ПОСТРОЕНИЕ КАРКАСА ВЫПОЛНЕНИЯ ПАРАЛЛЕЛЬНЫХ ПРОГРАММ НА ОСНОВЕ ГРАФОВОГО ПРЕДСТАВЛЕНИЯ АЛГОРИТМОВ С УЧЕТОМ ГЕТЕРОГЕННОСТИ ВЫЧИСЛИТЕЛЬНОГО ПОЛЯ

В.М.Баканов

Национальный исследовательский университет "Высшая школа экономики",
Россия, 101000, г.Москва, ул.Мясницкая, д.20
E-Mail: vbakanov@hse.ru

Аннотация: Рассмотрены подходы к решению проблемы создания рациональных методов разработки каркаса параллельной программы (планирования выполнения независимых по данным частей параллельной задачи) для реальных параллельных вычислительных систем и инструментальная программная среда (программный стенд) для реализации таких методов. Для моделирования и оптимизации методов используется встроенный скриптовый язык программирования Lua. Приводятся результаты применения некоторых предложенных стратегий построения рациональных планов выполнения параллельных программ.

Введение

В настоящее время из путей повышения производительности вычислительных систем доступен метод параллелизации обработки данных; путь же повышения тактовой частоты ограничивается фундаментальными законами и технической реализацией производства процессоров.

Однако параллелизации (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 исполненными параллельно (один из вариантов декомпозиции); основное требование к таким блокам (*зёрнам* или *гранулам*) параллелизма состоит в независимости (ортогональности) их по данным.

Анализ алгоритмов (часто именуемый *исследованием тонкой информационной структуры алгоритма*) 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 нашей стране разработкой методов анализа структуры алгоритмов на основе графовых моделей занимались В.В.Воеводин и Вл.В.Воеводин [1].

Ситуация усложняется необходимостью рационального использования ресурсов конкретной многопроцессорной вычислительной системы (МВС). Эти ресурсы (число и возможности параллельно работающего поля вычислителей) всегда ограничены, поэтому задача планирования использования ресурсов столь важна.

Несмотря на наличие архитектур параллельных вычислительных систем, не нуждающихся в априорном планировании (напр., потоковый вычислитель аппаратно определяет очередность выполнения операторов на этапе выполнения программ), именно априорному планированию при реализации программ на МВС в настоящее время отдается приоритет. Ярким примером является создание программ для параллельных вычислителей архитектуры сверхдлинного командного слова (VLIW, *Very Long Instruction Word*) - аппаратно-программная иделогема микропроцессорной архитектуры с явным параллелизмом команд (EPIC, *Explicitly Parallel Instruction Computing*).

Априорный (до этапа выполнения) анализ структуры алгоритмов не является самодостаточным и является лишь основой для оценки эффективности выполнения программы на конкретной МВС, учет же конкретики данной МВС во многих случаях затруднен аналитически и требует числового моделирования и оптимизации; в этом смысле направление разработки инструментальных программных систем для решения таких задач актуально.

Часто для представления и анализа программ используются (наряду с иными формами) информационные графы (напр., проект AlgoWiki, [2]). На практике для максимального повышения быстродействия оптимизации подвергаются критические участки программы, не имеющие ветвлений; в этом случае информационный граф достаточен для описания алгоритма.

Целью AlgoWiki является "дать исчерпывающее описание алгоритма, которое поможет оценить его потенциал применительно к конкретной параллельной вычислительной платформе". Естественным следующим (перед процедурой собственно кодирования) этапом в цикле разработки параллельного приложения должно являться моделирование выполнения программы на конкретной МВС (с учётом параметров вычислительного поля и размерности данных), при этом формируется рациональный каркас (фактически *план выполнения*) параллельной программы.

Фрагмент цикла разработки эффективной параллельной программы приведен ниже:

- графовое представление алгоритма (программы)
- теоретическое исследование свойств алгоритма по его графовому представлению
- создание и оптимизация каркаса (плана выполнения) параллельной программы
- разработка исходного кода с использованием выбранных технологий параллельного программирования.

Именно 3-й этап вышеприведенного списка является предметом рассмотрения данной работы.

