Разработка инструментальных средств анализа программ в среде рекурсивно-параллельного программирования

А.Г. Седов
Ярославский государственный университет им. П.Г. Демидова
г. Ярославль
Email: agsedov@gmail.com

Аннотация

В докладе описана модель выполнения программы RPC, называемая графом трассы, и разработанные средства его построения и визуализации. Дано формальное определение для графа трассы в терминологии схем рекурсивно-параллельной(РП) программ, рассмотренных в [1]. Сделан обзор изменений, которые планируется сделать как в самой модели выполнения РП-программы, так и в процессе ее построения.

I. Введение

Для проектирования, верификации и анализа параллельных и распределенных систем известен ряд подходов. Сложнее обстоит дело с рекурсивно параллельным методом программирования. Для развития этого метода была разработана концепция рекурсивно-параллельного языка С (сокращенно RPC) [2],[3]. Он обладает следующими свойствами:

- является языком параллельного программирования, ориентированным на архитектуру виртуальной мультипроцессорной ВС с динамическим распараллеливанием, именуемой в дальнейшем рекурсивно-параллельной машиной (сокращенно RPM);
- параллельная программа на языке RPC может быть транслирована стандартным компилятором С как в параллельный исполняемый код, выполняемый на RPM, так и последовательный исполнимый код, выполняемый на обычной последовательной ЭВМ;
- является средством для исследования параллелизма программ (или их моделей), а также - эффективности функционирования RPM при выполнении данных программ (или их моделей).

Для реализации языка RPC была создана среда рекурсивно параллельного программирования RPM Shell [4].

Существенной частью этой среды должна стать поддержка наработки и визуализации графа трассы. Граф трассы представляет собой модель выполнения РП программы. Он используется для решения комплекса задач, связанного с изучением свойств рекурсивно-параллельной программы и её отладкой: моделированием, исследованием потенциального параллелизма, сбором и визуализацией статистических данных.

II. Структура программы RPC.

Рекурсивно-параллельная программа представляет из себя иерархическое множество процедур двух типов, допускающих рекурсивный вызов.

Вычислительный процесс, протекающий в вычислительной системе, поддерживающей РП-стиль программирования, является иерархическим параллельным процессом. Компоненты этого процесса есть активации процедур, а также некоторые системные функции. Активацией мы будем называть вызов параллельной процедуры с конкретными значениями параметров. Любая компонента, соответствующая активации процедуры, в свою очередь может быть иерархическим параллельным процессом.

Активация процедуры является параллельной, если ее вызов осуществляется через специальный оператор порождения параллельного процесса (PCall). После вызова параллельной процедуры вычисления в родительской процедуре продолжаются без приостановки до точки синхронизации (до оператора Wait()).

Возврат из дочерней параллельной процедуры в родительскую процедуру осуществляется в точку синхронизации. Вычисления в дочерних процедурах и родительской процедуре могут выполняться параллельно. В точке синхронизации происходит ожидание завершения всех дочерних параллельных процессов, запущенных до момента попадания в данную точку родительской процедуры.

Одной из характерных особенностей RPC является использование блока параметров, содержащего локальные данные для передачи из вызывающей процедуры в вызываемую и обратно. В вызывающей процедуре определяется имя локальной переменной данного типа (имя блока параметров), через которое будет осуществляться

доступ к элементам блока параметров. Обращение к элементам блока параметров в дочерней процедуре можно производить только посредством макрокоманды P_(elem), где elem - имя элемента структуры.

Пример 1: Структура РП-программы для явного решения уравнения теплопроводности методом конечных разностей. Алгоритм решения заключается в циклическом вычислении массива распределения температур на основании предыдущего массива и массивов начальных условий (функция main). Процедура NextLayer либо рекурсивно делит переданный ей фрагмент массива на части (их количество определяется параметром branching), либо извлекает массив из общей памяти и производит вычисления.

```
struct NLParam
{
  int begin,end,min;
  int branching;
  float tau,h;
}
parallel(NextLayer, NLParam)
{
 if (end - begin > min)
 {//Если предел деления не достигнут
   for(int i=0;i<branching;i++)
   {
     struct NLParam pbl;
     //...Копирование блока параметров во вновь созданный блок pbl.
     //Пересчет начала и конца вектора:
     pbl.begin = begin + i*(end-begin)/branching;
     pbl.end = begin + (i+1)*(end-begin)/branching;
     PCall(NextLayer, &pbl);
   }
   Wait(); //Синхронизация вызовов.
 }
 else
 {
   //Извлечение из памяти фрагмента массива
   //Вычисления, запись результатов во временный массив в общей памяти.
 }
}
int main(...)
{
 //...Создание массива распределения температуры в общей памяти
 //...Создание временного массива того-же размера
 for(int i=0;i<IterCount;i++)
 {
   PCall(NextLayer, &pbl);
  Wait();
  //...Вывод нового массива распределений в файл
  //...Запись результата из временного массива в массив распределений
 }
}
```

