УДК 681.3

ПРИХОЖИЙ А.А., КАРАСИК О.Н., БНТУ

КООПЕРАТИВНАЯ МОДЕЛЬ ОПТИМИЗАЦИИ ВЫПОЛНЕНИЯ ПОТОКОВ НА МНОГОЯДЕРНОЙ СИСТЕМЕ

Исследована проблема повышения эффективности программирования и выполнения многопоточных приложений на многоядерных системах. Предложена кооперативная модель выполнения потоков, оптимизирующая порядок выполнения вычислительных операций и операций обмена данными, уменьшающая время исполнения многопоточного приложения путем сокращения критического пути на графе параллельного алгоритма, повышающая производительность приложения при росте числа потоков, исключающая конкуренцию между потоками, характерную для вытесняющей многозадачности.

The problem of the increase of efficiency of multi-thread applications on multi-core systems is investigated. The optimization cooperative model of threads execution has been proposed. It optimizes the execution order of the computational operations and the operations of data exchange, decreases the overall time of the multithread application execution by means of the reduction of the critical path in the concurrent algorithm graph, increases the application throughput at the growth of the number of threads, and excludes the competition among threads that is specific for preemptive multitasking.

Введение

Не смотря на то, что многоядерные системы распространены сегодня повсеместно и доступны через компьютеры, планшеты, телефоны, телевизоры и другие устройства, программирование таких систем остается чрезвычайно сложной задачей как с позиции распараллеливания алгоритмов, так и с позиции оптимального управления вычислительными ресурсами. Программирование и использование многоядерных систем поддерживается операционными системами с вытесняющей и кооперативной многозадачностью. При вытесняющей многозадачности планирование и управление выполнением потоков осуществляется операционной системой и заложенными в ней алгоритмами планирования. При кооперативной многозадачности эти вопросы полностью решаются программистом, получающим возможность оптимизации и использования для каждой задачи и каждой многоядерной архитектуры наиболее эффективных алгоритмов управления потоками и ресурсами. В работе [1] на примере задачи решения систем линейных алгебраических уравнений (СЛАУ) исследованы методы реализации многопоточных приложений на многоядерных системах, управляемых операционной системой с вытесняющей многозадачностью. В данной статье исследуются методы реализации многопоточных приложений с использованием кооперативной модели оптимизации выполнения потоков, разрабатываются пути повышения производительности параллельных программ и загрузки ресурсов многоядерной системы.

Блочно-параллельные методы Гаусса решения СЛАУ

Метод Гаусса решает СЛАУ вида $\boldsymbol{A}\times\boldsymbol{x} = \boldsymbol{b}$, где $\boldsymbol{A}$ – числовая матрица размером $M\times M$; $\boldsymbol{x}$ – вектор переменных размером M; $\boldsymbol{b}$ – числовой вектор размером M. Прямой ход приводит матрицу $\boldsymbol{A}$ к треугольному виду. Обратный ход находит значения переменных $\boldsymbol{x}$. Блочно-параллельные методы Гаусса [2] ориентированы на параллельную реализацию на многопроцессорных системах. Они разбивают матрицу $\boldsymbol{A}$ на блоки $\boldsymbol{A}^t$, $t = 0, \ldots, T-1$ размерностью $N\times M$ каждый с числом блоков $N = M/T$ и разбивают векторы $\boldsymbol{x}$ и $\boldsymbol{b}$ на блоки $\boldsymbol{x}^t$ и $\boldsymbol{b}^t$ соответственно размерностью N. Методы различаются порядком выбора активной строки матрицы $\boldsymbol{A}$. Метод μ1 выбирает при прямом ходе активные строки $\boldsymbol{A}^t_r$ при $t = 0, \ldots T-1$, $r = 0, \ldots, N-1$ в лексико-графическом порядке и пересчитывает оставшиеся строки $\boldsymbol{A}^t_i$, зануляя элементы

