А.Р. Хайрутдинов

МЕТОД РЕШЕНИЯ ЗАДАЧИ ОТОБРАЖЕНИЯ ДЛЯ ШАБЛОНОВ СИСТЕМЫ TEMPLET

(Самарский государственный аэрокосмический университет)

На сегодняшний день спектр решаемых с помощью параллельного программирования задач растёт. Между тем, остро строит проблема создания корректных и максимально эффективных программ. Она обусловлена как относительной новизной области, так и наличием качественно других эффектов, присущих именно параллельному выполнению алгоритмов: состоянию гонки, появлению тупиков, необходимости синхронизации и организации монопольного доступа к данным. Всё это влечёт за собой многократное усложнение поиска ошибок и выполнения отладки. Неудивительно, что предпринимаются попытки упрощения процесса разработки, и распространённым подходом к этому является использование технологии визуального программирования.

Существует достаточно большое число представителей данного направления. Одним из них является программный комплекс Templet [1], предназначенный для создания и использования типовых решений («темплетов») при разработке параллельных программ. На текущий момент в состав набора разработчика входит ряд шаблонов, применяемых для решения различных задач, в том числе схема конвейерной организации вычислений *Pipeline* [2].

Время исполнения параллельной программы зависит от многих факторов: адекватного выбора алгоритма, эффективности реализации на языке программирования, производительности вычислительного комплекса, распределения задач по вычислительным ресурсам, использования балансировки (статической или динамической) и других. Однако на практике наибольшую неопределённость вызывают последние два. Действительно, для большинства задач алгоритмы решения известны, и, как правило, существуют их корректные реализации. С другой стороны, от неудачного выбора схемы распределения ресурсов производительность программы может сильно пострадать. Это получило название «проблемы отображения» – задачи установления такого соответствия между исполняемыми единицами программ и имеющимися в наличии вычислительными ресурсами, чтобы итоговая программа наиболее эффективно использовала их. Анализ при помощи представления параллельного алгоритма в виде графа «операции-операнды» [3] позволяет дать лишь некоторые оценки характеристик, однако не отвечает на вопрос об оптимальном размещении. Таким образом, встаёт задача о поиске метода, с помощью которого можно было бы эффективно решить «проблему отображения».

Зависимость времени выполнения от схемы использования ресурсов (конфигурации) установить в явном виде невозможно. Поэтому уместным является использование имитационного моделирования, которое позволит, изменяя изначальную конфигурацию модели, получить на выходе характеристики моделируемой программы.

Возможны два подхода к созданию имитационной модели. Первый основан на особенности программной модели Templet. Процессы в ней являются процессами диффузного типа, другими словами, управляющий алгоритм отделён от прикладного кода и используется в виде подключаемой библиотеки времени исполнения. Если воспользоваться специально написанным модулем, реализующим имитацию реальной работы и собирающим информацию для монитора моделирования, то, таким образом, имитационную модель можно органично вписать в существующую программу. Такая библиотека времени исполнения реализована для конвейерной схемы [4].

Второй подход заключается в реализации имитационной модели в инструментальном средстве или на языке программирования. Задача моделирования многопроцессорных систем не нова. Существуют две парадигмы: моделирования системы с общей памятью и моделирования обменивающихся сообщениями вычислительных ядер – и модели вычислительных систем разной степени детализации (*SRAM*, *LogP*) [5]. В предлагаемом автором методе используется упрощённая модель, которая, однако, показала себя адекватной в дальнейших экспериментах.

Объекты программной модели Templet – процессы, каналы связи, сообщения – очевидным образом становятся объектами имитационной модели. Входные параметры модели:

- относительная производительность каждого узла – отношение производительности текущего узла к производительности самого вычислительно слабого;
- время исполнения задачи на самом малопроизводительном узле, мкс;
- размер пересылаемых сообщений, байт;
- пропускная способность каждого канала связи, байт/мкс;
- средняя латентность каждого канала связи, мкс;
- количество моделируемых итераций, шт.

Монитор моделирования собирает информацию о загруженности каждого узла в каждый моделируемый момент времени.

Допущения модели:

- все вычислительные задачи, начав своё исполнение на каком-либо ресурсе, заканчивают исполняться на нём, то есть отсутствует миграция запущенных задач;
- все вычислительные задачи имеют одинаковую сложность, то есть время их исполнения на одном и том же ресурсе примерно одинаково;

- связи между процессами представляются прямыми линиями связи без учёта их физических особенностей, например, наличия коммутаторов;
- по каналам связи передаются только сообщения, то есть накладные расходы на протокол передачи отсутствуют.

Допущение об одинаковой сложности задач может показаться слишком сильным, однако для рассматриваемой конвейерной схемы именно так и происходит на практике. Допущения, связанные с игнорированием физической структуры каналов связи и затрат на протокол, можно очевидным образом нивелировать заданием меньшей пропускной способности и большего размера сообщений.

Разработанная модель допускает задание алгоритмов балансировки (для этого их нужно реализовать в программном коде), однако в [6] показано, что статическая балансировка даёт приемлемый результат, поэтому в дальнейшем динамическая и комбинированная балансировки не использовались.

