УДК 004.42::[519.876.2:519.218.82]::004.272.2

Метаслой программы: объектно-событийное моделирование алгоритма, данных и функциональных характеристик. Концепция. Применение в параллельном программировании

Пекунов В.В., канд. тех. наук

Рассматриваются проблемы моделирования и предикции данных, состояния алгоритма и времени исполнения программы. Обосновано применение формализма объектно-событийных моделей в задачах построения и интерпретации предикционных моделей. Формализм расширен вводом понятия вложенных объектно-событийных моделей. Предложена концепция абстрагированного от программы метаслоя, осуществляющего моделирование. Дана парадигма применения предикционных моделей для динамической оптимизации и приближения/восстановления алгоритмов как обычных, так и параллельных программ.

Ключевые слова: метаслой программы, моделирование алгоритма, прогнозирование данных, прогнозирование времени исполнения, объектно-событийная модель, параллельное программирование, оптимизация программы.

Program Metalayer: Object-Event Simulation of Algorithm, Data and Parameters of Program. Key Idea. Application in Parallel Programming

V. V. Pekunov, Candidate of Engineering

The problems of simulation and of prediction of data, algorithm and execution time of program are considered. It is proposed to apply a formalism of object-event models (OEM) for the constructing and interpretation of predictive models. The nested OEMs are proposed to include in this formalism. The idea of modelling metalayer abstracted from the program is introduced. The ways of using predictive models for the dynamic optimization and approximation/copying of algorithms in cases of usual and parallel programs are given.

Keywords: program metalayer, algorithm simulation, data prediction, prediction of execution time, object-event model, parallel programming, program optimization.

Введение. Существует ряд нетривиальных задач, эффективное решение которых требует разработки специфических программ, нуждающихся в предикции времени исполнения тех или иных своих фрагментов, их внутреннего состояния или формируемых ими данных.

Предикция времени исполнения путем построения профилирующей модели необходима при решении ряда следующих задач, связанных с *параллельной обработкой данных*, особенно при работе в гетерогенной вычислительной среде: определения потребности в распараллеливании и его потенциальной эффективности; выбора схем распараллеливания [1]; оптимальной балансировки загрузки процессоров. Последняя проблема характерна для решений, связанных с применением гранулярного параллелизма («портфель задач»), характерного для T-систем, MC#, процедур с повторным входом [2], OpenMP 3.

Предикция внутреннего состояния путем построения модели хода исполнения может потребоваться при *выборе оптимального алгоритма*, предполагающем косвенную оценку времени исполнения через предикцию значений переменных состояния, характеризующих исполнение алгоритма. Если время исполнения является сложно определяемой функциональной зависимостью от переменных состояния, то такая косвенная оценка может быть достаточной и менее трудозатратной по сравнению с прямой. Рассмотрим задачи поиска/сортировки информации в сложно структурированных данных. Определяя глубину рекурсии при сортировке в дереве по характерной выборке, можно оценить ее эффективность для всего массива данных и при необходимости выбрать иной алгоритм. Аналогично, например, по соотношению элементов выборки в упорядоченных последовательностях (активно использующихся в теории и практике разработки баз данных) может выбираться вариант поиска: бинарный или интерполяцией.

Интересным вариантом применения такой предикции является *интеллектуализация разработки и отладки*, а именно – сверка получаемых графов переходов алгоритма с целевым, определяемым изначально или в процессе построения программы. Во втором случае имеем инкрементальный процесс: а) по исходной программе генерируется граф переходов, проверяемый программистом на корректность и достраиваемый им же до целевого; б) соответствующим образом модифицируется программа, ее запуск дает новый граф; в) полученный граф сверяется с текущим целевым; г) модифицируется или программа, или целевой граф. В данном контексте отметим работу [3], в которой отмечено применение схожего подхода (динамический анализ кода по трассе исполнения с обратной инженерией, т. е. с построением фактически полной модели программы) в целях оптимизации исходной программы.

Предикция формируемых данных может использоваться при *параллельном решении ряда задач математической физики*. Возможно

исключение части пересылок путем предикции (экстраполяционного или интерполяционного приближения) недостающих данных [3]. Также можно говорить о *копировании или приближении алгоритмов обработки данных без копирования реализующего их кода*, что представляет определенный интерес, например, если имеется ряд программных модулей, написанных на иных языках, взаимодействие с которыми затратно по времени. Тогда можно построить аппроксимирующую реальный алгоритм модель обработки данных, полностью или частично копирующую логику исходного алгоритма, которая и интегрируется в систему.