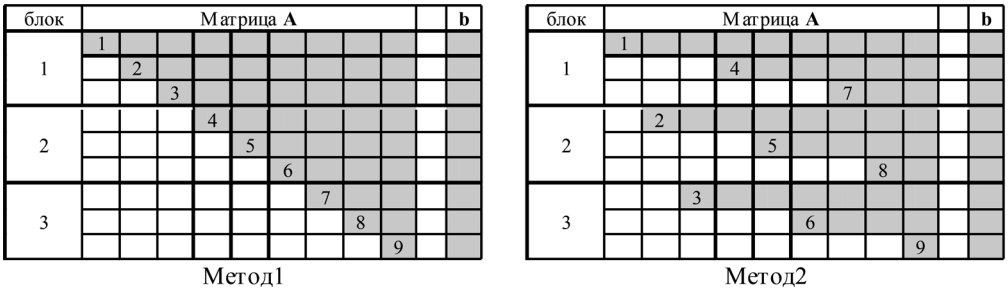


Рис. 1. Блочно-параллельные метод μ1 и метод μ2 решения СЛАУ методом Гаусса

$$D^1=\begin{vmatrix} 0 & 920 & 920 & 920 \\ 16 & 0 & 664 & 664 \\ 16 & 16 & 0 & 408 \\ 16 & 16 & 16 & 0 \end{vmatrix} \qquad D^2=\begin{vmatrix} 0 & 576 & 576 & 576 \\ 560 & 0 & 560 & 560 \\ 544 & 544 & 0 & 544 \\ 528 & 528 & 528 & 0 \end{vmatrix}$$

Рис. 2. Объем данных в числе слов, пересылаемых между потоками метода μ1 (слева) и метода μ2 (справа) для 64 переменных и 4 потоков

$\boldsymbol{A}^t_{i,r}$ при $i = r + 1, \ldots, N - 1$ блока t, а также пересчитывает строки и зануляет соответствующие элементы в блоках $\boldsymbol{A}^s$, $s = t + 1, \ldots, T - 1$ (рис. 1, слева). Метод μ2 выбирает активные строки $\boldsymbol{A}^t_r$ путем перескакивания по блокам как показано на рис. 1 справа. Матричные и векторные блоки распределяются по потокам $t = 0, \ldots, T - 1$, выполняемым на многоядерной системе параллельно и асинхронно. Потоки взаимодействуют друг с другом путем посылки и приема сообщений. Поведение потоков синхронизируется через номер активной строки и номер активного блока.

Объем вычислений в числе арифметических операций ×, /, –, выполняемых потоком t метода μ1 при прямом и обратном ходе, оценивается выражением

$$C_t^1 = \sum_{s=0}^{t-1}\sum_{k=0}^{N-1} 2N(M+1-sN-k) + \sum_{k=0}^{N-1} M+2-kT-t+2(N-k-1)(M+1-kT-t). \quad (1)$$

Объем вычислений, выполняемых потоком t метода μ2, оценивается выражением

$$C_t^2 = \sum_{k=0}^{N-1}\sum_{s=t+1}^{T-1} 2(N-k-1)(M+1-kT-s) + \sum_{k=0}^{N-1}\sum_{s=0}^{t-1} 2(N-k)(M+1-kT-s) + \sum_{k=0}^{N-1} M+2-kT-t+2(N-k-1)(M+1-kT-t). \quad (2)$$

Для СЛАУ с числом переменных $M = 64$, числом потоков $T = 4$ и числом строк $N = 16$ матрицы $\boldsymbol{A}$ в одном потоке вычислительная нагрузка возрастает с увеличением индекса потока и равна 15656, 41512, 59176, 68648 операций соответственно для метода μ1. Для метода μ2 вычислительная нагрузка на потоки распределена достаточно равномерно и равна 44672, 45744, 46784, 47792 операций соответственно. Объем данных в числе слов, пересылаемых между потоками, определяется матрицей D^1 для метода μ1 и матрицей D^2 для метода μ2 (рис. 2).

Кооперативное выполнение потоков в операционной системе

Windows Server 2012 R2 является операционной системой с вытесняющей многозадачностью. Она включает системный планировщик, устанавливающий порядок выполнения потоков и назначающий их на процессоры с учетом ряда параметров [3]. Потоки взаимодействуют посредством примитивов синхронизации, прямая передача управления одного потока другому не допускается. Начиная с Windows 7 и Windows Server 2008 R2, программисту доступен также механизм User-Mode Scheduling (UMS) планирования порядка выполнения потоков в обход системного планировщика [4]. Суть механизма иллюстрируется рис. 3. UMS включает процедуру, реализующую алгоритмы планирования и взаимодействия с операционной системой, поток планирования *UMSSchedulerThread* (UMSST) и поток пользователя *UMSWorkerThread* (UMSWT). Поток UMSST вызывает процедуру *ExecuteUmsThread* для передачи управления потоку UMSWT. Поток UMSWT возвращает управление потоку

