УДК 539.3

# О ПАРАЛЛЕЛЬНЫХ РЕШАТЕЛЯХ В КОНЕЧНОЭЛЕМЕНТНЫХ ПРОГРАММНЫХ КОМПЛЕКСАХ, ОРИЕНТИРОВАННЫХ НА МНОГОЯДЕРНЫЕ КОМПЬЮТЕРЫ С ОБЩЕЙ ОПЕРАТИВНОЙ ПАМЯТЬЮ

**С.Ю. Фиалко**[1]
д-р. техн. наук

[1]*Tadeusz Kościuszko Cracow University of Technology*

Рассматриваются прямые методы решения систем линейных алгебраических уравнений с симметричными разреженными матрицами, возникающими при применении метода конечных элементов к задачам строительной механики и механики деформированного твердого тела. Основное внимание уделяется методам достижения высокой производительности на многоядерных компьютерах с общей оперативной памятью (sharedmemorycomputers).

**Ключевые слова:** многофронтальный метод, суперузловой метод, метод конечных элементов, граф смежности, высокая производительность, параллельные вычисления.

**Введение.** Интенсивное развитие многоядерных компьютеров порождает необходимость совершенствования решателей – модулей конечноэлементных программных комплексов, ответственных за решение систем линейных алгебраических уравнений с разреженными матрицами.

В данной работе мы ограничимся рассмотрением прямых методов для симметричных разреженных матриц, возникающих при применении метода конечных элементов к задачам строительной механики и механики деформируемого твердого тела. Основное внимание будет уделено методам достижения высокой производительности на многоядерных компьютерах, массово используемых при решении таких задач.

«Узким местом» такой архитектуры является ограниченная пропускная способность системы оперативной памяти, разделяемой несколькими процессорами. Ключом к достижению высокой производительности и стабильной ускоряемости при увеличении количества процессоров является рациональное использование быстрой памяти кэш процессоров. Объем этой памяти ограничен, поэтому алгоритм, преобразовывающий большой объем данных, должен «уметь» данные, один раз помещенные в кэш, многократно использовать – реализовывать концепцию «cachereuse». Далеко не каждый алгоритм может с этим справиться. Поэтому возникает вопрос: а к каким вычислительным алгоритмам должна быть сведена данная задача (если это возможно), чтобы ускоряемость была стабильной? Одним из таких алгоритмов является матричное умножение плотных матриц *dgemm* –

generalmatrixmultiplication. Таким образом, возникает проблема сведения разложения разреженной матрицы высокого порядка к последовательности умножений и сложений плотных матриц относительно небольшой размерности.

Для повышения производительности вычислительных алгоритмов используется также векторизация вычислений, основанная на том, чтосовременные процессоры многоядерных компьютеров имеют специальные регистры, позволяющие за один цикл процессора выполнить несколько сложений или умножений. Однако для использования этих регистров необходимо разработать специальный алгоритм, поскольку обычный алгоритм для векторизации вычислений чаще всего оказывается непригодным.

**Постановка задачи.** Математическая формулировка задачи представляется так:

$$\mathbf{KX} = \mathbf{B}, \tag{1}$$

где **K–** разреженная симметричная матрица жесткости, **X** и **B** – пакет неизвестных и пакет векторов правой части (нагрузок). Решение данной задачи прямым методом состоит в разложении (факторизации) матрицы

$$\mathbf{K} = \mathbf{L} \cdot \mathbf{S}' \cdot \mathbf{L}^T, \tag{2}$$

где **L** – разреженная нижняя треугольная матрица, $\mathbf{S}'$ – диагональ знаков.

Затем выполняется прямая подстановка

$$\mathbf{LY} = \mathbf{B} \quad \rightarrow \quad \mathbf{Y}, \tag{3}$$

диагональное масштабирование

$$\mathbf{S}'\mathbf{Z} = \mathbf{Y} \quad \rightarrow \quad \mathbf{Z}, \tag{4}$$

и обратная подстановка

$$\mathbf{L}^T\mathbf{X} = \mathbf{Z} \quad \rightarrow \quad \mathbf{X}. \tag{5}$$

Часто вместо $\mathbf{LSL}^T$ разложения используется $\mathbf{LDL}^T$, где **D** – диагональная матрица [1].

**Прямые методы для разреженных матриц.** Одной из первых фундаментальных публикаций, посвященных методам решения систем линейных алгебраических уравнений (СЛАУ) с симметричными разреженными матрицами, является монография [2]. Описанные в ней методы упорядочения эффективно уменьшают количество ненулевых элементов в нижней треугольной матрице **L**, а представленные решатели идеально обходят операции с нулевыми элементами разреженных матриц. Однако недостатками этих решателей является отсутствие виртуализации (эти методы работают только в оперативной памяти) и распараллеливания, а также низкая производительность, обусловленная тем, что во внутренних циклах выполняются скалярно-векторные операции – вычисление скалярного произведения векторов или умножение вектора на скаляр. Эти алгоритмы содержат количество данных O($n$) и выполняют количество

операций также O($n$) [3]. Поэтому реализовать концепцию cachereuse для таких алгоритмов принципиально невозможно, и данные алгоритмы работают со скоростью медленной системы памяти, а процессор выполняет большое количество пустых циклов. Кроме того, хранение элементов разреженной матрицы столбец-по-столбцу в сжатом формате [2] не позволяет эффективно применить векторизацию вычислений.

**Многофронтальные методы.** В конце 90-х – начале 2000-х широкое применение получили многофронтальные решатели [5], [6], которые устранили перечисленные выше недостатки. Они используют любое упорядочение, строят дерево исключения и находят на этом дереве суперузлы. Дерево исключения – это структура данных, представляющая последовательность факторизации столбцов разреженной матрицы и зависимость между столбцами – какой из столбцов матрицы, расположенных справа от текущего, будет им поправляться в первую очередь. Каждая вершина такого дерева соответствует столбцу разреженной матрицы, а каждое ребро – зависимости между двумя столбцами. Если множество вершин дерева исключения принадлежит одной и той же ветви и имеют последовательную нумерацию, то такие вершины объединяются в суперузел. Каждому суперузлу в разреженной матрице соответствует плотный блок при главной диагонали, что позволяет объединить соответствующие столбцы матрицы в блок при появлении минимального количества нулевых элементов в блоке, расположенном ниже диагонального блоках.

