УДК 004.9

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

© Б.А. Князев

МГТУ им. Н.Э. Баумана, Москва, 105005, Россия

Рассмотрен метод визуализации поведения человека вербальной и невербальной форм, представляющих 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 нагрузки каждого потока и использования меньшего количества потоков в одном блоке. Представлены результаты сравнительного анализа пропускной способности центрального процессора и двух типов графического процессора, выполняющих предложенный алгоритм.

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

Введение. Вербальное и невербальное поведения могут рассматриваться как процессы, изменяющиеся во времени. Для решения таких задач, как безопасность, медицинская и психологическая диагностика, робототехника и др., необходима объективная оценка параметров данных процессов [1]. Оценка может осуществляться с помощью интерпретации этих параметров в виде временны́х и частотных графиков. При этом 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 находится 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/с и ≥ 10 КГц соответственно.

Длительность исследований, записанных на видео-и/или аудионосители, может достигать нескольких часов. Следовательно, количество отсчетов данных для визуализации N' равно объему данных N:

$$N = 3600\, F\, L \text{ точек}, \qquad (1)$$

где L — длительность исследования, ч; F — частота отсчетов, с.

Например, при стандартной частоте видеокадров $F = 25$ кадров/с ($T_{fps} = 0{,}4$ с/кадр) и длительности видеоматериала $L = 5$ ч 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 аудиосигнала $F_{\text{дискр}} = 11025$ Гц и длительности аудиоматериала $L = 5$ ч:

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

При этом объем данных для визуализации

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

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

Так, при $B = 24$, чему соответствует диапазон $20 \log_{10}(2^{24}) = 144$ дБ, необходимо отобразить $V = 2 \cdot 10^8 \cdot 24/8 = 6 \cdot 10^8$ байт $\approx 0{,}56$ Гб информации. При этом оценка использования ресурсов ЭВМ, применяемых для визуализации, показала, что при объеме данных всего 100 Кб, их отображение занимает около 100 с, размер выделенной памяти оперативного запоминающего устройства (ОЗУ) — около 40 Мб.

Существует следующая парадигма для визуализации данных, объем 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 с помощью либо методов аппроксимации кривой, либо 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].

Для задачи агрегирования также могут быть использованы методы параллельных вычислений, поскольку [9]:

• предназначены для вычисления независимых блоков данных;

• графические процессоры (*GPU*) имеют необходимую для данной задачи пропускную способность (до 63,4 Гб/с 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 выполняемых процессором

операций с плавающей запятой в секунду, умноженных на 10^9), в то время как для процессора *CPU Intel E7500* она достигает 23 GFLOPS (рис. 1).

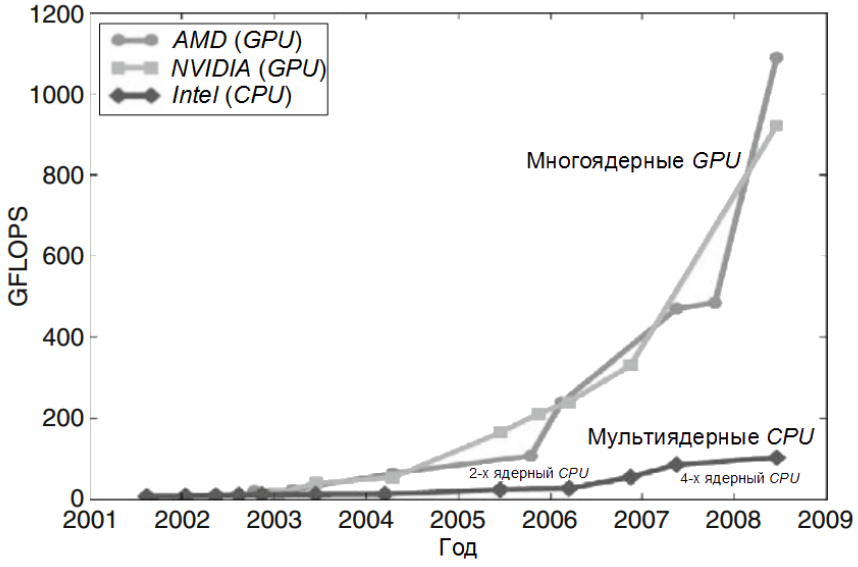


Рис. 1. Сравнение производительности процессоров *GPU* и *CPU* [9]: *AMD*, *NVIDIA* и *Intel* – фирмы-производители процессоров

Следовательно, повышение эффективности визуализации данных может осуществляться на двух уровнях:

• *агрегирование данных* → *размерность* ↓→ *быстрый обзор данных*;

• *параллельное агрегирование* → *скорость* ↑→ *быстрое масштабирование и навигация*.

