УДК 519.612, 519.684.4 DOI: 10.14529/cmse200203

ПАРАЛЛЕЛЬНОЕ РЕШЕНИЕ СИСТЕМ ЛИНЕЙНЫХ УРАВНЕНИЙ НА ГИБРИДНОЙ АРХИТЕКТУРЕ CPU+GPU*

© 2020 Н.С. Недожогин, С.П. Копысов, А.К. Новиков

Удмуртский государственный университет

(426034 Ижевск, ул. Университетская, д. 1)

E-mail: nedozhogin07@gmail.com, s.kopysov@gmail.com, sc_work@mail.ru

Поступила в редакцию: 29.01.2020

В статье рассматривается параллельная реализация решения систем линейных алгебраических уравнений на вычислительных узлах, содержащих центральный процессор (CPU) и графические ускорители (GPU). Производительность параллельных алгоритмов для классических схем метода сопряженных градиентов при совместном использовании CPU и GPU существенно ограничивается наличием точек синхронизации. В статье исследуется конвейерный вариант метода сопряженных градиентов с одной точкой синхронизации и возможностью распределения нагрузки между CPU и GPU при решении систем уравнений большой размерности. Численные эксперименты проведены на тестовых матрицах и вычислительных узлах разной производительности гетерогенного кластера, что позволило оценить вклад коммуникационных затрат. Алгоритмы реализованы при совместном использовании технологий MPI, OpenMP и CUDA. Предложенные алгоритмы помимо сокращения времени выполнения позволяют решать системы линейных уравнений и большего порядка, для которых не обеспечиваются необходимые ресурсы памяти одним GPU или вычислительным узлом. При этом конвейерный блочный алгоритм сокращает общее время выполнения за счет уменьшения точек синхронизации и объединения коммуникаций в одно сообщение.

Ключевые слова: параллельные вычисления, метод сопряженных градиентов, сокращение коммуникаций.

ОБРАЗЕЦ ЦИТИРОВАНИЯ

Недожогин Н.С., Копысов С.П., Новиков А.К. Параллельное решение систем линейных уравнений на гибридной архитектуре CPU+GPU // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. 2020. Т. 9, № 2. С. 40–54. DOI: 10.14529/cmse200203.

Введение

Ускорители вычислений содержатся в вычислительных узлах суперкомпьютеров и применяются достаточно успешно при решении многих вычислительных задач, несмотря на то, что центральный процессор (CPU) простаивает после запуска функций-ядер на ускорителе. Существует еще несколько важных условий, для которых многообещающим представляется совместное использование CPU и ускорителей (например, GPU) при параллельных вычислениях в рамках одной задачи.

Каждая архитектура CPU и GPU обладает уникальными особенностями и ориентирована на решение тех или иных задач, для которых характерна, например, высокая производительность или низкая латентность. Гибридные узлы, содержащие и совместно использующие CPU+GPU, могут обеспечить эффективное решение более широкого круга задач или одной задачи, для которой меняются параллельные свойства алгоритмов и определив

*Статья рекомендована к публикации программным комитетом Международной научной конференции «Параллельные вычислительные технологии (ПаВТ) 2020».

Работа выполнена при финансовой поддержке УдГУ в рамках конкурса грантов «Научный потенциал».

для ее выполнения то или другое исполнительное устройство. Отметим также высокую энергоэффективность гибридных вычислительных систем.

Совместное использование CPU+GPU связано с решением ряда задач: разделения вычислительной нагрузки между исполнительными устройствами с учетом производительности и пропускной способности памяти и ее размера; выявления параллельных свойств алгоритма на каждом шаге его выполнения и назначения устройства для выполнения.

Одной из трудоемких операций в численных методах является решение систем алгебраических уравнений (СЛАУ). В настоящее время предложено много параллельных алгоритмов, обеспечивающих высокую производительность и масштабируемость при решении больших разреженных систем уравнений на современных многопроцессорных архитектурах с иерархической структурой.

Построение гибридных решателей с комбинированием прямых и итерационных методов для решения СЛАУ позволяет использовать несколько уровней параллелизма [1–5]. Так в [6] построен и реализован гибридный метод решения систем уравнений дополнения Шура предобусловленными итерационными методами из подпространств Крылова при совместном использовании центральных процессоров (CPU) и графических ускорителей (GPU). При параллельном решении системы уравнений для дополнения Шура применялся классический предобусловленный метод сопряженных градиентов [7] для блочной упорядоченной матрицы и разделением вычислений при матричных операциях между CPU и одним или несколькими GPU. В настоящей работе рассматривается подход, сокращающий затраты на обмен данными между CPU и GPU за счет уменьшения числа точек глобальной синхронизации и конвейеризации вычислений.

Методы подпространства Крылова являются одними из наиболее эффективных вариантов решения крупномасштабных задач линейной алгебры. Однако, классические алгоритмы подпространства Крылова плохо масштабируются на современных архитектурах из-за наличия узких мест, связанных с синхронизацией вычислений. Конвейерные методы подпространства Крылова [8] со скрытыми коммуникациями обеспечивают высокую параллельную масштабируемость за счет перекрытия глобальных коммуникаций с вычислениями матрично-векторных операций и скалярных произведений векторов. Первые работы по сокращению коммуникаций были связаны с вариантом метода сопряженных градиентов (CG), имеющих одну коммуникацию на каждой итерации [9], с применением трехчленных рекуррентных соотношений CG [10].

Следующим этапом развития стало появление s-шаговых методов из подпространств Крылова [11], в которых итерационный процесс в s-блоке использует различные базисы подпространств Крылова. В результате удалось сократить число точек синхронизации до одной на s итераций. Однако, для большого числа процессоров (ядер) коммуникации все же могут занимать существенно больше времени, чем вычисление одного матрично-векторного произведения. В работе [12] предложен алгоритм CG, использующий вспомогательные вектора и перенос последовательной зависимости между вычислением матрично-векторного произведения и скалярными произведениями векторов. В данном подходе латентность коммуникаций заменяется дополнительными вычислениями.

