АЛГОРИТМИЧЕСКИЕ ОСОБЕННОСТИ СОЗДАНИЯ МНОГОСЕТОЧНОГО РЕШАТЕЛЯ СЛАУ НА РАСПРЕДЕЛЁННЫХ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМАХ С ГРАФИЧЕСКИМИ УСКОРИТЕЛЯМИ

Б.И. Краснопольский, А.В. Медведев

Введение. Решение систем линейных алгебраических уравнений долгие годы является одной из типичных задач для вычислительных систем. Такая востребованность обусловлена широким спектром задач, сводящихся в конечном счете к решению систем уравнений: например, большинство задач механики сплошных сред при использовании разностных методов и аппроксимации уравнений приводит к большим сильно-разреженным системам линейных алгебраических уравнений. В зависимости от типа исходного дифференциального уравнения и способа его аппроксимации могут использоваться различные численные методы для решения таких систем. К методам наиболее общего вида, активно используемым в настоящее время, можно отнести итерационные методы подпространства Крылова [1] в сочетании с методами неполной факторизации [1] или многосеточными методами [2] в качестве предобуславливателей. При этом многосеточные методы, помимо хорошего потенциала масштабируемости, оказываются предпочтительнее ввиду ряда ключевых особенностей, как то: экономичная асимптотика общего количества операций O(N), где N – размер матрицы системы; независимость скорости сходимости от размера задачи; инвариантность относительно переупорядочения элементов матрицы. Эти аспекты делают многосеточные методы наиболее востребованными при проведении масштабного численного моделирования на многопроцессорных вычислительных системах.

Одной из целей проводимой работы является оценка перспектив применения GPU-ускорителей для решения больших разреженных систем линейных алгебраических уравнений, и в частности для пакета OpenFOAM применительно к задачам судостроения. Актуальность данного вопроса обусловлена потребностью проведения длительных нестационарных расчетов сложных геометрий расчетных областей на подробных расчетных сетках (порядка 100 млн. неизвестных и более). Отсутствие гибридных реализаций методов в OpenFOAM (MPI+OpenMP, MPI+Pthreads и пр.) существенным образом ухудшает потенциал масштабируемости приложения при использовании всех ресурсов современных многоядерных процессоров, приводя к перегрузке адаптеров и коммуникационной сети вычислительных систем. Существенного сокращения времени решения задачи можно добиться путем уменьшения количества MPI-процессов, приходящихся на один узел, и увеличением общего количества узлов, задействованных в расчете, однако такой подход приводит к простою значительной части аппаратных ресурсов. Альтернативным вариантом может явиться использование на вычислительных узлах сопроцессоров (GPU или Intel Xeon Phi), когда даже при использовании одного или двух MPI-процессов в зависимости от количества установленных GPU-устройств, можно получить результаты, превосходящие по эффективности использование множества MPI-процессов на всех ядрах центрального процессора. Подробный обзор проектов, связанных с реализацией многосеточных методов на графических ускорителях был представлен в [3]. Отметим лишь, что ни одна из существующих на данный момент реализаций многосеточных методов не обеспечивает стабильной возможности эффективного использования распределенных вычислительных систем с графическими ускорителями (multiGPU).

Еще одной областью применимости проводимых исследований может стать моделирование задач многофазной фильтрации. Для моделирования неизотермической фильтрации широко используются системы нелинейных уравнений в частных производных, основывающиеся на законах сохранения массы, энергии и законе фильтрации Дарси. Нелинейность системы во многом связана с усложнёнными теплофизическими свойствами флюидов в пластовых условиях. Численное решение задач фильтрации, как правило, проводится в рамках полностью неявных конечно-разностных схем. Итерационный процесс расчета сводится к линеаризации описанной нелинейной системы и последующему решению системы линейных алгебраических уравнений. При этом получаемая матрица линейной системы, как правило, оказывается плохо-обусловленной, и решение такой системы уравнений представляет собой наиболее ресурсоемкую процедуру численного моделирования. Одним из возможных вариантов в данном случае также является использование многосеточных методов, некоторые аспекты применения которых для моделирования задач фильтрации и ускорения вычислений при использовании графических ускорителей обсуждаются, например, в [4].

