УДК 519.683.4

ОБ ОПТИМИЗАЦИИ ВЫЧИСЛИТЕЛЬНЫХ ПРИЛОЖЕНИЙ ДЛЯ МНОГОПРОЦЕССОРНЫХ СИСТЕМ С ОБЩЕЙ НЕОДНОРОДНОЙ ПАМЯТЬЮ

К. Ю. Богачев[1], **А. Р. Миргасимов**[1]

Для многопроцессорных систем с общей памятью с разным временем доступа процессоров к различным участкам памяти (так называемые NUMA-системы) рассмотрен подход к оптимизации многопоточных приложений, позволяющий максимально использовать вычислительные ресурсы системы при минимальных изменениях в коде приложения. Подход может быть использован в гибридных MPI-многопоточных программах на современных кластерных системах. Приведены результаты численных экспериментов на большом количестве реальных задач. Работа выполнена при поддержке РФФИ (проект № 09–01–00625-а)

Ключевые слова: высокопроизводительные вычисления, гибридные MPI-многопоточные программы, NUMA-системы.

1. Введение. После разработки в 2003 г. процессоров AMD Opteron стали широко доступны системы с разным временем доступа процессоров к различным участкам памяти (так называемые NUMA-системы, Non-Uniform Memory Access). Это привело к значительным изменениям как в операционных системах (планирование заданий, управление памятью) [1], так и в приложениях [2]. С появлением архитектуры процессоров Intel Woodcrest в 2006 г. интерес к NUMA-системам понизился. После создания в 2008 г. процессоров Intel Nehalem стало ясно, что все новые многопроцессорные конфигурации будут NUMA-системами.

Для реализации возможностей NUMA-систем в основном рассматривались две возможности: использование MPI (Message Passing Interface), при котором полностью решается проблема разного времени доступа к памяти ценой пересылки данных из одного процесса, работающего на узле с общей памятью, другому процессу, работающему там же, и использование потоков с разделением данных между ними, при котором требуется значительно менять код традиционного многопоточного приложения.

Ниже рассмотрена третья возможность, основанная на использовании особенностей управления памятью в операционных системах Linux и Windows 7 и не требующая значительного изменения программного кода многопоточного приложения. Проводится сравнение времени работы вычислительного приложения на NUMA-системах на примере решения задачи фильтрации вязкой сжимаемой многофазной смеси в анизотропной пористой среде в реальных нефтегазовых месторождениях.

2. Подход к оптимизации вычислительных приложений для NUMA-систем. Рассмотрим параллельную многопоточную реализацию алгоритма перемножения матрицы на вектор [3]. Скорость работы такой программы во многом зависит от скорости доступа к оперативной памяти. Это делает ее хорошим примером для демонстрации ключевых идей, лежащих в основе метода оптимизации программного кода для NUMA-систем.

В традиционной программной реализации память под матрицы и вектор выделяется в главном потоке. На NUMA-системе это приводит к тому, что физически матрица и вектор располагаются в локальной памяти одного NUMA-узла, что увеличивает как минимум в два раза время доступа к ним потоков, работающих на другом узле.

Покажем, как можно оптимизировать программный код для NUMA-систем, не изменив сам численный алгоритм, и получить ускорение 1.5–2 раза.

Допустим, что порядок матрицы равен $K = MT$, где T — число потоков; пусть матрица хранится в памяти по строкам. Поток с номером i вычисляет результат произведения строк матрицы с номерами $M(i-1)+1, \ldots, Mi$ на вектор. Будем говорить, что эти строки матрицы принадлежат i-му потоку. Таким образом, матрица делится на T непрерывных частей, каждая из которых обрабатывается собственным потоком.

[1] Московский государственный университет им. М. В. Ломоносова, механико-математический факультет, Ленинские горы, 119899, Москва; К. Ю. Богачев, доцент, e-mail: bogachev@mech.math.msu.su; А. Р. Миргасимов, аспирант, e-mail: mirgasimov.almaz@gmail.com

© Научно-исследовательский вычислительный центр МГУ им. М. В. Ломоносова

Ключевая идея оптимизации заключается в реализации двух условий, которые позволят минимизировать число обращений к нелокальным участкам памяти:

— каждый из вычислительных потоков работает на фиксированном узле NUMA-системы,

— те части матрицы и вектора, с которыми вычислительный поток оперирует бóльшую часть времени, физически находятся в локальной памяти этого узла.

Для выполнения первого условия в каждом из вычислительных потоков сразу после его создания сделаем системный вызов, который привяжет этот i-й поток к i-му логическому процессору.