Статья организована следующим образом. В разделе 1 рассмотрен конвейерный вариант метода сопряженных градиентов, приведен алгоритм и отличия от классического подхода. Раздел 2 посвящен вопросу совместного использования CPU и GPU, декомпозиции матрицы и обсуждению блочного варианта метода сопряженных градиентов. В разделе 3

приведены результаты численных экспериментов на тестовых матрицах из библиотеки университета Флорида. В заключении приводится краткая сводка результатов, полученных в ходе численных экспериментов, и указаны направления дальнейших исследований.

1. Конвейерный вариант метода сопряженных градиентов

Рассмотрим конвейерный вариант метода сопряженных градиентов, который математически эквивалентен классической форме предобусловленного метода CG и имеет такую же скорость сходимости.

Алгоритм 1: Конвейерный алгоритм метода CGwO

```
1  r = b − Ax
2  u = M^{-1} r
3  w = Au
4  γ_1 = (r, u)
5  δ = (w, u)
6  j = 0
   while ||r||_2/||b||_2 > ε do
7      m = M^{-1} w
8      n = Am
       if (j = 0) then
9          β = 0
       else
10         β = γ_1/γ_0
11     α = γ_1/(δ − βγ_1/α)
12     z = n + βz
13     w = w − αz
14     q = m + βq
15     s = w + βs
16     p = u + βp
17     x = x + αp
18     r = r − αs
19     u = u + αq
20     γ_0 = γ_1
21     γ_1 = (r, u); δ = (w, u)
22     j = j + 1
```

В этом алгоритме модификация векторов r_{j+1}, x_{j+1}, s_{j+1}, p_{j+1} и матрично-векторных произведений обеспечивает конвейерные вычисления. Вычисление скалярных произведений (строки 4, 5) может быть перекрыто с вычислением произведения на предобуславливатель (строка 2) и матрично–векторным произведением (строка 3). Однако, число триад в алгоритме увеличивается до восьми, в отличие от трех для классического варианта и четырех в [11]. В этом случае, возможно параллельное вычисление триад и двух скалярных произведений в начале итерационного процесса с одной точкой синхронизации.

Представленный в данной работе конвейерный вариант CG, может быть использован с любым предобуславливателем. Существуют два способа организации вычислений в предо-

бусловленном конвейерном CG, обеспечивающих компромисс между масштабируемостью и общим числом операций [13].

Таким образом, конвейерная схема CG, характеризуется: другой последовательностью вычислений; наличием глобальной коммуникации, которая может перекрываться с локальными вычислениями, такими как матрично-векторное произведение и операциями с предобуславливателем; возможностью организации асинхронных коммуникаций.

Выполнено сравнение двух вариантов метода сопряженных градиентов: классической схемы и конвейерной. В численных экспериментах (см. табл. 1) представлены результаты сравнения времени выполнения последовательного варианта классического CG и конвейерной схемы CGwO (алг. 1), исполняемых на CPU и GPU. Отметим, что в вариантах для GPU реализовывалось совместное вычисление всех скалярных произведений векторов в одной функции-ядре, независимо друг от друга. Для этого, при запуске ядра CUDA, задавалась размерность `Grid` иерархии нитей CUDA в двумерном виде: 3 набора блоков, каждый для выполнения вычислений над своей парой векторов. Это позволило сократить число обменов между памятью CPU и GPU, объединив все результирующие скаляры в одной коммуникации.

В тестовых расчетах использовались матрицы из коллекции университета Флориды (https://sparse.tamu.edu/). Вектор правых частей формировался, как построчная сумма элементов матрицы. Таким образом, решение системы $Ax = b$, размерности $N \times N$ (с числом ненулевых элементов nnz) представляет из себя вектор $x = (1, 1, \ldots, 1)^T$.

Для систем уравнений малой размерности время решения на CPU по классической схеме CG значительно меньше времени выполнения GPU при одном и том же числе итераций (см. табл. 1). Для больших систем затраты на синхронизации и пересылку между CPU и GPU перекрываются быстродействием GPU. В конвейерном варианте CGwO вычислительные затраты выполнения на GPU уменьшаются для всех рассмотренных систем уравнений практически трехкратно только за счет сокращения обменов между GPU и CPU при вычислении скалярных произведений.

2. Метод сопряженных градиентов при совместном использовании CPU и GPU

Рассмотрим применение Алгоритма 1 для параллельного решения сверхбольших систем уравнений на вычислительных узлах, каждый из которых содержит несколько CPU и GPU.

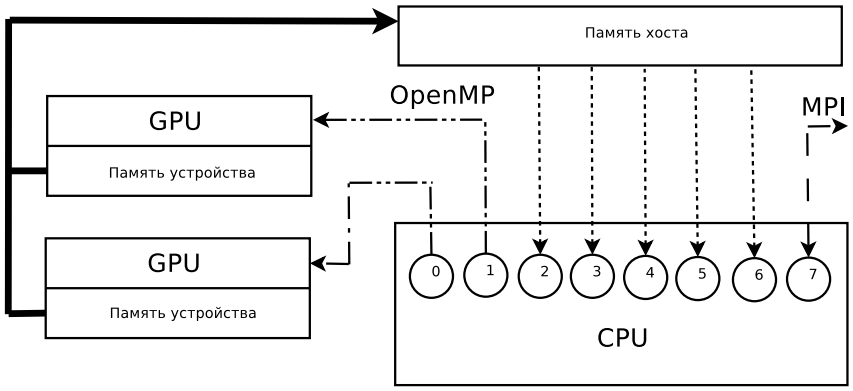

Рис. Гетерогенный вычислительный узел CPU+GPU

Для решения СЛАУ на нескольких GPU построим блочный алгоритм конвейерного метода сопряженных градиентов. Обмен данными между разными GPU в рамках одного вычислительного узла осуществляется с помощью технологии OpenMP, а обмен между разными вычислительными узлами — с помощью технологии MPI.