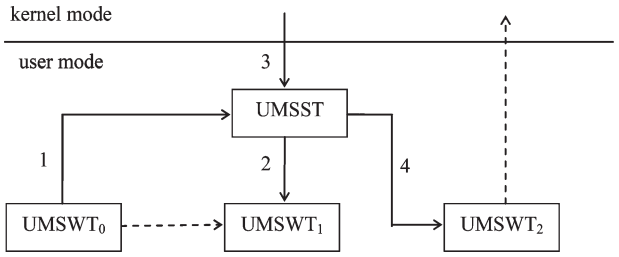


Рис. 3. Схема взаимодействия операционной системы, UMSST и UMSWT

UMSST либо при его блокировке ядром операционной системы, либо путем вызова процедуры *UmsThreadYield*. Связь между потоком UMSST и ядром операционной системы осуществляется посредством списка *UmsCompletionList*, в который попадают потоки пользователя, закончившие обработку в ядре операционной системы.

На рис. 3 дуга *1* описывает передачу управления от $UMSWT_0$ к UMSST посредством функции *UmsThreadYield*; дуга *2* – передачу управления от UMSST к $UMSWT_1$ посредством функции

ExecuteUmsThread; дуга *3* – получение из списка *UmsCompletionList* потока UMSWT, закончившего обработку в ядре операционной системы; дуга *4* – передачу управления от UMSST к $UMSWT_2$ посредством функции *ExecuteUmsThread*. Кооперативная модель выполнения потоков реализуется благодаря возможности контролируемой передачи управления от потока UMSWT потоку UMSST и обратно. Так поток $UMSWT_0$ возвращает управление потоку UMSST с параметрами, указывающими на дальнейшую передачу управления потоку $UMSWT_{1,}$.

Балансирование вычислительной нагрузки при назначении потоков на процессоры

Принцип балансирования нагрузки использован при построении блочно-параллельного алгоритма путем сбалансированного распределения данных и операторов по потокам. В результате получен метод μ*2* решения СЛАУ, который дает более сбалансированное распределение нагрузки по потокам по сравнению с методом μ1. Балансирование вычислительной нагрузки является также важнейшим принципом при назначении потоков на ядра разнородной многоядерной системы. В методе μ1 вычислительная нагрузка возрастает с увеличением индекса потока. Вследствие этого ядро, обслуживающее поток с меньшим индексом, будет простаивать в течение времени, которое необходимо другому ядру для завершения потока с большим индексом. С увеличением количества потоков появляется возможность балансирования вычислительной нагрузки между логическими процессорами и между ядрами.

Рассмотрим многоядерную систему, включающую два физических процессора с четырьмя ядрами каждый (всего 8 ядер с номерами 0, …, 7) и двумя логическими процессорами на каждом ядре (всего 16 логических процессоров с номерами 0, …, 15). Жесткая привязка *NUMA* для 16 потоков с номерами 0, …, 15 означает следующее их назначение на логические процессоры и ядра: 0,15→0,1(0), 1,14→2,3 (1), 2,13→4,5 (2), 3,12→6,7 (3), 4,11→8,9 (4), 5,10→10,11 (6), 6,9→12,13 (6), 7,8→14,15 (7). Запись 0,15→0,1 (0) означает, что потоки 0 и 15 назначаются на логические процессоры 0 и 1 соответственно, реализуемые одним ядром с номером 0. Такое назначение указывается в маске привязанности и идеальном процессоре каждого потока. Поскольку общее количество потоков T может превышать число 16 логических процессоров, потоки разбиваются на группы с номерами 0, …, 15. Группа с номером g включает потоки с номерами g, $g + 16$, $g + 32$, …, $g + T - 16$. Жесткая привязка *NUMA* групп потоков к логическим процессорам одинаково применима и к методу μ1 и к методу μ2. При использовании жесткой привязки *NUMA* вычислительная нагрузка C_p на логический процессор p рассчитывается по формуле

$$C_p = \sum_{t \in \Lambda_p} c_t , \qquad (3)$$

где $\Lambda_p = \{p/2,\ p/2 + 16,\ \ldots,\ p/2 + T - 16\}$ для процессоров с номерами $p = 0, 2, 4, 6, 8, 10, 12, 14$, и $\Lambda_p = \{15\text{-}p/2,\ 15 - p/2 + 16,\ \ldots,\ 15 - p/2 + T - 16\}$ для процессоров с номерами $p = 1, 3, 5, 7, 9, 11, 13, 15$, где / – операция деления нацe-