Объединение столбцов в блоки является ключевым моментом в увеличении производительности. В разреженной матрице мы не можем объединять столбцы произвольным способом, поскольку при этом происходит добавление большого количества нулевых элементов. Метод суперузлов позволяет сделать это более рационально.

Для объединенных столбцов при факторизации матрицы применяется алгоритм матричного умножения *dgemm*. В отличие от упомянутых выше алгоритмов умножения вектора на скаляр и вычисления скалярного произведения векторов, алгоритм матричного умножения хранит количество данных порядка $O(n^2)$ и выполняет количество арифметических операций порядка $O(n^3)$, что позволяет эффективно использовать память кэш процессора и регистры. Современные реализации *dgemm*, например[7],включают векторизацию вычислений, “cachereuse”, “registerreuse” (данные, один раз загруженные в регистры процессора, должны использоваться как можно большее число раз),переупаковку данных, уменьшающую количество кэш-промахов, а также максимальное использование конвейеров процессора [4], [8].

Поэтому данная реализация матричного умножения работает со скоростью быстрого процессора, а не медленной системы памяти.

Затем в соответствии с полученным суперузловым деревом исключений производится факторизация. В плотную матрицу, называемую фронтальной, считываются столбцы исходной матрицы, номера которых соответствуют номерам вершин, объединенных в данном суперузле. Нулевые строки во фронтальную матрицу не включаются. Каждой строке и каждому столбцу фронтальной матрицы ставится в соответствие номер глобального уравнения матрицы жесткости **K**. Затем к образованной таким образом фронтальной матрице добавляются матрицы поправок с суперузлов – предшественников данного суперузла. При этом алгебраически суммируются элементы матриц, имеющих одинаковые глобальные индексы. Если суперузел не имеет предшественников, то фронтальная матрица образуется только из элементов соответствующих столбцов исходной матрицы жесткости. Подробности изложены в [5], [6].

В результате на каждом шаге фронтальной факторизации выполняется частичное разложение фронтальной матрицы

$$\begin{pmatrix} \mathbf{A} & \mathbf{W}^T \\ \mathbf{W} & \mathbf{M} \end{pmatrix} = \begin{pmatrix} \mathbf{L}' & 0 \\ \tilde{\mathbf{W}} & \mathbf{U} \end{pmatrix} \cdot \begin{pmatrix} \mathbf{S} & 0 \\ 0 & \mathbf{I} \end{pmatrix} \cdot \begin{pmatrix} (\mathbf{L}')^T & \tilde{\mathbf{W}}^T \\ 0 & \mathbf{I} \end{pmatrix}, \tag{6}$$

где глобальные номера столбцов блоков **A**, **W** соответствуют номерам вершин данного суперузла, причем глобальные номера строк и столбцов блока **A** соответствуют номерам строк и столбцов плотного блока при главной диагонали нижней треугольной матрицы **L**; элементы матрицы **M** образуются при суммировании матриц поправок суперузлов – предшественников (при их отсутствии это нулевая матрица); **U** – матрица поправок для данного суперузла; $\mathbf{L}'$ – плотная нижняя треугольная матрица, **S** – диагональ знаков, **I**– единичная матрица. Из (6) следует:

$$\begin{aligned} &1.\quad \mathbf{A} = \mathbf{L}'\mathbf{S}(\mathbf{L}')^T \quad \rightarrow \quad \mathbf{L}', \mathbf{S} \\ &2.\quad \mathbf{L}'\mathbf{R} = \mathbf{W}^T \quad \rightarrow \quad \mathbf{R} \\ &3.\quad \mathbf{S}\tilde{\mathbf{W}}^T = \mathbf{R} \quad \rightarrow \quad \tilde{\mathbf{W}} \\ &4.\quad \mathbf{U} = \mathbf{M} - \tilde{\mathbf{W}}\mathbf{S}\tilde{\mathbf{W}}^T. \end{aligned} \tag{7}$$

На первом этапе факторизуется диагональный блок и определяются блок $\mathbf{L}'$ и диагональ знаков **S**. На втором этапе определяется блок **R** из решения СЛАУ с нижней треугольной матрицей $\mathbf{L}'$. На третьем этапе выполняется диагональное масштабирование и определяется блок $\tilde{\mathbf{W}}$. И на четвертом – вычисляется выражение, подобное дополнению Шура.

На всех этапах выполняются операции с плотными матрицами, причем соответствующие процедуры представлены библиотеками процедур высокой производительности, например [7]. Количество арифметических операций, выполняемых на последнем этапе, на несколько порядков больше, чем на всех предыдущих этапах. Именно здесь особенно важно достичь высокой производительности, что осуществляется применением алгоритма матричного умножения *dgemm*.

Следует отметить работу [9], в которой используется двухуровневая схема распараллеливания. Первый уровень основан на том, что для разреженных матриц высокого порядка суперузловое дерево исключений существенно разветвлено. Каждая ветвь этого дерева может обрабатываться на отдельном процессоре. Поскольку объем вычислений для каждой ветви существенно различный, на этом уровне не удается достичь сбалансированности нагрузки на процессоры. Кроме того, верхняя часть суперузлового дерева исключений разветвлена слабо и содержит фронтальные матрицы относительно большой размерности. Поэтому на втором уровне распараллеливается факторизация фронтальных матриц.

Представленные методы относятся к классу алгебраических решателей. Это означает, что входной информацией является собранная матрица жесткости **K**, хранимая столбец по столбцу в сжатом формате. Никакой другой информации не требуется.