На рисунке представлена схема организации параллельных вычислений на гетерогенном вычислительном узле. Для примера рассмотрим узел, содержащий центральный восьмиядерный процессор и два графических ускорителя. Число OpenMP потоков выбирается по числу доступных ядер CPU. Первые два потока отвечают за обмен данными и запуск на двух GPU. Нити 2–6 обеспечивают вычисления на CPU и могут выполнять вычислительную работу над отдельным блоком матрицы СЛАУ. Последняя нить осуществляет MPI-коммуникации и обмен данными с другими вычислительными узлами.

2.1. Разделение матрицы

Для разделения матрицы A на блоки построим граф $G_A(V, E)$, где $V = \{i\}$ — множество вершин, связанных со строчным индексом матрицы (число вершин равно числу строк матрицы A); $E = \{(i, j)\}$ — множество ребер. Две вершины i и j считаются связанными, если в матрице A есть ненулевой элемент с индексами i и j. Полученный граф делится на блоки, число которых равно d. Например, для разделения графа можно использовать алгоритмом послойного разделения [14], который сокращает затраты на коммуникации за счет обменов только с двумя соседними узлами. После этого каждой вершине графа ставится в соответствие свой GPU или CPU. На каждом исполнительном устройстве вершины делятся на внутренние и граничные. Последние связаны хотя бы с одной вершиной, принадлежащей другому блоку.

После разделения каждый блок A_k исходной матрицы содержит в себе следующие подматрицы:

- $A_k^{[i_k,i_k]}$ — матрица связей между внутренними узлами;
- $A_k^{[i_k,b_k]}$, $A_k^{[b_k,i_k]}$ — матрицы связей между внутренними и граничными узлами;
- $A_k^{[b_k,b_l]}$ — матрица связи между граничными узлами k-го и l-го блоков.

Тогда матрица A может быть записана в следующем виде:

$$A = \begin{pmatrix} A_1^{[i_1,i_1]} & A_1^{[i_1,b_1]} & \cdots & 0 & 0 \\ A_1^{[b_1,i_1]} & A_1^{[b_1,b_1]} & \cdots & 0 & A_1^{[b_1,b_d]} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & A_d^{[i_d,i_d]} & A_d^{[i_d,b_d]} \\ 0 & A_d^{[b_d,b_1]} & \cdots & A_d^{[b_d,i_d]} & A_d^{[b_d,b_d]} \end{pmatrix}.$$

Используя полученное разделение, матрично-векторное произведение $n = Am$ разделим на две составляющие:

$$n_k^b = A_k^{[b_k,i_k]} m_k^i + \sum_{l=1}^{l \leqslant d} A_k^{[b_k,b_l]} m_l^b, \qquad n_k^i = A_k^{[i_k,i_k]} m_k^i + A_k^{[i_k,b_k]} n_k^b, \tag{1}$$

здесь k соответствует исполнительному устройству. Блочное представление векторов, участвующих в алгоритме, наследуется от разделения матрицы. Например, вектор m имеет вид $m^T = \left(m_1^i, m_1^b, \ldots, m_k^i, m_k^b, \ldots, m_d^i, m_d^b\right)$. Такая реализация матрично-векторного произведения уменьшает затраты на коммуникации между блоками на каждой итерации сопря-

женных градиентов. Для выполнения этой операции, требуется обмен векторами m_k^b, размер которых меньше размерности начального вектора m.

Разделение на блоки матрицы предобуславливателя M производится подобным образом.

2.2. Блочный конвейерный алгоритм

Используя блочное разделение матрицы и векторов, выполним распределение блоков матрицы на имеющиеся CPU и GPU. Число и размеры блоков позволяют распределять нагрузку в соответствии с производительностью имеющихся исполнительных устройств, в том числе и выделение нескольких блоков на одно устройство. Представим параллельную блочную схему метода CGwO, выполняемого каждым k-ом устройством в виде алг. 2, в котором выделены две параллельные ветви, выполняемые соответственно CPU и GPU/CPU.

Алгоритм 2: Блочный алгоритм CGwO выполняемый на k-ом устройстве

Data: Разделение матрицы на блоки $A_k^{[i_k,i_k]}$, $A_k^{[i_k,b_k]}$, $A_k^{[b_k,i_k]}$, $A_k^{[b_k,b_l]}$.

1	$r = b$	
2	$u = M^{-1}r$	
	`// Параллельно выполняемые ветви алгоритма`	
	`//` $(\texttt{CPU} \vee \texttt{GPU})_k$	`// CPU`
3	$w_k^i = A_k^{[i_k,i_k]} \cdot u_k^i + A_k^{[i_k,b_k]} \cdot u_k^b$	Сборка векторов u_k^b
4	$w_k^b = A_k^{[b_k,b_k]} \cdot u_k^b + A_k^{[b_k,i_k]} \cdot u_k^i$	$w_h^b = \sum_{l=1,l\neq k}^{l\leqslant d} A_k^{[b_k,b_l]} \cdot u_k^b$
5	$w_k^b = w_k^b + w_h^b$	Копирование w_h^b на $\texttt{GPU}_k$
6	$m = M^{-1}w$	Сборка векторов m_k^b
7	$\gamma_{1k} = (r_k, u_k)$; $\delta_k = (w_k, u_k)$	Сборка $\delta = \sum_k \delta_k; \gamma_1 = \sum_k \gamma_{1k}$
8	$j = 0$	
	while $\|\|r\|\|_2/\|\|b\|\|_2 > \varepsilon$ **do**	
9	$n_k^i = A_k^{[i_k,i_k]} \cdot m_k^i + A_k^{[i_k,b_k]} \cdot m_k^b$	$n_h^b = \sum_{l=1,l\neq k}^{l\leqslant d} A_k^{[b_k,b_l]} \cdot m_k^b$;
10	$n_k^b = A_k^{[b_k,b_k]} \cdot m_k^b + A_k^{[b_k,i_k]} \cdot m_k^i$	Копирование n_h^b на GPU_k;
11	$n_k^b = n_k^b + n_h^b$	$\beta = ((j = 0)\,?\,0 : \gamma_1/\gamma_0)$;
12	$z = n + \beta z$	$\alpha = \gamma_1/(\delta - \beta\gamma_1/\alpha)$;
13	$w = w - \alpha z$	
14	$q = m + \beta q$	
15	$s = w + \beta s$	Сборка векторов w_k^b;
16	$p = u + \beta p$	
17	$x = x + \alpha p$	
18	$r = r - \alpha s$	Сборка векторов m_k^b;
19	$u = u + \alpha q$	Сборка $\delta = \sum_k \delta_k; \gamma_1 = \sum_k \gamma_{1k}$;
20	$m = M^{-1}w$	
21	$\gamma_0 = \gamma_1$	
22	$\gamma_{1k} = (r_k, u_k)$; $\delta_k = (w_k, u_k)$	
23	$j = j + 1$	

