УДК 519.6; 519.67; 519.683 doi 10.26089/NumMet.v19r321

ВОПРОСЫ РАЗРАБОТКИ ПАРАЛЛЕЛЬНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ДЛЯ МЕТОДА ДЕКОМПОЗИЦИИ ОБЛАСТИ

Я. Л. Гурьева[1], **Д. В. Перевозкин**[2]

Рассматриваются различные аспекты разработки параллельного программного обеспечения для метода декомпозиции области: использование технологии MPI-программирования для кластерных систем, точки выбора при проектировании параллельных программ методов декомпозиции области, необходимость реализации действия матрицы без явного ее представления, работа с множествами индексов при программной реализации операторов ограничения и продолжения, а также при обмене данными между подобластями. На ряде численных экспериментов для модельной задачи исследуются вопросы наилучшего выбора конфигурации запуска исполняемой программы на кластере для минимизации времени расчета и предлагается стратегия проведения серии вычислительных экспериментов.

Ключевые слова: метод декомпозиции области, параллельный алгоритм, масштабируемость, структуры данных, вычислительный эксперимент.

1. Введение. Разработка программного обеспечения и программных реализаций вычислительных алгоритмов, в том числе и программ для метода декомпозиции области (МДО), на этапе проектирования связана с необходимостью принятия конкретных решений в целом ряде вопросов. Сделанный выбор определяет как структуры данных, так и саму программную реализацию алгоритма МДО. Зачастую предварительное детальное планирование структур данных и конкретной программной реализации не представляется возможным, однако знание о возможных "подводных камнях" до этапа непосредственного кодирования позволит более детально и эффективно подойти к реализации конкретного алгоритма МДО на некотором языке программирования высокого уровня, например Си или Фортран. Описание ряда таких неявных или неочевидных на первый взгляд вопросов и содержит данная статья.

Кроме того, проведение вычислительного эксперимента требует от пользователя достаточного объема знаний об аппаратной конфигурации кластера. Цель подобного знания — выбрать такие параметры запуска параллельной программы, которые обеспечат минимизацию расхода одного из двух видов доступных ресурсов, а именно или времени, или аппаратных ресурсов кластерной системы. Некоторые вопросы проведения вычислительного эксперимента также обсуждаются ниже на ряде конкретных примеров.

В целом настоящая статья основана на обобщении практического опыта авторов по разработке параллельного программного обеспечения в рамках библиотеки Krylov [1] и решения с его помощью задач МДО. Статья построена следующим образом. В разделе 2 приводится постановка задачи и указываются аспекты МДО, существенно влияющие на создание кода. В разделе 3 содержится описание перехода от исходной матрицы системы линейных алгебраических уравнений (СЛАУ), заданной в явном виде, к понятию "действие" матрицы. В разделе 4 приводится реализация RAS-подхода (RAS — Restricted Additive Schwarz) и даны рекомендации по синхронизации параллельного кода МДО. Раздел 5 посвящен вычислительным экспериментам на разных конфигурациях запусков параллельной программы для одной модельной задачи. В заключении даются рекомендации по проведению вычислительного эксперимента для параллельных программ МДО на современных кластерных системах.

2. Постановка задачи и планирование программной реализации. Пусть задана разреженная СЛАУ большой размерности $N > 10^6$. Считаем, что матрица СЛАУ получена в результате аппроксимации по методу конечных разностей, конечных объемов или конечных элементов некоторой краевой задачи на сетке с числом узлов порядка нескольких миллионов. Матрица является разреженной, так как в строке матрицы, с количеством элементов равным N, ненулевыми являются всего несколько элементов $n \approx 10$, количество которых зависит от аппроксимационного шаблона. Для хранения и представления в памяти компьютера подобных матриц обычно используется некий специальный формат хранения, когда реально

[1] Институт вычислительной математики и математической геофизики СО РАН (ИВМиМГ СО РАН), просп. Лаврентьева 6, 6630090, Новосибирск; ст. науч. сотр. e-mail: yana@lapasrv.sscc.ru

[2] Институт вычислительной математики и математической геофизики СО РАН (ИВМиМГ СО РАН), просп. Лаврентьева 6, 6630090, Новосибирск; мл. науч. сотр., e-mail: dperevozkin@mail.ru

хранятся только позиции ненулевых элементов и их значения. Например, часто используется разреженный строчный формат [2].

Пусть требуется создать параллельную программную реализацию для решения заданной СЛАУ на основе двухуровневого итерационного метода декомпозиции области. Внешние итерации ведутся по некоторому известному итерационному алгоритму, выбор которого зависит от свойств матрицы СЛАУ. Универсальным и часто используемым является алгоритм FGMRES (Flexible Generalized Minimal Residual) [3]. На каждой внешней итерации требуется параллельно решать подсистемы меньших размерностей во всех подобластях. Для более подробного математического описания постановки задачи мы отсылаем читателя к статье [4].

Будем считать, что параллельный код создается для некоторой кластерной системы, состоящей из множества в общем случае неоднородных узлов. Кроме того, считаем, что для обеспечения работы параллельной программы на кластере имеется некоторое программное обеспечение стандарта MPI (Message Passing Interface) [5], которое для модели обмена сообщениями является механизмом построения параллельных программ. Основным понятием в MPI-подходе является понятие процесса, а само программное обеспечение MPI является библиотекой функций, предназначенной для поддержки работы параллельных процессов в терминах передачи сообщений [6]. Поэтому в нашем подходе после декомпозиции области на подобласти с примерно равным числом неизвестных соответствующая подобласти подсистема линейных уравнений назначается некоторому “своему” MPI-процессу. Общее количество MPI-процессов, на которых запускается параллельная программа, не меньше количества подобластей, что позволяет в дальнейшем использовать термины “подобласть” и “процесс” как синонимы. Привязка подобласти к MPI-процессу на программном уровне означает передачу данных о подсистеме в процесс, дальнейшую обработку этих данных (например, решение этой подсистемы уравнений), некоторый обмен данными с соседними подобластями, т.е. процессами, сборку глобального вектора искомого решения из подвекторов разных MPI-процессов.