В 1999–2000 годах автором данной статьи был разработан мультифронтальный метод подконструкций BSMFM (block substructure multifrontal method), представляющий собой суперэлементный подход с автоматическим разделением исходной конечноэлементной модели на подконструкции [4], [10], [11].Исходной информацией для данного метода является список узлов для каждого конечного элемента. Затем строится граф смежности для узлов, однозначно представляющий топологию расчетной модели. Далее выполняется упорядочение этого графа. Обычно в каждом узле мы имеем несколько уравнений, поэтому каждая вершина графа смежности для узлов соответствует плотному блоку в разреженной матрице. Кроме того, граф смежности для узлов содержит в несколько раз меньше информации, чем граф смежности для разреженной матрицы, используемый в алгебраических методах. Поэтому алгоритмы упорядочения в представляемом методе работают значительно быстрее и приводят к несколько меньшему количеству ненулевых элементов в факторизованной матрице [20].

Алгоритм упорядочения определяет последовательность исключения узлов расчетной модели, где под исключением узла мы понимаем исключение группы уравнений, связанных с этим узлом. Для того, чтобы

исключить узел, необходимо, чтобы все конечные элементы, сходящиеся в данном узле, были поданы на сборку. Такие узлы мы называем собранными. Каждый конечный элемент может участвовать в сборке только один раз. Кроме того, исключаемый узел может входить в состав частично собранных подконструкций на предыдущих шагах. Поэтому все подконструкции, образовавшиеся на предыдущих шагах и содержащие исключаемый на данном шаге узел, также участвуют в сборке.

Выполняется символическая факторизация, которая, опираясь на последовательность исключаемых узлов, списки узлов для каждого конечного элемента и каждой ранее собранной подконструкции, формирует дерево фронтов, каждый узел которого соответствует отдельной подконструкции с предыдущих шагов, причем исключенные ранее узлы в соответствующие списки узлов не входят. Вершины фронтального дерева перенумеровываются так, чтобы минимизировать объем памяти, предназначенной для хранения матриц поправок и, следовательно, количество медленных операций считывания элементов этих матриц из памяти или с диска. Далее происходит объединение вершин на последовательных участках фронтального дерева, что дает возможность на каждом шаге увеличить количество полностью собранных уравнений с целью повышения производительности вычислений.

Фронтальная матрица на каждом шаге формируется из уравнений, соответствующих не исключённым узлам текущей подконструкции. Полностью собранные уравнения образуют блоки **A** и **W** в (6). Далее применяется процедура (7), причем блоки полностью собранных уравнений копируются в специальный буфер, и их факторизация осуществляется в соответствующих ему адресах памяти. При переполнении этот буфер выгружается на диск. В случае дефицита оперативной памяти матрица **U** также записывается на диск, откуда в последствии считывается при сборке фронтов, соответствующих узловым вершинам фронтального дерева. Для больших задач, размерность которых значительно превышает возможности оперативной памяти, факторизованная матрица образуется на диске в самом конце разложения в виде плотных матричных блоков $\mathbf{L}', \widetilde{\mathbf{W}}$. При выполнении прямой подстановки, диагонального масштабирования и обратной подстановки эти блоки с помощью глобальных индексов используются без распаковки в глобальную матрицу **L**.

Оказалось, что алгоритмы для плотных матриц размерностью n, реализованные в библиотеках высокой производительности, даже в случае симметричных матриц требуют хранения $n^2$ элементов. Поэтому вместо использования процедуры dgemm на этапе 4 выражения (7)было разработано свое микроядро – программный код низкого уровня, напрямую использующий векторизацию вычислений и

распараллеливание, что позволило для каждой фронтальной матрицы хранить $n(n+1)/2 \approx n^2/2$ элементов. Детали изложены в [4]. Данный решатель внедрен в программный комплекс SCAD (www.scadsoft.com) и успешно используется там с 2002 г.

**Суперузловые методы.** Также как и многофронтальные суперузловые (supernodal) методы [12], [13], [14]относятся к классу прямых методов для разреженных матриц. Прежде всего строится граф смежности для разреженной матрицы жесткости. Затем производится упорядочение тем или иным методом, после чего выполняется символическая факторизация, определяющая положения ненулевых элементов факторизованной матрицы. Символическая факторизация производится на графе смежности [2], поэтому даже для матриц размерностью несколько миллионов уравнений длится всего несколько секунд. В результате создается ненулевая структура факторизованной матрицы, определяющая позиции всех ее ненулевых элементов.

Затем строится дерево исключений, на котором определяются суперузлы. Каждому суперузлу соответствует плотный блок при главной диагонали факторизованной матрицы. Выделение плотных блоков при главной диагонали определяет деление всей матрицы на блоки. К полученной таким образом блочной структуре матрицы применяется тот или иной алгоритм факторизации, причем вместо операций с отдельными элементами выполняются матричные операции над блоками. Ведущей процедурой при факторизации блочной матрицы является матричное умножение *dgemm*. Таким образом осуществляется замена типичных для наивных алгоритмов факторизации низкопроизводительных скалярно-векторных операций во внутреннем цикле на высокопроизводительное матричное умножение.

Одним из наиболее высокопроизводительных и хорошо ускоряемых с увеличением количества процессоров в архитектуре SMP (symmetrical multiprocessing), к которой относятся многоядерные компьютеры, является суперузловой метод PARDISO [15], включенный в библиотеку IntelMKL [7]. По сравнению с существующими многофронтальными методами для компьютеров с общей памятью PARDISO демонстрирует существенно более высокую производительность и ускоряемость. Объясняется это тем, что многофронтальные методы содержат избыточные пересылки данных «память – память», которые существенно ограничивают их производитель ность и ускоряемость при увеличении количества процессоров на компьютерах с общей памятью.

В PARDISO реализовано двухуровневое распараллеливание. Первый уровень связан с разветвлением суперузлового дерева исключений. При этом потоки, закончившие свою работу на первом уровне, не дожидаясь окончания работы соседей, начинают поправлять блок-столбцы,

размещенные правее (второй уровень распараллеливания). Оригинальный алгоритм двухуровневого распараллеливания обеспечивает хорошее сбалансирование вычислительной нагрузки на процессоры даже при большом количестве потоков, что обеспечивает устойчивый рост производительности при увеличении количества процессоров.

