УДК 681.3

С.Д. ПОГОРЕЛЫЙ, Ю.В. БОЙКО, А.Д. ГУСАРОВ, С.И. ЛОЗИЦКИЙ

ПОДХОД К ПАРАЛЛЕЛЬНОМУ РЕШЕНИЮ ОСНОВНОЙ ПОТОКОВОЙ ЗАДАЧИ БОЛЬШОЙ РАЗМЕРНОСТИ

Ключевые слова: *система алгоритмических алгебр, формализация алгоритмов, параллельные схемы алгоритмов, параллелизм по данным, распределенные системы.*

ВВЕДЕНИЕ

Решение задачи маршрутизации в Internet выполняется миллионы раз в сутки, поэтому время поиска оптимального маршрута становится критическим параметром алгоритмов маршрутизации и определяет эффективность функционирования сети в целом. Это, в свою очередь, требует максимально возможного повышения эффективности маршрутизаторов.

Выводы экспертов в области компьютерных технологий сходятся в том, что ресурс экстенсивного наращивания эффективности микропроцессорных систем за счет увеличения сложности и тактовой частоты процессоров себя исчерпал. В последующих исследованиях и разработках акцент перенесен на создание многоядерных процессоров (об этом свидетельствует появление таких чипов от Intel, AMD, SUN Microsystems, IBM), мультипроцессорных систем и кластеров, а значит и параллельных алгоритмов. В связи с этим группа по развитию IRTF (Internet Research Task Force), координирующая долгосрочные исследовательские проекты по стеку протоколов TCP/IP, занимается созданием новых протоколов и поиском перспективных методов решения задачи маршрутизации с целью преодоления существующих недостатков на качественно новом уровне.

Алгоритм Эдмондса–Карпа [1], являющийся одной из первых реализаций метода Форда–Фалкерсона [2, 4], рассматривается как альтернативный вариант использования в перспективных протоколах маршрутизации и сетевых устройствах [3, 10]. Существует множество архитектур, способных обеспечить повышение продуктивности этого алгоритма, и важный класс среди них — это параллельные архитектуры. Ранее [9] авторами был предложен подход к распараллеливанию и оптимизации данного алгоритма для применения на вычислительных системах с общей памятью, что может позволить в полной мере использовать преимущества современных многоядерных процессоров для решения основной потоковой задачи. Но такие вычислительные системы ограничивают максимальную размерность задачи сравнительно небольшим объемом доступной оперативной памяти. Поэтому для решения задач, оперирующих большими объемами данных, используются распределенные параллельные вычислительные архитектуры (кластеры). В связи с этим далее будем излагать применение математического аппарата САА–М [5] в качестве инструмента для формализации и модификации алгоритма Эдмондса–Карпа с целью получения спектра его параллельных регулярных схем, оптимизированных по критериям, которые определяются особенностями и ограничениями кластерных систем.

КРИТЕРИИ ОПТИМИЗАЦИИ

Математическая модель сети представляет взвешенный граф, который в данном случае удобно записать в виде матрицы смежностей. Такая структура данных может быть легко промоделирована в оперативной памяти компьютеров, и с ней легко и удобно работать, особенно когда речь идет о геометрическом распараллеливании. При этом необходима матрица потоков и массив вершин графа (см. ниже). При большой размерности задачи для размещения всех этих данных может потребоваться значительный объем оперативной памяти.

Одно из основных назначений распределенных систем — решение задач большой размерности, когда есть необходимость одновременно оперировать таким объемом данных, который не может быть размещен в оперативной памяти отдельной вычислительной машины. Отсюда вытекает *первый критерий*: поскольку размерность задачи большая, алгоритм должен исключить возможность размещения на одном узле данных больше, чем имеющейся в нем доступной памяти. В идеальном случае, если задача требует объема памяти M, а на каждом узле ее доступен объем m $(M > m)$, то каждую структуру данных из перечисленных выше следует равномерно разбить и разместить на n узлах кластера, где $n = \left\lceil \frac{M}{m} \right\rceil + 1$.

Узлы в распределенной системе связаны между собой каналами, пропускная способность которых значительно меньше, а задержки намного больше, чем соответствующие параметры в системах с общей памятью. Это один из основных недостатков кластерных систем [6]. Таким образом, можно сформулировать *второй критерий*: необходимо минимизировать объем данных, передаваемых между параллельными частями алгоритма, а также уменьшить частоты таких передач.

