Программный инструментарий анализа информационной структуры алгоритмов по их информационным графам

В.М.Баканов

Национальный исследовательский университет
"Высшая школа экономики"

Рассматривается реализация программного инструментария для выработки эффективных стратегий преобразования представлений информационных графов алгоритмов с целью как выявления скрытого параллелизма, так и определения рационального плана (расписания) выполнения параллельных частей программ при учете ограничений реальных многопроцессорных вычислительных систем. Для достижения гибкости разработки сценариев преобразований представлений графа используется встроенный скриптовый язык Lua.

Ключевые слова: информационный граф алгоритма, тонкая информационная структура программы, скрытый параллелизм, ярусно-параллельная форма информационного графа, преобразования представления информационного графа, встроенный скриптовый язык программирования Lua, рациональная стратегия построения плана выполнения параллельной программы.

1. Введение

В настоящее время из путей повышения производительности вычислительных систем доступен метод параллелизации обработки данных; иные пути ограничены физическими законами.

Однако параллелизация (фактически одновременное = параллельное выполнение различных частей одной программы на независимо работающих вычислительных блоках) требует от алгоритмистов и программистов дополнительных усилий по выявлению в алгоритме участков, могущих быть выполненными параллельно; основное требование к таким блокам (*зернам* или *гранулам*) параллелизма состоит в независимости (ортогональности) их по данным.

2. Обзор исследований в данной области

Анализ программ (обычно именуемый исследованием их тонкой информационной структуры, [1]) по такому критерию непрост, поэтому говорят о скрытом (не фиксируемом при поверхностном рассмотрении) параллелизме; мощным инструментом при таком анализе является представление программ графовыми моделями. В России разработкой методов анализа структуры алгоритмов на основе графовых моделей занимались авторы работы [1].

Ситуация усложняется необходимостью рационального использования ресурсов конкретной многопроцессорной вычислительной системы (МВС). Эти ресурсы (число параллельно работающих вычислителей, объемы памяти на каждом и др.) всегда ограничены, поэтому задача планирования использования ресурсов очень важна. Формально это задача распознавания и оптимизации, причем конкретные критерии оптимизации определяются поставленной исследователем целью.

Исследования в данной области обладают практической ценностью для создания эффективных распараллеливающих компиляторов для вычислительных систем архитектуры MIMD (*Multiple Instruction stream - Multiple Data stream*) по М.Флинну [1] при управлении от потока команд (*Instruction Flow*) с выявлением параллелизма на уровне программного обеспечения (собственно *распараллеливающего компилятора*). Имеются изустные сведения об обсуждении и проведении подобных исследований при разработке компиляторов для серии компьютеров Эльбрус в конце 80-х и начале 90-х г.г., и, уж, наверное, проекта Itanium.

Информационный граф алгоритма (вычислительная модель “операторы - операнды”, далее ИГА) суть наиболее простое представление конкретного алгоритма в графовом виде. ИГА описывает исключительно информационные зависимости алгоритма (вершины графа соответствуют отдельным операциям - преобразователям информации (арифметико-логические действия и т.п.); наличие дуги между вершинами *i*,*j* говорит о необходимости при выполнении операции *j* априорного существования аргументов (операндов), выработанных операцией *i*; в случае независимости выполнения операций *i* и *j* дуга между вершинами отсутствует). Свойства ИГА: ациклличность, направленность, детерминированность. Конечно, в данном случае термин "операторы" в известной степени условен, ибо под "операторами" можно понимать отдельные независимые по данным последовательности вычислений (*гранулы параллелизма*) любого (желательно приблизительно одинакового) размера.

Существуют и иные формы графового представления алгоритма - напр., с вершинами - распознавателями (это условные операторы, переключатели и др.), которые определяют последовательность срабатывания преобразователей при работе программы [1], однако использование их для моделирования практически возможно лишь для относительно несложных алгоритмов.

