**УДК 519.6**

# ДВУХУРОВНЕВОЕ РАСПАРАЛЛЕЛИВАНИЕ ЯВНЫХ РАЗНОСТНЫХ СХЕМ МЕТОДИКИ ЭГАК

В. Ю. Колобянин, А. А. Фёдоров, Н. Р. Антипина
(ФГУП "РФЯЦ-ВНИИЭФ", г. Саров Нижегородской области)

Дается описание двухуровневого распараллеливания (на общей и распределенной памяти) явных разностных схем методики ЭГАК. Описан переход от одноуровневой модели распараллеливания (только на распределенной памяти) к двухуровневой. Приводятся результаты численных экспериментов по исследованию масштабируемости.

*Ключевые слова:* методика ЭГАК, двухуровневое распараллеливание, эффективность распараллеливания, параллельные ЭВМ.

## Введение

Методика ЭГАК [1, 2] предназначена для расчета в параллельном режиме двумерных и трехмерных задач механики сплошной среды на неподвижных сетках. Ее применение требует наличия довольно мелкой счетной сетки и, как следствие, больших ресурсов ЭВМ. В первоначальной версии методики ЭГАК использовалась модель одноуровневого распараллеливания на распределенной памяти с интерфейсом передачи сообщений MPI. Однако с ростом сложности рассчитываемых задач существенно возрастают запросы к мощности вычислительных систем, которая растет в основном благодаря увеличению числа ядер. Использование MPI-распараллеливания при этом может приводить к существенному увеличению накладных расходов. Поэтому в методике ЭГАК реализована двухуровневая модель распараллеливания, которая подразумевает распараллеливание на общей (средствами OpenMP) и распределенной (средствами MPI) памяти. Это, по мнению авторов, позволяет снять ограничения одноуровневого распараллеливания (только на распределенной памяти) по масштабированию (количеству используемых MPI-процессов) и повысить эффективность вычислительной системы.

Отметим, что двухуровневый подход к распараллеливанию программ не является новым и подробно изложен, например, в работе [3]. Предлагаемый авторами подход на основе типовых программ позволяет осуществить переход от распараллеливания на распределенной памяти к двухуровневому распараллеливанию практически без изменений счетных модулей.

## Одноуровневая модель распараллеливания методики ЭГАК

Главной особенностью одноуровневой модели распараллеливания является использование типовых программ технологии нерегулярного поточечного распараллеливания ПараБал [4]. Разработчик счетного модуля программирует только вычисления для одной счетной ячейки сетки. Вся необходимая организация циклов по счетным ячейкам, совмещение вычислений и межпроцессорных обменов реализованы в типовых программах. Отметим, что большинство разностных схем ЭГАК являются явными, поэтому в данной работе речь пойдет о типовых программах именно для таких схем.

Алгоритм выполнения типовой программы следующий. Сначала вычисляются величины, относящиеся к *гранично-процессорным* ячейкам. Как только гранично-процессорная ячейка рассчитана, в специальную очередь ставится заявка на ее передачу. По заполнении очереди заявок счет гранично-процессорных ячеек прерывается, производятся запись передаваемой информации из области памяти рассчитанных ячеек в буфер и передача буфера процедурами асинхронного обмена. Далее цикл счета гранично-процессорных ячеек продолжается. По заверше-

нии счета гранично-процессорных ячеек и передачи оставшейся информации производится счет *внутрипроцессорных* ячеек. На фоне этих действий в приоритетном порядке происходит прием информации от других процессоров.

Таким образом, разработчику счетного модуля необходимо запрограммировать вычисления для одной ячейки, определить информацию, передаваемую между процессорами, и воспользоваться типовой программой по шаблону. Никаких межпроцессорных обменов или циклов по ячейкам он не программирует.

## Переход от одноуровневой модели распараллеливания к двухуровневой

В данном разделе описаны различные варианты реализации типовой программы, которые были запрограммированы и опробованы в ходе создания новой версии распараллеливания явных разностных схем методики ЭГАК.

**Однобуферная типовая программа.** Так как типовая программа с одноуровневым распараллеливанием (только MPI) уже существовала в старой версии, то за отправную точку была взята именно она.

Первая типовая программа с двухуровневым распараллеливанием создавалась в два этапа: сначала было реализовано распараллеливание на общей памяти в однопроцессорном режиме, а затем в многопроцессорном.

