байта (тип *short*), 4 байта (тип *int*), 4 байта (тип *float*) (табл. 2). Для каждого типа данных оценивалась пропускная способность процессора по формуле:

$$C = N \cdot sizeof(T)/T_{reduce}, \quad (11)$$

где T – тип входных данных; $sizeof(T)$ – размер типа входных данных в байтах, T_{reduce} – время работы алгоритма в секундах. Данные пиковых (максимальных) значений для C представлены в таблице 2.

В случае базового алгоритма дерева редукций максимальное количество отсчетов входных данных N_{MAX}, возможных для визуализации, вычисляется по формуле:

$$N_{MAX} = GridSize \cdot BlockSize, \quad (12)$$

где *GridSize* – максимальный размер сетки блоков (кол-во потоков в сетке); *BlockSize* – максимальный размер блока потоков (кол-во потоков в блоке). В то время как, в случае оптимизированного алгоритма дерева редукций данное значение намного больше и удовлетворяет требованиям данной задачи:

$$N_{MAX} = GlobalMemSize/sizeof(T), \quad (13)$$

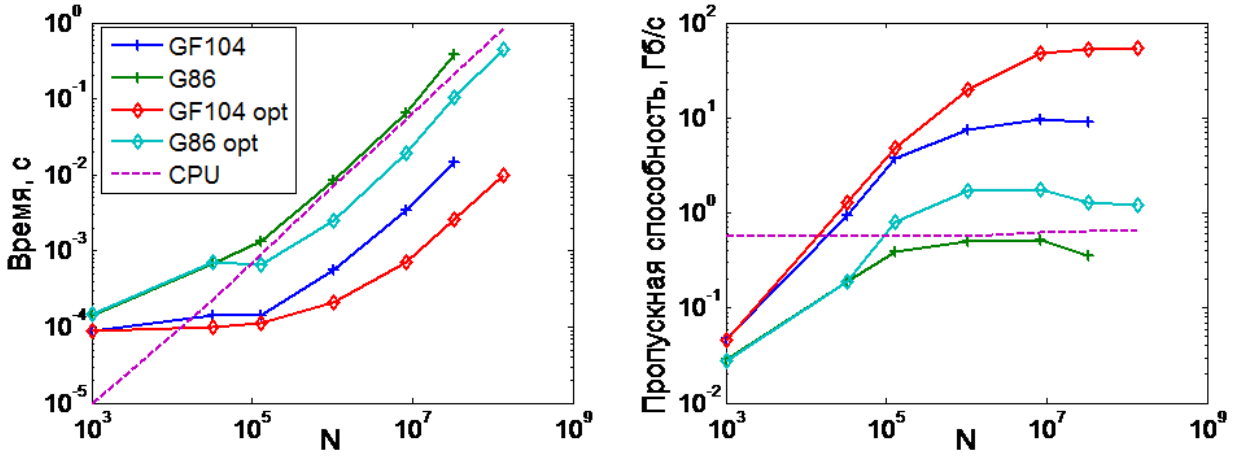
где *GlobalMemSize* – максимальный размер доступной глобальной памяти *GPU*.

Таблица 2

Пиковая пропускная способность C_{MAX}, Гб/с

	Базовый алгоритм дерева редукций				Оптимизированный алгоритм дерева редукций			
Тип данных (размер, байт)	*byte* (1)	*short* (2)	*int* (4)	*float* (4)	*byte* (1)	*short* (2)	*int* (4)	*float* (4)
N_{MAX} для *G86* (*GF104*)	2^{25} (2^{26})				2^{29} (2^{30})	2^{28} (2^{29})	2^{27} (2^{28})	2^{27} (2^{28})
CPU	0.14	0.29	0.57	0.38	0.14	0.29	0.57	0.38
G86	0.04	0.12	0.55	0.55	0.062	0.12	**1.8**	**1.9**
GF104	2.3	4.7	9.8	9.3	10.8	24.7	**54.7**	**54**

Время выполнения алгоритма на *CPU* линейно зависит от *N*, что подтверждает сложность алгоритма последовательной агрегации данных *O(N)* (рис. 6). Следовательно, пропускная способность процессора *CPU*, рассчитанная по формуле (11), имеет постоянное значение.



