Оценка ускорения параллельных поэлементных схем МКЭ на многоядерных архитектурах *

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

Институт механики УрО РАН

Исследуется универсальная модель масштабируемости для сравнения и выделения алгоритмов послойного разделения и многоядерных архитектур, ограниченных по пропускной способности памяти и скорости обработки данных при поэлементном матрично-векторном произведении.

Ключевые слова: метод конечных элементов, разделение расчетных сеток, поэлеметное матрично-векторное произведение, многоядерные процессоры, универсальная модель масштабируемости.

1. Введение

Важным при анализе масштабируемости разрабатываемых параллельных алгоритмов являются характеристики архитектур вычислительных узлов, на которых будут выполняться вычисления. Создание предсказательной модели параллельных вычислений, адекватно описывающей производительность создаваемых параллельных алгоритмов и программ на современных многоядерных процессорах и вычислительных системах, с учетом влияния иерархии подсистемы памяти и параллельности, является актуальной задачей.

Существующие модели и оценки параллельного ускорения S, рассмотренные в работах Амдаля и Хилла-Марти [1] и их многочисленных обобщениях, включают долю параллельных вычислений или число некоторых базовых аппаратных элементов r относящихся к многоядерной архитектуре системы

$$S_A(n) = \frac{1}{(1-f) + \frac{f}{n}}, \quad S_{HM}(n) = \frac{1}{\frac{(1-f)}{\mathrm{perf}(r)} + \frac{f}{n}}. \tag{1}$$

Здесь f — доля операций алгоритма, выполняемых параллельно. В последнем соотношении (1) при фиксированных параметрах n, f максимальное ускорение достигается при числе базовых ядер $r = n$ с производительностью каждого $\mathrm{perf}(r) < r$. Возможности определения подобных параметров для практических важных алгоритмов и современных архитектур существенно ограничены.

Развитие теоретических моделей масштабируемости вида (1) для многоядерных архитектур можно найти в [2, 3]. Для прогнозирования реальных ускорений параллельных вычислений необходим в том или ином виде учет причин его замедления при выполнении на различных многоядерных процессорах: обмен общими перезаписываемыми данными между кэшами процессора и между процессорами и основной памятью; блокировки синхронизации (сериализации) общих данных, доступных для записи; ожидания доступа к памяти для завершения операций и других. Из проведенного анализа литературы можно предположить, что только универсальная модель масштабируемости (USL), предложенная и развиваемая на протяжении ряда лет в работах Гюнтера [4, 5] позволяет спрогнозировать ускорения на различных современных архитектурах. Отметим, что модель USL первоначально была использована при оценке производительности вэб-серверов и серверов баз данных, и видимо поэтому не рассматривалась в подавляющем большинстве работ по достигаемым ускорениям в параллельных вычислениях. В данной работе, как представляется авторам, модель USL впервые применяется к численным алгоритмам.

*Работа выполнена при частичной финансовой поддержке РФФИ (гранты 14–01–00055-а, 16–01–00129-а).

Однако, важно не только смоделировать масштабируемость на основе результатов практического выполнения рассматриваемого алгоритма на реальной вычислительной системе, но и оценить механизмы замедления вычислений, а в дальнейшем и исключить причины их вызывающие при использовании современных и перспективных архитектур. Так, в большинстве случаев конечно-элементные аппроксимации строятся на неструктурированных расчетных сетках, что приводит к нерегулярному доступу к сеточным данным, размещаемым в оперативной памяти, и существенно ограничивает достижение высоких ускорений при параллельных вычислениях.

Параллельная эффективность поэлементных конечно-элементных схем снижается при суммирование компонент из поэлементных векторов [6]. Операция сборки относится к алгоритмам с плохой локализацией данных и невысоким потенциалом распараллеливания, ограниченным задержками при работе с памятью. При суммировании компонент, соответствующих общим узлам сетки возникают конфликты при записи в память, когда несколько параллельных вычислительных процессов обращаются к одной ячейке памяти. Для исключения подобных ситуаций применяется синхронизация, например, критическая секция OpenMP существенно замедляющая скорость параллельных вычислений [7]. Также используется перенумерация узлов неструктурированных сеток для минимизации кэш-промахов и конкуренции за память [8], раскраска граней ячеек в сеточных методах конечных объемов и Галеркина с разрывными базисными функциями [9] для оптимизации доступа к данным в памяти.

В данной работе для алгоритмического сокращения задержек доступа к общей оперативной памяти проектируются параллельные алгоритмы на неструктурированных сетках с быстрым доступом к памяти и локализованном размещении данных за счет отказа от использования части естественного параллелизма, получаемого на основе стандартных графовых разделений на независимые подобласти с минимизацией обмена данных между ними. Предлагается введение нового упорядочения конечно-элементных неизвестных, при котором одновременно в суммировании не участвуют расчетные ячейки, содержащие общие вершины, и не требуется дополнительных операций разрешения конфликтов доступа к памяти. Используя отношения соседства, выделяются слои расчетных ячеек, при помощи которых сетка разделяется на подмножества ячеек для каждого параллельного вычислительного процесса. Поэлементные векторы наследуют полученное упорядочение степеней свободы. Таким образом, суммирование (сборка) векторов вводится во взаимосвязи с упорядочением, как часть поэлементной операции композиции.

Рассматривается использование универсальной модели масштабируемости USL [5] для оценки эффективности выполнения поэлементных вычислений при послойном разделении неструктурированных сеток на различных многоядерных архитектурах.

