Оценка эффективности различных форматов представления разреженных матриц для вычислений на графических ускорителях

Р.М. Куприй[1,2], Б.И. Краснопольский[1], К.А. Жуков[2]

[1]Институт механики МГУ имени М.В. Ломоносова,
[2]Факультет ВМК МГУ имени М.В. Ломоносова

Использование графических ускорителей является распространенным способом ускорения вычислений благодаря их более высокой производительности и пропускной способности памяти по сравнению с центральными процессорами. Они находят достаточно широкое применение для расчета векторных и матрично-векторных операций, являющихся составной частью многих вычислительных приложений, и в частности, итерационных методов решения систем линейных алгебраических уравнений. Эффективное использование графических ускорителей требует адаптации алгоритмов и форматов хранения матриц с учетом особенностей архитектуры ускорителей. В литературе за прошедшее время было предложено множество различных форматов хранения данных, но не все они одинаково эффективны при организации вычислений с матрицами общего вида на современных графических процессорах. В данной работе проводится сравнение ряда популярных форматов хранения разреженных матриц и соответствующих реализаций операции умножения матрицы на вектор. Выбранные форматы (Vector CSR, Adaptive CSR, ELL-T, ELLR-T, Sliced ELL) реализованы в рамках библиотеки численных методов XAMG на языке CUDA. Проведена серия численных экспериментов по оценке эффективности реализованных форматов для набора тестовых матриц из SuiteSparse Matrix Collection и ряда синтетических сгенерированных матриц. Эксперименты проводились как для обособленной операции умножения разреженной матрицы на вектор, так и в составе итерационного метода решения систем линейных алгебраических уравнений. В сравнении с библиотекой cuSPARSE реализованные алгоритмы умножения матрицы на вектор продемонстрировали сопоставимую производительность. Среди реализованных форматов предпочтительным оказался формат Adaptive CSR.

Ключевые слова: формат хранения данных, графические ускорители, умножение разреженной матрицы на вектор, системы линейных алгебраических уравнений.

1. Введение

Решение систем линейных алгебраических уравнений (СЛАУ) является часто встречающейся подзадачей при моделировании различных физических процессов. Этот этап вычислений может занимать значительную часть всего времени расчета. Поэтому важно уделять внимание эффективной реализации и оптимизации этой подзадачи вместе с развитием современных вычислительных систем.

Возникающие в задачах математической физики системы уравнений зачастую имеют большие разреженные матрицы. Характерное число ненулевых элементов в строке матрицы колеблется в диапазоне 10^1–10^2, тогда как размер матрицы может составлять 10^5–10^{10} или более. Для разреженных матриц обычно хранятся только значения ненулевых элементов и информация об их расположении в матрице. Такой способ организации информации называется форматом хранения разреженных матриц.

Для решения систем уравнений широко используются различные итерационные численные методы [1]. Составным элементом этих методов является операция умножения раз-

реженной матрицы на вектор (SpMV):

$$b = Ax.$$

Здесь, $A \in \mathbb{R}^{n \times n}$ – матрица размера $n \cdot n$, $x, b \in \mathbb{R}^n$ – плотные вектора. Операция SpMV, а вследствие и сами итерационные методы, обладает низкой арифметической интенсивностью и относится к классу алгоритмов, чья производительность лимитируется в первую очередь пропускной способностью шины памяти вычислительной системы [2].

В настоящее время для ускорения вычислений приобрели широкое распространение графические ускорители (GPU). Популярность GPU обусловлена на порядки большей производительностью в сравнении с центральными процессорами. Графические ускорители обладают тысячами маломощных вычислительных ядер и высокоскоростной шиной памяти, что позволяет значительно ускорить выполнение алгоритмов, ограниченных не только вычислительными мощностями процессора, но и пропускной способностью шины памяти. Поскольку к последним алгоритмам относится и операция SpMV, то использование GPU может заметно ускорить решение СЛАУ.

Для достижения максимальной эффективности вычислений при реализации алгоритмов необходимо учитывать особенности архитектуры графических ускорителей. Особенно важна организация хранения данных и обеспечение свойств локальности при вычислениях на GPU. Для операции SpMV характер доступа к данным в памяти зависит от формата хранения разреженной матрицы, что оказывает прямое влияние на итоговую производительность вычислений. Существующие форматы различаются, прежде всего, способом представления дополнительной информации о расположении ненулевых элементов в матрице. Многообразие способов представления этой информации порождает большое количество разных форматов хранения, использующих специфические свойства матриц.

Одним из наиболее широко используемых и простых в реализации форматов хранения разреженных матриц является формат Compressed Sparse Row (CSR) [1, 3]. Этот формат применяется в таких известных библиотеках численных методов, как hypre [4] или PETSc [5]. Формат CSR универсален и прост в реализации, но не всегда демонстрирует оптимальную производительность. Поэтому были разработаны многочисленные модификации и вариации формата для повышения эффективности операций с разреженными матрицами. Для центральных процессоров был разработан ряд модификаций, например, Jagged Diagonal [6], Blocked CSR [7] или CSR5 [8]. Однако, эти форматы не учитывают особенности архитектуры ускорителей, поэтому для организации вычислений на GPU они мало применимы.

Другим популярным форматом является ELLPACK (ELL) [9]. Этот формат также прост в реализации, как и CSR, но более предпочтителен при использовании графических ускорителей, поскольку хорошо подходит для потоковой обработки. Он менее универсален в сравнении с CSR и применим только для матриц с ограниченным разбросом числа ненулевых элементов в строках матрицы. Значительное отклонение среднего числа ненулевых элементов в строках матрицы от максимального приводит к неэффективному использованию памяти и снижению производительности операций с матрицами. Для расширения области применимости этого формата и повышения производительности операций с разреженными матрицами при использовании графических ускорителей со временем были разработаны различные модификации, такие как ELLR-T [10], Sliced ELL [11] или SELL-C-σ [12].