Многосеточные методы. Алгебраические и геометрические многосеточные методы являются тематикой активных исследований на протяжении нескольких десятков лет. За это время разработано большое количество модификаций методов, ориентированных на различные области применения: решение систем линейных алгебраических уравнений для эллиптических дифференциальных уравнений с гладкими и негладкими функциями, решение систем дифференциальных уравнений, решение задач пластических деформаций материалов и другие. Краткий обзор таких методов приведен, например, в [5]. Для решения эллиптических уравнений из всего многообразия алгебраических методов обычно используется два: классический алгебраический многосеточный метод (classical algebraic multigrid, CAMG) [2] или

многосеточный метод, основанный на агрегировании элементов (smoothed aggregation multigrid, SAMG) [6]. К преимуществам классического многосеточного метода можно отнести наличие большого количества параметров, что обеспечивает робастность метода при решении плохо-обусловленных задач. SAMG оказывается проще как с точки зрения реализации, так и требует существенно меньшего объема вычислений на этапе построения иерархии матриц. В зависимости от вида матрицы, а также наличия или отсутствия необходимости перестраивать иерархию матриц при каждом следующем решении системы уравнений, преимущественным может оказаться использование как одного, так и другого метода. Темой данной статьи является сравнение двух типов многосеточных методов CAMG и SAMG на репрезентативных тестовых задачах, а также оценка перспектив использования этих методов при проведении расчетов на графических ускорителях.

Сравнение многосеточных методов. *Тестовые задачи(Представленные в данном разделе статьи результаты были получены на вычислительной системе «Зилант», построенной на базе процессоров 2x12 cores AMD Opteron 6174, IB DDR interconnect; компилятор — Intel C/C++ Compiler 12.0, библиотека MPI — Intel MPI 4.0.3. При параллельных расчетах на каждом узле задействовалось только одно вычислительное ядро)*. Результаты сравнения рассматриваемых численных методов обсуждаются на примере решения двух систем уравнений. Данные системы уравнений получены из пакета OpenFOAM в ходе расчета течения в ступени погружного насоса; матрицы соответствуют разностной аппроксимации эллиптического уравнения коррекции давления в рабочем колесе и направляющем аппарате. Размеры многоблочных структурированных сеток для указанных деталей составили 2.7 и 4.9 млн. неизвестных соответственно. Общий вид расчетных областей и топология сеток приведены на рис. 1.

Для решения тестовых матриц использовался итерационный метод подпространства Крылова BiCGStab [7] с предобуславливателем. В ходе предобуславливания выполнялась одна итерация V-цикла многосеточного метода. В случае классического многосеточного метода для построения иерархии матриц была использована библиотека hypre [8]. Построение иерархии матриц для SAMG было выполнено по аналогии с методами, реализованными в библиотеке CUSP [9, 10]. В обоих случаях использовался набор параметров методов, обеспечивающих минимальные времена решения систем уравнений. На этапе решения в качестве пре- и пост-сглаживателя применялся итерационный метод Чебышева. Для CAMG выполнялись две итерации, для SAMG приведено сравнение результатов для двух и трех итераций.

Иерархия матриц. В табл. 1 приведены некоторые статистические данные для полученных в ходе построения иерархии матриц многосеточных методов для тестовой задачи размером 2.7 млн. неизвестных; для второй тестовой задачи общие тенденции и закономерности оказываются аналогичными. Одним из главных различий между методами является существенная разница в количестве уровней порождаемой иерархии матриц. CAMG, несмотря на использование агрессивного подхода при выборе элементов матрицы следующего уровня (aggressive coarsening), демонстрирует достаточно слабую скорость уменьшения размера матриц, в среднем в 2.5 раза, по сравнению с SAMG, где таковая оказывается порядка 10. Изменением значений ряда параметров эта ситуация может быть несколько улучшена, однако следствием этого оказывается существенный рост количества итераций и замедление скорости сходимости метода, что приводит к увеличению суммарного времени решения задачи. Общее количество ненулевых элементов во всех матрицах из иерархии для CAMG оказывается вдвое большим по сравнению с SAMG, что влечет использование большего объема оперативной памяти. Матрицы для SAMG обладают несколько большим количеством ненулевых элементов по сравнению с матрицами классического многосеточного метода, однако распределение ненулевых элементов оказывается более компактным, так что в итоге разброс элементов по матрице может быть несколько меньше. Пример фрагментов топологии распределения ненулевых элементов для матрицы 5-го уровня CAMG и матрицы 2-го уровня SAMG приведены на рис. 3. Выбранные матрицы содержат примерно одинаковое количество строк и ненулевых элементов, при этом плотность распределения ненулевых элементов в диагональных блоках для второй матрицы оказывается заметно выше. Данное наблюдение, вероятно, не столь актуально при выборе в качестве базового CRS-формата представления данных, однако может быть полезным в ходе дальнейших исследований по поиску комбинированных форматов с выделением плотных подобластей в разреженных матрицах.