Формально процедура выявления параллелизма по информационному графу алгоритма (ИГА) состоит в построении его ярусно-параллельной формы (ЯПФ). Ярусно-Параллельная Форма (ЯПФ, SPF – *Stacked Parallel Form*) информационного графа алгоритма суть форма представления графа, при которой операторы, могущиеся выполняться независимо друг от друга (фактически параллельно) располагаются на одном уровне (ярусе). Представление графа в ЯПФ - одно из наиболее мощных средств выявления скрытого параллелизма в алгоритме. Формально ЯПФ строится на основе ИГА с помощью несложного алгоритма трудоемкостью порядка $O(N^2)$; на рис. 1 приведена ЯПФ графа процедуры вычисления вещественных корней полного квадратного уравнения.

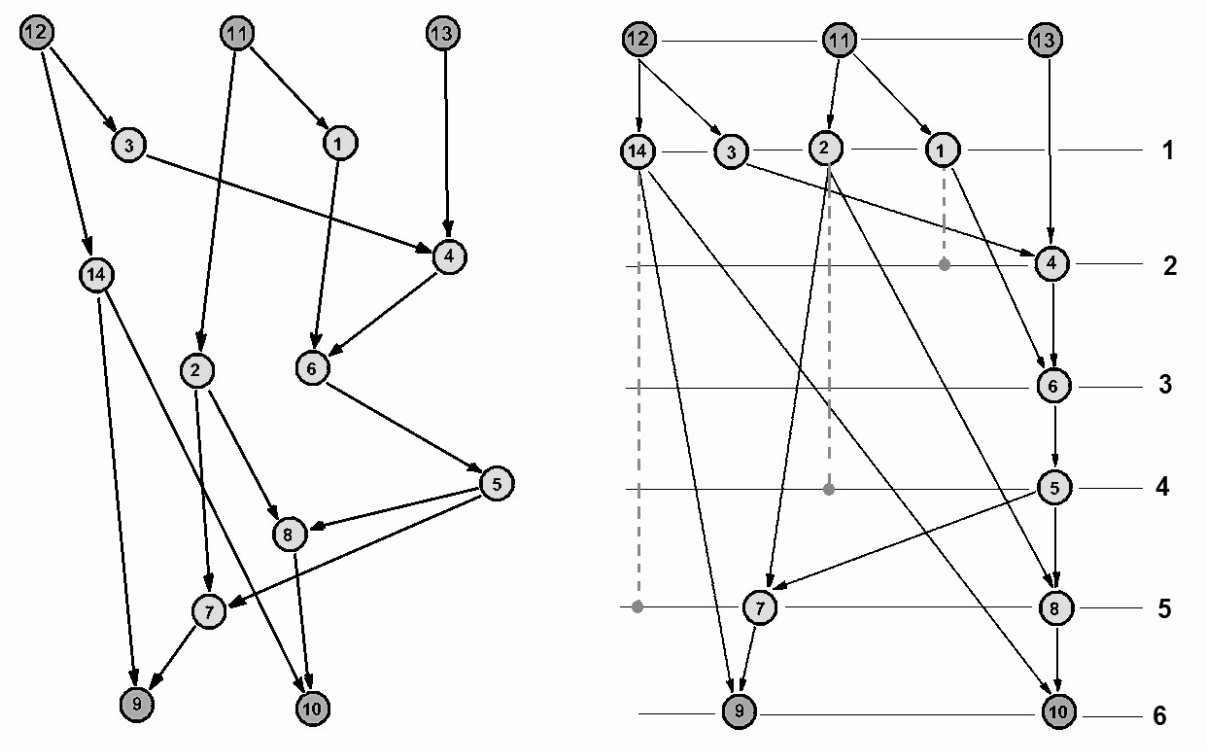

Рис. 1. Информационный граф (см. рисунок слева, общий вид: узлы 11,12,13 - входные данные, 9 и 10 - вычисление выходных данных) и его представление в ЯПФ (справа) процедуры нахождения вещественных корней полного квадратного уравнения (цифры крайние справа - номера ярусов ЯПФ)

Высота ЯПФ (вычисляемая чрез число ярусов) определяет общее время выполнения алгоритма, ширина ЯПФ – максимальное число задействованных отдельных (параллельно работающих) вычислителей (напр., отдельных ядер процессора) данной МВС.