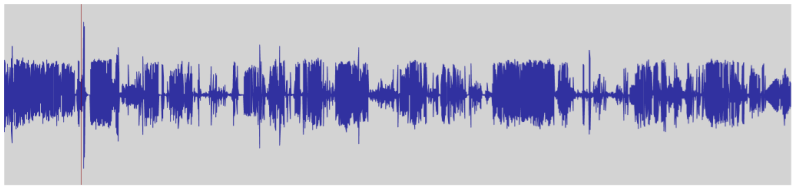
а) б)

Рис. 6. Сравнение времени выполнения алгоритма (а) и пропускной способности (б) в зависимости от количества отсчетов входных данных.

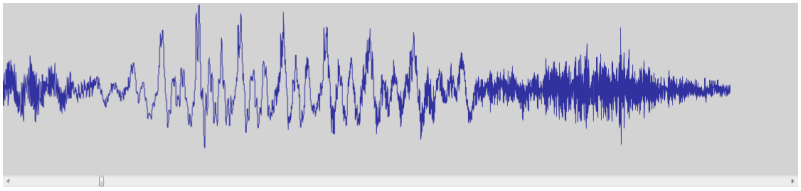
Время выполнения алгоритма на *GPU* имеет более сложную зависимость от N, что подтверждает постепенное изменение сложности алгоритма параллельной агрегации данных от $O(\log(N))$ *до* $O(N \cdot \log(N))$ в зависимости от количества параллельных блоков потоков, запускаемых одновременно. Так, до некоторого порогового значения $N \sim 10^5$ время выполнения алгоритма практически постоянно, так как у *GPU* есть ресурсы для запуска небольшого количества потоков одновременно. Далее время выполнения стремится к линейной зависимости, и пропускная способность стремится к пиковому значению. Также следует заметить, что целесообразность применения параллельного алгоритма возникает только при $N > 10^4$.

Для исследования возможностей эффективного масштабирования и навигации по данным использовались аудио сигналы длительностью 47 и 243 мин. (рис 7).

Вербальный сигнал №1: $N \approx 5 \cdot 10^8$, D = 00:47:09.840, B = 16, $F_{Дискр}$ = 96 КГц



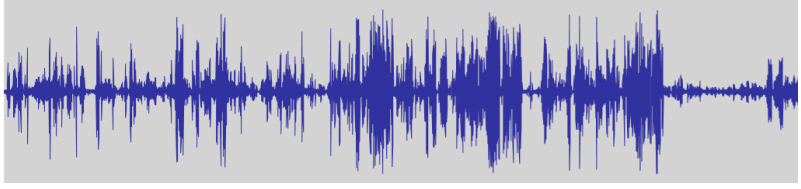
Масштаб s_x = 1



Масштаб $s_x = 2 \cdot 10^4$

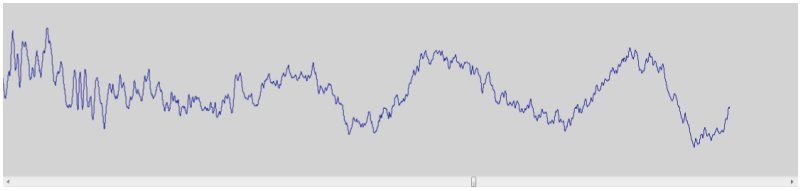
(эти данные для обоих графиков)
Загрузка *CPU*:
LoadC = 25%
Mem = 70 Мб = *const*
Загрузка G*PU*:
LoadG = 98%
MemG = 225 Мб

Вербальный сигнал №2: $N \approx 3 \cdot 10^9$, D = 04:23:10.784, B = 16, $F_{Дискр}$ = 96 КГц



