УДК 621.395

О параллельном алгоритме вычисления нормализующей константы замкнутой однородной сети массового обслуживания

Е. В. Скрипко

В данной работе оценивается эффективность применения технологии параллельного программирования для анализа сетей систем массового обслуживания (СеМО) большой размерности. На примере прямого алгоритма вычисления нормализующей константы замкнутой однородной СеМО разработан параллельный алгоритм её вычисления, основанный на декомпозиции пространства состояний СеМО. Получены оценки коэффициента ускорения по сравнению с последовательным алгоритмом в зависимости от размерности СеМО.

Ключевые слова: замкнутая сеть массового обслуживания, стационарное распределение вероятностей, нормализующая константа, пространство состояний.

1. Введение

Сети систем массового обслуживания известны как наиболее адекватные математические модели телекоммуникационных систем с пакетной коммутацией и стохастическим мультиплексированием ресурсов. Они широко применяются для анализа параметров качества современных информационных и вычислительных сетей и являются эффективным средством аналитического моделирования телекоммуникационных систем различного назначения на этапах их разработки, проектирования, развёртывания и эксплуатации [1]. Особенно эффективны СеМО при анализе задержек в телекоммуникационных сетях, в которых имеет место статистическое мультиплексирование различных ресурсов между большим количеством абонентов, таких как мультисервисные сети. Адекватные модели реальных телекоммуникационных систем, разработанные в классе СеМО, имеют большую размерность, которая может достигать десятков тысяч узлов и классов. Анализ этих моделей на вычислительных системах с традиционной, последовательной архитектурой крайне затруднителен или даже невозможен. Это обусловлено: 1) чрезвычайно большой вычислительной сложностью этих моделей, 2) громадным размером требуемой оперативной памяти, и 3) как следствие первых двух факторов, понижающейся точностью вычислений. Одним из наиболее перспективных путей преодоления выше перечисленных трудностей является использование для анализа таких моделей вычислительных систем с мультипроцессорной архитектурой и разработка параллельных версий алгоритмов, реализующих существующие методы анализа СеМО.

2. Постановка задачи

Будет рассмотрена замкнутая однородная сеть массового обслуживания. СеМО, в которой отсутствует источник/сток требований и в которой по её узлам (системам массового обслуживания) циркулирует постоянное число однотипных требований, является замкнутой и однородной.

В замкнутых и однородных СеМО, которые удовлетворяют условию локального баланса [2], стационарное распределение вероятностей $P(\vec{n})$ её состояний имеет мультипликативную форму. Это означает, что поведение каждого узла в вероятностном смысле не зависит от поведения всех остальных узлов:

$$P(\vec{n}) = \frac{1}{G(N,L)} \cdot \prod_{i=1}^{L} f_i(n_i), \tag{1}$$

где L – число узлов, $i = \overline{1,L}$;

N – общее число требований в узлах сети;

$G(N,L)$ – нормализующая константа распределения вероятностей $P(\vec{n})$;

$f_i(n_i)$ – функция, пропорциональная вероятности того, что в i-ом узле СеМО находится n_i требований. Она зависит от дисциплины обслуживания узла, интенсивности обслуживания требований в нём, числа обслуживающих приборов, а также количества требований, пребывающих в этом узле:

$$f_i(n_i) = \frac{\left(\frac{\omega_i}{\mu_i}\right)^{n_i}}{\sum_{S=1}^{n_i} \alpha_i(S)}, \tag{2}$$

где $\vec{\omega} = \omega_i$ – вектор, определяющий решение уравнения равновесия СеМО, и его элементы пропорциональны вероятностям пребывания некоторого требования в узлах СеМО. Это уравнение записывается в матричном виде следующим образом: $\vec{\omega} = \Theta\vec{\omega}$, в котором $\Theta = \theta_{i,j}$ – маршрутная матрица СеМО;

$\mu = (\mu_i)$ – вектор интенсивностей обслуживания в узлах;

$\alpha_i(S)$ – функция, пропорциональная суммарной интенсивности обслуживания в i-ом узле.

Метод прямого вычисления [3] нормализующей константы замкнутой однородной СеМО описывается следующей формулой:

$$G(N,L) = \sum_{\vec{n} \in \Omega} \prod_{l=1}^{L} f_i(n_i), \tag{3}$$

где $\Omega = \Omega(N,L) = \left\{ \vec{n} = n_1, \ldots n_i, \ldots n_L, \;\middle|\; n_i \geq 0, i \in \overline{1,L}, \sum_{i=1}^{L} n_i = N \right\}$ – пространство состояний СеМО.

