Перекрытие коммуникаций и вычислений в итерационных методах решения систем линейных уравнений на GPU*

В.А. Платонов <soh@ispras.ru>
А.В. Монаков <amonakov@ispras.ru>
Институт системного программирования РАН, 109004, Россия, г. Москва, ул. А. Солженицына, 25

Аннотация. Методы подпространства Крылова, такие как метод сопряжённых градиентов и стабилизированный метод бисопряжённых градиентов, давно используются для решения симметричных и несимметричных систем линейных алгебраических уравнений. Это находит широкое применение при численном решении дифференциальных уравнений, которые возникают, например, в задачах вычислительной физики. Однако при увеличении размеров расчетной сетки и, соответственно, количества вычислительных процессов значительную часть времени работы могут занимать коммуникации, во время которых расчёты простаивают. Это происходит из-за того, что в оригинальных формулировках методов результат скалярного произведения, которое требует редукции, требуется уже на следующем шаге метода, что приводит к барьерной синхронизации всех потоков. При значительном количестве итераций это может привести к деградации производительности. В статье рассматривается использование альтернативных формулировок методов подпространства Крылова, позволяющих перекрыть часть вычислений и параллельных коммуникаций, часто за счет увеличения объема вычислений. Нами предложены собственные реализации этих подходов для использования гибридного решателя с графическими ускорителями, использующими технологию CUDA, в рамках программного пакета OpenFOAM, а также описаны особенности их переноса на акселераторы. Для дальнейшей оптимизации используются асинхронные коллективные операции, предоставленные стандартом межпроцессного взаимодействия MPI-3, которые позволяют избавиться от барьерной синхронизации и снизить латентность операций обмена. Представлены результаты тестирования нашего подхода на одной из станадартных задач пакета OpenFOAM с расчётными сетками в 2 и 4 миллиона ячеек с использованием нескольких графических ускорителей.

Ключевые слова: метод сопряженных градиентов, стабилизированный метод бисопряжённых градиентов, AINV-предобуславливание, OpenFOAM, GPU, MPI

* Работа поддержана грантом РФФИ 13-07-12102

DOI: 10.15514/ISPRAS-2016-28(1)-5

Для цитирования: Платонов В.А., Монаков А.В. Перекрытие коммуникаций и вычислений в итерационных методах решения систем линейных уравнений на GPU. Труды ИСП РАН, том 28, вып. 1, 2016 г., стр. 81-92. DOI: 10.15514/ISPRAS-2016-28(1)-5

1. Введение

В настоящее время большинство высокопроизводительных гетерогенных систем дополняется акселераторами, чья специализированная архитектура может давать существенный прирост производительности на некоторых классах задач. Наиболее популярным типом подобных ускорителей являются графические акселераторы (GPU), архитектура которых оптимизирована для задач с высоким параллелизмом.

Решение задач вычислительной гидродинамики в конечном итоге сводится к решению нескольких систем линейных алгебраических уравнений, при этом коэффициенты системы представляют собой разреженную матрицу. Наиболее популярные численные методы для решения подобных систем – это методы подпространства Крылов, такие как метод сопряжённых градиентов или стабилизированнный метод бисопряжённых градиентов. Сам по себе перенос вычислений на графический ускоритель способен дать существенный прирост производительности [4], однако при расчёте на большом количестве вычислительных узлов заметную роль начинают играть коммуникации между процессами. Любое вычисление скалярного произведения подразумевает обмен частичными суммами между всеми процессами, что сопряжено с необходимостью синхронизации расчётов на всех процессах. Один из подходов к решению этой проблемы состоит в том, чтобы перекрывать коммуникации и вычисления, однако оригинальные формулировки методов подпространства Крылова строго последовательны и не предоставляют такой возможности. В данной работе рассматриваются альтернативные формулировки этих методов, которые позволяют перекрытие за счёт увеличения требуемой памяти и количества вычислений. Реализация выполнена в рамках открытого пакета для задач вычислительной гидродинамики OpenFOAM [1]. Кроме того, используется ранее разработанная нами библиотека, содержащая все необходимые операции (умножение матрицы на вектор, скалярное произведение, линейные комбинации) для реализации методов и использующая автоматическую настройку для повышения производительности на GPU [3,4].