Скорость сходимости, времена решения и масштабируемость решателя. Использованный в тестах набор параметров методов выбирался таким образом чтобы обеспечить близкие к оптимальным времена решения тестовых систем уравнений. При этом, анализ скорости сходимости относительной невязки показал, что рассматриваемые методы обеспечивают сходную скорость сходимости, и разброс для достижения какого-либо порогового значения не превышает нескольких итераций. (Приведенные результаты получены для последовательных запусков методов. При параллельных расчетах используемые методы обеспечивают практически идентичную скорость сходимости, разброс сохраняется в пределах нескольких итераций.). Как и ожидалось, использование трех итераций метода Чебышева в качестве сглаживателя обеспечивает лучшую скорость сходимости по сравнению с двумя итерациями.

Рис. 1. Топология расчетных сеток тестовых задач моделирования течения в ступени погружного насоса

Таблица 1. Иерархия матриц для исследуемых многосеточных методов, матрица 2.7 млн. неизвестных

Уровень	**Размер матрицы**		**Количество ненулевых элементов**		**Среднее количество ненулевых элементов в строке**	
	CAMG	SAMG	CAMG	SAMG	CAMG	SAMG
0	2712320	2712320	18663280	18663280	6,88	6,88
1	1078627	364846	16215587	12471418	15,03	34,18
2	427949	20661	12282755	2250385	28,70	108,92
3	181283	1132	8280097	114926	45,67	101,52
4	75393	107	4617527	7487	61,25	69,97
5	28974	24	2079474	576	71,77	24,00
6	10320	-	757444	-	73,40	-
7	3464	-	230534	-	66,55	-
8	1123	-	60759	-	54,10	-
9	356	-	13578	-	38,14	-
10	109	-	2675	-	24,54	-
11	46	-	798	-	17,35	-

При этом скорость сходимости SAMG с тремя итерациями метода Чебышева совпадает с CAMG. Полученные графики скорости сходимости от количества итераций методов приведены на рис. 2. В дополнение на графиках отмечены два уровня значений относительной невязки 10-6 и 10-8, которые часто используются в CFD-расчетах, в том числе при решении СЛАУ для уравнений коррекции давления.

Основные результаты по временам решения СЛАУ и количеству итераций для трех исследуемых комбинаций методов приведены в табл. 2. Так, время расчета одной итерации метода BiCGStab с классическим алгебраическим многосеточным предобуславливателем оказывается в полтора раза больше по сравнению с SAMG(2), хотя количество операций сглаживания и прочие параметры решателя в данном случае идентичны. Такая разница обусловлена вдвое большим количеством уровней и суммарным количеством ненулевых элементов матриц, порождаемых CAMG (табл. 1). Увеличение количества итераций сглаживания на каждом уровне многосеточного метода с двух до трех приводит к росту времени расчета на 30%, хотя таковое все равно на 20% меньше времени расчета одной итерации CAMG.

Наилучшие результаты по скорости сходимости до достижения двух пороговых значений демонстрируют предобуславливатели CAMG и SAMG(3), при этом разброс не превышает двух итераций в пользу того или иного метода. Более «легковесный» метод SAMG(2) демонстрирует худшую скорость сходимости и затрачивает на 3-5 итераций больше по сравнению с наиболее оптимальным методом. Однако, если сравнить время решения тестовых СЛАУ, то преимущественным оказывается использование именно этого метода. За счет меньшего времени, затрачиваемого на итерацию, более эффективным оказывается расчет нескольких дополнительных, но более экономичных с вычислительной точки зрения итераций. Так, если для расчетной сетки рабочего колеса ступени насоса итоговые времена для SAMG(2) и SAMG(3) оказываются

практически идентичными и на 25% меньшими по сравнению с CAMG, то для сетки направляющего аппарата преимущество SAMG(2) более существенно как по сравнению с CAMG, так и SAMG(3).

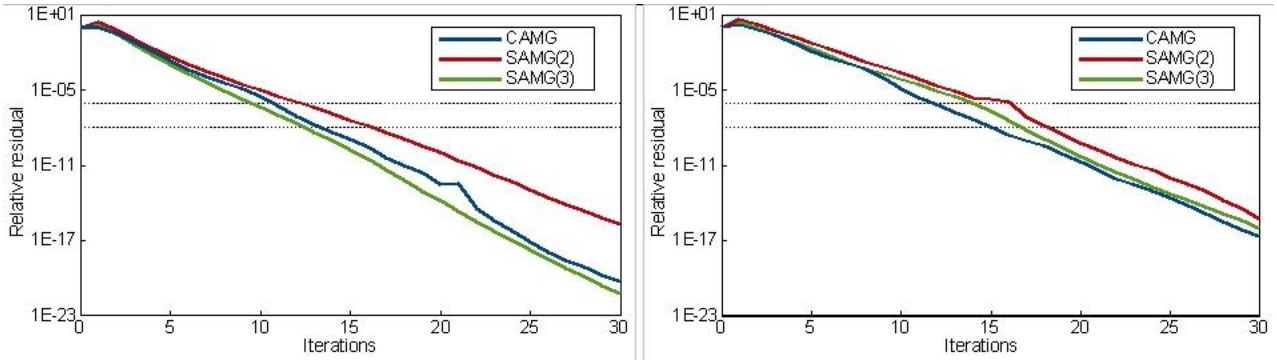

Рис. 2. Графики сходимости относительной нормы невязки. Слева – матрица 2.7 млн. строк; справа – 4.9 млн. строк