При представлении программы информационным графом (ИГА, вычислительная модель “операторы - операнды”) вершины графа соответствуют преобразователям информации (операторам), а дуги - информационным связям операторов (данным). В данном случае термин "операторы" в известной степени условен, ибо под "операторами" понимаются отдельные последовательности вычислений (*блоки, подзадачи*) любого (в принципе) размера (в зависимости от соглашений, принятых при декомпозиции исходной программы). Свойства ИГА: ацикличность, однонаправленность, детерминированность. Дуги ИГА нагружены величинами объема пересылаемых данных, вершины - значениями вычислительной сложности операторов (в машинных командах или тактах процессора). Эти данные не относятся к каркасу и формально конкретизируются на следующем этапе разработки параллельной программы (выбор технологии параллельного программирования и собственно кодирование) для оценки времени ее выполнения. Время пересылки данных определяется технологией обмена с учетом латентности коммуникационной среды, время выполнения операторов - параметрами процессора (возможность "плавающих вычислений", суперскалярности и др.).

Удобным методом выявления параллелизма является представление ИГА в ярусно-параллельной форме (ЯПФ); при этом операторы, могущиеся выполняться независимо друг от друга (фактически параллельно) располагаются на одном уровне (*ярусе*). Формально каждый ярус ЯПФ является одним из сечений ИГА, удовлетворяющих вышеприведенному условию; трудоемкость построения ЯПФ полиномиальная (квадратичная).

Глубина ЯПФ (определяемая числом ярусов) задает общее время выполнения алгоритма, ширина ЯПФ – максимальное число задействованных отдельных (параллельно работающих) вычислителей (напр., отдельных ядер процессора или узлов) данной МВС. В случае начала построения ЯПФ с исходных данных получаем "верхний" вариант ЯПФ (операторы располагаются на наиболее близких к начальным ярусах), при построении начиная с выходных данных получаем "нижний" вариант; эти два представления ограничивают весь диапазон возможного расположения операторов на ярусах ЯПФ). Обычно принимается, что в рамках каждого яруса операторы выполня-

ются синхронно, а поле параллельных вычислителей гомогенно.

Собственно каноническая ЯПФ может уже рассматриваться как некий *план выполнения* (*каркас*) частей параллельной программы. При этом все находящиеся на одном ярусе операторы исполняются параллельно на вычислительном поле МВС. Недостатком является при этом значительный (часто на многие порядки, причем характеризующийся резким ростом с увеличением размера обрабатываемых данных даже при неизменном алгоритме их обработки) разброс ширин ярусов ЯПФ. Подобный режим реализуем лишь на гипотетической МВС с бесконечно большим числом параллельных вычислителей (согласно *концепции неограниченного параллелизма*, [1]).

Относительный разброс числа ширин ярусов ЯПФ обычно характеризуют коэффициентом вариации (1):

$$CV = {}^{\sigma}/_{\bar{x}}\,, \qquad (1)$$

где σ - стандартное отклонение, $\bar{x}$ - среднее значение.

Разработка эффективных методов (*стратегий*) модификации ЯПФ в сторону снижения ***CV*** каждого программного задания (особенно для гетерогенного поля параллельных вычислителей) является важной задачей, т.к. при разумной стратегии балансировки всей МВС приводит к минимизации уровня дисбаланса установки в первую очередь вследствие минимизации делимого в аналогичном (1) выражении для всего параллельного вычислителя. В данной работе разработана программная среда (*программный стенд*) для разработки таких методов и предложены некоторые стратегии с анализом их количественных характеристик согласно критерия эффективности применения.

Теоретическая часть

В большинстве ЯПФ имеется *вариативность в расположении вершин графа* (операторов) между ярусами (возможность перемещения операторов между ярусами, которая ограничивается соблюдением информационных зависимостей между операторами); для ЯПФ графа на рис.1 слева оператор 5 может быть перемещен "вниз" на любой ярус с 2 по 5-й, оператор 11 - на любой ярус с 2 по 4-й, оператор 7 - на ярус 2. Эта особенность ЯПФ даёт возможность оптимизации ЯПФ (в смысле, напр., достижения наибольшей равномерности распределения операторов по ярусам); в случае рис.1 слева возможна "разгрузка" яруса 1 от операторов 5 и 11, в результате приведенная программа может быть выполнена на 2 (вместо 4-х) параллельно работающих вычислителях за то же время. В то же время встречаются ЯПФ с нулевой вариативностью (рис.1 справа).