Формат Adaptive CSR [13] является примером специализированного формата, разработанного для выполнения вычислений типа умножения матрицы на вектор на графических ускорителях. Этот формат является модификацией формата CSR и ориентирован на увеличение эффективности при работе с матрицами с нерегулярной топологией расположения ненулевых элементов.

Представленное в литературе множество форматов хранения разреженных матриц не позволяет однозначно ответить на вопрос о преимуществах и недостатках того или иного

формата на репрезентативном наборе матриц, и в частности для итерационных методов решения СЛАУ, включая многосеточные методы. В настоящей работе рассматривается реализация перечисленных форматов хранения матриц и операции умножения матрицы на вектор на современных графических ускорителях, и на наборе существенно различных тестовых матриц исследуется их эффективность при выполнении операций умножения матрицы на вектор и при решении СЛАУ. Оставшаяся часть текста организована следующим образом. Во втором разделе статьи представлены реализованные в хода проведенной работы форматы хранения данных и соответствующие алгоритмы умножения матрицы на вектор на GPU. В третьем разделе приведены детали программной реализации алгоритмов и методология проведенных численных экспериментов. Соответствующие результаты оценки производительности описаны в четвертом разделе. В заключении приведено обобщение полученных результатов.

2. Форматы хранения

В следующих параграфах описаны представленные в данной работе форматы хранения разреженных матриц и соответствующие алгоритмы операции SpMV на GPU. Реализации форматов приводятся на примере матрицы, изображенной на рис. 1.

$a_{0,0}$			$a_{0,3}$		
	$a_{1,1}$			$a_{1,4}$	$a_{1,5}$
		$a_{2,2}$			
			$a_{3,3}$		
$a_{4,0}$		$a_{4,2}$		$a_{4,4}$	
		$a_{5,2}$			$a_{5,5}$

$n = 6, nnz = 12$

Рис. 1. Пример разреженной матрицы

2.1. CSR

В формате CSR для представления разреженной матрицы используется три массива, *Val*, *Col* и *Row*. Массив *Val* содержит значения всех ненулевых элементов матрицы, упорядоченных построчно, а в массиве *Col* хранятся номера столбцов соответствующих элементов. Размер этих массивов равен количеству ненулевых элементов, nnz. Массив *Row* размером $n + 1$ содержит смещения от начала матрицы для первых ненулевых элементов в каждой строке, а также общее число ненулевых элементов. Содержимое этих массивов при представлении тестовой матрицы приведено на рис. 2.

Простейший вариант реализации операции SpMV на центральном процессоре для матрицы в формате CSR показан на рис. 3. Базовый алгоритм SpMV для GPU заключается в обработке каждой строки матрицы одним потоком. Такой способ является не эффективным, поскольку не позволяет задействовать ресурсы ускорителя в достаточной мере. Более эффективным является векторный алгоритм SpMV, предложенный в [14]. Он заключается в выделении целого варпа (группы вычислительных нитей) на обработку каждой строки матрицы. Однако, в настоящее время существуют различные одноименные вариации векторного алгоритма. Так, например, для более эффективной работы на современных GPU

Row	0		2			5	6	7			10		12
Col	0	3	1	4	5	2	3	0	2	4	2	5	
Val	$a_{0,0}$	$a_{0,3}$	$a_{1,1}$	$a_{1,4}$	$a_{1,5}$	$a_{2,2}$	$a_{3,3}$	$a_{4,0}$	$a_{4,2}$	$a_{4,4}$	$a_{5,2}$	$a_{5,5}$	

Рис. 2. Пример представления тестовой матрицы в формате CSR

```
for (i = 0; i < n; i++) {
    for (j = Row[i]; j < Row[i+1]; j++)
        y[i] += x[Col[j]] * Val[j];
}
```

Рис. 3. Реализация операции SpMV для формата CSR на CPU

один варп может выделяться для обработки сразу нескольких строк разреженной матрицы. Каждый варп делится на несколько групп последовательно пронумерованных потоков – векторов, каждый из которых обрабатывает свою строку матрицы. Размер варпов фиксирован и составляет 32 нити, в то время как размер вектора является гиперпараметром и определяется в зависимости от степени разреженности матрицы – от среднего числа ненулевых элементов в строке. В такой вариации векторного алгоритма используется два дополнительных параметра: *lane* – номер нити в векторе и TPV – число потоков в одном векторе. Размер вектора является делителем 32 и подбирается таким образом, чтобы наиболее равномерно распределить вычислительную нагрузку между потоками. Такой подход позволяет наиболее полно задействовать вычислительные ресурсы графического ускорителя.

```
int thread_id = blockDim.x * blockIdx.x + threadIdx.x;
int lane = threadIdx.x & (TPV - 1);
int row = thread_id / TPV;
if (row < nrows) {
    double sum = 0.0;
    for (j = Rows[row] + lane; j < Rows[row+1]; j+=TPV)
        sum += Vals[j] * x[Cols[j]];

    #pragma unroll
    for (int i = TPV >> 1; i > 0; i >>= 1)
        sum += __shfl_down_sync(0xffffffff, sum, i, TPV);

    if (lane == 0)
        y[row] += sum;
}
```

Рис. 4. Реализация векторного алгоритма SpMV для формата CSR на GPU

В общем виде код описанного алгоритма умножения матрицы на вектор представлен на рис. 4. Для накопления промежуточных сумм используется разделяемая переменная *sum*, а для получения конечного результата применяется схема редукции с накоплением суммы в одном потоке каждого вектора посредством примитива уровня варпа. У данного алгоритма есть существенный недостаток, который может привести к значительному снижению производительности вычислений: если в разреженной матрице имеется большой разброс числа ненулевых элементов в строках, то часть потоков выполнит обработку строк раньше потоков, обрабатывающих наиболее длинные строки. Такой дисбаланс приводит к простою вычислительных ядер графического ускорителя и, следовательно, к замедлению вычислений.

