Российская академия наук
Суперкомпьютерный консорциум университетов России

ПАРАЛЛЕЛЬНЫЕ ВЫЧИСЛИТЕЛЬНЫЕ ТЕХНОЛОГИИ (ПаВТ'2020)

Короткие статьи и описания плакатов

г. Пермь, 31 марта – 2 апреля 2020 г.

Челябинск,
Издательский центр ЮУрГУ
2020

УДК 004.75
П 18

Параллельные вычислительные технологии – XIV международная конференция, ПаВТ'2020, г. Пермь, 31 марта – 2 апреля 2020 г. Короткие статьи и описания плакатов. Челябинск: Издательский центр ЮУрГУ, 2020. 340 с.

ISBN 978-5-696-05111-6

https://doi.org/10.14529/pct2020

Данный сборник содержит статьи, включенные в программу Международной научной конференции «Параллельные вычислительные технологии 2020». Конференция проходила 31 марта – 2 апреля 2020 года в Пермском национальном исследовательском политехническом университете (г. Пермь). Подробную информацию о конференции можно найти в сети Интернет по адресу
http://agora.guru.ru/pavt.

Отпечатано с авторских оригиналов.

Одобрено Советом Высшей школы электроники и компьютерных наук ЮУрГУ

Рецензент:
В.В. Воеводин, член-корреспондент РАН

Ответственные за выпуск:
Л.Б. Соколинский, доктор физ.-мат. наук,
Я.А. Краева

Конференция проводится при поддержке
Российского фонда фундаментальных исследований

ISBN 978-5-696-05111-6

Параллельное решение линейных систем уравнений при совместном использовании CPU и GPU *

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

ИМИТиФ ФГБОУ ВО "Удмуртский государственный университет"

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 предлагается конвейерный вариант метода сопряженных градиентов с одной точкой синхронизации, возможностью асинхронных вычислений, распределения нагрузки между CPU и GPU при решении систем уравнений большой размерности.

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

1. Введение

Одной из трудоёмких операций в численных методах является параллельное решение систем линейных алгебраических уравнений (СЛАУ). В настоящее время предложено много подходов для достижения высокой производительности и масштабируемости 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 в 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, связанных с синхронизацией вычислений.

Конвейерные методы подпространства Крылова [8] со скрытыми коммуникациями обеспечивают высокую параллельную масштабируемость за счет перекрытия глобальных коммуникаций с вычислениями, выполняющихся матрично-векторных операций и скалярных произведений векторов.

Первые работы по сокращению коммуникаций были связаны: с вариантом метода сопряженных градиентов (CG) с одной точкой коммуникации на каждой итерации [9], трехчленные рекуррентные соотношения CG [10].

Следующим этапом развития явилось появление s-шаговых методов из подпространств

*Работа выполнена при финансовой поддержке РФФИ (грант 17-01-00402а).

Крылова [11], в которых итерационный процесс в s-блоке использует различные базисы подпространств Крылова, в результате чего удается 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 вариант CG, использующий вспомогательные вектора и перенос последовательной зависимости между вычислением матрично-векторного произведения и скалярными произведениями векторов. В данном методе латентность коммуникаций заменяется дополнительными вычислениями.

2. Конвейерный вариант метода сопряженных градиентов

Рассмотрим конвейерный вариант метода сопряженных градиентов, который математически эквивалент классической форме предобусловленного метода CG и имеет туже скорость сходимости.