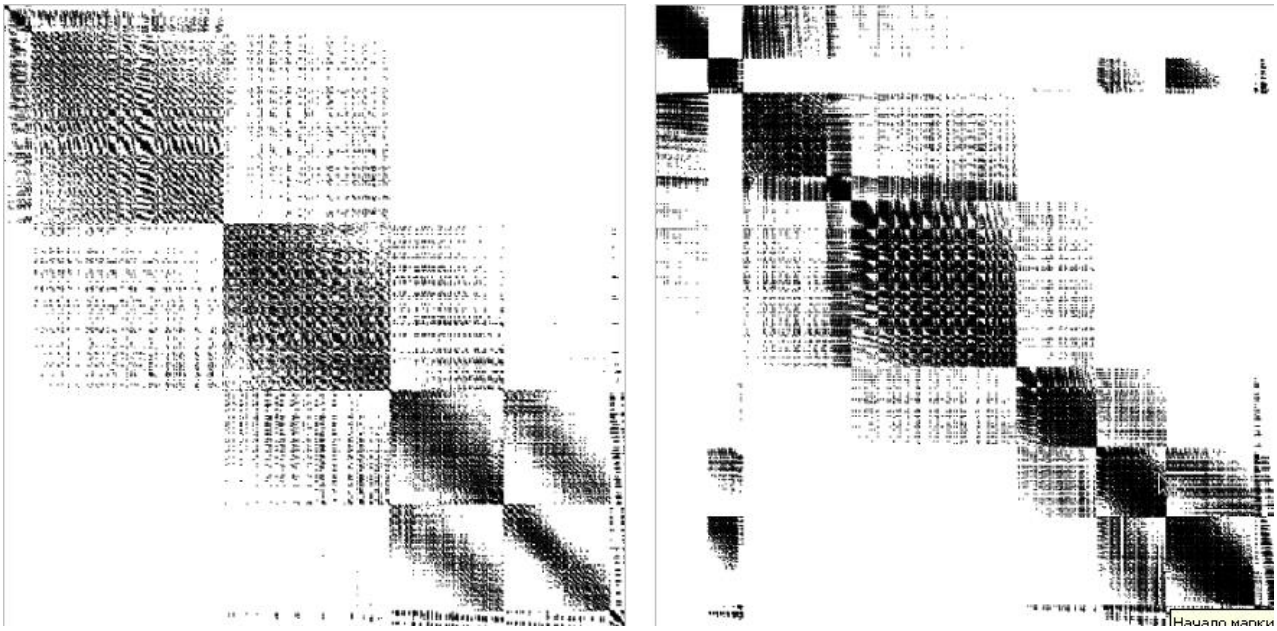

Рис. 3. Фрагменты топологии распределения ненулевых элементов в матрицах иерархии многосеточных методов для тестовой задачи размером 2.7 млн. неизвестных. Слева – CAMG, матрица 5-го уровня; справа – SAMG, матрица 2-го уровня

Проведено исследование масштабируемости трех комбинаций итерационного метода подпространства Крылова с многосеточными предобуславливателями. Полученные результаты зависимости ускорения времени расчета одной итерации метода BiCGStab с многосеточным предобуславливателем для матрицы размером 2.7 млн. строк приведены на рис. 4. В целом графики масштабируемости дублируют друг друга, выделяя незначительное превосходство SAMG-методов по сравнению с CAMG. На 40 узлах вычислительной системы «Зилант» получено ускорение в 34-37 раз, что свидетельствует о хорошей масштабируемости методов. Аналогичный график для второй тестовой задачи повторяет поведение приведенных зависимостей.

Таблица 2. Времена решения и количество итераций до сходимости для тестовых СЛАУ.