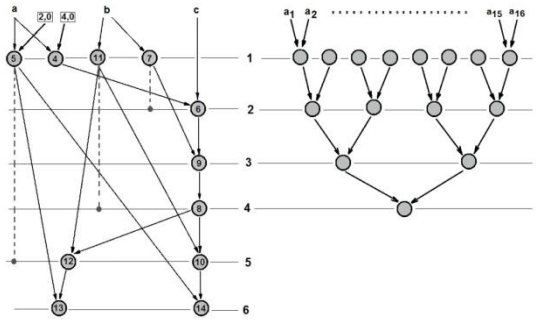


Рис. 1. Примеры ярусно-параллельной формы алгоритмов с потенциалом вариативности расположения операторов по ярусам (слева, процедура вычисления корней полного квадратного уравнения, пунктиром показан диапазон расположения операторов по ярусам) и без потенциала вариативности (справа, процедура сдваивания для 16 чисел); цифры в центре - номера ярусов ЯПФ.

Для количественной оценки уровня вариативности (*вариабельности*) ***V*** конкретной ЯПФ разумно использовать выражения:

$$V_P = \sum \frac{P_i}{P} \qquad (2)$$

и

$$V_T = \sum \frac{T_i^{max} - T_i^{min}}{T}\,, \qquad (3)$$

где в обоих случаях суммирование ведется по номерам операторов ***i***, $\boldsymbol{P_i}$ - число обладающих вариативностью операторов, ***P*** - общее число операторов, T_i^{max} и T_i^{min} - максимальный и минимальный номера ярусов данной ЯПФ, на которых может располагаться ***i***-тый оператор (фактически диапазон возможного их расположения), ***T*** - общее число ярусов ЯПФ. Выражения (2) учитывает вклад числа операторов со свой-

ством вариабельности, выражение (3) - величину собственно вариабельности, оба они дают ноль при отсутствии вариабельности и монотонно возрастают с её увеличением; логична также суперпозиция выражений (2,3) в форме их произведения.

Практическая полезность априорного вычисления уровня вариабельности ЯПФ состоит в использовании этой величины при принятии решений о возможностях (*потенциале*) преобразовании данной ЯПФ.

Общая задача близка к проблеме разбиения графов и согласно [3] относится к классу *NP*-полных. Для ИГА большого размера трудоёмкость оптимизации путем перебора запредельно велика (задача перестановок с учетом влияния изменения конфигурации ЯПФ после каждой), существующие алгоритмы разбиения графов являются эвристическими, с учетом этого правильнее говорить о нахождении не оптимального, а приближающегося к оптимальному *рационального* решения. Параметрами оптимизации служит все множество планов выполнения операторов параллельной программы (с учетом сохранения информационных связей), в качестве критерия обычно принимают максимум скорости выполнения и/или полноту использования существующего оборудования (многокритериальная оптимизация). С практической точки зрения необходимо иметь гибкий аппарат для решения задачи планирования с разной степенью оптимизации (вследствие *NP*-полноты задачи понимая эту процедуру как последовательное повышение качества эвристик в сторону улучшение критерия оптимизации) - от "быстрых" методов (основанных, напр., на использовании ЯПФ) до более трудоемких (методы "ветвей и границ", эволюционные и др.) для получения лучших решений. Практически только "быстрые" методы приемлемы для RunTime-использования в составе компиляторов, более трудоемкие эмпирики ([4] и др.) скорее подходят для лабораторных исследований (напр., при оптимизации выполнения нечасто изменяемых алгоритмов).

Ограничения МВС - число параллельно работающих вычислителей, объемы их памяти и функциональные возможности, параметры коммуникационной среды; выполнение программы с учётом этого достигается эквивалентными (не изменяющими информационных зависимостей в графе) преобразованиями ЯПФ-разбиений ИГА.