В случае однопроцессорного режима система обменов вырождается и остается только цикл по внутренним ячейкам, где вычисляется функция (значения рассчитываемых величин).

Чтобы распараллелить данный цикл на общей памяти, его внешний вид был изменен для более удобной работы в многопоточном режиме. Идея заключалась в следующем: каждый поток должен брать ячейку из списка и вычислять функцию в этой ячейке. Для начала вокруг данного цикла средствами OpenMP была создана OpenMP-параллельная область, внутри которой и порождались потоки, а для корректного захвата потоком ячейки из списка была создана критическая секция, в которой запрещалась работа более чем одного потока.

Затем было решено дать возможность каждому потоку при одном заходе в критическую секцию захватывать не одну, а некоторое фиксированное число ячеек, задаваемое один раз на уровне компиляции (число ячеек в порции выбиралось в ходе тестирования). Это позволило повысить эффективность распараллеливания на общей памяти в несколько раз. На рис. 1 приведен график зависимости эффективности OpenMP-распараллеливания от числа ячеек в порции (описание трехмерной тестовой задачи см. ниже). Видно, что при числе ячеек в порции $\geq 16$ эффективность распараллеливания достигает $\sim 60\,\%$, в то время как с одной захватываемой ячейкой эффективность составляет всего $\sim 15\,\%$. Таким образом, было решено зафиксировать число ячеек в порции, равное 16.

При распараллеливании в многопроцессорном режиме необходимо было выполнить параллельную обработку гранично-процессорных ячеек и организовать обмен сообщениями между процессорами, который включает в себя не только MPI-обмены, но также запись в выходной буфер и чтение из входного буфера.

Прежде всего создание OpenMP-параллельной области было вынесено за пределы обработки внутренних ячеек, и потоки, работающие на общей памяти, стали создаваться в самом начале выполнения типовой программы. OpenMP-параллельная область стала включать

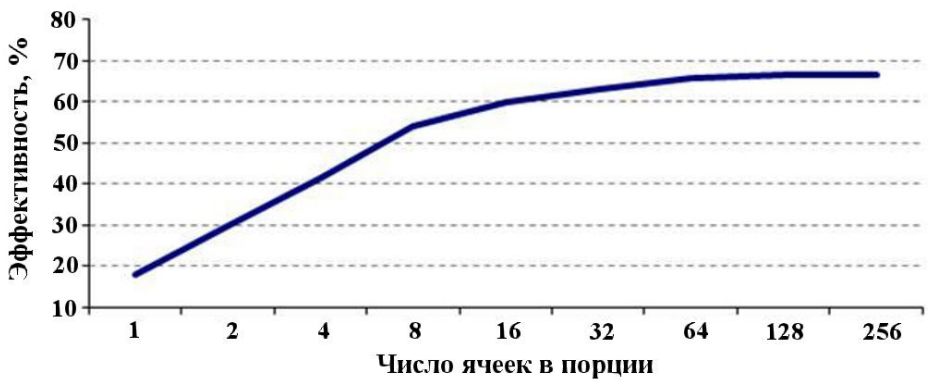


Рис. 1. Зависимость эффективности OpenMP-распараллеливания от числа ячеек в порции (16 потоков)

в себя также межпроцессорные обмены и обработку гранично-процессорных ячеек. Заполнение выходного буфера, чтение из входного буфера и передачу MPI-сообщений выполнял один конкретный поток (*мастер*-поток).

Затем по схеме, которая аналогична описанной выше, на общей памяти была реализована обработка гранично-процессорных ячеек. Отличие этого этапа от обработки внутренних ячеек состоит в том, что, помимо вычисления функции в ячейке, может происходить постановка этой ячейки в очередь заявок на передачу. Для того чтобы не возникло ситуации, когда один поток будет ставить ячейки в очередь заявок, а другой в то же время выполнять запись из очереди заявок в буфер, были введены критические секции.

Схематично выполнение полученной типовой программы, которую будем называть *однобуферной*, представлено на рис. 2. Темно-серым цветом отмечены этапы, которые выполняются в многопоточном режиме, белым — только мастер-потоком.