2.2. Adaptive CSR

Формат Adaptive CSR является модификацией классического формата CSR, которая направлена на балансировку загрузки графического ускорителя при выполнении операций с разреженными матрицами. Для этого последовательные строки матрицы объединяются в группы. Каждая группа строк обрабатывается отдельным блоком потоков по определенному алгоритму. Таким образом, для обработки коротких и длинных строк выделяется разное число потоков, не синхронизирующихся друг с другом. При этом, количество строк в группе зависит от размера общей памяти, выделяемой для каждого блока. Размер выделяемой общей памяти и число потоков в блоке являются гиперпараметрами.

Помимо трех массивов, используемых в формате CSR, в адаптивном формате используется два дополнительных массива: *Row_ blocks* и *Block_ threads*. Размеры этих массивов равны $k + 1$ и k соответственно, где k – число групп строк. В первом массиве хранятся номера первых строк в каждой группе, а последний элемент соответствует числу строк матрицы. Во втором массиве хранится число потоков для обработки одной строки в каждой группе. Пример заполнения массивов данных формата Adaptive CSR для тестовой матрицы приведен на рис. 5.

Row_ blocks	0						3						6
Block_ threads	4						4						
Row	0		2			5	6	7			10		12
Col	0	3	1	4	5	2	3	0	2	4	2	5	
Val	$a_{0,0}$	$a_{0,3}$	$a_{1,1}$	$a_{1,4}$	$a_{1,5}$	$a_{2,2}$	$a_{3,3}$	$a_{4,0}$	$a_{4,2}$	$a_{4,4}$	$a_{5,2}$	$a_{5,5}$	

Рис. 5. Пример хранения матрицы в формате Adaptive CSR

В зависимости от количества выделенных потоков на одну строку, для обработки группы строк используется один из трех алгоритмов: потоковый алгоритм SpMV, векторный алгоритм SpMV или отдельный алгоритм для обработки сверхдлинных строк матрицы. Если в группе больше двух строк, то применяется потоковый алгоритм SpMV: каждый поток выполняет умножение элемента матрицы на элемент вектора и сохраняет результат в общую память, а затем выполняется редукция по строке из общей памяти. Если для обработки строки выделяется один поток, то используется линейная редукция, иначе используется комбинированная редукция.

Если в группе одна или две строки, то для их обработки применяется векторный алгоритм SpMV. Для формата Adaptive CSR используется следующая реализация векторного алгоритма: для обработки строки выделяется целый блок потоков, используется общая память для сохранения вычисленных значений и применяется древовидная редукция по блоку. Обработка строки целым блоком означает, что для этой задачи выделяются все потоки блока. Такой вид алгоритма обусловлен тем, что в такие маленькие группы объединяются достаточно длинные строки.

В том случае, когда число ненулевых элементов в строке превышает размер выделенной общей памяти для одного блока, используется алгоритм обработки строк с редукцией через глобальную память. Для краткости изложения обобщенный код алгоритма не представлен в настоящей работе, с ним можно ознакомиться в оригинальной статье [13].

2.3. ELL

В формате ELL используется всего два массива *Val* и *Col* для представления разреженной матрицы. Ненулевые элементы каждой строки хранятся в виде пар *<значение, номер столбца>*. Все строки дополняются нулями до одинаковой длины, равной максимальному количеству ненулевых элементов в строке, *width*. Размер используемых массивов равен $width \cdot nrows$, где *nrows* – количество строк. Пример представления тестовой матрицы в формате ELL показан на рис. 6.

Col			*Val*			*Row_len*	*Slice_start*
0	3	0	$a_{0,0}$	$a_{0,3}$	0	2	0
1	4	5	$a_{1,1}$	$a_{1,4}$	$a_{1,5}$	3	
2	0	0	$a_{2,2}$	0	0	1	
3	0	0	$a_{3,3}$	0	0	1	9
0	2	4	$a_{4,0}$	$a_{4,2}$	$a_{4,4}$	3	
2	5	0	$a_{5,2}$	$a_{5,5}$	0	2	
							18

Рис. 6. Пример использования массивов *Col*, *Val*, *Row_len* и *Slice_start* для хранения матрицы в форматах ELL, ELLR и Sliced ELL

Как и в случае с базовым алгоритмом SpMV для формата CSR, обработка каждой строки матрицы одним потоком графического ускорителя не эффективна. Формат ELL, для которого реализуется операция SpMV с выделением нескольких потоков для обработки одной строки называется ELL-T. Число потоков T является гиперпараметром и подбирается в зависимости от характеристик матрицы и используемого графического ускорителя. В этом алгоритме используется объединение потоков в группы – вектора, как и в векторном алгоритме SpMV формата CSR. Обобщенный код алгоритма умножения матрицы на вектор в формате ELL-T представлен на рис. 7.

```
int thread_id = blockDim.x * blockIdx.x + threadIdx.x;
int lane = thread_id & (32 - 1);
int row = thread_id / TPV;
double temp_sum = 0.0;
if (row < nrows) {
    for (int j = lane; j < width; j += TPV) {
        int idx = row * width + j;
        temp_sum += Vals[idx] * x[Cols[idx]];
    }

    for (int i = TPV >> 1; i > 0; i >>= 1)
        temp_sum += __shfl_down_sync(0xffffffff, temp_sum, i, TPV);

    if (lane == 0)
        y[row] += temp_sum;
}
```

Рис. 7. Реализация операции SpMV для формата ELL-T на GPU

2.4. ELLR

Формат ELLR является модификацией базового формата ELL. В этой модификации используется вспомогательный массив *Row_len* размера *nrows*, в котором хранится число ненулевых элементов в каждой строке. Пример заполнения массива для тестовой матрицы показан на рис. 6. С использованием этого массива в операции SpMV обрабатываются только ненулевые элементы матрицы, благодаря чему задействованные потоки не совершают лишнюю работу. При этом, для сохранения выровненного и равномерного доступа к элементам массивов *Val* и *Col* дополнение строк нулевыми элементами остается. Такой подход позволяет в некоторых случаях уменьшить конкуренцию потоков за вычислительные ресурсы GPU.

