УДК 004.43 БК 22.18 **DOI: 10.14529/cmse230X0Z**

РАБОТА С ДАННЫМИ В УЧЕБНОМ ЯЗЫКЕ ПРОГРАММИРОВАНИЯ СИНХРО*

© 2023 Л.В. Городняя[1,2]

[1]*Институт систем информатики СО РАН*
(630090, Новосибирск, пр. им. М.А. Лаврентьва, д. 6),
[2]*Новосибирский государственный университет*
(630090, Новосибирск, ул. Пирогова, д. 2)
E-mail: gorod@iis.nsk.su,
Поступила в редакцию: ДД.ММ.ГГГГ

Статья является продолжением собственных предыдущих исследований автора в рамках многолетней работы по созданию учебного языка программирования СИНХРО, предназначенного для ознакомления с параллелизмом. Основное направление работ — уточнение понятий, способствующих подготовке небольших многопоточных программ при обучении параллельному программированию. Главный результат последнего года заключается в развитии механизма взаимодействия локальной и общей памяти. Дан приоритет парадигме функционального программирования, популярной при подготовке прототипов многопоточных программ. Это помогло преодолеть зависимость порядка вычислений от последовательности вхождения выражений в текст программы и размещения данных в памяти. Описаны отличия от привычных понятий программирования, сдерживающих решение задач организации параллельных вычислений и предельно распределенных систем из ряда потоков, взаимодействующих в терминах доступа к значениям переменных, возможно расположенных в общей памяти. Повышен базовый уровень воздействий на память. Часть из них укрупнены для предотвращения неожиданностей из-за асинхронности и ослабления императивности элементов распределенных систем. Добавлено понятие команд-двойников для управления императивной синхронизацией взаимодействующих устройств, полезное при решении вопросов освобождения памяти.

Ключевые слова: дисциплина доступа к памяти, функциональное программирование, многопоточные программы, неизменяемость данных, восстановление данных, освобождение памяти.

ОБРАЗЕЦ ЦИТИРОВАНИЯ

Городняя Л.В. Работа с данными в учебном языке программирования СИНХРО // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. 2023. Т. X, № Y. С. Z1–Z2. DOI: 10.14529/cmse230X0Z.

Введение

В лаборатории информационных систем ИСИ СО РАН разрабатывается учебный язык программирования СИНХРО, предназначенный для ознакомления с явлениями параллелизма, мета-программированием и особенностями много-поточного программирования [1, 2]. Прагматика языка соответствует парадигме функционального программирования (ФП), дополненной некоторыми, сравнительно безопасными, методами императивного программирования. При создании языка учтен опыт применения языков Lisp, Робик, SETL, БАРС, Sisal, Haskell, mpC, Oz. Приняты во внимания основные модели организации параллельных вычислений [3, 4]. и уточнен ряд понятий и методов, изменяющихся при переходе от обычного программирования к представлению много-поточных и многопроцессорных программ. Прежде всего это преодоление зависимости порядка вычислений от последовательности вхождения выражений в текст программы, а также повышение базового уровня

*Статья рекомендована к публикации программным комитетом Международной конференции «Суперкомпьютерные дни в России – 2022»

воздействий на память и взаимодействия с устройствами. Часть из них укрупнены для предотвращения неожиданностей из-за асинхронности и ослабления императивности элементов распределенных систем. Добавлено понятие команд-двойников для управления императивной синхронизацией взаимодействующих устройств.

Язык СИНХРО ориентирован на ознакомление с причинами неожиданностей при подготовке многопоточных программ, а также на формирование навыков профилактики неудачных взаимодействий процессов, учета равноправия независимых потоков, использования укрупненных воздействий на общую память, прогнозирования результатов автоматизированных преобразований программ и данных, включая оптимизацию и компиляцию программ. Учтены особенности отладки программ решений небольших учебных задач. В данной статье описаны решения, принятые на уровне ядра системы программирования. Рассмотрены подходы к решению проблем работы многопоточных программ над общей памятью. Язык СИНХРО позволяет управлять компиляцией многопоточных программ в многопроцессорные. Обучение метапрограммированию предполагается выполнять в форме подготовки сценариев отладки и преобразования программ, а также измерения вклада программируемых решений в производительность программ, что выходит за пределы данной статьи.

Изложение начинается с описания адаптации принципов ФП к подготовке многопоточных программ (раздел 1) с акцентом на равноправие параметров. Затем уточнены некоторые понятия программирования, требующие пересмотра в случае многопоточных программ (раздел 2), включая переход от потоков к процессам, достаточный для этого перехода уровень абстрактного процессорного комплекса, команды взаимодействия с устройствами и организации многопроцессорных комплексов. Далее рассмотрены механизмы повышения производительности многопроцессорных программ (раздел 3), включая восстановление данных и решения по организации работы с общей памятью. В заключении отмечены образовательные проблемы параллельных вычислений, перечислены представленные в статье результаты и намечены направления дальнейших исследований.