Предикция времени исполнения, внутреннего состояния и/или данных должна быть внешним по отношению к основной программе алгоритмом, работающим в параллельном фоновом процессе и никоим образом не увеличивающим время ее исполнения. Логичным решением является построение соответствующей системы фоновой предикции, функционирующей на отдельном ядре или группе ядер центрального процессора (ЦП) или графического процессора (ГП), максимально независимой от основной программы и взаимодействующей с ней посредством специфического программного интерфейса. Вышеуказанная постановка задачи предикции является достаточно новой, учитывая ее решение для целей самой программы. Задача сводится к группе подзадач построения математических (в том числе, бионических – нейросетевых или эволюционных) моделей данных в сочетании с вероятностными и/или логическими моделями алгоритмов с последующей интерпретацией моделей. Осуществляется динамический анализ потоков данных и трассы исполнения [3]. Подзадачи каждой группы однородны и подразумевают обработку регулярных, преимущественно матричных и векторных данных. Поэтому особый интерес представляет возможность применения ГП видеокарт класса nVidia GeForce с поддержкой технологии CUDA, содержащих матрицы однородных SIMT-процессоров, обладающих высокой производительностью обработки такого рода данных.

Введем понятие *метаслоя* (скрытого слоя, концептуально несколько схожего с пространством моделирования ECO [5]) основной программы, собирающего сведения о ходе ее исполнения и значениях контролируемых переменных при прохождении через заданные контрольные точки, осуществляющего построение и интерпретацию предикционных моделей алгоритма/данных в фоновом режиме и представляющего результаты предикции основной программе по запросу. Принятие решений о том или ином варианте исполнения осуществляется основной программой, хотя интересным представляется и вариант, при котором решение может приниматься непосредственно в метаслое, который в таком случае должен включать соответствующую систему дедуктивного вывода и/или продукционную систему.

Построение предикционных моделей может состоять в индукции алгоритма «входы → переменные состояния» и/или индукции функциональной модели, записанной в терминах «входы → выходы» или «входы + переменные состояния → выходы». Второй вариант для многих случаев представляется более перспективным, поскольку позволяет упростить как функциональную модель, так и процесс ее построения. Упрощение объясняется тем, что после корреляционного анализа в пространстве признаков с наибольшей вероятностью остаются преимущественно переменные состояния, обычно имеющие более ярко выраженную корреляцию с выходами. Поскольку такие переменные уже являются функциями входов и иных переменных состояния, их ввод в функциональную модель позволяет сократить анализ и исключить соответствующие фрагменты комплекса зависимостей «входы → переменные состояния».

Модели могут исполняться непосредственно или, для снижения времени отклика метаслоя, путем предварительной компиляции в код. Для их представления целесообразен выбор такого формализма, в рамках которого были бы строго определены не только операции логического синтеза модели, адекватно отражающей структуру и алгоритм основной программы, но и операции ее интерпретации (параметризацию и, возможно, непосредственное исполнение в целях предикции) и трансформации в шаблонный код, решающий задачу предикции. Очевидно, что это должен быть некий класс дедуктивно-индуктивных синтезирующих моделей. Указанным требованиям в полной мере отвечает формализм объектно-событийных моделей (ОСМ) [3, 6].

Представление модели алгоритма и данных. Входными данными при синтезе модели являются: трасса исполнения (список узлов-контрольных точек в порядке их прохождения); текущие значения контролируемых переменных программы и времени прохождения для каждого узла. Предполагается, что синтез структуры модели алгоритма (фактически – графа переходов) осуществляется путем логического анализа трассы исполнения (может использоваться машина дедуктивного вывода), содержащей перечень всех совершенных переходов. Как и при построении функциональных моделей, целесообразно искать закономерности переходов не только в значениях входных контролируемых переменных, по имеющемуся набору которых не всегда можно однозначно определить логику перехода, но и в значениях внутренних переменных состояния. Это позволит упростить и модель алгоритма.

Внутренними переменными могут быть счетчики активизации узлов, которые рассчитываются путем прохождения по узлам модели в порядке, определяемом трассой исполнения. Значения времени и контролируемых переменных присоединяются к значениям внутренних переменных, таким образом получаем журнал исполнения, анализ которого позволяет осуществить параметризацию моделей алгоритма и данных.