2. Классические методы подпространства Крылова

В OpenFOAM используется классический вариант метода сопряженных градиентов с предобуславливанием [2] для системы $\boldsymbol{A}\,\boldsymbol{x} = \boldsymbol{b}$:

1: $\boldsymbol{r} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}$
2: $\boldsymbol{p} = \boldsymbol{0}$

3: do
4: $\boldsymbol{s} = \boldsymbol{M}^{-1}\boldsymbol{r}$
5: $\delta_i = (\boldsymbol{s}, \boldsymbol{r})$
6: $\beta = \delta_i / \delta_{i-1}$
7: $\boldsymbol{p} = \boldsymbol{s} + \beta\boldsymbol{p}$
8: $\boldsymbol{s} = \boldsymbol{A}\,\boldsymbol{p}$
9: $\gamma = (\boldsymbol{s}, \boldsymbol{p})$
10: $\alpha = \delta_i / \gamma$
11: $\boldsymbol{x} = \boldsymbol{x} + \alpha\boldsymbol{p}$
12: $\boldsymbol{r} = \boldsymbol{r} - \alpha\boldsymbol{s}$
13: while (!stopCondition(|***r***|, ++i))

(где $\boldsymbol{M}^{-1}$ — предобуславливатель, ***r*** — вектор невязки).

Классическая формулировка требует хранить 6 векторов, а за одну итерацию цикла происходит два умножения матрицы на вектор, 3 линейных (вида ax+y) комбинации и 3 скалярных произведения (вместе с расчётом невязки).

Классический вариант стабилизированного метода бисопряжённых градиентов с предобуславливанием, использующийся в OpenFOAM для решения системы $\boldsymbol{A}\,\boldsymbol{x} = \boldsymbol{b}$:

1: $\boldsymbol{r} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}$
2: $\boldsymbol{p} = 0$
3: $\boldsymbol{rw} = \boldsymbol{r}$
4: do
5: $\rho_i = \rho_{i-1}$
6: $\rho_i = (\boldsymbol{r}, \boldsymbol{rw})$
7: $\beta = (\alpha\,\rho_i)/(\omega\rho_{-1})$
8: $\boldsymbol{p} = \boldsymbol{r} + \beta\boldsymbol{p} - \beta\omega\boldsymbol{v}$
9: $\boldsymbol{ph} = \boldsymbol{M}^{-1}\boldsymbol{p}$
10: $\boldsymbol{v} = \boldsymbol{A}\,\boldsymbol{ph}$
11: $\alpha = \rho_i/(\boldsymbol{rw}, \boldsymbol{v})$
12: $\boldsymbol{s} = \boldsymbol{r} - \alpha\boldsymbol{v}$
13: $\boldsymbol{sh} = \boldsymbol{M}^{-1}\boldsymbol{s}$
14: $\boldsymbol{t} = \boldsymbol{A}\,\boldsymbol{sh}$
15: $\omega = (\boldsymbol{t}, \boldsymbol{s})/(\boldsymbol{t}, \boldsymbol{t})$
16: $\boldsymbol{x} = \boldsymbol{x} + \alpha*\boldsymbol{ph} + \omega*\boldsymbol{sh}$
17: $\boldsymbol{r} = \boldsymbol{s} - \omega*\boldsymbol{t}$
18: while (!stopCondition(|***r***|, ++i))

Классическая формулировка метода требует хранить 9 векторов, а за одну итерацию цикла происходит 4 умножения матрицы на вектор, 6 линейных (вида ax+y) комбинаций и 4 скалярных произведения (вместе с расчётом невязки).

3. Перекрытие MPI-коммуникаций

При параллельной работе OpenFOAM происходит декомпозиция расчётной на сетки на отдельные, в простейшем случае, непересекающиеся подобласти, каждая из которых отдаётся одному из процессов. Таким образом большую часть расчётов можно выполнять независимо. Тем не менее, некоторые параметры счёта требуют информации со всей расчётной области. Если рассматривать непосредственно задачу решения СЛАУ, то переход к многопоточности означает следующее: каждый процесс в качестве матрицы использует диагональный блок оригинальной матрицы (соответствующий подобласти) и свою часть общих векторов решения и правой части. В процессе решения каждый процесс обновляет своё локальное решение и финальным ответом становится объединение всех локальных ответов в один вектор. Соответственно, линейные комбинации над локальными векторами можно производить без взаимодействия с другими процессами. Сложнее обстоит ситуация со скалярными произведениями и умножением матрицы на вектор. В первом случае результатом операции является сумма всех локальных ответов, вызывая общую синхронизацию, что негативно влияет на производительность, так как необходимо пассивно ожидать выполнения операции и получения результатов со всех процессов. Частным случаем скалярного произведения можно считать и подсчёт невязки. В случае умножения матрицы на вектор, локальной операции также недостаточно. Как несложно заметить, в результате декомпозиции, не все элементы оригинальной матрицы попали в одну из локальных матриц. Эти элементы матрицы, называемые граничными коэффициентами, возникают за счёт взаимодействия на границе между подобластями. Таким образом, для получения ответа, процессу необходимо произвести локальное умножение и добавить результат умножения граничных условий на соответствующую часть вектора соседнего (имеющего общую границу с текущим) процесса. Важно отметить, что синхронизация происходит только с соседями, а не со всеми процессами.