Как и в случае с форматом ELL, формат ELLR и соответствующий алгоритм умножения матрицы на вектор с использованием нескольких потоков для обработки одной строки называется ELLR-T. Ввиду отсутствия принципиальных отличий в реализации алгоритма, обобщенный код опущен для краткости изложения.

2.5. Sliced ELL

Формат Sliced ELL также является модификацией формата ELL. Эта модификация направлена на уменьшение заполнения строк нулевыми элементами по максимальной длине строки ненулевых элементов матрицы. Для этого, матрица делится на полосы заданного одинакового размера, для каждой полосы вычисляется максимальное число ненулевых элементов и организуется заполнение строк в каждой полосе. Для адресации начала каждой полосы используется дополнительный массив *Slice_start*. Он содержит смещения первых ненулевых элементов в каждой полосе от начала матрицы, а также общее число элементов матрицы. Размер массива определяется в зависимости от числа строк матрицы и длины одной полосы. Пример хранения использования вспомогательного массива при хранении разреженной матрицы приведен на рис. 6. Также возможна дальнейшая модификация этого формата путем сортировки строк и выделения полос разного размера для дальнейшего сокращения потребления памяти при заполнении строк нулевыми элементами.

3. Методика тестирования

Результаты, представленные в настоящей работе, были получены на рабочей станции с 8-ядерным процессором Intel Core i7-11700, двухканальной оперативной памятью DDR4 2933 МГц и видеокартой NVIDIA GeForce RTX 3060ti. Фактическая пропускная способность шины памяти для центрального процессора на тестах типа *stream* составляет 43 GB/s, а для видеокарты – 417 GB/s. Для компиляции библиотеки использовалась библиотека OpenMPI версии 4.1.2, компилятор GCC версии 11.4.2 с набором флагов оптимизации `-O3 -ffast-math -funroll-loops -ftree-vectorize -march=rocketlake -mtune=rocketlake -mavx2 -m64` и CUDA SDK версии 12.5.

Все проведенные численные эксперименты на CPU используют все доступные физические ядра и учитывают привязку процессов MPI к ядрам процессора. При проведении расчетов на графическом ускорителе использовался один MPI процесс, взаимодействовавший с GPU.

3.1. Детали реализации

Исследуемые форматы реализованы в библиотеке численных методов XAMG [15, 16]. Эта библиотека предназначена для решения больших разреженных СЛАУ со многими правыми частями, в том числе возникающих в результате дискретизации эллиптических дифференциальных уравнений. В ней реализован набор численных методов, включающий в

себя некоторые итерационные методы подпространства Крылова, алгебраический многосеточный метод и базовые методы типа Якоби или гибридного симметричного Гаусса-Зейделя, которые, в том числе, используются вместе с многосеточным методом в качестве сглаживателей. Библиотека обеспечивает иерархическое трехуровневое распараллеливание с гибридной моделью параллельного программирования MPI + Posix Shared Memory с учетом аппаратных особенностей NUMA-архитектуры.

Библиотека XAMG написана на языке C++ (в соответствии со стандартом C++11) с использованием шаблонов, что позволяет реализовывать форматы хранения матриц с шаблонными типами данных. Часть библиотеки написана на языке CUDA для использования графических ускорителей для проведения вычислений.

3.2. Тестовые матрицы и сценарии тестирования

Для оценки эффективности обсуждаемых форматов хранения данных при выполнении операции SpMV на графическом ускорителе проведено несколько серий численных экспериментов. Проведенные эксперименты включают в себя как оценку времени выделенной операции, умножения матрицы на вектор, так и решение систем линейных алгебраических уравнений. Для получения репрезентативной картины сопоставления различных форматов использовался набор тестовых матриц из коллекции Suite Sparse Matrix Collection (SSMC) [18] и несколько матриц, сгенерированных с помощью генератора матриц библиотеки XAMG. SSMC содержит большой набор матриц из разных предметных областей. Для проведения тестов было взято некоторое подмножество матриц, для которых можно было получить устойчивое решение с помощью выбранного численного метода (стабилизированный метод би-сопряженных градиентов с классическим алгебраическим многосеточным предобуславливателем). При этом время выполнения операции SpMV на GPU с выбранными матрицами варьируется в диапазоне от нескольких микросекунд до нескольких миллисекунд. Более подробное описание набора используемых матриц приведено в [20].

Тестирование эффективности различных форматов было выполнено в три этапа. На первом этапе проведено сравнение времени выполнения операции умножения матрицы на вектор для алгоритма Vector CSR, реализованного в библиотеке XAMG, и времени расчета этой операции в библиотеке NVidia cuSPARSE. На втором этапе проведены замеры времени умножения матрицы на вектор для форматов CSR, Adaptive CSR, ELL-T, ELLR-T и Sliced ELL, реализованных в библиотеке XAMG. На финальном этапе тестирования проведены численные эксперименты по оценке производительности алгоритма решения СЛАУ на центральном процессоре и графическом ускорителе. В этом случае выполнялся расчет фиксированного числа итераций численного метода. Все тесты выполнялись для чисел с плавающей точкой, представленных в двойной точности.