Для проверки адекватности разработанной модели были проведены несколько экспериментов, отличающихся начальной конфигурацией. Их суть состояла в сравнении измеренного реального времени выполнения программы с временем, полученным в результате моделирования. Выполняемой программой являлся численный алгоритм расчёта распространения плоской волны в волноводе, описанный в [7] и реализованный с использованием схемы *Pipeline*. Для большей достоверности время выполнения реальной программы измерялось в течение 30 запусков, и в качестве результата бралось среднее арифметическое полученных значений. Программа исполнялась в операционных системах (ОС) *Windows Vista SP2* и *Ubuntu 12* на компьютере с процессором *Intel Core 2 Duo T5750 @2.00GHz*, имеющим два ядра.

Результаты экспериментов приведены в таблице 1. Где n – количество процессов типа *Center*; $t_{реальн\ W}$ и $t_{реальн\ U}$ – среднее за 30 запусков время выполнения программы на ОС *Windows* и *Ubuntu* соответственно; $t_{им}$ – время выполнения, полученное моделированием; $t_{мод}$ – время моделирования. Конфигурация в эксперименте 1 – все процессы выполнялись на одном ядре; конфигурация в эксперименте 2 – процессы типа *Left* и *Right* выполнялись на одном ядре, а все процессы типа *Center* – на другом.

Таблица 1. Результаты проведения имитационного эксперимента

Эксперимент	n, шт.	$t_{реальн\ W}$, с	$t_{реальн\ U}$, с	$t_{им}$, с	$t_{мод}$, с
1	1	3,61	3,70	3,52	7,54
	3	4,70	4,86	4,11	10,25
2	1	2,17	2,34	2,06	9,98
	3	2,99	3,01	2,79	13,04

Согласно таблице 1, можно сделать вывод о том, что разработанная модель адекватно отображает реальный процесс. Следует также отметить, что с

увеличением количества элементов в модели время моделирования заметно возрастает.

Моделирование позволяет получить прогноз времени исполнения при заданной конфигурации, однако не отвечает на вопрос о том, является ли эта конфигурация наилучшей с точки зрения итоговой производительности программы. Таким образом, возникает оптимизационная задача минимизации целевой функции, представляющей собой зависимость времени выполнения программы от конфигурации вычислительных ресурсов. Пусть имеется P процессов и R ресурсов. Вектор $K = (k_1, k_2, \dots k_P)$, где $k_i \in [1, 2, \dots R]$, будет начальной конфигурацией, а $T_{опт} = \min T(K)$ – минимизируемой функцией.

Анализ функции $T(K)$ не представляется возможным, поэтому поиск наилучшего решения может быть осуществлён полным перебором. Если число процессов мало, то такое решение осуществимо, однако с увеличением размерности время поиска растёт экспоненциально. Поэтому встаёт вопрос об использовании другого метода оптимизации.

Автор предлагает воспользоваться аппаратом генетических алгоритмов [8], хорошо зарекомендовавших себя в задачах оптимизации, в которых целевая функция и её структура априорно неизвестны.

Использована классическая схема генетического алгоритма:

1) сгенерировать популяцию;
2) вычислить значение целевой функции для каждой особи;
3) завершить работу, если достигнут критерий остановки;
4) выполнить селекцию;
5) выполнить скрещивание;
6) выполнить мутацию;
7) сформировать новую популяцию;
8) перейти к шагу 2.

Пространство всех векторов конфигурации образует пространство поиска решения, а каждый вектор кодирует особь. Параметрами алгоритма являются:

- число итераций;
- размер популяции;
- стратегия формирования новой популяции;
- функция селекции;
- функция скрещивания;
- функция мутации;
- функция кроссовера.

Критерием остановки поиска является достижение заданного числа итераций. Начальная популяция генерируется случайным образом: каждая компонента k_j векторов конфигурации принимает значение, полученное розыгрышем дискретной случайной величины, равномерно распределённой на интервале $[1, R]$. Функция селекции – целевая функция оптимизационного поиска $T(K)$.

От выбора функций скрещивания, мутации и кроссовера – операторов генетического алгоритма – зависит скорость и качество процесса выполнения

поиска. Существует большое количество возможных операторов [8]. Для описываемой задачи применимыми оказались даже наиболее простые из них.

Система скрещивания – панмиксия: особи для скрещивания выбираются случайно с одинаковой вероятностью. Стратегия формирования новой популяции – элитная поколенческая репродукция: в новое поколение попадают N_r «наилучших» особей из родительского поколения; N_r принято равным ¼ от размера популяции. Функция мутации – точечная мутация: мутации подвергается случайный ген, особь-мутант выбирается случайно с вероятностью p_m=0,05. Функция кроссовера – одноточечный кроссовер: из векторов K_i и K_b, являющихся геномами особей, образующих «брачную пару», формируются два генома потомков $K^1=(k_{i,1},\ldots, k_{i,r}, k_{b,r+1},\ldots, k_{b,P})$ и $K^2=(k_{b,1},\ldots, k_{b,r}, k_{i,r+1},\ldots, k_{i,P})$, где r – точка разрыва, выбираемая случайно.