Алгоритм 1: Конвейерный алгоритм метода CGwO

```
1  r = b − Ax;
2  u = M^{-1} r;
3  w = Au;
4  γ1 = (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 = γ1/γ0 ;
11     α = γ1/(δ − βγ1/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     γ0 = γ1;
21     γ1 = (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,увеличивается число триад до восьми в алгоритме, в отличие от трех для классического варианта и четырех в [11]. В этом случае, возможно параллельные вычисление триад и двух скалярных произведений в начале итерационного процесса и одно обращение к памяти.

Представленный в данной работе конвейерный вариант CG, может быть использован с

любым предобуславливателем. Существуют два способа организации вычислений в предобусловленном конвейерном CG, обеспечивающих компромисс между масштабируемостью и общим числом операций [13].

Таким образом, конвейерная схема CG, характеризуется другим порядком вычислений, наличием глобальной коммуникации, которая может перекрываться с локальными вычислениями, такими как матрично-векторное произведение и операции с предобуславливателем, возможностью организации асинхронных коммуникаций.

Выполнено сравнение двух вариантов метода сопряженных градиентов: классической схемы и конвейерной.

В численных экспериментах (см. таблица 1) представлены результаты сравнения времени выполнения последовательного варианта классического CG и конвейерной схемы CGwO (Алгоритм 1), исполняемых на CPU и GPU. Отметим, что в вариантах для GPU реализовывалось совместное вычисления всех скалярных произведений векторов в одном месте, независимо друг от друга. Для этого, при запуске ядра GPU, задавалась размерность `Grid` иерархии нитей CUDA в двумерном виде: 3 набора блоков, каждый для выполнения вычислений над своей парой векторов. Это позволяет 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 уравнений малой размерности время решения на CPU по классической схеме CG значительно меньше времени выполнения GPU при одном и том же числе итераций #. 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.

3. 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.

Исследуем варианты совместного использования CPU, оперативная память которого позволяет размещать блоки матриц достаточно большого размера и нескольких графических ускорителей со сравнительно небольшим объемом доступной памяти.

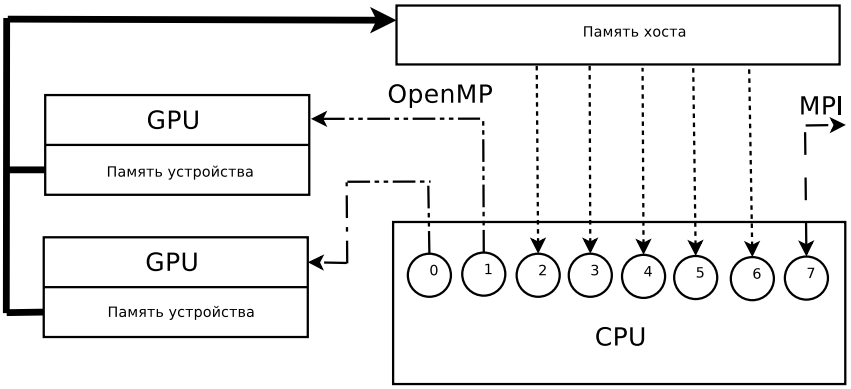

Рис. 1. Гетерогенный вычислительный узел CPU+GPU

Для решения СЛАУ на нескольких GPU построим блочный алгоритм конвейерного метода сопряжённых 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 рис. 1 представлена схема организации параллельных вычислений на гетерогенном вычислительном узле. Для примера рассмотрим узел, содержащий центральный восьмиядерный процессор и два графических ускорителя. Число OpenMP нитей процессора выбирается по числу доступных ядер CPU. Первые две OpenMP-нити 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.

3.1. Разделение матрицы

Для разделения матриц A на блоки, построим граф $G_A(V, E)$, где $V = \{i\}$ – множество вершин, связанных с строчным индексом матрицы (число вершин равно числу строк матрицы A); $E = \{(i, j)\}$ – множество рёбер. 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 необходимости обмена 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 сопряжённых 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.

3.2. Блочный конвейерный алгоритм

Используя блочное разделение матрицы, векторов выполним распределение блоков матрицы на имеющиеся CPU и GPU. Число и размеры блоков позволяет распределять нагрузку в соответствии производительности, 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 Алгоритма 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 Копирование w_h^b на $\texttt{GPU}_k$;
6 $w_k^b = w_k^b + w_h^b$;
7 $m = M^{-1}w$; Сборка векторов m_k^b;
8 $\gamma_{1k} = (r_k, u_k)$; $\delta_k = (w_k, u_k)$; Сборка $\delta = \sum_k \delta_k; \gamma_1 = \sum_k \gamma_{1k}$;
9 $j = 0$;
while $||r||_2/||b||_2 > \varepsilon$ **do**
