УДК 537.322.2

С. Л. Бабичев, К. А. Коньков, А. К. Коньков

Московский физико-технический институт (государственный университет)

Применение сетей Петри для диагностирования проблем синхронизации в вычислительных системах с общей памятью

Рассматриваются аспекты, связанные с применением сетей Петри для обнаружения проблем синхронизации в параллельных программах, использующих общую память. Приводятся схемы преобразования основных примитивов синхронизации в модель сети Петри, рассматриваются механизм сифонов для определения активности сети Петри, а также сведение решения задачи об отсутствии тупиков в сети Петри в задачу математического программирования.

Ключевые слова: параллельное программирование, сети Петри, примитивы синхронизации, отсутствие тупиков, математическое программирование.

1. Сети Петри как инструмент анализа вычислительных систем с общей памятью

Классическое последовательное программирование уже не приводит к созданию программ, эффективно использующих ресурсы современных вычислительных систем. Многопоточные программы, работающие с данными в общей памяти, часто недостаточно эффективны, поскольку избыточно используют примитивы синхронизации. При недостаточном использовании примитивов синхронизации есть риск получить программу, имеющую в каком-то виде проблемы с синхронизацией. Средства динамического анализа параллельных программ не всегда могут решить данные проблемы. Для ряда программ требуется статический анализ параллельных алгоритмов. Наиболее сложной проблемой является диагностирование возможности тупиков.

В данной работе для диагностирования тупиков в алгоритмах применяется метод преобразования алгоритма в эквивалентную сеть Петри с последующим её анализом. В качестве введения в сети Петри могут рассматриваться классические работы [1] и [2].

Значительное число работ посвящено исследованию динамических свойств вычислительных систем с помощью аппарата сетей Петри.

В работе [3] исследуется применение сетей Петри к моделированию вычислительных систем с разделяемой памятью. Достаточно много внимания уделяется расширенным сетям Петри с добавлениями в виде запрещающих дуг, переходам с ненулевым временем, промежуточных дуг. Предлагается система, называемая SOM, основанная на языке ADA для моделирования и верификации параллельных процессов.

В работе [4] предлагается оригинальный математический аппарат для эффективного обнаружения отсутствия тупиков в сетях Петри. Используется механизм сифонов и ловушек и методы математического программирования.

Ряд работ посвящён моделированию многопоточных приложений с использованием сетей Петри. Это, например, работа [5], в которой рассматривается оптимизация многопоточных приложений на многопроцессорных вычислительных системах (используются сети Петри с ненулевым временем перехода).

В статье [6] рассматривается верификация параллельных программ с использованием сетей Петри как графа доказательств.

Работа [7] рассматривает подмножество сетей Петри — CT-сети (Casual–Time nets) — для моделирования автоматов параллельного исполнения и предлагает средства для построения параллельных приложений и контролируемым потоком исполнения.

Работа [8] использует механизм инвариантов модели сети Петри для определения следующих ошибок синхронизации в программах: тупиков, бесконечных внутренних циклов и потере сообщений при преждевременном завершении. В [9] также предлагается использовать механизм инвариантов, но исследование не продвигается достаточно далеко.

Работы [10] и [11] предлагают модели сетей Петри для основных примитивов синхронизации, но не предлагают методов доказательства отсутствия ошибок синхронизации в моделируемых программах.

Терминология сетей Петри

Для описания математической модели сети Петри применяются следующие обозначения.

Обычная сеть Петри $C = (P, T, F, \mu_0)$ где P есть множество позиций, T есть множество переходов, $F \subseteq (P \times T) \cup (T \times P)$ есть множество направленных дуг и $\mu_0 : P \to IN$ есть начальная маркировка, $IN \in \{0..\infty\}$. Множество входных (соответственно, выходных) переходов из позиции p обозначается как ${}^\star p$ (соответственно p). Подобно этому множество входных (соответственно выходных позиций для перехода t обозначается как ${}^\star t$ (соответственно $t^\star$. Для любого подмножества позиций S, ${}^\star S$ (соответственно $S^\star$ обозначает множество переходов, содержащих по крайней мере одну выходную (соответственно входную) позицию, принадлежащую S.