Однако существенным недостатком PARDISO является то, что решатель работает только в оперативной памяти. Формально имеется режим OOC (out of core), в котором подключается дисковая память. Однако исследования автора данной статьи, а также и другие публикации, например[16], показывают, что для небольших задач PARDISOв режиме OOC работает значительно медленнее многофронтального метода, а для больших –приводит к аварийному завершению. Указанный недостаток делает PARDISO, как и другие суперузловые методы, представленные библиотеками высокой производительности, практически неприменимыми для решения больших конечноэлементных задач на офисных компьютерахс ограниченным объемом оперативной памяти.

Решатель PARFES (parallel finite element solver) [17], [18], [19] является также суперузловым методом с одноуровневым распараллеливанием, которое позволяет реализовать эффективный обмен с диском, если размерность задачи превышает объем оперативной памяти.

В качестве входной информации PARFES использует граф смежности для узлов конечноэлементной модели. Выполняется упорядочение этого графа и строится дерево исключений. Затем определяются суперузлы, в результате чего столбцы разреженной матрицы объединяются в блоки. Типичная структура блок-столбца представлена на рис. 2. Все блок-столбцы имеют плотные блоки при главной диагонали. Внедиагональные блоки могут быть пустыми, полностью заполненными и частично заполненными. Для пустых блоков память не распределяется, а для частично заполненных блоков хранятся только ненулевые строки.

В зависимости от размерности решаемой задачи и объема оперативной памяти автоматически выбирается один из трех режимов факторизации: CM, OOC и OOC1. Если задача может быть полностью решена в оперативной памяти, назначается режим CM (core memory). Режим OOC назначается для задач «средней размерности». При этом производительность и ускорямость решателя снижаются незначительно по сравнению с режимом CM, поскольку количество обменов с диском относительно невелико. Если же размерность задачи не позволяет использовать режим OOC, то назначается режим OOC1. Количество обменов с диском существенно возрастает, производительность и ускоряемость значительно снижаются, однако данный режим позволяет решать большие задачи на компьютерах с малым объемом оперативной памяти [18].

При сборке глобальной матрицы жесткости в режиме СМ ненулевые элементы располагаются в блоках. В режимах ООС и ООС1 собранная матрица в сжатом формате записывается на диск.

При численной факторизации применяется блочный looking-left алгоритм Холецкого [17] (рис. 1).

1. **do** $jb = 1, N_b$, где $N_b$ – количество блок-столбцов.
2. В режимах ООС / ООС1 распределить память для блоков блок-строки $jb$ и заполнить элементами исходной матрицы. В режиме СМ пропустить этот шаг.
3. Выполнить поправку блок-столбца jb блок-столбцами, расположенными слева: $\mathbf{A}_{ib,jb} = \mathbf{A}_{ib,jb} - \sum_{kb \in List[jb]} \mathbf{L}_{ib,kb}\mathbf{S}_{kb}\mathbf{L}^T_{jb,kb}$; $ib \geq jb$, $ib \in \mathbf{L}_{kb}$ (рис. 2). Здесь $ib$, $jb$, $kb$ – индексы блоков блок-строк, $L_{kb}$ – ненулевая структура блок-столбца $kb$. В поправке принимают участие только те блок-столбцы, которые поправляют блок-столбец jb: $kb \in$ List[$jb$]. В режиме ООС1 каждый блок-столбец $kb$, поправляющий блок-столбец $jb$, считывается с диска.
4. Факторизовать диагональный блок блок-столбца $jb$: $\mathbf{A}_{jb,jb} = \mathbf{L}_{jb,jb}\mathbf{S}_{jb}\mathbf{L}^T_{jb,jb} \rightarrow \mathbf{L}_{jb,jb}, \mathbf{S}_{jb}$.
5. Преобразовать блоки блок-столбца $jb$, размещенные под диагональным блоком: $\mathbf{L}_{jb}\mathbf{S}_{jb}\mathbf{L}_{ib,jb} = \mathbf{A}_{ib,jb} \rightarrow \mathbf{L}_{ib,jb}$.
6. В режиме ООС / ООС1 записать факторизованный блок-столбец на диск: $\mathbf{L}_{ib,jb} \rightarrow disk, ib \geq jb, ib \in L_{jb}$. Освободить занимаемую им память. В режиме СМ пропустить этот шаг.
7. **end do**

Рис. 1. Блочный алгоритм looking-left факторизации

Более 99% вычислений осуществляется на этапе 3. Поэтому именно он распараллеливается в первую очередь. Для обеспечения сбалансированности вычислительной нагрузки на процессоры используется специальный алгоритм отображения заданий на потоки [17]. При этом целая блок-строка для блок-столбцов $kb \in$ List[$jb$] отображается на один и тот же поток, что позволяет избежать применения синхронизации при записи результатов поправок в блоки блок-столбца jb. Затем определяются веса – количество ненулевых элементов в каждой блок-строке. Номера блок-строк сортируются в порядке убывания весов, после чего поочередно приписываются тому потоку, который имеет на

данный момент минимальную сумму весов, а вес приписанной блок-строки добавляется к сумме весов потока. В результате для каждого потока создаются очереди задач $Q_{ip}$, где $ip$ – номер потока. Каждый элемент очереди состоит из указателей блоков $\mathbf{L}_{ib,kb}$, $\mathbf{L}_{jb,kb}$ и номера $kb$ диагонали знаков $\mathbf{S}_{kb}$.

Затем осуществляется процедура, изображенная на рис. 3. В параллельном регионе выполняются $ip$ потоков. Каждый поток исполняет цикл *while* до тех пор, пока очередь $Q[ip]$ не окажется пустой. В каждой итерации данного цикла из очереди $Q[ip]$ выбирается ближайший элемент и сразу вычеркивается (знак / , пункт 3). Получив указатели на блоки $\mathbf{L}_{ib,kb}$, $\mathbf{L}_{jb,kb}$ и номер $kb$, выполняем матричное умножение (пункт 4 и рис. 2 – низ).