Как уже отмечалось выше, для представления таких моделей удобно использовать формализм ОСМ с классической двухслойной схемой интерпретации [3]: а) последовательным достраиванием и трансформацией модели в решающем слое с применением аппарата логического анализа и синтеза; б) исполнением (непосредственным или через компиляцию в программный код). На первом этапе модель может представлять собой единственный дедуктивный объект-анализатор, относящийся к решающему классу, принимающий на вход данные трассы исполнения и синтезирующий по ней индукционно-аналитическую ОСМ – граф переходов. На втором уровне данная ОСМ исполняется: проводит индукцию правил перехода и трансформации данных, после чего переходит (планируется новое событие) к предикции в режиме прямого исполнения правил или через генерацию исполняющего кода. Режим генерации кода является исходным режимом работы ОСМ и детально описан в работе [3] для блок-схемы, имеющей взаимно однозначное соответствие графу переходов. Режим прямой интерпретации может быть реализован по описанной ниже элементарной событийной схеме.

Структура ОСМ алгоритма и данных. ОСМ представляется графом (P, E), где P – множество объектов (узлов-контрольных точек) различных классов; E – множество дуг, представляющих направления возможных переходов. Дуги, замыкающие цикл, целесообразно представить информационными связями (без прямой активизации узла, инцидентного дуге по входу), прочие дуги – основными. Объекты (узлы) модели относятся к одному из классов, входящих в двухуровневую иерархию, родительским элементом которой является базовый класс S (узлы-инкрементеры без предикции), его наследники: P_t – узел с предикцией времени; P_d – узел с предикцией данных; P_{td} – узел с предикцией времени и данных.

В терминах ОСМ любой класс из множества классов модели C = {S, P_t, P_d, P_{td}} описывается пятеркой (N_c, I, O, F, M), где N_c – идентификатор (имя) класса; I и O – наборы входных и выходных контактов; F = F(N_c) – поля; M = M(N_c) – методы. Объекту любого из этих классов достаточно иметь один активизационный вход и один активирующий выход.

В ОСМ любой контакт является шестеркой (N_t, T, L, Min, Max, D), где N_t – идентификатор контакта; T – тип контакта (входной, выходной, анонимный); L – множество пар (класс, контакт), определяющих контакты классов и их наследников, к которым можно провести выходную связь; Min – минимальная степень контакта; Max – максимальная степень; D – ассоциативный кортеж, отражающий состояние контакта. Таким образом,

$$\forall c \in C : I = \{I_0\} = \{("вход", входной, \varnothing, 0, \infty, D)\};$$

$$\forall c \in C : O = \{O_0\};$$

$$O_0 = ("выход", выходной, \{(S, I_0)\}, 0, \infty, D).$$

Входные связи множеством L не специфицируются, поскольку легко выводятся из множества выходных связей. Минимальное множество полей, позволяющих описать семантику модели, имеет вид

F(S) = {ID, counter, R_{trans}, B_{trans}},

где ID – идентификатор текущего объекта (совпадающего с идентификатором узла-контрольной точки); counter – поле-счетчик, используемое исключительно в режиме исполнения модели (счетчик инкрементируется при каждом входе в узел, т. е. при активизации соответствующего объекта); R_{trans} = R_{trans}[i] – ассоциативный кортеж *взаимоисключающих* правил перехода (предикатов) по исходящим дугам; B_{trans} = B_{trans}[i] – ассоциативный кортеж булевых значений, определяющих наличие/отсутствие инициализации счетчика узла, инцидентного дуге с номером i по входу. Ее наличие определяет существование цикла, начало которого обозначено данным узлом. Если R_{trans}[i] = ∅, то дуга с номером i *отрицательная*, т. е. переход по ней осуществляется, если не выполняется ни одно из правил для прочих дуг (отрицательными, по определению, являются единственные безусловные дуги). Таковой целесообразно сделать одну дугу из числа инцидентных узлу по выходу, для которой существуют наиболее трудозатратные в расчете предикаты перехода.

С явным учетом наследования получим

$$F(P_t) = F(S) \cup \{T_{trans}\};$$

$$F(P_d) = F(S) \cup \{D_{trans}\};$$

$$F(P_{td}) = F(P_t) \cup F(P_d),$$

где T_{trans} = T_{trans}[i] и D_{trans} = D_{trans}[i] – ассоциативные кортежи моделей (правил) расчета времени и трансформации данных при переходах по исходящим дугам.