3. Применяемые при исследовании методы

Реальные ЯПФ информационных графов обладают большой неравномерностью распределения числа операторов по ярусам, при этом использование ресурсов МВС очень нерационально - максимум эффективности использования ресурсов обычно достигается при равномерном (или близком к равномерному) распределении операторов по ярусам.

Однако практически в любом ЯПФ имеется *вариативность* в расположении вершин графа (операторов) между ярусами (перемещение операторов между ярусами ограничивается соблюдением информационных зависимостей операторов в графе; для ЯПФ графа на рис.1 оператор 14 может быть перемещен "вниз" на любой ярус с 2 по 5-й, оператор 2 - на любой ярус с 2 по 4-й, оператор 1 - на ярус 2); возможный диапазон перемещений показан на рис.1 пунктиром. Эта особенность ЯПФ дает возможность оптимизации ЯПФ (в смысле, напр., достижения наибольшей равномерности распределения операторов по ярусам – при этом автоматически достигается максимум использования ресурсов МВС). Для ИГА большого размера трудоемкость такой оптимизации запредельно велика (задача перестановок с учетом влияния изменения конфигурации ЯПФ после каждой), существующие алгоритмы разбиения графов являются эвристическими (напр., KL-алгоритм [2]). Достижение цели (напр., равномерности распределения операторов по ярусам ЯПФ) возможно с помощью различных стратегий перестановки операторов по ярусам; выбор оптимальной (рациональной) стратегии для ЯПФ различной сложности – практически-полезная задача, имеющая прямое отношение к проблеме эффективного распараллеливания алгоритмов.

Итак, имеем две (*обязательно последовательно* выполняемые) задачи:

а) Выявление в заданном алгоритме собственно наличие параллелизма (для выполнения этого как нельзя лучше подходит метод представления ИГА в ярусно-параллельной форме).

б) Планирование параллельного выполнения алгоритма с учетом реалий (ограничений) конкретной МВС.

Ограничения МВС - число параллельно работающих вычислителей, объемы их памяти и др., выполнение программы с учетом этого достигается эквивалентными (не изменяющими информационных зависимостей в графе) преобразованиями ИГА в ЯПФ.

Основные параметры ЯПФ графа (применяются также и иные):

- B - высота ЯПФ графа, определяется количеством ярусов
- H_i – ширина i-того яруса (количество операторов на нем)
- H – ширина ЯПФ графа, определяется максимумом операторов по всем ярусам графа; $H = max(H_i)$
- $\overline{H}$ – средняя ширина ЯПФ; $\overline{H} = \frac{1}{N}\sum_{i=1}^{i=N} H_i$, N - число ярусов ЯПФ
- $K_i = H_i/\overline{H}$ – степень (коэффициент) заполнения i-того яруса ЯПФ графа
- M_{min} – минимальное количество (параллельно работающих) вычислителей, необходимое для реализации вычислений согласно данного графа; $M_{min} = max(H_i)$
- $K_{irregular}$ - степень (коэффициент) неравномерности распределения операторов по ярусам ЯПФ графа; $K_{irregular} = max(H_i)/min(H_i)$
- $\sigma_H = \sqrt{\frac{1}{N}\sum_{i=1}^{i=N}(H_i - \overline{H})^2}$ - среднеквадратичное отклонение ширин ярусов ЯПФ

Основные постановки оптимизационной задачи (не только возможны, но и полезны комбинации):

а) Наиболее полная загрузка неизвестного заранее числа параллельно работающих вычислительных узлов имеющейся МВС при минимальном времени выполнения задачи: $K_{irregular} \to 1{,}0$ при $B = B_0 = const$ (где B_0 - длина критического пути - минимальная из всех возможных высот у ЯПФ данного ациклического графа).