10 $n_k^i = A_k^{[i_k,i_k]} \cdot m_k^i + A_k^{[i_k,b_k]} \cdot m_k^b$; $n_h^b = \sum_{l=1,l\neq k}^{l\leqslant d} A_k^{[b_k,b_l]} \cdot m_k^b$;
11 $n_k^b = A_k^{[b_k,b_k]} \cdot m_k^b + A_k^{[b_k,i_k]} \cdot m_k^i$; Копирование n_h^b на GPU_k;
12
13 $n_k^b = n_k^b + n_h^b$; $\beta = ((j = 0)\,?\,0 : \gamma_1/\gamma_0)$;
14 $z = n + \beta z$; $\alpha = \gamma_1/(\delta - \beta\gamma_1/\alpha)$;
15 $w = w - \alpha z$;
16 $q = m + \beta q$;
17 $s = w + \beta s$; Сборка векторов w_k^b;
18 $p = u + \beta p$;
19 $x = x + \alpha p$;
20 $r = r - \alpha s$; Сборка векторов m_k^b;
21 $u = u + \alpha q$;
22 $m = M^{-1}w$; Сборка $\delta = \sum_k \delta_k; \gamma_1 = \sum_k \gamma_{1k}$;
23 $\gamma_0 = \gamma_1$;
24 $\gamma_{1k} = (r_k, u_k)$; $\delta_k = (w_k, u_k)$;
25
26 $j = j + 1$;

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 выполняющим 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 Алгоритмом 1 был перенесён шаг предобуславлива-

ния (строка 7 в строку 22). Это сделано для того, что бы совместить векторные операции на исполняющем устройстве и сборку вектора правых частей для выполнения матрично-векторного умножения в предобуславливании. В строке 13 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 Алгоритма 2 проводились при различных вариантов конфигурации вычислительных узлов, содержащих несколько CPU и GPU. В самом общем случае организации параллельных вычислений на нескольких гетерогенных вычислительных узлах, содержащих один и более CPU и несколько GPU реализуется 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 доступных ядер CPU. Восьмой поток OpenMP отвечает за коммуникациями между различными вычислительными узлами (с помощью технологии MPI, сборка векторов с помощью функции `Allgatherv`, сложение скаляров `Allreduce`) и различными GPU. В Алгоритме 2 операции выполняемые этим потоком представлены справа. Нулевой и первый потоки OpenMP связываются с одним из доступных GPU-устройств и отвечают за пересылку данными 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 данными между GPU-CPU (вызовы функций асинхронного копирования) и участвует с восьмым потоком в операции матрично-векторного умножения на граничных узлах (строки 4, 10 правая колонка). Оставшиеся параллельные потоки (второй - седьмой) производят вычисления как отдельное исполняющее устройство для своего блока матрицы. Операции, выполняемые исполняющими устройствами в Алгоритме 2 приведены слева.

Применение предобуславливателя строки 2, 7 и 22 подразумевает использование блочного матрично-векторного произведения вида (1) рассмотренного выше.

4. Численные эксперименты

Вычислительные эксперименты выполнялись на кластере «Уран» на вычислительных узлах (ВУ) нескольких типов 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-и ядерных процессора Intel® Xeon® E5-2660 (2.2 GHz), оперативная память 96 GB, кэш-память 2 x 20 MB Level 2 cache и 8 GPU Tesla M2090 (6 ГБ Global Memory), коммуникационная сеть 1 Гбит/с Ethernet.

- раздел «tesla-v100»: два 18-и ядерных процессора Intel(R) Xeon(R) Gold 6240 CPU @ 2.60GHz; оперативная память 384 GB; 8 GPU Nvidia Tesla V100 (32 ГБ Global Memory).

- раздел «tesla[21-30]»: 10 узлов по два 6-и ядерных процессора Intel® Xeon® X5675 (3.07GHz), оперативная память 192 GB, кэш-память 2 x 12 MB Level 2 cache и 8 GPU Tesla M2090 (6 ГБ Global Memory), коммуникационная сеть Infiniband 20 Гбит/с.

- раздел «tesla[33-45]»: 13 узлов по два 8-и ядерных процессора Intel® Xeon® E5-2660 (2.2 GHz), оперативная память 96 GB, кэш-память 2 x 20 MB Level 2 cache и 8 GPU Tesla M2090 (6 ГБ Global Memory), коммуникационная сеть Infiniband 20 Гбит/с.

- раздел «tesla[48-52]»: 5 узлов по два 8-и ядерных процессора Intel® Xeon® E5-2650 (2.6 GHz), оперативная память 64 GB, кэш-память 2 x 20 MB Level 2 cache и 3 GPU

Tesla K40m (12 ГБ Global Memory), коммуникационная сеть Infiniband 20 Гбит/с.

Таблица 1. Время решения методом CG на CPU и GPU, сек.

	N	nnz	# итераций	ВУ	Время, с	
					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	3,97E-01	1,33E-01
				K40m	3,69E-01	**1,29E-01**