Переход t разрешён или запускаем в μ_0, если для всех $\rho \in^\star t,\ \mu_0(p) \geq 1$. Переход можно запустить, если он разрешён. Новая маркировка μ' производится удалением одного маркера из каждой входной позиции и помещением одного маркера в каждую выходную позицию перехода. Этот процесс обозначается как $\mu[t > \mu'$. Расширение запуска обозначается как $\mu[\sigma > \mu'$, где σ — последовательность переходов, переводящих μ в μ'. Множество всех маркировок, достижимых из μ_0, обозначается как $R(\mu_0)$.

Матрица инцидентности $C = [c_{ij}]$ такова, что $c_{ij} = 1$, если $t_j \in^\star p_i \setminus p_i{}^\star$; $c_{ij} = -1$, если $t_j \in p_j{}^\star \setminus^\star p_i$; $c_{ij} = 0$ в остальных случаях. Для любого M такого, что $M_0[\sigma > M, M = M_0 + C\overline{\sigma}$, где $\overline{\sigma}$ называется *вектором количества запусков* — таким вектором, в котором i-й элемент обозначает количество вхождений t_i в σ. Вектор неотрицательных целых $y \neq 0$ такой, что $C_y = 0$ называется *t-инвариантом*. t-инвариант y называется минимальным, если не существует *t-инвариант* x такой, что $x \leq y$. Подобно, вектор неотрицательных целых $x \neq 0$ такой, что $x^T C = 0$ называется *p-инвариантом*.

Сети Петри могут применяться во многих приложениях — для моделирования поведения сложных недетерминированных комплексов, для генерации машинного кода в компиляторах и для колоссального количества разнообразных задач. В данном случае сети Петри будут применяться для решения достаточно частной задачи — обнаружения наличия тупиков в многопоточной вычислительной среде с совместными ресурсами. Таким образом, требуется определение одной характеристики — активности (liveness).

Если переход в сети Петри никогда не может быть запущен, такое состояние называется *тупиком*. Обнаружение возможности наступления состояния тупика в сети Петри показывает наличие возможности состояния тупика и в моделируемой вычислительной среде.

Для анализа тупиков в сети C с маркировкой μ применяется следующая классификация:

• Активность уровня 0 имеет переход t_j, который никогда не может быть запущен

• Активность уровня 1 имеет переход t_j, который потенциально запустим, то есть существует такая допустимая и достижимая маркировка $\mu' \subset R(C, \mu)$, в которой данный переход разрешён.

• Активность уровня 2 имеет переход t_j, если для всякого целого n существует последовательность запусков t_j, в котором t_j присутствует по крайней мере n раз.

• Активность уровня 3 имеет переход t_j, если существует бесконечная последовательность запусков, в которой t_j присутствует неограниченно часто.

• Активность уровня 4 имеет переход t_j, если для всякой $\mu \in R(C, \mu)$, существует такая

последовательность запусков σ что t_j разрешён в $\delta(\mu',\sigma)$.

Наличие в сети переходов с активностью 0 с определённостью подтверждает наличие тупиков в моделируемой системе.

Сеть Петри не содержит тупиков, если для любой достижимой маркировки имеется по крайней мере один разрешённый переход. Позиция p ограничена, если существует такая константа K, что $\mu(p) <= K$ для всех $\mu \in R(\mu_0)$. Сеть Петри называется ограниченной, если все позиции сети ограничены.

Подмножество позиций S называется *сифоном (syphon)*, если любой входящий переход в S является также выходящим переходом в S, то есть ${}^\star S \subseteq S^\star$. Соответственно S есть *ловушка (trap)*, если $S^\star \subseteq^\star S$. Сифон (ловушка) называется минимальным, если он не содержит других сифонов (ловушек).