III. Построение графа трассы

Граф трассы — это ориентированный граф, множество вершин которого представляет собой множество событий системы, а множество рёбер — отношение частичного линейного порядка между ними.

В ходе выполнения RPM программы фиксируются следующие основные типы событий:

- операции для работы с памятью: выделение, освобождение, запись, чтение;
- линейный участок (lk) — объединение всех прочих вычислений, происходящих внутри активации;
- ветвление (fork) — один или несколько параллельных вызовов RPM-процедуры;

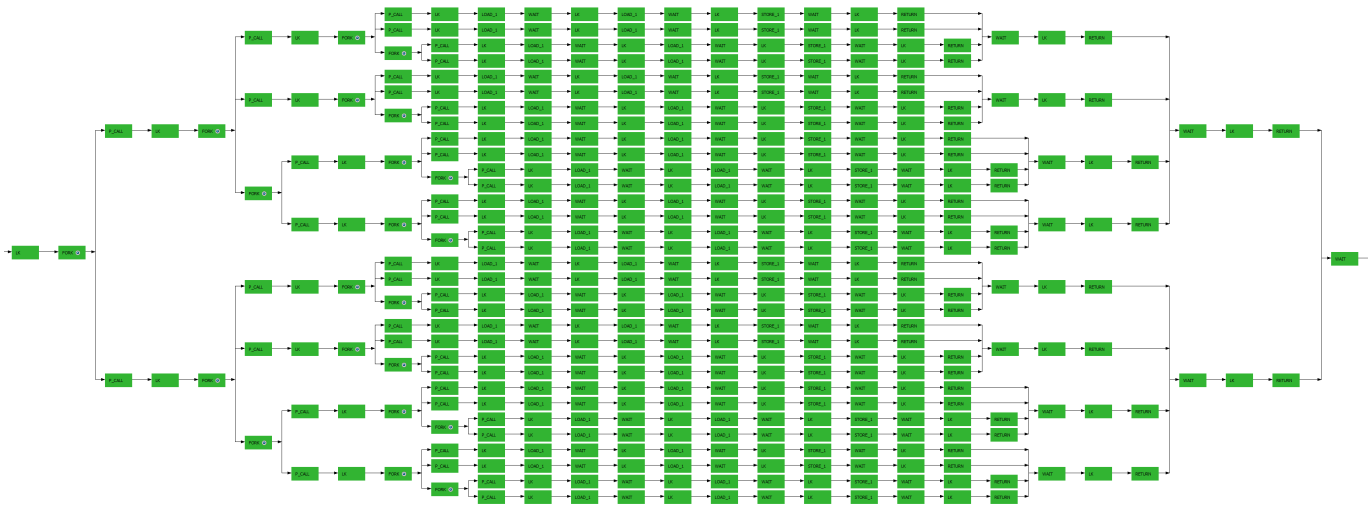

Рис. 1. Визуализация графа трассы в RPC-Viewer.

- синхронизация (wait) — вызов оператора Wait();

В файле графа трассы вместе со сведениями о связях между вершинами, для каждого типа событий сохраняется специфическая информация, необходимая для последующего моделирования выполнения программы. Так, для линейного участка вычислений сохраняется время его выполнения в тактах процессора. В вершинах, соответствующих операторам доступа к общей памяти сохраняется количество читаемых/записываемых элементов, их размер.