Анализ наличия процедур. Вложенные ОСМ. Анализ наличия процедур в графе переходов может быть произведен в решающем слое. Процедурой считается сетевой подграф модели, если на ведущих в него и исходящих из него дугах наборы значений переменных попарно эквивалентны, причем учитываются только те переменные, которые не модифицируются в указанном подграфе. При этом делается допущение об отсутствии глобальных переменных, изменяемых в процедуре. Сетевой

подграф, соответствующий процедуре, целесообразно описать *вложенной* (в основную) ОСМ со свои календарем событий. В общей ОСМ ссылки на такой фрагмент представляются K узлами, вызывающими вложенную ОСМ, где K – количество путей вызова процедуры. Контейнером для вложенной ОСМ является специальный *интерфейсный* узел (N_c, I, O, F, M), не имеющий методов, элемент M представляет собой вложенную ОСМ. Вводится понятие *анонимного контакта* ∅, всегда определяемого шестеркой (∅, «анонимный», L, 0, ∞, D), который существует у любого узла. Анонимный контакт интерфейсного узла допускает входную информационную связь от анонимного контакта обычного узла, не выполняющую иных функций кроме разрешения вызова вложенной ОСМ. Для такого контакта в случае интерфейсного узла L = ∅, а в случае обычного узла

$$L = \left\{(p,\varnothing)\middle| p \in P_I, P_I \subset P\right\},$$

где P_I – множество интерфейсных узлов.

Вызывающий узел помещает в кортеж D анонимного контакта данные, которые копируются в кортеж D анонимного контакта интерфейсного узла, выполняющий функции общего кортежа данных «почтовый ящик» во вложенной ОСМ. Далее вызывающий узел запускает вложенную ОСМ и, по окончании ее работы, забирает по информационной связи результаты из ее «почтового ящика» в кортеж D.

Параметризация правил модели. Любое из правил расчета времени и трансформации данных, сопоставленных переходам, является тройкой (V_{inp}, out, f(X)), где V_{inp} – вектор идентификаторов входных переменных, являющихся аргументами расчетной функции f(X), причем компоненты векторов V_{inp} и X имеют прямое позиционное соответствие; out – идентификатор выходной переменной.

В настоящее время f(X) являются полиномиальными зависимостями от одной или многих переменных, построенными соответственно с применением линейной (в рамках критерия МНК) или нелинейной регрессии в соответствии с подходом МГУА [7]. Отбор переменных-аргументов реализуется классическим корреляционным анализом. Выбор МГУА обусловлен требованием оперативности получения модели: методы данного класса имеют повышенную, по сравнению, например, с нейронными сетями, скорость сходимости строящейся блочно-функциональной модели к решению. Это компенсирует повышенную сложность модели, характерную для инкрементально-блочных подходов.

Правила (предикаты) перехода находятся путем комплексного логического анализа журнала исполнения A_{pv}^{ks} – матриц значений переменных $v \in V$ при p-м по счету переходе программы из точки k в точку s, причем

$$k,s \in K;\ p = \overline{1,M_{ks}};$$

$$V = \{time\} \cup R;\ R = Z \cup K,$$

где K – множество идентификаторов узлов-контрольных точек, совпадающих с идентификаторами соответствующих внутренних переменных; M_{ks} – количество наличествующих в трассе случаев прохождения из k в s; time – идентификатор переменной времени; Z – множество идентификаторов контролируемых входных переменных.

Пусть из рассматриваемого узла f возможны переходы в N_F узлов, идентификаторы которых содержаться в векторе F. Правило перехода $rule_{fg}$ в g-й узел ($g \in F$) может быть определено с помощью простых алгоритмов поиска классификационных правил по типу алгоритма Кора [7] настолько, насколько это возможно в текущем контексте, т. е. при имеющемся наборе классифицирующих признаков R. На каждом этапе поиска определяется переменная b с наибольшей дискриминативностью d_j, т. е. такая, которой соответствует: а) наибольшее количество строк матрицы A_{pb}^{fg}, у которых значения в столбце b отличаются от значений в том же столбце матриц A_{pb}^{fs}, $s \neq g$; б) наибольшее количество разных значений такого рода. Соответственно,

$$b = \arg\left(\max_j\left(d_j\right)\right);\ d_j = \alpha P_j^{fg} + \beta \operatorname{card}\left(G_j^{fg} / U_j^{fg}\right);$$

$$U_j^{fg} = \bigcup_{s \neq g}\left\{A_{pj}^{fs}\middle| p = \overline{1,M_{fs}}\right\};\ G_j^{fg} = \left\{A_{pj}^{fg}\middle| p = \overline{1,M_{fg}}\right\};$$

$$P_j^{fg} = \sum_{s \neq g} \operatorname{card}\left\{p \in \overline{1,M_{fs}} \middle| A_{pj}^{fs} \in \left(G_j^{fg} / U_j^{fg}\right)\right\},$$