Нагрузка на логические процессоры в процентах в зависимости от количества потоков для СЛАУ размером 15872, решаемой методом μ1

Количество потоков			16	32	64	128	256	512	992	1984	3968	7936
Количество строк матрицы на один поток			992	496	248	124	62	31	16	8	4	2
Группа потоков	Логический процессор	Ядро	Вычислительная нагрузка на логический процессор (%)									
0	0	0	0,58	3,73	5,07	5,68	5,97	6,11	6,18	6,22	6,23	6,24
15	1		9,36	8,13	7,27	6,78	6,52	6,39	6,32	6,29	6,27	6,26
1	2	1	1,67	4,15	5,25	5,76	6,01	6,13	6,19	6,22	6,24	6,24
14	3		9,29	7,96	7,15	6,71	6,49	6,37	6,31	6,28	6,27	6,26
2	4	2	2,70	4,56	5,42	5,84	6,05	6,15	6,20	6,22	6,24	6,24
13	5		9,14	7,78	7,04	6,65	6,45	6,35	6,30	6,28	6,26	6,26
3	6	3	3,65	4,94	5,59	5,92	6,09	6,17	6,21	6,23	6,24	6,24
12	7		8,92	7,58	6,91	6,58	6,42	6,33	6,29	6,27	6,26	6,26
4	8	4	4,53	5,31	5,76	6,00	6,12	6,19	6,22	6,23	6,24	6,25
11	9		8,63	7,36	6,78	6,51	6,38	6,31	6,28	6,27	6,26	6,25
5	10	5	5,34	5,66	5,92	6,08	6,16	6,21	6,23	6,24	6,24	6,25
10	11		8,26	7,12	6,65	6,44	6,34	6,30	6,27	6,26	6,26	6,25
6	12	6	6,07	5,99	6,07	6,15	6,20	6,22	6,24	6,24	6,25	6,25
9	13		7,82	6,86	6,51	6,37	6,31	6,28	6,26	6,26	6,25	6,25
7	14	7	6,73	6,30	6,23	6,23	6,24	6,24	6,25	6,25	6,25	6,25
8	15		7,31	6,59	6,37	6,30	6,27	6,26	6,26	6,25	6,25	6,25

ло. Вычислительная нагрузка c_t потока t оценивается по формуле (1) для метода μ1 и формуле (2) для метода μ2.

Таблица показывает зависимость доли нагрузки на каждый логический процессор и каждое ядро в зависимости от числа потоков для СЛАУ размером 15872 при условии, что потоки назначаются на процессоры жесткой привязкой *NUMA*, а система решается методом μ1. С увеличением числа потоков нагрузка распределяется более равномерно. При решении СЛАУ методом μ2 и использовании той же привязки потоков к процессорам распределение нагрузки является еще более равномерным и составляет 6.25% на каждый логический процессор независимо от числа потоков.

Ускорение параллельного алгоритма кооперативной моделью

Эффективность кооперативной модели сильно зависит от распределения вычислений по потокам. Внутри каждого потока, операторы посылки и приема сообщений разбивают описываемые потоком вычисления на части. Критический путь в многопоточном параллельном алгоритме формируется из вычислительных частей и операций посылки-приема сообщений между потоками. Если потоки назначены на один процессор, операции посылки-приема выполняются быстрее, если на разные процессоры – медленнее. Различные вычислительные части одного потока могут иметь различную вычислительную сложность. Очевидно, что нахождение критического пути на графе параллельного многопоточного алгоритма является нетривиальной задачей. Более того, число вариантов разбиений вычислений по потокам и упорядочение вычислений внутри потоков с установлением точек посылки-приема сообщений огромно, при этом каждый вариант характеризуется своим критическим путем. Так, только число вариантов назначений потоков на процессоры равно P^T, где P – число процессоров, T – число потоков. Отметим, что в блочно-параллельных методах Гаусса блоки строятся из строк матрицы СЛАУ, а элементарную вычислительную часть внутри блока будем ассоциировать с одной строкой.

