ПОСТРОЕНИЕ ИНФОРМАЦИОННОГО ГРАФА ПАРАЛЛЕЛЬНОЙ ПРОГРАММЫ НА ОСНОВЕ ДАННЫХ ПРОФИЛИРОВАНИЯ И ТРАССИРОВКИ

Е.А. Киселев

1. Введение

Исследования существующих способов повышения эффективности использования вычислительных и коммуникационных ресурсов многопроцессорных систем привели к постановке задачи нахождения оптимального распределения ветвей параллельной программы по процессорам. В зависимости от выбранного критерия оптимизации, решение задачи может быть направлено как на уменьшение времени выполнения параллельных программ, так и на более полную утилизацию коммуникационных и вычислительных ресурсов системы.

При решении задачи оптимального размещения необходимо учитывать как архитектурные особенности вычислительной системы (ВС), так и структуру параллельной программы. Параллельная программа представляется в виде *информационного графа*, вершины которого соответствуют ветвям, а ребра характеризуют взаимодействие между ними.

Данная работа направлена на разработку подхода к автоматизированному построению информационного графа параллельной программы. Во втором разделе статьи рассматриваются существующие подходы к построению информационных графов программ и способы вычисления их характеристик. Третий раздел посвящен описанию предложенного подхода к построению информационного графа на основе данных профилирования и трассировки параллельной программы. В заключении приводятся сведения об апробации полученных результатов на ВС Межведомственного Суперкомпьютерного Центра Российской Академии наук МВС-100К.

2. Способы и средства построения информационного графа

В общем случае информационный граф параллельной программы представляется как связный неориентированный граф $G=(X,U)$, где X – множество вершин, соответствующее ветвям программы, U – множество ребер, представляющих информационные связи между ветвями. На множестве вершин графа задана весовая функция $w(X)$. Вес w_i вершины x_i характеризует объем вычислений, выполненных ветвью программы с номером i (*вычислительная составляющая*). На множестве ребер графа задана весовая функция $c(U)$. Вес c_{ij} ребра $(x_i,x_j)\in U$ отражает характер информационного обмена между *i*-й и *j*-й ветвями программы (*коммуникационная составляющая*).

Вычислительная составляющая параллельной программы может быть выражена либо количеством вычислительных операций, выполняемых каждой ветвью, либо временем выполнения вычислительных операций. В первом случае предполагается, что все вычислительные операции программы однородны и оказывают одинаковую нагрузку на вычислительный ресурс. При использовании второго подхода предполагается, что ветви программы выполняются на однородных вычислительных ресурсах.

Коммуникационная составляющая, как правило, выражается в виде объема данных, переданных между ветвями за время работы программы, либо соответствует интенсивности выполнения операции обмена данными.

Можно выделить три подхода к построению информационного графа параллельной программы:

1. *На основе данных о работе программы, полученных от пользователя.* Информационный граф программы формируется на основе представления пользователя о работе программы.
2. *На основе результатов профилирования и трассировки.* Информационный граф формируется на основе данных, подготовленных программными средствами профилирования и трассировки.
3. *На основе данных мониторинга коммуникационных и вычислительных ресурсов ВС*. Информационный графа параллельной программы строится на основе данных, полученных от специализированных средств мониторинга состояния вычислительных и коммуникационных ресурсов ВС. Данные средства протоколируют процесс выполнения каждой ветви программы и позволяют оценить взаимодействия между ними путем анализа сетевого трафика.

Первый подход является достаточно трудоемким, поскольку требует непосредственного участия пользователя при построении информационного графа, а так же менее точным, т. к. пользователь указывает все характеристики вершин и ребер на свое усмотрение.

При использовании второго подхода, для анализа работы параллельных программ могут применяться различные средства профилирования и трассировки, результаты которых обрабатываются пользователем вручную или с использованием средств визуализации (например, *Upshot* [1], *Jumpshot* [2], *Paraprof 3D Profiler* [3], *Vampir* [4] и др). Подобные средства могут быть использованы для построения информационного графа

параллельной программы на основе анализа временных диаграмм, графов межпроцессорных обменов и вызовов функций. При таком подходе требуется участие пользователя, однако характеристики информационного графа являются более точными. В качестве примера использования подхода можно привести работу [5], где проводилась оценка использования коммуникационных операций в MPI-приложениях на основе данных профилирования и трассировки. Автором была разработана программа-анализатор, вычисляющая объем коллективных операций и операций типа точка-точка, длительность их выполнения и объем переданной информации.