Существует несколько основных методов подхода к анализу активности сетей Петри — метод дерева достижимости, метод матричных уравнений и метод сифонов и ловушек. И метод дерева достижимости, и метод матричных уравнений могут показать необходимость, но не могут показать достаточность для доказательства наличия тупиков в модели. В данной работе используется преобразование уравнений сети Петри в уравнения для метода математического моделирования.

2. Моделирование многопоточной обработки и синхронизации с помощью сетей Петри

Процесс формирования корректного алгоритма параллельной обработки и верификации состоит из ряда шагов, которые могут быть описаны следующим образом:

1. Разработка модели алгоритма.
2. Перевод модели алгоритма на язык сетей Петри.
3. Верификация построенной модели Петри и возвращение к первому пункту для уточнения модели алгоритма.

Для перевода модели на язык сетей Петри требуется дать отображение основных конструкций входной модели на выходной язык. В настоящей работе применяются примитивы синхронизации двух типов: *Mutex* и *Event*.

Примитив типа Mutex

Примитив типа *Mutex* представляет средство взаимоисключения вычислительных потоков имеет следующие методы:

- *WaitAndLock* — ожидать освобождения с немедленным захватом
- *Release* — освобождение.

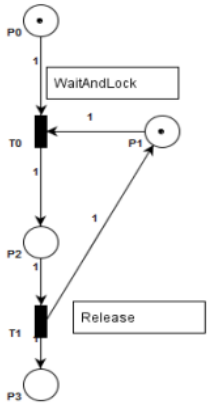

Рис. 1. Схема сети Петри примитива Mutex

Mutex имеет атрибут владельца, то есть, свойство принадлежности вычислительному потоку, который его захватил. Этот примитив требует поддержки операционной системой, может привести к контекстному переключению потоков и не переводит вычислительное ядро в состояние активного ожидания.

Примитив *Mutex* можно представить в виде модели сети Петри — см. рис.1.

Наличие в позиции **P1** маркера показывает, что *Mutex* находится в свободном состоянии. Эта позиция должна использоваться совместно несколькими вычислительными потоками. При появлении маркера в позиции **P0** и наличии маркера в позиции **P1** становится возможным переход **T0**, маркер в позиции **P1** теперь не присутствует, что делает невозможным осуществление переходов, зависящих от наличия маркера в позиции **P1**.

После захвата маркера вычислительный поток исполняет код критической секции (позиция **P2**), исполняется переход **T1**, который передаёт маркер в позицию **P1**, делая тем самым возможным осуществление связанных с наличием маркера в этой позиции переходов.

Примитив типа Event

Примитив типа *Event* используется для установления факта возникновения какого-либо события, может находиться в двух состояниях — сигнальном и несигнальном и имеет следующие методы:

- *Set* — установить объект в сигнальное состояние;
- *Reset* — установить объект в несигнальное состояние;
- *Wait* — ожидать наступления сигнального состояния.

В отличие от объектов типа *Mutex* объекты типа *Event* не имеют атрибута владельца. В настоящей работе рассматривается вариант объектов типа *Event*, требующий ручного перевода состояния в несигнальное после того, как ровно один из вычислительных потоков получил доступ к нему. Примитив *Event* наряду с примитивом *Mutex* требует поддержки операционной системой, может привести к переключению контекста вычислительных потоков и также не переводит вычислительное ядро в состояние активного ожидания.

В данной работе учитывается следующий факт: если примитив *Event* находится в сигнальном состоянии, то последующие вызовы методов *Set* не оказывают на это состояние влияния. Если примитив *Event* находится в несигнальном состоянии, то последующие вызовы метода *Reset* не оказывают на это состояние влияния. Данные условия приводят к более точной и более сложной модели, чем в работах [10] и [11].