1. Функциональное программирование и параллелизми

Функциональное программирование — одна из первых парадигм, направленных не столько на получение эффективной реализации заранее созданных и хорошо изученных алгоритмов, сколько на продуктивность решения новых и исследовательских задач, изучение которых продолжается в процессе программирования [5–11]. Для таких задач правильность и полнота решений важнее эффективности и производительности полученных программ. Семантические принципы ФП, такие как всеобщность (универсальность), самоприменение (рекурсия) и равноправие параметров функций, позволяют функции и значения представлять такими же символами, что и любые данные для компьютерной обработки. Представления функций и значений могут использовать рекурсивные символьные формы. Порядок вычисления параметров не важен, они не зависят друг от друга и вычисляются в одной области видимости. Прагматические принципы реализации ФП, такие как гибкость ограничений на размеры блоков памяти, неизменяемость обрабатываемых данных, однозначность и строгость (чистота) результата функции, позволяют системам ФП предотвращать простои памяти автоматизацией оперативного анализа достижимости данных, освобождения и повторного использования памяти, хранящей недостижимые данные. Представление каждого данного помещается в новую часть свободной памяти без искажения аргументов

функции, они могут быть полезны для других функций. Любое количество результатов функции может быть представлено в виде единой символьной формы, причем значения одинаковых форм в одной области видимости совпадают.

Представление алгоритмов в виде чисто функциональных программ дает важные следствия. Из семантических принципов вытекает возможность метапрограммирования, верификации и факторизации программ на автономно развиваемые модули. Прагматические принципы допускают интуитивные модели непрерывности процессов, обратимости действий и линейных потоков из унарных функций, служащих основой для разработки прототипов на ранних стадиях жизненного цикла многопоточных программ. Таким образом, комплекс семантических и прагматических принципов чистого ФП обеспечивает поддержку подготовки программ при организации параллельных вычислений благодаря сведению к наборам независимых потоков. Кроме того, любой фрагмент, выполнение которого маловероятно или несвоевременно, может быть перемещен из представления функции в отложенное или опережающие действие. Такие «ленивые» вычисления дают возможность оперативно перераспределять нагрузку процессоров. Любое конечное множество потоков может работать как пространство итераций для определенной на нем функции при автоматическом распараллеливании [12].

При переходе к многократно применяемым программам и суперкомпьютерным вычислениям успех приложения и производительность программ становятся важнее, чем их полная формальная корректность и предельная эффективность. В дополнение к принципам и следствиям в реальные языки и системы программирования для производственного ФП обычно включают механизмы практичного компромисса, которые, внешне не нарушая функциональный стиль представления программ, встраиваются как специальные функции. Например, Lisp 1.5, Clisp, Cmucl, Clojure и другие члены семейства языка Lisp обычно предоставляют возможности такого рода:

- контроль типов данных, несколько ограничивающий принцип универсальности механизмом статического и динамического анализа типов данных, допускает частичные функции;
- схемы циклов, позволяющие преодолеть типичные опасения по поводу сложности реализации принципа самоприменения, уменьшают рекурсивность определений и наследуют привычные шаблоны управления вычислениями;
- возможность восстановления данных, исключающая чрезмерное потребление и обременительное освобождение памяти, аккуратно отклоняется от принципа неизменности данных в пользу программируемого повторного использования памяти;
- программируемые прогнозы объема памяти и скорости выполнения программ, поддерживают управление выбором моментов включения механизмов автоматизации, смягчая несколько затратный принцип гибкости ограничений («сборщик мусора» - garbage collector);
- псевдофункции, выполняя нагрузку по взаимодействию с устройствами, ввод-вывод данных и доступ к файлам, внешне сохраняют общую функциональную схему управления вычислениями, что несколько уводит от принципов равноправия параметров и неизменяемости данных;
- мемоизация, снижая сложность многократно повторяемых вычислений, сохраняет доступ к успешному опыту ранее выполненных вычислений, слегка модифицируя принцип чистого результата.

Высокопроизводительное программирование требует учета перспектив многократного использования и улучшения программ [13]. Эксперименты на суперкомпьютерах показали, что системные решения могут вносить значительный вклад в производительность параллельных вычислений, важно, что такой вклад может превышать теоретические оценки. Квалификация разработчика программных систем обычно включает в себя умение изобретать новые решения задач и навыки ответственного улучшения готовых решений.

Представленные на посвященной современным тенденциям ФП конференции (Trends in Functional programming) доклады убедительно показали нацеленность ФП на решение задач организации параллельных вычислений [14]. Строго говоря, ФП способно выполнять роль проектно-конструкторского отдела для производственного программирования. В некоторых источниках по измерению трудоемкости программирования и производительности программ утверждается, что опыт ФП повышает качество программирования в любой парадигме [15].

2. Уточнение семантики

При переходе к параллельному программированию многие привычные понятия претерпевают изменения, начиная с понятия «данное». Данными являются уже не только представления значений и функций, но еще и представления процессоров и разнообразных устройств. Такое обобщение позволяет сферу успешного программирования обработки данных распространить на параллельные вычисления и распределенные системы.

2.1. Равноправие элементов структур данных

Многопоточная программа допускает асинхронное выполнение потоков и действий, что означает в соответствии с принципом равноправия параметров функций независимость порядка вычисления выражений от порядка их вхождения в структуры данных, рассматриваемые как функции над этими выражениями. Для приобретения навыков учёта такой независимости в языке СИНХРО предлагается в записи структур данных разнести смысл скобок и разделителей. Круглые скобки означают, что доступ к данным возможен последовательно, а квадратные символизируют произвольный доступ. Если разделителем в перечне данных является точка с запятой «;», то элементы перечня вычисляются последовательно, в порядке вхождения в запись выражения, а в случае запятой «,» — в произвольном порядке, независимо от порядка записи (табл.1).