Третий подход позволяет исключить участие пользователя в процессе построения информационного графа, однако требует наличия специализированных аппаратно-программных средств, поставляемых разработчиками ВС.

В настоящее время существует широкий набор средств профилирования и трассировки параллельных программ, которые входят как в состав сред разработки (Intel Trace Analyzer and Collector, GNU Gprof Profiler, IBM X-Windows Performance Profiler), так и распространяются отдельно (VampirTrace [4], TAU Performance System [3], MPI Parallel Environment [6]). Как правило, такие программные средства являются кроссплатформенными и совместимы с различными моделями параллельного программирования: MPI, OpenMP, Shmem. В современных средствах трассировки существуют механизмы фильтрации вызовов функций и сжатия данных трассировки, что позволяет существенно уменьшить объем генерируемой информации и снизить влияние на процесс выполнения программы. Использование данных средств позволяет получить необходимую информацию для построения информационного графа параллельной программы.

3. Построение информационного графа на основе данных профилирования и трассировки

Для того чтобы исключить участие пользователя на этапе анализа данных профилирования, трассировки и формирования информационного графа, было разработано программное средство, учитывающее как коммуникационную, так и вычислительную составляющую параллельной программы. Для описания параллельной программы была выбрана следующая модель. Пусть программа состоит из N ветвей и выполняется в течение времени T. Каждой ветви ставится в соответствие уникальный номер $i \in [0;N-1]$. Обозначим за T_i – время выполнения ветви с номером i; $Tcomm_i$ – суммарное время выполнения коммуникационных операций ветви i. Тогда вычислительная составляющая ветви i может быть определена следующим образом:

$$w_i = 1 - \frac{Tcomm_i}{T_i} \quad (1)$$

Обозначим за v_{ij} объем данных, переданных за время T, а за s_{ij} – общее число операций приема/передачи данных между ветвями с номерами i и j. Тогда характеристика информационного обмена между ветвями i и j может быть выражена через средний объем данных, переданных за одну операцию приема/передачи данных:

$$c_{ij} = \frac{v_{ij}}{s_{ij}} \quad (2)$$

При построении информационного графа веса вершин и ребер вычисляются в соответствии с выражениями (1) и (2) на основе следующих данных профилирования и трассировки параллельной программы:

- время выполнения параллельной программы;
- время выполнения каждой ветви программы;
- объем данных, переданных между ветвями за весь промежуток времени;
- количество и длительность операций приема/передачи данных.

Поскольку в процессе трассировки учитываются все пересылки между ветвями программы, включая передачу служебной информации (например, при синхронизации ветвей), при увеличении числа ветвей в параллельной программе размер информационного графа может существенно увеличиваться. Это затрудняет его хранение и использование при решении задачи распределения ветвей параллельной программы по процессорам ВС. Для преодоления данного ограничения можно перестраивать информационный граф для различных топологий сети (общая шина, 2D-решетка, 3D-тор, дерево, кольцо, звезда) исключая лишние ребра графа и пересчитывая весовые характеристики оставшихся. Применение данного подхода к формированию информационного графа программы позволяет уменьшить размер информационного графа и дает возможность оценить вероятность возникновения конкуренции за коммуникационные ресурсы между ветвями параллельной программы для ВС с различной структурой и топологией сети.

В настоящей работе рассматривается подход, основанный на построении покрывающего дерева максимального веса $G_T(X,T)$ графа $G(X,U)$. Задача состоит в нахождении подмножества $T \subset E$, связывающего все вершины из X, для которого суммарный вес (3) максимален.

$$c(T) = \sum_{(i,j) \in T} c_{ij} \quad (3)$$

Пусть задано отображение $\Gamma: X \to X$ (т. е. если $\Gamma(x_i)=x_j$, то $\exists(x_i, x_j)\in U$). Алгоритм построения покрывающего дерева G_T графа G основан на алгоритме Прима [15] и состоит из следующих шагов:

Инициализация. Пусть $X_1=\{x_s\}$, где x_s произвольно выбранная вершина из $X, T_1=\varnothing$.