Для удовлетворения второго условия с минимальным переписыванием программы воспользуемся особенностью механизма выделения памяти в операционных системах, поддерживающих NUMA. В системах Linux и Windows 7 при выделении памяти посредством функции malloc отображение затребованной виртуальной памяти на физическую строится не сразу. Когда поток обращается к блоку виртуальной памяти, для которого еще не построено отображение, происходит страничный промах (page fault), который инициирует создание отображения запрошенного блока виртуальной памяти на локальную память того узла NUMA-системы, с которого был произведен запрос.

При традиционном подходе выделения памяти в многопоточных приложениях главный поток запрашивает память у операционной системы и инициализирует ее, скажем, нулями. На NUMA-системе это приводит к тому, что вся запрошенная память выделится в локальной памяти узла, на котором работал главный поток. Остальные потоки в это время простаивают, дожидаясь завершения этой операции.

Модифицируем указанный подход, чтобы его можно было эффективно использовать на NUMA-системе. Будем выполнять операцию выделения памяти в два этапа. На первом этапе главный поток запрашивает у операционной системы память требуемого объема. Затем следует точка синхронизации, после которой указатель на затребованную память можно использовать во всех потоках. На втором этапе выделенная память параллельно инициализируется всеми потоками, причем каждый из потоков инициализирует только свою часть запрошенной памяти.

Предложенный метод инициализации памяти в два этапа эффективен по двум причинам. Во-первых, та часть памяти, к которой поток будет обращаться чаще всего, выделяется в локальной памяти узла NUMA-системы, на котором тот работает. Во-вторых, ускоряется операция ее инициализации за счет параллелизма в работе памяти. Минусом этого метода является наличие дополнительной точки синхронизации потоков.

Эти модификации затрагивают ту часть программного кода, которую можно назвать подготовительной стадией алгоритма — создание вычислительных потоков, заполнение матрицы и вектора. Тем самым алгоритмически существенная часть программного кода, ответственная за непосредственное перемножение матрицы на вектор, остается неизменной.

На основе этих идей представленный метод оптимизации кода для NUMA-систем можно применить к любой многопоточной программе, в которой вычислительные потоки бóльшую часть времени проводят, обращаясь к своей части всей выделенной процессу памяти. Получаемый выигрыш зависит от доли операций с памятью в общем времени работы программы.

3. Постановка задачи фильтрации. Рассмотрим следующие стандартные уравнения многофазной изотермической модели черной нефти в трехмерной области, использованные также в работах [5, 6]:

$$
\begin{aligned}
&\frac{\partial}{\partial t}\,(\phi N_c) = \operatorname{div} \sum_{l=\mathrm{o,w,g}} x_{c\,l}\xi_l\left(\boldsymbol{k}\,\frac{k_{r\,l}}{\mu_l}\,(\nabla p_l - \gamma_l \nabla \boldsymbol{D})\right) + q_c, \quad c = 1, \ldots, n_c,\\
&p_{\mathrm{o}} - p_{\mathrm{g}} = P_{c\,\mathrm{og}}, \quad p_{\mathrm{o}} - p_{\mathrm{w}} = P_{c\,\mathrm{ow}},\\
&S_{\mathrm{w}} + S_{\mathrm{o}} + S_{\mathrm{g}} = 1,\\
&N_{\mathrm{w}} = \frac{S_{\mathrm{w}}}{B_{\mathrm{w}}}\,, \quad N_{\mathrm{o}} = \frac{S_{\mathrm{o}}}{B_{\mathrm{o}}}\,, \quad N_{\mathrm{g}} = \frac{S_{\mathrm{g}}}{B_{\mathrm{g}}} + R_{\mathrm{g,o}}\,\frac{S_{\mathrm{o}}}{B_{\mathrm{o}}}\,.
\end{aligned}
\tag{1}
$$

