УДК 004.272.2: 519.63

ФОРМИРОВАНИЕ МАТРИЦ КОНЕЧНО-ЭЛЕМЕНТНЫХ СИСТЕМ В GPGPU[1]

© 2014 г. ***А.К. Новиков, С.П. Копысов, И.М. Кузьмин, Н.С. Недожогин***

Институт механики УрО РАН, Ижевск

sc_work@mail.ru

Поступила в редакцию 28.03.2014

Рассматриваются параллельные алгоритмы формирования матриц конечно-элементных систем, выполняемые на графических процессорах. Обсуждаются особенности реализации предлагаемых алгоритмов в технологии CUDA. Приводятся результаты вычислительных экспериментов, выполненных при моделировании задач механики сплошной среды на гибридных вычислительных узлах.

Ключевые слова: параллельные алгоритмы, численное интегрирование матриц конечных элементов, CUDA, вычисления на графических процессорах.

Введение

Вычисления общего назначения на графических процессорах (GPGPU) позволяют уменьшить время решения в десятки раз [1], но требуют разработки новых параллельных алгоритмов и программ, учитывающих особенности гибридной аппаратной архитектуры и программного обеспечения GPU. В первую очередь как наиболее трудоемкие рассматриваются алгоритмы решения систем линейных алгебраических уравнений [2, 3].

При решении многомерных задач также актуально распараллеливание процесса формирования конечно-элементных систем, который связан с численным интегрированием по объему и поверхности конечных элементов (локальные матрицы жесткости, векторы распределенной нагрузки). Интегрирование глобальных конечно-элементных матриц состоит из двух шагов: интегрирования локальных матриц конечных элементов и их объединения (суммирования) в матрицу коэффициентов системы сеточных уравнений.

Для численного интегрирования локальных матриц и векторов преимущественно применяются квадратурные формулы Гаусса–Лежандра [4], высокая точность которых достигается при небольшом числе точек интегрирования в заданном направлении. В многомерном случае существенно увеличивается число точек интегрирования и, соответственно, вычислительные затраты. Вычисления осуществляются независимо, что делает этот шаг перспективным для распараллеливания, в том числе для GPGPU [5].

Формирование матриц конечно-элементных систем на GPU

В методе конечных элементов процесс численного интегрирования матриц масс, демпфирования и жесткости состоит из интегрирования матрицы каждого конечного элемента и сложения (сборки) полученных матриц. Покажем поэлементную сборку матриц на примере матрицы жесткости

$$\mathbf{K} = \int_V \mathbf{B}^T \mathbf{D}\mathbf{B} dV \approx \sum_{e=1}^{m} \int_{V_e} \mathbf{B}^{eT} \mathbf{D}^e \mathbf{B}^e dV_e = \sum_{e=1}^{m} \mathbf{C}_e{}^T \mathbf{K}^e \mathbf{C}_e, \quad (1)$$

здесь $\mathbf{K}^e \in R^{N_e \times N_e}$ – локальная (элементная) матрица жесткости; $\mathbf{C} = [\mathbf{C}_e]_{1\times m} \in Z^{N_e \times N}$ – матрица связности; m – число конечных элементов; N_e – число степеней свободы в одном конечном элементе; N – число неизвестных в системе уравнений.

Операция сборки матрицы $\mathbf{K} = [k_{ij}]_{N\times N}$ в (1) не обладает такой же степенью параллелизма, как интегрирование локальных матриц. Для уменьшения конфликтов при сборке на GPU рассматриваются следующие варианты упорядочения конечных элементов, узлов сетки и степеней свободы (неизвестных). Первый вариант – создание подмножеств конечных элементов, топологически не связанных в данной расчетной сетке. Второй вариант – упорядочение (смещение нумерации) узлов внутри элемента так, что при одновременном обращении к конечным элементам, имеющим общий узел, ребро или грань, только один из конечных элементов осуществляет доступ к соответствующей строке или элементу k_{ij}. Третий вариант – смещение нумерации неизвестных в узле сетки, например, в трех конечных элементах параллельно выполняется доступ к элементам k_{ij}, k_{i+1j}, k_{i+2j}.

При задании граничных условий первого рода необходимо подправить матрицу коэффициентов. Для этого на GPU передаются номера

этих степеней свободы в конечных элементах, значения граничных условий. В локальных матрицах $\mathbf{K}^e$ элементы строк и столбцов, соответствующие степеням свободы с заданными граничными условиями первого рода, полагаются равными нулю, кроме диагональных элементов матрицы.