Будем использовать следующие параметры ЯПФ графа:

- H - глубина (*высота*) ЯПФ графа, определяется количеством ярусов
- W_i - ширина *i*-того яруса (количество операторов на нём)
- W - ширина ЯПФ графа, определяется максимумом операторов по всем ярусам; $W = max(W_i)$
- $\bar{W} = \frac{1}{N}\sum W_i$ - средняя ширина ЯПФ, $\boldsymbol{i}$=1-$\boldsymbol{N}$, где $\boldsymbol{N}$ - число ярусов ЯПФ
- k - степень (коэффициент) неравномерности распределения операторов по ярусам ЯПФ графа; $k = max(W_i)/min(W_i)$
- $\sigma=\sqrt{\frac{1}{N-1}\sum(W_i - \bar{W})^2}$ - стандартное (среднеквадратичное) отклонение ширин ярусов.

Можно выявить типовые постановки оптимизационной задачи:

а) Максимально полная загрузка неизвестного заранее (но не меньшего $\bar{W}$) числа параллельно работающих вычислителей имеющейся МВС при минимальном времени выполнения задачи: $k \to 1{,}0$ при $H = H_0 = const$ (где H_0 - длина критического пути - минимальная из всех возможных высот ЯПФ данного графа).

б) Наиболее полное использование ресурсов (вычислительных узлов) имеющейся МВС при допущении увеличения времени выполнения (отход от канонической ЯПФ): $max(W_i) \le W_0$ при $H \ge H_0$ (здесь W_0 - число доступных параллельно работающих вычислителей данной МВС); однако желательно $H - H_0 \to min$ (минимизация времени выполнения). Может быть полезно представить этот процесс как подготовку состоящей из не более чем заданного числа W_0 заведомо независимых (ортогональных) по операндам связки команд для вычислительной системы архитектуры VLIW в идеологеме EPIC).

Реализационная часть

Для реализации и оптимизации эвристических алгоритмов поиска рациональных (стремящихся к оптимальным) планов выполнения параллельных частей парал-

лельных программ разработана специализированное программное обеспечение инструментального уровня (*программный стенд*), в котором для описание стратегий решения задачи использован встроенный скриптовый язык программирования (ВСЯП) Lua [5]. Данный ВСЯП выбран главным образом вследствие его компактности (при его применении замечено возрастание размера исполняемого файла родительского приложения не более чем на несколько десятков kБайт) и простоты обучаемости (в основном Lua следует синтаксису языка C); широкие возможности (поддерживается объектно-ориентированная модель программирования и др.) и полная бесплатность исходных текстов только подтверждают правильность выбора. В целом возможностей Lua вполне достаточно для реализации наиболее трудоемких методов оптимизации задачи планирования. Клиентская часть программной системы (предварительное сообщение приведено в работе [6]) написана на языке программирования C++, является 32-битным GUI-приложением Windows (рис.2) и доступна для свободной выгрузки с WEB-сайта автора http://vbakanov.ru/spf@home/.

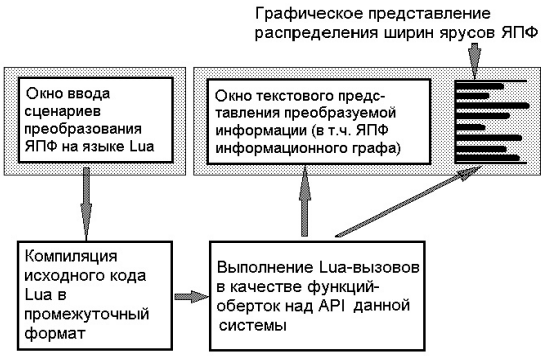


Рис. 2. Укрупненная схема взаимодействия компонентов разработанного *программного стенда* и принципы функционирования пользовательского интерфейса системы.

Набор API-вызовов разработанной системы позволяет реализовывать любой из возможных критериев оптимизации (а также их комбинации), т.к. выбор критерия осуществляет собственно пользователь (на основе задач, им поставленных).

Программный интерфейс рассматриваемой системы включает три типа API-вызовов (всего до 30 штук, причём каждый является Lua-"оберткой" соответствующей C-функции):

- Информационные (служат для получения информации об ИГА и его ЯПФ; именно на основании этих данных в дальнейшем принимается решение о применимости одной из стратегий обработки ИГА для решения поставленной задачи, реализуемой в дальнейшем на ВСЯП).
- Акционные (служат для реализации конкретных стратегий решения задачи построения расписания).
- Вспомогательные (вывод рассчитанных данных в текстовом и графическом виде для обмена данными с иными приложениями, работа с файловой системой и т.п.).