Здесь n_c — количество компонентов в смеси; $R_{\mathrm{g,o}} = R_{\mathrm{g,o}}(p_{\mathrm{o}})$ — растворимость газа в нефтяной фазе; $B_l = B_l(p_l)$ — коэффициент объемного расширения фазы l, $l = \mathrm{o}\ (\mathrm{oil}), \mathrm{g}\ (\mathrm{gas}), \mathrm{w}\ (\mathrm{water})$; пористость среды $\phi = \phi(p_{\mathrm{w}}, p_{\mathrm{o}}, p_{\mathrm{g}}, x, y, z)$; $x_{c\,l} = x_{c\,l}(p_{\mathrm{w}}, p_{\mathrm{o}}, p_{\mathrm{g}}, \boldsymbol{N})$ — молярная доля компонента c, $c = 1, \ldots, n_c$ в фазе l; $\boldsymbol{N} = (N_1, \ldots, N_{n_c})$ — вектор молярных плотностей; $\xi_l = \xi_l(p_l, \boldsymbol{N})$ — молярная плотность фазы l; $\boldsymbol{k} = \boldsymbol{k}(p_{\mathrm{w}}, p_{\mathrm{o}}, p_{\mathrm{g}}, x, y, z)$ — тензор абсолютной проницаемости; $k_{r\,l} = k_{r\,l}(S_{\mathrm{w}}, S_{\mathrm{g}})$ — относительная проницаемость фазы l; $\mu_l = \mu_l(p_l)$ — вязкость фазы l; $\boldsymbol{\gamma}_l = \rho_l g$ — вертикальный градиент гидростатического давления в фазе l; $\boldsymbol{g}$ — гравитационная постоянная; $\boldsymbol{D} = \boldsymbol{D}(x, y, z)$ — вектор глубины (сверху вниз); $\rho_l = \rho_l(p_l)$ — массовая плотность фазы l; $P_{c\,\mathrm{og}} = P_{c\,\mathrm{og}}(S_{\mathrm{g}})$ — капиллярное давление в системе нефть–газ; $P_{c\,\mathrm{ow}} = P_{c\,\mathrm{ow}}(S_{\mathrm{w}})$ — капиллярное давление в системе вода–нефть; $q_c = q_c(p_{\mathrm{w}}, p_{\mathrm{o}}, p_{\mathrm{g}}, \boldsymbol{N}, t, x, y, z)$ — источник

компонента c (скважина).

Неизвестными в этой системе являются

1) $N_c = N_c(t, x, y, z)$ — молярная плотность компонента c (для модели черной нефти компонентами служат вода, нефть и газ);

2) $S_l = S_l(t, x, y, z)$ — насыщенность фазы l;

3) $p_l = p_l(t, x, y, z)$ — давление в фазе l.

В самом распространенном случае количество компонентов n_c равно трем. Для этой системы уравнений задаются начальные условия, а также на внешней границе резервуара ставятся условия непротекания.

4. Практическое применение подхода к оптимизации вычислительных приложений для NUMA-систем. Продемонстрируем практическое применение оптимизации кода для NUMA-систем на примере решения задачи (1) фильтрации вязкой сжимаемой многофазной смеси в анизотропной пористой среде, реализованного в гидродинамическом пакете моделирования tNavigator.

Для аппроксимации уравнений по времени используется полностью неявная схема. Аппроксимацией по пространственным переменным методом конечных объемов на неравномерной сетке первоначальная задача сначала сводится к системе нелинейных алгебраических уравнений вида

$$\boldsymbol{F}(\boldsymbol{p}, \boldsymbol{N}_1, \ldots \boldsymbol{N}_{n_c}) = 0,$$

где $\boldsymbol{p} = \left(p^i\right)$, $\boldsymbol{N}_c = \left(N_c^i\right)$ — векторы значений давления и молярных плотностей в блоках сетки. Для решения системы нелинейных уравнений $\boldsymbol{F}(\boldsymbol{U}) = 0$, $\boldsymbol{U} \equiv (\boldsymbol{p}, \boldsymbol{N})$ используется метод Ньютона

$$\boldsymbol{U}^{m+1} = \boldsymbol{U}^m - \left(\frac{\partial \boldsymbol{F}\left(\boldsymbol{U}^m\right)}{\partial \boldsymbol{U}}\right)^{-1} \boldsymbol{F}\left(\boldsymbol{U}^m\right).$$

Здесь $\dfrac{\partial \boldsymbol{F}\left(\boldsymbol{U}^m\right)}{\partial \boldsymbol{U}}$ — матрица $R^{n_c * K} \times R^{n_c * K}$, где K — число блоков сетки. На каждом шаге метода Ньютона решается линейная система с несимметричной матрицей $\dfrac{\partial \boldsymbol{F}\left(\boldsymbol{U}^m\right)}{\partial \boldsymbol{U}}$, т.е. задача сводится к решению системы линейных алгебраических уравнений

$$A\boldsymbol{x} = \boldsymbol{r}, \tag{2}$$

в которой матрица A — якобиан из метода Ньютона.

Построение системы линейных уравнений и ее решение занимают бóльшую часть времени работы пакета tNavigator. Именно эти два этапа были подвергнуты оптимизации для NUMA-систем.

Перед тем как приступать к оптимизации параллельного многопоточного кода для NUMA-систем, необходимо определить, какие объекты, имеющие размер порядка K (назовем их большими), используются в программе и можно ли каждый из этих объектов разбить на части, которые в основном обрабатываются отдельными потоками.