В ходе тестирования этого варианта типовой программы выявились некоторые проблемы, которые существенно снижали эффективность распараллеливания:

1. Наличие критической секции не позволяет одновременно ставить ячейки в очередь на передачу.
2. Наличие критической секции при работе со списками ячеек не позволяет нескольким потокам одновременно набирать счетные ячейки для обработки.
3. На ряде этапов, где счетная нагрузка небольшая, а размер передаваемой информации относительно велик, обмены не "перекрываются" счетом.

Ввиду указанных проблем авторы реализовали следующий вариант типовой программы.

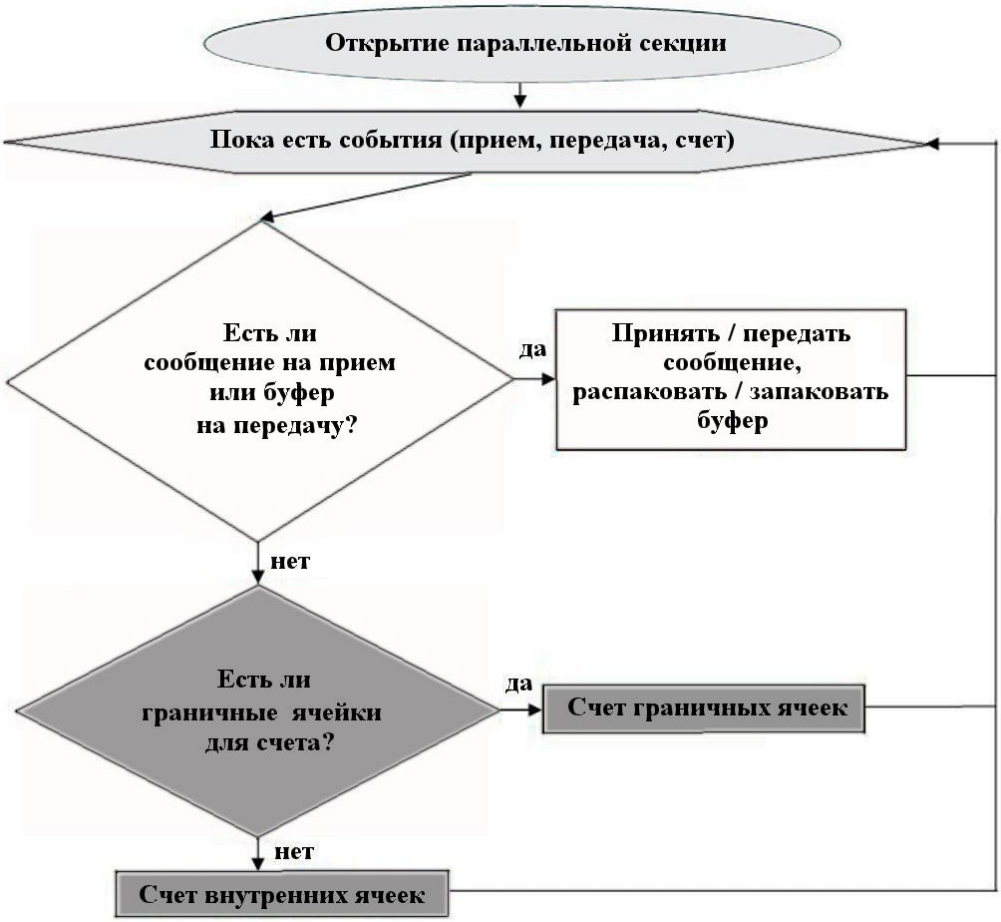


Рис. 2. Блок-схема однобуферной типовой программы

**Многобуферная типовая программа.** Как отмечено выше, при выполнении однобуферной типовой программы возникали ситуации, когда обмены не перекрывались счетом, т. е. упаковка и распаковка обменных буферов занимали довольно много времени. Чтобы уменьшить это время, была реализована схема, в которой работа между потоками распределена следующим образом:

1. Все OpenMP-потоки вычисляют значения величин в граничных ячейках.
2. Один OpenMP-поток вызывает MPI-функции, связанные с отправкой и приемом MPI-сообщений. Назовем такой поток *обменным*.
3. Несколько OpenMP-потоков упаковывают свои буферы для отправки MPI-сообщений обменным потоком. Назовем такие потоки *отправляющими*.
4. Несколько OpenMP-потоков распаковывают буферы MPI-сообщений, принятые обменным потоком от других MPI-процессов. Назовем такие потоки *принимающими*. Принимающие и отправляющие потоки могут быть разными, но могут и пересекаться. Потоки, которые могут как упаковывать, так и распаковывать буферы, будем называть *буферными*.
5. Остальные потоки вычисляют значения величин во внутренних ячейках. Назовем такие потоки *счетными*.