Одним из наиболее важных приложений численного интегрирования локальных конечно-элементных матриц является использование конечных элементов высоких порядков. Часто применяются конечные элементы высокого порядка, построенные на основе иерархических базисных функций [4]. В этом случае функции более высокого порядка получаются добавлением слагаемых соответствующей степени, а матрицы жесткости, которые строятся на их основе, включают в себя ранее вычисленные матрицы элементов меньших порядков.

Иерархические конечно-элементные аппроксимации

Иерархические конечные элементы высокого порядка, используемые в работе, строятся на основе полиномов Лежандра. Аппроксимацию искомой функции u (перемещений) запишем следующим образом: $u \approx \sum_{i=0}^{k} \psi_i u_i + \sum_{j=2}^{p} \psi_j a_j$, где ψ_i – линейные базисные функции; ψ_j – иерархические базисные функции; u_i – узловые перемещения; a_j – обобщенные параметры; k – число узлов в элементе; p – порядок аппроксимации.

Запишем трехмерные базисные функции для конечного элемента кубической формы с локальными координатами $-1 \le r,s,t \le 1$. Узловые базисные функции конечного элемента имеют вид $\psi_v = (1+r_v r)(1+s_v s)(1+t_v t)/8$, $v = 1,8$, где v – номер узла в элементе.

С увеличением порядка аппроксимации p на элементе добавляются базисные функции, связанные с обобщенными параметрами на ребрах элемента $\psi_e^p = (1+r_e r)(1+s_e s)P_p(t)/4$, $p \ge 2$, где $P_p(t)$ – проинтегрированный полином Лежандра степени p; e – ребро в элементе. Это соотношение записано для ребра e, параллельного оси t. Для ребер, параллельных осям r или s, функции ψ_e^p получаются перестановкой соответствующих координат.

Базисные функции для граней элемента имеют вид $\psi_f^{mn} = (1+r_f r)P_m(s)P_n(t)/2$, $p \ge 4$, $m,n \ge 2$, $m+n \le p$; здесь f – грань элемента, перпендикулярная оси r. Базисные функции для остальных граней получаются также перестановкой координат.

При $p \ge 6$ в базисных функциях используются внутренние обобщенные параметры. В данной работе использовались аппроксимации до шестого порядка.

Выбор порядка во многом определяет вычислительные затраты. Число точек интегрирования для квадратур Гаусса–Лежандра удовлетворяет неравенству

$$n_G \ge ((3p-2)/2)^d, \qquad (2)$$

где p – максимальная степень полинома в конечном элементе, d – размерность пространства. С точки зрения вычислительных затрат, интегрирование по трёхмерной области по схеме Гаусса–Лежандра требует $O(n_G)$ вычислений значений функций и для элементов высокого порядка становится существенным.

Декомпозиция вычислений для гибридных архитектур

В случае вычислений на гибридных вычислительных системах декомпозиция алгоритмов начинается с распределения вычислений между GPU и центральным процессором (CPU). При интегрировании матриц масс, демпфирования и жесткости рассматриваются следующие варианты распределения вычислений для гибридной вычислительной системы:

1. На GPU численно интегрируется множество матриц, например $\mathbf{K}^e$, $e = 1,2,\ldots,m$. Полученные матрицы передаются в оперативную память для сборки матрицы $\mathbf{K}$ на CPU.

2. Вычисление матриц $\mathbf{K}^e$, задание граничных условий Дирихле на GPU и сборка системы уравнений выполняются на графическом ускорителе. В этом случае передача матриц исключается, а центральный процессор выполняет функцию управляющего процессора, обеспечивает ввод/вывод и постпроцессную обработку результатов.

3. Интеграция вычисления локальных матриц жесткости и задания граничных условий первого рода в вычисление матрично-векторного произведения для методов подпространств Крылова, выполняемых на GPU. Локальная матрица $\mathbf{K}^e$, после учета граничных условий первого рода, умножается на вектор $\mathbf{p}^e \in R^{N_e}$ – соответствующую часть вектора направления $\mathbf{p}$, например, в методе сопряженных градиентов.