где α и β – коэффициенты настройки, которые целесообразно выбирать из условия $\beta > \alpha$. При ненулевой величине $\max(d_j)$ из матрицы A_{pb}^{fg} исключаются строки, в которых переменная b имеет любое из дискриминирующих значений из множества G_j^{fg} / U_j^{fg}. В правило вводится соответствующее условие check(b), покрывающее данные значения, представляющее собой либо единственный шаблонный предикат $cond_0(b)$, либо дизъюнкцию нескольких таких предикатов $cond_i(b)$, таких что

$$\bigcup_i T_i \equiv G_j^{fg} / U_j^{fg};\quad T_i = \left\{t \in U_j^{fg} \cup G_j^{fg} \middle| \operatorname{cond}_i(t)\right\};$$

$$\bigcup_i F_i \equiv U_j^{fg};\quad F_i = \left\{f \in U_j^{fg} \cup G_j^{fg} \middle| \neg \operatorname{cond}_i(f)\right\},$$

причем в качестве $cond_i(b)$ рассматриваются *шаблонные выражения* для проверки отдельных значений и значений, входящих в ряды с арифметической или геометрической прогрессией, наиболее часто встречающиеся в классических алгоритмах.

В случае нулевой величины $\max(d_j)$ в конъюнктивное правило перехода в g-й узел вводится остаточное вероятностное условие

$$check(b) = \sum_{s=1}^{g-1} M_{fs} \Big/ \sum_{s=1}^{N_F} M_{fs} \;\; \xi_1 \;\; q \;\; \xi_2 \;\; \sum_{s=1}^{g} M_{fs} \Big/ \sum_{s=1}^{N_F} M_{fs},$$

где ξ_1, ξ_2 – знаки отношения строгого или нестрогого следования; q – равномерно распределенная случайная величина в диапазоне [0; 1]. При этом из всех матриц A_{pv}^{fg} исключаются все оставшиеся в них строки. Таким образом, модель алгоритма принимает либо вероятностно-логическую, либо чисто вероятностную форму.

Процедура поиска условий повторяется, пока хотя бы одна из матриц A_{pv}^{fg} не пуста. Результирующее правило перехода $rule_{fg}$ определяется дизъюнкцией всех найденных для данного перехода условий check(b).

Сделаем несколько замечаний, касающихся ввода такой разметки исходной программы (выбора мест расстановки и классов контрольных точек), которая позволила бы получить наиболее правдоподобную предикцию. Поскольку модели расчета времени строятся для переходов между узлами, то очевидно, что необходимо ввести маркировку (класс P_t) как минимум для входной и выходной точек алгоритма. Кроме того, рекомендуется маркировать точки (классы S, P_t), факт прохождения через которые прямо или косвенно влияет на передачу управления в иных точках, это часто позволяет повысить адекватность правил перехода. Если предполагается использование построенных моделей при различных значениях входных переменных, то необходимо исполнить исходный алгоритм для нескольких наиболее характерных комбинаций таких переменных.

Исполнение ОСМ. Как прямое, так и опосредованное компиляцией исполнение объектно-событийной модели алгоритма и данных осуществляется через ее интерпретацию, выполняемую по обычной двухуровневой схеме [3]. Режим интерпретации (прямое исполнение или опосредованное компиляцией) модели определяется специализацией классов или параметрами входящих в нее объектов.

Рассмотрим *режим прямого исполнения*, который подразумевает последовательную передачу управления от одного узла к другому. Для наиболее эффективного исполнения целесообразно передавать управление в ходе обработки одного и того же события, если же это невозможно (при замыкании цикла), – планировать следующие события исполнения.

Пусть в общем кортеже данных «почтовый ящик» MAIL[имя_ячейки] для текущего события с идентификатором $EV_{тек}$ хранятся значения $ID_{акт}$ – идентификатор исполняемого (получающего управление) при наступлении данного события объекта-узла и ссылка на кортеж VARS текущих значений всех переменных модели, т. е.

