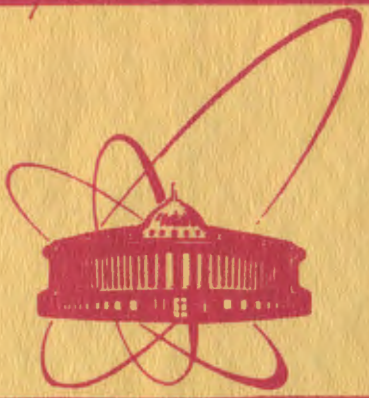

объединенный
институт
ядерных
исследований
дубна

Р11-81-752

С.Г.Каданцев, В.А.Ростовцев

ВИРТУАЛЬНАЯ ПАМЯТЬ ДЛЯ ФУНКЦИЙ В ЛИСПЕ

Направлено на Европейскую конференцию
по алгебраическим преобразованиям на ЭВМ
/Марсель, апрель 1982 г./

1981

1. ВВЕДЕНИЕ

В [1] мы бегло описывали работу по использованию вторичной памяти в системе REDUCE–2, проводимую в Лаборатории вычислительной техники и автоматизации ОИЯИ. В настоящем препринте эта тема рассматривается более детально, с учетом достигнутого к настоящему времени уровня и возможного дальнейшего развития.

Система программирования алгебраических преобразований REDUCE–2 используется в ОИЯИ на вычислительных машинах CDC-6500, ЕС-1040 и, с октября 1981 года, ЕС-1060. На машинах серии ЕС применяется версия REDUCE–2 от 15 апреля 1979 года под управлением операционной системы (ОС)ЕС в режиме мультипрограммирования с переменным числом задач. На машине CDC-6500 система REDUCE–2 работает под управлением операционной системы NOS BE 1.0 и базируется на адаптированной к этой операционной системе версии ЛИСП-интерпретатора UT LISP 4.1 [1,2]. До апреля 1981 года использовалась версия системы REDUCE–2 от 15 апреля 1979 года. Рабочий вариант ее был сгенерирован из "Общей версии"[3]. Внедрение системы REDUCE–2 на машинах серии ЕС проводилось Р.Н.Федоровой, а на машине CDC-6500 - В.А.Ростовцевым.

Как было отмечено в [1] операционная система на нашей машине CDC-6500 выделяет для задачи пользователя максимально 48К слов оперативной памяти. Этого недостаточно для системы REDUCE–2. Для разрешения этого противоречия мы пользовались простейшим средством - изменением с пульта параметра в операционной системе, определяющего размер памяти, предоставляемой задаче. Конечно, эти манипуляции с операционной системой недоступны обычному пользователю, поэтому система REDUCE на машине CDC-6500 почти не применялась физиками в их исследованиях. В связи с этим мы решили использовать имеющиеся в интерпретаторе UT LISP 4.1 средства для хранения интерпретируемых функций на диске и динамического возврата их в оперативную память по обращению [4]. В этом случае необходимость работать с системой REDUCE в режиме "чистой" интерпретации приводит к существенному увеличению расхода машинного времени. Тем не менее мы нашли, что эти средства могут оказаться полезными при другом подходе к их использованию. С их помощью мы вводим в систему REDUCE новый аппарат для работы пользователя с внешними файлами [9].

Предлагаемый алгоритм виртуальной памяти в ЛИСПе для компилированных функций позволяет решить поставленную задачу без чрезмерного дополнительного расхода машинного времени. Здесь мы имеем в виду машины серии CDC-6000, однако предлагаемые нами дополнения и изменения в ЛИСПе с соответствующими модификациями могут быть реализованы и на других машинах.

2. ВИРТУАЛЬНАЯ ПАМЯТЬ ДЛЯ ФУНКЦИЙ В UT LISP

В общих чертах средства виртуализации памяти для функций в UT LISP 4.1 работают следующим образом. Функция ЛИСПа DISKOUT получает в качестве одного из своих аргументов список предварительно определенных функций типа (F)EXPR. Для каждой функции из этого списка строится компактный список, соответствующий ее определяющему выражению, и через буфер заносится в некоторый файл на диске. Этот файл имеет имя VIRFN и заранее определен в самом интерпретаторе. Пользователь может изменить имя файла. С точки зрения операционной системы файл VIRFN организован как последовательный, состоящий из единственной записи. Однако в интерпретаторе моделируется произвольный доступ с пословной адресацией. Таким образом, для каждого записанного на диск определяющего выражения существует его "дисковый адрес". Соответствующий атом - имя функции получает свойство с индикатором VIRFN. Значением этого свойства является однозначно построенное информационное слово, располагающееся в пространстве полных слов. В этом информационном слове хранится признак типа функции (EXPR/FEXPR), длина /в машинных словах/ сжатого определяющего выражения и счетчик, определяющий в дальнейшем критерий удаления данной функции из оперативной памяти.