4. Численные эксперименты

4.1. Сравнение производительности операции SpMV в библиотеках XAMG и cuSPARSE

Для подтверждения релевантности представленных далее результатов на первом этапе тестирования получены результаты сопоставления эффективности базовой реализации векторного варианта алгоритма SpMV для формата CSR в библиотеках XAMG и NVidia cuSPARSE. Времена расчета для библиотеки NVidia cuSPARSE, нормированные на времена для библиотеки XAMG, представлены на рис. 8. Значения выше 1 означают более высокую производительность библиотеки XAMG; значения меньше 1 указывают на преимущество библиотеки NVidia cuSPARSE.

За исключением матрицы #14, сравнение результатов показывает превосходство библиотеки XAMG над cuSPARSE в среднем на 13%. Анализ матрицы #14 показал, что она

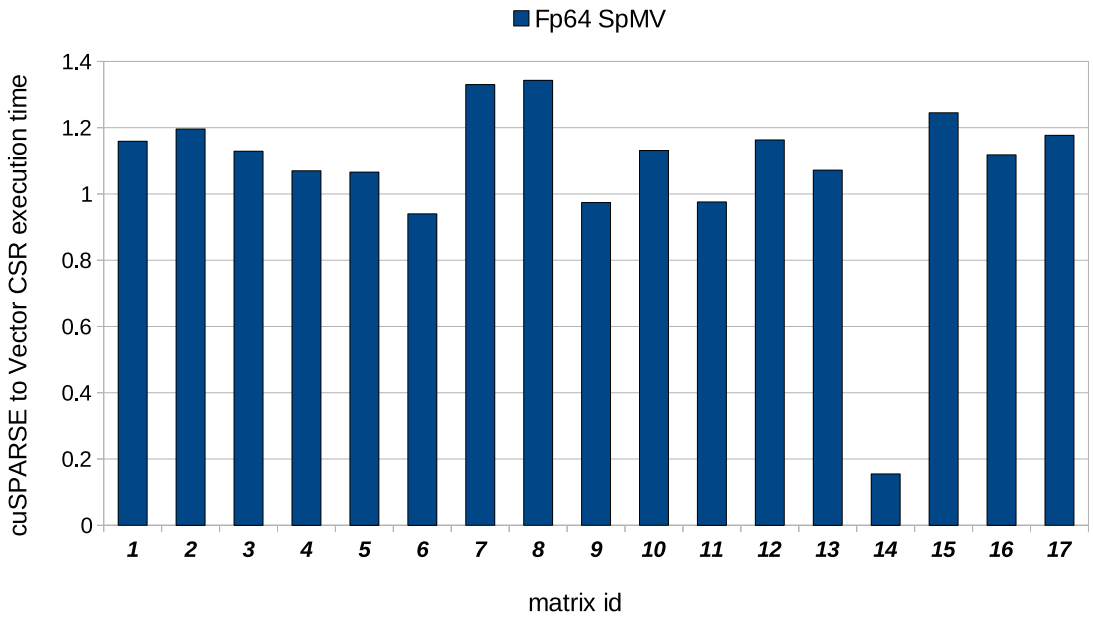

Рис. 8. Отношение времени выполнения операции умножения матрицы на вектор для формата CSR в библиотеках NVidia cuSPARSE и XAMG

обладает группой строк с кратно бо́льшим числом ненулевых элементов по сравнению со средним числом ненулевых элементов в матрице. В базовом варианте формата CSR с векторным алгоритмом SpMV нет балансировки загрузки ресурсов GPU, что и приводит к низкой производительности реализации операции SpMV в библиотеке XAMG для данной матрицы. Полученные результаты тестирования позволяют сделать вывод, что производительность векторного варианта алгоритма SpMV для формата CSR в библиотеке XAMG сопоставима с производительностью алгоритмов библиотеки NVidia cuSPARSE.

4.2. Сравнение производительности операции SpMV для различных форматов хранения разреженных матриц

Для реализованных в библиотеке XAMG форматов хранения матриц CSR, Adaptive CSR, ELL-T, ELLR-T и Sliced ELL проведено тестирование алгоритмов умножения матрицы на вектор. Полученные результаты тестирования представлены на двух графиках на рис. 9 и 10. На первом графике приведено относительное ускорение для формата Adaptive CSR в сравнении с Vector CSR. За исключением матрицы #14, адаптивный формат, в среднем, оказывается медленнее на 2%. При этом, разброс результатов составляет ± 10%. Это объясняется разницей в эффективности подобранных гиперпараметров используемых алгоритмов для каждой конкретной матрицы. Применение формата Adaptive CSR для матрицы #14 позволяет ускорить выполнение операции SpMV более чем в 6 раз. Такой результат достигается за счет рационального использования ресурсов GPU – длинные строки обрабатываются отдельно от коротких выделенным блоком потоков, что позволяет сбалансировать загрузку графического ускорителя.