2. Поэлементные вычисления в МКЭ

Конечно-элементные вычисления предполагают применение алгоритмов, в которых кроме параллельных и последовательных вычислений существуют операции, объединяющие (суммирующие) данные из параллельных ветвей алгоритма, на основе некоторого разделения. Эти операции будем называть композицией или операциями композиции [7]. Будем полагать, что композиция состоит из двух этапов, взаимосвязанных, но разнесенных по времени выполнения: 1) разделение с заданными свойствами (локализация данных и сбалансированность вычислительной нагрузки); 2) объединение / суммирование (сборка) распределенных данных. Этап разделения предшествует соответствующему шагу вычислительной схемы метода конечных элементов, а этап сборки является частью этого шага. Как правило, в методе конечных элементов под сборкой (ансамблированием) подразумевается суммирование локальных матриц жесткости конечных элементов в глобальную матрицу жесткости системы уравнений.

В поэлементных схемах [10] сборка переносится на этап решения конечной элементной системы и применяется уже не к матрицам, а к векторам — результатам матрично-векторного произведения вида

$$q = Kp = \sum_{e=1}^{m} C_e^T \tilde{K}_e C_e p = \sum_{e=1}^{m} C_e^T \tilde{q}_e = \sum_{e=1}^{m} q_e, \tag{2}$$

здесь q, p, q_e — вектора размера N; K — глобальная матрица жесткости размера $N \times N$; $\tilde{K}_e$ — локальная матрица жесткости конечного элемента e, имеющая размер $N_e \times N_e$; C_e — матрица инцидентности $N_e \times N$ отображает локальное пространство номеров неизвестных (степеней свободы) $[1, 2, \ldots, N_e]$ в глобальное $[1, 2, \ldots, N]$; m — число конечных элементов. Такое отображение в конечно-элементном приложении реализуется при косвенной индексации неизвестных с использованием номеров узлов сетки или при умножении на матрицу инцидентности, которое эффективно выполняется на графических ускорителях.

При распараллеливании выражения (2) в рамках модели общей памяти векторы p и q находятся в общей памяти параллельных процессов/нитей, а матрицы $\tilde{K}_e$, C_e, C_e^T и разреженный вектор q_e в памяти соответствующего процесса. В этом случае конфликты, приводящие к ошибкам, возникают при суммировании векторов q_e, находящихся в разных процессах и имеющих ненулевые компоненты с одинаковыми номерами.

Представим произведение $q = Kp$ в виде двух операций: поэлементное произведение $\tilde{q}_e = \tilde{K}_e C_e p$, $e = 1, 2, \ldots, m$ и сборку $q_e = \sum_{e=1}^{m} C_e^T \tilde{q}_e$, которые существенно отличаются по вычислительным и коммуникационным затратам, как следствие различной локализации данных (доступ по локальным и глобальным номерам узлов), особенно в случае неструктурированных сеток, и, кроме того, имеют разный потенциал распараллеливания. При этом, каждый параллельный процесс выполняет вычисления над $m_i \approx m/n$ конечными элементами, где i — номер процесса, а n — число процессов.

Запишем полученные выражения в матричном виде

$$q = Kp = \sum_{i=1}^{n} C^{T^{(i)}} \tilde{K}^{(i)} C^{(i)} p = \sum_{i=1}^{n} C^{T^{(i)}} \tilde{q}^{(i)} = \mathcal{A}(\tilde{q}), \tag{3}$$

где $C^{(i)}$ — матрица инцидентности, составленная из m_i матриц C_e; $\tilde{K}^{(i)}$ — квазидиагональная матрица, состоящая из m_i матриц $\tilde{K}^{(i)}$, $\tilde{q}^{(i)}$ — вектор, компонентами которого являются m_i векторов $\tilde{q}_e$ и $\mathcal{A}$ — оператор сборки вектора q в рамках операции композиции.

Таким образом, необходимо на расчетной сетке выделить подмножества конечных элементов, не имеющих общих вершин для исключения одновременного доступа к памяти и каждое подмножество связать с параллельным процессом/нитью, а сборку одновременно выполнять над несвязанными расчетными ячейками в поэлементном матрично-векторном произведении вида (3).

3. Послойное разделение

Для алгоритмического сокращения задержек доступа к памяти рассмотрим варианты построения параллельных конечно-элементных алгоритмов на неструктурированных сетках с улучшенной локальностью, обеспечивающей эффективное обращение к сеточным данным в оперативной памяти.

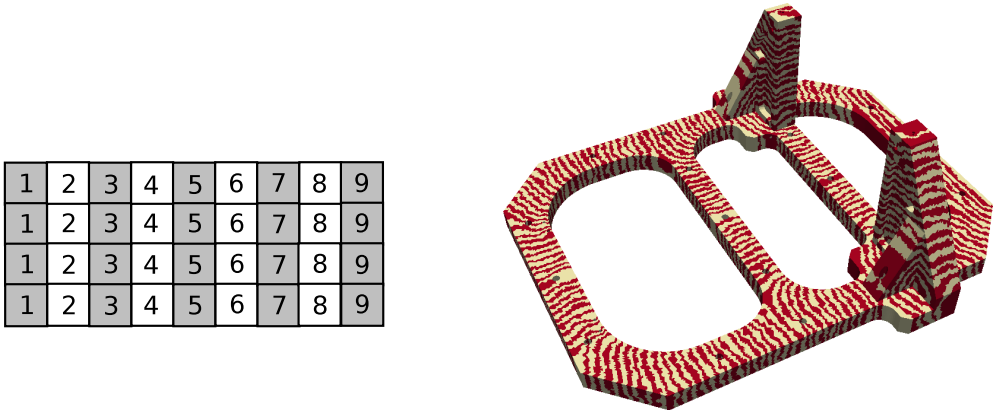
Будем считать, что два подмножества ячеек сетки не связаны в данный момент времени, если одновременно извлекаемые из этих подмножеств ячейки сетки не содержат общих вершин. Таким образом, ограничение на связанность ячеек накладывается только на момент обращения к подмножеству. Это означает, что ячейки внутри подмножества и сами подмножества могут быть топологически связаны [7].