б) Наиболее полное использование ресурсов (вычислительных узлов) имеющейся МВС при допущении увеличения времени выполнения: $max(H_i) \leq M_0$ при $B \geq B_0$ (здесь M_0 - число доступных параллельно работающих вычислителей в заданной МВС); однако желательно $B-B_0 \rightarrow min$ (минимизация времени выполнения). Может быть полезно представить этот процесс как подготовку состоящей из не более чем заданного числа M_0 заведомо независимых (ортогональных) по данным команд для вычислительной системы архитектуры VLIW (*Very Long Instruction Word*) в иделогеме EPIC (*Explicitly Parallel Instruction Computing* - микропроцессорная архитектура с явным параллелизмом команд).

По данным [1] поставленная задача выработки рационального расписания выполнения параллельных частей программ относится к классу *NP*-полных (комбинаторная задача с ограничениями). С практической точки зрения необходимо иметь мощный и гибкий аппарат для нахождения приемлемых решений этой задачи (исходя из сказанного приемлемы эвристические методы).

Одной из тенденций современного программирования является использование встраиваемых скриптовых языков (ВСЯ) для расширения возможностей программных комплексов в области сложной обработки данных. Историю собственно скриптовых языков можно проследить с 60-х годов (языки пакетной обработки и языки командных оболочек, используемые чаще всего в пакетном режиме обработки), однако большую часть прошедшего времени применение их ограничивалось системными проектами (уровень операционных систем), только в последнее время проявляется тенденция использования ВСЯ в программном обеспечении меньшего масштаба.

Встраиваемые скриптовые языки вследствие присущей гибкости описания алгоритмов дают богатые возможности сложного преобразования данных, недостижимые традиционными методами (напр., привычными системами организации интерфейса с пользователем - меню, кнопками и др.). В случае ВСЯ разработчик (иногда и квалифицированный пользователь) описывает (обычно при этом вызовы ВСЯ становятся "оберткой" функций API "родительского" приложения, при этом ВСЯ фактически становится предметно-ориентированным языком программирования) необходимые действия на ВСЯ, в дальнейшем "родительское" приложение использует это описание для совершения заданных действий. Успехи в разработке ВСЯ позволили существенно уменьшить размер их кода (часто такие языки поставляются в виде исходных текстов в режиме Open Source), в этом случае при встраивании размер "родительского" приложения возрастает незначительно. В настоящее время известны встраиваемые системы на основе Basic-подобного языка (семейство программ 1С), Java (JavaScript, JScript), Python, Ruby, Tcl, PHP, Perl, REBOL, NewLISP и др.

Недостатками скриптовых языков являются повышенное время исполнения вследствие интепретируемости (идеологема применения этих ВСЯ в качестве "оберток" над вызовами быстрых компилируемых языков эту проблемы практически снимает), отсутствие удобной интегрированной среды разработки (IDE) и маркетингового бюджета.

Интересным ВСЯ является Lua, разрабатываемый с 1993 г. в католическом университете Рио-де-Жанейро [3]. Язык Lua является нетипизированным, написан исключительно на ANSI C и полностью Open Source, применяет подход интерпретации исходного текста с промежуточной формой перед выполнением программы, позволяет использовать объектно-ориентированную модель программирования, реализует невытесняющую многонитевость, имеет много стандартных библиотек и пакетов расширений. Язык Lua применен как встроенный при разработке таких приложений, как Adobe Lighroom, Nmap, Word of Warcraft, Stalker и др.

Встроенный скриптовый язык Lua использован при разработке описываемой в данной работе программной системы выявления скрытого параллелизма и составления расписания выполнения параллельных частей программы. Данная программная система написана на языке программирования C++ и является 32-битным приложением Windows (рис. 2); в связи с предполагаемым большим числом вычислений предполагается интегрирование ее в инфраструктуру BOINC (*Berkeley Open Infrastructure for Network Computing*) и расположение на сервере ВУЗ'а.