На втором графике (рис. 10) приведены отношения времен выполнения операции SpMV в формате Vector CSR к ELL-T, ELLR-T и Sliced ELL. В среднем, использование формата ELL-T оказывается на 2% медленнее векторного алгоритма SpMV для формата CSR. Для матриц #10 и #15 использование дополнительной памяти для обеспечения выровненного доступа к элементам массива приводит к значительному замедлению вычислений, что обусловлено большим объемом лишних вычислений при обработке строк матрицы. Для остальных матриц результаты тестов сопоставимы с форматом CSR, поскольку отношение

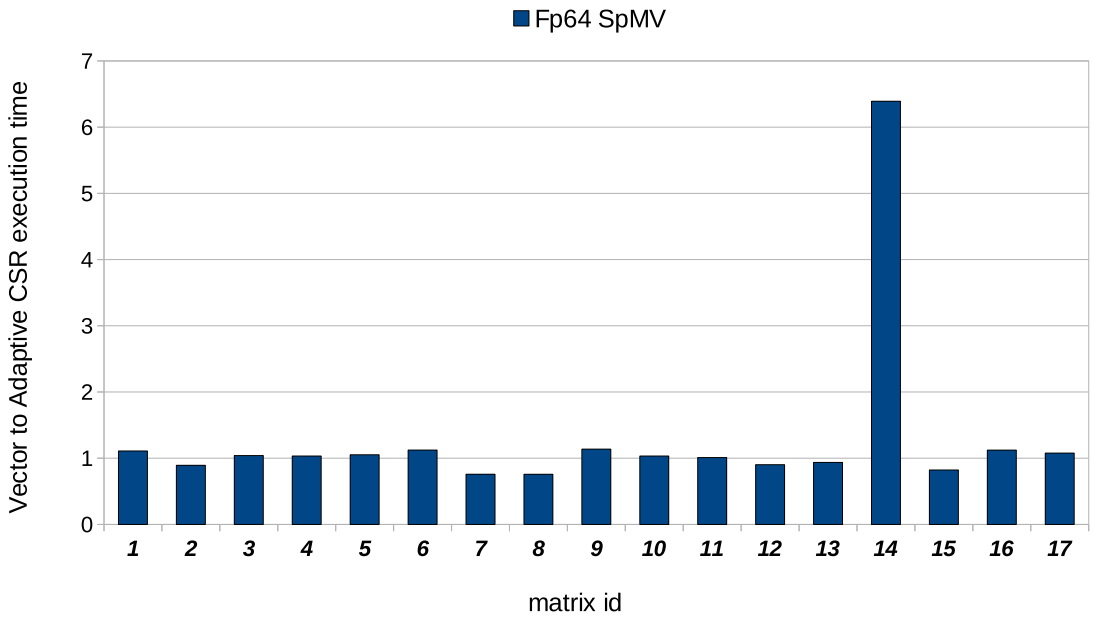

Рис. 9. Отношение времени выполнения операции умножения матрицы на вектор для форматов Vector CSR и Adaptive CSR в библиотеке XAMG

среднего числа ненулевых элементов в строке к максимальному числу ненулевых элементов составляет не менее 0.85.

Для модификации ELLR-T наблюдается замедление в среднем на 3% в сравнении с Vector CSR. При этом, для матриц #10 и #15 использование вспомогательного массива позволяет избавиться от лишних вычислений с нулевыми элементами при обработке строк, поэтому операция SpMV выполняется быстрее в сравнении с оригинальным форматом ELL-T. Однако, для остальных матриц, использование дополнительного массива для устранения лишней работы потоков не столь эффективно, и лишь замедляет вычисления из-за роста накладных расходов на чтение элементов массива *Row_ len* и увеличения числа используемых регистров при выполнении алгоритма на GPU.

Использование формата Sliced ELL приводит к замедлению вычислений в среднем на 15% при выполнении операции SpMV. Для матриц #10 и #15 раздельное заполнение отдельных полос матрицы нулевыми элементами также помогает увеличить эффективность вычислений в сравнении с форматом ELL-T. Однако, в сравнении с форматом CSR все еще наблюдается значительное замедление. Для остальных матриц также наблюдается замедление выполнения операции SpMV в сравнении с CSR. Разделение матриц на полосы и использование вспомогательного массива приводит к усложнению алгоритма SpMV и возникновению накладных расходов на чтение его элементов. Экономия памяти в данном случае не в полной мере компенсирует потери производительности при вычислениях на GPU, что приводит к итоговому замедлению расчета.

Таким образом, на основе проведенной серии тестов можно сделать вывод о том, что использование форматов Adaptive CSR и ELL-T является предпочтительным для рассмотренного набора тестовых матриц. Форматы Adaptive CSR и ELL-T демонстрируют уровень производительности, сопоставимый с Vector CSR, однако при этом обладают большей универсальностью и менее чувствительны к наличию в матрице единичных строк с большим числом ненулевых элементов. Формат ELLR-T является более сложной модификацией формата ELL-T, но при этом на рассмотренном множестве тестовых матриц он не демонстрирует никакого прироста производительности. Наконец, формат Sliced ELL демонстрирует худшие результаты по времени выполнения операции умножения матрицы на вектор среди рассмотренных форматов.