В данной работе предлагается модель и алгоритм параллельной агрегации данных, которые позволяют 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 большого объема данных.

В основу решения поставленной задачи заложены следующие принципы:

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

• высота графика без масштабирования — $H \le 2^B$,

а также следующие условия и ограничения:

• постоянный доступ к изначальным данным;

• хранение промежуточных данных только в ОЗУ;

• требуемая разрешающая способность визуализации данных — $Q \le$ $\le \min(T_{fps}, 1/F_{\text{дискр}})$;

• сложность алгоритма параллельной агрегации → $O(\log(N))$.

Модель визуализации данных M_S. ***Общий случай.*** Модель визуализации данных может быть описана в виде следующего теоретико-множественного представления:

$$M_S = < S, F, R, P >, \tag{3}$$

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

$$P = < W,\ H,\ x_0,\ S_X,\ S_Y > , \tag{4}$$

здесь 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 (этот параметр нужен для перемещения по графику); s_X, s_Y — масштабы визуализации по осям X и Y соответственно.

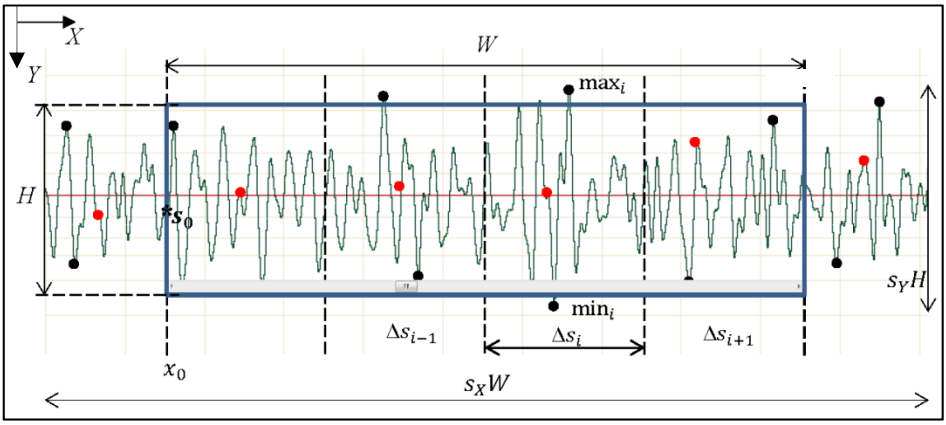


Рис. 2. График исходных данных S: $W \times H$ — видимая область отображения графика (черными точками показаны отсчеты, которые следует выбрать из соответствующих блоков данных; красными – неточно выбранные отсчеты)

Таким образом, модель визуализации может быть представлена следующим образом:

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

где Δs_i, Δs_j — непересекающиеся *i*-й и *j*-й блоки данных, принадлежащие исходным данным S; $L(\Delta s_i)$ — размер блока данных Δs_i, отображаемых на *i*-м пикселе, $L(\Delta s_i) = N/(s_X W)$, N — размер всех исходных данных S; s_0 — отсчет данных S, соответствующий позиции x_0, $s_0 = x_0/(s_X W)$; $s_{i,m}$ — 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

$$N' = nW, \; N' \neq N. \quad (6)$$

Параметры s_X и x_0 должны быть заданы пользователем, они управляют масштабом и сдвигом по оси *X*. Параметр s_Y можно задавать для управления масштабом по оси *Y* либо вычислять по умолчанию для вписывания графика в область окна.

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 Δx_i будут вычислены $\max_i$ и $\min_i$ (рис. 3). При этом изменение масштаба по оси *X* влияет только на размер блока данных Δs_i: $L(\Delta s_i) = f(1/s_X)$.