Операции, выполняемые параллельно, записаны в одной строке алгоритма. Векторные операции на каждом исполняющем устройстве происходят в два этапа, для внутренних и граничных узлов. Обозначения внутренних и граничных узлов для векторов опущены, за

исключением матрично-векторного произведения. Скалярные произведения векторов выполняются независимо каждым исполняющим устройством над своими частями векторов. Суммирования промежуточных скаляров происходит в параллельных потоках, отвечающих за коммуникации, что является точкой синхронизации на каждой итерации алгоритма. В блочном CGwO по сравнению с алг. 1 был перенесен шаг предобуславливания (строка 7 в строку 20). Это сделано для того, чтобы совместить векторные операции на исполняющем устройстве и сборку вектора правых частей для выполнения матрично-векторного умножения в предобуславливании. В строке 11 справа используется тернарный оператор: если $j = 0$, то $\beta = 0$, в других случаях $\beta = \gamma_1/\gamma_0$. Нижний индекс h в алгоритме применяется для векторов, которые хранятся только в памяти CPU.

Численные эксперименты по применению алг. 2 проводились при различных вариантах конфигурации вычислительных узлов, содержащих несколько CPU и GPU. В самом общем случае, параллельные вычисления на нескольких гетерогенных вычислительных узлах, содержащих один и более CPU и несколько GPU, реализуются с помощью сочетания технологий: MPI, OpenMP и CUDA. Рассмотрим организацию вычислений на примере кластера, в составе которого имеется два вычислительных узла (8 ядер CPU и 2 GPU). Каждому вычислительному узлу ставится в соответствие параллельный процесс MPI. В параллельном процессе порождается 9 параллельных потоков OpenMP, что на один больше, чем доступные ядра CPU. Восьмой поток OpenMP отвечает за коммуникации между различными вычислительными узлами средствами технологии MPI (сборка векторов с помощью функции `Allgatherv`, сложение скаляров `Allreduce`) и различными GPU. В алг. 2 операции, выполняемые этим потоком, представлены справа. Нулевой и первый потоки OpenMP связываются с одним из доступных GPU-устройств и отвечают за пересылку данных между GPU-CPU (вызовы функций асинхронного копирования) и вспомогательные вычисления. Каждое доступное GPU-устройство (в дальнейшем рассматривается как исполняющее устройство) связывается с одним из параллельных потоков OpenMP, который отвечает за пересылку данных между GPU-CPU (вызовы функций асинхронного копирования) и участвует с восьмым потоком в операции матрично-векторного умножения на граничных узлах (строки 4, 9 правая колонка). Оставшиеся параллельные потоки (второй–седьмой) производят вычисления как отдельное исполняющее устройство для своего блока матрицы. Операции, выполняемые исполняющими устройствами в алг. 2 приведены слева.

Применение предобуславливателя строки 2, 6 и 20 подразумевает использование блочного матрично-векторного произведения вида (1) рассмотренного выше.

3. Численные эксперименты

Вычислительные эксперименты выполнялись на вычислительных узлах (ВУ) нескольких типов кластера «Уран» суперкомпьютерного центра ИММ УрО РАН (СКЦ ИММ УрО РАН).

Используемые вычислительные узлы имеют следующие характеристики:

- раздел «debug»: 4 узла tesla [31–32, 46–47] по два 8-ядерных процессора Intel Xeon E5-2660 (2,2 ГГц), оперативная память — 96 ГБ, 8 GPU Nvidia Tesla M2090 (6 ГБ графической памяти), коммуникационная сеть — Gigabit Ethernet (1 Гбит/с).
- раздел «tesla-v100»: два 18-ядерных процессора Intel Xeon Gold 6240 CPU (2,60 ГГц); оперативная память — 384 ГБ; 8 GPU Nvidia Tesla V100 (32 ГБ графической памяти).

- раздел «tesla [21–30]»: 10 узлов по два 6-ядерных процессора Intel Xeon X5675 (3,07 ГГц), оперативная память — 192 ГБ, 8 GPU Nvidia Tesla M2090 (6 ГБ графической памяти), коммуникационная сеть — Infiniband (20 Гбит/с).
- раздел «tesla [33–45]»: 13 узлов по два 8-ядерных процессора Intel Xeon E5-2660 (2,2 ГГц), оперативная память — 96 ГБ, 8 GPU Nvidia Tesla M2090 (6 ГБ графической памяти), коммуникационная сеть — Infiniband (20 Гб/с).
- раздел «tesla [48–52]»: 5 узлов по два 8-ядерных процессора Intel Xeon E5-2650 (2,6 ГГц), оперативная память — 64 ГБ, 3 GPU Nvidia Tesla K40m (12 ГБ графической памяти), коммуникационная сеть — Infiniband (20 Гб/с).

