Язык фрагментированного программирования Freepal

Г.А. Щукин
Новосибирский государственный технический университет

Разрабатываемый язык фрагментированного программирования высокого уровня Freepal предназначен для создания параллельных фрагментированных программ. Программа на Freepal представляется в виде структурированных множеств фрагментов данных, кода и вычислений и не содержит привязки к конкретной архитектуре вычислительной системы или жестко заданного способа исполнения. За исполнение Freepal-программы отвечает runtime-система. Freepal-компилятор переводит инструкции программы в обращения к runtime-системам, при этом он строит управление на множестве фрагментов вычислений для корректного исполнения программы.

Введение

Развитие высокопроизводительных параллельных вычислительных систем требует наличия высокоуровневых средств создания эффективных параллельных программ.

Широко распространенные в настоящее время средства параллельного программирования, такие как MPI и OpenMP, являются инструментами низкого уровня, т.к. предоставляют только некоторый набор низкоуровневых примитивов для разработки параллельных программ.

Создание эффективной параллельной программы с помощью таких низкоуровневых средств требует от программиста существенных познаний во многих областях: от программирования до архитектуры параллельных вычислительных систем. Кроме того, такие динамические свойства программы, как балансировка загрузки или настройка на ресурсы, зачастую приходится проектировать и реализовывать заново для каждой новой задачи. Вследствие всего этого время и затраты на разработку и отладку программы существенно возрастают тем больше, чем сложнее является задача.

Наличие средств высокого уровня позволило бы решить эти и многие другие проблемы.

Фрагментированное программирование

Попытки создания высокоуровневых систем параллельного программирования предпринимаются уже давно. Большинство таких систем ос-

новано на идее расширения существующих последовательных языков дополнительными операторами/функциями параллельного программирования, предназначенными для порождения в программе ветвей для параллельного исполнения, синхронизации и т.д.

Другой подход используется в разрабатываемой технологии фрагментированного программирования [1]. В модели фрагментированного программирования программа представляется как множество объектов – фрагментов. Фрагменты подразделяются на три типа: фрагменты данных (данные, которые нужно обработать/вычислить), фрагменты кода (операции, которые можно применить к фрагментам данных) и фрагменты вычислений (указания, какие фрагменты кода применять к каким фрагментам данных и в каком допустимом порядке).

Фрагментированная программа является параллельной, если есть фрагменты вычислений, которые можно исполнить параллельно. В дальнейшем нас будут интересовать именно такие фрагментированные программы.

Особенностью технологии фрагментированного программирования является сохранение фрагментированной структуры программы во время ее исполнения, что дает возможность работать с фрагментами данных и вычислений как с отдельными объектами. Меняя размер, количество, распределение по вычислительным узлам и порядок исполнения фрагментов, возможно автоматическим образом обеспечить динамические свойства программы: коммуникации на фоне счета, настройку на ресурсы, динамическую балансировку нагрузки и т.д.

Исполнение фрагментированной программы на многопроцессорной вычислительной системе с разделяемой памятью является непростой задачей. Поэтому управление процессом исполнения программы возложено на runtime-систему [2, 3]. Использование runtime-системы позволяет иметь единый механизм исполнения для всех фрагментированных программ и обеспечить реализацию их динамических свойств.

Runtime-система содержит высокоуровневые функции для создания и исполнения фрагментов, их размещения и перемещения по вычислительным узлам и т.д. Тем не менее написание фрагментированных программ непосредственно с помощью интерфейса runtime-системы не обеспечивает нужный уровень абстракции и платформонезависимости. Желательно наличие специального языка высокого уровня как средства создания программ.

Язык Freepal

Язык Freepal является частью разрабатываемой системы фрагментированного программирования и предназначен для описания фрагментированных программ.

Основные особенности языка исходят непосредственно из модели фрагментированного программирования.

Во-первых, Freepal предназначен для задания множеств фрагментов данных, кода и вычислений (основных составляющих фрагментированных программ), их параметров и зависимостей между ними. Программа на Freepal не содержит жесткой привязки к определенной архитектуре вычислительных систем или строгой спецификации способа своего исполнения. Одна и та же Freepal-программа может быть исполнена на вычислительных машинах разного типа, с общей или разделяемой памятью, с разным числом процессоров и топологией связей узлов. Таким образом, на языке составляется только описание фрагментированного алгоритма, а его исполнение на конкретной вычислительной системе реализует скрытая от программиста runtime-система.