Фактическое освобождение памяти, занятой определяющим выражением такой функции, происходит либо автоматически, либо по указанию пользователя /например, с помощью функции REMPROP/.

Восстановление определяющих выражений в оперативной памяти производится интерпретатором автоматически и полностью скрыто от пользователя. А именно, если у атома, находящегося в позиции функции, интерпретатор не обнаруживает ни одного из свойств (F)EXPR/(F)SUBR, то проверяется наличие свойства VIRFN. По информации, хранящейся в слове - значении этого свойства, а также по значениям некоторых внутренних переменных подпрограмма интерпретатора DISKIN определяет возможность размещения соответствующего определяющего выражения в списке свободной памяти. Если такое размещение невозможно, то путем "удаления" соответствующего числа определяющих выражений "виртуальных"

функций освобождается необходимое место и требуемое определяющее выражение считывается с диска и помещается в свободную память. При этом предполагается, что все атомы, имевшиеся в системе в момент "виртуализации" данной функции, сохранены. Поэтому при построении списка /определяющего выражения/ указатели на атомы в этом выражении не изменяются, и переработки требуют лишь указатели на неатомные объекты. Затем в список свойств атома /имени функции/ вносится свойство (F)EXPR, значением которого является построенное определяющее выражение, и работа интерпретатора с данной функцией продолжается обычным образом. Кроме того, пользователь может вызвать определяющее выражение "виртуальной" функции в оперативную память с помощью функции ЛИСПа GETD.

Как уже упоминалось в[1], этот аппарат в интерпретаторе UT LISP 4.1 не был полностью отлажен. Кроме того, не предусматривалась возможность расширения файла виртуальных функций, созданного в одном прогоне ЛИСП-программы, при последующих прогонах. Выполнив отладку и дополнив аппарат виртуальных функций упомянутой возможностью, мы сгенерировали вариант системы REDUCE-2 без интегратора, который полностью размещается в памяти объемом 48К слов, то есть в стандартной памяти, выделяемой задаче пользователя на машине CDC-6500. Для сравнения можно отметить, что при этом пространство свободной памяти и пространство полных слов суммарно имеет объем 14256 слов. В то же время в варианте без матричного пакета и пакета высоких энергий без виртуальных функций при общем объеме выделенной оперативной памяти 64К слов под свободную память и память полных слов оставалось 9000 слов.

Однако описанный аппарат виртуальных функций вынуждает использовать систему REDUCE-2 только в режиме интерпретации. Это наряду с довольно интенсивным дисковым обменом приводит к существенному увеличению расхода машинного времени. Так, при выполнении теста системы REDUCE, имеющегося на дистрибутивных лентах, описанный виртуализованный вариант системы израсходовал приблизительно 40 мин времени центрального процессора. В то же время полностью скомпилированный вариант системы/версия от 31.03.81/ на тот же тест израсходовал чуть больше 2 мин времени центрального процессора. Не касаясь особенностей самой новой версии системы REDUCE, отметим, что новый компилятор генерирует весьма эффективный и сравнительно компактный объектный код. Благодаря этому несколько уменьшился объем оперативной памяти, занимаемой самой системой REDUCE-2. Так, сокращенный ее вариант/без матричного пакета и пакета высоких энергий/ размещается в стандартной памяти 48К слов. При этом суммарный объем свободной памяти и памяти полных слов составляет 7948 слов.

Таким образом, использование средств виртуальной памяти для некомпилированных функций в целях построения эффективного эксплуатационного варианта системы REDUCE-2 оказывается нерациональным.

3. ВИРТУАЛЬНАЯ ПАМЯТЬ ДЛЯ ДВОИЧНЫХ ПРОГРАММ

Итак, задача экономии оперативной памяти, занимаемой самой системой REDUCE-2/или любой другой большой системой программ, основанной на ЛИСПе/, продолжает оставаться актуальной. При этом, очевидно, речь должна идти о виртуализации скомпилированного кода функций ЛИСПа. Среди большого количества работ, посвященных этому вопросу, отметим как наиболее близкие нашим целям, например, работы /6,7/.