Pres_Poisson	14822	715804	661	M2090	4,98E-01	3,13E-01
				K40m	4,16E-01	**2,74E-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	2,04E+01	2,56E+01
				K40m	**8,17E+00**	1,17E+01
G3_circuit	1585478	7660826	592	M2090	3,43E+00	4,32E+00
				K40m	**1,94E+00**	2,92E+00

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 в таблицах выделены лучшее время решения системы для каждой матрицы.

Конвейерный алгоритм CGwO показал сокращение времени выполнения на небольших СЛАУ для которых характерна небольшая вычислительная нагрузка, за счёт чего сокращение коммуникаций обеспечивает меньшие временные затраты. Отметим, что классический 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 немного больше варианта блочного CG, что связано с выполнением дополнительных векторных операций, которые не перекрываются сокращением коммуникаций. С уменьшением числа итераций, например для решения большой системы с **G3_circuit**) при примерно равном числе уравнений с **thermal2** время выполнения алгоритмов CG и CgwO на одном GPU возрастает незначительно. Для системы (**thermal2** и **G3_circuit**) увеличение затрат становится более явным.

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

Значительное влияние характеристик сети на производительность блочных методов

можно увидеть в таблицах 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 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 таблицах 2 и 3 и любая строка в табл. 1). Использование коммуникационной сети Infiniband 20 Гбит/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 таблице 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. В табл. 1 видно, что использовании конвейерного алгоритма для матрицы **Muu** даёт 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 за счёт того, что сокращения коммуникаций не перекрывает дополнительные векторные операции (таблица 1). Использование блочных алгоритмов сокращает вычислительную нагрузку на один GPU, тем самым позволяет при некотором разделении матрицы и векторов (размерности около 200000 на каждом GPU) получить ускорении вычислений. Например, для системы **Fault_639** порядка 638802 достигается небольшое ускорение при разделении на три блока, для системы **Inline_1** на два блока.

Анализ результатов показал, что значительное время занимает синхронизация нескольких GPU в рамках одного вычислительного узла. В дальнейшем планируется исследовать механизмы синхронизации и суммирования векторов в рамках одного узла, для сокращения вычислений. При вычислении на нескольких вычислительных узлах, на общую производительность алгоритма влияют коммуникационные затраты. 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 сопряжёнными градиентами на одном узле на матрицах **Inline_1**, **Fault_639** за счёт сокращения вычислительной нагрузки на одном GPU.

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

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 счёт уменьшения точек синхронизации и объединения коммуникаций в одно сообщение.

Таблица 2. Время решения методом блочным CG на CPU/GPU, сек.

Матрица/ВУ	BlockCG/число блоков						
	2		3		8		
	#CPU/#GPU						
	1 / 2	2/1	1/3	3/1	1/8	4/2	8/1
Plat362/M2090	2,22E+00	**1,55E+00**					
/K40m	2,39E+00	1,92E+00	2,90E+00	1,56E+00			
1138_bus/M2090	2,07E+00	1,84E+00					
/K40m	2,21E+00	1,90E+00	2,92E+00	1,85E+00			
/debug	2,02E+00	**1,25E+01**					
Muu/M2090	8,71E-01	6,12E-01			2,7E+00	1,84E+00	7,4E-01
/K40m	1,04E+00	6,59E-01	1,03E+00	**5,64E-01**		6,97E+00	
Kuu/M2090	1,60E+00	**1,30E+00**			5,2E+00	3,7E+00	1,41E+00
/K40m	1,59E+00	1,29E+00	2,00E+00	1,36E+00		1,94E+00	
Pres_Poisson/M2090	2,18E+00	**1,55E+00**			7,8E+00	3,15E+00	2,0E+00
/K40m	2,13E+00	1,60E+00	2,68E+00	1,66E+00		2,3E+00	
Inline_1/M2090	4,37E+01	3,76E+01					
/K40m	3,29E+01	2,66E+01	3,50E+01	**2,20E+01**			
Fault_639/M2090	3,69E+01	3,18E+01			7,18E+01		2,09E+01
/K40m	2,72E+01	2,20E+01	2,96E+01	**1,98E+01**		2,89E+01	
/debug	2,75E+01	9,38E+01					
thermal2/M2090	3,94E+01	1,46E+01			4,92E+01	1,63E+01	1,15E+01
/K40m	1,55E+01	1,18E+01	1,88E+01	**1,00E+01**			