Соответствующие подмножества ячеек формируются при помощи разделения сетки Ω

на неперекрывающиеся слои ячеек $s_j, j = 1, 2, \ldots, n_s$ (см. рисунок 1а) и последующего объединения слоёв в подмножества ячеек (подобласти) Ω_i, $i = 1, 2, \ldots, n_\Omega$ (см. рис. 2).

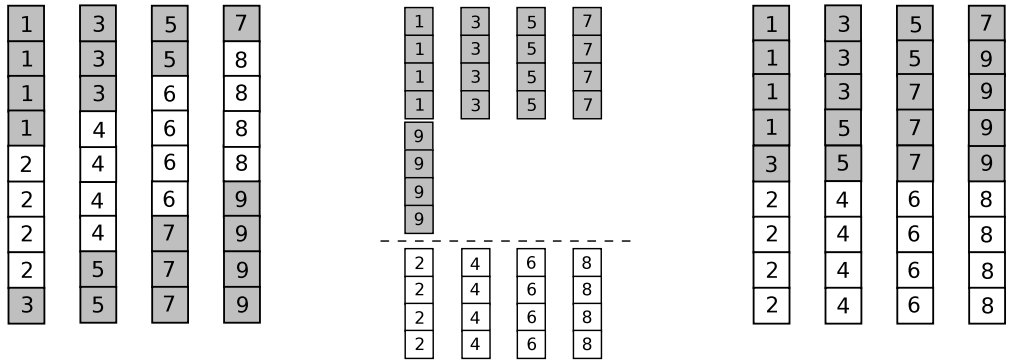
Рассмотрим несколько вариантов формирования подобластей Ω_i из слоёв неструктурированной сетки [7], приводящих к упорядочению ячеек сетки и исключающих одновременный доступ из параллельных процессов (нитей) к конечным элементам, имеющим общие вершины. В блочном варианте слои объединялись последовательно, полученное объединение слоев разделялось на подобласти, содержащие примерно равное число конечных элементов (см. рис. 2а). При объединении по чётности вводилась схема нумерации слоёв (см. рис. 2в). Штриховая линия на рисунке 2б обозначает завершение одной параллельной области OpenMP и начало следующей (используются две директивы `#pragma omp parallel`).

Сравним предложенные варианты упорядочения ячеек по влиянию на сбалансированность



а) б)

Рис. 1: Разделение на слои: а) схема выделения слоёв; б) разделение на 137 слоёв неструктурированной трехмерной сетки из 485843 тетраэдров.



а) б) в)

Рис. 2: Схемы объединения слоёв в блоки для четырех процессов ($n_\Omega = 4$): а) блочная схема (bl); б) схема по чётности номеров слоёв (ev); в) сбалансированное объединение по чётности номеров слоёв (ev+bal).

вычислительной нагрузки. Для этого рассмотрим результаты поэлементного матрично-векторного произведение без сборки векторов (данная операция полностью параллельная) на неструктурированной сетке (см. рис. 1) вычисленного на восьмиядерном процессоре Xeon E5-2690 с четырьмя вариантами разделения сетки: блочном (bl), по четности (ev), он же сбалансированный (ev+bal) и многоуровневого графового разделения METIS). Полученные результаты показывают возможности рассматриваемых алгоритмов разделений для

неструктурированных сеток и получение сбалансированных по числу расчетных ячеек подобластей/слоёв (см. рис. 3). Достижение линейного ускорения в данном случае ограничивается только дисбалансом вычислительной нагрузки (число ячеек), который меньше уровня разбалансированности, получаемой методами многоуровневого разделения дуального графа расчетной сетки [11]. Над столбцами гистограммы указан дисбаланс вычислительной нагрузки в процентах. В вариантах блочного разделения и по четности слоёв с выравниванием слоёв по числу ячеек (ev+bal) дисбаланс вычислительной нагрузки много меньше одного процента. Используемые алгоритмы разделения слоёв позволяют разрешать пробле-

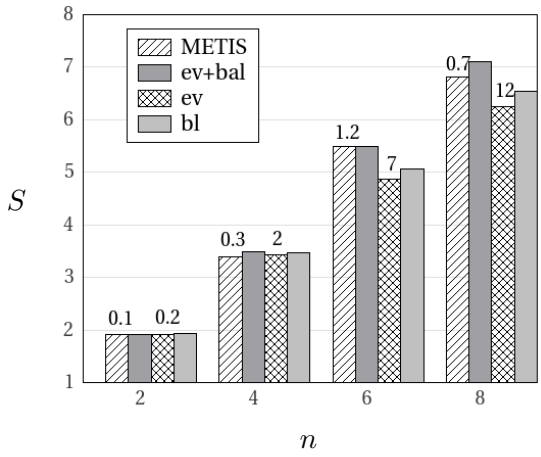


Рис. 3: Ускорение при вычислении матрично-векторного произведения на процессоре Xeon E5-2690 при различных вариантах разделения сетки

мы балансировки вычислительной нагрузки на нерегулярных сетках и создают условия для рассмотрения всех возможных причин замедления ускорения для параллельных операций в поэлементных схемах МКЭ.