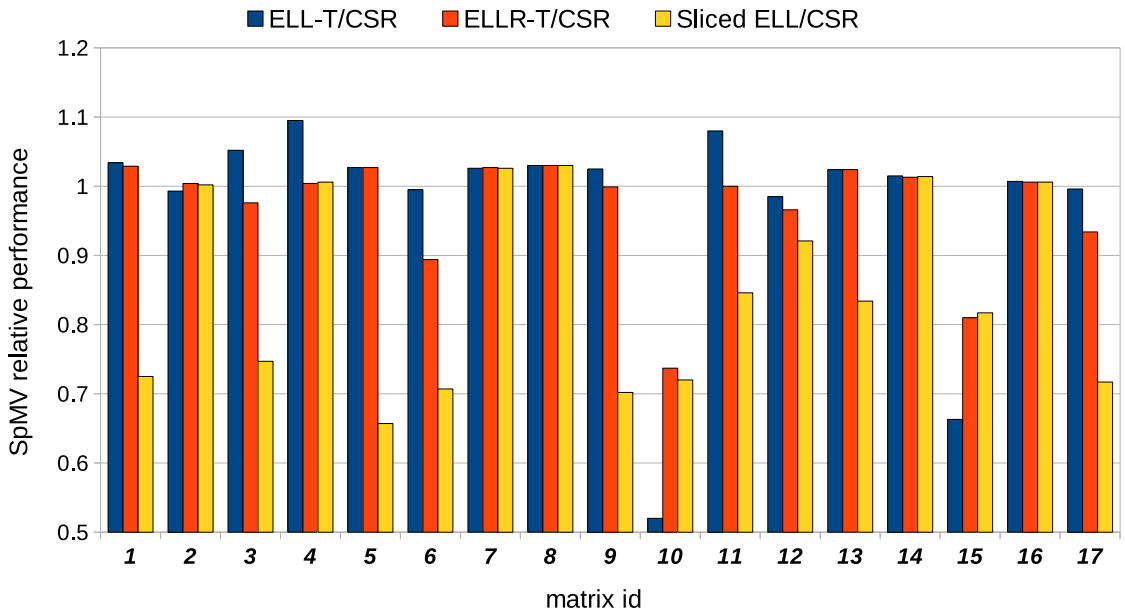

Рис. 10. Результаты сравнения производительности операции SpMV при использовании модификаций формата ELL

4.3. Решение систем линейных алгебраических уравнений

Третья серия экспериментов проведена для оценки эффективности использования формата Adaptive CSR на GPU в сравнении с форматом CSR на CPU при решении СЛАУ. Потенциальное ускорение при переносе вычислений можно оценить с помощью соотношения пропускной способности шины памяти графического ускорителя и центрального процессора. Для используемой вычислительной системы это соотношение составляет порядка 10. Однако, итерационные методы подпространства Крылова и многосеточный метод совмещают множество операций с векторами и операции умножения матрицы на вектор. Для каждой из этих операций реализуется вызов отдельного CUDA-ядра, что сопряжено с определенными накладными расходами. Это приводит к тому, что при проведении вычислений с матрицами нижних уровней многосеточного метода использование графических ускорителей может быть не целесообразным [17].

Полученные в ходе тестирования результаты для набора из 17 матриц приведены на рис. 11. Данные на графике представлены в виде отношения времен вычислений на центральных процессорах и графических ускорителях. Приведенные результаты демонстрируют среднее ускорение расчетов в 5.2 раза, что ниже оценки, основанной на отношении пропускной способности памяти. Это, с одной стороны, демонстрирует возможность существенного ускорения расчетов в библиотеке XAMG при использовании графических ускорителей, а с другой – иллюстрирует необходимость дальнейших оптимизаций кода в части миграции потока вычислений с графического ускорителя на центральный процессор, а также уменьшения числа вызовов CUDA-ядер за счет внедрения технологии слияния ядер (loop fusion) для достижения результатов, близких к теоретическим оценкам. Указанные вопросы будут являться предметом дальнейших исследований.

5. Заключение

В работе рассмотрено несколько форматов хранения разреженных матриц при вычислениях на графических ускорителях: Vector CSR, Adaptive CSR, ELL-T, ELLR-T и Sliced ELL. Рассмотренные форматы и соответствующие алгоритмы для операции умножения матри-

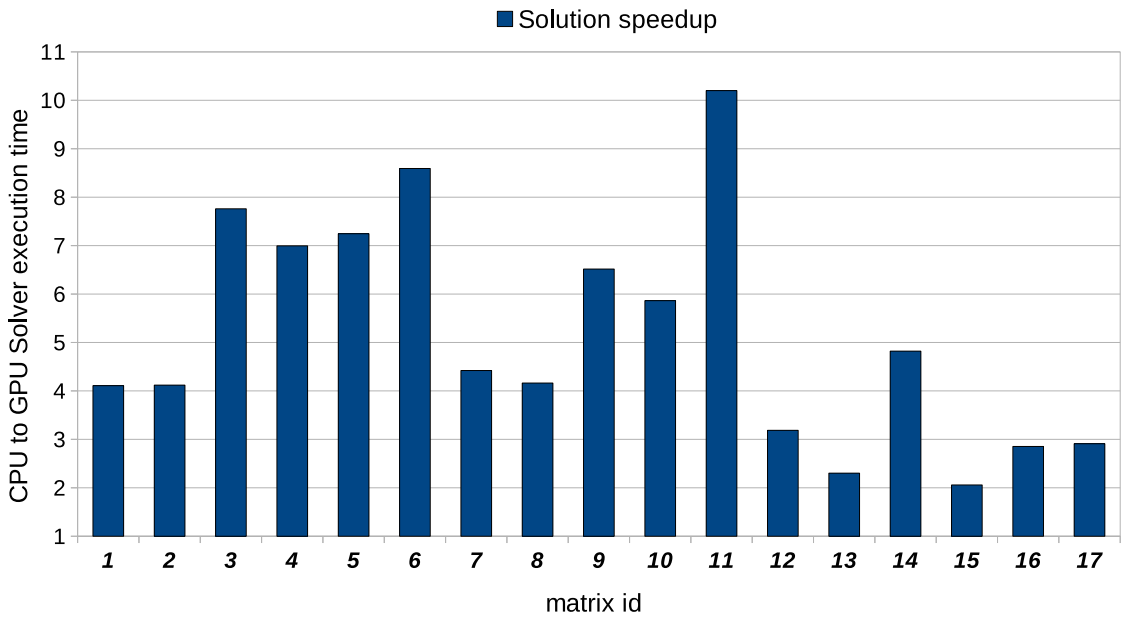

Рис. 11. Ускорение решения СЛАУ при использовании графического ускорителя вместо центрального процессора

цы на вектор реализованы в библиотеке XAMG на языке CUDA. Для набора из 17 матриц проведена оценка времени выполнения операции умножения матрицы на вектор при использовании разных форматов хранения данных и исследовано потенциальное ускорение решения СЛАУ при переходе в расчетах от центральных процессоров к графическим ускорителям.