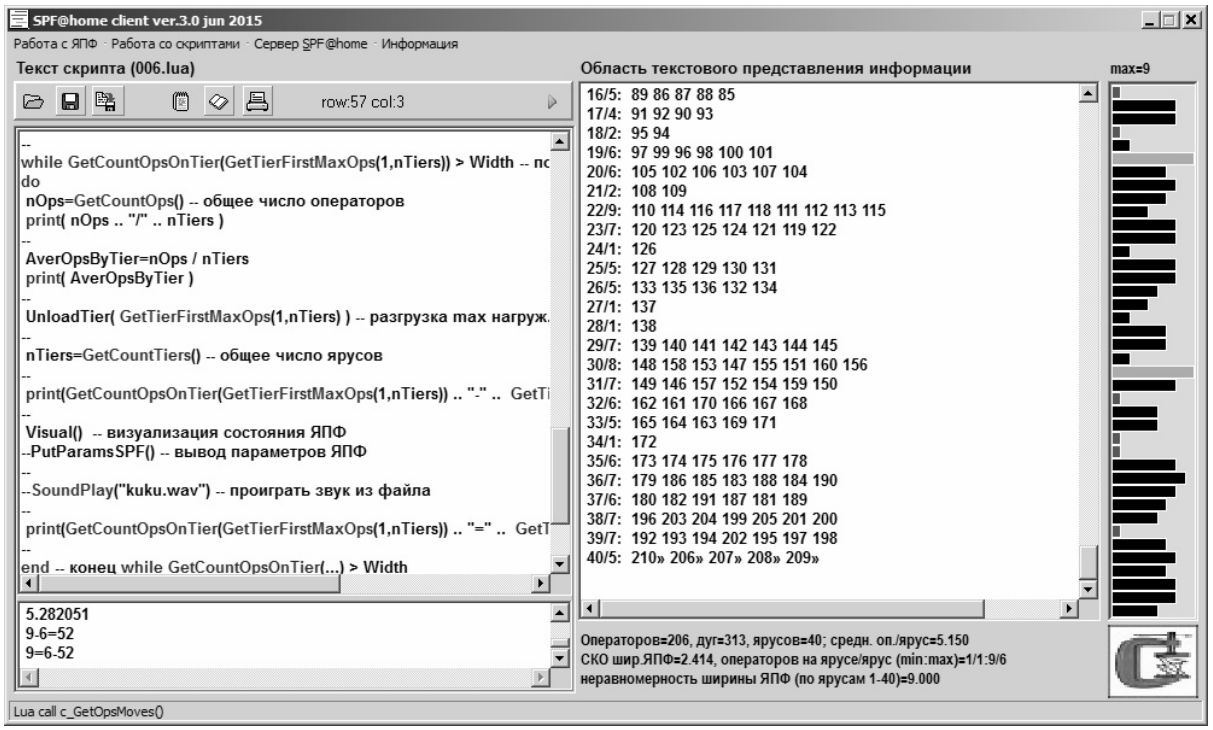

Рис. 2. Главное окно программного модуля выявления скрытого параллелизма и составления расписания выполнения параллельных частей программы

Программный инструментарий (фактически API пользователя) разработанной системы позволяет реализовывать любой из перечисленных критериев оптимизации (а также их комбинации), т.к. выбор критерия осуществляет собственно пользователь (на основе задач, им поставленных). К родственным по тематике с данным проектам можно отнести V-Ray и ПАРУС (оба разработки МГУ им. М.В.Ломоносова, Россия); известны также системы METIS и ParMETIS (университет Миннесоты, 1998-2003), функционал которых для данного случая явно избыточен.