Как видно из (3), для вычисления нормализующей константы необходимо провести суммирование по всему пространству состояний Ω произведений функции $f_i(n_i)$. Реализация данной задачи на вычислительных средствах с традиционной архитектурой занимает очень большое количество времени. Есть много альтернативных способов решения данной проблемы, например, применение метода интегрального представления нормализующей константы [4] или метода анализа средних [2, 5], но их применение ограничено соответст-

вующим классом СеМО. В данной работе была сделана оценка эффективности применения методов параллельного программирования для вычисления нормализующей константы наиболее универсальным методом – методом прямого вычисления нормализующей константы по формуле (3).

Если рассматривать идеальный случай, то время, затрачиваемое на выполнение какой-либо задачи на n процессорах, должно быть в n раз меньше. Реальная ситуация выглядит по-другому. Согласно закону Амдала [6], ускорение работы программы на n процессорах определяется следующим образом:

$$S \le \frac{1}{d + \frac{1-d}{n}}, \tag{4}$$

где d – доля непараллельного кода в программе. Для систем, использующих модель общей памяти, долю непараллельного кода составляют операторы, которые выполняются только главной нитью программы, а для систем, использующих механизм передачи сообщений, эта доля состоит из операторов, выполнение которых дублируется на всех процессорах. Оценить данную величину можно только практически, выполняя программу на разном числе процессоров.

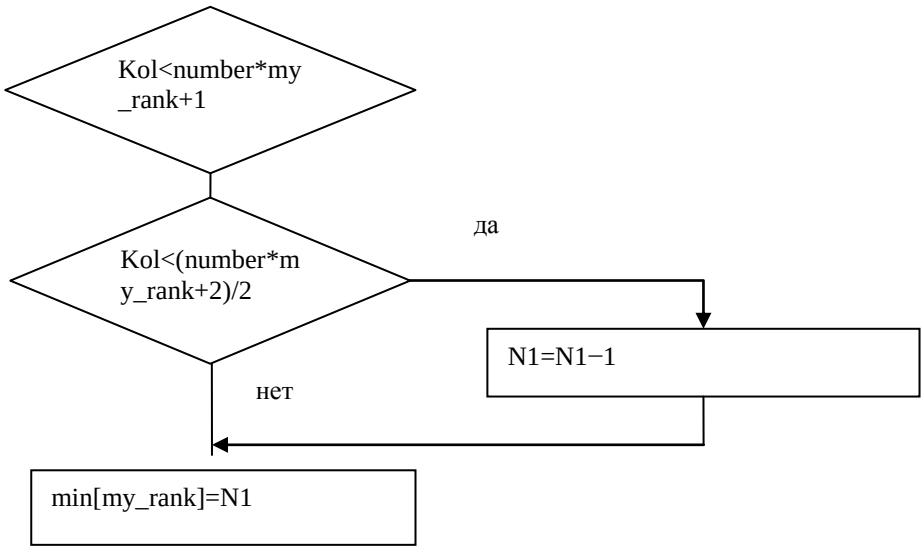
Для решения задачи распараллеливания вычислений нормализующей константы была выбрана технология MPI (Message Passing Interface) [7, 8]. В её основе лежит идея передачи сообщений между процессами. Выбор MPI основан на следующих преимуществах данной технологии:

– операции MPI определяются не процедурно, а логически, т.е. внутренние механизмы выполнения операций скрыты от пользователя;
– используются непрозрачные объекты (группы, коммуникаторы, типы и т.д.);
– возможность использования в языках Фортран, Си, Си++;
– способность параллельных программ выполняться на гетерогенных системах, т.е. на системах, состоящих из процессоров с различной архитектурой [9].

В данной статье выбран метод декомпозиции пространства состояний по количеству требований в первом узле. Общее количество состояний системы определяется следующим образом:

$$C_{L-1}^{L+N-1} = \frac{(L+N-1)!}{N!\ L-1\ !}\ . \tag{5}$$

Если зафиксировать количество требований, находящихся в первом узле, то количество требований, циркулирующих в остальных узлах сети, будет определяться по аналогичной формуле. Следовательно, при увеличении количества требований в первом узле на единицу, количество состояний, у которых в первом узле находится фиксированное число требований $N1$, будет увеличиваться в $\frac{L1+N2-1}{N2}$ раз, где $L1 = L-1$ – количество узлов СеМО за исключением первого узла, $N2 = N - N1$ – количество требований, которые циркулируют в узлах сети за исключением требований, находящихся в первом узле. Основываясь на вышесказанном, можно разбить пространство состояний СеМО на части по количеству процессоров (см. рис. 1).



number – среднее количество состояний, приходящееся на один процессор;
my_rank – ранг процессора;
N1 – фиксированное количество требований в первом узле, с которого начинается первое подпространство, обрабатываемое на процессоре с номером my_rank
kol – размер подпространства состояний

Рис. 1. Алгоритм декомпозиции пространства состояний

Далее вычисление произведений функции $f_i(n_i)$ в каждой группе можно производить независимо от остальных групп с последующим суммированием результатов (рис. 2 и 3).