Таблица 1

Время решения алгоритмом CG на CPU и GPU, сек

Матрица	N	nnz	# итераций	ВУ	Время, сек	
					CG	CGwO
Plat362	362	5786	991	M2090	6,88E-01	**3,07E-01**
				K40m	4,13E-01	3,12E-01
1138_bus	1138	4054	717	M2090	3,81E-01	**1,84E-01**
				K40m	5,31E-01	2,01E-01
				debug	6,82E-01	1,90E-01
Muu	7102	170134	12	M2090	2,64E-01	4,68E-03
				K40m	3,31E-01	**4,55E-03**
Kuu	7102	340200	378	M2090	4,31E-01	**1,31E-01**
				K40m	4,39E-01	1,35E-01
Pres_Poisson	14822	715804	661	M2090	6,72E-01	3,13E-01
				K40m	6,346E-01	**2,73E-01**
Inline_1	503712	36816342	5642	M2090	4,74E+01	5,17E+01
				K40m	**3,06E+01**	3,37E+01
Fault_639	638802	28614564	4444	M2090	3,83E+01	4,32E+01
				K40m	**2,44E+01**	2,77E+01
				debug	2,44E+01	2,77E+01
thermal2	1228045	8580313	2493	M2090	1,35E+01	1,82E+01
				K40m	**8,33E+00**	1,18E+01
G3_circuit	1585478	7660826	592	M2090	3,43E+00	4,32E+00
				K40m	**1,94E+00**	2,92E+00
Quenn_4147	4147110	399499284	8257	M2090	5,46E+02	5,78E+02
				K40m	**3,55E+02**	3,75E+02

Результаты сравнения двух алгоритмов метода сопряженных градиентов на системах уравнений, содержащих тестовые матрицы, представлены в табл. 1. Результаты приведены для нескольких типов вычислительных узлов при использовании одного графического ускорителя. Матрицы упорядочены по увеличению порядка системы уравнений (N) и числа ненулевых элементов (nnz). Жирным выделено лучшее время решения системы в каждом случае.

Конвейерный алгоритм CGwO показал сокращение времени выполнения на небольших СЛАУ, для которых характерна небольшая вычислительная нагрузка, за счет чего сокращение коммуникаций обеспечивает меньшие временные затраты. Отметим, что классиче-

ский алгоритм CG был реализован на основе CUBLAS, а в варианте CGwO применяются матричные и векторные операции собственной GPU-реализации.

Для систем **Inline_1** и **Fault_639** время выполнения конвейерного алгоритма на 10 и 13,5 % больше относительно блочного варианта CG, что связано с выполнением дополнительных векторных операций, которые не перекрываются сокращением коммуникаций. С уменьшением числа итераций время выполнения алгоритмов CG и CgwO на одном GPU возрастает незначительно. Так, например, для матриц **G3_circuit** и **thermal2** с примерно равным числом уравнений значительно различаются результаты. Для системы с матрицей **thermal2** увеличение затрат становится более существенным, в сравнении с **G3_circuit**.

В табл. 2 и 3 представлены результаты выполнения блочных вариантов алгоритмов при вычислениях на нескольких вычислительных узлах. Здесь обозначение #CPU/#GPU означает число используемых графических ускорителей #GPU на каждом вычислительном узле, число которых равно #CPU. Как отмечалось ранее, при выполнении алгоритма на одном вычислительном узле с двумя ускорителями (например, 1CPU/2GPU) коммуникации обеспечивались только с помощью технологии OpenMP. В случае 8CPU/1GPU — на восьми вычислительных узлах при задействовании одного GPU на каждом, обмены осуществлялись на основе MPI.

Таблица 2

Время решения блочным алгоритмом CG на CPU/GPU, сек

Матрица/ВУ	BlockCG/число блоков			
	2	3	8	
	#CPU/#GPU			
	2/1	3/1	4/2	8/1
Plat362/M2090	**1,55E+00**	—	—	—
/K40m	1,92E+00	1,56E+00	—	—
1138_bus/M2090	1,84E+00	—	—	—
/K40m	1,90E+00	1,85E+00	—	—
/debug	**1,25E+01**	—	—	—
Muu/M2090	6,12E-01	—	1,84E+00	7,4E-01
/K40m	6,59E-01	**5,64E-01**	6,97E+00	—
Kuu/M2090	**1,30E+00**	—	1,35E+00	1,41E+00
/K40m	1,29E+00	1,36E+00	1,94E+00	—
Pres_Poisson/M2090	**1,55E+00**	—	1,57E+00	2,0E+00
/K40m	1,60E+00	1,66E+00	2,3E+00	—
Inline_1/M2090	3,76E+01	—	—	—
/K40m	2,66E+01	**2,20E+01**	—	—
Fault_639/M2090	3,18E+01	—	**1,55E+01**	2,09E+01
/K40m	2,20E+01	1,98E+01	2,89E+01	—
/debug	9,38E+01	—	—	—
thermal2/M2090	1,46E+01	—	9,77E+00	1,15E+01
/K40m	1,18E+01	**1,00E+01**	—	—
G3_circuit/M2090	4,27E+00	—	4,82E+00	**3,26E+00**
/K40m	4,04E+00	3,510E+00	4,06E+00	—
Quenn_4147/M2090	1,33E+02	1,55E+02	—	—

Значительное влияние характеристик сети на производительность блочных методов можно увидеть в табл. 2 и 3 для систем уравнений с матрицами **Fault_639** и **1138_bus**. Вычисления для этих систем уравнений выполнялись на различных вычислительных узлах (ВУ) с разной пропускной способностью и латентностью коммуникационной сети. При численных экспериментах на ВУ (раздел «debug»), соединенных коммуникационной сетью Gigabit Ethernet, затраты на коммуникации значительно увеличивают время выполнения алгоритма CG. Например, в варианте **1138_bus** на аппаратном разделе «debug» время выполнения конвейерного алгоритма в 3,6 раза меньше (строка «debug» в табл. 2 и 3 и любая строка в табл. 1). Использование коммуникационной сети Infiniband 20 Гб/с позволяет сократить время выполнения для всех представленных систем уравнений (строки «M2090» и «K40m»).