Примерами информационных вызовов являются - получить общее число вершин и дуг ИГА, смежную вершину по заданной, общее число ярусов ЯПФ и число операторов на заданном ярусе, диапазон ярусов допустимого расположения каждого оператора, характер распределение ширин ярусов ЯПФ, число и параметры параллельных вычислителей и др., акционных – создать ЯПФ по указанному ИГА (реализовано в виде отдельного API-вызова), переместить заданный оператор на конкретный ярус ЯПФ (с учетом ненарушения информационных зависимостей), создать новый пустой ярус, уничтожить пустой ярус и др.; выполнение API-функций подробно протоколируется в файле с уникальным именем. Данные расчетов выдаются как в текстовом, так и в графическом (ленточная диаграмма распределения операторов по ярусам ЯПФ) виде и могут быть сохранены в распространенных форматах.

В случае применение данной системы в качестве компоненты распараллеливающего компилятора граф алгоритма априорно существует (компилятор при работе производит анализ программы на информационные зависимости - т.е. фактически строит граф). Автор для получения корректного ИГА ряд лет использует программный симулятор вычислителя потоковой (DATA-FLOW) архитектуры, в котором программа отлаживается (не только в смысле учета информационных зависимостей, но и количественного решения) и в дальнейшем экспортируется в файловый формат списка смежных вершин. Особенностью этого си-

мулятора является возможность моделирования асинхронного режима выполнения операторов (для каждого задается время исполнения в условных единицах).

Одна из типовых задач – определение эффективных стратегий балансировки ЯПФ информационного графа алгоритма (здесь под балансировкой понимается равномерность распределения операторов по ярусам ЯПФ). На первой стадии из всех реалий конкретной МВС учитывается только число параллельных вычислителей; разработанные методы учета гетерогенности (по параметрам объема памяти, возможностей вычислений с различными типами данных, векторных вычислений и т.п.) вычислительного поля с возможностью выбора выполнения определенного оператора на соответствующем по параметрам вычислителе потребовали дополнения системы приблизительно 15 API-вызовами. Предложена компактная схема дополнения каждого (или группы) операторов и экземпляров вычислительного поля тегами, позволяющими делать выводы о возможности выполнения конкретного оператора на данном вычислителе.

Дополнительные теги призваны информировать о мерах дуг ИГА, причем физическая интерпретация этих величин отдается конкретному разработчику - он может нагрузить их размером передаваемых данных или же временем их передачи.

Для случая гетерогенности поля вычислителей разработана стратегия, дополняющая ЯПФ разбиением ярусов ЯПФ на подъярусы. Если $\boldsymbol{O}_{\mathbf{type}} \leq \boldsymbol{C}_{\mathbf{type}}$, то $\boldsymbol{O}$ операторов типа ***type*** выполняются параллельно на $\boldsymbol{C}$ вычислителях того же типа, при большем числе операторов они выполняются вынужденно последовательно на подъярусах (рис.3). Задача балансировки выполнения по подъярусам здесь также актуальна и решается в процессе распределения операторов по подъярусам.

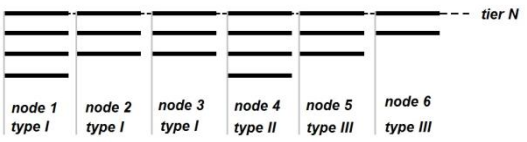
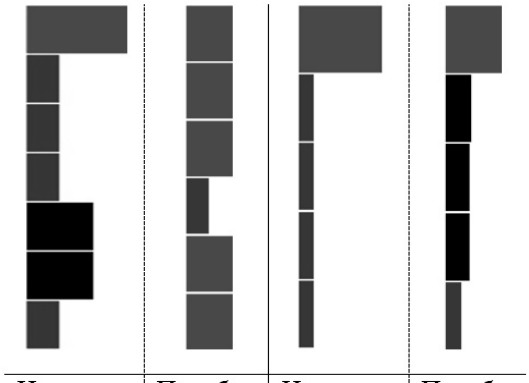


Рис. 3. Выполнение параллельной программы на гетерогенном поле вычислителей согласно схемы расцепления ярусов: всего 6 параллельных вычислителей (*node*) трех типов - *type I, type II* и *type III* по 2, 1 и 3 штуки соответственно для *N*-ного яруса ЯПФ.

