Оптимизация внутреннего представления разреженных матриц для ускорителей NVIDIA

А. Монаков, amonakov@ispras.ru

Институт системного программирования РАН

5 марта 2015 г.

Разреженные матрицы

- Доля ненулевых элементов мала
  $\rightarrow$ Специальные форматы: нужно хранить только позиции и значения ненулевых элементов
- Возникают в численных расчетах и обработке графов
- Основное вычислительное ядро: уможение матрицы на вектор:

$$y = Ax : y_i = \sum_{j:A_{ij} \neq 0} A_{ij} x_j$$

- Только две арифметические операции на элемент
  $\rightarrow$ Ограничены пропускной способностью памяти
- Рассматриваем задачи, в которых расположение элементов не известно заранее
  $\rightarrow$ Иначе лучше использовать подходы без явного хранения матрицы

Задача

Разработать представление для разреженных матриц:

- Оптимизировать операцию домножения на вектор
  $\rightarrow$ Для случаев, когда требуется домножение на транспонированную матрицу, хранить её явно
- Подходящее для слабо структурированных матриц
  $\rightarrow$ Количество элементов в строке может различаться
  $\rightarrow$ Матрица может не быть блочной
- Не требовать переупорядочивания
- Для многократно используемых матриц:
  $\rightarrow$ Возможно нетривиальные преобразования
  $\rightarrow$ Можно использовать автотюнинг

Общие соображения оптимизации SpMV для GPU

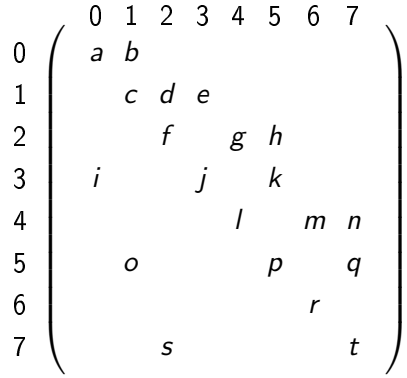
Основное ограничение на GPU: пропускная способность глобальной памяти

- Данные матрицы: streaming-доступ
- Данные вектора множителя: произвольный доступ, но как правило есть локальность locality
- Данные вектора произведения: желательно обеспечить когерентный доступ при записи

Это мотивирует дизайн формата данных:

- Оптимизировать эффективность реализации ядра:
  - Минимизировать синхронизации
  - Максимизировать когерентность обращений в память
- Сократить потребление памяти
  $\rightarrow$ Но методы «компрессии» должны хорошо работать на GPU
- Желательна когерентность при записи вектора произедения

CSR

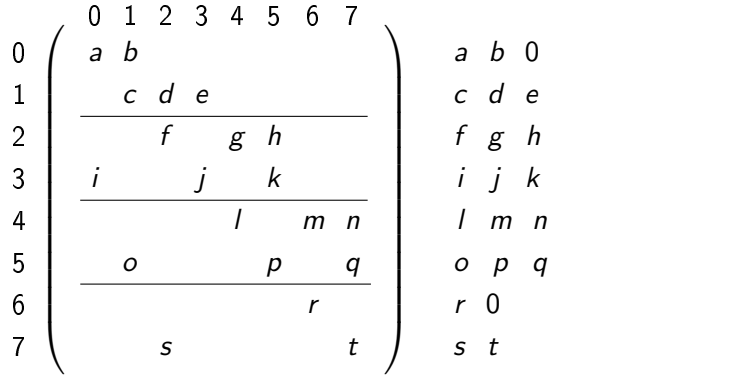


rowptr	0		2			5			8			11			14			17	18		20
value	*a*	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*	*j*	*k*	*l*	*m*	*n*	*o*	*p*	*q*	*r*	*s*	*t*	
column	0	1	1	2	3	2	4	5	0	3	5	4	6	7	1	5	7	6	2	7	