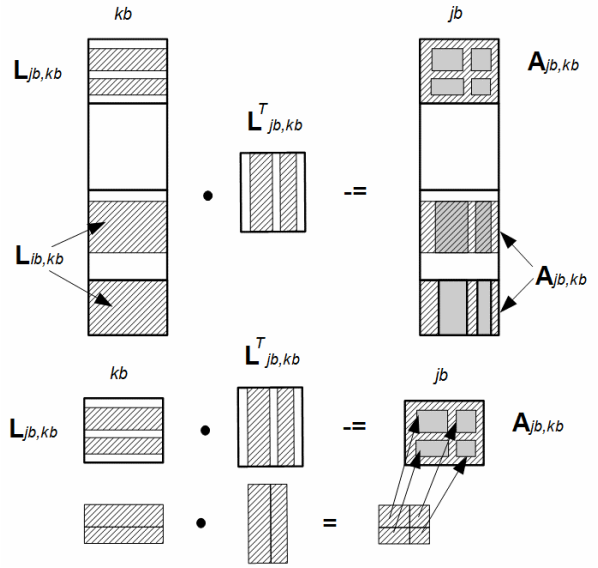


Рис. 2. Поправка блок-столбца $jb$ блок-столбцом $kb$, расположенным слева. Ненулевые строки в блоках заштрихованы. Серам цветом показаны поправляемые подблоки в блок-строке $jb$. Внизу показан пример упаковки при умножении блоков

1. начало параллельного региона: $ip \in [0, \mathrm{Pr}ocNumb-1]$
2. while(очередь$Q[ip]$ не пуста)
3. $\{\mathbf{L}_{ib,kb}, \mathbf{L}_{ib,kb}, kb\} \leftarrow Q[ip]$; $Q[ip] \leftarrow (Q[ip] \backslash \{\mathbf{L}_{ib,kb}, \mathbf{L}_{ib,kb}, kb\})$
4. $\mathbf{A}_{ib,jb} = \mathbf{A}_{ib,jb} - \mathbf{L}_{ib,kb}\mathbf{S}_{kb}\mathbf{L}^{T}_{jb,kb}$;
5. endwhile
6. конец параллельного региона

Рис. 3. Алгоритм параллельной поправки (см. рис. 1, пункт 3)

Насколько быстро будет выполняться матричное умножение, настолько быстро будет факторизована матрица. Поскольку для матриц небольшой размерности процедура матричного умножения существенно снижает производительность, для частично заполненных блоков выполняется упаковка (рис. 2, низ).

При использовании процедуры *dgemm* из библиотеки IntelMKL производительность PARFES на процессорах AMDOpteron архитектуры Bulldozer оказалась сниженной. Процедура *dgemm* из библиотеки ACML (AMD core math library), адаптированная под процессоры AMD, в отдельных тестах матричного умножения показывает хорошие результаты, однако при ее использовании в однопоточном режиме в алгоритме PARFES, представленном на рис. 3, приводит к деградации производительности. Вероятно, замысел разработчиков ACML рассчитан на то, что при многопоточных вычислениях на процессорах AMD необходимо использовать многопоточную версию процедуры *dgemm*. Однако алгоритм PARFES (рис. 3) требует использования процедуры матричного умножения в однопоточном режиме во избежание конфликта между распараллеливанием внешними распараллеливанием внутри многопоточной процедуры *dgemm*, в результате чего происходит деградация производительности вычислений.

Поэтому была разработана своя низкоуровневая процедура *microkern*_8x4_*AVX* с целью обеспечить векторизацию вычислений, «registerreuse» и оптимизировать конвейеры процессора [19]. Оказалось, что данная процедура не уступает в производительности процедуре *dgemm* из библиотеки IntelMKL.

С 2014 г. PARFES включен в набор решателей SCAD 21.

**Численные результаты.** В примере 1 приводится сравнение PARFES, BSMFM, PARDISO из библиотеки IntelMKL 10.2.2.025. и многофронтального решателя Boeingsparsesolver именитого программного комплекса ANSYS 15.0 (табл. 1, 2). Исследования проводились на компьютере с 4-ядерным процессором IntelCore™ i7-27600QM, 2.4/3.4 *ГГц*, RAM DDR3 8*GB*, OS Windows 7 (64 bit) Professional, SP1. Для того, чтобы обеспечить идентичность расчетных моделей, приготавливаемых различными программными комплексами ANSYS 15.0 и SCAD 21.1,была создана модельная задача – простая модель, топологически подобная многоэтажному зданию с несущими стенами. Размерность задачи составляет 1 236 246 уравнений.

Все решатели работали в оперативной памяти при использовании четырех потоков (*ProcNumb*=4). Решатель ANSYS записывает разложенную матрицу на диск, поэтому в колонке «режим работы с ОП» поставлен прочерк.

Для решателя ANSYS 15.0 приводится чистое время численной факторизации. Для мультифронтального решателя BSFMF приведенное время содержит еще и продолжительность процесса агрегации, который неотделим от численной факторизации и составляет для данной задачи 5 *с*. Таким образом, производительность этих двух решателей отличается незначительно.

Таблица 1

Продолжительность численной факторизации для различных решателей, задача 1, размерность расчетной модели – 1 236 246 уравнений

| Метод | Продолжительность численной факторизации, *с* | Размер факторизованной матрицы, *GB* | Метод упорядочения | Режим работы с ОП |
|---|---|---|---|---|
| ANSYS 15.0 | 50 | 4.145 | ? | – |
| BSMFM | 63 | 3.175 | METIS | СМ |
| PARDISO | 26.5 | 3.679 | METIS | СМ |
| PARFES | 28.8 | 3.771 | METIS | СМ |

Метод упорядочения, используемый решателем ANSYS, неизвестен, поэтому в соответствующей позиции в таблице 1 стоит знак вопроса. Однако размер факторизованной матрицы в МВ оказался больше, чем для решателя BSMFM. Остальные решатели использовали упорядочение METIS [20].

PARFES и PARDISO показали практически одинаковое время.

В примере 2 рассматривается подобный объект, но с большим количеством этажей. Размерность задачи составляет 2 360 106 уравнений. Результаты приведены в таблице 2.