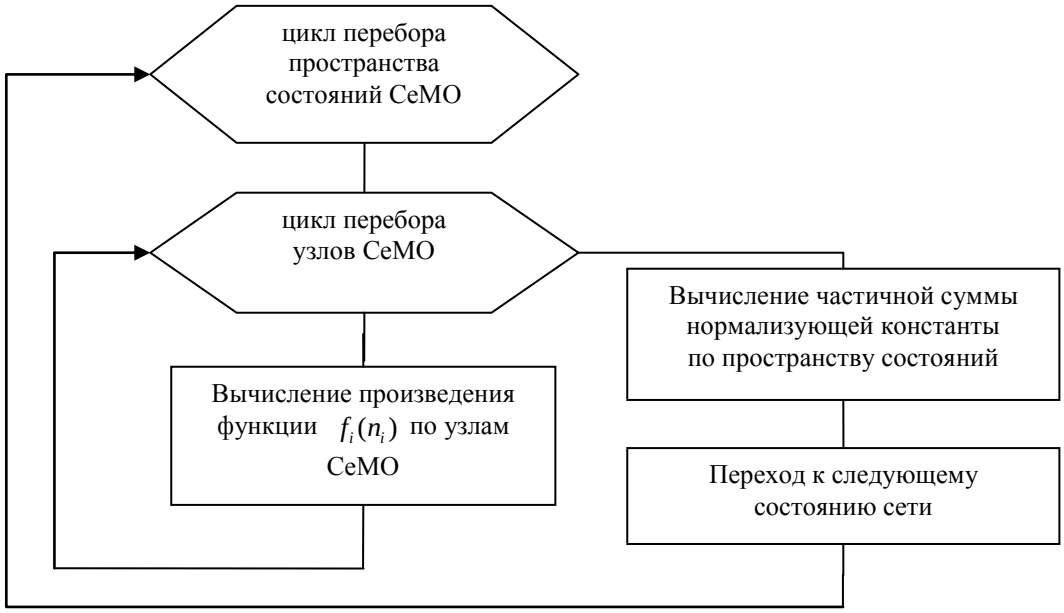


Рис. 2. Алгоритм прямого вычисления нормализующей константы

Рис. 3. Параллельный алгоритм вычисления нормализующей константы

Фрагменты последовательной и параллельной версии программ вычисления нормализующей константы приведены на рисунках 4 и 5.

```
for (i=0;i<=(space_states−1);i++)   // цикл перебора пространства состояний сети
{
   P[i]=1;
   for (t=0; t<=L−1;t++)            // цикл по узлам сети

         P[i]=P[i]*f_j(next[t]); // вычисление произведения функций  f_i(n_i)
         G=G+P[i];                    // вычисление нормализующей константы
         nextstate(n, next);              // переход к  следующему состоянию сети
}
```

Рис. 4. Фрагмент последовательной программы вычисления нормализующей константы

```
MPI_Init (&argc, &argv); // инициализировать MPI
MPI_Comm_rank (MPI_COMM_WORLD, &my_rank); // номер данной ветви
MPI_Comm_size (MPI_COMM_WORLD, &num_proc); // общее число ветвей
c2=c1/num_proc;
number=ceil(c2);
MPI_Bcast(&number, sizeof(number), MPI_INT, 99, MPI_COMM_WORLD); //
широковещательная рассылка
if (my_rank >=0 && my_rank<=num_proc−1)
        for (i=kol;i<=kol+1;i++)              // цикл по подпространству состояний
    ……….
                if  (i<space_states)
                {
                        P[i]=1;
        for (t=0; t<=L−1;t++)
            P[i]=P[i]*f_j(next[t]);
                        G=G+P[i];
                        nextstate(n, next);
                }
……..
for (i=0; i<num_proc−1; i++)
{
        MPI_Recv (&Gi, sizeof(Gi), MPI_DOUBLE, i, 99, MPI_COMM_WORLD, &status);
        // получение сообщений
        G += Gi;                        // суммирование вычислений
}
```

Рис. 5. Фрагмент параллельной программы вычисления нормализующей константы

3. Результаты

С целью исследования временны́х характеристик разработанного алгоритма был проведён ряд экспериментов на кластере Центра параллельных вычислительных технологий Сибирского государственного университета телекоммуникаций и информатики (ЦПВТ СибГУТИ) на базе двухъядерных процессоров Intel Xeon 5150 [2]. В экспериментах были задействованы от 4 до 16 процессоров. Результаты представлены в табл. 1. В этой таблице приведены данные по ускорению выполнения операции вычисления нормализующей константы при разном количестве требований N и фиксированном числе узлов СеМО $L=15$. Коэффициент ускорения времени выполнения параллельной программы по сравнению с последовательной вычислялся по следующей формуле:

$$S = \frac{T_{serial}}{T_{parall}} \quad , \tag{6}$$

где T_{serial} и T_{parall} − времена выполнения последовательной и параллельной программ соответственно.