В предыдущих версиях среды RPM уже были реализованы библиотека для наработки графа трассы, модуль визуализации графа, программа имитационного моделирования и визуализатор статистики. С течением времени эти решения устарели. Однако, идея РП-программирования по-прежнему сохраняла актуальность, а ключевой элемент среды – библиотека для выполнения рп-программ на сети модернизировалась. Возникла задача реализовать эти возможности заново. Первым этапом работы стала библиотека трассировки графа и его визуализатор. Для того, чтобы обеспечить совместимость с уже существующими средствами работы с графом, было решено сохранить старый бинарный формат его вывода.

На данный момент наработку графа трассы осуществляет библиотека функций для последовательного выполнения RPC-кода. В ней реализованы все функции RPC, с добавлением вызовов специальных операторов для сбора статистики. Построение графа трассы происходит в ходе выполнения программы, вершина за вершиной.

Для того, чтобы наглядно представить граф трассы, был написан визуализатор RPC-Viewer. Он выводит граф на экран в виде своеобразной иерархической структуры, позволяя сворачивать/разворачивать конкретные активации, изучать содержимое выбранных пользователем вершин.

IV. Модель RPPS

Задачей первого этапа работы была наработка графа трассы в формате, полностью совместимом с существовавшими прежде средствами анализа. Однако, требование совместимости серьезно ограничивает возможности библиотеки. В частности, из-за недостатка информации отсутствует возможность связать структуру графа трассы с кодом изначальной программы, определить в визуализаторе, какой конкретно рекурсивно-параллельной функции соответствует вызов активации.

Новые возможности и задачи возникают также в связи с потенциальным расширением функциональности языка RPC, и написанием транслятора в RPC-код из языка-расширения. Во-первых, транслятор должен существенно упростить алгоритм наработки графа, добавляя в код вспомогательные операторы трассировки. Во-вторых, потребуется отразить в структуре графа новые конструкции.

Для того чтобы описать в одних терминах структуру рекурсивно-параллельной программы и конкретное ее выполнение рассмотрим модель RPPS (Схема рекурсивно-параллельной программы).

Схемой программы называется конечный граф с корнем $G = (Q_G, q_0, \mapsto_G, L_G)$,где

- $Q_G = \{q_0, q_1, ..., q_n\}$ — конечное множество вершин; Каждая вершина является узлом одного из пяти типов: действия, выбора, вызова, синхронизации, типа end;
- q_0 - начальная вершина (корень);
- $Q_G \to Q_G^*$ - функция, позволяющая для каждого $q \in Q_G$ найти вершины — преемники вершины q;

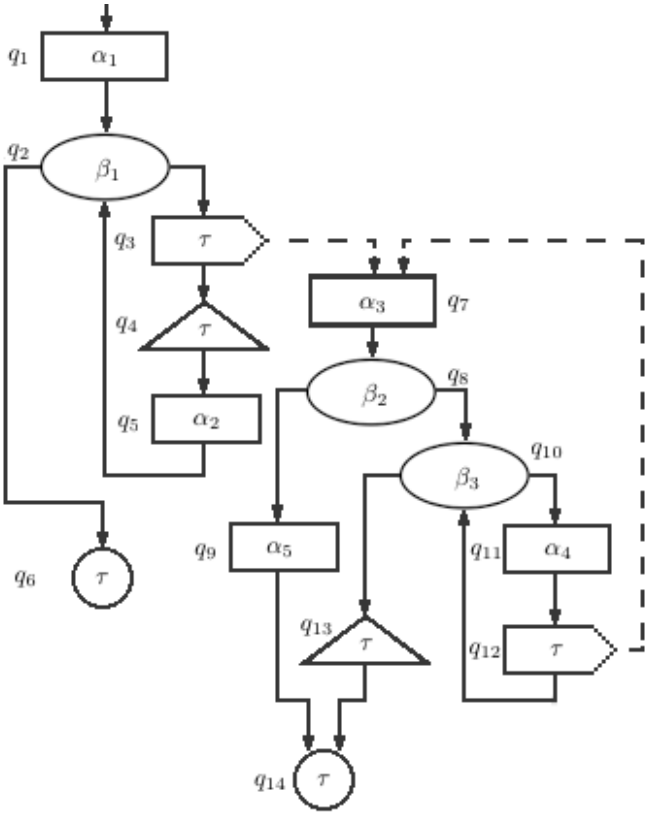

Рис. 2. Модель RPPS для программы из примера 1.