Для того чтобы показать многообразие вопросов, которые необходимо принять во внимание разработчикам параллельного программного обеспечения МДО на этапе планирования программной реализации и самого процесса кодирования, перечислим лишь некоторые из них:

— разделение вычислительной области на подобласти может быть проведено на геометрическом или алгебраическом уровнях; последний требует применения сведений из теории графов для достижения сбалансированного разбиения с целью обеспечения как можно более равномерной загрузки MPI-процессов;

— чем является сам разделитель? Он может содержать узлы с неизвестными, т.е. являться отдельной самостоятельной подобластью, а может не содержать узлов вообще и всего лишь делить исходную геометрию расчетной области;

— как влияют на выбор разделителя аппроксимационный шаблон и расположение неизвестных по отношению к сеточным сущностям, таким как сеточные узлы, сеточные ребра, сеточные грани, сеточные объемы?

— как определить и построить разделитель, чтобы, во-первых, получить как можно более сбалансированные по числу неизвестных подобласти, во-вторых, обеспечить как можно более быструю сходимость внешнего итерационного процесса (это означает, что передача информации между подобластями осуществляется быстро), в-третьих, обмен информацией между соседними подобластями осуществлялся эффективно?

— как можно осуществить обращение матрицы в отдельной подобласти? Что именно понимается под предобусловливателем Якоби в подобласти? Двумя возможными интерпретациями при программировании этого аспекта являются или зануление связей с соседними подобластями и обращение подматрицы в подобласти, или учет связей с соседними подобластями по типу условий Дирихле (т.е. изменение правых частей в уравнениях для узлов, имеющих соседей из других подобластей, зануление соответствующих коэффициентов в подматрице подобласти и, наконец, обращение полученной подсистемы);

— когда и какой информацией следует обмениваться в процессе итерационного решения? Как подготовить буферы для обмена? Можно ли сократить общее число обменов? Какие действия из процесса итерационного решения можно вести на фоне обменов (совмещать счет и обмены информацией)?

— что такое подматрица, соответствующая некоторой подобласти? Она может быть задана явно, например, в некотором известном матричном формате, а может вообще не быть заданной в указанном выше виде, и взамен может быть известно только лишь ее действие на некоторый исходный вектор;

— как разделить глобальные переменные по подобластям? Как осуществить переход к локальной нумерации в каждой подобласти? Как, имея части некоторого глобального вектора распределенными по подобластям в виде локальных векторов, осуществить "сборку" этого глобального вектора?

— как обеспечить синхронизацию вычислений в параллельной программе МДО?

— какое представление матрицы/подматриц использовать и как представить операции с подматрицами, чтобы учесть особенности вычислительных способностей современных процессоров? Как эффективно использовать возможности систем команд AVX/AVX2 (Advanced Vector Extensions)?

Необходимо отметить, что указанными аспектами не ограничивается весь круг вопросов, которые приходится решать разработчику на этапе кодирования МДО. Часть этих вопросов с точки зрения вычислительной алгебры и вычислительной геометрии была упомянута в статье авторов [4]. Остановимся далее на некоторых наиболее интересных с точки зрения кодирования МДО перечисленных выше аспектах подробнее.

3. Действие матрицы вместо явного представления. Будем считать, что исходной задачей является решение СЛАУ с заданной матрицей и правой частью:

$$Au = f, \quad A = \{a_{i,j}\} \in \mathcal{R}^{N,N}, \quad u = \{u_i\}, \quad f = \{f_i\} \in \mathcal{R}^N. \tag{1}$$

Матрица является результатом аппроксимации некоторой двумерной или трехмерной краевой задачи в некоторой расчетной области Ω. Считаем, что матрица является разреженной, т.е. в каждой строке матрицы число ненулевых элементов nnz гораздо меньше размерности системы: $nnz \ll N$. Матрица в коде представляется в некотором разреженном формате хранения, например в разреженном строчном формате CSR (Compressed Sparse Row) [7].

Первым этапом в подходе МДО является разбиение исходной сущности на части, в нашем случае — разбиение расчетной области на подобласти. На уровне матрицы этому соответствуют два момента:

1) разбиение матрицы на подматрицы, соответствующие подобластям;

2) возможная модификация подматриц в зависимости от связей соответствующей подобласти с соседними подобластями.

Остановимся на этом подробнее. Пусть в некоторой подобласти Ω_l, содержащей узел сетки, которому соответствует неизвестная с номером i, уравнение для этой неизвестной записывается в виде

$$a_{i,i}u_i + \sum_{j \neq i} a_{i,j}u_j = f_i.$$

Здесь $a_{i,i}$ — диагональный элемент в строке с номером i матрицы A, $a_{i,j}$ — внедиагональные элементы этой строки, отражающие связь узла с соседними узлами в соответствии с аппроксимационным шаблоном. Выделим в этой сумме две части: одну, соответствующую соседним узлам в "своей" подобласти Ω_l, и другую, соответствующую узлам из соседних подобластей. Запишем это следующим образом:

$$a_{i,i}u_i + \sum_{\substack{j \neq i; \\ i,j \in \Omega_l}} a_{i,j}u_j + \sum_{\substack{k \neq i; \\ k \notin \Omega_l}} a_{i,k}u_k = f_i. \tag{2}$$

Предположим, что значения неизвестных u_k в соседних для Ω_l подобластях уже вычислены. Тогда можно переписать (2) следующим образом:

$$a_{i,i}u_i + \sum_{\substack{j \neq i; \\ i,j \in \Omega_l}} a_{i,j}u_j = f_i - \sum_{\substack{k \neq i; \\ k \notin \Omega_l}} a_{i,k}u_k. \tag{3}$$

Если в этом уравнении перейти от одного узла к вектору, длина которого равна числу неизвестных в подобласти, от одного диагонального коэффициента перейти к подматрице A_l, отвечающей за связи узлов только в данной подобласти (а не с узлами из соседних подобластей), предположить, что данная подматрица обратима, домножить полученное уравнение на обратную подматрицу, подставить верхние

индексы n в левую часть уравнения, а $n-1$ — в правую часть, то получим некоторый итерационный процесс вычисления решения в данной подобласти. Его можно записать в виде

$$u^n = g - Tu^{n-1},$$

или на операторном уровне

$$(I+T)u = g \equiv A_l^{-1} f_l,$$

где I — тождественный оператор, T — так называемый оператор перехода. Далее, именно для реализации этого итерационного процесса и применяется один из итерационных методов в подпространствах Крылова.