Матрица	**Рабочее колесо, 2.7 млн. неизвестных**			**Направляющий аппарат, 4.9 млн. неизвестных**		
Метод	CAMG	SAMG(2)	SAMG(3)	CAMG	SAMG(2)	SAMG(3)
Время расчета 1 итерации, сек	3.45	2.23	2.88	6.16	4.02	5.23

Кол-во итераций (невязка 10-6)	11	13	10	12	17	14
Время решения, сек (невязка 10-6)	181467,0	584454,0	321740,0	510264,0	702253,0	630658,0
Кол-во итераций (невязка 10-8)	14	17	13	16	19	17
Время решения, сек (невязка 10-8)	230958,0	764286,0	418262,0	680352,0	784871,0	765799,0

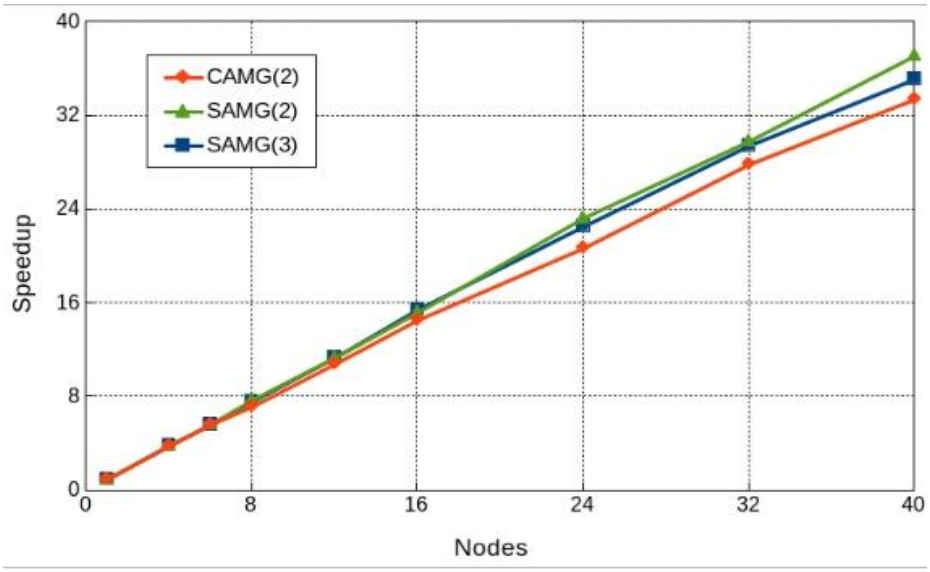

Рис. 4. Ускорение времени расчета одной итерации метода BiCGStab с многосеточным предобуславливателем, матрица 2.7 млн. строк

Особенности реализации многосеточных методов на распределенных системах с графическими ускорителями. *Выбор формата хранения данных.* Используемый формат хранения данных в значительной степени влияет на производительность базовых алгоритмов, входящих в решатель. Наиболее трудоёмкими оказываются операции, выполняемые с участием матриц, таким образом, формат хранения разреженных матриц заслуживает анализа и выбора с точки зрения возможности получения максимума производительности выполнения указанных операций. В [3] был проведен предварительный анализ, показывающий, что скорость выполнения операции умножения матрицы на вектор на графических ускорителях значительным образом зависит от выбора формата хранения матрицы, и выбор оптимального формата неоднозначен. Формат CSR, относящийся к наиболее универсальным и позволяющим легко реализовать оптимальный алгоритм умножения матрицы на вектор на CPU, не всегда обеспечивает такую производительность для этой операции на GPU, которая позволила бы считать её оптимальной. В ряде случаев оказывается, что преимущества таких форматов хранения, как ELLPACK и модификаций на его основе, проявляются несколько ярче, чем разница, наблюдаемая для CPU-реализаций. Остаётся актуальным вопрос о том, как априорно определить, для каких матриц ELLPACK-форматы хранения приведут к лучшим результатам GPU-реализации алгоритма умножения матрицы на вектор. Замечено только, что для матриц, получаемых на верхних уровнях многосеточного метода для рассматриваемых практических задач преимущества ELLPACK-форматов проявляются чаще и отчётливее. Таким образом, напрашиваются два вывода о форматах хранения разреженных матриц:

- использование отличных от CSR форматов хранения матриц для алгоритмов на GPU оправдано, и критерии, при которых целесообразнее хранить матрицы в таких форматах стоят усилий по их разработке;
- гибридные форматы хранения матриц, комбинирующие такие форматы, как ELLPACK и DIA с CSR или COO достаточно перспективны, в первую очередь, именно для GPU-реализаций алгоритмов.

Уменьшение объёмов чтения памяти в базовых алгоритмах. Поскольку все базовые алгоритмы рассматриваемой задачи являются ограниченными пропускной способностью памяти, любое уменьшение объёмов чтения из памяти, оправданное алгоритмически или за счёт особой поддержки со стороны оборудования, должно быть рассмотрено и, по возможности, использовано. Этот вопрос нельзя назвать специфическим по отношению к GPU-реализациям рассматриваемой задачи, но для GPU стоит выделить наиболее важные возможности. Во-первых, возможен переход на одинарную точность представления чисел с плавающей точкой, чтобы сократить расходы на хранение с 64 бит на число до 32 бит. Понижение точности

вычислений в этом случае может оказаться не таким существенным и вполне приемлемым. Эксперименты и исследования показали [11], что применение «смешанной» точности для задач рассматриваемого типа вполне оправдано, выгода может достигать 15-20%. Интересно, что GPU аппаратно поддерживает также арифметику с половинной точностью (half precision). Возможно, что использование такой пониженной точности для некоторых матриц может быть также оправданным, поэтому такой вариант исследования тоже имеет перспективы.