Основной цикл.

Шаг 1. Для каждой вершины $x_j\notin X_1$ найти вершину $a_j\in X_1$, такую, что

$$c(a_j, x_j)=\max_{x_i\in X_1}\{c(x_i, x_j)\}=b_j$$

и приписать вершине x_j пометку $[a_j, b_j]$, где a_j – лучшая для x_j в смысле максимума веса инцидентного ребра (a_j, x_j) вершина из X_1, а b_j – вес этого ребра. Если такой вершины a_j нет, т. е. $\Gamma(x_j)\cap X_1=\varnothing$, приписать вершине x_j пометку $[0,\infty]$.

Шаг 2. Выбрать такую вершину x_j^p, что

$b_j^p=max\{b_j\}, x_j\notin X_1$

Обновить данные: $X_1=X_1\cup x_j^p$, $T_1=T_1\cup(a_j^p, x_j^p)$. Если $|X_1|=n$, то останов. Ребра в T_1 образуют покрывающее дерево максимального веса. Если $|X_1|\neq n$, то перейти к шагу 3.

Шаг 3. Для всех $x_j\notin X_1$, таких, что $x_j\in\Gamma(x_j^p)$, обновить пометки следующим образом:

a) если $b_j<c(x_j^p, x_j)$, то положить $b_j=c(x_j^p, x_j), a_j=x_j^p$ и вернуться к шагу 2

b) если $b_j\geqslant c(x_j^p, x_j)$, то перейти к шагу 2.

Вычислительная сложность данного алгоритма не превышает $O(n^2)$.

Веса ребер дерева G_T информационного графа G пересчитываются следующим образом. Пусть вершине x_j графа G инцидентны ребра $(x_j, x_1), (x_j, x_2), \ldots, (x_j, x_k)$, где $k\leq|E|$, расположенные в порядке возрастания веса (т. е. $c(x_j, x_1)\leqslant c(x_j, x_2)\leqslant\ldots\leqslant(x_j, x_k)$). Поскольку для любых двух вершин покрывающего дерева G_T существует единственный путь, вес ребра (x_j, x_k) можно пересчитать в соответствии с (4).

$$c(x_j, x_k)=\sum_{s=1}^{k} c_{(x_j, x_s)} \qquad (4)$$

Рассмотренный подход был реализован в виде программного средства построения информационного графа MPI-приложения IGtrace. Вершины информационного графа G соответствуют MPI-процессам приложения, ребра графа представляют информационные связи между ними.

В качестве формата представления данных профилирования и трассировки был выбран OTF[4]. Формат поддерживается широким набором средств профилирования и трассировки (включая такие программы как TAU Performance System и VampirTrace). Информационный граф MPI-приложения или его покрывающее дерево могут быть сохранены в одном из двух форматов: graphML[7] или DOT[8]. Форматы поддерживаются различными библиотеками программирования, ориентированными для работы с графами (Boost graph library [9], igraph [10], R [11]) и редакторами графов (Graphviz [12], yWorks yEd, Gephi [13], Visone [14] и др.).

Процесс построения информационного графа приложения с помощью IGtrace осуществляется в два этапа (рис. 1). На первом этапе необходимо подготовить файл трассы приложения. Для этого можно использовать любое средство трассировки, поддерживающее формат OTF. В качестве примера можно рассмотреть программу VampirTrace. На этапе сборки MPI-приложения VampirTrace автоматически встраивает в исходный текст программы функции, необходимые для формирования файлов трассы. После запуска приложения на ресурсах ВС, в рабочем каталоге программы будет сформирован набор файлов трассы в формате OTF. На втором этапе выполняется запуск программы IGtrace с указанием имени сформированного otf-файла; формата файла, для хранения информационного графа: graphML или Dot, а так же типа информационного графа: полный граф или покрывающее дерево.