Здесь ключевым моментом является то, что от матрицы системы A в области и подматрицы A_l в подобласти мы перешли к действию, выражаемому оператором $I+T$, соответствующая матрица которого неизвестна в том смысле, что у нас нет ее представления в виде портрета и элементов в отличие от исходной матрицы A с известными портретом и элементами, хранимыми в формате CSR. В данном случае можно сказать, что известно "действие" оператора $I+T$ в том смысле, что ясен способ его программной реализации. Этот момент является очень важным, так как при применении итерационных методов, основанных на подпространствах Крылова, одной из основных вычислительных операций является операция умножения матрицы на вектор, которое в нашем случае будет требовать бóльших усилий при программной реализации, чем простая операция умножения на известную матрицу. В нашем случае на уровне программной реализации действию оператора T на подвектор из некоторой подобласти согласно формулам (2)–(3) будет соответствовать следующий набор алгоритмических действий:

— зануление правой части подсистемы;

— получение аналогичных подвекторов из соседних к данной подобласти подобластей (строго говоря, нас интересуют только те значения подвекторов, с узлами которых связаны приграничные узлы рассматриваемой подобласти);

— модификация правой части подсистемы: добавление в нулевую правую часть величин $\sum\limits_{\substack{k\neq i;\\ k\notin\Omega_l}} a_{i,k}u_k$;

— решение подсистемы СЛАУ с известной подматрицей и насчитанной правой частью прямым или некоторым итерационным методом.

В итоге полученный подвектор решения и будет искомым действием оператора на заданный исходный подвектор вектора решения.

4. Реализация RAS-подхода и синхронизация в параллельной программе МДО. Как правило, обязательным этапом решения СЛАУ с помощью методов алгебраической декомпозиции областей является построение разбиения исходной сеточной области на подобласти и вычисление сопутствующих данных. К этим данным относится и информация, необходимая для организации обмена между различными процессами. Обычно каждому процессу соответствует некоторый набор неизвестных и/или строк СЛАУ, причем в общем случае этот набор не является последовательным (например, в смысле перечисления номеров неизвестных). Кроме того, иногда возникает необходимость указать множества смежных с подобластью узлов, или расширенные подобласти. Здесь представляется уместным ввести следующие понятия.

Множеством индексов назовем конечное множество натуральных чисел $I = \{i_1, i_2, \ldots, i_k\}$. Определим также множество $L(I)$ как линейную оболочку векторов $e_{i_1}, e_{i_2}, \ldots, e_{i_k}$ стандартного базиса, т.е.

$$L(I) = \mathrm{Span}\{e_i \mid i \in I\}.$$

Иначе говоря, $L(I)$ можно представить как множество сеточных функций, имеющих I своим носителем.

Далее рассмотрим часто применяемый ограничительный метод Шварца (Restricted Additive Schwarz, RAS; самое раннее упоминание метода в литературе относится к 1998 г. [8], годом позже вышла еще одна статья тех же авторов [9]). Отличительной особенностью этого метода является то, что после решения вспомогательных СЛАУ в расширенных подобластях решение в узлах перехлеста (т.е. в узлах, принадлежащих сразу нескольким подобластям) вычисляется не путем суммирования решений в подобластях с некоторыми весами, а путем фиксированного выбора решения из той или иной подобласти.

На уровне программной реализации номера элементов векторов u, f и номера строк матрицы A из системы (1) соответствуют некоторому множеству индексов $V = \{1, 2, \ldots, N\}$. Следуя [9], разобьем множество V на непересекающиеся подмножества (подобласти) S_i, $i = 1, \ldots, p$, где p — количество подобластей,

таким образом, чтобы покрыть без пересечений все V:

$$\bigcup_{i=1}^{p} S_i = V, \quad S_i \bigcap S_j = \emptyset, \quad i \neq j.$$

Рассмотрим расширенные подобласти S_i^δ, единственное требование к которым, вообще говоря, заключается в том, чтобы $S_i^\delta \subset S_i$ для всех $i = 1, \ldots, p$. Определим для каждой из подобластей S_i, S_i^δ, $i = 1, 2, \ldots p$, операторы ограничения R_i, R_i^δ (см. [9]) следующим образом: пусть $u \in L(V)$, тогда $R_i^\delta u = u\big|_{S_i^\delta}$, $R_i u = u\big|_{S_i}$, и оператор продолжения P_i:

$$(P_i v)_j = \begin{cases} 0, & j \notin S_i, \\ v_j, & j \in S_i. \end{cases}$$

Например, если имеем вектор $v = \{v_5, v_9, v_{13}\}$, то $P_i v = \{0, 0, 0, 0, v_5, 0, 0, 0, v_9, 0, 0, 0, v_{13}, 0, \ldots, 0\}$, где последний 0 стоит на месте с номером, совпадающим с размерностью N исходной системы. Определим подматрицу в подобласти S_i как $A_i = A\big(R_i^\delta\big)^t$, а ее диагональный блок как $A_{ii} = R_i^\delta A\big(R_i^\delta\big)^t$. Тогда с алгебраической точки зрения действие предобусловливающего оператора, реализующего RAS, записывается следующим образом:

$$M_{RAS}^{-1} = \sum_{i=1}^{p} P_i R_i A_{ii}^{-1} R_i^\delta, \tag{4}$$

где $R_i^\delta : L(V) \to L(S_i^\delta)$, $A_{ii}^{-1} : L(S_i^\delta) \to L(S_i^\delta)$, $R_i : L(S_i^\delta) \to L(S_i)$ и $P_i : L(S_i) \to L(V)$.

