УДК 004.432.42

СТРАТЕГИИ ОПТИМИЗАЦИИ ВЫПОЛНЕНИЯ ФУНКЦИОНАЛЬНО-ПОТОКОВЫХ ПАРАЛЛЕЛЬНЫХ ПРОГРАММ

Цветцих Д.В.
Научный руководитель – д.т.н., профессор Легалов А.И.
Сибирский федеральный университет

В настоящее время проблема создания параллельных программ, переносимых между различными параллельными вычислительными системами (ПВС). Поскольку ПВС с общей памятью в виде компьютеров с многоядерными процессорами распространены повсеместно, наибольший интерес представляет решение задачи переноса параллельных программ между такими ПВС. Идея, определяющая направление исследований в данной области, заключается в следующем: абстрагировать программиста от ручного управления ресурсами. Это требует создания языка программирования, обеспечивающего распараллеливание за счет собственной семантики и «исполнителя», способного эффективно выполнять программы на заданной ПВС. Уже существует язык программирования «Пифагор», который позволяет создавать архитектурно-независимые программы, обладающие максимальным параллелизмом. Однако задача эффективного выполнения архитектурно-независимых программ на конкретных ПВС остается нерешенной.

Причина в том, что программы, написанные на «Пифагоре», обладают определенной спецификой, так как они написаны в функционально-потоковом стиле. Данный стиль предполагает выполнение программ по готовности данных, а не по заранее определенному порядку следования операторов. Поэтому функционально-потоковые программы имеют следующую особенность: они поддерживают максимальный параллелизм на уровне команд за счет средств языка программирования. Языковое средство, благодаря которому доступна эта возможность, называется параллельным списком. Параллельный список группирует задачи, которые могут быть выполнены асинхронно. Рассмотрим на примере задачи о ханойских башнях.

```
rHanoy << funcdef X {
  src << X:1;    dst << X:2;   n   << X:3;
  return << .^ [(n,0):(<=, >):?]^
  (
    {.},
    {
     ( [ // формирование параллельного списка из трех задач
     (src, ((3,src):-,dst):-, (n,1):-):rHanoy:[],
     (src, dst),
     (((3,src):-,dst):-, dst, (n,1):-):rHanoy:[]
     ] )
    }
  )
};
```

В программе параллельный список обозначается квадратными скобками. Формируется параллельный список, который содержит три независимые задачи, затем идет рекурсивный вызов функции rHanoy. После этого дробление продолжается, и формируется дерево задач (рис. 1). Разделение задачи на части продолжается рекурсивно до тех

пор, пока это дает выигрыш. Здесь нужно отметить отличие функционально-потоковой программы от последовательной программы. Если в последовательной программе можно явно ограничить глубину рекурсии, тем самым определив размер минимальной задачи, которая уже не будет разбиваться на части, то в функционально-потоковой программе этого делать нельзя. Дело в том, что функционально-потоковая программа должна обладать максимальным параллелизмом, чтобы обеспечить нагрузку на ресурсы любой ПВС. Поэтому размер неделимой задачи должен определяться «исполнителем», исходя из архитектуры ПВС, на которой происходит выполнение программы. Кроме того, во время выполнения программы тот же самый исполнитель должен оценивать количество операций, которое требуется для любой задачи из дерева. Если это количество операций меньше заданного минимума для данной ПВС, дальнейшее дробление задачи на подзадачи не происходит.

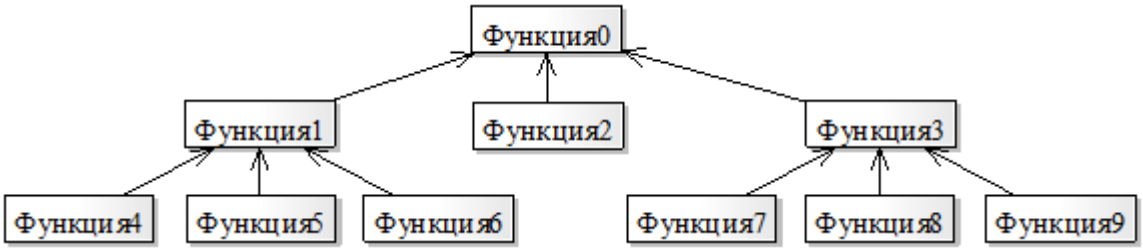

Рис. 1. Дерево задач, формируемое функцией rHanoy

После формирования древовидной нагрузки появляется другая задача – найти способ обхода дерева, который позволит уменьшить количество используемой памяти. Рассмотрим обход дерева в глубину и в ширину.