Часть проблем взаимодействия потоков алгоритмически не разрешима, поэтому в задачи ознакомления с параллелизмом входит научиться предвидеть такие опасности и отлаживать программы при обнаружении неудачных взаимодействий потоков. Наполнение многопоточной программы может развиваться независимо от схем управления вычислениями в отдельных потоках, а схемы можно реорганизовывать без дополнительной отладки наполнения, используя факторизацию на автономно развиваемые модули. Схемы работают подобно макросам, но их применение нагружено дополнительным контролем соответствия параметров объявленным видам фрагментов. Выражение может использовать размещенные в общей памяти данные, но не изменяет их. Директива обрабатывает память и при благополучном исходе дает результат подобно выражению.

Таблица 1. Структуры данных

№ п/п	Синтаксис	Семантика
1	’(’Выр(’;’Выр)...’)’	Список, заполняемый последовательно вычисляемыми элементами, в порядке записи в программе
2	’(’Выр(’,’Выр)...’)’	Список, заполняемый в порядке записи в программе асинхронно вычисляемыми элементами, возможно в другом порядке
3	’[’Выр (’;’Выр)...’]’	Вектор, заполняемый последовательно вычисляемыми элементами, в порядке записи в программе
4	’[’Выр (’,’Выр)...’]’	Вектор, заполняемый в порядке записи в программе асинхронно вычисляемыми элементами, возможно в другом порядке

2.2. От потоков к процессам

Компилятор многопоточной программы строит функционально эквивалентную ей многопроцессорную программу для размещения на многопроцессорном комплексе. Набор потоков при компиляции преобразуется в набор процессов, каждый из которых выполняется отдельным процессором, но может быть иначе при решении проблем балансировки нагрузки в соответствии с принципом гибкости ограничений, распространенному с памяти на процессоры.

Контекст исполнения программы при переходе к процессам — общая память, данные из неё доступны всем процессам многопроцессорной программы. Общая память содержит представления глобальных, возможно изменяемых, переменных. Существует и локальная память, соответствующая процессам, созданным по определениям потоков, функций и циклов, подчиненная иерархии определений в программе — области видимости. Обработка общей и локальной памяти на уровне много-поточной программы выглядит одинаково, что позволяет при отладке перемещать фрагменты в разные позиции программы. Возможно хранение в общей памяти определений и имён. Типы именуемых данных можно устанавливать по виду их значения или функции. Имеются распознающие функции-предикаты ATOM, NUMBER, LIST, ARRAY и др.

Процесс выполняется как непустой ряд команд процессора, допускающий наращивание при исполнении. Очередная команда может начинать выполнение строго после начала предыдущей команды, но не обязана дожидаться её завершения. Каждый процессор работает по шагам, соответствующим выполнению одной команды. После шага происходит переход к общему механизму управления многопроцессорной конфигурацией.

Компилятор по каждому потоку строит корректный ряд команд процесса, согласованных для выполнения действий потока. Завершение набора процессов может включать выполнение сверток для получения единого результата, что несколько влияет на их синхронизацию. Между двумя соседними командами процесса может быть выполнена команда другого процесса, а это требует особого внимания при организации воздействий на общую память.

Решение учебной задачи строится в два шага. Сначала программа решения выглядит как чисто функциональная схема, наполняемая выражениями — фрагментами вычислений без действий над общей памятью. Потом возможен гладкий переход к более эффектив-

ной версии программы методом замены некоторых выражений на директивы, т.е. чистых функций на функционально эквивалентные им псевдофункции или процедурные аналоги с воздействиями на память и синхронизацией потоков по мере необходимости — принцип независимости параметров и факторизация. Для простоты изложения здесь не рассматривается разнообразие категорий систем команд процессоров и видов используемой памяти с различной дисциплиной доступа.

2.3. Абстрактный процессорный комплекс

Абстрактный процессорный комплекс (АПК) способен выполнять вычисления несколько более широкого класса, чем обычно задано семантикой языка программирования. Кроме собственно процессоров к выполнению программы привлекаются дополнительные устройства, которые рассматриваются как особые процессоры, система команд которых не определена на уровне языка. Они способны выполнять некоторые действия по запросам от других процессоров. Система команд локального процессора поддерживает обработку данных, их размещение и реорганизацию в своей локальной или общей памяти, и управление ходом выполнения процессов, включая взаимодействие процессоров и устройств, и резервирование данных для их защиты от случайных изменений, подобно средствам операционных систем [16]. Программа использует общую память, данные из которой доступны отдельным процессорам, выполняющим программу. Шаги разных процессоров асинхронны, но могут происходить одновременно.