Сравнение строк таблиц, соответствующих «K40m» и «M2090», показывает, что использование GPU нового поколения обеспечивает существенное уменьшение времени выполнения. Особенно это заметно на системах больших размерностей **thermal2**, **G3_circuit**, где использование ускорителя Tesla K40m позволяет получить ускорение до 2,5 раз в сравнении с Tesla M2090 (в 2,5 раза в табл. 1 для системы **thermal2**, для остальных матриц в среднем в 1,5 раза быстрее).

При сокращении вычислительной нагрузки более явно сказывается уменьшение числа точек синхронизации и объединения пересылок за одну транзакцию. Это показывает сравнение систем с матрицами **Kuu** с **Muu**. При равном числе уравнений и ненулевых элементов обусловленность этих матриц существенно отличается и, как следствие, число итераций в методе сопряженных градиентов различно. Из табл. 1 видно, что использование конвейерного алгоритма для матрицы **Muu** дает ускорение в 70 раз по сравнению с матрицей **Kuu**, где ускорение только 2,8.

На матрицах большой размерности **Inline_1**, **Fault_639**, **thermal2** и **G3_circuit** ускорения не наблюдается, потому что сокращение коммуникаций не перекрывает затраты на дополнительные векторные операции (табл. 1). Использование блочных алгоритмов сокращает вычислительную нагрузку на один GPU, тем самым позволяет при некотором разделении матрицы и векторов (размерности ≈ 200000 на каждом GPU) получить ускорении вычислений. Например, для системы **Fault_639** порядка 638802 достигается невысокое ускорение (в 1,2–1,5 раза) при разделении на три блока, для системы **Inline_1** — на два блока.

Анализ результатов показал, что уменьшение объема данных за счет разделения матрицы и сокращение точек синхронизации незначительно снижают влияние коммуникационных затрат на общую производительность алгоритма. Только использование ВУ, соединенных Infiniband позволило получить ускорение при вычислениях на нескольких вычислительных узлах. Разделение матрицы на блоки позволило сократить время выполнения конвейерного блочного алгоритма по сравнению с классическим, выполняемом на одном узле, на матрицах **Inline_1**, **Fault_639** за счет сокращения вычислительной нагрузки, приходящейся на один графический ускоритель.

На системах большого порядка **thermal2**, **G3_circuit**, решаемых блочным вариантом алгоритма CGwO, сокращение коммуникаций и точек синхронизации так же не перекрывают возрастающие затраты на дополнительные векторные операции.

В табл. 4 представлены результаты по ускорению блочного конвейерного алгоритма метода сопряженных градиентов при разделении на большее число подобластей (12 и 16)

Таблица 3

Время решения блочным конвейерным алгоритмом CGwO на CPU/GPU, сек

Матрица/ВУ	BlockCGwO/число блоков			
	2	3	8	
	#CPU/#GPU			
	2/1	3/1	4/2	8/1
Plat362/M2090	**1,22E+00**	—	—	—
/K40m	1,28E+00	1,31E+00	—	
1138_bus/M2090	**9,28E-01**	—	—	—
/K40m	1,03E+00	1,04E+00	—	—
/debug	5,36E+00	—	—	—
Muu/M2090	2,29E-01	—	5,8E-01	**1,8E-01**
/K40m	2,89E-01	2,88E-01	—	—
Kuu/M2090	**6,43E-01**	—	0,80E+00	7,9E-01
/K40m	6,81E-01	7,95E-01	1,26E+00	—
Pres_Poisson/M2090	9,57E-01	—	1.08E+00	**9,5E-01**
/K40m	1,02E+00	1,19E+00	1,76E+00	—
Inline_1/M2090	3,65E+01	—	—	—
/K40m	2,48E+01	**1,97E+01**	—	—
Fault_639/M2090	3,03E+01	—	**1,49E+01**	1,69E+01
/K40m	2,05E+01	1,78E+01	2,13E+01	—
/debug	5,25E+01	—	—	—
thermal2/M2090	1,49E+01	—	9,33E+00	**8,92E+00**
/K40m	1,18E+01	9,65E+00	—	—
G3_circuit/M2090	3,99E+00		4,78E+00	**2,69E+00**
/K40m	3,27E+00	2,77E+00	3,86E+00	
Quenn_4147/M2090				1,4E+02

и вычислениях на узлах с GPU Nvidia Tesla M2090. Ускорение считалось относительно варианта на одном GPU из табл. 1. Все эксперименты производились при эксклюзивном использовании вычислительного узла, но не сети.

Таблица 4

Ускорение алгоритма CGwO

Матрица	Число блоков					
	12			16		
	#CPU/#GPU					
	2/6	3/4	12/1	2/8	4/4	16/1
Kuu	0,06	0,03	0,21	0,045	0,087	0,22
Inline_1	3,19	3,41	3,51	3,09	3,40	3,80
Fault_639	3,09	3,23	4,10	2.94	3,29	2,11
thermal2	1,56	1,95	2,15	1,50	1,55	2,22
Quenn_4147	4,73	5,01	5,07	4,77	5,50	5,34

Из представленных результатов видно, что использование многих узлов с одним GPU или небольшого числа узлов с несколькими GPU (не более 4-х на узел) дает примерно равное ускорение параллельного алгоритма. При использовании 8 ускорителей на узле ускорение снижается незначительно (7 %). Помимо этого большое число подобластей матрицы увеличивает затраты на коммуникации, которые не перекрываются за счет сокращения времени выполнения матричных и векторных операций. При разделении на 16 подобластей приложение выполняется на вычислительных узлах, содержащих CPU разных поколений (Intel Xeon X5675 и Intel Xeon E5-2660), что приводит к уменьшению ускорения для матриц **Quenn_4147** и **Fault_639**.

Заключение

Гибридные вычислительные узлы, содержащие и совместно использующие CPU+GPU позволяют обеспечить эффективное решение более широкого круга задач или одной задачи, для которой меняются параллельные свойства алгоритмов, и назначить на выполнение то или другое исполнительное устройство. Отметим также высокую энергоэффективность гибридных вычислительных систем в тех случаях, когда равномерно загружены центральные процессоры и графические ускорители вычислений.