Схематично выполнение *многобуферной* типовой программы представлено на рис. 3. Темно-серым цветом отмечены этапы, которые выполняются в многопоточном режиме, белым — только мастер-потоком.

В одной из первых реализаций многобуферной программы подразумевалось, что принимающие и отправляющие потоки будут в свободное от основной работы время заниматься вычислением значений величин в ячейках. Но от этого пришлось отказаться, так как время счета в случае полного разделения функций между счетными и буферными потоками оказалось меньше, чем в случаях совмещения.

Алгоритм выполнения многобуферной типовой программы следующий:

1. Рассчитываются значения величин в граничных ячейках.
2. Далее начинается этап обменов, выполняемый принимающими, передающими и обменным потоками. Остальные потоки производят вычисление величин во внутренних ячейках.
3. С соседнего MPI-процесса приходят сообщения.


Рис. 3. Блок-схема многобуферной типовой программы

4. Обменный поток принимает каждое MPI-сообщение и кладет его в буфер того принимающего потока, который в этот момент не занят распаковкой своего буфера.
5. Принимающие потоки занимаются распаковкой своих буферов, которые заполнил обменный поток. После того как потоки заканчивают распаковку, они очищают свои буферы и обменный поток может положить в них другие сообщения.
6. В это время передающие потоки упаковывают свои буферы, после чего обменный поток отправляет буфер каждого передающего потока в отдельности соседним процессам.
7. После отправки буфера соседним процессам передающий поток может снова заполнять свой буфер новыми значениями.

Особо стоит обратить внимание на формирование буфера для передачи сообщения соседним процессам. Есть по крайней мере два варианта:

1. Заранее распределять граничные ячейки по передающим потокам, чтобы каждый поток "знал", какие ячейки он должен передать соседям.
2. Каждый передающий поток всякий раз при начале выполнения функции формирования буфера должен забирать порцию ячеек из общего списка граничных ячеек, чтобы сразу же их запаковать и передать соседям.

Второй подход подразумевает наличие критической секции на доступ к списку, а первый — небольшую разбалансированность, так как одна и та же гранично-процессорная ячейка может передаваться сразу нескольким соседям. Таким образом, передающий поток, которому досталась такая ячейка, сделает несколько больше передач соседям, чем поток, который отправляет ячейки только одному соседу. После тестирования этих вариантов решено было остановиться на первом, так как именно он показал наименьшее время выполнения.

Таким образом, разные потоки могут одновременно заниматься разной работой: обменный поток вызывает MPI-функции, с помощью которых принимаются и передаются сообщения; передающие потоки упаковывают буферы для передачи соседним процессам; принимающие потоки распаковывают буферы с сообщениями от соседних процессов; счетные потоки вычисляют значения величин во внутренних ячейках (граничные ячейки рассчитываются всеми потоками).

## Новая типовая программа

Все рассмотренные реализации типовой программы для явных разностных схем основывались на выполнении цикла, в котором обработка событий (принятие и отправка сообщений, заполнение и распаковка буфера, счет внутренних и гранично-процессорных ячеек и т. д.) производилась по мере их наступления для порции ячеек, т. е. на каждой итерации цикла могли происходить разные события.

С целью уменьшения времени выполнения было решено реализовать новый вариант типовой программы, в котором цикл отсутствует, а обработка событий для всех ячеек происходит в определенном порядке.

Таким образом, новый вариант типовой программы имеет более линейную структуру, его блок-схема представлена на рис. 4. После открытия параллельной секции сначала выполняется инициализация требуемых для счета переменных, потом производится расчет величин гранично-процессорных ячеек. Далее все гранично-процессорные ячейки собираются в единый буфер для отправки сообщений. Выделенный поток производит неблокирующий прием сообщений в единый входной буфер и неблокирующую отправку сообщений с информацией из единого выходного буфера определенными порциями. Оставшиеся потоки выполняют расчет величин, относящихся к внутрипроцессорным ячейкам; после завершения операций приема и передачи к ним присоединяется выделенный под MPI-обмены поток. Далее всеми потоками производится распаковка информации из единого выходного буфера, и параллельная секция закрывается.