$MAIL[EV_{тек}] = (ID_{акт}, VARS)$,

причем имеет место биекция

$var\_refs : VARS \leftrightarrow V$.

В начале интерпретации (при обработке системного события «инициализация») в ячейку $MAIL[EV_{вызов}]$ помещаются: идентификатор ID_0, соответствующий узлу, исполняемому первым; ссылка на кортеж переменных VARS, в который заносятся входные значения внешних переменных, нулевое время и начальные нулевые значения счетчиков. Здесь $EV_{вызов}$ – идентификатор следующего системного события «вызов».

При активизации объектов-узлов в ответ на текущее событие $EV_{тек}$ к *исполнению* соответствующего метода-обработчика приступает объект, идентификатор которого равен значению $ID_{акт}$, взятому из соответствующей ячейки $MAIL[EV_{тек}]$.

Наиболее просто передача управления осуществляется в случае перехода по основной дуге: в этом случае в общем кортеже данных в ячейке $MAIL[EV_{тек}]$ в поле $ID_{акт}$ заносится идентификатор узла, в который осуществляется переход. Метод, ответственный за реакцию на текущее событие в таком узле, в соответствии с логикой работы ОСМ будет активизирован позднее соответствующего метода текущего исполняемого узла и сможет корректно обработать факт получения управления. Если же необходим переход по информационной дуге (что невозможно для ОСМ), то планируется следующее событие $EV_{след}$, в общий кортеж данных помещается новая ячейка с этим идентификатором, в нее передается идентификатор узла, в который осуществляется переход, и ссылка на кортеж текущих значений переменных, после чего обработка текущего события заканчивается. Таким образом, указанный узел получит управление при обработке следующего события.

Как только календарь спланированных событий исчерпывается, интерпретация ОСМ завершается. Особо выделим случай, когда интерпретацию необходимо провести несколько раз – если модель содержит одно или несколько чисто вероятностных правил. Это достаточно редкий вариант с потенциально значительной погрешностью, который наиболее целесообразно рассматривать лишь в контексте предикции времени исполнения. В этом случае полученные временные характеристики осредняются, возможно, с предварительным отсевом нетипичных значений путем элементарного кластерного анализа (оставляется один наибольший кластер).

Рассмотрим реализацию методов-обработчиков. Введем операцию применения правила $T = (V_{inp}, out, f(X))$, обозначив ее apply(T) со следующим содержанием:

$VARS[out] = f(X)$;

$\forall i : X_i = VARS\left[\left(V_{inp}\right)_i\right]$.

Операцию инкремента переменной x обозначим inc(x).

Запишем основное содержание методов, активизируемых при обработке любого из событий, следующих после $EV_{вызов}$, в ходе прямого исполнения модели:

$$\breve{M}(S) = \begin{vmatrix} inc(counter) \\ link = \begin{cases} i \in \tilde{N}_l \ \ if \ \ R_{trans}[i] \vee \\ \qquad \vee \begin{pmatrix} R_{trans}[i] = \varnothing \wedge \\ \wedge \forall y \in \tilde{N}_l, y \neq i : \neg R_{trans}[i] \end{pmatrix}, \\ \varnothing \ \text{ if } \ \left(\tilde{N}_l = \varnothing\right) \vee \\ \qquad \vee \begin{bmatrix} \left(\forall i \in \tilde{N}_l : \neg R_{trans}[i]\right) \wedge \\ \qquad \wedge \left(\neg \exists i \in \tilde{N}_l : R_{trans}[i] = \varnothing\right) \end{bmatrix}; \end{cases} \end{vmatrix}$$

$$\breve{M}(P_t) = apply(T_{trans}[link]);$$

$$\breve{M}(P_d) = apply(D_{trans}[link]);$$

$$\breve{M}(P_{td}) = \breve{M}(P_t) \circ \breve{M}(P_d);$$

$$\forall c \in C, \ \forall j \in N_{ev} : \left(M(c)\right)_j = \begin{vmatrix} \varnothing \ \text{ if } \ ID \neq ID_{акт}, \\ \breve{M}(c) \circ activate(link) \ \text{ if } ID = ID_{акт}, \end{vmatrix}$$

где $\tilde{N}_l$ – множество всех допустимых номеров выходных дуг; link – номер дуги, инцидентной по входу узлу, которому будет передано управление; N_{ev} – множество номеров всех событий. Знаком «∘» обозначена конкатенация операций, подразумевающая их строго последовательное исполнение слева направо. Все обозначения относятся к активному в данный момент объекту. Операция activate(link) передает управление узлу, инцидентному по входу дуге link.