Накладные расходы в гетерогенных системах с графическими ускорителями. Запуск вычислительных ядер на графическом ускорителе и копирование данных из памяти ускорителя и в его память – две операции, которые требуют значительного времени, поскольку включают в себя целый ряд действий, предполагающих накладные расходы, такие как коммуникация с драйвером, планирование выполнения вычислительного ядра, выполнение обмена по каналам шины PCI-Express. Однако, запуск нескольких ядер и получение информации из памяти ускорителя в ходе построения алгоритмов – действия неизбежные, по крайней мере, в рамках архитектуры Fermi. Вычислительное ядро не может самостоятельно контролировать запуск других ядер, значит достаточно сложные алгоритмы могут приводить к возникновению длинных последовательностей вызовов ядер, в то время как каждый вызов ядра и завершение его работы связаны с некоторыми, довольно существенными накладными расходами, причем эти накладные расходы проблематично оценить априорно ввиду отсутствия для этого четких спецификаций. Необходимость разбивать алгоритм выполнения задачи на графическом ускорителе на отдельные вычислительные ядра возникает также из-за отсутствия операций глобальной синхронизации в между нитями, входящими в вычислительное ядро графического ускорителя. Например, задача редукции массива может быть завершена только с завершением работы вычислительного ядра. Иными словами, в архитектуре Fermi нет способа запрограммировать алгоритмы, включающие в себя редукции в сочетании с другими алгоритмами, использующими результат редукции, в виде одного ядра, такие алгоритмы всегда приводят к последовательности запусков ядер.

Приведём пример иллюстрирующий то, в какой степени описываемые накладные расходы могут быть существенными для рассматриваемого класса задач. Обратим внимание на то, каким образом выполняется операция умножения разреженной матрицы на вектор в варианте GPU-кластера. Матрица хранится на узлах распределённой вычислительной системы будучи разбитой по строкам. В соответствии с этим разбиением хранится и вектор-множитель. Для выполнения умножения алгоритму потребуются компоненты вектора-множителя, которые не являются локально хранимыми на данном узле. Узел должен получить эти компоненты с других узлов распределённой системы. Значит, именно по мере получения этих компонент и должна выполняться операция соответствующего частичного произведения матрицы на вектор. Анализ показывает, что, учитывая целесообразность получения нелокальных значений вектора-множителя посредством MPI большими блоками, наиболее экономичным способом организации алгоритма умножения будет разбиение локального фрагмента матрицы данного узла дополнительно на блоки по колонкам, так, чтобы каждый блок соответствовал компоненту вектора-множителя с некоторого соседнего узла (назовём такие блоки сегментами). В общем случае, число таких сегментов может быть равно числу задействованных в расчете узлов, хотя может быть и меньше этого числа, если в результате разбиения появятся сегменты, не содержащие ни одного элемента.

Если число используемых для расчёта одной и той же задачи узлов увеличить вдвое, то с одной стороны, уменьшится объём вычислений, приходящихся на каждый узел, с другой стороны, увеличится (возможно, несколько меньше, чем вдвое) число сегментов, на которые требуется разбить локальный фрагмент матрицы.

Вычисления для каждого из сегментов мы вынуждены выполнять в отдельно запускаемом вычислительном CUDA-ядре, неизбежно увеличивая накладные расходы на запуск этих ядер. То есть, наблюдаются одновремененно два эффекта: уменьшение времени выполнения одного вычислительного ядра и увеличение числа самих вычислительных ядер, которые требуется запустить. Таким образом, удельный вес накладных расходов на запуск ядер окажется тем выше, чем больше узлов используется по двум причинам: во-первых, увеличивается число операций, предполагающих накладные расходы, и во-вторых, общее время вычислений для одного сегмента при этом естественным образом сокращается, и накладные расходы на этом фоне проявляются ярче.

Анализируя такой характер накопления доли накладных расходов, можно предсказать, что масштабируемость описываемой вычислительной схемы должна деградировать при увеличении числа узлов в системе независимо от других эффектов, ограничивающих масштабируемость.