Таблица 1. Коэффициент ускорения времени работы параллельной программы при анализе СеМО из L=15 узлов

Количество процессоров	Количество требований N / Размерность пространства состояний			
	50 / $4.786 \cdot 10^{13}$	100 / $3.126 \cdot 10^{17}$	150 / $6.6 \cdot 10^{19}$	200 / $3.138 \cdot 10^{21}$
4	2.11	2.16	2.19	2.21
8	3.27	3.33	3.38	3.42
16	4.11	4.16	4.21	4.24

Как видно из табл. 1, применение параллельного алгоритма не даёт увеличение скорости в n раз из-за присутствия в программе непараллельного кода. Объём непараллельного кода пропорционален количеству узлов СеМО L, количеству требований N, циркулирующих в узлах сети, и числу используемых процессоров n.

4. Заключение

В результате анализа полученных результатов можно сделать вывод, что применение методов параллельного программирования для расчёта характеристик СеМО большой размерности (несколько тысяч узлов и классов требований) является очень перспективным и при использовании современных высокопроизводительных кластерных вычислительных систем может быть достигнуто значительное ускорение времени расчёта такой СеМО. Применение параллельного вычисления только нормализующей константы замкнутой однородной СеМО может дать четырёхкратное ускорение времени вычислений. Это ускорение достигается за счёт декомпозиции пространства состояний СеМО на ряд подпространств и параллельного вычисления частичных сумм по этим подпространствам. Причём оно становится более значительным при повышении размерности СеМО. Кроме того, в результате декомпозиции производится суммирование произведений примерно одного масштаба, что позволяет увеличить точность вычисления нормализующей константы.

Список литературы

1. Вишневский В.М., Теоретические основы проектирования компьютерных сетей— М:, Техносфера — 2003 — 512 стр.
2. Митрофанов Ю. И., Беляков В. Г., Курбангулов В. Х. Методы и программные средства аналитического моделирования сетевых систем. – Препринт, М.: Научный совет по комплексной проблеме Кибернетика АН СССР, 1982, 68 стр.
3. Курносов М. Г. Руководство пользователя кластерной вычислительной системы Центра параллельных вычислительных технологий.– Новосибирск: СибГУТИ, 2007.
4. Mitra D., and McKenna J.: Asymptotic expansions and integral representations of moments of queue lengths in closed Markovian networks, J. ACM, 1984, 31, (2), pp. 346 – 360.
5. Reiser M., Lavenberg S. S., Mean-value analysis of closed multichain queuing networks, Journal of the ACM, 1980, v.27, No 2, pp. 126 – 141.
6. Amdahl. G. Validity of the single-processor approach to achieving large-scale computing capabilities. // Proc. 1967 AFIPS Conf., AFIPS Press. - 1967. - v. 30. - p. 483.
7. Message-Passing Interface Forum, Document for a Standard Message-Passing Interface,1993. Version 1.0. URL: `http://www.unix.mcs.anl.gov/mpi/`
8. Message-Passing Interface Forum, MPI-2: Extensions to the Message-Passing Interface,1997. URL: `http://www.unix.mcs.anl.gov/mpi/`

9. Корнеев В. Д. Параллельное программирование в MPI. – 2-е изд., испр. – Новосибирск: Изд-во ИВМиМГ СО РАН, 2002.

Статья поступила в редакцию 24.11.2009

Скрипко Елена Владимировна
аспирант кафедры БИСС Сибирского государственного университета телекоммуникаций и информатики (630102, Новосибирск, ул. Кирова, 86)
e-mail: s_el@inbox.ru

About parallel algorithm for evaluation of normalizing constant for closed homogeneous queueing network

Skripko E. V.

Efficiency of using parallel programming methods for analysis of high dimensional closed queueing networks is considered at this article. Parallel algorithm for evaluation of normalizing constant of closed homogeneous queueing network is described. It is based on decomposition of closed queueing network state space. Acceleration of parallel algorithm in comparison with serial one is evaluated.

Keywords: closed queueing network, stationary probability distribution, normalizing constant, state space.