При спецификации команд АПК для языка СИНХРО использовалась предложенная П.Лендиным машина SECD [7], дополненная командами наиболее известных виртуальных машин, включая JVM [17–20]. Машина SECD работает над четырьмя регистрами: S — стек для промежуточных результатов, E — контекст для размещения именованных значений, C — управляющая вычислениями программа, D — резервная память (Stack, Environment, Control-list, Dump). Регистры приспособлены для хранения выражений в форме символов или списков. К регистрам системы команд машины SECD добавлен регистр «M» (Memory), что дает уточненное обозначение многопроцессорного комплекса: M(SECD)+ — абстрактный многопроцессорный комплекс над общей памятью. Это обозначение символизирует, что M — общая память для всех процессоров, (SECD)+ — что хотя бы одни процессор обязателен, общее число процессоров произвольно и не исключено изменение их числа в динамике. M — общая память, условно подчинена принципу неизменяемости данных, расширенному механизмом восстановления данных. Она состоит из регистров произвольного доступа, хранящих счетчик числа потоков, использующих эти регистры, имя переменной, ее текущее значение и протокол произошедших изменений, устроенный как вектор или список. Поддержана возможность восстановления значений, оттесненных присваиваниями. Состояние машины АПК полностью определяется содержимым этих пяти регистров.

m s e c d → m' s' e' c' d' – переход от старого состояния машины к новому.

Разница между SECD и M(SECD)+ сводится к операциям над данными в общей памяти. Регистр M является списком списков или стеков, каждый элемент которого начинается с имени глобальной переменной, вслед за которым расположено её текущее значение, а дальше следует протокол изменения значений, выглядящий как последовательность номеров процессов с каждым последующим установленным этим процессом значением. В определённом смысле состояние общей памяти можно рассматривать как непременное дополнение к формальному результату работы программы, технически похожее на мемоизацию.

Фактически исполняются команды по очереди, последовательно, начиная с первых в регистре управляющей программы, хотя можно считать, что они исполняются в произвольном порядке по мере готовности. . Такое определение может быть машинно-независимым и переносимым. Размер стека не ограничен. Каждая команда абстрактного многопроцессорного комплекса «знает» число используемых при ее работе элементов стека S и их форматы, элементы она удаляет из стека S и вместо них размещает один выработанный результат для обычных команд или набор результатов для многопроцессорных команд, размещая число процессов в верхнем элементе стека. Каждая команда АПК «знает» число используемых при ее работе элементов стека S и их форматы, элементы она удаляет из стека S и вместо них размещает один выработанный результат для обычных команд или набор результатов для многопроцессорных команд, размещая число процессов в верхнем элементе стека. Всегда известно число текущих результатов в стеке S, которые можно явно свернуть в единственный строгий результат специальной сверткой, необходимость в которой выясняет компилятор и размещает свертку в регистре C абстрактной машины. Свёртка может быть встроенной в язык или программируемой, позволяющей указанное число элементов в стеке свести в одно данное. Это может быть список, структура, сумма, произведение, максимум, минимум, последний и т.п. Свертки обычно коммутативны, за редким исключением — первый или последний по времени вычислений или позиции вхождения в программу. Фильтр можно рассматривать как частный случай свертки.

Суммарно комплект команд АПК включает в себя действия, часть из которых определены в книге П. Хендерсона (LD, LDC, LDF, AP, RTN, RAP, DUM, SEL, JOIN, CONS, CAR, CDR, CONS, ATOM, EQ, SUB, ADD, MUL, DIV, STOP) [7]. Этот набор команд в языке СИНХРО пополнен для решения проблем работы с общей памятью и внешними устройствами, особенности которых проявляются на командах воздействия на общую память, взаимодействия общей и локальной памяти (табл. 2), доступа к устройствам и организации параллельных процессов (табл.3) [17–20]:

Таблица 2. Команды работы с памятью

№ п/п	**Синтаксис**	**Семантика**
1	LDM	копирование данного из общей памяти в стек;
2	SET	запись в общую память комплекса из стека;
3	LET	сохранение локальных значений в общей памяти;
4	DEL	удаление верхнего элемента стека;
5	MLL	пересылка в головной элемент списка из другого списка;
6	MLV	пересылка и списка в указанный элемент вектора;
7	MVL	пересылка из вектора в головной элемент списка;
8	MVV	пересылка в указанный элемент вектора из другого вектора;
9	CHNG	обмен данными в общей памяти комплекса.

Усложненные, точнее укрупненные, команды пересылки и обмена данными в общей памяти (MLL, MLV, MVL, MVV, CHNG) нужны, чтобы исключить возникновение временных интервалов между взаимосвязанными присваиваниями в общей памяти. Иначе может возникать так называемая «фантомная» память или доступ к формально уже удаленным данным, что иногда обнаруживается при использовании JVM в сетях, стандарт на которую был утвержден более 20 лет назад. Асинхронное выполнение воздействий на память при ее

освобождении не исключает, что некоторое время ссылка из программы на память удалена, а адрес блока памяти не помечен как свободный, или, наоборот, блок памяти помечен как свободный, а ссылка на него из программы не удалена. Технические детали определения этих и остальных команд приведены в статье [21].

2.4. Внешние устройства и процессоры

При работе с устройствами через конечное время вырабатывается сигнал, говорящий или об успешном выполнении действия, или о причине отказа в его завершении. Выработка сигнала об отказе внешнего устройства приводит к запоминанию в протоколе информации об ущербно выработанных результатах, что можно учесть при отладке, и/или к выполнению остаточной программы в стиле смешанных вычислений (табл.3).