Предлагаемое нами решение в общих чертах описывается далее в настоящем препринте. Идейно оно наиболее близко примыкает к работам /7,8/. Рассматривая виртуализацию компилированных функций, мы исходили из некоторых соображений, вытекающих из нашего опыта виртуализации некомпилированных функций. Мы отказываемся от более простого, но менее гибкого варианта автоматической виртуализации всех компилируемых функций. Разбиение множества компилируемых функций на подмножества резидентных и виртуальных функций оставлено на усмотрение пользователя. Для организации связывания виртуальных компилированных функций используется свойство (F)SUBR с косвенной /непрямой/ адресацией тела подпрограммы. Значением этого свойства в случае виртуальной функции является специальное информационное слово, хранящееся в пространстве полных слов. Вызов виртуальных скомпилированных функций в оперативную память производится по обращению автоматически и полностью скрыт от пользователя.

Будем далее для краткости называть функции ЛИСПа типа (F)EXPR /то есть нескомпилированные/ функциями, резидентные функции ЛИСПа типа(F)SUBR /то есть скомпилированные/ - блоками кодов, а виртуальные функции ЛИСПа типа(F) SUBR - сегментами.

В существующих ЛИСП-системах с компиляторами блоки кодов загружаются в фиксированную область оперативной памяти / BPS - пространство двоичных программ/ и являются абсолютными подпрограммами. Связывание с объектами системы, расположенными вне BPS, производится динамически. "Внутренние" адреса блоков кодов не меняются в процессе использования системы, как и расположение каждого из блоков кодов в BPS. Поэтому связывание /обращение/ блоков кодов между собой можно выполнять статически с помощью прямой передачи управления. Связывание функций с блоками кодов выполняется динамически с помощью свойства (F)SUBR атома - имени соответствующего блока кодов.

Однако эта схема непригодна в случае сегментов. Сегмент должен быть перемещаемым, так как хотя он и располагается /после вызова в оперативную память/ в BPS, однако его место в этой области не фиксировано. Связывание любого объекта /функции, блока кодов или другого сегмента/ с сегментом может быть только косвенным, через значение свойства (F)SUBR. При этом необходимо иметь возможность определять, находится ли данный сегмент в BPS или нет. В последнем случае, прежде чем выполнить связывание, необходимо произвести вызов сегмента в оперативную память.

Предлагаемая нами схема работы с сегментами заключается в следующем. Скомпилированные функции после первого прохода LAP хранятся на внешнем файле. Перед вводом этого файла пользователь определяет список сегментов, то есть тех скомпилированных функций, которые будут виртуальными. В BPS выделяется некоторая область для сегментов. Файл читается как обычно, скомпилированные определения функций обрабатываются во время второго прохода ЛИСП-ассемблера и, если они не определены как сегменты, загружаются в BPS. Определения скомпилированных функций, имена которых включены в список сегментов, транслируются в формат сегмента и переписываются в предварительно определенный файл на диске. В качестве значения свойства (F)SUBR имени функции формируется специальная информационная ячейка в пространстве полных слов. В эту ячейку помещается команда передачи управления на специальную подпрограмму интерпретатора, выполняющую вызов сегмента в оперативную память, и другая необходимая информация.

При первом выполнении некоторого блока кодов или сегментов, если в нем есть обращения к какому-либо другому блоку кодов, сегменту или функции, работает специальная подпрограмма интерпретатора /LINKIT/. В соответствии с типом вызываемого объекта эта подпрограмма производит связывание. Если при этом вызываемый объект является блоком кодов, то обращение к нему через LINKIT заменяется, как обычно, на команду перехода к этому блоку кодов. Если же вызываемый объект является сегментом, то обращение через LINKIT заменяется на команду перехода к его информационному слову.

При выполнении такой передачи управления сразу же следом за ней выполняется команда передачи управления, хранящаяся в информационном слове. Если сегмент уже загружен в оперативную память, то это будет команда передачи управления на его вход. Если он в оперативной памяти отсутствует, то это будет команда перехода на подпрограмму интерпретатора, выполняющую вызов сегментов с внешнего файла /например, CDISKIN /. Она считывает требуемый сегмент с диска, определяет наличие необ-

ходимого места в пространстве двоичных программ, при необходимости "удаляя" некоторые сегменты из оперативной памяти, настраивает требуемый сегмент по месту, загружает его и в соответствующем информационном слове заменяет свой адрес на адрес входа в загруженный сегмент.Затем осуществляется вход в сегмент.При удалении одного из сегментов из оперативной памяти в его информационном слове адрес входа сегмента заменяется на адрес подпрограммы CDISKIN, длина прибавляется к счетчику свободных ячеек BPS, а начальный адрес заносится в таблицу свободных блоков BPS. После того как необходимое число сегментов будет удалено,освободившаяся память собирается в один непрерывный блок, в который и помещается вновь вызываемый с внешнего файла сегмент.

