УДК 004.432.42:004.272.2

УВЕЛИЧЕНИЕ ПРОИЗВОДИТЕЛЬНОСТИ ТРАНСЛЯТОРА ЯЗЫКА ЛИСП БЛАГОДАРЯ ИСПОЛЬЗОВАНИЮ ПАРАЛЛЕЛЬНЫХ ВЫЧИСЛЕНИЙ

Савков К.Г., Дацун Н.Н.
Донецкий национальный технический университет

Рассматривается создаваемый транслятор языка Лисп, увеличивающий производительность, благодаря использованию параллельных вычислений. Этот транслятор решает проблему автоматического распараллеливания выполнения функциональных программ. Основной целью создания стало исследование возможных путей увеличения производительности транслятора, таких как анализ текста программ с целью определения возможности параллельного выполнения, исследование механизма управления выполнением параллельных программ, а также анализ результатов их выполнения.

Язык Лисп выбран ввиду своей специфики. Созданный Дж. Маккарти в 1962 году теоретический «чистый» Лисп|1| имел очень ограниченный набор функций для манипуляций со списками, но был задуман как теоретическое средство для рекурсивных построений. Кроме математической ясности и предельной четкости Лисп позволяет сформулировать и запомнить «идиомы», характерные для проектов по искусственному интеллекту. Функциональный стиль написания программ отлично подходит к решению математических и логических задач, построению экспертных систем. Кроме этого, Лисп поддерживает очень высокий уровень абстракции. Одной из особенностей языка является одинаковое представление данных и программ, что особенно полезно при генерации кода в CASE-системах.

Создаваемый транслятор не обладает обширной базой библиотек, но их можно подключать, а существующих библиотек вполне достаточно для использования численных методов, экспертных систем и математической логики [2]. Являясь интерпретируемым языком, он не привязан к конкретной аппаратуре, что делает его более гибким. Сама идея параллельного выполнения Лисп-программ не нова, на сегодняшний день существует множество трансляторов, выполняющих параллельные программы. Единый подход к параллельному выполнению появился в Multilisp как способ определения «ленивых» вычислений - future. Однако он требовал непосредственного указания параллелизма в тексте программы. QLisp расширяет использование отложенных вычислений до целой библиотеки [3]. Программы, написанные в функциональном стиле без побочных эффектов, могут быть распараллелены без каких-либо указаний со стороны программиста, однако все реализации Лиспа требуют непосредственное указание параллельности. Таким образом, возникает проблема рефакторинга старых программ. Именно эту проблему и призван решить создаваемый транслятор. Чистые функциональные программы как таковые переменных не имеют, но для облегчения написания программ в Лиспе существуют аналоги переменных - свойства атомов. Транслятор параллельно выполняет несколько процессов при невмешательстве одних процессов в другие. В качестве техники устранения взаимного вмешательства используется модификация метода непересекающегося множества переменных. Управление выполнением параллельной программы основано на парадигме портфеля задач.

1. Непересекающиеся множества переменных

Множество записи процесса - это множество переменных, которым он присваивает значения (и, возможно, считывает их), а множество чтения процесса - это множество переменных, которые процесс считывает, но не изменяет. Множество ссылок процесса - это множество переменных, которые встречаются в утверждениях доказательства корректности данного процесса. Если множество записи одного

процесса не пересекается со множеством ссылок другого процесса и наоборот, то эти два процесса не могут влиять друг на друга [4].

На основании данного метода базируется механизм определения параллельности выполнения Лисп-программ. Модификацией служит использование множества атомов и их свойств вместо переменных. В том случае, когда множества используемых атомов в S-выражениях не пересекаются, S-выражения могут быть выполнены параллельно. Множество используемых атомов определяется двумя способами - путем анализа исходного текста (априорным) и путем анализа хода выполнения (апостериорным).

2. Портфель задач

Используемая в существующих реализациях парадигма параллельного выполнения Лисп-программ - портфель задач. Задача является независимой единицей работы. Задачи помещаются в портфель, разделяемый несколькими рабочими процессами. Каждый рабочий процесс получает задачу из портфеля, выполняет ее, возможно порождая новые задачи в портфеле. Обработка заканчивается, когда портфель пуст. Этот подход можно использовать для реализации рекурсивного параллелизма; тогда задачи будут представлены рекурсивными вызовами. Его также можно использовать для решения итеративных проблем с фиксированным числом независимых задач.