Таблица 3. Команды работы с памятью

№ п/п	**Синтаксис**	**Семантика**
1	WRT	вывод данных из стека на внешнее устройство и сигнал успеха-провала;
2	RD	ввод данных от внешнего устройства в стек и сигнал успеха-провала;
3	ANY	выравнивание стека, если данное с устройства или на устройство не введено из-за провала.
	Для	организации параллельных вычислений требуются еще команды:
4	KIT	комплекс = из процессоров для выполнения действий;
5	ROW	поток или ряд действий на одном процессоре;
6	REZ	размещение заданного числа результатов процесса в свой стек;
7	WAIT	приостановка с ожиданием сигнала от указанного процесса (завершения или сообщения);
8	SEND	сообщение указанному процессу;
9	NEXT	ожидание события или завершения действия указанного процесса.

Так обеспечивается подключение комплексов процессоров для неупорядоченных наборов потоков, мощность комплекса известна. На каждом процессоре происходит размещение процессов в виде ряда действий, длина ряда известна в каждый момент. Предполагается выполнение последовательности действий ряда на одном процессоре, завершаемых передачей результатов процесса в стек с локализацией воздействий на транзакционную общую память с явной обработкой ошибок. Реализация передачи сообщений подобна рандеву в языке ADA — обмен происходит между двумя активными процессами и каждый знает что и кому он передает или от кого сообщение получает. Процесс-отправитель перед выполнением SEND может дожидаться готовности процесса-получателя, то есть проверять, что получатель сейчас выполняет команду WAIT, иначе ожидать ее, чтобы произошло рандеву. Процесс может не знать, что его завершения кто-то ждет.

Многопроцессорная программа считается идеальной, компилятор ее строит без учета возможных аварийных ситуаций, считает, что все данные на устройстве ввода расположены в соответствии с запросами программы, именно те, что нужны. Состояния стеков к моменту выполнения команд сформированы вполне корректно. Регистры локальной памяти подчинены принципу неизменяемости данных. Новые данные размещаются в первом элементе

списка, а к прежним значениям, хранимым в общей памяти, можно вернуться при необходимости, например, при отладке посмотреть историю изменения данных. При выполнении обратимых действий поддержана и обратимость воздействий на общую память. Идеальная конфигурация многопроцессорной программы работает на как бы бесконечной общей памяти, не решает проблемы ее исчерпания и необходимости освобождения от ставших ненужными данных. Поскольку процессоры рассматриваются как одна из категорий данных, то можно от понимания памяти как одного из регистров машины перейти к отдельному процессору общей памяти, обладающему своей системой команд.

3. Уточнение прагматики

Уточнение семантики влечёт некоторое усложнение реализационных решений в системе программирования для подготовки многопоточных программ, допускающих взаимодействия процессов и использование общей памяти.

3.1. Восстановление данных

В проекте системы программирования для языка СИНХРО уточнен принцип неизменяемости данных, характерный для ФП. Работа с памятью поддержана в транзакционном стиле, подобно обработке записей в базах данных, т.е. каждое воздействие на память или выполняется полностью, или восстанавливается состояние до начала выполнения не завершившейся команды [22]. Каждый элемент общей памяти в любой момент времени обрабатывается только в одном процессе программы, подобно захвату-освобождению файла. Хранение данных в общей памяти сопровождается протоколом изменений, чтобы при отладке можно было видеть какой процесс внес изменения и, при необходимости, вернуть утраченное значение. Поэтому не так уж опасен и возможен побочный эффект присваивания, если допускается восстановление прежних значений переменных. Такая реализация позволяет поддержать транзакции и обратимость обработки памяти при отладке учебных программ. В стеке размещается список результатов однократных присваиваний (SSA-формы1). К верхнему элементу контекста прицепляется этот список в хвост, вслед за аргументами. Предполагается, что имена формальных параметров и локальных данных различны.

3.2. Шкала доступа к общей памяти

Освобождение общей памяти требует механизмов, подобных опробованным в практике операционных систем при решении вопросов управления распределенными системами с совмещением работы независимых программ [16]. Можно рассмотреть вариант многопроцессорного комплекса, допускающее управление доступом к общей памяти с использованием паспортов, хранящих шкалу необходимых регистров общей памяти для доступа к глобальным переменным. Паспорт одновременно доступен и локальному процессору, и процессору общей памяти. В этом случае локальный процесс при «сборке мусора» формирует новое состояние такой шкалы, доступной процессору общей памяти. Локальные процессы вместо произвольного доступа к общей памяти в таком случае обладают лишь шкалой для доступа к определенным регистрам. При освобождении общей памяти появляется возможность частичного освобождения памяти, не задействованной в объединении шкал регистров от локальных процессов, без приостановки всех процессоров. В частности, при отказе отдельного процессора можно по шкале глобальных переменных выполнить уменьшение счетчиков доступа к этим переменным для решений по освобождению или восстановлению памяти.

3.3. Команды-двойники доступа к общей памяти

Несколько проще выглядит модель комплекса со специальным процессором общей памяти, поддерживающим императивную синхронизацию. Такой процессорный комплекс состоит из ряда обычных процессоров и одного пассивного процессора общей памяти, с которым могут взаимодействовать активные локальные процессоры. Между собой они взаимодействуют только через общую память. Каждый процессор выполняет одну последовательность команд, среди которых встречаются команды запросов к процессору общей памяти. Это активные команды, выполняемые без особенностей, за исключением того, что команды запросов к общей памяти имеют двойников среди команд процессора общей памяти. Это пассивные команды, работающие как ленивые вычисления, возбуждаемые при выполнении запросов от локальных процессоров. Пассивные команды процессора общей памяти собраны в ряд очередей, каждая из которых соответствует конкретному локальному процессору и содержит двойников в том же порядке, что и последовательность активных команд. Пара запрос и его двойник выполняются неразрывно в стиле рандеву языка Ada. Как и при пересылках, механизм рандеву исключает случайное вмешательство сторонних процессов. При этом происходит обмен данными между локальной памятью активного процессора и общей памятью пассивного процессора (см. табл.4).