ELLPACK

$$
\begin{array}{c}
 \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7
\end{array}
\begin{array}{c}
\begin{array}{cccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array} \\
\left(\begin{array}{cccccccc}
a & b & & & & & & \\
 & c & d & e & & & & \\
 & & f & & g & h & & \\
i & & & j & & k & & \\
 & & & & l & & m & n \\
 & o & & & & p & & q \\
 & & & & & & r & \\
 & & s & & & & & t
\end{array}\right)
\end{array}
\qquad
\left(\begin{array}{ccc}
a & b & 0 \\
c & d & e \\
f & g & h \\
i & j & k \\
l & m & n \\
o & p & q \\
r & 0 & 0 \\
s & t & 0
\end{array}\right)
\qquad
\left(\begin{array}{ccc}
0 & 1 & * \\
1 & 2 & 3 \\
2 & 4 & 5 \\
0 & 3 & 5 \\
4 & 6 & 7 \\
1 & 5 & 7 \\
6 & * & * \\
2 & 7 & *
\end{array}\right)
$$

Sliced ELLPACK (SELLPACK)



sliceptr	0						6						12						18			22
value	*a*	*c*	*b*	*d*	0	*e*	*f*	*i*	*g*	*j*	*h*	*k*	*l*	*o*	*m*	*p*	*n*	*q*	*r*	*s*	0	*t*
column	0	1	1	2	∗	3	2	0	4	3	5	5	4	1	6	5	7	7	6	2	∗	7

Особенности Sliced ELLPACK

Гибрид между CSR и ELLPACK:

- Совпадает с CSR для slice size $= 1$
- Совпадает с ELLPACK для бесконечного slice size

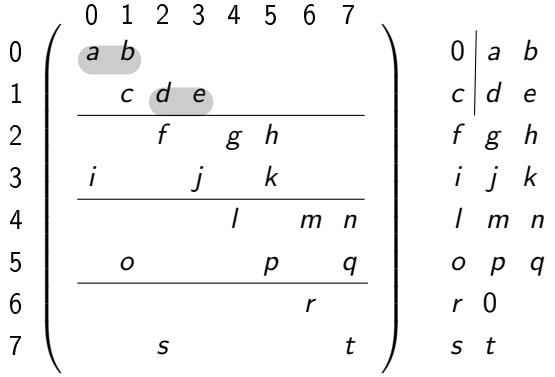
Возможность автотюнинга

- Размер слайса
- Количество нитей/warp'ов на слайс
- Возможны слайсы разных размеров в одной матрице

Возможности улучшения:

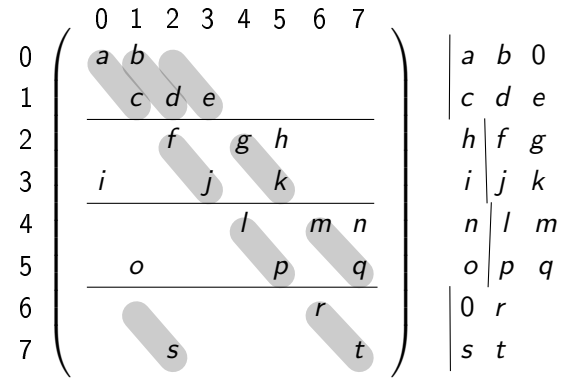
- Компактное хранение для блоков 1×2, 1×4
- Компактное хранение для диагональных последовательностей элементов в слайсе

Sliced ELLPACK with Short Blocks



`sliceptr`	(0, 0)						(6, 4)						(12, 10)						(18, 16)		(22, 20)	
`value`	0	*c*	*a*	*b*	*d*	*e*	*f*	*i*	*g*	*j*	*h*	*k*	*l*	*o*	*m*	*p*	*n*	*q*	*r*	*s*	0	*t*
`column`	∗	1	0		2		2	0	4	3	5	5	4	1	6	5	7	7	6	2	∗	7

Sliced ELLPACK with Diagonals



sliceptr	(0, 0, 0)							(6, 0, 3)							(12, 2, 5)							(18, 4, 7)	(22, 4, 9)
value	*a*	*c*	*b*	*d*	0	*e*	*h*	*i*	*f*	*j*	*g*	*k*	*n*	*o*	*l*	*p*	*m*	*q*	0	*s*	*r*	*t*	
column							5	0					7	1									
diagoffs	0		1		2				0		2				0		2		−5		0		