Таким образом во время многопроцессорного запуска OpenFOAM, можно выделить три разных типа коммуникаций внутри решателя СЛАУ:

- Обмен граничными коэффициентами при умножении матрицы на вектор
- Редукция результатов скалярного произведения
- Подсчёт невязки

При увеличении количества параллельных процессов, коммуникации начинают играть всё большую роль в производительности системы. Во-первых, повышается стоимость обмена данными. Во-вторых, глобальные синхронизации вызывают простой всех потоков до завершения обмена данными. Эффективным способом борьбы со второй проблемой является перекрытие коммуникаций и вычислений, то есть выполнение расчётов во время асинхронной передачи данных. Например, при умножении матрицы на вектор, обмен граничными коэффициентами можно производить параллельно с локальным произведением, а редукцию невязки выполнять параллельно с выполнением следующей итерации. Сложнее обстоит ситуация со скалярными произведениями: в классических формулировках методов подпространства Крылова применить данный подход не представляется возможным, так как алгоритм строго последователен в том смысле, что результат выполнения любого шага алгоритма требуется на следующем шаге и соответственно все необходимые коммуникации должны быть выполнены к тому моменту. Для гетерогенных систем с акселераторами дополнительная сложность заключается в том, что для запуска редукции необходимо дождаться завершения соответствующего ядра на ускорителе, при этом крайне желательно запустить новые независимые ядра на время коммуникации.

В рамках нашей работы мы использовали асинхронные глобальные редукции, предоставленные стандартом MPI-3 и реализованные в актуальных версиях пакетов OpenMPI и mvapich. Нами были реализованы необходимые абстракции для использования в рамках пакета OpenFOAM.

4. Переформулированные методы подпространства Крылова

Существует несколько реализаций метода сопряженных градиентов, направленных на увеличение перекрытия вычислений и коммуникаций. В них возникает компромисс между степенью перекрытия и количеством дополнительно проводимых вычислений и хранимых векторов. Для использования на GPU нами был использован следующий вариант [5]:

1: $\boldsymbol{r} = \boldsymbol{b} - \boldsymbol{Ax}$
2: $\boldsymbol{u} = \boldsymbol{M}^{-1}\boldsymbol{r}$
3: $\boldsymbol{w} = \boldsymbol{Au}$
4: $\gamma_{-1} = \infty$
5: $\gamma_0 = (\boldsymbol{r}, \boldsymbol{u})$
6: $\delta = (\boldsymbol{w}, \boldsymbol{u})$
7: do
8: $\boldsymbol{m} = \boldsymbol{M}^{-1}\boldsymbol{w}$
9: $\boldsymbol{n} = \boldsymbol{Am}$
10: $\beta = \gamma_i / \gamma_{i-1}$

11: $\alpha = \gamma_i / (\delta - \beta\,\gamma_i / \alpha)$
12: $\boldsymbol{z} = \boldsymbol{n} + \beta\boldsymbol{z}$
13: $\boldsymbol{q} = \boldsymbol{m} + \beta\boldsymbol{q}$
14: $\boldsymbol{s} = \boldsymbol{w} + \beta\boldsymbol{s}$
15: $\boldsymbol{p} = \boldsymbol{u} + \beta\boldsymbol{p}$
16: $\boldsymbol{x} = \boldsymbol{x} + \alpha\boldsymbol{p}$
17: $\boldsymbol{r} = \boldsymbol{r} - \alpha\boldsymbol{s}$
18: $\boldsymbol{u} = \boldsymbol{u} - \alpha\boldsymbol{q}$
19: $\boldsymbol{w} = \boldsymbol{w} - \alpha\boldsymbol{z}$
20: $\gamma_i = (\boldsymbol{r}, \boldsymbol{u})$
21: $\delta = (\boldsymbol{w}, \boldsymbol{u})$
22: while (!stopCondition(|$\boldsymbol{r}$|, ++i))