Рассмотрим, как в данном случае реализуется (4) с точки зрения участвующих в решении вычислительных устройств и программ. Первое существенное отличие от конвенциональной алгебраической записи заключается в том, что в целях повышения эффективности векторы из $L(V)$ обычно хранятся в распределенном виде. Например, каждый i-й процесс может хранить только те компоненты вектора, которые соответствуют подобласти S_i. В таком случае действие оператора R_i^δ включает в себя обмен данными с другими процессами и сборку вектора правой части. При этом в отдельном i-м процессе происходят следующие действия:

— ограничение исходного вектора на $L\big(S_i \cap S_j^\delta\big)$ для всех j, где j — номера соседних процессов, т.е. таких, что $S_i \cap S_j^\delta \neq \emptyset$;

— отправка полученных на предыдущем шаге значений соседним процессам средствами MPI;

— соответствующий прием посылок;

— продолжение принятых компонент вектора на $L\big(S_i^\delta\big)$ и их сложение с вектором правой части для решения СЛАУ в подобласти;

— после применения оператора $A_{i,i}^{-1}$ — ограничение получившегося вектора на $L(S_i)$.

Вообще говоря, данная последовательность действий не является единственной и зависит от выбора способа хранения векторов в многопроцессорной среде. В то же время следует отметить необходимость выполнения подобных операций вне зависимости от того, как распределены данные в памяти параллельной программы МДО в случае RAS-подхода.

Остановимся теперь на синхронизации вычислений по подобластям в параллельной программной реализации МДО.

В абстрактной параллельной программе синхронизацией называется согласование по времени выполнения параллельных вычислений в разных MPI-процессах в том смысле, что после достижения всеми процессами так называемой точки синхронизации выполнение вычислений в каждом из параллельных процессов может быть продолжено до достижения этим процессом следующей точки синхронизации [10]. При использовании модели передачи сообщений (в нашем случае — MPI) в параллельном коде МДО с внешними итерациями в подпространствах Крылова работу параллельных процессов можно синхронизировать с помощью функций обмена данными, а также глобальными редуцированными операциями [11].

Для методов в подпространствах Крылова такую глобальную операцию следует использовать при вычислении скалярных произведений различных векторов (например, векторов невязки). Глобальная операция `mpi_allreduce` служит двоякой цели: во-первых, собирает значения скалярного умножения двух

глобальных векторов, распределенных по процессам, при этом частичные скалярные произведения со "своими" подвекторами вычисляются одновременно во всех подобластях, а во-вторых, служит точкой синхронизации параллельных частей программы, работающих каждая со своей выделенной подобластью, т.е. со своим набором данных. Отметим, что операция скалярного произведения является в градиентных итерационных методах неоднократной на каждой итерации (например, в методе BiCGStab (BiConjugate Gradient Stabilized) [12] она встречается четыре раза на итерации), поэтому использование упомянутой глобальной операции вынужденно является тоже неоднократным.

Подобласти взаимодействуют между собой через обмен данными в виде передачи значений неизвестных векторов в узлах, соседствующих через границу раздела подобластей. Использование неблокирующих операций `mpi_isend` и `mpi_irecv` позволяет вести вычисления на фоне передачи данных, например вычисление правой части для подсистемы в подобласти (3).

5. Параллельная программа МДО в вычислительном эксперименте. С ростом количества подобластей, на которые делится расчетная область, и соответствующим увеличением количества MPI-процессов, на котором запускается параллельная программа, естественно было бы ожидать, что время выполнения программы по крайней мере не будет расти. Более того, в рамках существующей традиции научного сообщества в идеальном случае ожидается следующий "идеальный" результат от использования параллелизма в разрабатываемом программном обеспечении: пропорциональное уменьшение времени счета при фиксированной размерности задачи и одновременном росте числа MPI-процессов или невозрастание времени при умеренном росте размерности задачи и решении на заданном числе MPI-процессов по сравнению с временем решения этой задачи в непараллельном режиме.

Однако в качестве реальных данных можно привести, например, данные вычислительного эксперимента из [13]: с ростом числа подобластей от 8 до 256 (в 32 раза) в двумерной задаче диффузии–конвекции на структурированной сетке при числе узлов в одной подобласти 128^2 время решения задачи растет от 1.9 сек. до 30 сек. (вместе с ростом числа итераций метода GMRES (Generalized Minimal Residual) от 65 до 295), т.е примерно в 15 раз.

Далее приведены результаты расчетов с использованием авторского параллельного кода, являющегося частью библиотеки Krylov, для трехмерной краевой задачи в единичном кубе $[0,1]^3$ для уравнения диффузии на кубической сетке с числом узлов $N = 200^3$. При этом число неизвестных равно 8×10^6.

Применялась геометрическая декомпозиция: кубическая область разбивалась на параллелепипедальные подобласти, число которых равно числу использованных в запуске кластерных узлов nn, умноженному на число ppn MPI-процессов на каждом узле. Так, например, при $nn = 12$, $ppn = 8$ имеем $nn \times ppn = 96$, поэтому разбиение на подобласти по трем направлениям d_1, d_2, d_3 было взято как $6 \times 4 \times 4 = 96$, где d_i — количество частей, на которые разбивается ребро исходного куба по направлению i-й координатной оси. Отметим, что так как объем обменной информации прямо пропорционален (в трехмерном случае) общей площади поверхности соседних подобластей, а минимальная площадь поверхности — у куба, то с геометрической точки зрения нужно стремиться к как можно более "кубическому" разбиению на подобласти. Для нашей задачи это означает, что в некотором смысле "идеальными" являются разбиения на 2^3, 4^3, 8^3 и т.д. подобластей. Необходимо отметить, что в случае, когда декомпозиция должна быть осуществлена алгебраически, т.е. разбита на части (подматрицы) исходная матрица, применяются алгоритмы, основанные на теории графов, которые реализованы, например, в пакете MUMPS (MUlti-frontal Massively Parallel Solver) [14].