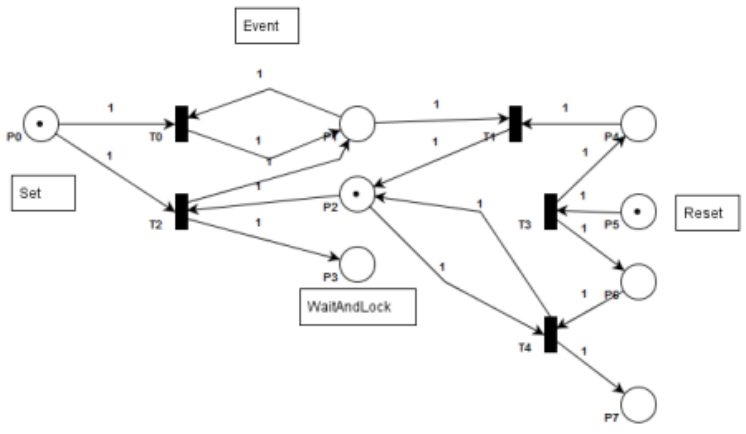

Рис. 2. Схема сети Петри примитива Event

На рис. 2 изображена сеть Петри для примитива *Event*.

Позиция **P0** представляет собой вход метода *Set*. Если примитив *Event* уже находится в состоянии сигнал (что определяется наличием маркера в позиции **P1**), то операция помещения маркера в позицию **P0** будет эквивалентна пустой операции. Если же данный примитив не находится в состоянии сигнал, то операция помещения маркера в позицию **P0** приводит примитив в состояние сигнал.

Позиция **P5** представляет собой вход метода *Reset*. Помещение маркера в данную позицию переводит примитив *Event* в состояние сигнал не установлен в независимости от первоначального состояния примитива.

Выходом метода *Reset* является позиция **P7** — появление маркера в данной позиции сигнализирует о завершении работы метода.

Если в позиции **P3** уже имеется маркер, то метод *Wait* завершается успехом.

Исследование свойств совокупной сети Петри

Совокупная сеть Петри вычислительной модели может формироваться как вручную, так и автоматизированным способом. Автоматизированный способ предполагает либо полный анализ текста на исходном языке программирования, что возможно, но технически трудоёмко, либо синтез сети Петри по описанию на упрощённом языке, содержащем только конструкции, связанные с управлением вычислительными потоками и механизмами синхронизации. Синтаксический и семантический анализ подобного языка можно реализовать с помощью общедоступных средств построения компиляторов — *bison*, *yacc*, *flex* и прочее.

После синтеза совокупной сети Петри модели требуется определить, возможны ли в ней тупики. Имеются три основных подхода к выявлению данного свойства — построение дерева достижимости, метод матричных уравнений и метод сифонов и ловушек.

В данной работе применяется метод сифонов и ловушек.

Введём две функции от сифона S:

$f(S) = \min\{(S)|\mu \in R(\mu_0)\}$,

где $\mu(S)$ есть общее количество маркеров в S, то есть $\mu(S) = \sum\limits_{p \in S} \mu(p)$.

$F(S) = \min\{\mu(S)|\mu = \mu + CY, \mu \geq 0, Y \geq 0\}$,

где C есть матрица инцидентности сети Петри, μ — маркировка сети, и есть вектор.

Отношение $\mu = \mu_0 + CY$ является *уравнением состояния сети*. Из базовой теории сетей Петри следует, что любая достижимая маркировка μ удовлетворяет уравнению состояния, но обратное неверно.

Сифон S есть потенциальный тупик тогда и только тогда, когда $f(S) = 0$.

Это подразумевает, что $F(S) \geq f(s)$. Следовательно, любой сифон S такой, что $F(S) > 0$ не является потенциальным тупиком. Таким образом, сеть Петри не содержит тупиков, если каждый сифон S данной сети либо содержит маркированную ловушку, либо $F(S) > 0$.