При обходе в ширину идем от листьев к корню дерева. Такой обход обеспечивает максимальный параллелизм, однако требует много памяти, поскольку сначала нужно полностью построить дерево задач. Обход дерева в глубину использует мало памяти, однако не обеспечивает параллелизма. Поэтому при обходе дерева в глубину нужно говорить не об одном универсальном параллельном исполнителе, а о нескольких независимых последовательных исполнителях, а также о диспетчере, который распределяет задачи между исполнителями.

Исходя из сказанного выше, опишем схему исполнителя функционально-потоковых программ. Он состоит из диспетчера и нескольких независимых интерпретаторов языка программирования «Пифагор». При запуске исполнителя диспетчер анализирует архитектуру ПВС. При этом учитывается количество процессоров и объем памяти ПВС. На основе анализа вычисляется размер минимальной задачи, которая не подвергается декомпозиции на подзадачи. Кроме того, создается оптимальное для данной ПВС число последовательных интерпретаторов «Пифагора». Каждый интерпретатор имеет собственное дерево задач, которое он обходит в глубину. По мере построения дерева динамически определяется количество операций, необходимое для выполнения данной задачи. Кроме того, диспетчер осуществляет динамическую балансировку нагрузки. Если у одного интерпретатора большое количество задач, а другой уже выполнил все свои задачи, то происходит перехват задачи из дерева задач занятого потока (рисунок 2).

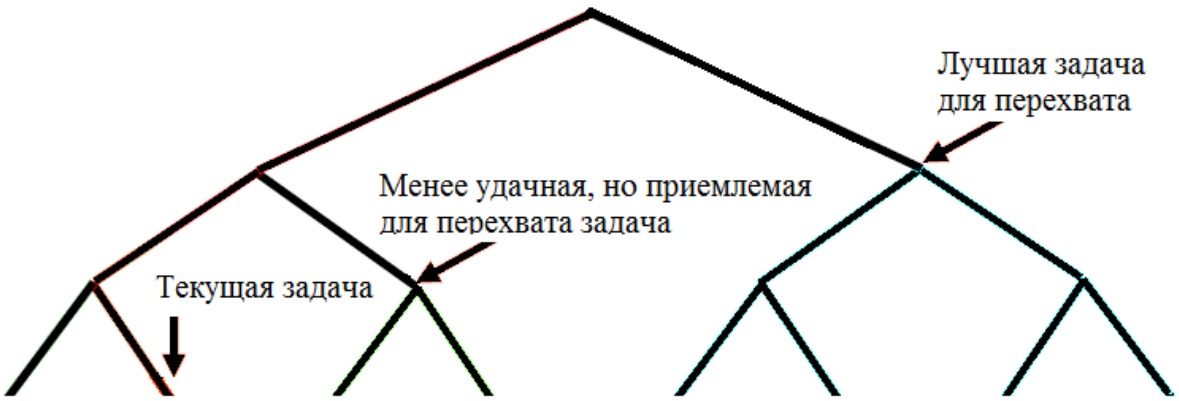

Рис. 2. Выбор задачи оптимальной для перехвата

Для передачи на выполнение другому интерпретатору выбирается задача, наиболее «далекая» от выполняемой в настоящее время задачи.

В заключение стоит отметить, что функционально-потоковое программирование, с одной стороны, предлагает удобный и универсальный механизм создания архитектурно-независимых параллельных программ. С другой стороны, привязка такой программы к ресурсам для ее эффективного выполнения сопряжена с определенными сложностями. Однако, как показал анализ, эти проблемы являются решаемыми, поэтому функционально-потоковое программирование в перспективе может выйти за рамки научного инструмента, предназначенного для отработки проблем, связанных с созданием переносимых параллельных программ, и стать полноценным промышленным стандартом.