4. Особенности организации доступа к памяти на многоядерной архитектуре

Эффективность распараллеливания кроме сбалансированности вычислений существенно зависит и от многоядерной архитектуры и пропускной способности подсистемы памяти, так как параллельное выполнение многократно увеличивает поток обмена данными с оперативной памятью.

В многопроцессорных узлах и многоядерных процессорах к оперативной памяти могут обращаться несколько процессоров или ядер. В этом случае необходимо своевременно отслеживать положение данных в кэше и оперативной памяти и синхронизировать их изменения — поддерживать когерентность (согласованность) между кэшем и оперативной памятью. Обеспечить это возможно, если в алгоритме операции записи в одну и ту же ячейку памяти будут выполнятся строго последовательно (сериализованы), т.е. две подряд идущие операции записи в одну и ту же ячейку памяти будут наблюдаться другими ядрами именно в том порядке, в котором они появляются в процессоре, выполняющем эти операции записи — последовательное выполнение операций записи. Операция чтения ячейки памяти одним ядром, которая следует за операцией записи в ту же ячейку памяти другим ядром, должны быть достаточно далеко отделены друг от друга по времени.

Для аппаратного сокращения времени доступа вводится несколько уровней кэш-памяти, что повышает пропускную способность, значительно сокращая число обменов данными между процессором и оперативной памятью за счет их локализации на разных уровнях

кэш-памяти. Другим методом снижения конфликтов при обменах с памятью является использование многопроцессорной архитектуры с неодинаковым доступом в общую память (NUMA). Время доступа становится недетерминированным и зависит от того, на каком уровне иерархии находятся нужные данные, а также от конфликтов, которые возникают при обращении к одним и тем же ячейками памяти.

Приведем для сравнения наиболее популярные многоядерные архитектуры с точки зрения представления организации доступа и разделения общих ресурсов и на которых выполнялось тестирование рассматриваемых алгоритмов разделения и организации потоков данных:

- восьмиядерный узел NUMA архитектуры, в котором установлены: два четырехядерных CPU Intel Xeon E5-2609, 64 ГБ оперативной памяти, кэш первого уровня L1= 512 КБ (4 x 32 КБ инструкций, 4 x 32 КБ данные), кэш второго уровня L2= 1 МБ (4 x 256 КБ), кэш третьего уровня L3=10 МБ. Максимальная пропускная способность памяти — 51,2 ГБ/с.

- вычислительный четырехсокетовый узел с шестиядерными процессорами Core AMD Opteron(tm) Processor 8435 с поддержкой технологии HyperTransport (или HT Assist) для ускорения выполнение запросов в четырехпроцессорных серверных системах с повышением скорости обмена данными до показателя 4,8 ГТ/с на один линк и объемом оперативной памяти 64 ГБ, кэш первого уровня L1= 384 КБ (6 x 64 КБ), кэш второго уровня L2= 3 МБ (6 x 512 КБ), кэш третьего уровня L3=6 МБ.

- сопроцессор Intel Xeon Phi 7110X содержит 61 ядро с поддержкой 4-х одновременных потоков на ядро на частоте 1.1 ГГц, контролеры памяти, интерфейс PCEe, двунаправленная шина, кэш первого уровня L1= 32 КБ +32 КБ, кэш второго уровня L2= 512 КБ с аппаратной предвыборкой 16 потоков, организация кэшей L1, L2 инклюзивная (дублирование информации, находящейся в L1 и L2), быстрый доступ к кэшам соседних ядер (распределенный каталог), оперативная память 8 ГБ. Пропускная способность памяти 320 ГБ/с.

- ускоритель NVIDIA GTX 980 или GPU GM204 использует четыре кластера GPC (Graphics Processing Clusters), в каждом четыре потоковых мультипроцессора SMM, содержащих 32 АЛУ, что даёт 2048 потоковых процессоров (PS). Глобальная память (4096 МБ, размер кэша L2 — 2 МБ, унифицированный механизм L1—24 КБ); разделяемая память — 96 КБ; текстурная память L1 — 24 КБ. Пропускная способность памяти 224.3 ГБ/с. Частота ядра 1126 МГц. В каждом SMM есть четыре warp-планировщика.

Отметим отличие и в способах доступа к памяти. Так, не все универсальные процессоры имеют встроенные контроллеры памяти, а у всех ускорителей обычно есть по несколько контроллеров. В отличие от обычных CPU с одним устройством контроля исполнения, кэш и несколькими арифметико-логическими устройствами, ускоритель GPU состоит из набора мультипроцессоров, каждый из которых имеет своё устройство контроля исполнения, область разделяемой памяти, области регистров и значительно расширенный по сравнению с CPU набор арифметико-логических устройств. Кроме того, на ускорителях применяется более быстрая память с существенно большей пропускной способностью, что также весьма важно для параллельных вычислений, оперирующих с сверхбольшими потоками данных.

Доступ к памяти GPU легко прогнозируемый — если из памяти читается или записываются данные, то через некоторое время настанет очередь и соседних данных. Отличия есть и в кэшировании данных. Если CPU используют кэш для увеличения производительности за счёт снижения задержек доступа к памяти, то GPU используют кэш для увеличения полосы пропускания. В CPU задержки доступа к памяти снижаются при помощи кэш-памяти большого размера и предсказания ветвлений кода. В ускорителях проблема задержки доступа к памяти решается при помощи одновременного исполнения тысяч потоков — при

ожидании одним из потоков данных из памяти, ускоритель выполняет вычисления другого потока без ожидания и задержек.