- $L_G : Q_G \to \Lambda_\tau \times Q_G^*$ маркировка вершин графа номерами из Λ_τ;

Вершина q' является преемником (последователем) узла q, когда $\mapsto_G$ связывает узел q с другим узлом q'. Вершины типа end не имеют преемников. Вершины типа pcall связаны с двумя вершинами, одна из них есть преемник, а другая - вызываемая вершина. Схема программы для примера (1) изображена на рисунке 2.

Множеством иерархических состояний схемы $G \in RPPS_\Lambda\tau$ называется наименьшее множество $M_G = \{\xi, \zeta, \varphi, ...\}$ такое, что если

- $q_1, ..., q_n$ есть вершины из Q_G
- $\xi_1, ..., \xi_n \in M_G$

тогда мультимножество $\xi = \{(q_1, \xi_1), ..., (q_n, \xi_n)\}$ принадлежит множеству иерархических состояний M_G.

Выполнение РП-программы можно описать последовательностью иерархических состояний $X = \xi_1 \longmapsto \xi_2 \longmapsto \xi_3 ... \xi_n$.

Дадим теперь формальное определение графу трассы.

Графом трассы рекурсивно-параллельной программы будем называть конечный граф $T = (Q_T, q_0, \mapsto_T)$, где

- $Q_T = \{q_0, q_1, ..., q_n\}$ - конечное множество вершин, при этом существует функция $f : Q_T \to Q_G$, ставящая в соответствие каждой вершине из Q_T вершину из Q_G.
- q_0 - начальная вершина (корень);
- $\delta : Q_T \to Q_T^*$ — функция, позволяющая для каждого $q \in Q_T$ найти вершины-преемники вершины q;
  Если $f(q_i)$ — вершина типа выбора, действия, синхронизации, то $|\delta(q_i)| = 1$. Если $f(q_i)$ — вершина типа end, то $|\delta(q_i)| \leq 1$

Структура T будет отличаться от существующей сейчас структуры прежде всего наличием нового элемента, который раньше игнорировался при наработке графа трассы — действия выбора. RPPS для RPC кода будет строиться при помощи транслятора. G_T будет демонстрироваться в визуализаторе вместе с T.

V. Заключение

Таким образом, в работе рассмотрены подходы к созданию среды РП-программирования. Их осуществление предполагает:

- реализацию построителя модели по заданному коду RPC;
- дальнейшую доработку формата графа трассы и визуализатора.
- разработку средств имитационного моделирования;
- изучение возможностей построенных инструментов для верификации RPC-программ;
- расширение возможностей языка RPC;

Построение верифицирующего транслятора предполагает использование таких формализмов, как иерархическая система взаимодействующих автоматов и вложенные сети Петри [5]. Описанные средства моделирования и визуализации имеют значение для дальнейшего развития языка RPC и разработки его расширений[6].

Список литературы

[1] О. Б. Кушнаренко *Семантика рекурсивно-параллельных программ и методы их анализа* // Диссертация для получения звания доктора Университета Жозефа Фурье - Гренобль и Ярославского государственного университета, 1997.

[2] Badin N.M, Brodsky G.M., Sokolov V.A. *A Recursive Parallel Programming Language and its application to algebraic computations* // Joint NCC and IIS Bulletin Computer Science Vol. 11, Институт систем информатики им. Ершова, СО РАН, Новосибирск, Россия, 1999, стр. 1-14.

[3] Бадин Н.М., Бродский Г.М., Соколов В.А. *Языковые средства рекурсивно-параллельного программирования* // Сб. научных трудов "Актуальные проблемы современной математики", т.3. – Новосибирск: НИИ МИОО, 1997, с. 19-28.

[4] В. В. Васильчиков *Средства параллельного программирования для вычислительных систем с динамической балансировкой загрузки*// Ярославль, ЯрГУ, 2001.

[5] И. А. Ломазова. *Вложенные сети Петри: моделирование и анализ распределенных систем с объектной структурой*//Москва, Научный мир, 2004.

[6] A. G. Sedov. *Execution Analysis of ARPC Programs in the Environment of the Recursive Parallel Programming*//Proceedings of the 6th Spring/Summer Young Researchers Colloquium on Software Engineering, Perm 2012.