В качестве *параметра эффективности* собственно стратегии преобразования ЯПФ предлагается использовать (аналогично задачам сортировки) число перестановок операторов с яруса на ярус для получения заданного результата (минимизируется), превышение высоты ЯПФ над величиной критического пути удобно рассматривать в качестве штрафной функции.

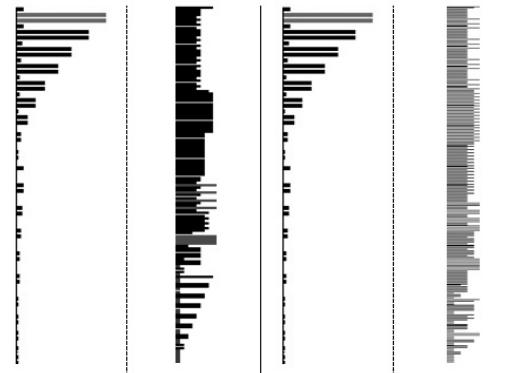
Экспериментальная часть

На рис.4 в графической интерпретации (показаны ленточные диаграммы распределения числа операторов по ярусам) приведены результаты применения различных стратегий (обе относятся к классу "быстрых" методов) преобразования ЯПФ для двух случаев - стратегии #01 (перемещение операторов с наиболее нагруженных на наименее нагруженные яруса при условии сохранения высоты ЯПФ, критерий остановки алгоритма – перебор всех операторов, могущих быть перенесенными “с холмов во впадины” с целью балансировки ЯПФ) и стратегии #02 (балансировка при условии увеличения высоты ЯПФ, критерий останова – “ужатие” (снижение ширины ЯПФ) до величины, не превышающей заданной) для нескольких ИГА распространенных вычислительных процедур различной сложности.



Исходная ЯПФ (H=6; W=4; k=4; σ=1,07; $\overline{W}$=1,83; CV=0,58)	Преобразованная ЯПФ (H=6; W=2; k=2; σ=0,373; $\overline{W}$=1,83; CV=0,2)	Исходная ЯПФ (H=5; W=125; k=5; σ=40; $\overline{W}$ =45; CV=0,89)	Преобразованная ЯПФ (H=5; W=85; k=3,8; σ=25,3; $\overline{W}$=45; CV=0,56)
а) Вычисление вещественных корней полного квадратного уравнения		б) Умножение квадратных матриц пятого порядка классическим методом	

Преобразование ЯПФ согласно стратегии #01 (условие H=*const*)



Исходная ЯПФ (H=63; W=90; k=90; σ=21,8; $\overline{W}$=12,8; CV=1,7)	Преобразованная ЯПФ (H=141; W=10; k=10; σ=2,69; $\overline{W}$=5,71; CV=0,47)	Исходная ЯПФ (H=63; W=90; k=90; σ=21,8; $\overline{W}$=12,8; CV=1,7)	Преобразованная ЯПФ (H=230; W=5; k=5; σ=1,15; $\overline{W}$=3,50; CV=0,33)
в) Решение системы линейных уравнений (СЛАУ) десятого порядка методом Гаусса (W_0=10)		г) Решение СЛАУ десятого порядка методом Гаусса (W_0=5)	

Преобразование ЯПФ согласно стратегии #02 (условие W<=W_0)

Рис. 4. Диаграммы распределения операторов по ярусам для исходных и преобразованных ЯПФ с помощью различных стратегий преобразования для разных алгоритмов (вариант гомогенного поля параллельных вычислителей).

Видно, что примененные стратегии преобразования ЯПФ снижают *CV* конкретных задач в 2-5 раз для случаев рис.4 а-г) соответственно.

Эффективность обработки данных также продемонстрирована в работах [6,7] применением стратегий #01 и #02 к информационных графам распространенных вычислительных процедур: решение СЛАУ порядков 2, 3, 5 и 10 соответственно классическим методом Гаусса, линейная аппроксимация по методу наименьших квадратов для 10 и 20 точек, определение коэффициента парной корреляции для 10 и 20 точек, умножение квадратных матриц порядков 2, 3, 5 и 10 классическим методом. Неравномерность распределения числа операторов по ярусам ЯПФ оценивалась величинами k и σ.