Накладные расходы копирования проявляются, например, в ситуациях, когда ветвления алгоритма зависят от результатов, находящихся в памяти графического ускорителя после очередной операции. Однако, поскольку управление алгоритмом выполняется хост-процессором с помощью запуска вычислительных ядер на ускорителе, то может возникнуть необходимость в синхронном копировании данных из памяти ускорителя всего лишь для принятия решения о ветвлении алгоритма. Типичный пример подобной операции, выполняемой в рамках рассматриваемого типа задач – вычисление нормы невязки для определения момента остановки итерационного процесса.

Средством борьбы с описанной весьма неприятной проблемой накладных расходов запусков вычислительных ядер и синхронного копирования данных из памяти ускорителя может быть появившаяся в последних вариантах архитектуры Kepler поддержка возможности полного управления потоком выполнения непосредственно в контексте графического ускорителя. Не исключено, что в использовании этого подхода возникнет немало технологических трудностей, но эти возможности требуют исследования.

Согласование копирования данных в память устройства с MPI-коммуникациями. Одной из трудностей программирования графических ускорителей можно назвать тот факт, что основные реализации библиотеки MPI не взаимодействуют в явной форме с графическими ускорителями. Для копирования блоков памяти с этих устройств и на эти устройства в случае взаимодействия по данным между узлами необходимо осуществлять по мере необходимости явные вызовы операций копирования с/на устройство. Это создаёт дополнительную сложность программирования алгоритмов при использовании асинхронного шаблона коммуникаций. По-видимому, исследования показывают, что построение эффективной схемы согласования тут возможно в том числе на архитектуре Fermi, если использовать асинхронный режим копирования данных в/из памяти ускорителя и отдельные потоки выполнения CUDA (CUDA streams).

В ходе реализации алгоритма умножения матрицы на вектор на CUDA для архитектуры Fermi учтены все описанные выше соображения, некоторые из полученных при этом графиков масштабируемости показаны на рис. 5. Тестовые расчёты выполнены для нескольких матриц, возникающих на различных уровнях иерархии многосеточного CAMG метода для тестовой задачи «Рабочее колесо», 2.7 млн. неизвестных. Для матрицы четвёртого уровня (L4) с числом ненулевых элементов около 78 тысяч особенно заметен эффект деградации масштабируемости. Время выполнения операции произведения матрицы на вектор для этой матрицы на 12-ти GPU-узлах составляет порядка 400 микросекунд, в процессе вычисления запускается до 10 вычислительных CUDA-ядер. Очень грубо можно оценить порядок накладных расходов, связанных с запуском ядер на GPU и копированием данных как 100-200 микросекунд (более точные оценки не удаётся получить, пользуясь известными источниками и средствами), таким образом упомянутые накладные расходы, по нашим оценкам, могут достигать 25-50% от общего времени выполнения операции для этой матрицы.

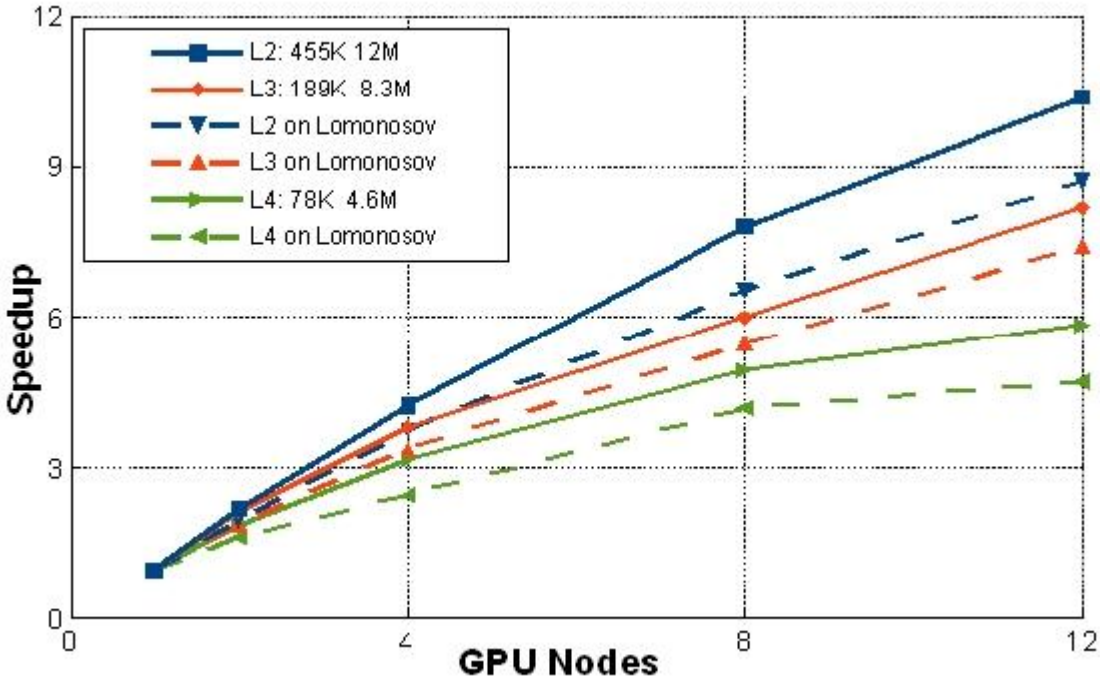

Рис. 5. График масштабируемости операции умножения разреженной матрицы на вектор на вычислительных системах с GPU-ускорителями БелГУ и «Ломоносов» для матриц нескольких уровней. Рабочее колесо, 2.7 млн. неизвестных