Таблица 2

Продолжительность численной факторизации для различных решателей, задача 2, размерность расчетной модели – 2 360 106 уравнений

| Метод | Продолжительность численной факторизации, *с* | Размер факторизованной матрицы, *GB* | Метод упорядочения | Режим работы с ОП |
|---|---|---|---|---|
| ANSYS 15.0 | 240 | 8.777 | ? | – |
| BSMFM | 285 | 6.561 | MMD | OOC |
| PARDISO | Не хватает объема ОП (error -2) | 7.513 | METIS | ОС |
| | Ошибка записи/ чтения при обращении к файлу данных OOC (error -11) | 7.530 | METIS | OOC |
| PARFES | 125 | 7.349 | MMD | OOC |

Многофронтальный решатель BSFMF работал несколько дольше, чем решатель ANSYS 15.0. Сборка матрицы жесткости составила 20 *с*.

PARDISO в режиме ОС не хватило объема оперативной памяти, а в режиме ООС произошла ошибка ввода-вывода. PARFES решил эту задачу примерно в два раза быстрее, чем решатель ANSYS 15.0.

Оба приведенных примера показывают, что рассмотренные суперузловые методы для компьютеров с общей памятью работают существенно быстрее, чем данные многофронтальные.

В примере 3 рассматривается задача из коллекции задач SCAD Soft. Расчетная модель представлена на рис. 4 и содержит 2 989 476 уравнений. Данная задача оказалась очень трудной для прямых методов, поскольку грунт моделируется объемными конечными элементами, которые порождают относительно высокую плотность разреженной матрицы, размер которой после факторизации составляет 37 *GB* (метод упорядочения – METIS).

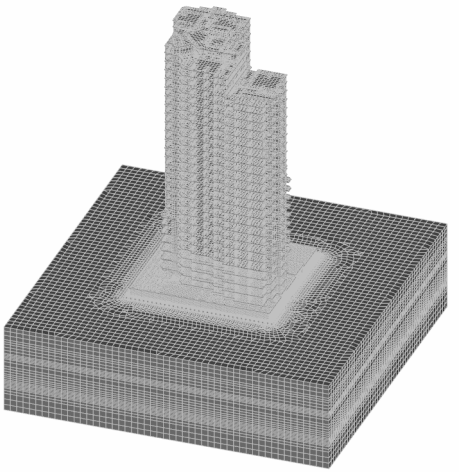

Рис. 4. Расчетная модель задачи о взаимодействии здания и основания (2 989 476 уравнений)

Данная задача решалась PARFES на 16-ядерном компьютере с процессором AMD Opteron 6276 CPU 2.3/3.2 *ГГц*, 64 *GB* DDR3 RAM, OS Windows Server 2008 R2 Enterprise SP1, 64 bit.PARFES работал в режиме СМ. На рис. 5 представлен график ускоряемости при увеличении количества загруженных ядер процессора.

Идеальная ускоряемость представлена на графике прямой линией и проходит через точки {0; 0}, {1, 1}, {2, 2} и т.д. Ее смысл очевиден – если задача решается на двух процессорах, то в идеале производительность решателя должна быть в два раза выше, чем в случае использования одного процессора, если на 4 – то в 4 раза выше, и т.д.

Однако данный процессор поддерживает режим Turbo Core – он уменьшает тактовую частоту с 3.2 *ГГц* до 2.3 *ГГц* при увеличении количества загруженных ядер. Поэтому идеальная ускоряемость на данном процессоре физически не достижима, и этот факт представлен кривой *id_tb* – квадратной параболой, проходящей через точки {0, 0},

{1, 1} (на одном загруженном ядре процессор работает с максимальной тактовой частотой) и {16, 11.5}. Последнее число получено из соображений, что при загрузке 16-ти ядер процессор снизит тактовую частоту до минимальной – 2.3 *ГГц*. На основании этого составлена пропорция: 16·2.3/3.2 = 11.5.

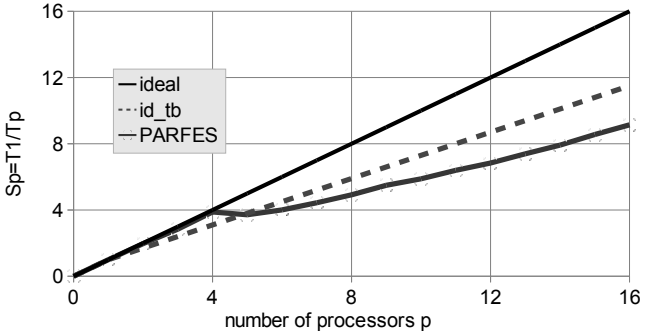


Рис. 5. Ускоряемость PARFES с увеличением количества процессоров

Кривая PARFES соответствует реальным вычислениям. Каждая точка определялась как $S_p = T_1/T_p$, где $p$ – количество загруженных ядер, $S_p$ – ускоряемость (speedup), $T_1$ – время факторизации матрицы на одном ядре, а $T_p$ – на $p$ ядрах.

До 4-х загруженных ядер PARFES демонстрирует идеальную ускоряемость. Далее происходит снижение тактовой частоты процессора, что уменьшает ускоряемость, которая, несмотря на это, остается стабильной вплоть до 16 ядер. Таким образом, одноуровневое распараллеливание может обеспечить стабильную ускоряемость даже при большом количестве потоков, что является заслугой представленного выше алгоритма планирования заданий на потоки.

На 16 загруженных ядрах факторизация матрицы длилась 1194 *с*. Полное время решения задачи –1803 *с*.

Решение этой задачи представлено также в [19]. В данной статье приведены данные для модифицированного микроядра *microkern*_8x4_*AVX*, отличающегося от используемого в упомянутой работе применением инструкций процессора FMA4, позволяющих в одной команде выполнить одновременно умножение и сложение, что сокращает количество циклов процессора с 8 до 5 на процессорах данной архитектуры. Соответственно для данной задачи возросла и производительность решателя с 34 843MFLOPS до 45 515MFLOPS.

**Заключение**. Полученные результаты свидетельствуют о том, что на многоядерных компьютерах с общей оперативной памятью суперузловые

методы решения систем линейных алгебраических уравнений метода конечных элементов являются более эффективными, чем многофронтальные.