Отличие ускорителя вычислений на архитектуре MIC от GPU возникает благодаря содержанию кэш-памяти, когерентной со всеми его ядрами и использованием специфичных наборов инструкций наряду с набором x86-инструкций. Исполнение команд становится более предсказуемым, поэтому планирование инструкций и задач могут быть выполнены компилятором. Каждое ядро поддерживает четырехнаправленный одновременный мультитридинг с четырьмя копиями каждого регистра процессора. Содержит явные инструкции по контролю кэш-памяти, предназначенные для уменьшения процесса замусоривания кэша во время потоковых операций, которые записывают и считывают данные только однократно. Поддерживается также явная предвыборка в L1, L2 кэши.

Важным при анализе масштабируемости параллельных алгоритмов являются способы разрешения проблем организации доступа к памяти на современных архитектурах, некоторые из них разрешаются на аппаратном уровне или компилятором и должны быть каким-то образом учтены при описании архитектуры вычислительной системы, на которой будут выполняться вычисления и поведение которой необходимо спрогнозировать.

5. Универсальная оценка масштабируемости

Модель вычислений является следующим, более высоким уровнем абстракции и представляет ещё более формальную модель соответствующей модели архитектуры. Она позволяет строить функции стоимости, отражающие временные затраты, необходимое для выполнения алгоритма на ресурсах вычислительной системы, заданной модели архитектуры. Такая модель вычислений должна обеспечивать простой метод для проектирования и оценки параллельных алгоритмов.

В работе Гюнтера [5] (см. библиографию в ней) показана возможность представления и оценки масштабируемости/ускорения на основе обобщения закона Амдаля, учитывающего накладные расходы связанные с согласованием вычислений между идентичными ядрами/-процессорами и с поддержанием когерентного состояния системы.

Модель Гюнтера (USL) считается универсальной в том смысле, что не предполагается каких-либо свойств алгоритма (например, доли f параллельных вычислений см. (1)), программного обеспечения или конкретной архитектуры многоядерной системы и др. Согласно этой модели ускорение имеет глобальный максимум (максимальное достигаемое ускорение) с увеличением числа ядер/процессоров определяемый дополнительными издержками, связанными с поддержанием согласованности общих структур данных и их размещения в памяти с доступом и обновлением параллельными выполняемыми потоками.

Применение данной модели заключается в проведении измерений параллельного ускорения $S(n) = t_1/t_n$, где t_1, t_n — время выполнения алгоритма одним или n — потоками и получения наиболее точного вида функции ускорения для конкретного алгоритма и вычислительной системы с оценкой двух параметров модели. Ускорение произвольной вычислительной системы (алгоритмической, программной и архитектурной) представляется рациональной функцией.

Ускорение S_U с ростом числа параллельных процессов n в модели USL записывается как

$$S_U(n) = \frac{n}{1 + \sigma(n-1) + \lambda n(n-1)}, \tag{4}$$

где параметры σ, λ определены в пределах $0 \leqslant \sigma$, $0 < \lambda < 1$. При $\lambda = 0$ соотношение (4) принимает вид закона Амдаля с долей последовательных вычислений, определяемой как $\sigma = (1 - f)$. Знаменатель в (4) состоит из трех слагаемых, каждое имеет определенную физическую интерпретацию: параллелизм — линейная масштабируемость возможна, если отдельные составляющие системы (процессоры, потоки и т.п.) могут работать без взаимодействий $\sigma, \lambda = 0$; масштабируемость ограничена конкуренцией — конфликтами между раз-

личными частями вычислительной системы (алгоритма), вызванных эффектами сериализации и очередей $\sigma > 0, \lambda = 0$; масштабируемость ограничена когерентностью — задержками, вызванными поддержанием системы в согласованном состоянии, связана с работой кэш-памяти в различных программных и аппаратных частях вычислительной системы $\sigma, \lambda > 0$.

Таким образом, параметры модели можно представить отвечающими за два конкретных механизма, которые ограничивают линейное ускорение параллельного алгоритма с ростом числа параллельных процессов или ядер. Отметим, что второе слагаемое растет линейно, а третье квадратично с увеличением числа n взаимодействующих процессов. Тогда вычислительные алгоритмы, архитектуры могут быть классифицированы или охарактеризованы в соответствии со значениями параметров σ и λ. Абсолютные значения параметров модели не определяют детальных механизмов и данных по оценке организации вычислений и работе с памятью, но достаточно хорошо характеризуют модель параллелизма и архитектуру системы при рассмотрении полученных ускорений для одного и того же алгоритма, разделения данных на разных типах многоядерных архитектур. Однако, появляется возможность оценки и прогнозирования случаев чрезмерных затрат на согласование и синхронизацию доступа к общим данным между потоками.