Парадигма портфеля задач имеет несколько полезных свойств. Во-первых, она весьма проста в использовании. Достаточно определить представление задачи и выяснить, как распознается завершение работы алгоритма. Во-вторых, программы, использующие портфель задач, являются масштабируемыми в том смысле, что их можно использовать с любым числом процессоров; для этого достаточно изменить количество рабочих процессов. Однако производительность программы при этом может не измениться. Эта парадигма упрощает реализацию балансировки нагрузки. Если длительности выполнения задач различны, то, вероятно, некоторые из задач будут выполняться дольше других. Но пока задач больше, чем рабочих процессов, общие объемы вычислений, осуществляемых рабочими процессорами, будут примерно одинаковыми. Эта парадигма используется практически во всех реализациях Лиспа, однако сам вид задачи скрыт от пользователя внутренними представлениями и структурами данных [5]. В трансляторе реализовано представление задачи как текста программы, что значительно упрощает отладку. Хотя преобразование текста программы во внутреннее представление и несет в себе накладные расходы, оно не изменяет сложность трансляции, так как она остается линейной, просто с другим коэффициентом. Изолированная задача для параллельного исполнения представлена в том же виде, в каком она была написана программистом, таким образом, это наглядный пример преобразования программы в результате выполнения.

3. Адаптивное распараллеливание

Практическая возможность параллельного выполнения некоторых частей программы не всегда целесообразна, так как расходы на передачу задачи другому процессору могут превышать время ее выполнения. Определение целесообразное передачи задачи другому процессору зависит от множества факторов, в том числе и от состояния системы в целом. Использование парадигмы портфеля задач позволяет балансировать нагрузку на процессоры, однако все это не имеет практического смысла, если не дает выигрыша в скорости выполнения. Показатели скорости, а точнее времени выполнения задачи, также могут быть определены априорно и апостериорно. Априорный способ определения времени выполнения задачи основан на зависимости времени выполнения задачи от ее размера[6]. К ним относятся линейное, квадратичное, логарифмическое, кубическое и экспоненциальное время выполнения.

Учитывая примененный способ представления задачи, для передачи по сети не пригодны простейшие задачи логарифмического и линейного времени выполнения. Тем не менее, если это многопроцессорная машина с разделяемой памятью, и передавать задачу по сети не нужно, она вполне может быть выполнена параллельно с выигрышем во времени.

Кроме непосредственного применения программы как транслятора языка, она может использоваться для анализа выполнения рекурсивных программ. Дальнейшее применение этой программы позволит увеличить производительность уже существующего программного обеспечения, написанною на языке Лисп, а также выявить элементы программ, подлежащие распараллеливанию в теории и на практике. Стоит отмстить, что, добавив необходимые библиотеки, транслятор может использоваться практически везде, явным тому подтверждением служит Common Lisp. Применение рекурсивных программ не ограничивается численными методами или экспертными системами, при помощи них решаются важные задачи искусственного интеллекта и обрабатываются графические изображения. Несмотря на то, что интерес к Лиспу как к языку упал в последнее время, более современные языки (кроме произошедших от него) до сих пор не могут сравниться с Лиспом по оригинальности подхода. На сегодняшний день в программировании неуклонно растет уровень абстракции - универсальных, полиморфных объектов, агентов. При этом многие идеи были уже давно реализованы в Лиспе. А сегодняшние тенденции на рынке процессоров к многоядерности лишний раз подчеркивают важность функционального стиля написания программ без побочных эффектов.

Литература

[1] McCarthy J. *Recursive functions of symbolic expressions and their computation by machine, part I.* Cambridge, MIT, 1960.

[2] Лавров С.С., Силагадзе Г.С. *Автоматическая обработка данных. Язык Лисп и его реализация*. - М: Наука, 1978.

[3] Goldman Ron, Richard P. Gabriel, *Qlisp: Parallel Processing in Lisp*. IEEE Software, vol. 06, no. 4, pp. 51-59, Jul/Aug, 1989.

[4] Эндрюс Г. *Основы многопоточного, параллельного и распределенного программирования.* - М; Вильямс, 2003.

[5] Хьюз К., Хьюз Т. *Параллельное и распределенное программирование с использованием C++*. -М.: Вильямс, 2004.

[6] Седжвик Р. Фундаментальные алгоритмы на С. Анализ/Структуры данных/ Сортировка/Поиск/Алгоритмы на графах - СПб; ООО "ДиаСофтЮП", 2003.