В работе рассмотрена параллельная реализация решения систем линейных алгебраических уравнений на вычислительных узлах, содержащих центральный процессоры и графические ускорители. При совместном использовании CPU и GPU производительность параллельных алгоритмов для классических схем метода сопряженных градиентов существенно ограничивается наличием точек синхронизации. Предложен конвейерный вариант метода сопряженных градиентов с одной точкой синхронизации, возможностью асинхронных вычислений, распределения нагрузки между несколькими GPU, находящимися, как на одном вычислительном узле, так и для кластера GPU при решении систем уравнений большой размерности. Для дальнейшего увеличения эффективности вычислений предполагается исследовать не только коммуникационную нагрузку алгоритмов, но и распределение вычислительной нагрузки между CPU и GPU. Для получения более надежных временных оценок коммуникаций необходимо проведение серии вычислительных экспериментов на вычислительных системах с полностью монопольным режимом работы с большим числом гетерогенных узлов.

Из анализа полученных в ходе численных экспериментов данных можно сделать следующие выводы: использование конвейерного алгоритма снижает коммуникационные затраты, но увеличивает вычислительные. Для систем небольших размеров или с небольшим числом итераций это сокращает время выполнения алгоритма при использовании одного GPU. Для систем больших размерностей, сокращение времени выполнения в сравнении с CG, возможно только при разбиении матрицы на подобласти достаточно малой размерности, при котором уменьшение коммуникаций перекрывают увеличенные вычислительные затраты.

Предложенные блочные алгоритмы, помимо сокращения времени выполнения, позволяют решать системы линейных уравнений и большего порядка, для которых не обеспечиваются необходимые ресурсы памяти одним GPU или вычислительным узлом. При этом, конвейерный блочный алгоритм сокращает общее время выполнения за счет уменьшения точек синхронизации и объединения коммуникаций в одно сообщение.

Литература

1. Agullo E., Giraud L., Guermouche A., Roman J. Parallel hierarchical hybrid linear solvers for emerging computing platforms // Comptes Rendus Mecanique. 2011. Vol. 333. P. 96–103. DOI: 10.1016/j.crme.2010.11.005.
2. Gaidamour J., Henon P. A parallel direct/iterative solver based on a Schur complement approach // 11th IEEE International Conference on Computational Science and Engineering (San Paulo, Brazil, July, 16–18, 2008). IEEE. 2008. P. 98–105. DOI: 10.1109/CSE.2008.36.
3. Giraud L., Haidar A., Saad Y. Sparse approximations of the Schur complement for parallel algebraic hybrid solvers in 3D // Numerical Mathematics. 2010. Vol. 3, no. 3. P. 276–294. DOI: 10.4208/nmtma.2010.33.2.
4. Rajamanickam S., Boman E.G., Heroux M.A. ShyLU: A Hybrid-Hybrid Solver for Multicore Platforms // IEEE 26th International Parallel and Distributed Processing Symposium (Shanghai, China, May, 21–25, 2012). 2012. P. 631–643. DOI: 10.1109/IPDPS.2012.64.
5. Yamazaki I., Rajamanickam S., Boman E., Hoemmen M., Heroux M., Tomov S. Domain decomposition preconditioners for communication-avoiding Krylov methods on a hybrid CPU/GPU cluster // Proceedings of International Conference for High Performance Computing, Networking, Storage and Analysis (SC14). 2014. P. 933–944. DOI: 10.1109/SC.2014.81.
6. Kopysov S., Kuzmin I., Nedozhogin N., Novikov A., Sagdeeva Y. Scalable hybrid implementation of the Schur complement method for multi-GPU systems // The Journal of Supercomputing. 2014. Vol. 69. P. 81–88. DOI: 10.1007/s11227-014-1209-7.
7. Hestenes M.R., Stiefel E. Methods of Conjugate Gradients for Solving Linear Systems // Journal of Research of the National Bureau of Standards. 1952. Vol. 49. P. 409–436. DOI: 10.6028/jres.049.044.
8. Cornelis J., Cools S., Vanroose W. The Communication-Hiding Conjugate Gradient Method with Deep Pipelines // CoRR abs/1801.04728. 2018. https://arxiv.org/abs/1801.04728.
9. D'Azevedo E.F., Romine C.H. Reducing communcation costs in the conjugate gradient algorithm on distributed memory multiprocessors // Technical Report ORNL/TM-12192, Oak Ridge National Lab, 1992. DOI: 10.2172/10176473.
10. Wilkinson J.H., Reinsch C. Linear Algebra. Springer, Berlin, Heidelberg, 1971. 441 p. DOI: 10.1007/978-3-662-39778-7
11. Chronopoulos A.T., Gear C.W. s-step iterative methods for symmetric linear systems // Journal of Computational and Applied Mathematics. 1989. Vol. 25, no. 2. P. 153–168. DOI: 10.1016/0377-0427(89)90045-9.
12. Ghysels P., Vanroose W. Hiding global synchronization latency in the preconditioned Conjugate Gradient algorithm // Parallel Computing. 2014. Vol. 40, no. 7. P. 224–238. DOI: 10.1016/j.parco.2013.06.001.
13. Gropp W. Update on libraries for blue waters. Bordeaux. France. 2010. URL: http://jointlab.ncsa.illinois.edu/events/workshop3/pdf/presentations/Gropp-Update-on-Libraries.pdf (дата обращения: 07.11.2019).
14. Кадыров И.Р., Копысов С.П., Новиков А.К. Разделение триангулированной многосвязной области на подобласти без ветвления внутренних границ // Ученые записки Казан-

ского университета. Серия: Физико-математические науки. 2018. Т. 160. № 3. С. 544–560. DOI: 10.26907/2541-7746.