Поскольку обмен данными между узлами все же занимает значительную часть времени, следует по возможности нагрузить их полезной работой в процессе таких обменов (*третий критерий*).

Существуют две основные концепции параллелизма: параллелизм по данным (одинаковые действия выполняются одновременно над разными непересекающимися подмножествами данных) и параллелизм по коду (разные действия полностью или частично параллельно выполняются над одними и теми же данными). Настоящая статья основана на концепции параллелизма по данным. Для этого матрицы разбиваются на необходимое количество приблизительно одинаковых частей. Их размер определяется первым критерием и количеством доступных узлов, на которых могут производиться вычисления. Каждый узел содержит определенную часть данных и при необходимости дополнительной информации должен инициировать акт обмена данными с соседним узлом (или соседними узлами). Существует ряд способов моделирования распределенных вычислений, среди которых наиболее распространенными можно назвать MPI [7, 8], параллельные виртуальные машины PVM, Java и другие. С учетом особенностей каждого из них можно ввести дополнительные критерии оптимизации, специфичные для конкретного способа, но это уже вопрос более технического, чем алгоритмического характера. Здесь внимание будет сосредоточено в основном на критериях, общих для всех распределенных систем.

ФОРМИРОВАНИЕ CAA-M-СХЕМ

Далее используются следующие обозначения: $G = G(V, E)$ — ориентированный взвешенный граф (графоид); $V = V(G)$ — множество вершин графа ($|V| = n$); $E = E(G)$ — множество ребер графа; s (source) – исток; t (target) — сток ($s \neq t$; $s, t \in V$); $c = c(e)$ — вес, или пропускная способность ребра e; $f = f(e)$ — поток через ребро e ($e \in E$); f^0 — полный поток в графе G.

Определим базовые начальные условия и исходные данные:

- для описания работы алгоритма используются:

1) две матрицы $n \times n$, характеризующие сеть: матрицу смежностей (содержит веса ребер) и матрицу потоков (содержит потоки через все ребра);

2) массив, содержащий все вершины графа;

3) очередь FIFO для временного хранения вершин в процессе поиска пути;

- вершины пронумерованы от 0 до $n-1$ в произвольном порядке;
- с каждой вершиной графа ассоциируются, кроме номера, еще два свойства: *dad* (содержит номер вершины, из которой была достигнута данная вершина во время поиска пути к стоку) и *vstd* (visited, в классическом алгоритме поиска в ширину свойство может принимать только два значения: true или false; впоследствии его тип будет изменен);

• обращение к свойствам проводится по аналогии с обращением к полям структур в языках программирования; если идентификатор вершины используется без указания свойства, то подразумевается ее номер.

В краткой форме алгоритм Эдмондса–Карпа можно записать в виде регулярной САА–схемы

$$Edmonds-Karp=(f^0:=0)*\underset{\alpha(BFS)}{\{}Update\},\qquad(1)$$

где предикат $\alpha(BFS)$ принимает значение истина, если в результате поиска в остаточной сети не найдено пути из s в t; оператор $Update$ включает в себя необходимые модификации в матрице потоков и в полном потоке f^0. Отличительной и ключевой особенностью алгоритма (1) является отыскание аугментального пути, где используется алгоритм поиска в ширину (BFS) [4], классический вариант которого можно подать следующей регулярной интерпретированной САА-схемой:

$$BFS1=(Q:=\{s\})*(q:=s)*Reset\_visited1*\qquad(2)$$

$$*\ \underset{[Q=\varnothing]\vee[t.vstd]}{\{}$$

$$(v:=0)*\underset{[v\ge n]}{\{}[(q,v)\in E(G)]\wedge[\overline{v.vstd}]\wedge[c(q,v)>f(q,v)]$$

$$(Include(Q,v)\vee Œ)*inc(v)\}*Exclude(Q,q)$$

$$\}$$

Здесь Q — очередь, $\varnothing$ — пустое множество, q всегда указывает на голову очереди; $Reset\_visited1$ присваивает значение $false$ свойству $vstd$ всех вершин, кроме s, которая получает значение $true$; Œ обозначает тождественный оператор; $c(q,v)$ — вес ребра (q,v); $f(q,v)$ — поток через соответствующее ребро; оператор $Include(Q,v)$ помещает в конец очереди Q вершину с номером v, помечает ее как посещенную ($v.vstd:=true$) и запоминает, что она была достигнута из вершины q ($v.dad:=q$); $inc(v)$ увеличивает v на единицу; оператор $Exclude(Q,q)$ извлекает вершину из головы очереди.