Проведенные численные эксперименты показали, что среди рассмотренных форматов наиболее эффективным и универсальным является формат Adaptive CSR. Производительность операции SpMV с адаптивным алгоритмом сопоставима с векторным вариантом формата CSR. Главным его преимуществом является возможность эффективно обрабатывать матрицы с существенным дисбалансом числа ненулевых элементов в строках. Формат ELL также позволяет получать на большинстве матриц близкие времена для операции умножения матрицы на вектор, однако в отдельных случаях наблюдается заметное увеличение времени расчета. Рассмотренные модификации формата ELL несколько уступают по производительности исходному формату. Эффективность ELL-подобных форматов сильно зависит от характеристик разреженной матрицы, в частности от соотношения среднего и максимального числа ненулевых элементов в строке.

На примере стабилизированного метода би-сопряженных градиентов с классическим алгебраическим многосеточным методом в качестве предобуславливателя получена оценка потенциального ускорения времени решения систем линейных алгебраических уравнений при переносе вычислений с центральных процессоров на графические ускорители. Наблюдаемое ускорение вычислений при использовании формата Adaptive CSR для хранения разреженных матриц на GPU для текущей версии библиотеки XAMG составляет в среднем 5.2 раза.

Литература

1. Saad Y. Iterative methods for sparse linear systems. SIAM, 2nd edition. Philadelpha, PA. 2003. 520 p.

2. Williams S., Waterman A., Patterson D. Roofline: An insightful visual performance model

for multicore architectures // Communications of the ACM. 2009. Vol. 52, no. 4. P. 65–76. DOI: 10.1145/1498765.1498785.

3. Barrett R. et al. Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods. SIAM, 1994.

4. Falgout R.D., Yang U.M. hypre: A Library of High Performance Preconditioners // Lecture Notes in Computer Science. 2002. Vol. 2331. P. 632–641. DOI: 10.1007/3-540-47789-6_66.

5. Balay S. et al. PETSc Web page. URL: https://petsc.org (дата обращения: 15.01.2025).

6. Anderson E., Saad Y. Solving Sparse Triangular Linear Systems on Parallel Computers // International Journal of High Speed Computing. 1989. Vol. 1. P. 73–95. DOI: 10.1142/S0129053389000056.

7. Vassiliadis S., Cotofana S., Stathis P. Block based compression storage expected performance // Kluwer International Series in Engineering and Computer Science. 2001. Vol. 657. P. 389–406. DOI: 10.1007/978-1-4615-0849-6_26.

8. Liu W., Vinter B. CSR5: An Efficient Storage Format for Cross-Platform Sparse Matrix-Vector Multiplication // Association for Computing Machinery. 2005. P. 339–350. DOI: 10.1145/2751205.275120.

9. Kincaid D., Oppe T., Young D. ITPACKV 2D user's guide // Technical report. 1989. URL: https://www.osti.gov/servlets/purl/7093021 (дата обращения: 15.02.2023).

10. Vázquez F., Ortega G., Fernández J.J., Garzón E.M. Improving the Performance of the Sparse Matrix Vector Product with GPUs // 2010 10th IEEE International Conference on Computer and Information Technology. 2010. P. 1146–1151. DOI: 10.1109/CIT.2010.208.

11. Monakov A., Lokhmotov A., Avetisyan A. Automatically Tuning Sparse Matrix-Vector Multiplication for GPU Architectures // High Performance Embedded Architectures and Compilers. HiPEAC 2010. Lecture Notes in Computer Science. Vol. 5952. P. 111–125. URL: 10.1007/978-3-642-11515-8_10

12. Kreutzer M. et al. A unified sparse matrix data format for efficient general sparse matrix-vector multiplication on modern processors with wide SIMD units // SIAM Journal on Scientific Computing. 2014. Vol. 36, no. 5. P. 401–423. DOI: 10.1137/130930352.

13. Greathouse J.L., Daga M. Efficient Sparse Matrix-Vector Multiplication on GPUs Using the CSR Storage Format // Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. 2014. P. 769–780. DOI: 10.1109/SC.2014.68.

14. Bell N., Garland M. Efficient Sparse Matrix-Vector Multiplication on CUDA // NVIDIA Corporation, technical report. 2008.

15. Krasnopolsky B., Medvedev A. XAMG: A library for solving linear systems with multiple right-hand side vectors // SoftwareX. 2021. Vol. 14. Article 100695. DOI: 10.1016/j.softx.2021.100695.

16. Krasnopolsky B., Medvedev A. XAMG: Source code repository. URL: `https://gitlab.com/xamg/xamg` (дата обращения: 21.01.2025).

17. Krasnopolsky B., Medvedev A. Acceleration of large scale OpenFOAM simulations on distributed systems with multicore CPUs and GPUs // Parallel Computing: On the Road to Exascale (conf. proc.). Vol. 27 of Advances in Parallel Computing. 2016. P. 93–102. DOI: 10.3233/978-1-61499-621-7-93.

18. Davis T.A., Hu Y. The university of Florida sparse matrix collection // ACM Transactions on Mathematical Software. 2011. Vol. 38, no. 1. P. 1–25. DOI: 10.1145/2049662.2049663.

19. NVIDIA cuSPARSE Library. 2025. URL: https://docs.nvidia.com/cuda/cusparse (дата обращения: 21.01.2025).

20. Kuprii R.M., Krasnopolsky B.I., Zhukov K.A. Estimating the Effect of Indices Compression in the CSR-like Data Storage Formats for Matrix-Vector Multiplications and Solving Linear Systems // Lobachevskii J Math. 2023. Vol. 44. P. 3100–3111. DOI: 10.1134/S1995080223080346.