уц-

Таблица 4. Дублеты — парные команды (активные + пассивные)

№ п/п	Синтаксис	Семантика	Синтаксис	Семантика
1	GIVE	запрос регистра для переменной в общей памяти	TAKE	выбор регистра для переменной, счетчик надо увеличить на 1
2	SAFE	запрос на хранение данного в переменной в общей памяти	WRITE	размещение данного, в протокол - указатель на данное и имя процесса
3	READ	запрос на чтение данного	VAR	доступ процесса к данному из общей памяти
4	UNDO	запрос на получение предыдущего значения переменной	UNDO	доступ к предыдущему значению переменной, состояние регистров переменной не меняется.
5	FREE	объявление, что переменная не нужна	FREE	освобождение памяти, счетчик кратности уменьшается на 1

Для локальных процессоров возможны запросы на регистр для переменной, на хранение данного в переменной по ее регистру, на чтение данного из переменной по регистру, на получение предыдущего данного из переменной по адресу и на объявление, что больше переменная не нужна.

Команды-двойники процессора общей памяти, пассивно ждущие запросов от локальных процессоров или отладчика, одновременно с выполнением активных команд выполняют выбор регистра для переменной с указанным номером (счетчик кратности доступа к нему

надо увеличить на 1), размещение данного (если данное — указатель на вектор или список, то они копируются полностью в общую память и в протокол вносится копия указателя на данное и номер активного процесса), чтение данного для передачи локальному процессору (состояние регистра не меняется), чтение предыдущего данного, освобождение памяти (счетчик кратности доступа уменьшается на 1).

Процессор общей памяти может функционировать как элемент распределенной системы, включающийся по мере поступления запросов от локальных процессоров, и не требующий приостановки всех процессоров на время решения проблемы освобождения памяти.

Представленный здесь метод использует ряд процессоров со своей локальной памятью и процессор общей памяти. Локальные процессоры могут работать автономно, по мере необходимости обращаясь к общей памяти и время от времени освобождая свою локальную память. В локальной памяти хранится шкала регистров доступа к общей памяти, которая может то расширяться, то сужаться по мере инструкций из программы или сужаться в результате «сборки мусора». Общая память состоит из двух частей — регистров прямого доступа и кучи для хранения протоколов, а также структур данных и старых значений после присваивания на случай необходимости восстановления состояний памяти. Время от времени при исчерпании той или иной области общей памяти или просто по графику запускается алгоритм, похожий на «Stop-copy» в системах ФП. Система команд абстрактного комплекса образует два уровня, на каждом из которых можно видеть подсистемы организации вычислений, работы с памятью, управления процессами и структурирования данных. Один уровень реализует работу отдельного потока над локальной памятью, другой поддерживает взаимодействие потоков над общей памятью и внешними устройствами. Напоминает «рассредоточенное представление сосредоточенных действий» А.Л. Фуксмана.

Заключение

На этапе ознакомления с проблемами параллелизма полезно научиться строить и отлаживать небольшие многопоточные программы взаимодействия процессов, выполняемых на модели произвольной многопроцессорной конфигурации над общей памятью. При обучении параллельному программирования опыт такой работы поможет видеть и понимать разные явления, происходящие при взаимодействии потоков, предвидеть проблемы и накапливать рецепты решения проблем, возникающих при подготовке и отладке многопоточных программ, что особенно ярко может влиять на переход к суперкомпьютерам [23]. Главный результат ознакомления — формирование интуитивных понятий, соответствующих реальной сложности параллельного программирования.

При создании языка СИНХРО для этих целей принципы ФП адаптированы к учебным задачам подготовки многопоточных программ. Более конкретно, принцип неизменяемости данных, характерный для ФП, распространен на работу с переменными в общей памяти. Этот механизм дополнен средствами восстановления данных на основе протоколов изменения значений переменных. Предложено решение проблемы освобождения общей памяти как композиции методов «сборки мусора» и учета кратности доступа к переменным из потоков, взаимодействующих через использование данных, хранящихся в общей памяти.

В данной статье описана адаптация принципов ФП к подготовке многопоточных программ (раздел 1), позволившая уточнить семантику структурных выражений, требующую пересмотра в случае многопоточных программ из-за различий в пространствах допустимых процессов (раздел 2) с учетом принципа равноправия параметров. Рассмотрены осо-

бенности перехода от потоков к процессам и представлен достаточный для этого перехода уровень абстрактного процессорного комплекса, в систему команд которого включены команды обмена данными и работы с общей памятью, описаны команды взаимодействия с периферийными устройствами и организации многопроцессорных комплексов. Определены дополнительные механизмы для повышения производительности многопоточных программ (раздел 3), включая восстановление данных и решения по организации работы с общей памятью, использующие шкалы доступа к памяти и команды-двойники.