Масштаб s_x = 1

(эти данные для обоих графиков)
Загрузка *CPU*:
LoadC = 25 %
Mem = 40 Мб = *const*



Загрузка G*PU*:
LoadG = 96 %
MemG = 663 Мб

Масштаб $s_x = 2 \cdot 10^5$

Рис. 7. Результаты визуализации, масштабирования и навигации по аудиоданным с применением оптимизированного алгоритма дерева редукций. *D* – длительность аудио материала, *LoadC* и *LoadG* – загрузка *CPU* и *GPU* при масштабировании и навигации по оси *X*, *Mem* – загрузка ОЗУ, *MemG* – загрузка памяти *GPU*.

5. Заключение

В данной работе исследован метод визуализации данных большого размера, таких как данных о вербальном и невербальном поведениях человека. Разработана модель визуализации данных с учетом возможности масштабирования и перемещения по графикам, а также разработан алгоритм параллельной агрегации данных, заключающийся в нахождении экстремумом блоков данных. На основе предложенной модели и алгоритма разработано программное обеспечение, позволяющее проводить исследование эффективности данного алгоритма. По результатам исследования производительность процессора *G86* оказалась выше производительности *CPU* только для 32-битных типов данных; при том производительность *GF114* оказалась в ~100 раз выше *CPU* для чисел с плавающей запятой, приблизив сложность параллельного алгоритма дерева редукций к минимальной. Данное исследование показывает возможность эффективной визуализации данных большого размера и навигации по ним с помощью ПО на основе *CUDA API* и процессора *GF114*. Использование более слабых видеопроцессоров или применение алгоритма для данных небольшой размерности нецелесообразно, так скорость работы при этом может не превосходить скорость реализации на *CPU*. В последующих работах предлагается дальнейшая оптимизация алгоритма за счет априорных знаний о сигнале и

использование бикубической аппроксимации для сглаживания графика при большом масштабе.

Список литературы

1. Алфимцев А.Н. Разработка и исследование методов захвата, отслеживания и распознавания динамических жестов: Дис. канд.техн.наук. Москва, 2008, 167 с.

2. Analysis of finger-tapping movement// Jobbagy A. [et al.] Neurosci Methods. 2005. 141(1). p. 29-39

3. Ильин Е. П. Психомоторная организация человека: Учебник для вузов. СПб.: Питер, 1-е издание, 2003. 384с.

4. York JL, Biederman I. Hand movement speed and accuracy in detoxified alcoholics// Alcohol Clin Exp Res. 1991. 15. p. 982-990

5. B. Fasel, J. Luettin. Automatic facial expression analysis: a survey// Pattern Recognition. 2003. Vol. 36, Iss. 1. p. 259-275

6. Сапожков М.А. Электроакустика. Учебник для вузов. М.: Связь, 1978. 272 с.

7. Shneiderman B. The eyes have it: a task by data type taxonomy for information visualizations // Proceedings of the IEEE Symposium on Visual Languages, pp. 336-343, 1996

8. Shneiderman B. Extreme visualization: squeezing a billion records into a million pixels // Proceedings of the 2008 ACM SIGMOD international conference on Management of data, pp. 3-12, 2008, Vancouver, Canada

9. David B. Kirk, Wen-mei W. Hwu. Programming Massively Parallel Processors: A Hands-on Approach. San Francisco: Morgan Kaufmann Publishers Inc., CA, USA, 2010, 280 p.

10. Harris M. Optimizing Parallel Reduction in CUDA, NVIDIA Developer Technology URL: http://developer.download.nvidia.com/assets/cuda/files/reduction.pdf (дата обращения: 22.05.2013)

11. Volkov V. Better performance at lower occupancy // Proceedings of the GPU Technology Conference, GTC 10.

12. Comparison of Nvidia graphics processing units // Wikipedia. The Free Encyclopedia. URL:http://en.wikipedia.org/wiki/Comparison_of_Nvidia_graphics_processing_units (дата обращения: 22.05.2013)