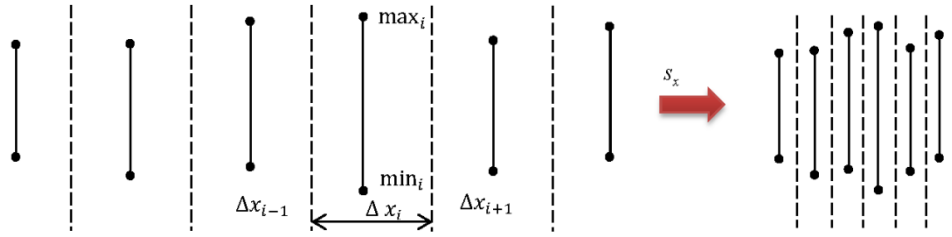


Рис. 3. Вычисление агрегирующей функции и последующее масштабирование

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

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

где $k = 1,2;\ n = 2.$

Тогда, в соответствии с (6)

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

при разрешении дисплея $W \times H = 1920 \times 1080 \to N = 2 \cdot 1920 = 3840$ точек.

Таким образом, при заданном B (например, $B = 24$) имеем постоянный объем визуализируемых данных в соответствии с формулой (2): $V = NB/8 = 11,25$ Кб, не зависящий от масштаба s_X и количества отсчетов исходных данных N.

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

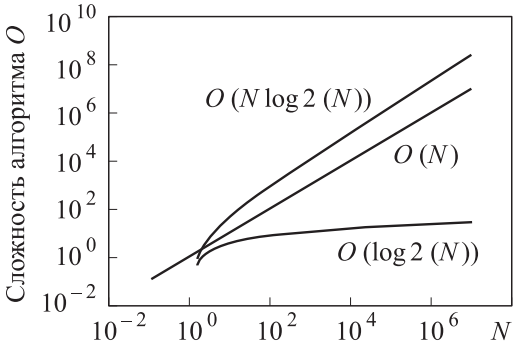


Рис. 4. Сложность алгоритмов при последовательной и параллельной реализации (красным и синим показаны соответственно наихудший и наилучший случаи параллельной реализации; зеленым — случай последовательной реализации)

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

• количество потоков на один мультипроцессор (768, 1024, 1536);

• количество блоков потоков (*thead block*) на один мультипроцессор (8,16);

• число мультипроцессоров (16, 30);

• объем разделяемой памяти (*shared memory*) (16, 32 Кб);

• количество регистров на один мультипроцессор (8192, 16384, 32768),

а также от других параметров.

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

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

• инициализировать адресное пространство в глобальной памяти *GPU*;

• разделить входной массив на блоки данных в соответствии с моделью M_S так, чтобы количество блоков данных было равно ширине окна в пикселах W;

• вычислить локальный минимум на каждом блоке.

Базовый алгоритм дерева редукций не эффективен по следующим причинам. Максимальное количество обрабатываемых элементов ограничено размерами сетки блоков (*grid*) (2^{16} для процессоров *G80* и *GF104*) и потоков на один блок (2^9 и 2^{10} для процессоров *G80* и *GF104* соответственно). Таким образом, максимальный размер входных данных $2^{25}\ldots 2^{26}<3{,}3\ \ldots\ 6{,}7\cdot 10^7$, в то время как для рассматриваемой задачи необходимо $2\cdot 10^8$ элементов. Каждый поток обрабатывает максимум $2^9\ldots 2^{10}$ элементов и соответственно не полностью использует ресурсы регистров и разделяемой памяти (*shared memory)*, а также и всего процессора [11]. Следовательно, при этой реализации сложность алгоритма будет стремиться к наихудшей $\left(O\left(N\log\left(N\right)\right)\right)$ и проигрывать даже последовательной реализации (см. рис. 4).

Названные ограничения можно 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, поскольку ее объем ограничен лишь объемом глобальной памяти процессора (640 Мб и 768 — 2048 Мб для процессоров *G80* и *GF104* соответственно). Кроме того, 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. Следовательно, для решения задачи необходимо вычислять 1024 … 4096 элементов в потоке и от 64 до *W* блоков по 128 … 256 потоков (рис. 5).