Для проверки отсутствия тупиков сети Петри с использованием данного свойства проблема должна быть решена для каждого минимального сифона, не содержащего маркированной ловушки. Этого можно избежать, преобразуя эти проблемы в эквивалентную задачу квадратичного программирования — как показывает следующая теорема:

Пусть $\{S_1, S_2..., S_n\}$ есть множество минимальных сифонов, которые не содержат маркированных ловушек. Сеть Петри не содержит тупиков, если $F > 0$,

где

$$F = \min\{\sum_{i=1}^{n} z_i \mu(s_i) | \mu = \mu_0 + CY,\ \sum_{i=1}^{n} z_i = 1,\ \mu \geq 0,\ Y \geq 0,\ z_i \geq 0\}.$$

Во избежание явного перечисления предпочтительнее использовать выражение $F(S)$, определённое следующим образом:

$F(S) = \min\{\mu(S) | \mu = \mu_0 + CY,\ \mu \geq 0,\ Y \geq 0\}$,

где C есть матрица инцидентности сети Петри, а μ и Y есть вектора чисел. Выражение $\mu = \mu_0 + CY$ суть уравнение состояния сети. Любой сифон S такой, что $F(S) > 0$, не является потенциальным тупиком.

Данная проблема — задача линейного программирования и может быть разрешена за полиномиальное время. Использование линейного программирования требует проверки всех минимальных сифонов, и эффективность данного метода зависит от их количества. Хорошо известно, что общее количество минимальных сифонов (ловушек) быстро растёт за пределы практического применения и что, в худшем случае, этот рост экспоненциальный.

В настоящем случае вместо использования полиномиальных по вычислительной сложности алгоритмов решалась задача математического программирования:

$G^{MP} = Minimize \sum\limits_{p \in P} \nu_p$,

где

T — множество переходов сети Петри,

$F \subseteq (P \times T) \bigcup (T \times P)$ есть множество направленных дуг,

$\nu_p = 1\{p \notin S\}$,

$z_t = 1\{t \notin S^\star\}$,

$z_t \geq \sum\limits_{p \in {}^\star t} \nu_p - |{}^\star t| + 1, \forall t \in T$,

$\nu_p \geq z_t,\ \forall (t, p) \in F$,

$\nu_p, z_t \in \{0, 1\}$,

$\mu = \mu_0 + CY$,

$\mu \geq 0$,

$Y \geq 0$.

Сеть Петри не содержит тупиков, если $G^{MP} = |P|$.

Методы решения задач математического программирования изложены в книге [12].

Практическое применение.

```
shared mutex DataLockMutex;
shared event DataReadyEvent;
shared event ResultReadyEvent;

thread MainThread {
  forever {
    DataLockMutex.Wait;
    DataReadyEvent.Set;
    @work;
    ResultReadyEvent.WaitAndLock;
    DataLockMutex.Release;
    ResultReadyEvemt.Reset;
  }
}

thread PoolThread {
  forever {
    DataReadyEvent.WaitAndLock;
    @work;
    DataReadyEvent.Reset;
    ResultReadyEvent.Set;
  }
}
```

Рис. 3. Задача на языке описания модели

Данный метод был применен для определения отсутствия тупиков в модели статического пула вычислительных потоков, применяемого для реализации модифицированной защищённой среды [13]. По алгоритму, выраженному на внутреннем

языке описания модели (рис. 3), была синтезирована сеть Петри, изображенная на рис. 4, по которой была сформирована задача математического программирования. Решение этой задачи показало, что условия отсутствия тупиков удовлетворяются и, следовательно, моделируемая сеть Петри не содержит тупиков.