Остановимся на распараллеливании процесса формирования локальной матрицы жесткости $\mathbf{K}^e$. Последовательный алгоритм формирования $\mathbf{K}^e$ включает следующие шаги:

– построение функций формы элемента ψ_e;

– переход от глобальной системы координат к локальной;

– вычисление матрицы Якоби $\mathbf{J}$, обратного преобразования $\mathbf{J}^{-1}$ и якобиана $\det \mathbf{J}$;

– вычисление $\mathbf{B}^e = [\mathbf{B}_1^e \mathbf{B}_2^e \ldots \mathbf{B}_{N_e}^e]$;

– численное интегрирование по объёму для вычисления матрицы жесткости

$$\mathbf{K}^e = [\mathbf{K}_{\alpha\beta}^e]_{N_e \times N_e},$$

$$\mathbf{K}_{\alpha\beta}^e \approx \sum_{i=1}^{n_i} \sum_{j=1}^{n_j} \sum_{k=1}^{n_k} \mathbf{B}_\alpha^{e\,T} \mathbf{D}^e \mathbf{B}_\beta^e w_i w_j w_k \det \mathbf{J}, \qquad (3)$$

где $\mathbf{B}_\alpha^e$ – матрица производных ψ_e в точке (x_{1i}, x_{2j}, x_{3k}); $\mathbf{D}^e$ – матрица характеристик материала (среды); $n_l = \sqrt[3]{n_G}$, $l = i, j, k$, а n_G берется из условия (2); w_i, w_j, w_k – весовые коэффициенты квадратур Гаусса.

Выделим уровни распараллеливания приведенного выше последовательного алгоритма: *I уровень* – отдельных конечных элементов одного порядка; *II уровень* – точек интегрирования Гаусса в случае конечных элементов высоких и/или разных порядков. Параллельные алгоритмы, соответствующие уровням распараллеливания, обозначим алгоритм I и алгоритм II (AI и AII).

Кроме того, возможны параллельные вычисления блоков матрицы жесткости $\mathbf{K}^e$ и произведения матриц производных $\mathbf{B}^e$ и упругих постоянных $\mathbf{D}^e$. Особенность построения иерархических аппроксимаций на основе существующих полиномов меньших степеней также можно использовать, вычисляя в параллельных процессах только соответствующие слагаемые при формировании матриц $\mathbf{B}^e$ и $\mathbf{K}^e$.

Особенности программной реализации

Рассматриваемые параллельные алгоритмы численного интегрирования локальных матриц жесткости реализованы в виде библиотеки CUNInt (CUda Numerical Integration), которая интегрирована в пакет конечно-элементного анализа FEStudio [6]. Алгоритмы I, II программно реализованы в виде kernel-функций CUDA.

Учитывая ограничения на число нитей в блоке – N_{tib} – и число блоков в сетке нитей CUDA, при программной реализации алгоритма AI используем двумерную сетку блоков, для варианта AII и элементов высоких порядков достаточно одномерной сетки.

В AI выполняется m CUDA-нитей, каждая из которых вычисляет одну матрицу жесткости $\mathbf{K}^e$ и не имеет общих данных с другими нитями. Каждой нити передаются массивы координат узлов конечных элементов, характеристик материала, порядки аппроксимирующих полиномов.

С повышением порядка полиномов увеличиваются размеры матриц производных функций формы как в локальных, так и в глобальных координатах, которые для шестигранных конечных элементов с тремя неизвестными в узлах при $p = 5$ требуют 21312 байт для хранения матриц $\mathbf{B}_\alpha^{e\,T}$ и $\mathbf{D}^e \mathbf{B}_\beta^{e\,T}$ и 1776 байт – для матрицы локальных производных. Матрица $\mathbf{K}^e$, хранящаяся с учетом ее симметрии, занимает в этом случае 198024 байт памяти и не помещается полностью в регистровую, разделяемую или локальную память графического ускорителя, поэтому всё множество матриц $\mathbf{K}^e$ хранится в глобальной памяти.

Алгоритм II рассматривался в рамках статической модели, с фиксированным числом нитей $m \times n_G$ и суммированием матриц $\mathbf{K}^e$, получаемых нитями в разных точках интегрирования. Как показали эксперименты, использование атомарной операции суммирования медленнее, чем реализация алгоритма I, обладающего меньшей степенью параллелизма.