Как видно из рис. 4, все потоки участвуют во всех этапах выполнения типовой программы, кроме этапа с операциями приема и передачи MPI-сообщений. Но так как эти операции неблокирующие, то затраченное на них время успешно перекрывается временем счета и работы с буфером.

## Тестирование

Так как каждая новая типовая программа устраняла недостатки предыдущей, при тестировании решено было сравнивать только исходный вариант (одноуровневая модель распараллеливания) и последний вариант типовой программы, описанный в предыдущем разделе.

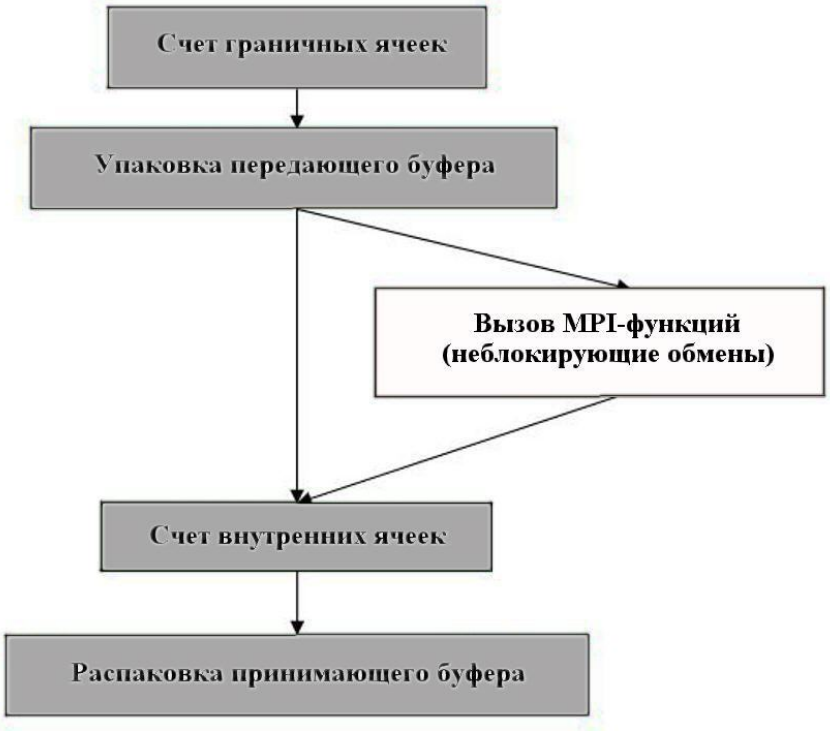


Рис. 4. Блок-схема новой типовой программы

Для тестирования была выбрана следующая задача. В расчетной области ($0 < X < 10$; $0 < Y < 10$; $0 < Z < 10$) содержится идеальный газ ($\rho_1 = 1$; $e_1 = 0$; $\gamma_1 = 1{,}4$; $u = 0$), в подобласти ($0 < X < 1$; $0 < Y < 1$; $0 < Z < 1$) задается энергия $e_2 = 1$. Используемые процессы — лагранжева газовая динамика и пересчет величин на новую сетку (эйлеров этап).

Задача считалась 20 шагов. Оценивалось среднее время выполнения шага. Исследование эффективности распараллеливания проводилось методом деления (сильная масштабируемость, общее число ячеек в задаче $200^3$) и методом умножения (слабая масштабируемость, число ячеек $200^3$ на узел).

Введем следующие обозначения: $t_1$ — время выполнения программы одним параллельным процессом*; $t_p$ — время выполнения программы $p$ параллельными процессами; $E_p = (t_1/pt_p) \times$ $\times\ 100\%$ — эффективность распараллеливания методом деления; $E_p = (t_1/t_p) \cdot 100\%$ — эффективность распараллеливания методом умножения.

Сначала проверим, насколько сильно отличается эффективность распараллеливания новой и старой (только MPI) схем. Поскольку в старой схеме не реализовано распараллеливание на общей памяти, сравнение проводилось в MPI-режиме. Результаты продемонстрированы на рис. 5. Видно, что эффективность распараллеливания (и соответственно время выполнения) старой и новой схем близки.