Заключение. В статье рассмотрены вопросы применения различных многосеточных методов для ускорения сходимости итерационных методов подпространства Крылова при решении СЛАУ для эллиптических уравнений. Показано, что использование SAMG-методов в качестве предобуславливателей, несмотря на чуть худшие показатели скорости сходимости по сравнению с CAMG, может оказаться предпочтительным за счет существенно меньшего времени расчета каждой итерации. Обсуждается ряд технологических сложностей, возникающих при реализации многосеточных методов на распределенных системах с графическими ускорителями. Показана масштабируемость параллельной реализации одного из базовых алгоритмов метода (умножение разреженной матрицы на вектор), показано, в какой степени накладные расходы, связанные с запуском вычислений и передачей данных на ускоритель ограничивают этот показатель.

Перспективой дальнейшей работы является исследование альтернативных подходов для построения агрегирования элементов во время построения матриц перехода между уровнями, оценка качества получаемых множеств элементов с точки зрения скорости сходимости итерационного процесса решения СЛАУ и разработка распределенной гибридной реализации многосеточного метода.

Требуется также дополнительная работа по поиску путей преодоления технологических сложностей реализации базовых операций метода на GPU-ускорителях. Перспективными направлениями поиска можно назвать исследование альтернативных гибридных форматов хранения матриц, используемых в ходе вычислений, а также исследование некоторых дополнительных возможностей архитектуры Kepler, которые, возможно, снизят накладные расходы при выполнении ряда базовых операций, тем самым улучшив показатели масштабируемости.

Работа частично поддержана грантом РФФИ 12-01-31002-мол-а. Расчеты выполнены на вычислительных системах «Ломоносов» Суперкомпьютерного комплекса МГУ имени М.В. Ломоносова, «Зилант» ЗАО «Т-Сервисы» и кластере Белгородского государственного университета.

ЛИТЕРАТУРА:

1. Y. Saad. Iterative methods for sparse linear systems, 2-nd edition. SIAM, 2003. 528 p.
2. U. Trottenberg, C.W. Oosterlee, A. Schuller. Multigrid. N.Y.: Academic Press, 2001. 631 p.
3. Б.И. Краснопольский, А.В. Медведев. О решении систем линейных алгебраических уравнений на многоядерных вычислительных системах с графическими ускорителями // Параллельные вычислительные технологии (ПаВТ'2013): труды международной научной конференции (1–5 апреля 2013 г., г. Челябинск). Челябинск: Издательский центр ЮУрГУ, 2013. 637 с., с. 409–420.
4. S.A.A. Lonsethagen. Krylov Subspace Accelerated Algebraic Multigrid for Mimetic Finite Differences on GPUs. Master's thesis, Norwegian University of Science and Technology, 2012.
5. R.D. Falgout. An Introduction to Algebraic Multigrid // Computing in Science and Engineering, vol. 8, issue 6, 2006, p. 24-33.
6. P. Vanek, J. Mandel, M. Brezina. Algebraic multigrid based on smoothed aggregation for second and fourth order problems // Computing, 56, 1996, p. 179–196.
7. H.A. Van der Vorst. BI-CGSTAB: A fast and smoothly converging variant of BI-CG for the solution of nonsymmetric linear systems, // SIAM J. Sci. Stat. Comput., vol. 13, No. 2, 1992, p. 631-644.
8. Hypre: a library of high performance preconditioners. https://computation.llnl.gov/casc/linear_solvers/sls_hypre.html (дата обращения: 30.05.2013)
9. Bell N., Garland M. Cusp: Generic Parallel Algorithms for Sparse Matrix and Graph Computations. URL: http://cusp-library.googlecode.com (дата обращения: 30.05.2013).
10. N. Bell, S. Dalton, L. Olson. Exposing Fine-Grained Parallelism in Algebraic Multigrid Methods. NVIDIA Technical Report NVR-2011-002, 2011
11. D. Göddeke, R. Strzodka. Cyclic Reduction Tridiagonal Solvers on GPUs Applied to Mixed-Precision Multigrid // IEEE Trans. Parallel Distrib. Syst. 22(1), 2011, p. 22-32