Первое множество больших объектов (обозначим его через T_1) включает в себя различные свойства фаз, компонентов и породы в каждом блоке сетки. Во-первых, это главные переменные решаемой задачи — молярные плотности компонентов и давление в фазе “нефть”. Во-вторых, все, что так или иначе входит в коэффициенты уравнения (1) — относительные проницаемости, вязкости, насыщенности фаз, пористость, проводимость, коэффициенты масштабирования фазовых проницаемостей и т.д. Этот список на практике получается гораздо длиннее приведенного и зависит от множества физический явлений, учитываемых в математической модели. В-третьих, это производные перечисленных свойств по главным переменным.

Указанные данные, в основном, используются для вычисления коэффициентов системы линейных уравнений (2). Это процедуру можно представить следующим циклом: для каждого блока $i = 1, \ldots, K$, используя данные i-го блока и геометрически соседних с ним, посчитать коэффициенты линейной системы, описывающих физику процесса в i-м блоке.

В многопоточном приложении множество всех блоков разбивается между M потоками — i-й поток обрабатывает блоки с номерами $K_{i-1}, \ldots, K_i - 1$, где $1 = K_0 < K_1 < \ldots < K_M = K$. Как правило, разбиение получается таким, что геометрически соседние блоки обрабатываются одним потоком. Таким образом, i-й поток при вычислении своей части коэффициентов линейной системы, в основном, будет обращаться к элементам в позициях $K_{i-1}, \ldots, K_i - 1$ массивов размера K из множества T_1.

Система линейных уравнений (2) решается методом BCGS (Biconjucate Gradient Stabilized) [7] с использованием предобусловливателя класса ILU (Incomplete LU-decomposition) [4]. На этом этапе численные операции выполняются на другом множестве больших объектов, обозначим его через T_2. В него входят

Таблица 1

Тестовая система 1: процессор Intel Xeon E5520 2.27 GHz, число ядер = 4, число процессоров = 2, оперативная память DDR-III 1067 MHz 12 Gb		
а	Умножение матрицы $K \times K$ на вектор длины K	Размерность задачи $K = 20000$ Объем использованной памяти — 3052 Mb
б	Решение системы линейных уравнений методом BCGS	Размер матрицы $K = 1094421$ Число ненулевых элементов в матрице = 7484949 Объем использованной памяти — 702 Mb
в	Расчет реального месторождения	Размерность задачи (число блоков в модели) $K = 1094421$ Объем использованной памяти — 1921 Mb Время расчета с использованием одного потока — 1 час 2 минуты

Таблица 2

Тестовая система 2: процессор Intel Xeon X7560 2.27GHz, число ядер = 8, число процессоров = 4, оперативная память DDR-III 1067MHz 128Gb	
Расчет реального месторождения	Размерность задачи (число блоков в модели) $K = 2418989$ Объем использованной памяти — 6133 Mb Время расчета с использованием одного потока — 26 часов 47 минут

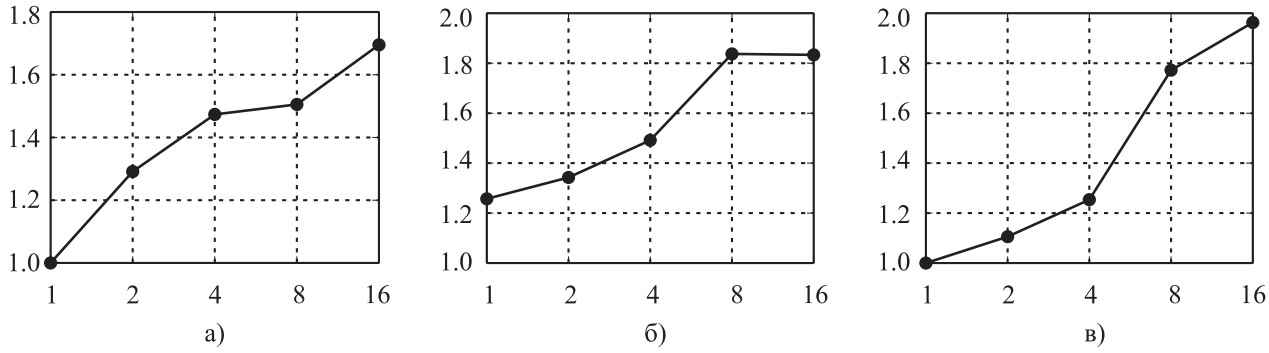

Рис. 1. Графики ускорения работы программ после применения оптимизаций для NUMA-систем в зависимости от числа потоков: а) умножение матрицы на вектор, б) решение системы линейных уравнений, в) расчет реального месторождения

матрица A, хранящаяся в MSR-формате, матрица предобусловливателя для каждого потока, располагающаяся в выделенной ему памяти, и некоторое количество векторов длины K, используемых в методе BCGS. При решении системы i-й поток, в основном, оперирует со строками $K_{i-1}, \ldots, K_i - 1$ матрицы A, с элементами в тех же позициях векторов длины K и со своей матрицей предобусловливателя.