Следует заметить, что потенциал "балансировки" ЯПФ при неизменной его высоте ограничен, поэтому больший практический интерес представляют стратегии с увеличением высоты ЯПФ.

Моделирование позволило определить эффективность разработанных методик - напр., зависимость трудоемкости преобразований ЯПФ (в единицах перестановок операторов с яруса на ярус) от заданного предела ширины ЯПФ для графов распространенных алгоритмов и созданным программным графогенератором близка к полиномиальной функции от степени "ужатия" ширины ЯПФ.

Автор считает потенциально эффективным применение технологий искусственного интеллекта для анализа параметров исходного графа (выполняется описанными ранее информационными API-вызовами), применения определенных правил вывода для обработки встретивший ситуации и далее преобразования представлений ИГА с помощью акционных API-вызовов (предварительное сообщение [7]).

В этом случае процесс (итерационный в общем случае) состоит из этапов получения (с помощью информационных функций) данных об ИГА (в частности, об его ЯПФ),

распознавания ситуации и выбора эффективной стратегии построения плана выполнения параллельных частей программы для данного ИГА и выполнения Lua-скриптов (с возможной их модификацией), реализующих выбранную стратегию построения этого плана.

Заключение

Проведенные вычислительные эксперименты показывают, что ранее описанные (однако же не являющимися излишне изощренными) стратегии в самом деле позволяют снизить неравномерность ширин ЯПФ (а значит, и уровень дисбаланса вычислительной нагрузки данной МВС) при заданных ограничениях, однако с разной эффективность для различных ИГА. В целом наблюдается повышение эффективности стратегии при увеличении сложности (и, соответственно, вариативности) ИГА. Особенна интересна и практически важна задача априорного (до проведения собственно преобразований ЯПФ) предсказания уровня эффективности стратегий (задача распознавания ситуаций).

Б**о**льшего эффекта можно добиться путем совершенствования стратегий и разумного их совместного применения.

Применение встроенного скриптового языка программирования для моделирования методов оптимального планирования задач для параллельных вычислительных систем позволило добиться гибкости и эффективности в реализации поставленной цели, что было бы затруднительным при использовании иных подходов.

Результаты исследований применимы для анализа алгоритмов (в плане выявления эффективности выполнения параллельных программ на конкретных МВС), при разработке распараллеливающих компиляторов и в процессе обучения специалистов в области технологий параллельного программирования.

Литература

1. *Воеводин В.В., Воеводин Вл.В.* Параллельные вычисления. — СПб.: БХВ-Петербург, 2002. — 608 с.
2. *AlgoWiki*. Открытая энциклопедия свойств алгоритмов. [Электронный ресурс]. Дата обновления: 18.10.2016. URL: http://algowiki-project.org (дата обращения: 21.01.2017).
3. *Гери М., Джонсон Д.* Вычислительные машины и труднорешаемые задачи. — М.: Мир, 1982. — 416 с.
4. *Гаврилюк А.Б., Алексеев В.А*. Метод оптимального статического планирования задач в распределенных вычислительных системах с использованием генетического алгоритма. // Проблемы программирования. — Киев: 2004. № 1, с. 52-59.
5. *Иерузалимски Роберту.* Программирование на языке Lua. — М.: ДМК Пресс, 2014. — 382 с.
6. *Баканов В.М.* Программный инструментарий анализа информационной структуры алгоритмов по их информационным графам. // Параллельные вычислительные технологии (ПаВТ’2016): труды международной научной конференции (г. Архангельск, 28.03–01.04.2016). — Челябинск: Издательский центр ЮУрГУ, 2016, с. 432-441.
7. *Баканов В.М.* Построение рациональных планов выполнения параллельных частей программ и применение методов искусственного интеллекта для решения подобных задач. // 15-я конференция по искусственному интеллекту с международным участием КИИ-2016 (CAI-2016, г. Смоленск, 03-07.10.2016): труды конференции. — Смоленск, филиал НИУ МЭИ в Смоленске, 2016, т. 3, с. 237-244.