Внешний итерационный процесс проводился по методу FGMRES. Решение СЛАУ в подобластях осуществлялось тем же методом с предобусловливателем ILU0 (Incomplete LU preconditioner) [2]. Критерием остановки как внешнего, так и внутреннего итерационного процесса являлось выполнение условия

$$\left\|r^n\right\|_2 \leqslant \varepsilon \|f\|_2,$$

причем для внутреннего процесса было выбрано значение $\varepsilon = 0.1$, а для внешнего — $\varepsilon = 10^{-7}$.

Для проведения вычислительных экспериментов авторами был использован кластер НКС-1П ЦКП Сибирского суперкомпьютерного центра ИВМиМГ СО РАН [15] с установленным на нем программным обеспечением Intel MPI 4.1.

В табл. 1 в каждой клетке приведены три числа по вертикали: разбиение $d_1 \times d_2 \times d_3$ области на подобласти по каждому из направлений, под ним — время вычислений в секундах на кластерных узлах Broadwell и третье число — время передачи данных в секундах (т.е. общее время расчета задачи на данной конфигурации запуска есть сумма двух времен в ячейке). Такое отдельное представление данных о временах позволяет оценить и проследить стоимость пересылок и их долю по отношению к вычислениям в рассматриваемом методе МДО (или DDM, от Domain Decomposition Method). Данные таблицы полу-

чены в зависимости от конфигурации запуска, т.е. значений nn, которым соответствуют столбцы, и ppn, которым соответствуют строки. Во время проведения эксперимента были доступны все 20 узлов данного вида, на каждом из которых установлены два 16-ядерных процессора Intel®Xeon®E5-2697A v4. Таким образом, в серии вычислительных экспериментов естественными ограничениями для nn и ppn были 20 и 32 соответственно. Полужирным шрифтом в данной таблице выделен минимум общего времени расчета по столбцам.

Таблица 1

Время расчета на кластерных узлах Broadwell

$ppn \setminus nn$	1	2	4	8	12	16	18	20
2	2x1x1 83,19 3,93	2x2x1 47,86 4,40	2x2x2 25,09 3,85	4x2x2 19,60 4,32	3x4x2 15,96 4,15	4x4x2 13,44 3,82	3x3x4 **12,49** **3,71**	5x4x2 **12,43** **3,72**
4	2x2x1 64,38 5,60	2x2x2 27,06 4,07	4x2x2 20,15 4,25	4x4x2 **13,52** **3,74**	3x4x4 **14,15** **3,78**	4x4x4 **11,59** **3,96**	6x3x4 13,22 4,58	5x4x4 13,21 4,69
8	2x2x2 **40,15** **5,30**	4x2x2 22,59 4,06	4x4x2 **15,13** **3,69**	4x4x4 15,33 4,33	6x4x4 20,23 5,67	8x4x4 19,41 7,06	6x6x4 28,56 7,76	5x8x4 30,08 8,31
12	3x2x2 49,38 6,64	3x4x2 22,07 4,36	3x4x4 17,22 4,43	6x4x4 21,26 6,34	6x6x4 29,13 8,90	6x8x4 37,59 11,25	6x6x6 39,59 11,46	5x6x8 47,11 12,79
16	4x2x2 42,09 5,18	4x4x2 **18,34** **3,56**	4x4x4 16,40 4,68	8x4x4 26,07 8,33	6x8x4 37,44 11,54	8x8x4 50,57 14,79	6x6x8 55,25 15,2	5x8x8 62,17 17,40
20	5x2x2 50,47 7,83	5x4x2 22,44 4,98	5x4x4 20,37 5,92	5x8x4 30,91 10,49	5x6x8 46,73 14,34	5x8x8 62,53 18,04	10x6x6 75,95 20,66	10x5x8 81,93 21,05
24	3x4x2 60,86 6,77	3x4x4 23,42 4,02	6x4x4 23,27 6,81	6x8x4 37,95 12,33	6x6x8 55,22 15,96	6x8x8 72,20 19,20	9x6x8 88,61 22,58	10x6x8 99,57 26,03
28	7x2x2 52,12 7,14	7x4x2 22,59 4,69	7x4x4 24,45 7,50	7x8x4 43,60 14,46	7x6x8 64,81 20,06	7x8x8 86,48 25,75	7x9x8 104,96 30,13	7x10x8 118,29 33,04
32	4x4x2 43,75 5,29	4x4x4 19,28 3,76	8x4x4 26,80 8,92	8x8x4 49,70 16,21	6x8x8 70,90 19,50	8x8x8 100,59 30,14	9x8x8 124,22 34,30	10x8x8 139,14 37,86

Сравнение времен столбцов для $nn = 1$ и $nn = 2$ говорит о том, что запуск на двух узлах дает примерно двукратное уменьшение времени расчета по сравнению с запуском на одном узле. Однако это уменьшение пропадает с переходом к $nn = 4, 8, \ldots$ и увеличением ppn, причем время начинает даже превосходить время расчета на одном узле, и это происходит тем скорее, чем больше значение ppn.

Отметим, что если использовать только один узел кластера (см. данные только первого столбца), то применение MPI даже на одном узле может ускорить (до некоторого значения ppn) расчет: так, запуски при $ppn = 8, 12, 16$ дают примерно одинаковый результат, который лучше случая с $ppn = 2$ примерно в два раза.