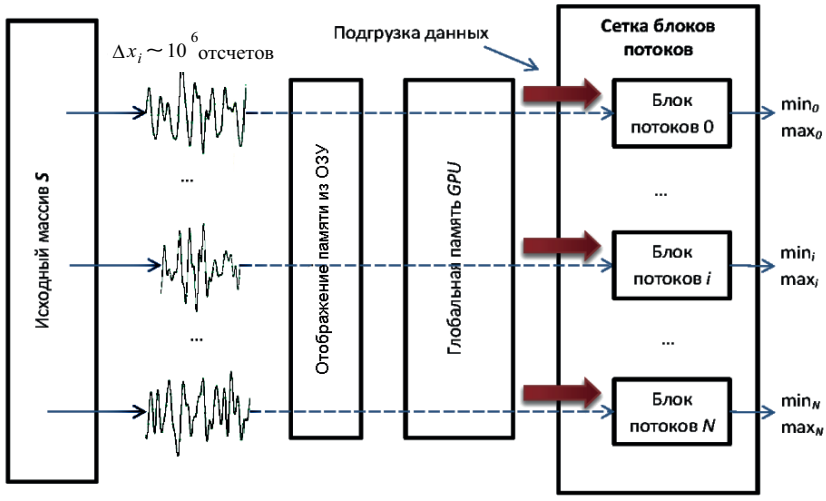


Рис. 5. Алгоритм параллельной агрегации данных

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

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

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

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];
}
```

Эксперимент и результаты. Для оценки эффективности обзора масштабирования и навигации данных проводилось два исследования: оценка скорости алгоритма агрегирования и пропускной способности *CPU* и *GPU*, а также оценка загруженности ресурсов *CPU* и *GPU* при построении большого объема данных.

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/с	Производительность, GFLOPS	Объем видеопамяти *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 объемами: 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 следующей формуле:

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

где *sizeof* (*T)* — размер типа входных данных T в байтах; T_{reduce} — время работы алгоритма, с. Данные пиковых (максимальных) значений для C представлены в табл. 2.

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

$$N_{\max} = GridSize \cdot BlockSize, \tag{12}$$

где *GridSize* — максимальный размер сетки блоков (количество потоков в сетке); *BlockSize* — максимальный размер блока потоков (количество потоков в блоке).

В случае оптимизированного алгоритма дерева редукций данное значение намного больше и удовлетворяет требованиям данной задачи:

$$N_{\max} = GlobalMemSize / sizeof(T), \tag{13}$$

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

Таблица 2

Пиковая пропускная способность $C_{\max}$, Гб/с

Алгоритм дерева редукций								
базовый					оптимизированный			
Тип данных (размер, в байтах)	*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.

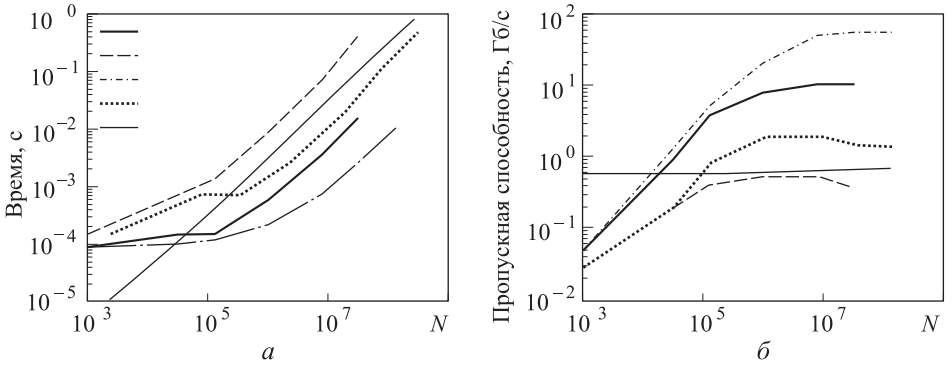


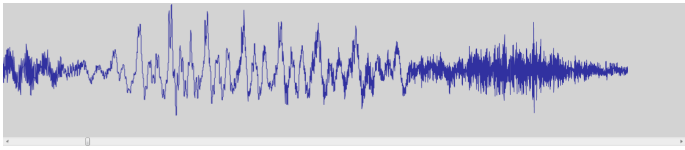
Рис. 6. Сравнение времени выполнения алгоритма (*а*) и пропускной способности (*б*) в зависимости от количества отсчетов входных данных

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 $O(N\log(N))$ 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, а пропускная способность — к пиковому значению. Следует также заметить, что целесообразность применения параллельного алгоритма возникает только при $N > 10^4$.

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

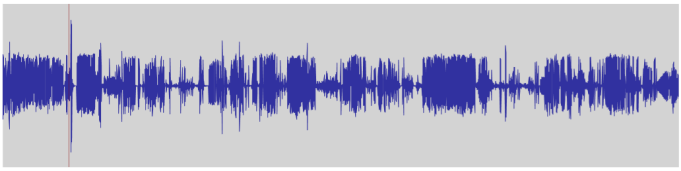
Заключение. В настоящей работе исследован метод визуализации данных большого объема, таких как данные о вербальном и невербальном поведении человека. Разработаны модели визуализации данных с учетом возможности их масштабирования и перемещения по графикам, а также алгоритм параллельной агрегации данных, заключающийся в нахождении экстремумов их блоков. На основе предложенной модели и алгоритма создано программное обеспечение, позволяющее проводить исследование эффективности названного алгоритма.

Вербальный сигнал № 1: $N \approx 5 \cdot 10^8$, D = 00:47:09,840, B = 16, $F_{дискр}$ = 96 кГц



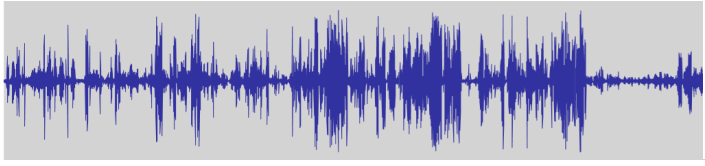