Для просмотра и редактирования файла, содержащего описание информационного графа может быть использован любой редактор графов, поддерживающий форматы graphML или Dot. Примеры информационных графов MPI-приложений, построенных с помощью разработанного программного средства, приведены на рисунках 2 и 3. Жирными линиями на рисунках выделены ребра покрывающего дерева максимального веса.

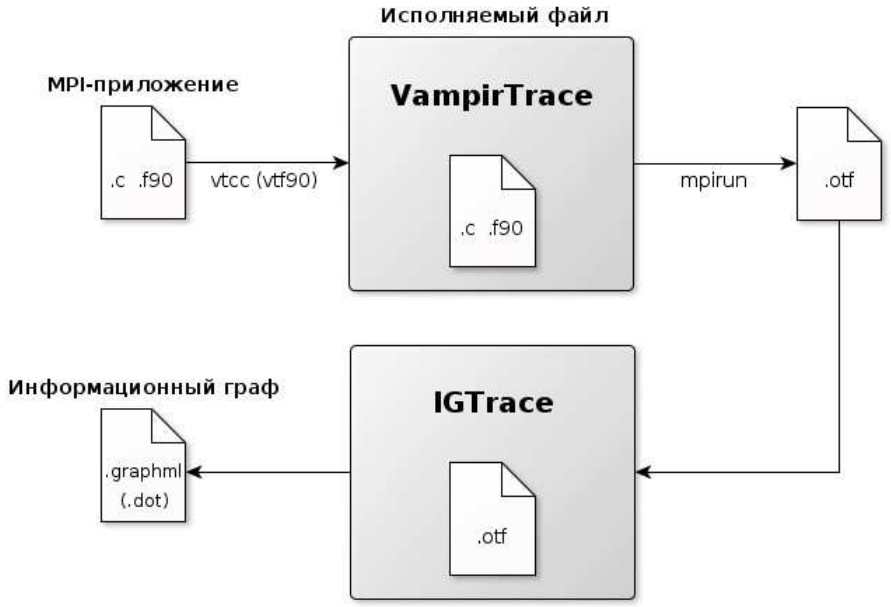

Рис. 1. Этапы построения информационного графа MPI-приложения

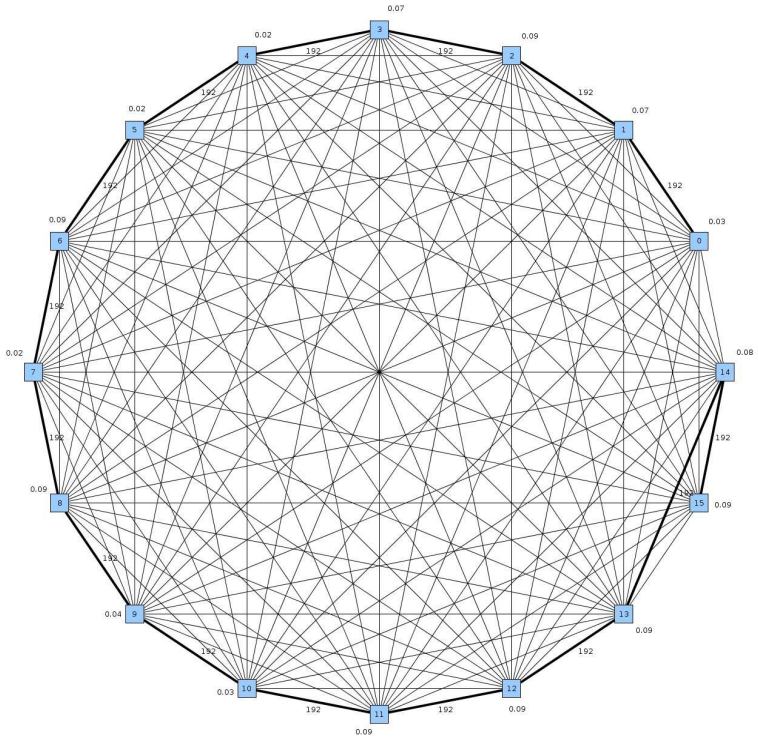

Рис. 2. Информационный граф MPI-приложения для решения уравнения Лапласа методом Якоби