В программной реализации AII используется разделяемая память ускорителя, а вычисления организуются следующим образом. Блоком нитей CUDA ($N_{tib} = n_G$) параллельно вычисляются локальные матрицы $\mathbf{K}^e$ во всех точках интегрирования одного конечного элемента, каждая нить выполняет вычисления в соответствующей точке интегрирования. Полученное в (3) значение $\mathbf{K}_{\alpha\beta}^e$ помещается в разделяемую память ускорителя, в которой создается массив размерности $\min\{2^k \mid 2^k \geq n_G\}$, $k \in N$. Элементы этого массива суммируются при помощи алгоритма сдваивания, результат операции передается в глобальную память, где хранится $\mathbf{K}_{\alpha\beta}^e$.

Результаты вычислительных экспериментов

Предложенные варианты распараллеливания применялись при решении динамической зада-

Таблица

Время вычисления (с) и ускорение на GPU

3D сетка, $m=10000$, $(N_e \times N_e)$	t_{CPU}				t_{CPU}/t_{GPU}			
	E5430		E5-2609		GTX580		GTX680	
	-O0	-O3	-O0	-O3	I	II	I	II
$p=1$, $n_G=8$, (24×24)	1.23	0.17	1.12	0.14	0.41	0.39	0.74	0.67
$p=2$, $n_G=8$, (60×60)	5.73	0.74	5.20	0.58	1.09	0.90	1.08	1.05
$p=3$, $n_G=64$, (96×96)	108.30	13.30	97.45	10.52	2.68	7.07	1.48	6.41
$p=4$, $n_G=125$, (150×150)	491.10	59.02	441.04	47.52	2.69	8.41	1.41	6.75
$p=5$, $n_G=343$, (222×222)	2867.6	337.83	2598.9	293	2.81	5.82	1.5	5.01

чи теории упругости в плоской и трехмерной постановке (на сетке шестигранных конечных элементов, далее обозначенной как 3D сетка) [7]. Вычислительные эксперименты выполнены на узлах кластера X4 (Институт механики УрО РАН) следующих конфигураций: два CPU Intel Xeon E5430 Quad Core 2.66 GHz, 8 ГБ оперативной памяти и одна видеокарта GeForce GTX 580 (1536 CUDA ядер, 3 ГБ графической памяти); два CPU Intel Xeon E5-2609 Quad Core 2.4 GHz, 32 ГБ оперативной памяти и две видеокарты GeForce GTX 680 (1536 CUDA ядер, 4 ГБ графической памяти в каждой).

В приводимых результатах под ускорением понимается отношение времени t_{CPU} вычисления m матриц жесткости исполняемым кодом с оптимизацией третьего уровня (-O3) на одном ядре центрального процессора к времени вычисления на некоторой конфигурации вычислительной системы: графическом ускорителе (t_{GPU}), двух ускорителях, восьми ядрах CPU ($t_{8\times GPU}$) и их комбинациях. Время вычислений на GPU включает вычисления, копирование данных в память ускорителя и возвращение результата в оперативную память. Результаты получены при вычислениях с двойной точностью.

При интегрировании матриц жесткости для четырехугольных конечных элементов первого порядка ($p=1$) предложенные параллельные алгоритмы обеспечивают ускорение вычислений, начиная с $\lg m \geq 4$, для шестигранных элементов – при $\lg m \geq 5$. В этих случаях вариант AI эффективнее AII в полтора раза в двумерном и в 1.2 раза в трехмерном случае. Предложен вариант AII, в котором блок нитей интегрирует несколько матриц $\mathbf{K}^e$ из (1), а N_{tib} равно числу нитей в warp. Этот вариант обеспечил ускорение в полтора раза относительно AI в двумерном случае и двукратное – в трехмерном случае.

Проведены эксперименты по интегрированию матриц $\mathbf{K}^e$ для сетки с фиксированным числом $m=10000$ при различных порядках аппроксимации $p=1,2,\ldots,5$. При $p\geq3$ (см. таблицу) AII обеспечил ускорение относительно варианта AI до трех раз при вычислениях на видеокарте GeForce GTX 580 и до четырех–пяти раз на GTX 680.

На видеокартах GeForce GTX 680 выполнено сравнение с ускорениями, полученными при распараллеливании средствами технологии OpenMP (см. рис. 1, 2). В ходе экспериментов получено, что восемь нитей OpenMP обеспечили ускорение $t_{CPU}/t_{8\times CPU}$ от семи до восьми раз, а применение одного GPU – только в пять с половиной раз (рис. 2). Параллельное выполнение на двух GPU обеспечивалось вызовом kernel-функции численного интегрирования (алгоритм II) из двух нитей OpenMP, благодаря этому при $p=5$ получено ускорение примерно в полтора раза относительно восьми нитей OpenMP, запущенных на восьми ядрах CPU.