В переформулированном варианте алгоритма можно выделить важные особенности:

- глобальную редукции невязки и скаляров в строках 20-21 можно выполнить одной асинхронной all-reduce коммуникацией на трехкомпонентном векторе $\{\gamma_i, \delta, |\boldsymbol{r}|\}$, которая будет перекрыта с умножением матрицы и предобуславливанием в строках 8-9
- за счёт удачного расположения операций вычисления в строках 12-22 требуют запуска всего двух вычислительных ядер, что уменьшает количество накладных расходов на запуск ядер, а также выигрывает за счёт переиспользования данных
- обновление граничных коэффициентов при умножении на матрицу можно перекрыть с обновлением векторов $\boldsymbol{q}$, $\boldsymbol{s}$, $\boldsymbol{p}$, $\boldsymbol{x}$, $\boldsymbol{r}$, $\boldsymbol{u}$, если вынести обновление векторов $\boldsymbol{z}$ и $\boldsymbol{w}$ в отдельное ядро
- новая формулировка требует хранить 11 векторов, а за одну итерацию цикла происходит два умножения матрицы на вектор, 8 линейных (вида ax+y) комбинаций и 3 скалярных произведения (вместе с расчётом невязки).

Таким образом, нам удалось серьёзно улучшить эффективность коммуникаций алгоритма, поскольку за счёт их объединения, а также перекрытия с вычислениями удалось избавить от всех точек синхронизации процессов. Ценой этой эффективности стали значительно возросшие требования к памяти, а также количество линейных комбинаций, таким образом для систем с малой стоимостью коммуникаций новый вариант алгоритма может показывать производительность ниже, чем классический метод.

Для стабилизированного метода бисопряжённых градиентов также существует несколько переформулированных вариантов, повышающих эффективность коммуникаций. Первым был предложен вариант с двумя

точками синхронизации [6], затем только с одной точкой синхронизации [7] и наконец, вариант не содержащий явных точек синхронизации [8].

1: $\boldsymbol{r} = \boldsymbol{b} - \boldsymbol{Ax}$

2: $\boldsymbol{rw} = \boldsymbol{r}$

3: $\boldsymbol{z} = \boldsymbol{M}^{-1}\,\boldsymbol{r}$

4: $\boldsymbol{vh} = \boldsymbol{z}$

5: $\rho = (\boldsymbol{r}, \boldsymbol{rw})$

6: $\rho_{i-1} = \rho_i$

7: do

8: $\boldsymbol{v} = \boldsymbol{A}\,\boldsymbol{vh}$

9: $\gamma = (\boldsymbol{v}, \boldsymbol{rw})$

10: $\boldsymbol{s} = \boldsymbol{M}^{-1}\,\boldsymbol{v}$

11: $\alpha = \rho_i / \gamma$

12: $\boldsymbol{th} = \boldsymbol{z} - \alpha\boldsymbol{s}$

13: $\boldsymbol{t} = \boldsymbol{A}\,\boldsymbol{th}$

14: $\boldsymbol{x} = \boldsymbol{x} + \alpha\boldsymbol{vh}$

15: $\boldsymbol{r} = \boldsymbol{r} - \alpha\boldsymbol{v}$

16: $\theta = (\boldsymbol{t}, \boldsymbol{r})$

17: $\varphi = (\boldsymbol{t}, \boldsymbol{t})$

18: $\psi = (\boldsymbol{t}, \boldsymbol{rw})$

19: $\boldsymbol{z} = \boldsymbol{M}^{-1}\,\boldsymbol{t}$

20: $\omega = \theta/\varphi$

21: $\boldsymbol{x} = \boldsymbol{x} + \omega\boldsymbol{th}$

22: $\boldsymbol{r} = \boldsymbol{r} - \omega\boldsymbol{t}$

23: $\rho_{i-1} = \rho_i$

24: $\rho_i = -\omega\psi$

25: $\beta = (\alpha\rho_i) / (\rho_{i-1}\omega)$

26: $\boldsymbol{z} = \boldsymbol{th} - \omega\boldsymbol{z}$

27: $\boldsymbol{vh} = \boldsymbol{z} + \beta(\boldsymbol{vh} - \omega\boldsymbol{s})$

28: while (!stopCondition(|***r***|, ++i))

В переформулированном варианте алгоритма можно выделить важные особенности:

- глобальную редукцию скаляра в строке 9 и невязки в строке 22 можно перекрыть с последующим предобуславливанием в строке 10
- глобальную редукцию в строках 16-18 можно выполнить одной коммуникацией на трехкомпонентном векторе $\{\theta, \varphi, \psi\}$, которая будет перекрыта с последующим предобуславливанием в строке 19
- операции 14-18 и 20-27 возможно объединить в два ядра для уменьшения накладных расходов и уменьшения доступов в память
- новая формулировка требует хранить 9 векторов, а за одну итерацию цикла происходит четыре умножения матрицы на вектор, 8 линейных (вида ax+y) комбинаций и 5 скалярных произведений (вместе с расчётом невязки)

Аналогично варианту с методом сопряжённых градиентов, переформулированный метод значительно повышает качество коммуникаций (все коммуникации перекрываются вычислениями) ценой увеличения количества операций.

5. Результаты тестирования

Описанные оптимизации были реализованы и протестированы в модуле решения линейных систем для проекта FOAM-Extend. Для тестирования производительности использовались гетерогенные узлы с процессорами Xeon E5 2620 (6 ядер, 2.0 GHz) и акселераторами GeForce GTX TITAN. В качестве эталонной процессорной реализации использовались методы PCG и BiCGStab с предобуславливателем DIC. При запуске на центральном процессоре использовались все его физические ядра, по одному потоку на ядро.

На тесте cavity (сетка с 2 млн. ячеек) с приложением icoFoam были получены следующие результаты:

Количество процессоров/ ускорителей	PCG	BiCGStab	GPUPCG	GPUBCGS
1	235	301	125	143
2	113	145	63	75
Ускорение	2.07	2.07	1.98	1.90

На тесте cavity (сетка с 4 млн. ячеек) с приложением icoFoam были получены следующие результаты:

Количество процессоров/ ускорителей	PCG	BiCGStab	GPUPCG	GPUBCGS
1	602	791	290	348
2	303	400	155	187
Ускорение	1.98	1.97	1.87	1.86

Из полученных результатов можно заметить, что текущая реализация позволяет достичь ускорения около 2 раз на одном ускорителе относительно 6-ти ядерного процессора. При этом увеличение размера сетки положительно сказывается на приросте производительности. Проведённые оптимизации коммуникаций позволяют сохранять хорошую масштабируемость при увеличении количества ячеек.

7. Заключение

В данной работе представлено несколько оптимизаций для гибридной реализации метода сопряженных градиентов и стабилизированного метода бисопряжённых градиентов. С предложенными оптимизациями на GPU-акселераторах достигается значительно большая производительность по сравнению с многоядерными процессорами, но только для задач достаточно большого размера. Проведённые оптимизации коммункаций позволяют повысить масштабирования решения. В будущем планируется разработать гибридный вариант для алгебраического многосеточного метода и добавить возможность использовать его в качестве предобуславливателя уже существующих решателей.

Список литературы

[1]. Страница OpenFOAM — http://openfoam.org/
[2]. Y. Saad. Iterative Methods for Sparse Linear Systems. SIAM. 2003.
[3]. А. Монаков, Е. Велесевич, В. Платонов, А. Аветисян. Инструменты анализа и разработки эффективного кода для параллельных архитектур. Труды Института системного программирования РАН, том 26 (выпуск 1), 2014 г.. стр. 357-374. DOI: 10.15514/ISPRAS-2014-26(1)-14
[4]. А.В. Монаков, В.А. Платонов, Оптимизация метода решения линейных систем уравнений в OpenFOAM для платформы MPI + CUDA. Труды Института системного программирования РАН, том 26 (выпуск 3), 2014 г., стр. 91-102. DOI: 10.15514/ISPRAS-2014-26(3)-4
[5]. P. Ghysels, W. Vanroose. Hiding Global Synchronization Latency in the Preconditioned Conjugate Gradient Algorithm. Submitted to Parallel Computing, 2012.

[6]. Thierry Jacques, Laurent Nicolas, Christian Vollaire, 7th International Conference, HPCN Europe 1999 Amsterdam, The Netherlands, April 12–14, 1999 Proceedings, pp 1025-1031
[7]. L. Yang, R. Brent, The improved BiCGStab method for large and sparse unsymmetric linear systems on parallel distributed memory architectures, in: Fifth International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP'02). Proceedings., IEEE ComputerSociety, Los Alamitos, CA, USA, 2002, pp. 324–328. doi:10.1109/ICAPP.2002.1173595
[8]. Boris Krasnopolsky, The reordered BiCGStab method for distributed memory computer systems, Procedia Computer Science, Volume 1, Issue 1, May 2010, Pages 213-218, ISSN 1877-0509, http://dx.doi.org/10.1016/j.procs.2010.04.024.