Преимущество кооперативной модели выполнения потоков заключается в том, что увеличение количества потоков увеличивает возможность изменения порядка обработки строк

№	ЛП 0			ЛП 1		
1	T0	K0	C0			
2	T0	П0	C1	T1	П0	C4
3	T0	П0	C2	T1	П0	C5
4	T0	П0	C3	T1	П0	C6
5	T0	K1	C1	T1	П0	C7
6	T0	П1	C2	T1	П1	C4
7	T0	П1	C3	T1	П1	C5
8	T0	K2	C2	T1	П1	C6
9	T0	П2	C3	T1	П1	C7
10	T0	K3	C3	T1	П2	C4
11				T1	П2	C5
12				T1	П2	C6
13				T1	П2	C7
14				T1	П3	C4
15				T1	П3	C5
16				T1	П3	C6
17				T1	П3	C7
18				T1	K4	C4
19				T1	П4	C5
20				T1	П4	C6
21				T1	П4	C7
22				T1	K5	C5
23				T1	П5	C6
24				T1	П5	C7
25				T1	K6	C6
26				T1	П6	C7
27				T1	K7	C7

а

№	ЛП 0			ЛП 1		
1	T0	K0	C0			
2	T0	П0	C1	T1	П0	C2
3	T2	П0	C4	T1	П0	C3
4	T2	П0	C5	T3	П0	C6
5	T0	K1	C1	T3	П0	C7
6	T2	П1	C4	T1	П1	C2
7	T2	П1	C5	T1	П1	C3
8				T3	П1	C6
9				T3	П1	C7
10				T1	K2	C2
11	T2	П2	C4	T1	П2	C3
12	T2	П2	C5	T3	П2	C6
13				T3	П2	C7
14				T1	K3	C3
15	T2	П3	C4	T3	П3	C6
16	T2	П3	C5	T3	П3	C7
17	T2	K4	C4			
18	T2	П4	C5	T3	П4	C6
19	T2	K5	C5	T3	П4	C7
20				T3	П5	C6
21				T3	П5	C7
22				T3	K6	C6
23				T3	П6	C7
24				T3	K7	C7

б

№	ЛП 0			ЛП 1		
1	T0	K0	C0			
2	T0	П0	C1	T1	П0	C2
3	T0	K1	C1	T1	П0	C3
4	T2	П0	C4	T1	П1	C2
5	T2	П0	C5	T1	П1	C3
6	T2	П1	C4	T1	K2	C2
7	T2	П1	C5	T1	П2	C3
8	T2	П2	C4	T1	K3	C3
9	T2	П2	C5	T3	П0	C6
10	T2	П3	C4	T3	П0	C7
11	T2	П3	C5	T3	П1	C6
12	T2	K4	C4	T3	П1	C7
13	T2	П4	C5	T3	П2	C6
14	T2	K5	C5	T3	П2	C7
15				T3	П3	C6
16				T3	П3	C7
17				T3	П4	C6
18				T3	П4	C7
19				T3	П5	C6
20				T3	П5	C7
21				T3	K6	C6
22				T3	П6	C7
23				T3	K7	C7

в

Рис. 4. Шаги работы: *а* – метода μ1 на 2 потоках и методов; *б* – μ1; *в* – μ1к на 4 потоках

матрицы и позволяет строкам с более высоким индексом начать обрабатываться в более ранний момент времени. Продемонстрируем это на примере СЛАУ с матрицей размером 8×8, разбиваемой на 2 и 4 потока и решаемой методами μ1 и μ2 на 2 логических процессорах ЛП0 и ЛП1 (рис. 4, 5). На рис. 4, 5 используются следующие обозначения: T*i* – поток *i*, C*j* – строка *j* матрицы СЛАУ, K*r* – операция расчета коэффициентов активной строки C*r* при прямом ходе метода Гаусса, П*r* – операция пересчета коэффициентов строки C*j* от активной строки C*r* при прямом ходе, ОК*r* – операция расчета коэффициентов активной строки C*r* при обратном ходе, ОП*r* – операция пересчета коэффициентов строки C*j* от активной строки C*r* при обратном ходе. Операции, входящие в критический путь, выделены курсивом.