Особенности усложненных форматов

Компромисс между стоимостью преобразования и потреблением памяти

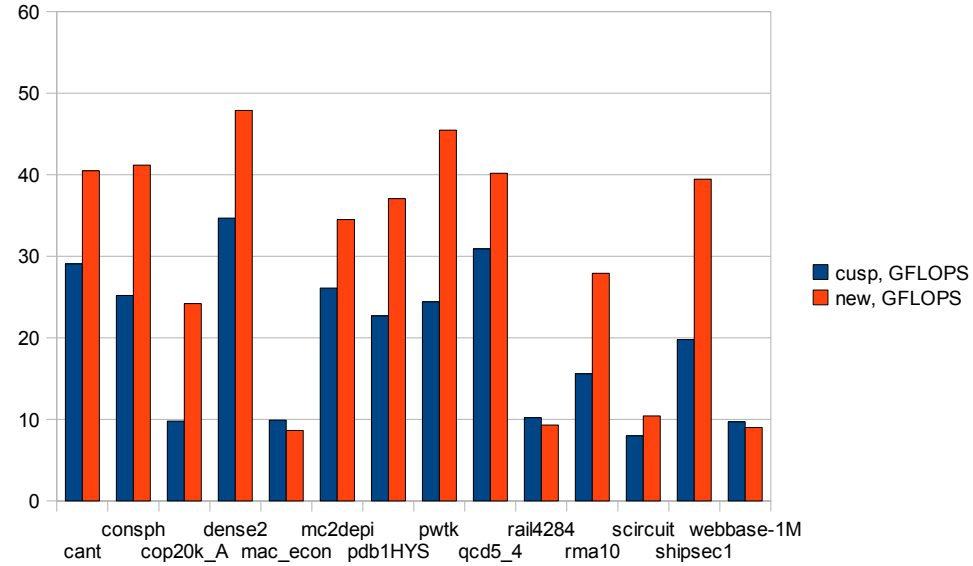
- Нужен дополнительный проход, чтобы выделить блоки/диагонали
- $\approx$ 1 GB/s — приблизительная скорость преобразования
- $\approx$ 0.2 GB/s — для диагонального варианта
- Стоимость амортизируется, если матрица используется множество раз

Работа на этапе автотюнинга увеличивается

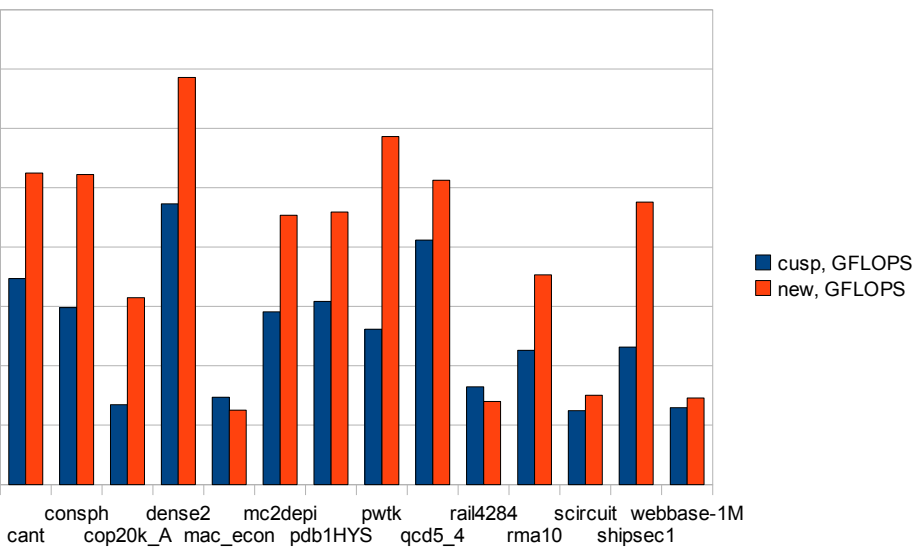
Реализация доступна на https://github.com/amonakov/ispm-sparse-lib

Спасибо!

Производительность: Single Precision (Tesla M2090)



Производительность: Double Precision (Tesla M2090)



Потребление памяти

Testcase	SpMV bytes, CSR	SpMV bytes, Proposed
cant.mtx	32M	24M
consph.mtx	49M	43M
cop20k	22M	22M
dense2.mtx	32M	24M
mc2depi.mtx	23M	13M
pdb1HYS.mtx	35M	33M
pwtk.mtx	95M	76M
qcd5	15M	13M
rail4284.mtx	90M	140M
rma10.mtx	19M	24M
scircuit.mtx	9M	23M
shipsec1.mtx	64M	56M
webbase-1M.mtx	36M	64M