Теперь выясним, какое соотношение числа MPI-процессов и OpenMP-потоков оптимально. Запись $N \times M$ будет означать, что на вычислительном узле выполняется $N$ MPI-процессов по $M$ OpenMP-потоков на каждом.

Результаты исследования приведены на рис. 6. Из рисунка следует, что наиболее высокая эффективность распараллеливания достигается в режиме $2 \times 14$ (это соответствует архитектуре двухпроцессорного узла с общей памятью, содержащего 14 ядер в каждом процессоре). Таким образом, самый оптимальный режим для одного узла — 2 MPI-процесса по 14 OpenMP-потоков на каждом.

После выбора оптимального режима для одного вычислительного узла была проверена масштабируемость новой схемы на нескольких вычислительных узлах (рис. 7). Видно, что эффективность распараллеливания достаточно высокая и не опускается ниже 70 %. При этом в ре-

*Под параллельным процессом понимается либо MPI-процесс, либо OpenMP-поток.

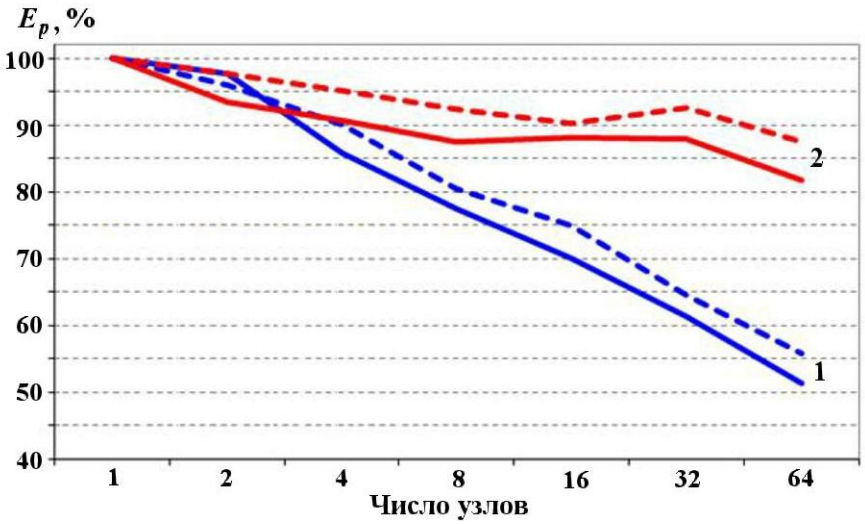


Рис. 5. Эффективность распараллеливания старой и новой схем в MPI-режиме: 1 — сильная масштабируемость; 2 — слабая масштабируемость; - - - — старая схема; —— — новая схема

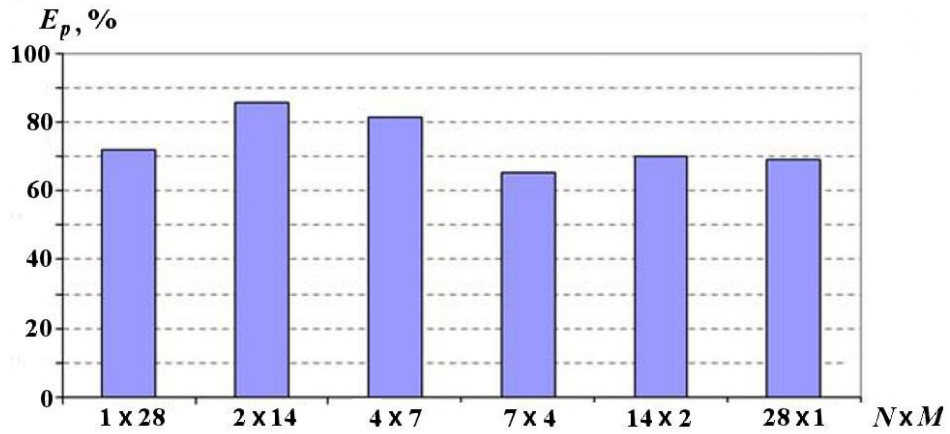


Рис. 6. Эффективность двухуровневого распараллеливания для одного вычислительного узла (сильная масштабируемость)