Решатель PARFES сочетает высокую производительность при работе в оперативной памяти, устойчивую ускоряемость при увеличении количества загруженных ядер и возможность подключения диска для решения больших задач на компьютерах с ограниченным объемом оперативной памяти.

СПИСОК ЛИТЕРАТУРЫ


1. Matrixcomputations/ *G.H. Golub, C.F. VanLoan*, Thirded. –John Hopkins University Press, 1996, 694 p.
2. Численное решение больших разреженных систем уравнений / *А. Джордж, Дж. Лю*. – М.: Мир, 1984. – 333 с.
3. Applied numerical linear algebra / *J.W. Demmel*. – Philadelphia, SIAM, 1997. – 421 p.
4. Прямые методы решения систем линейных уравнений в современных МКЭ-комплексах / *С.Ю. Фиалко*. – М.: СКАД СОФТ, 2009. – 160 с.
5. *DuffI. S., ReidJ. K.*ThemultifrontalsolutionofindefinitesparsesymmetriclinearsystemsACMTransactionsonMathematicalSoftware. – 1983. – Vol. 9. – P. 302–325.
6. *Dobrian F., Kumfert G., Pothen A.* The Design of Sparse Direct Solvers using Object-Oriented Techniques –In: BruasetA.M., Langtangen H.P., Quak E. (eds.) –Modern Software Tools in Scientific Computing. Springer-Verlag,2000,P 89–131.
7. Intel® Math Kernel Library Reference Manual – BLAS Routines. https://software.intel.com/sites/products/documentation/doclib/iss/2013/mkl/mklman/index.htm.
8. *Goto K., Van DeGeijn R.A.* Anatomy of High-Performance Matrix Multiplication. – ACM Transactions on Mathematical Software. – 2008. –Vol34. No. 3/ – P. 1–25.
9. *Amestoy P.R., Duff I.S., L'Excellent J.Y.* Multifrontal parallel distributed symmetric and unsymmetric solvers. – Comput. Meth. Appl. Mech. Eng. – 2000. –Vol. 184. – P. 501–520.
10. *Фиалко С.Ю.* К исследованию напряженно-деформированного состояния тонкостенных оболочек с массивными ребрами. – Прикл. Механика. – т. 40. –, № 4. – С. 84–92.
11. *Fialko S.* A Sparse Shared-Memory Multifrontal Solver in SCAD Software. IEEE Xplore Digital Library. Computer Science and Information Technology, 2008. IMCSIT 2008. InternationalMulticonferenceon.20 – 22 Oct. – 2008. – P. 277 – 283. Doi:10.1109/IMCSIT.2008.4747252 .
12. *Chen Y., Davis T.A., Hager W.W., Rajamanickam S.* Algorithm 8xx: CHOLMOD,supernodal sparse Cholesky factorization and update/downdate. – Technical report TR-2006-005. – CISE Dept, Univ. of Florida, Gainesville, FL. – 2006. URL:http://www.cise.ufl.edu/tr/DOC/REP-2006-290.pdf (Accessed 20.12.2014).
13. *Demmel J.W., Eisenstat S.C., Gilbert J.R., Li X.S., Liu J.W.H.* A supernodal approach for sparse partial pivoting. – SIAM J. Matrix Anal. Appl. – 1999. – Vol 20. – No. 3. – P. 720 – 755.
14. *Protkin V., Toledo S.* The design and implementation of a new out-of-core sparse Cholesky factorization method. – ACM Transactions on Mathematical Software. – 2004. – Vol. 30. – No.1. – P. 19 – 46.
15. *Schenk O., Gartner K.* Two-level dynamic scheduling in PARDISO: Improved scalability on shared memory multiprocessing systems. – Parallel Computing. – 2002. –Vol. 28. – P. 187–197.
16. *Pardo D., MyungJin Nam, Carlos Torres-Verdín, Michael G. Hoversten, Iñaki Garay*. Simulation of marine controlled source electromagnetic measurements using a parallel Fourier hp-finite element method. – Comput. Geosci. – 2011. – Vol. 15. P. 53–67.

17. *Fialko S.* PARFES: A method for solving finite element linear equations on multi-core computers. – Advances in engineering software. – 2010. – Vol.40. – No.12. – P. 1256 – 1265.
18. *Fialko S.* Parallel Finite Element Solver for Multi-Core Computers. IEEE Xplore Digital Library. Computer Science and Information Technology (FedCSIS), 2012 Federated Conference on. – P. 525 – 532. http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6354298.
19. *Fialko S.* Application of AVX (Advanced Vector Extensions) for Improved Performance of the PARFES – Finite Element Parallel Direct Solver. IEEE Xplore Digital Library.Computer Science and Information Systems (FedCSIS), 2013 Federated Conference on. – P. 447 – 454. http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=6644039&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D6644039
20. *Karypis G., Kumar V.* METIS: Unstructured Graph Partitioning and Sparse Matrix Ordering System. – Technical report. – Department of Computer Science. – University of Minnesota. Minneapolis. – 1995.