Рассмотрены варианты распараллеливания с распределением вычислений между центральными процессорами и GPU (в варианте «8 нитей OpenMP + AII (1 GPU)» – всего выполнялось девять нитей OpenMP и «8 нитей OpenMP + AII (2 GPU)» – десять нитей, дополнительные нити обслуживали вычисления на графических процессорах). При однородном ($p=5$) порядке аппроксимации достигнуто шестнадцатикратное ускорение, что более чем в два раза выше, чем в варианте восьми нитей OpenMP, запущенных на восьми ядрах CPU, и эквивалентно вычислениям на шестнадцати ядрах центральных процессоров. В этом случае 40% матриц

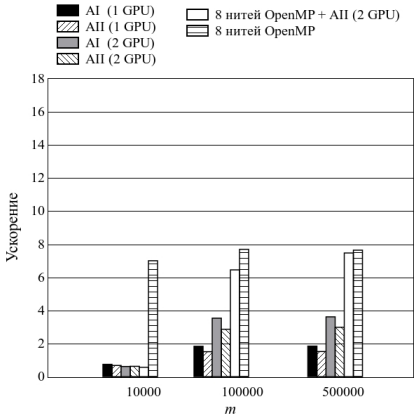

Рис. 1. Ускорение при интегрировании матриц конечных элементов первого порядка

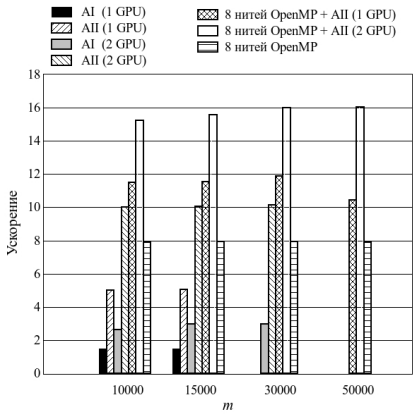

Рис. 2. Ускорение при интегрировании матриц конечных элементов пятого порядка

$\mathbf{K}^e$ интегрировалось на ядрах CPU и 60% – на двух GPU. Разделение вычислительной нагрузки между ядрами CPU и видеокартами также позволило рассматривать варианты, превышающие доступный объем графической памяти. Гибридный подход также успешно конкурирует с вычислениями только на GPU. На рис. 2 показано, что ускорение в варианте «8 нитей OpenMP + AII (1 GPU)» больше, чем при выполнении алгоритма II на двух графических ускорителях. Здесь 70% элементных матриц интегрировалось на ядрах CPU, а остальные – на графическом ускорителе.

В случае неоднородной аппроксимации ($p \in \{1,3\}$) на сетке из $m = 100000$ шестигранных конечных элементов получено тринадцатикратное ускорение при интегрировании $m\big|_{p=1} = 50000$ матриц $\mathbf{K}^e$ на восьми ядрах CPU и $m\big|_{p=3} = 50000$ матриц на двух GPU. Неоднородное распределение порядков, характерное для адаптивных версий метода конечных элементов, с повышением порядка аппроксимирующих функций требует балансировки нагрузки на нескольких уровнях: между центральными процессорами и графическими ускорителями, между ядрами CPU, между ядрами графических ускорителей.

Заключение

Проведенные исследования показали, что в ряде случаев возможности современных компиляторов по оптимизации программного кода и параллельные технологии, применяемые для центральных процессоров, успешно конкурируют с GPGPU, а параллельные алгоритмы должны использовать ядра центральных и графических процессоров.

Эффективное применение графических процессоров в больших математических моделях связано с разделением вычислительной нагрузки между всеми ресурсами гибридной архитектуры с учетом их производительности и требуемой памяти. При интегрировании матриц конечных элементов высоких порядков существенное ускорение достигается, когда на один ускоритель приходится в несколько раз больше конечных элементов, чем на одно ядро CPU. При неоднородном распределении порядков p-/hp-версии метода конечных элементов конечные элементы меньших степеней предпочтительнее интегрировать на ядрах CPU, а больших степеней – на графических процессорах.

Отметим, что массивный параллелизм GPU наиболее эффективно использовался при распараллеливании на уровне точек интегрирования, а предложенные алгоритмы хорошо масштабируются при увеличении числа используемых графических процессоров.