Результаты сравнения времени работы переборного алгоритма с описанным генетическим алгоритмом приведены в таблице 2. Значения целевой функции были рассчитаны заранее, поэтому время их вычисления не входит во время поиска решения. P – число процессов, R – количество вычислительных узлов, $t_{переб}$ – время работы алгоритма полного перебора, $n_{ит}$ – количество итераций генетического алгоритма, $t_{ген}$ – время работы генетического алгоритма. Расчёты выполнялись на компьютере с процессором *Intel Core 2 Duo T5750 @2.00GHz*.

Таблица 2. Сравнение алгоритма полного перебора и генетического алгоритма при использовании заранее рассчитанных значений целевой функции

P, шт.	R, шт.	$t_{переб}$, с	$n_{ит}$, шт.	$t_{ген}$, с
3	4	0,06	20	0,11
5	4	0,57	100	0,49
3	12	2,31	100 000	38,52
5	12	7,45	100 000	38,06

Таблица 2 подтверждает то, что генетические алгоритмы устойчивы к увеличению размерности задачи.

На первый взгляд может показаться странным выбор их в качестве метода решения с учётом того, что время выполнения переборного алгоритма получилось значительно меньше. Однако в действительности переборный алгоритм оказался лучше лишь потому, что значения целевой функции были рассчитаны заранее, и фактически переборный алгоритм превратился в алгоритм поиска минимального элемента в наборе данных.

В процессе его выполнения целевая функция должна быть рассчитана P^R раз. Так как этот расчёт представляет собой процесс моделирования, описанный выше, то время работы алгоритма значительно возрастает при отсутствии заранее рассчитанных значений. Это иллюстрирует таблица 3, в которой приведены данные эксперимента, где каждое запрашиваемое значение целевой функции, если оно не было уже рассчитано, вычислялось в результате нового процесса моделирования.

Таблица 3. Сравнение алгоритма полного перебора и генетического алгоритма без использования заранее рассчитанных значений целевой функции

P, шт.	R, шт.	$t_{переб}$, с	$n_{ит}$, шт.	$t_{ген}$, с
3	4	267	20	613
3	5	978	100	2523
5	4	2002	100	2536
5	5	9399	100	2538

Таким образом, в работе предложен метод анализа временных характеристик «темлетов» для решения «задачи отображения», состоящий в совместном использовании имитационного моделирования (оценка времени выполнения программы) и генетического алгоритма (оптимизация). Метод используется для решения задачи поиска оптимальной конфигурации, которая обеспечивает наименьшее время выполнения программы. Он прошёл экспериментальную проверку на примере шаблона *Pipeline*, применим в случаях большого числа элементов в схеме и позволяет получить решение за приемлемое время.

Использованная литература:

1. Востокин, С.В. Программный комплекс параллельного программирования Graphplus templet/ С.В. Востокин, А.Р. Хайрутдинов, В.Г. Литвинов // Вестник Самарского государственного технического университета. Серия: Физ.-мат. науки. – 2011. – №4 (25). – С. 146–153.
2. Востокин, С.В. Применение комплекса параллельного программирования Graphplus templet в моделировании / С.В. Востокин, В.Г. Литвинов, А.Р. Хайрутдинов // Программные продукты и системы. – 2012. – №3 (99). – С. 12–16.
3. Гергель, В.П. Теория и практика параллельных вычислений: учеб. пособие/ В.П. Гергель. – М.: Интернет-Университет Информационных технологий; БИНОМ. Лаборатория знаний, 2007. – 423 с.
4. Востокин, С.В. Имитационная модель параллельных вычислений в парадигме конвейер / С.В. Востокин, А.Р. Хайрутдинов // Свидетельство о государственной регистрации программы для ЭВМ. Рег. № 2014613431 от 26 марта 2014 г.
5. Sinnen, O. Task Scheduling for Parallel Systems / O. Sinnen. – New Jersey: John Wiley & Sons, 2007. – 296 pp.
6. Хайрутдинов, А.Р. Исследование алгоритмов балансировки методом дискретно-событийного моделирования / А.Р. Хайрутдинов, С.В. Востокин // Вестник Самарского государственного аэрокосмического университета имени академика С.П. Королёва (национального исследовательского университета). – 2011. – №6 (30). – С. 283–288

7. Головашкин, Д.Л. Анализ прохождения электромагнитного излучения через дифракционную линзу / Д.Л. Головашкин, В.А. Сойфер // Автометрия. – 1999. – №5. – С. 119–121.
8. Батищев, Д.И. Применение генетических алгоритмов к решению задач дискретной оптимизации. Учебно-методический материал по программе повышения квалификации «Информационные технологии и компьютерное моделирование в прикладной математике» / Д.И. Батищев, Е.А. Неймарк, Н.В. Старостин. – Нижний Новгород, 2007. – 88 с.