Отмеченное свойство вычислительных потоков обращаться, в основном, только к своим частям массивов дает возможность разбивать большие объекты на части. Таким образом, теперь оптимизация программного кода для NUMA-систем становится такой же, как и в случае перемножения матрицы на вектор:

— использование системного вызова, привязывающего i-й поток к i-му логическому процессору;

— использование двухэтапной инициализации памяти под массивы из множеств T_1 и T_2, в которой каждый из потоков инициализирует только свою часть этих массивов.

5. Численные эксперименты. Описанный подход был реализован в составе гидродинамического пакета tNavigator и протестирован на параллельной ЭВМ с двумя процессорами Intel Nehalem и на параллельной ЭВМ с четырьмя процессорами Intel Nehalem EX. Для тестирования была выбрана гидродинамическая модель реального месторождения, содержащая несколько миллионов блоков. Тестовая система 1 представлена в табл. 1, тестовая система 2 — в табл. 2.

Графики ускорения работы программ после применения оптимизаций для NUMA-систем показаны на рис. 1а–1в.

На рис. 2а показан график ускорения работы программ после применения оптимизаций для NUMA-систем, на рис. 2б — график масштабируемости параллельного расчета.

Оптимизация программы для NUMA-системы позволила достичь ускорения в 21 раз на системе с 32 физическими ядрами, в то время как ускорение неоптимизированной версии равно 11.

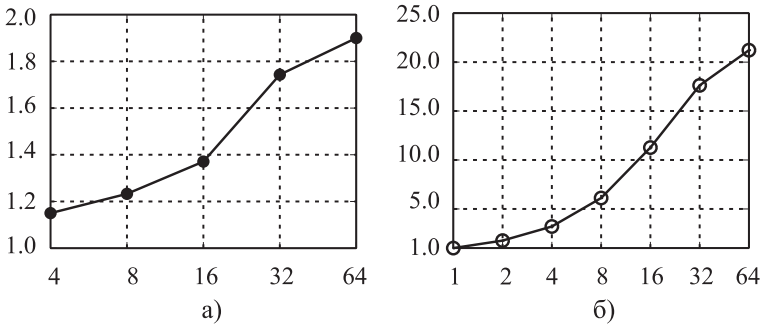

Рис. 2. Расчет реального месторождения, на горизонтальной оси отложено число потоков

6. Заключение. Предложен подход, позволяющий значительно сократить время на разработку эффективных вычислительных приложений для NUMA-систем. Результаты численных экспериментов показывают, что при использовании всех физических ядер параллельной ЭВМ оптимизированное для NUMA-систем приложение работает в $1.5 - 2$ раза быстрее, чем его исходная версия.

СПИСОК ЛИТЕРАТУРЫ

1. *Lameter C.* Local and remote memory: memory in a Linux/NUMA system (ftp://ftp.kernel.org/pub/linux/kernel/people/christoph/pmig/numamemory.pdf, 2006).
2. *David E.Ott.* Optimizing software applications for NUMA. DDJ Magazine, 2009.
3. *Богачев К.Ю.* Основы параллельного программирования. М.: Бином, 2003.
4. *Богачев К.Ю., Жабицкий Я.В.* Блочные предобусловливатели класса ILU для задач фильтрации многокомпонентной смеси в пористой среде // Вестн. Моск. ун-та. Матем. Механ. 2009. № 5. 19–25.
5. *Богачев К.Ю., Горелов И.Г.* Применение параллельного предобусловливателя CPR к задаче фильтрации вязкой сжимаемой жидкости в пористой среде // Вычислительные методы и программирование. 2008. **9**, № 2. 35–41.
6. *Богачев К.Ю., Мельниченко Н.С.* О пространственной аппроксимации методом подсеток для задачи фильтрации вязкой сжимаемой жидкости в пористой среде // Вычислительные методы и программирование. 2008. **9**, № 2. 42–50.
7. *Saad Y.* Iterative methods for sparse linear systems. Philadelphia: SIAM, 2003.

Поступила в редакцию
21.05.2010