Работа выполнена при финансовой поддержке РФФИ (гранты 13-01-00101-а, 14-01-00055-а, 14-01-31066-мол_а) и Программы Президиума РАН № 18 при поддержке УрО РАН (проект 12-п-1-1005).

Примечание

Статья рекомендована к публикации Программным комитетом Международной научной конференции «Параллельные вычислительные технологии 2014».

Список литературы

1. Morozov D.N., Chetverushkin B.N., Churbanova N.G., Trapeznikova M.A. An Explicit Algorithm for Porous Media Flow Simulation using GPUs // Proc. of the Second Int. Conf. on Parallel, Distributed, Grid and Cloud Comp. for Eng. Stirlingshire. UK, 2011. Paper 19.

2. Губайдуллин Д.А., Никифоров А.И., Садовников Р.В. Об особенностях использования архитектуры гетерогенного кластера для решения задач механики сплошных сред // Вычисл. методы и программирование. 2011. Т. 12. С. 450–460.

3. Kopysov S.P., Kuzmin I.M., Nedozhogin N.S. et al. Hybrid Multi-GPU solver based on Schur complement method // Lecture Notes in Computer Science. 2013. V. 7979. P. 65–79.

4. Szabo B., Duster A., Rank E. The p-version of the Finite Element Method // Encyclopedia of Computational Mechanics. Volume 1: Fundamentals. 2004. P. 119–139.

5. Plaszewski P., Maciol P., Banas K. Finite Element Numerical Integration on GPUs // Lecture Notes in Computer Science. 2010. V. 6067. P. 411–420.

6. Копысов С.П., Красноперов И.В., Рычков В.Н. Объектно-ориентированный метод декомпозиции области // Вычисл. методы и программирование. 2003. Т. 4. № 1. С. 176–193.

7. Копысов С.П., Кузьмин И.М., Тонков Л.Е. Алгоритмическое и программное обеспечение решения задач взаимодействия конструкции с жидкостью/газом на гибридных вычислительных системах // Компьютерные исследования и моделирование. 2013. Т. 5. № 2. С. 153–164.

FORMING MATRICES OF FINITE ELEMENT SYSTEMS IN GPGPU

A.K. Novikov, S.P. Kopysov, I.M. Kuzmin , N.S. Nedozhogin

Parallel algorithms for forming matrices of finite element systems running on GPUs are considered. The features of the proposed algorithms in CUDA technology are discussed. The results of numerical simulation of continuum mechanics problems on hybrid computing nodes are presented.

Keywords: parallel algorithms, numerical integration of finite element matrices, CUDA, computing on GPUs.

References

1. Morozov D.N., Chetverushkin B.N., Churbanova N.G., Trapeznikova M.A. An Explicit Algorithm for Porous Media Flow Simulation using GPUs // Proc. of the Second Int. Conf. on Parallel, Distributed, Grid and Cloud Comp. for Eng. Stirlingshire. UK, 2011. Paper 19.

2. Gubajdullin D.A., Nikiforov A.I., Sadovnikov R.V. Ob osobennostyah ispol'zovaniya arhitektury geterogennogo klastera dlya resheniya zadach mekhaniki sploshnyh sred // Vychisl. metody i programmirovanie. 2011. T. 12. S. 450–460.

3. Kopysov S.P., Kuzmin I.M., Nedozhogin N.S. et al. Hybrid Multi-GPU solver based on Schur complement method // Lecture Notes in Computer Science. 2013. V. 7979. P. 65–79.

4. Szabo B., Duster A., Rank E. The p-version of the Finite Element Method // Encyclopedia of Computational Mechanics. Volume 1: Fundamentals. 2004. P. 119–139.

5. Plaszewski P., Maciol P., Banas K. Finite Element Numerical Integration on GPUs // Lecture Notes in Computer Science. 2010. V. 6067. P. 411–420.

6. Kopysov S.P., Krasnoperov I.V., Rychkov V.N. Ob"ektno-orientirovannyj metod dekompozicii oblasti // Vychisl. metody i programmirovanie. 2003. T. 4. № 1. C. 176–193.

7. Kopysov S.P., Kuz'min I.M., Tonkov L.E. Algoritmicheskoe i programmnoe obespechenie resheniya zadach vzaimodejstviya konstrukcii s zhidkost'yu/gazom na gibridnyh vychislitel'nyh sistemah // Komp'yuternye issledovaniya i modelirovanie. 2013. T. 5. № 2. C. 153–164.