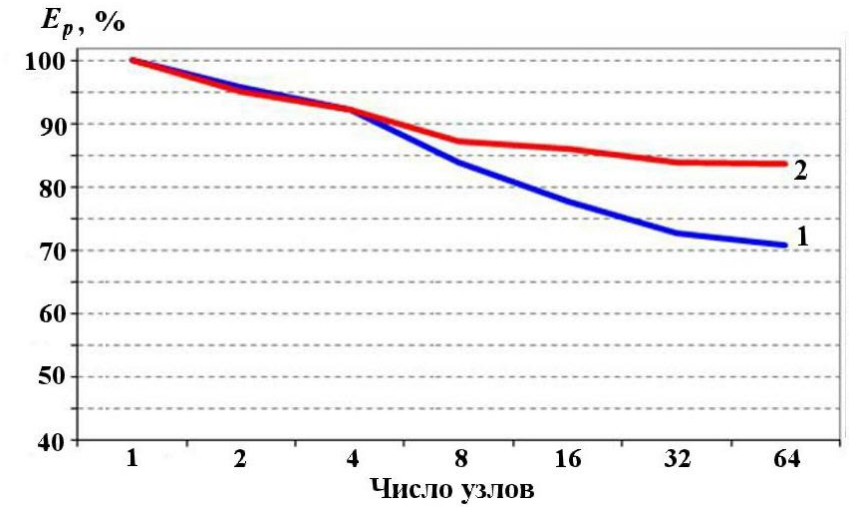


Рис. 7. Эффективность распараллеливания новой схемы на нескольких вычислительных узлах (2 MPI-процесса по 14 OpenMP-потоков на каждом узле): 1 — сильная масштабируемость; 2 — слабая масштабируемость

жиме сильного масштабирования она значительно выше, чем при одноуровневом распараллеливании (см. рис. 5).

## Заключение

В данной работе исследованы различные по сложности подходы к распараллеливанию явных разностных схем методики ЭГАК. Основные изменения касались типовой программы, которая является ядром всех счетных модулей. Тестирование на трехмерной газодинамической задаче показало, что новая программа с двухуровневым распараллеливанием масштабируется лучше, чем старая программа с одноуровневым распараллеливанием. При распределении счетной сетки по 125 тысяч ячеек на каждый узел производительность счета по новой программе возросла в 1,27.

## Список литературы

1. *Дарова Н. С., Дибиров О. А., Жарова Г. В. и др.* Комплекс программ ЭГАК. Лагранжево-эйлерова методика расчета двумерных газодинамических течений многокомпонентной среды // Вопросы атомной науки и техники. Сер. Математическое моделирование физических процессов. 1994. Вып. 2. С. 51—58.
2. *Янилкин Ю. В., Беляев С. П., Бондаренко Ю. А. и др.* Эйлеровы численные методики ЭГАК и ТРЭК для моделирования многомерных течений многокомпонентой среды // Труды РФЯЦ-ВНИИЭФ. 2008. Вып. 12. С. 54—65.
3. *Воропинов А. А., Соколов С. С., Новиков И. Г.* Двухуровневое распараллеливание в модели смешанной памяти для расчета задач газодинамики в методике ТИМ-2D // Вопросы атомной науки и техники. Сер. Математическое моделирование физических процессов. 2008. Вып. 1. С. 51—59.
4. *Беляев С. П.* Метод мелкозернистого распараллеливания с динамической балансировкой на примере задачи газовой динамики и вычислительные эксперименты на параллельной системе // Там же. 2000. Вып. 1. С. 45—49.

Статья поступила в редакцию 23.01.17.

TWO-LEVEL PARALLELING OF EXPLICIT DIFFERENCE SCHEMES IN EGAK CODE / V. Yu. Kolobyanin, A. A. Fedorov, N. P. Antipina (FSUE "RFNC-VNIIEF", Sarov, Nizhny Novgorod region).

The paper describes the two-level paralleling (in shared memory and distributed memory) of explicit difference schemes in the EGAK code. The transfer from the one-level paralleling model (in distributed memory) to the two-level model is described. Results of numerical scalability tests are presented.

*Keywords*: the EGAK code, two-level paralleling, paralleling efficiency, parallel computers.