Загрузка CPU:
LoadC = 25 %
Mem = 70 Мб =
= const
Загрузка *GPU*:
LoadG = 98 %
Mem G = 225 Мб

Масштаб $s_x = 1$

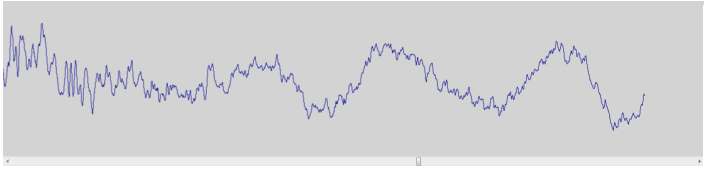


Масштаб $s_x = 2 \cdot 10^4$

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



Загрузка CPU:
LoadC = 25 %
Mem = 70 Мб = const



Загрузка GPU:
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 аудиоматериала; *LoadC* и *LoadG* – загрузка *CPU* и *GPU* при масштабировании и навигации по оси *X*; *Mem* – загрузка ОЗУ; *MemG* – загрузка памяти *GPU*

По результатам исследования производительность процессора *G86* оказалась выше производительности процессора *CPU* только для 32-битных типов данных. При этом производительность *GF114* оказалась в ~100 раз выше процессора *CPU* для чисел с плавающей запятой, при снижении сложности параллельного алгоритма *reduction three* до минимальной. Настоящее исследование показывает возмож-

ность эффективной визуализации данных большого объема и навигации по ним с помощью программного обеспечения на основе *CUDA API* и процессора *GF114*. Использование более слабых видеопроцессоров или применение алгоритма для данных небольшой размерности нецелесообразно, поскольку скорость работы при этом может не превосходить скорость реализации на *CPU*.

ЛИТЕРАТУРА

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

[2] Jobbagy A. Analysis of finger-tapping movement. *Neurosci Methods,* 2005, 141(1), pp. 29 — 39.

[3] Ильин Е.П. *Психомоторная организация человека*: Учебник для вузов. Санкт-Петербург, Питер, 2003, 384 с.

[4] York J.L, Biederman I. Hand movement speed and accuracy in detoxified alcoholics. *Alcohol Clin Exp Res*. 1991, vol. 15, pp. 982 — 990.

[5] Fasel B., Luettin J. Automatic facial expression analysis: a survey. *Pattern Recognition*, 2003, vol. 36, iss. 1, pp. 259 — 275.

[6] Сапожков М.А. *Электроакустика*. Учебник для вузов. Москва, 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*, 1996, pp. 336 — 343.

[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*, Vancouver, Canada, 2008, pp. 3 — 12.

[9] David B. Kirk, Wen-mei W. Hwu. *Programming Massively Parallel Processors: A Hands-on Approach*. San Francisco: Morgan Kaufmann Publishers Inc., 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

[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

Статья поступила в редакцию 24.06.2013

Ссылку на эту статью просим оформлять следующим образом:

Князев Б.А. Алгоритм параллельной агрегации данных для визуализации данных о вербальном и невербальном поведении человека. *Инженерный журнал: наука и инновации*, 2013, вып. 11. URL: http://engjournal.ru/catalog/it/hidden/1064.html

Князев Борис Александрович родился в 1988 г., окончил кафедру «Защита информации» МГТУ им. Н.Э. Баумана в 2011 г. Аспирант кафедры «Системы обработки информации и управление» МГТУ им. Н.Э. Баумана, инженер НИИЦ БТ МГТУ им. Н.Э. Баумана. Автор шести научных работ в области искусственного интеллекта, компьютерного зрения, параллельных вычислений. e-mail: bknyazev@bmstu.ru