Критическим местом алгоритма (1) является алгоритм поиска пути, т.е. BFS, где присутствует максимальная вложенность циклов (α-итераций), поэтому внимание сосредоточено на оптимизации именно этой части. Применение концепции параллелизма по данным в схеме (2) описано в [9]. В алгоритме из [9] был изменен тип свойства $vstd$. Кроме того, при распараллеливании на два звена в каждое из них была введена буферная очередь Q_1 (Q_2) для временного хранения вершин, найденных звеном, а также использованы синхронизаторы. Внесение указанных модификаций позволило сформировать параллельную схему алгоритма поиска в ширину

$$BFS2=(Q:=\{s\})*(q_{1,2}:=s)*Reset\_visited2*(\alpha_{1,2}:=true)*\qquad(3)$$

$$*\ \underset{\overline{\alpha}_1\wedge\overline{\alpha}_2\vee[t.vstd>0]}{\{}$$

$$S(\alpha_1)*search_1 2*lock*[Q_1\ne\varnothing]((Q:=Q+Q_1)*$$
$$*nextq_1*(\alpha_2:=true)\vee nextq_1*$$
$$*[q_1=\times]((\alpha_1:=false)\vee Œ))*unlock$$

$$\bullet$$
$$\vee$$

$$S(\alpha_2)*search_2 2*lock*[Q_2\ne\varnothing]((Q:=Q+Q_2)*$$
$$*nextq_2*(\alpha_1:=true)\vee nextq_2*$$
$$*[q_2=\times]((\alpha_2:=false)\vee Œ))*unlock$$

$$\}$$

Здесь q_1,q_2 — локальные в каждом процессе переменные, которые содержат элемент очереди Q, обрабатываемый в данный момент в соответствующем звене;

операции $next\,q_1$, $next\,q_2$ присваивают переменным q_1, q_2 следующие элементы из общей очереди в том порядке, в котором они в нее поступали. Если следующий элемент отсутствует, то присваивается значение $\times$. Операторные скобки *lock...unlock* выделяют критические секции алгоритма, где происходит работа с общей очередью. Поскольку параллельные ветви работают асинхронно (в САА-М обозначается операцией асинхронной дизъюнкции $\overset{\bullet}{\vee}$), для их синхронизации использован синхронизатор $S(\alpha)$, который в САА-М определен следующим образом: $S(\alpha) \overset{def}{=} \underset{\alpha}{\{}\,\text{Œ}\}$. Операция $search_1 2$ ($search_2 2$) производит поиск вершин в своем подмножестве V_1 (V_2), достижимых из q_1 (q_2), и ставит их в конец очереди. Эти операции можно представить следующими схемами:

$$search_1 2 = (Q_1 := \varnothing)^* (v_1 := 0)^* \underset{v_1 in V_1}{\{} ([(q_1, v_1) \in E_1] \wedge [v_1.vstd = 0] \wedge$$

$$\wedge [c(q_1, v_1) > f(q_1, v_1)](Include(Q_1, v_1) \vee \text{Œ})^* inc(v_1)\},$$

$$search_2 2 = (Q_2 := \varnothing)^* (v_2 := \left\lceil \frac{n}{2} \right\rceil)^* \underset{v_2 in V_2}{\{} ([(q_2, v_2) \in E_2] \wedge [v_2.vstd = 0] \wedge$$

$$\wedge [c(q_2, v_2) > f(q_2, v_2)](Include(Q_2, v_2) \vee \text{Œ})^* inc(v_2)\}.$$