Инструментарий системы включает три типа API-вызовов (всего около 30 штук, причем каждый является Lua-"оберткой", соответствующей C-функции):

- Информационные (служат для получения информации о ИГА и его ЯПФ; именно на основании этих данных принимается решение о применимости одной из стратегий преобразования ЯПФ графа, реализуемой в дальнейшем на языке Lua).
- Акционные (служат для эквивалентных, т.е. не меняющих информационные связи в ИГА, преобразований ЯПФ).
- Вспомогательные (работа с файловой системой и т.п.).

Примерами информационных вызовов являются - получить общее число ярусов ЯПФ, число операторов на заданном ярусе, получить диапазон ярусов для возможного нахождения на них заданного оператора и др., акционных – переместить заданный оператор на конкретный ярус ЯПФ (с учетом ненарушения информационных зависимостей), создать новый пустой ярус, уничтожить пустой ярус и др. В процессе выполнения акционных функций собственно ИГА не изменяется (т.е. программа остается той же самой) - меняется лишь представление графа (в данном случае - его ЯПФ).

Исходный (и преобразуемый в дальнейшем) информационный граф алгоритма может быть получен следующими методами:

1) Построен вручную согласно текстовому описанию алгоритма или же представлению его на псевдоязыке или в виде блок-схемы (для вновь разрабатываемых алгоритмов).

2) Путем анализа исходного текста программы на одном из традиционных языков программирования (подобные анализаторы существуют, однако зачастую выдают противоречивые результаты).

3) Как результат работы программы-графогенератора (преимущество - можно задавать априори определенные параметры ИГА, дающие максимальное его приближение к реальным; однако использование обычно ограничивается тестовыми примерами).
4) В случае применение данной системы в качестве компоненты распараллеливающего компилятора наличие ИГА априорно (компилятор при работе производит анализ программы на информационные зависимости - т.е. фактически строит ИГА).

Автор для составления *корректного* ИГА использует программный симулятор вычислителя потоковой (DATA-FLOW) архитектуры [4], в котором программа отлаживается и в дальнейшем экспортируется в файловый формат ИГА (список смежных вершин).

Одна из наиболее сложных целей проекта – определение наиболее эффективной стратегии балансировки ярусно-параллельной формы информационного графа алгоритма; здесь под балансировкой понимается равномерность распределения операторов по ярусам ЯПФ. Задача имеет прямое отношение к проблеме эффективного распараллеливания алгоритмов и, в частности, к вопросу наиболее эффективного использования ресурсов МВС. На первой стадии из всех реалий конкретной МВС учитывается только число параллельных вычислителей, но в перспективе стоят планы учета дополнительных параметров.

В качестве параметра эффективности каждого предложенного решения использовалось число перестановок операторов с яруса на ярус ЯПФ для получения заданного результата.

4. Результаты и их обсуждение

Эффективность обработки данных иллюстрирована табл. 1 (описанная на Lua стратегия балансировки ширин ярусов ИГА без увеличения высоты ЯПФ) и табл. 2,3 (стратегия снижения ширины ЯПФ к заданной величине совместно с балансировкой) для нескольких ИГА различной сложности.

Преобразования ЯПФ выполнены для различных алгоритмов, заданных информационными графами. Информационные графы slau_2a, slau_3a, slau_5a и slau_10a - решение СЛАУ порядков 2,3,4,5,10 соответственно классическим методом Гаусса, mnk_10 и mnk_20 – линейная аппроксимация по методу наименьших квадратов для 10 и 20 точек, korr_10 и korr_20 – определение коэффициента парной корреляции для 10 и 20 точек, m_matr_5 и m_matr_10 – умножение квадратных матриц порядков 5 и 10 классическим методом.

Стратегия преобразования ЯПФ, с помощью которой получены приведенные в табл. 1 результаты, названа (условно) "Бульдозер" и основана на перемещении операторов с наиболее нагруженных ярусов на недогруженные (так нож бульдозера срезает холмы и сдвигает их материал во впадины); критерий остановки алгоритма – перебор всех операторов, могущих быть перенесенными "с холмов во впадины" с целью балансировки ЯПФ. Табл. 2 и 3 – результат применения стратегии "Бисекция" – наиболее нагруженные ярусы разгружаются путем перенесения половины находящихся на них операторов на вновь образованные ярусы, созданные под данным ярусом; критерий останова – "ужатие" ЯПФ до ширины, не превышающей заданной.