На основе данных полученного параллельного ускорения $S = t_1/t_n$ для представленных выше алгоритмов послойного разделения выполнено определение масштабируемости (4), (5) параллельной операции сборки векторов $q = \mathcal{A}(\tilde{q})$ на неструктурированной сетке для нескольких архитектур и технологии параллельного программирования с ростом числа параллельных процессов n для двух технологий OpenMP и CUDA (см. таблица 1). Численные эксперименты выполнялись над сборкой векторов-результатов поэлементного

Таблица 1: Параметры модели USL для многоядерных процессоров при сборке вектора $q = \mathcal{A}(\tilde{q})$ на неструктурированной сетке

Параметры	Xeon E5-2609	Opteron 8435	Xeon E5-2690	Xeon Phi 7110X	GeForce GTX 980
n	4	6	8	61	2048
σ	0.1	0.021	0.08	0.006	0.0018
λ	0.0001	0.02	0.0001	0.00006	0.00000044
R^2	0.99	0.982	0.9965	0.994	0.999
$S_{U\max}/n_{U_{max}}$	8.41/ 94	3.56/7	10.09/96	46.76/129	318.96/1511
$S_{\max}/n_{\max}$	3.1/4	3.48/5	5.19/8	39.43/60	219/384

матрично-векторного произведения для произвольной неструктурированной сетки со сгущениями ячеек, представленной на рисунке 1. Для сравнения параметров модели Гюнтера на различных архитектурах процессоров было выбрано разделение по чётности номеров слоёв с выравниванием нагрузки, оптимизирующее движение данных и обращения к памяти. Таким образом, вычисления проводились над одними данными, для их разделения по числу потоков использовался один тот же алгоритм, программное обеспечение OpenMP с распараллеливанием цикла по конечным элементам по номеру потока. Для повышения эффективности выполнения операции сборки на GPU алгоритм модифицировался появлением новых уровней распараллеливания.

Для нахождения параметров модели (4) использовался нелинейный регрессионный анализ. Полученные данные приведены в таблице 1. Степень соответствия между эксперимен-

тальными данными S и расчетными по USL — S_U оценивается мерой R^2. Полученные значения R^2 объясняют изменения ускорения зависимостью от n при некоторых σ и λ.

Полученные параметры модели σ, λ позволили оценить также число процессорных ядер, при котором достигается максимальное ускорение $n_{U_{\max}} = \sqrt{(1-\sigma)/\lambda}$ для рассматриваемого алгоритма, выполняемого на центральных процессорах и ускорителях вычислений. Среди полученных результатов отметим замедление вычислений при параллельной сбор-

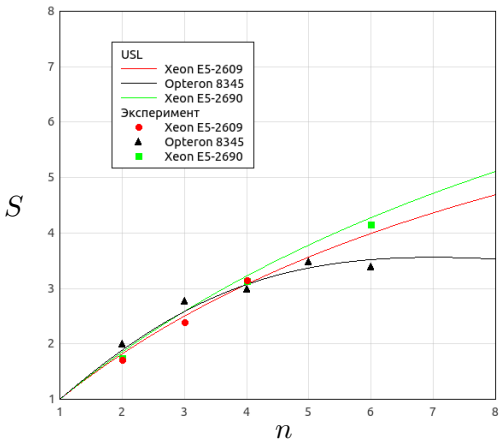


Рис. 4: Масштабируемость операции сборки поэлементных векторов на универсальном процессоре

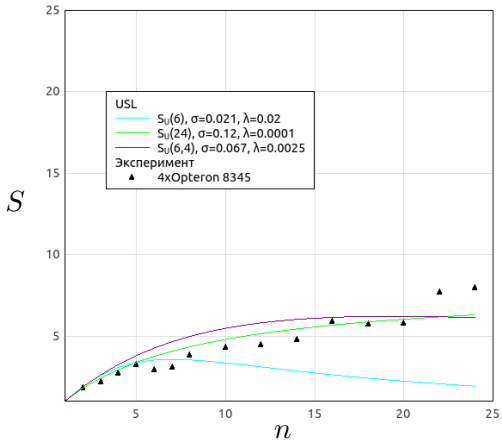


Рис. 5: Масштабируемость операции сборки поэлементных векторов на многосокетовом узле

ке векторов в модели общей памяти для графового разделения с критической секцией. В предложенных алгоритмах послойного разделения критическая секция не требуется, все ограничения по доступу к общей памяти разрешаются и как следствие, достигается хорошая масштабируемость на универсальных процессорах и ускорителях вычислений.

На рисунке 4 приведены экспериментальные результаты (маркеры) измерения ускорения операции сборки для различных архитектур универсальных процессоров и кривые, построенные на основе модели USL по (4) . Существенное трех-пятикратное ускорение параллельной сборки достигается даже для неструктурированных сеток, которое составляет примерно половину максимально возможного ускорения при увеличении числа ядер на порядок для архитектуры процессоров Xeon E5. На шестиядерных процессорах Opteron с трехуровневой организацией кэша издержки на согласовании размещения данных приводят к тому, что максимальное ускорение наблюдается только на пяти задействованных ядрах.

Поэлементная сборка векторов может эффективно выполняться и на ускорителях вычислений (см. рисунок 6), что подтверждают экспериментальные результаты и оценка параметров модели USL для перспективных ускорителей MIC с увеличенным числом ядер. Для сравнения были выполнены эксперименты на квазиструктурированных сетках, операция сборки на MIC выполняется практически с той же масштабируемостью, что и соответствует полученным значениям параметров модели σ, λ, характеризующих когерентность кэша между всеми используемыми ядрами (ускорение примерно в 40 раз на 61-ом ядре). В рассматриваемом алгоритме сборки доступ к памяти скрыт за темпом вычислений (минимальные параметры σ, λ среди всех архитектур) при вычислениях на графическом ускорителе GTX 980 и максимальное ускорение могло быть достигнуто на такой архитектуре при 1511 ядрах (см. рисунок 7). Экспериментально достигнутое ускорение ограничено только возможностью на заданной сетке выделить требуемое и сбалансированное число слоёв и наличием соответствующего объема оперативной памяти.