Направления дальнейших исследований связаны с расширением реализации языка СИНХРО в форме рада диалектов, поддерживающих последовательность целей обучения, таких как ознакомление с феноменами параллелизма, приобретение опыта метапрограммирования и навыков подготовки многопоточных программ, изучение безопасных методов представления взаимодействующих процессов и эксперименты по многопроцессорному программированию. Диалекты можно использовать автономно и совместно.

Автор благодарен Дмитрию Владимировичу Мажуге, выполнившему для языка СИНХРО реализацию виртуального многопроцессорного комплекса на языке Clojure.

Литература

1. Городняя Л.В. Язык параллельного программирования СИНХРО, предназначенный для обучения. Новосибирск, Препринт ИСИ СО РАН № 180, 2016. 30 с.
2. Городняя Л.В. О курсе «Начала параллелизма» // Ершовская конференция по информатике. Секция «Информатика образования». (Новосибирск, 27 июля 2011 г.). С. 51-54.
3. Воеводин В. В. Параллельные вычисления. СПб.: БХВ-Петербург, 2002. 608 с.
4. Хоар Ч. Взаимодействующие последовательные процессы. Издательство «Мир», 1989. 264 с.
5. McCarthy J. LISP 1.5 Programming Manual. The MIT Press, Cambridge, 1963. 106 p. https://doi.org/10.7551/mitpress/5619.001.0001
6. Backus J. Can programming be liberated from the von Neumann style? A functional stile and its algebra of programs. – Commun. ACM 21, 8, 1978. P. 613-641. https://doi.org/10.1145/359576.359579
7. Хендерсон П. Функциональное программирование. М.: Мир, 1983. 349 с.
8. Лавров С.С. Функциональное программирование // Компьютерные инструменты в образовании. 2002, N 2-4. С. 42-52.
9. Лавров С.С., Городняя Л.В. Функциональное программирование. Принципы реализации языка Лисп // Компьютерные инструменты в образовании. 2002, N5. С. 49-58
10. Городняя Л.В. Основы функционального программирования. М.: Интернет-Университет Информационных технологий. 2004. 272 с.
11. Городняя Л.В. Первые реализации языка Lisp в СССР // Материалы второй Международной конференции «Развитие вычислительной техники и ее программного обеспечения в России и странах бывшего СССР» (SoRuCom-2011) (Великий Новгород, 12–16 сентября 2011 г.). С. 95-100.
12. Cann D. C. SISAL 1.2: A Brief Introduction and tutorial. Preprint UCRL-MA-110620. Lawrence Livermore National Lab., Livermore – California, May, 1992. 128 p.

13. Бурдонов И.Б., Косачев А.С. Семантики взаимодействия с отказами, дивергенцией и разрушением. Часть 2. Условия конечного полного тестирования // Вестник Томского государственного университета, № 2(15), 2011.

14. Trends in Functional programming/ 22nd International Symposium, 22nd International Symposium, TFP 2021, Virtual Event, February 17–19, 2021, Revised Selected Papers Springer, LNCS 12834. 137 p. URL: https://link.springer.com/book/10.1007/978-3-030-83978-9 (дата обращения: 05.04.2023).

15. Erann Gat Lisp as an Alternative to Java. URL: https://flownet.com/gat/papers/lisp-java.pdf (дата обращения: 05.04.2023).

16. Иртегов Д.В. Введение в операционные системы. СПб.: БХВ-Петербург, 2008. 1040 с.: ил. – ISBN 978-5-94157-695-1.

17. Кнут Д.Э. Искусство программирования, том 1, выпуск 1. MMIX — RISC-компьютеры нового тысячелетия. М.: Вильямс, 2017. 160 с.

18. Вирт Н. Построение компиляторов. М.: ДМК Пресс, 2010. ISBN 978-5-94074-585-3, 0-201-40353-6

19. Айлиф Дж. Принципы построения базовой машины. М.: Мир, 1973. 119 с.

20. Эванс Б., Гоф Дж, Ньюленд К. Java: оптимизация программ. Практические методы повышения производительности приложений в JVM. М.: Диалектика, 2019. 448 с. – ISBN 978-5-907114-84-5.

21. Городняя Л. В. Абстрактная машина языка программирования учебного назначения СИНХРО // Вестник НГУ. Серия: Информационные технологии. 2021. Т. 19, No 4. С. 16-35.

22. Грабер М. Введение в SQL. М.: Лори, 1996. 337 с.

23. Левин В.К. Отечественные суперкомпьютеры семейства МВС. URL: `http://parallel.ru/mvs/levin.html` (дата обращения: 05.04.2023).

Городняя Лидия Васильевна, к.ф.-м.н., доцент, кафедра программирования, Новосибирский государственный университет (Новосибирск, Российская Федерация), gorod@iis.nsk.su

---

DOI: 10.14529/cmse230X0Z

WORKING WITH DATA IN THE EDUCATIONAL PROGRAMMING LANGUAGE SYNHRO

© 2023 L.V. Gorodnyaya[1,2]

[1] *A.P. Ershov Institute of Informatics Systems (6, Acad. Lavrentjev pr., 6, Acad. Lavrentjev pr., Novosibirsk 630090, Russia 630090, Russiaa),*

[2] *Novosibirsk State Universisty (630090, Novosibirsk-90, 2 Pirogova Str. Russia)*

E-mail: gorod@iis.nsk.su,

Received: ДД.ММ.ГГГГ

The article is devoted to clarifying the concepts that are useful in preparing small multi-threaded programs for teaching parallel programming. The approach was formed in the process of creating the SYNHRO language.