Качество достигнутых преобразований ЯПФ в табл. 1-3 характеризуется безразмерными показателями $I_\sigma = \frac{\sigma_H^{origin}}{\sigma_H^{reformed}}$ и $I_K = \frac{K_{irregular}^{origin}}{K_{irregular}^{reformed}}$, где первый показывает, во сколько раз при применении данной стратегии снизилась величина СКО распределения числа операторов по ярусам, второй - то же по коэффициенту нерегулярности (*original* и *reformed* - показатели до и после преобразования ЯПФ); трудоемкость преобразования - числом перестановок операторов с яруса на ярус.

Как видно из данных табл. 1-3, описанные (не являющимися излишне изощренными) стратегии в самом деле позволяют достичь требуемого результата (наблюдается снижение неравномерности ширин ЯПФ по ярусам), однако с совершенно разной эффективность для различных ИГА. В целом наблюдается повышение эффективности стратегии при увеличении сложности (вариативности) ИГА. Особенна интересна и практически важна задача априорного (до проведения собственно преобразований ЯПФ) предсказания уровня их эффективности (задача распознавания). Большего эффекта можно добиться путем совершенствования стратегий и разумного их совместного применения.

Таблица 1. Эффективность применения стратегии балансировки ширины ЯПФ ИГА при неизменной высоте ЯПФ

файл описания ИГА / *число дуг / узлов / ярусов первоначального ЯПФ*	*средняя ширина ЯПФ*	*показатель* I_σ	*показатель* I_K	*число перестановок операторов*
slau_2a.edg / 18/9/7	1,29	1,00	1,00	0
slau_3a.edg / 56/28/13	2,15	1,13	1,00	4
slau_5a.edg / 230/115/25	4,60	1,06	1,00	13
slau_10a.edg / 1610/805/63	12,8	1,05	1,00	81
mnk_10.edg / 132/66/16	4,13	1,002	1,00	1
mnk_20.edg / 252/126/26	4,85	1,001	1,00	1
korr_10.edg / 174/88/15	5,87	1,001	1,00	4
korr_20.edg / 334/168/25	6,72	1,47	1,41	22
m_matr_5.edg / 450/225/5	45,0	1,58	1,32	30
m_matr_10.edg / 3800/1300/10	190	1,97	1,67	407

Таблица 2. Эффективность применения стратегии получения ЯПФ не более заданной ширины совместно с балансировкой ширин ярусов ЯПФ (1)

файл описания ИГА / *число дуг / узлов / ярусов первоначального ЯПФ*	*заданная ширина преобразованной ЯПФ <=10*			
	возрастание высоты ЯПФ	*показатель* I_σ	*показатель* I_K	*число перестановок операторов*
slau_2a.edg / 18/9/7	1,00	1,00	1,00	0
slau_3a.edg / 56/28/13	1,00	1,00	1,00	0
slau_5a.edg / 230/115/25	1,16	1,80	2,00	32
slau_10a.edg / 1610/805/63	2,24	8,10	9,00	926
mnk_10.edg / 132/66/16	1,19	2,99	3,67	21
mnk_20.edg / 252/126/26	1,19	3,72	4,20	51
korr_10.edg / 174/88/15	1,20	3,28	4,00	32
korr_20.edg / 334/168/25	1,28	6,16	7,75	91
m_matr_5.edg / 450/225/5	6,40	43,3	3,76	342

m_matr_10.edg / 3800/1300/10	27,2	306	7,52	5232

Таблица 3. Эффективность применения стратегии получения ЯПФ не более заданной ширины совместно с балансировкой ширин ярусов ЯПФ (2)