Во-вторых, Freepal является параллельным языком, т.к. ориентирован на описание множеств параллельно исполняющихся фрагментов вычислений. Тем не менее он использует некоторые черты последовательных императивных языков для упрощения написания параллельных программ, их разбора и отладки.

Фрагменты данных

В языке Freepal фрагмент данных играет роль переменной программы. Существуют два основных вида фрагментов данных: атомарные и структурированные.

Атомарный фрагмент данных – базовый неделимый объект программы. Для декларации атомарного фрагмента данных используется выражение **df** <Тип> <Имя>, где тип – один из базовых встроенных типов языка (для целых или вещественных чисел и т.д.). Если атомарный фрагмент данных – массив, для типа дополнительно указываются размеры массива.

В следующем примере a – целое число, b – массив из 10 вещественных чисел:

df int a;

df double[10] b.

Структурированный фрагмент данных – именованное множество фрагментов данных (атомарных или также структурированных). Приме-

ры: массив (множество однотипных индексированных элементов) или запись (множество элементов разного типа).

В следующем примере c – одномерный массив из 20 целых чисел, d – двумерный массив массивов вещественных чисел:

df int c[20];

df double[10] d[30, 50].

Главное отличие структурированных фрагментов данных состоит в том, что в ходе исполнения программы их элементы могут быть распределены по узлам мультикомпьютера и обрабатываться по отдельности.

Фрагменты кода и вычислений

Фрагмент кода в языке Freepal – функция, которая принимает входные фрагменты данных и вырабатывает из них выходные фрагменты данных. Фрагмент вычислений – вызов фрагмента кода с фактическими фрагментами данных в качестве аргументов.

Фрагменты кода, как и фрагменты данных, могут быть атомарными и структурированными.

Атомарный фрагмент кода является неделимой единицей исполнения и может принимать на вход и вырабатывать только атомарные фрагменты данных. При этом сам код, обрабатывающий данные, может быть написан на стороннем (последовательном) языке (таком, как C или Fortran). Это позволяет использовать внутри атомарных фрагментов кода уже существующие библиотеки оптимизированных функций. Можно сказать, что атомарный фрагмент кода во Freepal аналогичен элементарной операции в процедурном языке.

В отличие от атомарного структурированный фрагмент кода описывается с помощью выражений Freepal и может работать со структурированными фрагментами данных. Структурированный фрагмент кода – аналог процедуры в процедурном языке.

Кроме входных и выходных аргументов, атомарные и структурированные фрагменты кода могут иметь список параметров – дополнительный входной набор (скалярных) значений, которые могут использоваться как константы внутри кода.

Для декларации (атомарного/структурированного) фрагмента кода используется выражение **code**/**func** [<Выход>] <Имя> **<<**Параметры**>>** **(**<Вход>**)** <Тело>, где вход, выход и параметры – списки формальных аргументов в виде последовательностей <Тип> <Имя> (каждый из трех списков является необязательным и может отсутствовать), тело – код для исполнения.

Для декларации фрагмента вычислений используется выражение <Выход> = <Имя>**<**<Параметры>**>(**<Вход>**)**, где имя – имя вызываемого фрагмента кода, вход и выход – списки имен фактических фрагментов данных, параметры – список фактических значений параметров (размеры списков и типы фрагментов данных должны совпадать с указанными в декларации фрагмента кода).

Выражения языка

В языке Freepal поддерживаются следующие выражения для построения структурированных фрагментов кода:

- Декларация фрагмента данных.
- Декларация фрагмента вычислений.
- Блок – последовательность выражений, заключенная в фигурные скобки: **{** <Выр1> <Выр2> … <ВырN> **}**. Тело структурированного фрагмента кода является блоком. Если блок содержит декларацию фрагмента данных, этот фрагмент данных считается принадлежащим этому блоку и может использоваться только в его пределах. Блоки могут быть вложены друг в друга.
- Индексированное множество – описание множества выражений (чаще всего фрагментов вычислений) в виде **[**<Индексация>**]** <Выражение>, где индексация – последовательность конструкций <Переменная индексации>**=**<Нижний предел>**..**<Верхний предел> (нижний и верхний пределы могут задаваться произвольными арифметическими выражениями). Для каждой комбинации значений переменных индексации из указанных пределов порождается копия выражения (для которой переменные индексации являются параметрами-константами с соответствующими значениями). Аналог такого выражения в других языках – параллельный цикл for.
- Итерационное множество – описание счетного (потенциально бесконечного) множества: **while** <Логическое условие> <Выражение>. Если условие истинно, выражение исполняется, после чего условие проверяется снова и т.д. Исполнение завершается, когда условие становится ложным. Аналог цикла while.
- Условный оператор: **if** <Логическое условие> <Выр1> [**else** <Выр2>]. В зависимости от истинности условия исполняется первое или второе выражение. Часть с **else** может отсутствовать.

Порядок исполнения

Важным моментом во Freepal является определение допустимого порядка исполнения фрагментов. Так как язык является параллельным, по умолчанию предполагается, что все объявленные фрагменты вычислений могут быть исполнены параллельно.

Однако такое полностью параллельное исполнение практически не встречается в реальных задачах. В подавляющем большинстве случаев между фрагментами вычислений имеются зависимости по фрагментным данным (например, в виде производитель-потребитель). В связи с этим в языке используется следующий принцип: если один фрагмент вычислений вырабатывает фрагмент данных, а далее в тексте программы находится другой фрагмент вычислений, который его потребляет, между ними устанавливается порядок по исполнению, контролирующий, чтобы второй фрагмент вычислений запустился только после (но не обязательно сразу) завершения исполнения первого. Этот принцип позволяет обеспечить корректный порядок исполнения для зависимых фрагментов вычислений и в то же время сохранить параллельное исполнение для остальных независимых фрагментов вычислений.

Обнаружение зависимостей и построение порядка исполнения производится компилятором Freepal при компиляции фрагментированной программы. Если не все зависимости можно определить при компиляции, построение и уточнение зависимостей производится во время исполнения runtime-системой. Уточнение может производиться до уровня атомарных фрагментов данных и вычислений.

Для тех случаев, когда нужно изменить порядок исполнения по умолчанию, могут использоваться следующие операторы языка:

- **<<**: <ФВ1> **<<** <ФВ2> – последовательное исполнение фрагментов вычислений, даже если между ними нет явных зависимостей по данным.
- ||: <ФВ1> || <ФВ2> – параллельное исполнение фрагментов вычислений, даже если между ними есть зависимость. Может применяться в случае редукции в один фрагмент данных, когда порядок вычисления элементов редукции не важен (runtime-система обеспечит корректный доступ к переменной редукции).

Компиляция и исполнение

При поддержке runtime-системой основных конструкций языка компиляция Freepal-программы может быть произведена путем трансляции выражений языка в обращения к runtime-системе. Таким образом, для

каждого фрагмента кода генерируется его эквивалент, использующий функции runtime-системы, а исполнение фрагментов вычислений заключается в исполнении такого сгенерированного кода.

Из этого следует, что наличие эффективной runtime-системы [3] обеспечивает эффективность самого языка в целом.

Заключение

Разработаны синтаксис, семантика и основные принципы языка фрагментированного программирования Freepal.

В дальнейшем планируется исследование и разработка алгоритмов и приемов оптимизации фрагментированной Freepal-программы на стадии компиляции.

Литература

1. **Kraeva M.A., Malyshkin V.E.** Assembly Technology for Parallel Realization of Numerical Models on MIMD-Multicomputers // Int. Journal on Future Generation Computer Systems. – 2001. – Vol. 17, №. 6. – P. 755–765.
2. **Kalgin K.V., Malyshkin V.E., Nechaev S.P., Tschukin G.A.** Runtime System for Parallel Execution of Fragmented Subroutines // PACT–2007, LNCS. – 2007. – Vol. 4671. – P. 544–552.
3. **Щукин Г.А.** Runtime-система FP RTS управления исполнением фрагментированных программ на мультикомпьютере // Сборник докладов Пятой Сибирской конференции по параллельным и высокопроизводительным вычислениям, Томск, 1–3 декабря 2009. – Томск: Изд-во Том. ун-та, 2009. – С. 182–186.