Глобальный минимум времени по данным таблицы — это $11{,}59 + 3{,}96 = 15{,}55$ сек. Он, достигается при $nn = 16$ и $ppn = 4$. Времена, соответствующие движению от побочной диагонали таблицы к нижнему правому углу таблицы (что означает запуск на все большем числе узлов с все большим числом MPI-процессов на узел), увеличиваются по отношению к выделенным временам из-за большого числа подобластей, т.е. из-за “дорогих” по времени обменов данными: так, максимальное время обменов по данным таблицы — $37, 86$ секунд, и оно получается именно при максимальных значениях nn и ppn. Это говорит о том, что в задачах МДО не следует стремиться к использованию максимального числа задействованных кластерных узлов и одновременно максимального числа MPI-потоков на узел в надежде минимизировать время расчета задачи заданного фиксированного размера.

Рассмотрим выделенные времена. Нетрудно заметить, что они находятся в верхней половине таблицы, а при числе узлов кластера большем восьми — примерно на одной горизонтали. Глобальный минимум меньше чем на 10% отличается от соседних данных, а начиная с $nn = 8$ время расчета почти не изменяется. Процент времени на обмены относительно общего времени расчета задачи составляет минимум 11,6% и достигается при запуске на одном узле, при переходе к двум узлам увеличивается до 16,2%, а при запуске на числе узлов от 4 до 20 находится в диапазоне 19,6%–25%.

Таблица 2
Время расчета на кластерных узлах Intel Xeon Phi

$ppn \setminus nn$	1	2	4	8	12
1	1x1x1 150,69 0	2x1x1 230,38 11,10	2x2x1 146,86 15,77	2x2x2 71,40 12,69	3x2x2 63,98 13,93
2	2x1x1 233,07 12,56	2x2x1 147,52 16,07	2x2x2 71,50 12,84	4x2x2 49,79 11,73	3x4x2 37,07 9,39
4	2x2x1 148,89 15,77	2x2x2 73,41 13,09	4x2x2 50,40 11,68	4x4x2 30,48 8,08	3x4x4 **23,74** **7,02**
8	2x2x2 78,57 13,53	4x2x2 54,01 12,29	4x4x2 32,54 8,35	4x4x4 **22,55** **6,34**	6x4x4 26,15 7,44
12	3x2x2 71,95 14,08	3x4x2 42,27 10,25	3x4x4 26,74 7,57	6x4x4 27,76 8,26	6x6x4 34,38 10,72
16	4x2x2 61,57 13,93	4x4x2 36,90 9,25	4x4x4 **24,56** **6,91**	8x4x4 32,42 10,22	6x8x4 45,11 14,80
20	5x2x2 55,18 12,44	5x4x2 35,13 9,00	5x4x4 27,56 8,12	5x8x4 36,83 12,79	5x6x8 54,85 16,59
24	3x4x2 48,65 11,42	3x4x4 29,48 8,20	6x4x4 28,28 8,79	6x8x4 43,81 15,23	6x6x8 65,09 19,12
28	7x2x2 54,56 14,50	7x4x2 31,93 8,48	7x4x4 32,69 10,44	7x8x4 52,78 17,88	7x6x8 75,12 22,27
32	4x4x2 43,53 10,79	4x4x4 **26,62** **7,14**	8x4x4 33,07 11,32	8x8x4 59,84 19,74	6x8x8 83,04 24,90
40	5x4x2 50,49 15,37	5x4x4 30,86 8,56	5x8x4 39,71 13,87	5x8x8 71,85 22,82	10x6x8 115,38 35,34
48	3x4x4 37,35 10,22	6x4x4 30,98 8,82	6x8x4 44,81 16,58	6x8x8 83,13 26,72	9x8x8 144,71 46,32
56	7x4x2 36,65 9,86	8x4x4 31,73 9,32	7x8x4 50,82 18,53	7x8x8 97,54 32,06	7x12x8 185,61 64,28
64	4x4x4 **30,01** **8,04**	8x4x4 33,86 9,84	8x8x4 57,37 20,73	8x8x8 115,64 40,37	12x8x8 220,98 76,51

Данные в табл. 2 аналогичны данным первой таблицы, а отличие состоит в том, что расчеты были

проведены на вычислительных узлах Intel Xeon Phi 7290 (KNL — Intel Knights Landing nodes). На время проведения эксперимента были доступны 12 из 16 имеющихся на кластере таких узлов, на каждом из которых установлен один 72-ядерный процессор. Полужирным шрифтом выделен минимум общего времени расчета по столбцам.

Глобальный минимум по данным этой таблицы — это 22,55 + 6,34 = 28,89 сек. Он достигается при $nn = 8$ и $ppn = 8$. По сравнению с запуском непараллельного варианта ($nn = ppn = 1$) расчет происходит быстрее примерно в пять раз. В этом варианте запуска достигается и минимальное время, затрачиваемое на обмены — оно составляет примерно 22% от общего времени. Выделенные данные располагаются, как и в табл. 1, около побочной диагонали таблицы, а данные ниже и выше этой диагонали говорят об увеличении общего времени расчета. Кроме того, можно заметить, что выделенные минимумы в этой таблице соответствуют кубическому разбиению на подобласти в 4 из 5 случаях (при этом некубическое разбиение максимально приближено к кубическому при заданной конфигурации запуска).

Для осмысления приведенных данных в целом необходимо сделать следующее замечание. На производительность параллельной программы влияет множество факторов. Среди них определяющими можно назвать следующие: доступные вычислительные ресурсы (количества кластерных узлов и процессоров на них), объем доступной кэш-памяти, пропускная способность шины памяти, величина задержки при доступе к некэшированным данным, задержка и пропускная способность сети, связывающей узлы кластерной системы. Следуя по строкам приведенных выше таблиц слева направо, мы увеличиваем количество доступных программе вычислительных ресурсов. Однако вслед за этим неизбежно растут и затраты на коммуникацию между запущенными программой MPI-процессами. Следуя же по столбцам сверху вниз, мы пытаемся увеличить загрузку аппаратной части, не изменяя при этом количество доступных ресурсов. При этом также происходит уменьшение размеров подобластей, что может способствовать более эффективному использованию кэш-памяти. Как видно из таблиц, до определенного момента эта стратегия действительно позволяет более полно задействовать имеющиеся ресурсы, однако затем время решения задачи начинает расти. Это связано как с уже упомянутыми коммуникационными затратами, так и с исчерпанием пропускной способности шины памяти.