файл описания ИГА / *число дуг / узлов / ярусов первоначального ЯПФ*	*заданная ширина преобразованной ЯПФ <=5*			
	возрастание высоты ЯПФ	*показатель* I_σ	*показатель* I_K	*число перестановок операторов*
slau_2a.edg / 18/9/7	1,00	1,00	1,00	0
slau_3a.edg / 56/28/13	1,15	2,11	2,00	6
slau_5a.edg / 230/115/25	1,56	3,99	4,00	70
slau_10a.edg / 1610/805/63	3,65	18,9	18,0	1234
mnk_10.edg / 132/66/16	1,31	3,74	4,40	27
mnk_20.edg / 252/126/26	1,35	6,37	8,40	67
korr_10.edg / 174/88/15	1,53	5,96	6,40	51
korr_20.edg / 334/168/25	1,64	10,8	12,4	124
m_matr_5.edg / 450/225/5	12,8	80,0	3,76	451
m_matr_10.edg / 3800/1300/10	54,4	540	7,52	6152

5. Выводы

Предложена программная система (инструментарий) для анализа информационной структуры программ по их представлению в виде информационного графа, позволяющая не только выявлять скрытый параллелизм, но и разрабатывать рациональные расписания выполнения частей параллельных программ с учетом параметров реальных многопроцессорных систем. Исключительно применение встроенного скриптового языка Lua для реализации стратегий целенаправленного изменения ярусно-параллельной формы графового представления позволило добиться гибкости и эффективности в реализации поставленной цели; это было бы немыслимо иными методами.

На примерах показана действенность данного подхода для решения поставленных задач. Результаты исследований могут быть использованы как для теоретического анализа алгоритмов, так и в практике при разработке эффективных распараллеливающих компиляторов. Опыт показывает эффективность применения данной разработки при усвоении основ параллелизации студентами ВУЗ'ов, при этом естественным образом реализуются разработки исследовательского направления.

Литература

1. Воеводин В.В., Воеводин Вл.В. Параллельные вычисления. — СПб. БХВ-Петербург, 2002. — 608 с.

2. Kernighan B.W., Lin S. An efficient heuristic procedure for partitioning graphs // The Bell System Technical Journal, vol.49, № 2, pp.291-307, feb.1970.
3. Иерузалимски Р. Программирование на языке Lua. — М.: ДМК Пресс, 2014. — 382 с.
4. Баканов В.М. Управление динамикой вычислений в процессорах потоковой архитектуры для различных типов алгоритмов. // ПРОГРАММНАЯ ИНЖЕНЕРИЯ, 2015, № 9, с. 20-24.

Software analysis tools information structure algorithms by their information graphs

V.M.Bakanov

National Research University "Higher School of Economics"

The realization of software tools for developing effective strategies to transform ideas information graphs algorithms for the purpose of identifying how parallelism, and the definition of management plan (schedule) of parallel parts of the program taking into account the limitations of the real multiprocessor systems. To achieve flexibility in the development of scenarios graph representations of transformations using embedded scripting language Lua.

Keywords: information graph algorithm, the thin structure of the program information, hidden parallelism, parallel-stacked form of the information graph, convert the graph representation of the information, built-in scripting language Lua programming, rational strategy of building plan of the parallel program.

References

1. Voevodin V.V., Voevodin Vl.V. Parallel'nye vychisleniya [Parallel computing]. SPb.: BHV-Petersburg, 2002. - 608 c.
2. Kernighan B.W., Lin S. An efficient heuristic procedure for partitioning graphs // The Bell System Technical Journal, vol.49, № 2, pp.291-307, 1970.
3. Ieruzalimski R. Programmirovanie na yazyke Lua. [Programming in Lua]. M.: DMK Press, 2014. 382 c.
4. Bakanov V.M. Management dynamic computing processors in a streaming architecture for various types of algorithms. // SOFTWARE ENGINEERING, 2015, № 9, p. 20-24.