В связи с изменением типа свойства *vstd* (теперь его тип совпадает с типом значений пропускных способностей $c(e)$) в схеме (3) использована новая операция $Reset\_visited 2$, которая присваивает значение 0 свойству *vstd* всех вершин, кроме *s*, получающей максимально возможное значение; в операции $Include(Q, v)$ поле $v.vstd$ вместо *true* теперь принимает значение $\min(q.vstd, c(q,v) - f(q,v))$. В результате при нахождении очередного пути в остаточном графе значение $t.vstd$ равно максимальной величине, на которую можно увеличить поток вдоль этого пути и соответственно полный поток. С учетом этого оператор *Update* можно подать следующей САА-схемой:

$$Update = (f^0 := f^0 + t.vstd)^* (v := t)^* \underset{v=s}{\{} (d := v.dad)^*$$

$$*(f(d,v) := f(d,v) + t.vstd)^* (f(v,d) := f(v,d) - t.vstd)^* (v := d)\}.$$

Исходя из первого критерия, дополнительные локальные переменные (q_i, v_i) и множества (Q_i, V_i) должны физически располагаться на тех вычислительных машинах, где выполняется процесс, с которым они ассоциированы, и все параллельные процессы вынуждены обмениваться данными с очередью *Q*. Таким образом, в каком бы месте ни находилась очередь *Q*, операции добавления к ней найденных вершин будут связаны с передачей данных медленными каналами связи. Предлагается два варианта размещения общей очереди.

1. Размещение на отдельном узле (рис. 1). В этом случае работа алгоритма приобретает особенности клиент–серверного приложения, поскольку всем звеньям, хранящим и обрабатывающим части графа, придется обращаться к узлу, содержащему очередь, чтобы поместить или извлечь вершину. При таком подходе каждая параллельная вычислительная часть (клиентская) обменивается данными только с одним звеном (серверным) и в случае добавления вершин в очередь, и при извлечении их из очереди.

2. Каждое звено содержит свою копию очереди (рис. 2). В этом случае операция помещения новых вершин в очередь будет связана с пересылкой одних и тех же данных на все узлы. Преимущество такого подхода заключается в том, что каждое звено содержит весь набор вершин, посещаемых в процессе поиска пути, и может просматривать их без необходимости обмена между узлами.

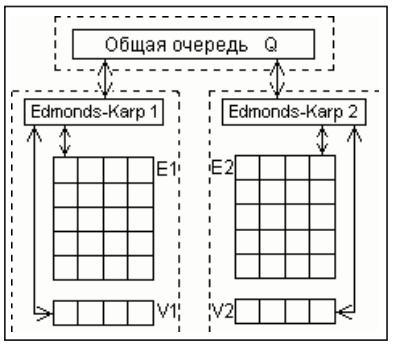

Рис. 1

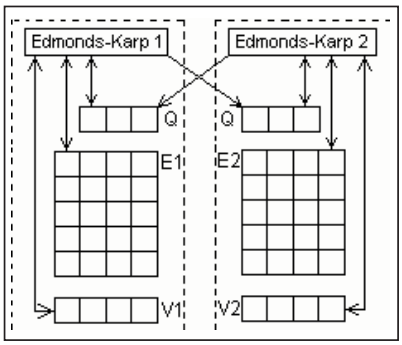

Рис. 2

Аналогична реализация и для первого варианта, если из основной очереди брать не по одной вершине, а сразу все вершины, не обработанные данным звеном. Детали операции будут зависеть от конкретной реализации алгоритма, языка программирования и возможностей системы. Независимо от размещения очереди согласно второму критерию существует необходимость минимизации трафика между процессами и количеством операций, заключенных в критические секции. Для этого предлагается модифицировать схему (3) так, чтобы критическая секция (важнейшей частью которой является добавление в общую очередь новых вершин) выполнялась лишь тогда, когда процессом пересмотрены все доступные вершины из общей очереди:

$$BFS\,3 = (Q := \{s\})^* (q_{1,2} := s)^* Reset\_visited2^* (\alpha_{1,2} := true)^*$$
$$^* \quad \{$$
$$\overline{\alpha}_1 \wedge \overline{\alpha}_2 \vee [t.vstd > 0]$$
$$S(\alpha_1)^* search_1\,2^* nextq_1{}^* [q_1 = \times](lock^* [Q_1 \neq \varnothing]$$
$$((Q := Q + Q_1)^* (\alpha_2 := true) \vee (\alpha_1 := false))^* unlock \vee \text{Œ})$$
$$\bullet$$
$$\vee$$
$$S(\alpha_2)^* search_2\,2^* nextq_2{}^* [q_2 = \times](lock^* [Q_2 \neq \varnothing]$$
$$((Q := Q + Q_2)^* (\alpha_1 := true) \vee (\alpha_2 := false))^* unlock \vee \text{Œ})$$
$$\}$$

Очевидно, что с математической точки зрения модификация сводится к вынесению за скобки операции выборки следующей вершины из общей очереди $next\,q_1$ ($next\,q_2$), что позволяет теперь не выполнять процедуру обмена данными, если процессу необходимо выполнять дальнейшие действия.