Так как универсальная модель не имеет внутренней структуры, она может быть использована для моделирования более общих, но только однородных вычислительных систем, таких как многопроцессорные (многосокетовые) вычислительные узлы, содержащие

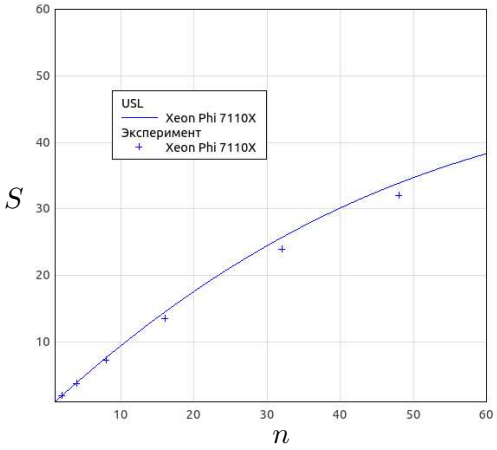


Рис. 6: Масштабируемость операции сборки поэлементных векторов на ускорителях вычислений MIC

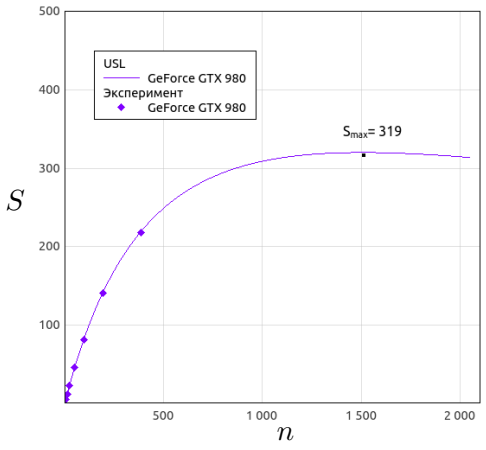


Рис. 7: Масштабируемость операции сборки поэлементных векторов на ускорителе вычислений GPU

одинаковые процессоры и кластеры, состоящие из вычислительных узлов с процессорами той же архитектуры и производительности.

Определим параметры многопроцессорного вычислительного узла как: p — число многоядерных процессоров в одном вычислительном узле; $n_p = n/p$ — число вычислительных ядер, приходящихся на один процессор; σ_p — параметр, характеризующий конкуренцию между процессорами; λ_p — согласованность данных между несколькими процессорами одного вычислительного узла. Тогда можно записать обобщение соотношения (4) для определения универсальной масштабируемости вычислительного узла, содержащего p процессоров/сокетов

$$S_U(n,p) = \frac{pS(n)}{1 + \sigma_p(p-1)S(n) + \lambda_p(p-1)pS^2(n)}, \tag{5}$$

где $S_U(n,p)$ — ускорение, определяемое и учитывающее накладные расходы многосокетового узла исходя из ускорения одного многоядерного процессора $S(n)$. При $p = 1$, $\sigma_p = 0$ последнее выражение преобразуется в (4). Для использования данной модели при оценке ускорения многосокетового узла $S_U(n,p)$ необходимо учесть, что все его процессоры также должны быть однородными. В дальнейшем, соотношение (5) может быть также обобщено для оценки ускорения вычислительного кластера $S_U(n,p,n_c)$, содержащего n_c — вычислительных узлов, соединенных коммуникационной сетью и определением соответствующих параметров σ_c, λ_c.

Для четырехсокетового узла с процессорами Opteron 8345 и неоднородным доступом к памяти были проведены измерения вычислительных затрат и трёх оценок ускорения (см. рис. 5). Первая кривая $S_u(6)$ построена по данным одного шестиядерного процессора и показывает существенное отличие от экспериментальных результатов при модельном увеличения числа ядер. Кривая, обозначаемая $S_U(24)$, получена при аппроксимации результатов по модели (4) при выполнении операции сборки на всех доступных двацатичетырех ядрах без учета неоднородного доступа к памяти соседних процессоров. В этом случае, оценка ускорения достигает семи при наличии большого числа ядер. При учете модели ускорения для одного процессора (4) и NUMA архитектуры в четырехсокетовом узле в рамках модели (5) получены более достоверные результаты по прогнозируемым максимальным ускорениям порядка шести.

Результаты моделирования ускорения достаточно хорошо характеризуют используемую модель параллельных вычислений при рассмотрении полученных ускорений для одного и того же алгоритма и разделения данных на разных типах многоядерных архитектур.

На основе данной модели возможно рассмотрение применимости изменений алгоритмов, архитектуры или программного обеспечения с целью минимизации значения параметров σ, λ и сокращения издержек.

Послойным разделением можно улучшить локальность и некоторых других параллельных алгоритмов метода конечных элементов, декомпозиции области, требующих групповых коммуникационных операций (обращение каждого процесса к локальной памяти всех других процессов). Новизна построенных алгоритмов определяется тем, что разрабатываемые алгоритмы базируются на модели вычислений, актуальной для современных и перспективных вычислительных систем и учитывающей несколько уровней иерархий памяти и параллельности, согласованных между собой. Исследования масштабируемости алгоритмов в модели USL позволяют спрогнозировать поведения подсистемы памяти, качества распараллеливания и показывают дальнейшие пути к улучшению достижимой производительности конечно-элементных вычислений на неструктурированных сетках.