Недожогин Никита Сергеевич, ст. преподаватель, кафедра вычислительной механики, Удмуртский государственный университет (Ижевск, Российская Федерация)

Копысов Сергей Петрович, д.ф.-м.н., профессор, кафедра вычислительной механики, Удмуртский государственный университет (Ижевск, Российская Федерация)

Новиков Александр Константинович, к.ф.-м.н., доцент, кафедра вычислительной механики, Удмуртский государственный университет (Ижевск, Российская Федерация)

DOI: 10.14529/cmse200203

PARALLEL SOLVING OF LINEAR EQUATIONS SYSTEMS ON HYBRID ARCHITECTURE CPU+GPU

© 2020 N.S. Nedozhogin, S.P. Kopysov, A.K. Novikov

Udmurt State University (Universitetskaya 1, Izhevsk, 426034 Russia)

E-mail: nedozhogin07@gmail.com, s.kopysov@gmail.com, sc_work@mail.ru

Received: 29.01.2020

The article discusses the parallel implementation of solving systems of linear algebraic equations on computational nodes containing a central processing unit (CPU) and graphic accelerators (GPU). The performance of parallel algorithms for the classical conjugate gradient method schemes when using the CPU and GPU together is significantly limited by the synchronization points. The article investigates the pipeline version of the conjugate gradient method with one synchronization point, the possibility of asynchronous calculations, load balancing between the CPU and GPU when solving the large linear systems. Numerical experiments were carried out on test matrices and computational nodes of different performance of a heterogeneous cluster, which allowed us to estimate the contribution of communication costs. The algorithms are implemented with the joint use of technologies: MPI, OpenMP and CUDA. The proposed algorithms, in addition to reducing the execution time, allow solving large linear systems, for which there are not enough memory resources of one GPU or a computing node. At the same time, block algorithm with the pipelining decreases the total execution time by reducing synchronization points and aggregating some messages in one.

Keywords: parallel calculations, the method of conjugate gradients, reduction of communications.

FOR CITATION

Nedozhogin N.S., Kopysov S.P., Novikov A.K. Parallel Solving of Linear Equations Systems on Hybrid Architecture CPU+GPU. *Bulletin of the South Ural State University. Series: Computational Mathematics and Software Engineering.* 2020. Vol. 9, no. 2. P. 40–54. (in Russian) DOI: 10.14529/cmse200203.

References

1. Agullo E., Giraud L., Guermouche A., Roman J. Parallel hierarchical hybrid linear solvers for emerging computing platforms. Comptes Rendus Mecanique. 2011. Vol. 333. P. 96–103. DOI: 10.1016/j.crme.2010.11.005.

2. Gaidamour J., Henon P. A parallel direct/iterative solver based on a Schur complement

approach. 11th IEEE International Conference on Computational Science and Engineering (San Paulo, Brazil, July, 16–18, 2008). IEEE. 2008. P. 98–105. DOI: 10.1109/CSE.2008.36.

3. Giraud L., Haidar A., Saad Y. Sparse approximations of the Schur complement for parallel algebraic hybrid solvers in 3D. Numerical Mathematics. 2010. Vol. 3, no. 3. P. 276–294. DOI: 10.4208/nmtma.2010.33.2.

4. Rajamanickam S., Boman E.G., Heroux M.A. ShyLU: A Hybrid-Hybrid Solver for Multicore Platforms. IEEE 26th International Parallel and Distributed Processing Symposium (Shanghai, China, May, 21–25, 2012). 2012. P. 631–643. DOI: 10.1109/IPDPS.2012.64.

5. Yamazaki I., Rajamanickam S., Boman E., Hoemmen M., Heroux M., Tomov S. Domain decomposition preconditioners for communication-avoiding Krylov methods on a hybrid CPU/GPU cluster. Proceedings of International Conference for High Performance Computing, Networking, Storage and Analysis (SC14). 2014. P. 933–944. DOI: 10.1109/SC.2014.81.

6. Kopysov S., Kuzmin I., Nedozhogin N., Novikov A., Sagdeeva Y. Scalable hybrid implementation of the Schur complement method for multi-GPU systems. The Journal of Supercomputing. 2014. Vol. 69. P. 81–88. DOI: 10.1007/s11227-014-1209-7.

7. Hestenes M.R., Stiefel E. Methods of Conjugate Gradients for Solving Linear Systems. Journal of Research of the National Bureau of Standards. 1952. Vol. 49. P. 409—436. DOI: 10.6028/jres.049.044.

8. Cornelis J., Cools S., Vanroose W. The Communication-Hiding Conjugate Gradient Method with Deep Pipelines. CoRR abs/1801.04728. 2018. https://arxiv.org/abs/1801.04728.

9. D'Azevedo E.F., Romine C.H. Reducing communcation costs in the conjugate gradient algorithm on distributed memory multiprocessors. Technical Report ORNL/TM-12192, Oak Ridge National Lab, 1992. DOI: 10.2172/10176473.

10. Wilkinson J.H., Reinsch C. Linear Algebra. Springer, Berlin, Heidelberg, 1971. 441 p. DOI: 10.1007/978-3-662-39778-7

11. Chronopoulos A.T., Gear C.W. s-step iterative methods for symmetric linear systems. Journal of Computational and Applied Mathematics. 1989. Vol. 25, no. 2. P. 153–168. DOI: 10.1016/0377-0427(89)90045-9.

12. Ghysels P., Vanroose W. Hiding global synchronization latency in the preconditioned Conjugate Gradient algorithm. Parallel Computing. 2014. Vol. 40, no. 7. P. 224–238. DOI: 10.1016/j.parco.2013.06.001.

13. Gropp W. Update on libraries for blue waters. Bordeaux. France. 2010. Available at: http://jointlab.ncsa.illinois.edu/events/workshop3/pdf/presentations/Gropp-Update-on-Libraries.pdf (accessed: 07.11.2019).

14. Kadyrov I.R., Kopysov S.P., Novikov A.K. Partitioning of triangulated multiply connected domain into subdomains without branching of inner boundaries. Uchenye Zapiski Kazanskogo Universiteta. Seriya Fiziko-Matematicheskie Nauki. 2018. Vol. 160, no. 3. P. 544–560. (in Russian) DOI: 10.26907/2541-7746.