Рис. 4, *а* показывает шаги работы метода μ1 при разбиении СЛАУ на 2 потока T0 = {0, 1, 2, 3} и T1 = {4, 5, 6, 7}. Длина критического пути составляет 27 шагов, загрузка ЛП0 составляет 37,0%, ЛП1 – 96,3%. Рис. 4, *б* показывает работу μ1 при разбиении СЛАУ уже на 4 потока T0 = {0, 1}, T1 = {2, 3}, T2 = {4, 5}, T3 = {6, 7}, причем потоки 0, 2 назначены на ЛП0, потоки 1, 3 – на ЛП1. Длина критического пути сократилась до 24 шагов, загрузка ЛП0 увеличилась до 58.3%, загрузка ЛП1 уменьшилась до 91,7%. Рис. 4, *в* показывает работу кооперативного метода μ1к при разбиении СЛАУ на 4 потока. По сравнению с методом μ1, метод μ1к реализует кооперативную модель выполнения потоков и позволяет переставлять операции K*r*, П*r*, ОК*r* и ОП*r* с целью сокращения длины критического пути и увеличения загрузки оборудования. Благодаря этому он уменьшает до 23 шагов длину критического пути и увеличивает загрузку ЛП0 до 60,9%, ЛП1 – до 95,7%. Увеличение количества потоков с 2 до 4 увеличивают коэффициент распараллеленности блочно-параллельного алгоритма μ1 с 36/27 = 1,33 до 36/24 = 1,5, а переход от метода μ1 к методу μ1к еще больше увеличивают коэффициент распараллеленности, до 36/23 = 1,57.

№	ЛП 0			ЛП 1		
1	Т0	К0	С0			
2	Т0	П0	С4	Т1	П0	С1
3	Т2	П0	С2	Т1	П0	С5
4	Т2	П0	С6	Т3	П0	С3
5				Т3	П0	С7
6				Т1	К1	С1
7	Т0	П1	С4	Т1	П1	С5
8	Т2	П1	С2	Т3	П1	С3
9	Т2	П1	С6	Т3	П1	С7
10	Т2	К2	С2			
11	Т2	П2	С6	Т1	П2	С5
12	Т0	П2	С4	Т3	П2	С3
13				Т3	П2	С7
14				Т3	К3	С3
15	Т0	П3	С4	Т3	П3	С7
16	Т2	П3	С6	Т1	П3	С5
17	Т0	К4	С4			
18	Т2	П4	С6	Т1	П4	С5
19				Т3	П4	С7
20				Т1	К5	С5
21	Т2	П5	С6	Т3	П5	С7
22	Т2	К6	С6			
23				Т3	П6	С7
24				Т3	К7	С7

а

№	ЛП 0			ЛП 1		
1	Т0	К0	С0			
2	Т2	П0	С2	Т1	П0	С1
3				Т1	К1	С1
4	Т2	П1	С2	Т3	П0	С3
5	Т2	К2	С2	Т3	П1	С3
6	Т4	П0	С4	Т3	П2	С3
7	Т4	П1	С4	Т3	К3	С3
8	Т4	П2	С4	Т5	П0	С5
9	Т4	П3	С4	Т5	П1	С5
10	Т4	К4	С4	Т5	П2	С5
11	Т6	П0	С6	Т5	П3	С5
12	Т6	П1	С6	Т5	П4	С5
13	Т6	П2	С6	Т5	К5	С5
14	Т6	П3	С6	Т7	П0	С7
15	Т6	П4	С6	Т7	П1	С7
16	Т6	П5	С6	Т7	П2	С7
17	Т6	К6	С6	Т7	П3	С7
18				Т7	П4	С7
19				Т7	П5	С7
20				Т7	П6	С7
21				Т7	К7	С7