К данным обеих таблиц нужно сделать еще одно замечание. С ростом количества подобластей увеличивается количество итераций используемого метода: например, при переходе от 16 подобластей ($nn \times ppn = 2 \times 8$) к 64 ($nn \times ppn = 8 \times 8$), а затем к 128 ($nn \times ppn = 16 \times 8$) количество итераций увеличивается от 138 до 157 и до 183 и одновременно растет размерность подпространств Крылова, т.е. количества векторов, сохраняемых по ходу работы программы. Эти два фактора тоже оказывают существенное влияние на время работы программы.

Для определения “узких” мест рассматриваемой программной реализации МДО на одном из вариантов (для 32 подобластей) был запущен Intel Trace Analyzer and Collector [16]. Из результатов его работы следует, что 67% времени уделяется на работу непараллельной части программы, на работу MPI тратится 14% времени, а на работу OpenMP — примерно 18% времени (в реализации итерационного алгоритма были использованы функции из библиотеки Intel MKL [17], которые хорошо оптимизированы для работы на платформах Intel, в частности с использованием OpenMP). Их этих данных следует замечание по поводу оптимизации рассматриваемого МДО-кода: следует стремиться к уменьшению объема непараллельной части кода, а также переходить к гибридному распараллеливанию, сочетающему технологии как MPI, так и OpenMP.

6. Заключение. В настоящей статье рассмотрены разнообразные аспекты такой комплексной проблемы, как создание параллельного программного обеспечения для методов декомпозиции областей. Широкий класс методов декомпозиции областей представлен как многоликая сеточно-геометрическая, матрично-алгебраическая и информационно-графовая структуры, взаимообогащающие это актуальное направление исследований и прикладных разработок.

Трудно дать конкретные рекомендации пользователям подобного программного обеспечения по его запуску на современных кластерных системах. Например, в статье [18] говорится, в частности, о выборе оптимальной конфигурации запуска: “К сожалению, в обсуждаемых в данной работе алгоритмах решения разреженных СЛАУ выбор оптимального режима запуска затруднен ... оценка размеров необходимой памяти и, тем более, оптимального соотношения числа процессов/потоков на одном узле является достаточно сложной задачей. Необходимо отметить, что пакет MUMPS успешно справляется с оценкой необходимых объемов памяти без помощи пользователя. Тем не менее, выбор оптимального числа процессов MPI и потоков параллельного BLAS на вычислительном узле необходимо выполнять самостоятельно”.

Можно констатировать, что для представленного класса задач стратегия проведения серии вычислительных экспериментов на ряде близких к некоторым оптимальным (заранее не известным, но доста-

точно небольшим) значений числа используемых в запусках кластерных узлов и числа MPI-потоков на узел является единственно надежной в условиях современного сочетания программного и аппаратного обеспечения, предоставляемого конечному пользователю. Следует ожидать, что при запуске параллельного кода МДО при использовании максимально возможного на данной кластерной системе количества ресурсов время расчета не будет минимальным.

При этом для увеличения эффективности параллельной реализации МДО необходимо проводить отдельную дополнительную работу по настройке кода и переходу на гибридную параллелизацию, которая сочетает в себе технологии MPI и OpenMP и, как правило, дает наилучшую производительность параллельного приложения.

Зачастую требованием к разработчику параллельного программного обеспечения (ПО) для МДО является получение решения задачи за как можно меньшее время. Однако приведенные данные говорят о том, что этот критерий здесь не является определяющим. Возможно, пришло время сменить парадигму оценки качества параллельного ПО в целом, когда определяющим условием выбора в пользу той или иной параллельной реализации некоторого алгоритма должно стать не величина сильной или слабой масштабируемости этого ПО, а опять же сама возможность получения решения большой задачи с использованием все большего числа узлов доступной кластерной системы (даже несмотря на вероятное существенное увеличение времени выполнения программы).

Работа поддержана грантом Российского научного фонда № 14–11–00485П, вычислительная часть поддержана грантом РФФИ № 18–01–00295 А.

СПИСОК ЛИТЕРАТУРЫ

1. *Бутюгин Д.С., Гурьева Я.Л., Ильин В.П., Перевозкин Д.В., Петухов А.В., Скопин И.Н.* Функциональность и технологии алгебраических решателей в библиотеке Krylov // Вестник ЮУрГУ. Серия “Вычислительная математика и информатика”. 2013. **2**, № 3. 92–105.
2. *Саад Ю.* Итерационные методы для разреженных линейных систем. М.: Изд-во Моск. ун-та, 2013.
3. *Saad Y.* A flexible inner-outer preconditioned GMRES algorithm // SIAM J. Sci. Comput. 1993. **14**, N 2. 461–469.
4. *Гурьева Я.Л., Ильин В.П., Перевозкин Д.В.* Алгебро-геометрические и информационные структуры методов декомпозиции областей // Вычислительные методы и программирование: Новые вычислительные технологии. 2016. **17**. 132–146.
5. http://mpi-forum.org/
6. http://parallel.ru/vvv/mpi.html
7. *Писсанецки С.* Технология разреженных матриц. М.: Мир, 1988.
8. *Cai X.C., Farhat C., Sarkis M.* A minimum overlap restricted additive Schwarz preconditioner and applications in 3D flow simulations // Contemporary Mathematics. 1998. **218**. 479–485.
9. *Cai X.C., Sarkis M.* A restricted additive Schwarz preconditioner for general sparse linear systems // SIAM Journal on Scientific Computing. 1999. **21**, N 2. 792–797.
10. https://www.intuit.ru/studies/courses/4447/983/lecture/14927?page=3
11. *Корнеев В.Д.* Параллельное программирование в MPI. Новосибирск: ИВМиМГ СО РАН, 2002.
12. http://en.wikipedia.org/wiki/Biconjugate_gradient_method
13. *Giraud L., Tuminaro R.S.* Algebraic domain decomposition preconditioners. Technical Report ENSEEIHT-IRIT RT/APO/06/07. Toulouse: Université Paul Sabatier, 2006.
14. http://mumps.enseeiht.fr/
15. ЦКП Сибирский суперкомпьютерный центр ИВМиМГ СО РАН. http://www2.sscc.ru/Default.htm
16. https://software.intel.com/en-us/intel-trace-analyzer
17. https://software.intel.com/en-us/mkl
18. *Лебедев С.А., Мееров И.Б., Сысоев А.В., Бартенев Ю.Г., Бастраков С.И., Козинов Е.А., Лебедев И.Г., Малова А.Ю., Стаканов А.Н., Старостин Н.В.* Оптимизация и применение пакета MUMPS для решения трехмерных стационарных задач прочности на кластерных системах // Научный сервис в сети Интернет: все грани параллелизма. Труды Международной суперкомпьютерной конференции (23–28 сентября 2013 г., г. Новороссийск). М.: Изд-во Моск. ун-та, 2013. 233–237.