REFERENCES

1. Matrixcomputations/ *G.H. Golub, C.F. VanLoan*, Thirded. –John Hopkins University Press, 1996, 694 p.
2. Computer solution of sparse positive definite systems / *A.George, J.W.H. Liu.* – Moscow, Mir, 1984, 333 p. (In Russian)
3. Applied numerical linear algebra / *J. W. Demmel*. – Philadelphia, SIAM, 1997. – 421 p.
4. The direct methods for solution of the linear equation sets in modern FEM software / S. Fialko. – Moscow, SCAD SOFT, 2009, 160 p. (in Russian).
5. *DuffI. S., ReidJ. K.*Themultifrontalsolutionofindefinitesparsesymmetriclinearsystems. – ACMTransactionsonMathematicalSoftware. – 1983. – Vol. 9. – P. 302–325.
6. *Dobrian F., Kumfert G., Pothen A*. The Design of Sparse Direct Solvers using Object-Oriented Techniques –In: Bruaset A.M., Langtangen H.P., Quak E. (eds.) – ModernSoftwareToolsinScientificComputing. Springer-Verlag, 2000,P 89–131.
7. Intel® Math Kernel Library Reference Manual – BLAS Routines. https://software.intel.com/sites/products/documentation/doclib/iss/2013/mkl/mklman/index.htm.
8. *Goto K., Van DeGeijn R*. A. Anatomy of High-Performance MatrixMultiplication. – ACM Transactions on Mathematical Software. – 2008. –Vol34. No. 3/ – P. 1–25.
9. *Amestoy P.R., Duff I.S., L'Excellent J.Y.* Multifrontal parallel distributed symmetric and unsymmetric solvers. – Comput. Meth. Appl. Mech. Eng. – 2000. –Vol. 184. – P. 501–520.
10. *Fialko S.* Stress-Strain Analysis of Thin-Walled Shells with Massive Ribs. – Int. App. Mech. – 2004. Stress-Strain Analysis of Thin-Walled Shells with Massive Ribs. Int. App. Mech. – 2004. –Vol. 4. – No 4. – P. 432–439.
11. *Fialko S.* A Sparse Shared-Memory Multifrontal Solver in SCAD Software. IEEE Xplore Digital Library. Computer Science and Information Technology, 2008. IMCSIT 2008. InternationalMulticonferenceon. 20 – 22 Oct. – 2008. – P. 277 – 283. Doi:10.1109/IMCSIT.2008.4747252 .
12. *Chen Y., Davis T.A., Hager W.W., Rajamanickam S.* Algorithm 8xx: CHOLMOD, supernodal sparse Cholesky factorization and update/downdate. – Technical report TR-2006-005. – CISE Dept, Univ. of Florida, Gainesville, FL. – 2006. URL:http://www.cise.ufl.edu/tr/DOC/REP-2006-290.pdf (Accessed 20.12.2014).
13. *Demmel J.W., Eisenstat S.C., Gilbert J.R., Li X., Liu J.W.H.* A supernodal approach for sparse partial pivoting. – SIAM J. Matrix Anal. Appl. – 1999. – Vol 20. – No. 3. – P. 720 – 755.
14. *Protkin V., Toledo S.* The design and implementation of a new out-of-core sparse Cholesky factorization method. – ACM Transactions on Mathematical Software. – 2004. – Vol. 30. – No.1. – P. 19 – 46.

15. *Schenk O., Gartner K.* Two-level dynamic scheduling in PARDISO: Improved scalability on shared memory multiprocessing systems. – Parallel Computing. – 2002. –Vol. 28. – P. 187–197.
16. *Pardo D., MyungJin Nam, Carlos Torres-Verdín, Michael G. Hoversten, Iñaki Garay*. Simulation of marine controlled source electromagnetic measurements using a parallel Fourier hp-finite element method. – Comput. Geosci. – 2011. – Vol. 15. P. 53–67.
17. *Fialko S.* PARFES: A method for solving finite element linear equations on multi-core computers. – Advances in engineering software. – 2010. – Vol.40. – No.12. – P. 1256 – 1265.
18. *Fialko S.* Parallel Finite Element Solver for Multi-Core Computers. IEEE Xplore Digital Library. Computer Science and Information Technology (FedCSIS), 2012 Federated Conference on. – P. 525 – 532. http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6354298.
19. *Fialko S.* Application of AVX (Advanced Vector Extensions) for Improved Performance of the PARFES – Finite Element Parallel Direct Solver. IEEE Xplore Digital Library. Computer Science and Information Systems (FedCSIS), 2013 Federated Conference on. – P. 447 – 454. http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=6644039&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D6644039
20. *Karypis G., Kumar V.* METIS: Unstructured Graph Partitioning and Sparse Matrix Ordering System. – Technical report. – Department of Computer Science. – University of Minnesota. Minneapolis. – 1995.

*Фіалко С.Ю.*

**ПРО ПАРАЛЕЛЬНІ ВИРІШУВАЧІ В СКІНЕЧНОЭЛЕМЕНТНИХ ПРОГРАМНЫХ КОМПЛЕКСАХ, ОРІЄНТОВАНИХ НА БАГАТОЯДЕРНІ КОМП'ЮТЕРИ ІЗ ЗАГАЛЬНОЮ ПАМ'ЯТТЮ**

Обговорюються прямі методи розв'язування систем лінійних алгебраїчних рівнянь з симетричними розрідженими матрицями, що виникають при застосуванні методу скінчених елементів до завдань механікі конструкцій та механікі деформованого твердого тіла. Головна увага приділяється методам досягнення високої продуктивності на багатоядерних комп'ютерах з загальною пам'яттю (sharedmemorycomputers).

**Ключові слова:** багатофронтальний метод, супервузловий метод, метод скінчених елементів, граф суміжності, висока продуктивність, паралельні обчислювання.

*Fialko S.Yu.*

**ABOUT PARALLEL SOLVERS IN FINITE ELEMENT SOFTWARE, ORIENTED TO SHARED MEMORY MULTIPROCESSOR COMPUTERS**

**Short abstract.**The direct methods for solution of linearequation sets arising when the finite element method is applied to problems of structural and solid mechanicsare considered. Main attention is paid to achievement of high performance on multicore shared memory computers.

**Extended abstract.**The direct methods for solution of linearequation sets arising when the finite element method is applied to problems of structural and solid mechanics are considered. Main attention is paid to achievement of high performance on multicore shared memory computers.We confine ourselves to the consideration of the multifrontal and supernodal methods, present a brief description and emphasize their advantages and disadvantages for the considered class of problems on shared memory multicore computers. We compare the performance of Boeing sparse direct solver implemented in famous ANSYS 15.0 finite element software, supernodal solver PARDISO from Intel Math Kernel Library with block substructure multifrontal solver BSMFM and supernodal parallel finite element solver PARFES, implemented in SCAD software (www.scadsoft.com).

Then, we present the stable speed up and performance on computer with 16-core AMD Opteron 6276 processordemonstrated by PARFES.

**Keywords:** multifrontal method, supernodal methods, finite element method, adjacency graph, high performance, parallel computing.