G3_circuit/M2090	5,97E+00	4,27E+00			1,61E+01	4,82E+00	**3,26E+00**
/K40m	5,33E+00	4,04E+00	6,27E+00	3,510E+00		4,06E+00	

Таблица 3. Время решения блочным конвейерным CGwO на CPU/GPU, сек.

Матрица / ВУ	BlockCGwO/число блоков						
	2		3		8		
	#CPU/#GPU						
	1/2	2/1	1/3	3/1	1/8	4/2	8/1
Plat362/M2090	1,96E+00	**1,22E+00**					
/K40m	1,99E+00	1,28E+00	2,72E+00	1,31E+00			
1138_bus/M2090	1,56E+00	**9,28E-01**					
/K40m	1,55E+00	1,03E+00	2,19E+00	1,04E+00			
/debug	1,56E+00	5,36E+00					
Muu/M2090	6,79E-01	2,29E-01			2,8E+00	5,8E-01	**1,8E-01**
/K40m	5,65E-01	2,89E-01	6,69E-01	2,88E-01			
Kuu/M2090	1,11E+00	**6,43E-01**			4,3E+00	1.23E+00	7,9E-01
/K40m	1,15E+00	6,81E-01	1,54E+00	7,95E-01		1,26E+00	
Pres_Poisson/M2090	1,60E+00	9,57E-01			6,7E+00	1.68E+00	**9,5E-01**
/K40m	1,56E+00	1,02E+00	2,13E+00	1,19E+00		1,76E+00	
Inline_1/M2090	4,57E+01	3,65E+01					
/K40m	3,29E+01	2,48E+01	3,46E+01	**1,97E+01**			
Fault_639/M2090	3,99E+01	3,03E+01			7,73E+01		**1,69E+01**
/K40m	2,82E+01	2,05E+01	3,19E+01	1,78E+01		2,13E+01	
/debug	2,80E+01	5,25E+01					
thermal2/M2090	3,53E+01	1,49E+01			5,92E+01	1,54E+00	**8,92E+00**
/K40m	1,71E+01	1,18E+01	1,99E+01	9,85E+00			
G3_circuit/M2090	7,17E+00	3,99E+00			1,80E+01	4,78E+00	**2,69E+00**
/K40m	5,95E+00	3,27E+00	6,61E+00	2,77E+00		3,86E+00	

5. 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 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. Предложен конвейерный вариант метода сопряженных градиентов с одной точкой синхронизации, возможностью асинхронных вычислений, распределения нагрузки между несколькими GPU, находящимися как над одном вычислительном узле, так и для кластера GPU при решении систем уравнений большой размерности. В дальнейшем предполагается для повышения эффективности вычислений провести исследования распределение не только коммуникационной нагрузки, но и вычислительной нагрузки между CPU и GPU.

Литература

1. Agullo E., Giraud L., Guermouche A., Roman J. Parallel hierarchical hybrid linear solvers for emerging computing platforms // Comptes Rendus Mecanique. 2011. V. 333. P. 96–103.

2. Gaidamour J., Henon P. A parallel direct/iterative solver based on a Schur complement approach // In: IEEE 11th International Conference on Computational Science and Engineering, San Paulo. 2008. P. 98–105.

3. Giraud L., Haidar A., Saad Y. Sparse approximations of the Schur complement for parallel algebraic hybrid solvers in 3D // Numerical Mathematics. 2010. V. 3. P. 276–294.

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, 2012. P. 631-643.

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 // In Proceedings of International Conference for High Performance Computing, Networking, Storage and Analysis (SC14). 2014. P. 933-944.

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. v. 69. P. 81-88.

7. Hestenes M. R., Stiefel E. Methods of Conjugate Gradients for Solving Linear Systems // Journal of Research of the National Bureau of Standards. 1952. V. 49. P. 409–436.

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

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.

10. Уилкинсон Дж. Х., Райнш К. Справочник алгоритмов на языке Алгол. Линейная Алгебра. М: Машиностроение, 1976.

11. Chronopoulos A.T., Gear C.W. s-step iterative methods for symmetric linear systems // Journal of Computational and Applied Mathematics. 1989. V. 25. № 2. P. 153-168.

12. Ghysels P., Vanroose W. Hiding global synchronization latency in the preconditioned Conjugate Gradient algorithm // Parallel Computing. 2014. V. 40. № 7. P. 224–238.

13. Gropp W. Update on libraries for blue waters, Bordeaux, France, 2010. http://jointlab.ncsa.illinois.edu/events/workshop3/pdf/presentations/Gropp-Update-on-Libraries.pdf>.

14. Кадыров И. Р., Копысов С. П., Новиков А. К. Разделение триангулированной многосвязной области на подобласти без ветвления внутренних границ // Ученые записки Казанского университета. Серия: Физико-математические науки. 2018. Т. 160. Кн. 3. С. 544-560.

Индекс по фамилиям

Содержание

Короткие статьи

Плакаты