Поступила в редакцию
19.04.2018

Issues of Parallel Software Development for the Domain Decomposition Methods

Y. L. Gurieva [1] **and D. V. Perevozkin** [2]

[1] *Institute of Computational Mathematics and Mathematical Geophysics, Siberian Branch of Russian Academy of Sciences; prospekt Lavrentyeva 6, Novosibirsk, 630090, Russia; Ph.D., Senior Scientist, e-mail: yana@lapasrv.sscc.ru*

[2] *Institute of Computational Mathematics and Mathematical Geophysics, Siberian Branch of Russian Academy of Sciences; prospekt Lavrentyeva 6, Novosibirsk, 630090, Russia; Junior Scientist, e-mail: dperevozkin@mail.ru*

Received April 19, 2018

Abstract: Various aspects of parallel software development for the domain decomposition methods are considered: the application of MPI programming technology for cluster systems, the choice points in the design of parallel programs for the domain decomposition methods, the need to implement a matrix action without its explicit representation, the work with index sets in the software implementation of restriction and continuation operators as well as in the data exchange between subdomains. On a series of numerical experiments for a model problem, the questions of the best choice of the configuration of launching an executable program on a cluster are studied to minimize the computation time and a strategy for performing such experiments is proposed.

Keywords: domain decomposition method, parallel algorithm, scalability, data structures, numerical experiment.

References

1. D. S. Butyugin, Y. L. Guryeva, V. P. Il'in, et al., "Parallel Algebraic Solvers Library Krylov," Vestn. South Ural Univ. Ser. Vychisl. Mat. Inf. **2** (3), 92–105 (2013).

2. Y. Saad, *Iterative Methods for Sparse Linear Systems* (SIAM, Philadelphia, 2003; Mosk. Gos. Univ., Moscow, 2013).

3. Y. Saad, "A Flexible Inner-Outer Preconditioned GMRES Algorithm," SIAM J. Sci. Comput. **14** (2), 461–469 (1993).

4. Y. L. Gurieva, V. P. Il'in, and D. V. Perevozkin, "Algebraic-Geometric and Information Structures of Domain Decomposition Methods," Vychisl. Metody Programm. **17**, 132–146 (2016).

5. MPI Forum. https://www.mpi-forum.org. Cited May 22, 2018.

6. Message Passing Interface (MPI). http://parallel.ru/vvv/mpi.html. Cited May 22, 2018.

7. S. Pissanetzky, *Sparse Matrix Technology* (Academic, London, 1984; Mir, Moscow, 1988).

8. X.-C. Cai, C. Farhat, and M. Sarkis, "A Minimum Overlap Restricted Additive Schwarz Preconditioner and Applications in 3D Flow Simulations," Contemp. Math. **218**, 479–485 (1998).

9. X.-C. Cai and M. Sarkis, "A Restricted Additive Schwarz Preconditioner for General Sparse Linear Systems," SIAM J. Sci. Comput. **21** (2), 792–797 (1999).

10. Intel Parallel Programming Professional (Introduction). https://www.intuit.ru/studies/courses/4447/983/lecture/14927?page=3

11. V. D. Korneev, *Parallel Programming with MPI* (Inst. Comput. Math. Math. Geophys., Novosibirsk, 2002) [in Russian].

12. Biconjugate Gradient Method. http://en.wikipedia.org/wiki/Biconjugate_gradient_method. Cited May 22, 2018.

13. L. Giraud and R. S. Tuminaro, *Algebraic Domain Decomposition Preconditioners*, Technical Report ENSEEIHT-IRIT RT/APO/06/07 (Université Paul Sabatier, Toulouse, 2006).

14. MUMPS: MUltifrontal Massively Parallel sparse direct Solver. http://mumps.enseeiht.fr. Cited May 22, 2018.

15. Siberian Supercomputing Center. http://www2.sscc.ru/Default.htm. Cited May 22, 2018.

16. Intel Trace Analyzer and Collector. https://software.intel.com/en-us/intel-trace-analyzer. Cited May 22, 2018.

17. Intel Math Kernel Library. https://software.intel.com/en-us/mkl. Cited May 22, 2018.

18. S. A. Lebedev, I. B. Meerov, A. V. Sysoev, et al., "Optimization and Application of the MUMPS Package for Solving the Three-Dimensional Stationary Strength Problems on Cluster Systems," in *Proc. Int. Supercomputer Conf., Novorossiysk, Russia, September 23–28, 2013* (Mosk. Gos. Univ., Moscow, 2013), pp. 233–237.