Литература

1. Hill M. D., Marty M. R. Amdahl's Law in the Multicore Era // Computer. 2008. Vol. 41, No. 7. P. 33–38.
2. Yavits L., Morad A., Ginosar R. The effect of communication and synchronization on Amdahl's law in multicore systems // Parallel Computing. 2014. Vol. 40, No 1. P. 1-16.
3. Al-Babtain B.M., Al-Kanderi F.J., Al-Fahad M.F., Ahmad I. A Survey on Amdahl's Law Extension in Multicore Architectures // International Journal of New Computer Architectures and their Applications. 2013. Vol. 3. No 3. P. 30-46.
4. Gunther N. J. A new interpretation of Amdahl's law and geometric scalability // CoRR. 2002. Vol. cs.DC/0210017. URL: http://arxiv.org/abs/ cs.DC/0210017.
5. Gunther N. J., Puglia P., Tomasette K. Hadoop superlinear scalability // Communications of the ACM. 2015. Vol. 58, No. 4. P. 46–55.
6. Markall G. R., Slemmer A., Ham D. A., Kelly P. H. J., Cantwell C. D., Sherwin S. J. Finite element assembly strategies on multi-core and many-core architectures // Int. J. Numer. Meth. Fluids. 2013. Vol. 71, No. 1. P. 80-97.
7. Копысов С.П., Новиков А.К., Пиминова Н.К. Параллельная композиция в поэлементной схеме метода конечных элементов. Параллельные вычислительные технологии (ПаВТ'2016): Труды межд. науч. конф. (Архангельск, 28 марта – 1 апреля 2016 г.). Челябинск: Изд. центр ЮУрГУ, 2016. С. 555-560.
8. Löhner R. Cache-efficient renumbering for vectorization // International Journal for Numerical Methods in Biomedical Engineering. 2010. Vol. 26, No 5. P. 628–636.
9. Giuliani A., Krivodonova L. Edge coloring in unstructured CFD codes // CoRR. 2016. Vol. abs/1601.07613. URL: http://arxiv.org/abs/1601.07613.
10. Kopysov S. P., Novikov A. K. Parallel element-by-element conjugate gradients method with decreased communications costs // Int. Summer School "Iterative Methods and Matrix Computations". Rostov-on-Don : Rostov State University, 2002. P. 450-454.
11. Karypis G., Kumar V. A fast and high quality multilevel scheme for partitioning irregular graphs // SIAM Journal on Scientific Computing. 1999. Vol. 20, № 1. P. 359-392.

Scalability element-by-element FEM algorithms on manycore platforms

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

Institute of Mechanics UB RAS

In this work, the Universal Law of Computational Scalability is used to evaluate the performance of FEM algorithms on multicore/manycore platforms. The evaluation is focused on the algorithms with layer-by-layer mesh partitioining and their most critical operation, matrix-vector multiplication.

Keywords: finite element, partitioning schemes, element-by-element matrix-vector multiplication, multicore processors, universal law of computational scalability.

References

1. Hill M. D., Marty M. R. Amdahl's Law in the Multicore Era // Computer. 2008. Vol. 41, No. 7. P. 33–38.
2. Yavits L., Morad A., Ginosar R. The effect of communication and synchronization on Amdahl's law in multicore systems // Parallel Computing. 2014. Vol. 40, No 1. P. 1-16.
3. Al-Babtain B.M., Al-Kanderi F.J., Al-Fahad M.F., Ahmad I. A Survey on Amdahl's Law Extension in Multicore Architectures // International Journal of New Computer Architectures and their Applications. 2013. Vol. 3. No 3. P. 30-46.
4. Gunther N. J. A new interpretation of Amdahl's law and geometric scalability // CoRR. 2002. Vol. cs.DC/0210017. URL: http://arxiv.org/abs/ cs.DC/0210017.
5. Gunther N. J., Puglia P., Tomasette K. Hadoop superlinear scalability // Communications of the ACM. 2015. Vol. 58, No. 4. P. 46–55.
6. Markall G. R., Slemmer A., Ham D. A., Kelly P. H. J., Cantwell C. D., Sherwin S. J. Finite element assembly strategies on multi-core and many-core architectures // Int. J. Numer. Meth. Fluids. 2013. Vol. 71, No. 1. P. 80-97.
7. Kopysov S.P., Novikov A.K., Piminova N.K. Parallel'naya kompozitsiya v poelementnoy skheme metoda konechnykh elementov. Parallel'nye vychislitel'nye tekhnologii (PaVT'2016): Trudy mezhd. nauch. konf. (Arkhangel'sk, 28 Marta – 1 Aprelya 2016) [Parallel Computational Technologies (PCT'2016): Proceedings of the International Scientific Conference (Arhangelsk, Russia, March, 28 – April, 1, 2016)]. Chelyabinsk: Publishing of the South Ural State University , 2016. P. 555-560.
8. Löohner R. Cache-efficient renumbering for vectorization // International Journal for Numerical Methods in Biomedical Engineering. 2010. Vol. 26, No 5. P. 628–636.
9. Giuliani A., Krivodonova L. Edge coloring in unstructured CFD codes // CoRR. 2016. Vol. abs/1601.07613. URL: http://arxiv.org/abs/1601.07613.
10. Kopysov S. P., Novikov A. K. Parallel element-by-element conjugate gradients method with decreased communications costs. Int. Summer School "Iterative Methods and Matrix Computations". Rostov-on-Don : Rostov State University, 2002. P. 450- 454.
11. Karypis G., Kumar V. A fast and high quality multilevel scheme for partitioning irregular graphs // SIAM Journal on Scientific Computing. 1999. Vol. 20, No 1. P. 359-392.