б

Рис. 5. Шаги работы: *а* – метода μ2 на 4 потоках; *б* – метода μ2к на 8 потоках

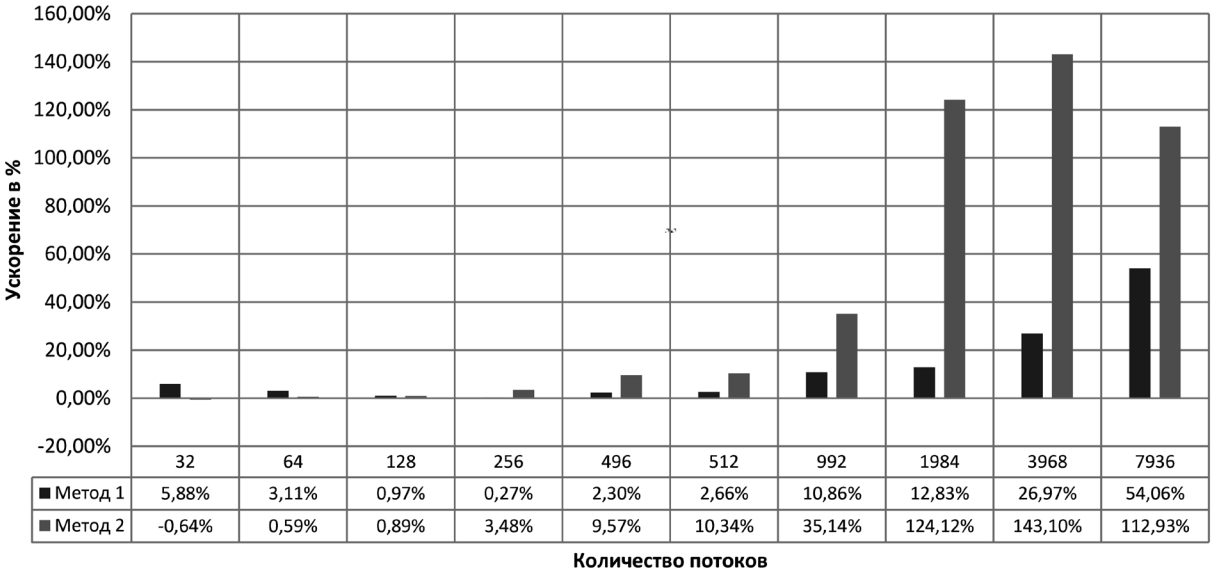
Положительные свойства кооперативной модели выполнения потоков проявляются также на методе μ2 решения СЛАУ. Рис. 5, *а* показывает работу μ2 при разбиении СЛАУ на 4 потока Т0 = {0, 4}, Т1 = {1, 5}, Т2 = {2, 6}, Т3 = {3, 7}, причем потоки 0, 2 назначены на ЛП0, потоки 1, 3 – на ЛП1. Длина критического пути составляет 24 шага, коэффициент распараллеленности – 1,5, загрузка ЛП0 и ЛП1 стала более равномерной: 66,7% и 83,3% соответственно. Метод μ2к изменяет порядок выполнения операций при разбиении СЛАУ на 8 потоков Тi = {i}, причем потоки 0, 2, 4, 6 назначены на ЛП0, потоки 1, 3, 5, 7 – на ЛП1 (рис. 5, *б*). Длина критического пути уменьшилась до 21 шага, коэффициент распараллеленности увеличился до 1,71, загрузка ЛП0 и ЛП1 возросла до 76,2% и 95,2% соответственно.

Принципиальным отличием метода μ1 от метода μ1к и метода μ2 от метода μ2к является строгая упорядоченность обработки строк. В μ1 и μ2 нельзя выполнить операцию Кr+1 до тех пор пока не завершено использование результатов операции Кr. В то же время μ1к (рис. 4, *в*) и μ2к (рис. 5, *б*) группируют операции по потокам благодаря механизму UMS.

При увеличении числа логических процессоров до 16 и числа потоков до 7936 для матрицы СЛАУ размером 15872 возможности переупорядочения операций и сокращения критического пути кооперативной моделью возрастают.

Анализ кооперативной модели оптимизации выполнения потоков

Более точный анализ свойств алгоритмов μ1к и μ2к базируется на детальном рассмотрении времени выполнения операций блочно-параллельных алгоритмов. Вычислительные операции Кr, Пr, ОКr и ОПr имеют различную вычислительную сложность. В частности, сложность Кr и ОКr зависит от номера активной строки r, а операций Пr и ОПr также от номера строки j, которая пересчитывается в зависимости от r. Более трудоемкие операции посылки-приема сообщений вставляются между операциями Кr (ОКr) и операциями Пr (ОПr). В результате получаем граф операций (задач), который может использоваться для синтеза



	32	64	128	256	496	512	992	1984	3968	7936
Метод 1	5,88%	3,11%	0,97%	0,27%	2,30%	2,66%	10,86%	12,83%	26,97%	54,06%
Метод 2	-0,64%	0,59%	0,89%	3,48%	9,57%	10,34%	35,14%	124,12%	143,10%	112,93%

Рис. 6. Ускорение μ1к по сравнению с μ1 и μ2к по сравнению с μ2 для СЛАУ размером 15872

и оптимизации параллельных вычислений на многоядерной системе. Особенностью синтеза является назначение операций (потоков) на процессоры и зануление операций посылки-приема сообщений в пределах одного процессора (ядра). Критический путь на таком графе является более точной версией критического пути, построенного из шагов работы алгоритма. Сумма времен выполнения всех операций характеризует вычислительную сложность алгоритма на одном процессоре. Сумма времен выполнения операций, лежащих на критическом пути, характеризует вычислительную сложность алгоритма на многоядерной системе.