Overlapping communications and computations in GPU-based iterative linear solvers *

V.A. Platonov <soh@ispras.ru>
A.V. Monakov <amonakov@ispras.ru>
Institute for System Programming of the Russian Academy of Sciences RAS, 25 Alexander Solzhenitsyn Str., Moscow, 109004, Russian Federation

Abstract. Krylov subspace methods such as Conjugate Gradient and Biconjugate Gradient Stabilized methods are well known approaches for solving symmetric and asymmetric systems of linear algebraic equations, such as systems usually arising from partial differential equations in computational mathematics problems, like Navier-Stokes equations in fluid dynamics. With increasing sizes of meshes and numbers of computational nodes, network communications time may become an issue: stalls during global reductions have increasing duration, preventing useful computations. This happens because, in original formulations of methods, computing a dot product requires a global reduce operation, and its value is required on the next step, so each process has to stop until all others reach this point, like in a barrier synchronization. We research alternative formulations of conjugate gradient methods (Preconditioned Conjugate Gradient and BiCGStab) for GPU-based iterative linear system solvers. They allow to have an overlap of parallel computations and communications, at the cost of increased amount of computations and memory requirements. We describe an implementation of our approach for GPU-accelerated hybrid systems in OpenFOAM, an open source framework for computational fluid dynamics. Asynchronous collective communications from MPI-3 parallel programming API are used to avoid full barrier synchronization and reduce latency. Experimental results on 2 and 4 million cases from standard OpenFOAM problems are presented.

Keywords: conjugate gradient method, bicgstab method, AINV preconditioning, OpenFOAM, GPU, MPI.

DOI: 10.15514/ISPRAS-2016-28(1)-5

For citation: Platonov V.A., Monakov A.V. Overlapping communications and computations in GPU-based iterative linear solvers**.** *Trudy ISP RAN/Proc. ISP RAS*, vol. 28, issue 1, 2016, pp. 81-92 (in Russian). DOI: 10.15514/ISPRAS-2016-28(1)-5

* The paper is supported by RFBR grant 13-07-12102

References

[1]. OpenFOAM — http://openfoam.org/

[2]. Y. Saad. Iterative Methods for Sparse Linear Systems. SIAM. 2003.

[3]. A. Monakov, E. Velesevich, V. Platonov, A. Avetisyan. Instrumenty analiza i razrabotki jeffektivnogo koda dlja parallel'nyh arhitektur (Analysis and development tools for efficient programs on parallel architectures). Trudy ISP RAN [Proceedings of ISP RAS], volume 26 (issue 1), 2014, pp. 357-374 (in Russian). DOI: 10.15514/ISPRAS-2014-26(1)-14

[4]. A. Monakov, V. Platonov. Optimizacija metoda reshenija linejnyh sistem uravnenij v OpenFOAM dlja platformy MPI + CUDA (Optimizations for linear solvers in OpenFOAM for MPI + CUDA platform). Trudy ISP RAN [Proceedings of ISP RAS], volume 26 (issue 3), 2014, pp. 91-102 (in Russian). DOI: 10.15514/ISPRAS-2014-26(3)-4

[5]. P. Ghysels, W. Vanroose. Hiding Global Synchronization Latency in the Preconditioned Conjugate Gradient Algorithm. Submitted to Parallel Computing, 2012.

[6]. Thierry Jacques, Laurent Nicolas, Christian Vollaire, 7th International Conference, HPCN Europe 1999 Amsterdam, The Netherlands, April 12–14, 1999 Proceedings, pp 1025-1031

[1] L. Yang, R. Brent, The improved BiCGStab method for large and sparse unsymmetric linear systems on parallel distributed memory architectures, in: Fifth International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP'02). Proceedings., IEEE ComputerSociety, Los Alamitos, CA, USA, 2002, pp. 324–328. doi:10.1109/ICAPP.2002.1173595

[7]. Boris Krasnopolsky, The reordered BiCGStab method for distributed memory computer systems, Procedia Computer Science, Volume 1, Issue 1, May 2010, Pages 213-218, ISSN 1877-0509, http://dx.doi.org/10.1016/j.procs.2010.04.024.