Анализируя рассмотренные подходы, можно заключить, что принципиально имеются все предпосылки увеличить количество параллельных звеньев алгоритма. Для этого введем переменную p, которая параметризует схему по количеству звеньев (в предыдущих схемах $p = 2$). Далее необходимо разбить данные на определенное количество частей, разделив цикл по v в схеме (2) на интервалы $v_i \in [\left\lceil \frac{n^* i}{p} \right\rceil, \left\lceil \frac{n^* (i+1)}{p} \right\rceil - 1]$ и обеспечить синхронизацию, оповещая о добавлении к общей очереди новых вершин. Прежде чем записать параметризованную схему, исключим громоздкие обозначения, перейдя к неинтерпретированной схеме:

$$A \overset{def}{=} (Q := \{s\})^* (\text{for } k := 0 \text{ to } p-1 \text{ do } q_k := s)^* Reset\_visited2$$
$$B \overset{def}{=} \text{for } k := 0 \text{ to } p-1 \text{ do } \alpha_k := true \quad \text{(оповестить все процессы)}$$
$$L \overset{def}{=} lock$$
$$U \overset{def}{=} unlock$$

$\mathrm{C}_i \overset{def}{=} Include(Q, Q_i)$

$\mathrm{D}_i \overset{def}{=} nextq_i$

$\mathrm{R}(\alpha_i) \overset{def}{=} (\alpha_i := false)$

$\beta \overset{def}{=} \prod_{k:=0}^{p-1} \overline{\alpha}_k$ (условие того, что путь не может быть найден)

$\tau \overset{def}{=} [t.vstd > 0]$

$\gamma_i \overset{def}{=} [Q_i \neq \varnothing]$

$\phi_i \overset{def}{=} [q_i = \times]$

$\mathrm{G}_i \overset{def}{=} [Q_i := \varnothing] * (v_i := \left\lceil \frac{n * i}{p} \right\rceil)$

$\chi_i \overset{def}{=} [v_i \geq \left\lceil \frac{n * (i+1)}{p} \right\rceil]$ (условие выхода за пределы подмножества V_i)

$\varphi_i \overset{def}{=} [(q_i, v_i) \in E_i] \wedge [v_i.vstd = 0] \wedge$ (условие того, что вершину v_i
$\wedge [c(q_i, v_i) > f(q_i, v_i)]$ нужно добавить в очередь)

$\mathrm{I}_i \overset{def}{=} Include(Q_i, v_i)$

$\mathrm{H}_i \overset{def}{=} inc(v_i)$

$\mathrm{P} \overset{def}{=} Update$

С учетом изложенного оптимизированную параметризованную регулярную САА–М–схему алгоритма Эдмондса–Карпа для p параллельных звеньев запишем в следующем виде:

$$
\begin{aligned}
&E-K-p=\{\\
&\quad \mathrm{A}*\mathrm{B}*\ \{\\
&\qquad {\scriptstyle \beta\vee\tau}\\
&\qquad\quad S(\alpha_0)*\mathrm{G}_0*\ \{[\varphi_0](\mathrm{I}_0\vee \text{Œ})*\mathrm{H}_0\}*\mathrm{D}_0*\\
&\qquad\qquad {\scriptstyle \chi_0}\\
&\qquad\quad *[\varphi_0](\mathrm{L}*[\gamma_0](\mathrm{C}_0*\mathrm{B}\vee \mathrm{R}(\alpha_0))*\mathrm{U}\vee \text{Œ})\\
&\qquad\qquad\qquad \overset{\bullet}{\vee}\ldots\overset{\bullet}{\vee}\\
&\qquad\quad S(\alpha_i)*\mathrm{G}_i*\ \{[\varphi_i](\mathrm{I}_i\vee \text{Œ})*\mathrm{H}_i\}*\mathrm{D}_i*\\
&\qquad\qquad\quad {\scriptstyle \chi_i}\\
&\qquad\quad [\varphi_i](\mathrm{L}*[\gamma_i](\mathrm{C}_i*\mathrm{B}\vee \mathrm{R}(\alpha_i))*\mathrm{U}\vee \text{Œ})\\
&\qquad\qquad\qquad \overset{\bullet}{\vee}\ldots\overset{\bullet}{\vee}\\
&\qquad\quad S(\alpha_{p-1})*\mathrm{G}_{p-1}*\ \{\ [\varphi_{p-1}](\mathrm{I}_{p-1}\vee \text{Œ})*\mathrm{H}_{p-1}\}*\mathrm{D}_{p-1}*\\
&\qquad\qquad\quad {\scriptstyle \chi_{p-1}}\\
&\qquad\quad *[\varphi_{p-1}](\mathrm{L}*[\gamma_{p-1}](\mathrm{C}_{p-1}*\mathrm{B}\vee \mathrm{R}(\alpha_{p-1}))*\mathrm{U}\vee \text{Œ})\\
&\qquad \}*\\
&\quad *\mathrm{P}\}\\
&\qquad \overline{\tau}
\end{aligned}
$$

В последней схеме учтены первых два критерия оптимизации. Очевидно, чтобы удовлетворить третий критерий, необходимо выполнить некоторое локальное распараллеливание на каждом узле, поскольку узел должен одновременно обмениваться данными с соседними узлами и выполнять определенные вычисления. Воз-

можности библиотеки MPI [7, 8] позволяют реализовать фоновый обмен сообщениями, во время которого исполнение основного кода не блокируется, что может быть использовано в качестве дополнительной оптимизации при моделировании схемы.

ЗАКЛЮЧЕНИЕ

В настоящей статье получены схемы алгоритма Эдмондса–Карпа, ориентированные на применение в распределенных параллельных архитектурах, использование которых становится все более актуальным, а иногда и бескомпромиссным. Благодаря применению САА-М в качестве инструмента формализации алгоритма становятся доступными мощные средства для генерации по формализованной регулярной схеме программного кода, моделирующего алгоритм средствами конкретного языка программирования (например, Си, Си++). Следует отметить, что полученные схемы были реализованы на языке программирования Си с использованием парадигмы MPI и промоделированы на различных кластерах. Результаты подтвердили эффективность подходов и предложенных критериев оптимизации при решении основной потоковой задачи большой размерности.

СПИСОК ЛИТЕРАТУРЫ

1. Edmonds J., Karp R.M. Theoretical improvements in algorithmic efficiency for network flow problems // J. Assoc. Comput. Mach. — 1972.— N 19. — P. 248–264.
2. Ford L.R., Fulkerson D.R. Maximal flow through a network // Canadian J. of Math. — 1956.— N 8. — P. 399–404.
3. Погорілий С.Д. Програмне конструювання. — К.: ВПЦ «Київський університет», 2005. — 440 с.
4. Сэджвик Р. Фундаментальные алгоритмы на С++. Ч. 5: Алгоритмы на графах. — СПб: ООО DiaSoft, 2002. — 496 с.
5. Ющенко Е.Л., Цейтлин Г.Е., Грицай В.П., Терзян Т.К. Многоуровневое структурное проектирование программ: Теоретические основы, инструментарий. — М.: Финансы и статистика, 1989. — 208 с.
6. Таненбаум Э., ван Стеен М. Распределенные системы. Принципы и парадигмы. — СПб: Питер, 2003. — 877 с.
7. Багачёв К.Ю. Основы параллельного программирования. — М.: БИНОМ. Лаборатория знаний, 2003. — 342 с.
8. http://www–unix.mcs.anl.gov/mpi
9. Pogorilyy S.D., Gusarov A.D. Paralleling of Edmonds–Karp net flow algorithm // Appl. Comput. Math. — 2006. — **5**, N 2 — P. 121–130.
10. Mekkittikul A., McKeown N. Scheduling VOQ switches under non-uniform traffic, CSL Technical report, CSL-TR-97-747, Stanford University, 1997.

Поступила 27.03.2008