На параметры многопоточного приложения влияют и другие факторы. Чрезмерное увеличение количества активных потоков может привести к повышению количества прерываний ContextSwitch [1] и снижению производительности всего приложения. Нарушение строгого порядка выполнения потоков невозможно без использования кооперативной модели и UMS, что приводит к серьезному усложнению программного кода. Особенно сильно прерывания ContextSwitch могут увеличить время выполнения обратного хода, которое для СЛАУ размером 15872 составляет лишь 0.00945% от общего объема вычислений. Несмотря на недостатки, применение кооперативной модели выполнения потоков гарантирует следующие преимущества: сокращение критического пути за счет лучшего порядка выполнения потоков; уменьшение числа прерываний ContextSwitch за счет уменьшения конкуренции между потоками за процессорное время; быстрое переключение между потоками за счет использования UMS.

Экспериментальное исследование кооперативной модели

Эксперименты проведены на многоядерной системе, оснащенной двумя процессорами Intel®Xeon®CPUE5520 и 16 GB оперативной памяти, работающей с частотой 1 GHz. Каждый процессор оснащен технологией Hyper-Threading Technology и включает 4 ядра, работающие с частотой 2,26 GHz, имеет разделяемую между ядрами кэш память емкостью 8 MB, а каждое ядро имеет локальную кэш память первого уровня емкостью 64 KB и второго уровня емкостью 256 KB. Кроме того, каждый из 2-х физических процессоров выполняет доступ к локальной и удаленной памяти по принципу точка-точка с использованием технологии NUMA и технологии QPI. Управление многоядерной системой осуществляет ОС Windows Server 2012 R2 64 бит.

На рис. 6 представлены две зависимости для СЛАУ размером 15872: ускорение μ1к по сравнению с μ1 в процентах в зависимости от числа потоков (от 32 до 7936) и ускорение μ2к по сравнению с μ2 также в зависимости от числа потоков. Они показывают, насколько модель с кооперативным выполнением потоков превосходит модель с вытесняющей многозадачностью. Для метода μ1к ускорение снижается с 5,88% до

0,27% при росте числа потоков с 32 до 256, а затем увеличивается до 54.06% при росте числа потоков с 256 до 7936. Для метода μ2к получено замедление на 0.64% при 32 потоках. Затем при росте числа потоков с 64 до 3968 ускорение монотонно увеличивается с 0,59% до 143,1%, а при 7936 потоках ускорение снижается до 112,93%. Такая динамика изменения ускорения объясняется изменением состава доминирующих факторов на различных диапазонах значений параметров многопоточного приложения.

Заключение

Предложена кооперативная модель оптимизации выполнения потоков на многоядерной системе, повышающая производительность многопоточного приложения за счет сокращения критического пути при перестановке операций распараллеливаемого алгоритма, масштабирования задачи при росте числа потоков, исключения конкуренции между потоками, минимизации взаимодействия потоков с ядром операционной системой.

Литература

1. Прихожий, А.А. Исследование методов реализации многопоточных приложений на многоядерных системах / А.А. Прихожий, О.Н. Карасик // Информатизация образования, 2014, № 1. – С. 43-62.

2. Ортега, Дж. Введение в параллельные и векторные методы решения линейных систем / Дж. Ортега // М.: Мир, 1991. – 367 с.

3. Рихтер, Дж. Windows для профессионалов. Создание эффективных WIN32-приложений с учетом специфики 64-разрядной версии Windows / Дж. Рихтер // Питер, PP, 2001. –752 с.

4. Probert, D. Inside Windows 7 User Mode Scheduler / D. Probert // Channel9 [Электронный ресурс]: – Режим доступа: http://channel9.msdn.com/shows/Going+Deep/Dave-Probert-Inside-Windows-7-User-Mode-Scheduler-UMS/. – Дата доступа: 30.12.2014