Priority is given to the paradigm of functional programming, which is popular in the preparation of prototypes of multi-threaded programs. A description is given of differences from the usual ideas that hinder the solution of problems of organizing parallel computing and extremely distributed systems from a number of threads interacting in terms of access to the values of variables, possibly located in a shared memory. A mechanism for the interaction of local and shared memory is proposed. The main models of organizing parallel computing are taken into account and a number of concepts and methods are refined, which change during the transition from conventional programming to the representation of multi-threaded and multiprocessor programs. First of all, this is overcoming the dependence of the order of calculations on the sequence of occurrence of expressions in the program text. Further the basic mechanism of influences on memory changes. Some of them are made more complex to prevent surprises due to asynchrony and weakening the imperativeness of executing elements of distributed systems. The concept of twin commands has been added to control the imperative synchronization of interacting devices, useful in resolving memory freeing issues.

Keywords: memory access discipline, functional programming, multi-threaded programs, data immutability, data recovery, memory freeing.

FOR CITATION

Gorodnyaya L.V. Working with data in the educational programming language SYNHRO. Bulletin of the South Ural State University. Series: Computational Mathematics and Software Engineering. 2023. Vol. X, no. Y. P. Z1–Z2. (in Russian) DOI: 10.14529/cmse230X0Z.

References

1. Gorodnyaya L.V. Parallel programming language SYNHRO for learning. Novosibirsk. Preprint of ISI SB RAS No. 180, 2016. 30 p.
2. Gorodnyaya L.V. About the course "The Beginnings of Parallelism"// Ershov Conference on Informatics. Section "Informatics of education". July 27, 2011. Novosibirsk. 2011. With. P. 51-54. (in Russian).
3. Voevodin VV Parallel Computing. St. Petersburg: BHV-Petersburg, 2002. 608 p. (in Russian).
4. Hoare C. A. R. Interacting Sequential Processes Mir Publishing House, 1989. 264 p. (in Russian).
5. McCarthy J. LISP 1.5 Programming Manual. The MIT Press, Cambridge, 1963. 106 p. https://doi.org/10.7551/mitpress/5619.001.0001
6. Backus J. Can programming be liberated from the von Neumann style? A functional stile and its algebra of programs. Commun. ACM 21, 8, 1978. P. 613-641. https://doi.org/10.1145/359576.359579
7. Henderson P. Functional programming. M.: Mir, 1983. 349 p. (in Russian).
8. Lavrov S.S. Functional programming. // Computer tools in education. 2002, No. 2-4. (in Russian).
9. Lavrov S.S., Gorodnyaya L.V. Functional programming. Lisp language implementation principles. // Computer tools in education. 2002, N5. P. 49-58 (in Russian).
10. Gorodnyaya L.V. Fundamentals of functional programming. M.: Internet University of Information Technologies. 2004. 272 p.

11. Gorodnyaya L.V. First implementations of the Lisp language in the USSR // Proceedings of the Second International Conference Development of Computing Technology and Its Software in Russia and the Former USSR Countries (SoRuCom-2011). P. 95-100. (in Russian).

12. Cann D. C. SISAL 1.2: A Brief Introduction and tutorial. Preprint UCRL-MA-110620. Lawrence Livermore National Lab., Livermore – California, May, 1992. 128 p.

13. Burdonov I.B., Kosachev A.S. Semantics of interaction with failures, divergence and destruction. Part 2. Conditions for final full testing // Bulletin of Tomsk State University, No. 2(15), 2011. 011.htm (in Russian).

14. Pieter Koopman, Steffen Michels, Rinus Plasmeijer. Dynamic Editors for Well-Typed Expressions // Trends in Functional programming/ 22nd International Symposium, TFP 2021, February 17–19, 2021. Springer, LNCS 12834. P. 44–66.

15. Erann Gat Lisp as an Alternative to Java. URL: `https://flownet.com/gat/papers/lisp-java.pdf` (accessed: 05.04.2022).

16. Irtegov D.V. Introduction to operating systems St. Petersburg: BHV-Petersburg, 2008. 1040 p. (in Russian).

17. Donald Knuth D.E. The Art of Programming, Volume 1, Issue 1. MMIX - RISC Computers of the New Millennium. M.: Williams, 2017. 160 p. (in Russian).

18. Wirth N. Compiler Construction. Moscow: DMK Press, 2010. ISBN 978-5-94074-585-3, 0-201-40353-6 (in Russian).

19. Ailiffe J.K. Principles of building a base machine. M.: Mir, 1973. 119 p. (in Russian).

20. Evans B., Gough J., Newland K. Java: program optimization. Practical methods for improving application performance in the JVM. M .: Dialectics, 2019. 448 p. ISBN 978-5-907114-84-5. (in Russian).

21. Gorodnyaya L.V. Abstract machine of the programming language for educational purposes SYNHRO. Bulletin of NGU. Series: Information technologies. 2021. V. 19, No 4. P. 16–35. (in Russian).

22. Graber M. Introduction to SQL. M.: Lori, 1996. 377 p. (in Russian).

23. Levin V.K. National Family of MVS Supercomputers. URL: `http://parallel.ru/mvs/levin.html` (accessed: 05.04.2022). (in Russian).