В *режиме исполнения через компиляцию* ОСМ в процессе интерпретации генерирует код в соответствии с четырехэтапной схемой (размещение, инициализация, вызов, деинициализация) [3], при которой каждый объект порождает соответствующие фрагменты кода C++, реализующие инкремент счетчика и условные переходы с исполнением правил трансформации, модификацией переменной времени и, если это действие определено для дуги, инициализацией счетчика объекта, к которому переходит управление. Полученный код компилируется в функцию в составе динамически подгружаемого модуля.

В любом из режимов исполнения ОСМ результатами являются значения времени и/или внешних переменных, полученные при завершении моделирования.

Исследование основных вариантов применения метаслоя. *Предикция времени исполнения* позволяет, в частности, легко определить необходимость в распараллеливании расчета. Например, при исследовании динамики брюсселятора решается система из двух дифференциальных уравнений для различных сочетаний начальных значений переменных, хранимых в матрице размера K×K. Проведя несколько экспериментов для различных $K = 2^p$, $p \in Z$, с помощью метаслоя легко построить предикционную модель времени time(K). Зная издержки P(N) на запуск параллельного расчета на N процессорах, учитывая закон Амдала [8], легко получить условие эффективности распараллеливания:

$$P(N) < \frac{N-1}{N} time(K).$$

В таблице приведены данные экспериментальных замеров[1] времени параллельного (с распределением витков цикла по ядрам с помощью OpenMP, расписание цикла guided) и однопроцессорного решения указанной задачи при различных K, а также результаты проверки условия эффективности.

Замеры времени решения

K	800	160	32	6	1
Решение на одном ядре	1059	212	8.25	0.3	0.009
Решение на двух ядрах	539	107	4.47	0.47	0.414
Условие эффективности	+	+	+	–	–

Очевидно, что предикция времени исполнения в метаслое позволила избежать непроизводительных затрат на распараллеливание в двух последних случаях. Важно, что абстракция метаслоя от кода позволяет ввести такой прием профилирующей оптимизации в любой алгоритмический язык программирования с различными интерфейсами распараллеливания.

Предикция данных часто встречается при решении задач математической физики в частных производных с распараллеливанием по пространству. Требование непрерывности производных на стыках блоков обрабатываемых областей подразумевает выполнение обменов данными, что может быть длительной операцией. Иногда удается сократить временные затраты, заменив обмен экстраполяцией требуемых данных. Данная задача рассмотрена в работе [3], в которой показана возможность повышения эффективности распараллеливания некоторых задач моделирования распространения загрязнений на 4÷11 % за счет сокращения количества обменов. Построение соответствующей регрессионной модели данных вполне может быть реализовано в метаслое.

Комплексная предикция алгоритма и данных подразумевает параметризацию механизмов исполнения, которая вводит внутренние переменные состояния и тем самым упрощает как процесс построения, так и структуру функциональных моделей данных и времени исполнения, иногда повышает точность предсказания. Улучшение прогностических свойств возможно, например, в случае алгоритмов, в

[1] Результаты получены в системе с двухъядерным процессором Intel T4400, 2,2 ГГц. Компилятор Visual C++ 2008 Professional.

которых искомые величины являются такими функциями от входных параметров, что сложность их интерполяционного приближения превышает возможности встроенного прогностера, определяемые линейной регрессией или нелинейной в рамках МГУА.

Особенно заметно повышение точности прогноза, если производится предикция данных, итерационно рассчитываемых в ходе работы некоего алгоритма, например, численно решающего (методом Эйлера) дифференциальное уравнение, описывающее апериодическое звено 1-го порядка. Моделирование алгоритма (цикла расчета) со вводом внутренней переменной-счетчика в сочетании с элементарным приближением преобразования данных на одной итерации приводит к фактическому копированию *логики алгоритма обработки и аппроксимации выполняемых в нем преобразований* с закономерным итогом – абсолютная погрешность имеет уровень машинной ($\approx 10^{-13}$). Это уже не прогноз, а фактический расчет по индуцированной модели, причем трудоемкость исполнения модели приблизительно равна трудоемкости исполнения исходной программы, меньшие издержки были бы возможны лишь за счет устранения каких-либо проверок, вывода данных или иных необязательных в расчете действий, характерных для исходного алгоритма. Попытка прямого приближения «начальные значения → конечные значения» сводится к несомненно более трудной задаче интерполяции численного решения, которая может давать вполне заурядную погрешность порядка $10^{-3} \div 10^{-1}$, однако при существенно меньшей трудоемкости по сравнению с расчетом. Выбор того или иного варианта зависит от потребностей в копировании алгоритма расчета или его приближении. Возможны и промежуточные варианты.