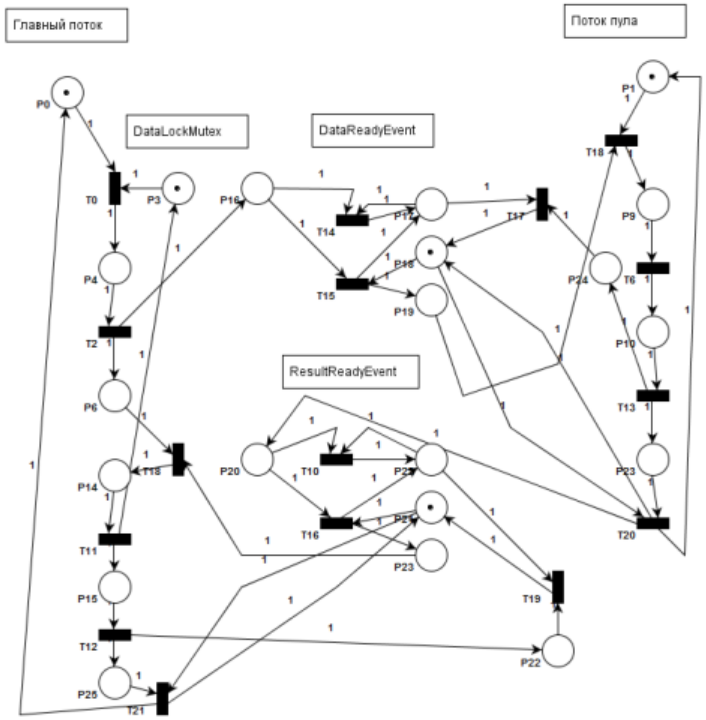

Рис. 4. Синтезированная сеть Петри

Выводы

В данной работе решалась задача определения применимости алгоритма по реализации пула вычислительных потоков с использованием механизма сетей Петри. Было впервые применена уточнённая модель сети Петри примитива синхронизации Event, учитывающая современную семантику данного примитива. По языку описания модели была синтезирована эквивалентная сеть Петри. Анализ эквивалентной сети Петри методом математического программирования позволил установить, что данная сеть, а следовательно и оригинальный алгоритм не содержит тупиков.

Весьма интересной и перспективной задачей является автоматизация построения сетей Петри по заданным спецификациям вычислительных потоков и расширение числа моделей синхронизирующих примитивов.

Второй интересной задачей может быть автоматическое построение наиболее эффективной модели многопоточных вычислений с использованием наименьшего количества примитивов синхронизации.

Литература

1. *Питерсон Дж.* Теория сетей Петри и моделирование систем. — М.: Мир, 1984. — 264 с.

2. *Murata Tadao* Petri Nets: Properties, Analysis and Applications // Proceedings of the IEEE. — 1989. — V. 77, N 4.

3. *Vallejo F., Gregorio J.A., Gonzalez Harbour M., Drake J.M.* Shared Memory Multiprocessor operating System with an Extended Petri Net Model // IEEE transactions on parallel and distributing systems. — 1994. — V. 5, N 7, July.

4. *Feng Chu and Xiao-lan Xie* Deadlock Analysis of Petri Nets Using Siphons and Mathematical Programming, // IEEE Transactions of Robotics and Automation. 1997. — V. 13, N. 6, December.

5. *Govindarajan F., Suciu W.M., Zuberek* Timed Petri NetModels of Multithreaded Multiprocessor Architectures // IEEE Preceedings if the 7-th International Workshop on Petri Nets and Performance Models. — Saint Malo, June, 1997.

6. *Takaoka Tadao* A Systematic Approach to Parallel Verification // Department of Computer Science of Ibaraki University, August, 1995.

7. *Pommereua F.* Petri Nets as Executable Specifications of High-Level Timed Parallel Systems. — 2005.

8. *Bruce P. Lester*Detection of Control Flow Errors in parallel Programs at Compile Time // International Journal of Distributed and parallel Systems (IJDPS). — 2010. — V. 1, N 2, November.

9. *Padidar S.* Parallel Program verification: A Brief Introduction, January, 2010.

10. *Kavi K.M., Moshtaghi A., Deng-Jyi Chen* Modeling Multithreaded application using Petri nets, // International Journal of Parallel Programming. — 2002. — V. 30, Iss 5. — P. 1–23, October.

11. *Kavi K.M., Bukhles P.B., Bhat U.N.* Isomorphism Between Petri net and Dataflow Graphs // IEEE Transactions on Software Engineering. — 1987. — V. SE-13, N 10.

12. *Minoux M.* Programmation Mathematique: Theorie and Algorithms. — Dunod, Paris, France, 1983.

13. *Бабичев С.Л., Коньков А.К., Коньков К.А.* Дополнительная защита ресурсов операционной системы методом криптографической защиты данных // Сб. науч. трудов. Моделирование процессов обработки информации. — М.: МФТИ, 2007. — С. 251–259.

Поступила в редакцию 24.11.2011.