Формат сегмента включает в себя слово-заголовок, таблицу перемещения и массив слов, содержащих собственно тело сегмента. Слово-заголовок сегмента необходимо потому, что часть информационного слова сегмента занята командой перехода /на сегмент или на подпрограмму интерпретатора CDISKIN /. В оставшейся части информационного слова и в слове-заголовке хранится информация о сегменте, необходимая для работы алгоритма виртуализации: необходимый объем памяти, тип, дисковый адрес, счетчик использования. Таблица перемещения содержит информацию об относительных адресах в теле сегмента, требующих настройки при загрузке его в оперативную память. После загрузки в оперативной памяти располагаются только слово-заголовок и настроенное по месту тело сегмента.

4. ЗАКЛЮЧЕНИЕ

Рассматриваемый алгоритм работы с виртуальными компилированными функциями не требует большого дополнительного расхода оперативной памяти: на каждый сегмент расходуется две дополнительные ячейки: одна постоянная в пространстве полных слов и одна временная в пространстве двоичных программ. Незначительно увеличивается и время связывания для сегментов, находящихся к моменту обращения к ним в оперативной памяти, - при каждом таком обращении выполняется лишь одна дополнительная команда перехода на тело сегмента. Основные потери времени приходятся на обмен с дисками и на загрузку сегментов в оперативную память. Однако эти неизбежные дополнительные затраты времени дают возможность экономить оперативную память. Это, во-первых, позволяет работать с полной системой типа REDUCE-2 при ограниченных ресурсах памяти, а во-вторых, увеличивать память под списки, то есть под собственно задачу пользователя. Благодаря этому может уменьшиться число необходимых сборок мусора, что частично компенсирует затраты времени на виртуализацию.

Описанный выше алгоритм рассматривался в применении к машинам серии CDC-6000. Однако он с небольшими видоизменениями, обусловленными конкретными особенностями машин и операционных систем, применим практически к любой системе программирования, основанной на ЛИСПе и имеющей в своем составе компилятор и LAP. Необходимые изменения относятся лишь к программе связывания компилированных функций и второму проходу LAP и совершенно не затрагивают ни компилятор, ни первый проход LAP, которые обычно реализованы на ЛИСПе и не являются внутренними частями интерпретатора. К интерпретатору же добавляется лишь ряд необходимых подпрограмм, не нарушающих его общей структуры.

По нашему мнению, виртуализация скомпилированных функций ЛИСПа на машинах серии ЕС ЭВМ позволит расширить круг пользователей системы REDUCE-2 и решаемых ими задач. На машине БЭСМ-6 без такой виртуализации какая-либо работа с системой REDUCE-2 вообще невозможна. Поэтому наряду с реализацией этих средств в интерпретаторе UT LISP мы ведем аналогичную работу и на этих машинах.

ЛИТЕРАТУРА

1. Kadantsev S.G., Rostovtsev V.A. SIGSAM Bulletin, 1981, 14, p. 14.
2. Ростовцев В.А. В кн.: Совещание по программированию и математическим методам решения физических задач. ОИЯИ, Д10,11-11264, Дубна, 1978, с. 175.
3. Hearn A.C. REDUCE Newsletter, No.6, University of Utah, Salt Lake Sity, 1979, p. 1.
4. LISP Reference Manual CDC-6000. The University of Texas at Austin, Computer Center, CCUM-2, 1975.
5. Griss M.L., Hearn A.C. Software Practice & Experience, 1981, 11, p. 541.
6. Bobrow D.G., Murphy D.L. CACM, 1967, 10, p. 155.
7. Maly K. Intern. J.Computer Math., 1974, 4, p. 69.
8. Шура-Бура М.Р., Мартынюк В.В. ЖВМиМФ, 1964, т.4, с. 963.
9. Ростовцев В.А. ОИЯИ, Р11-81-751, Дубна, 1981.

Рукопись поступила в издательский отдел
27 ноября 1981 года.