Учет внутренних параметров путем моделирования алгоритма позволяет повысить точность прогноза времени в задачах, в которых оно является, например, алгоритмически нелинейной или даже разрывной функцией от входных переменных. Это многие расчетные задачи со сложными алгоритмическими зависимостями, например внутренним ветвлением в одном или нескольких вложенных циклах. Например, алгоритмически нелинейным по времени может являться процесс решения итерационно-сеточным методом уравнения, описывающего прогиб пластины под действием нелинейной по пространству нагрузки f(x,y,t):

$$\frac{\partial w}{\partial t} = k_1\left(\frac{\partial^2 w}{\partial x^2} + \frac{\partial^2 w}{\partial y^2}\right) - f(x,y,t);$$

$$f(x,y,t) = k_2 \int_0^{h(x,y)} \tau(x,y,z,t)dz,$$

где w – прогиб; k_1, k_2 – коэффициенты; τ – функция переменной линейной плотности тела, создающего нагрузку по вертикальной оси; h – функция, выражающая вертикальный размер тела. Интеграл рассчитывается каким-либо численным методом (с фиксированным шагом) с заданной точностью. Ввод внутренних параметров позволяет в несколько раз снизить погрешность предикции, доведя ее до 0,5÷2 %. При распараллеливании решения задачи по пространству предикция позволяет, в частности, заблаговременно более равномерно распределить нагрузку процессоров при параллельном вычислении f(x,y,t).

Заключение

Применение предикционных моделей алгоритмов и времени исполнения, формируемых в рамках метаслоя, позволяет проводить динамические профилировку и оптимизацию распараллеливаемых программ независимо от конкретного алгоритмического языка программирования и большинства стандартных интерфейсов распараллеливания. На реальных примерах показано, что предикция алгоритма, проецирующая его состояние на функциональные модели времени и данных, в ряде случаев упрощает такие модели и снижает погрешность предсказания. Предикция данных иногда может быть использована в качестве альтернативы передаче данных при параллельном решении некоторых задачах математической физики в частных производных, позволяющей уменьшить время расчета.

В перспективе предполагается расширение предикционных возможностей метаслоя для параллельных приложений путем построения моделей направлений и объемов пересылок данных.

Список литературы

1. **Пекунов В.В.** Модель образования и распространения твердых, жидких и газообразных загрязнителей. Оптимальное распараллеливание // Математическое моделирование. – 2009. – Т.21. – №3. – С.69–82.
2. **Пекунов В.В.** Процедуры с планированием повторного входа в языках высокого уровня при традиционном и параллельном программировании // Информационные технологии. – 2009. – №8. – С.63–67.
3. **Аветисян А.И., Иванников В.П., Гайсарян С.С.** Анализ и трансформация программ / Всероссийский конкурсный отбор обзорно-аналитических статей по приоритетному направлению «Информационно-телекоммуникационные системы», 2008.
4. **Пекунов В.В.** Автоматизация параллельного программирования при моделировании многофазных сред. Оптимальное распараллеливание // Автоматика и телемеханика. – 2008. – №7. – С.170–180.
5. **Бобровский С.И.** Технологии Delphi 2006. Новые возможности. – СПб.: Питер, 2006.
6. **Пекунов В.В.** Дедуктивный вывод объектно-событийных моделей. Применение при решении задач динамики многофазных сред // Вестник ИГЭУ. – 2008. – Вып.4. – С. 81.

7. **Дюк В., Самойленко А.** Data mining: Учебный курс. – СПб.: Питер, 2001.

8. **Воеводин В.В., Воеводин Вл.В.** Параллельные вычисления. – СПб.: БХВ-Петербург, 2002.

Пекунов Владимир Викторович,
ГОУВПО «Ивановский государственный энергетический университет имени В.И. Ленина»,
кандидат технических наук, доцент кафедры высокопроизводительных вычислительных систем,
e-mail: pekunov@mail.ru