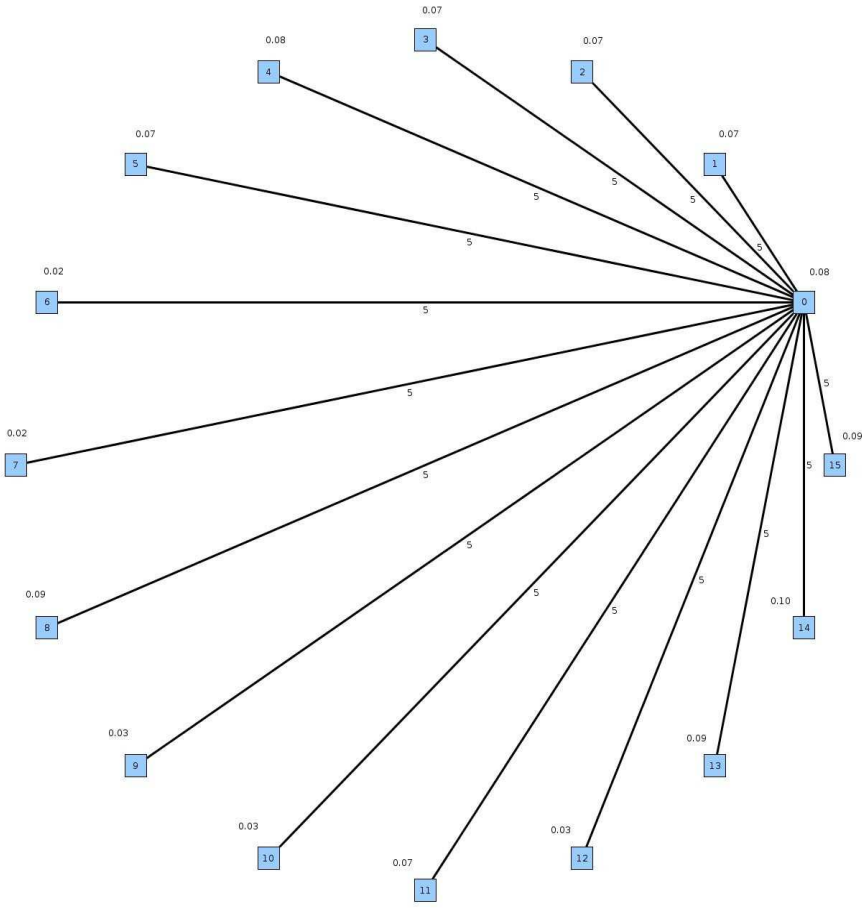

Рис. 3. Информационный граф MPI-приложения для вычисления числа Пи

4. Заключение

Результаты проведенного исследования были апробированы на суперкомпьютере МВС-100К. Размер файла, содержащего информационный граф из 500 вершин и 250000 ребер в формате graphML составил 17,7 Мбайт, в формате Dot – 7,4 Мбайта. Размер файла, содержащего покрывающее дерево указанного графа в формате graphML, составил 61 Кбайт, в формате Dot – 23 Кбайта. Время выполнения MPI-программы, с учетом формирования данных профилирования и трассировки, увеличилось незначительно (менее 5%). Полученные результаты позволяют сделать вывод, что рассмотренный подход может применяться для автоматизированного построения информационных графов параллельный программ.

ЛИТЕРАТУРА:

1. V. Herrarte, E. Lusk. Studying Parallel Program Behavior with Upshot. Technical Report ANL–91/15, Argonne National Laboratory, Argonne, IL 60439, 1991.
2. Omer Zaki, Ewing Lusk, William Gropp, Deborah Swider. Toward Scalable Performance Visualization with Jumpshot. High-Performance Computing Applications, vol. 13, No. 2, 1999, pp 277-288.
3. Sameer S. Shende, Allen D. Malony. The TAU Parallel Performance System. The International Journal of High Performance Computing Applications, Vol. 20 No. 2, summer 2006, pp. 287-311.
4. Robert Henschel. Introducing OTF / Vampir / VampirTrac. Technical Report. Center for Information Services and High Performance Computing (ZIH). 2007.
5. М.Г. Курносов. Модели и алгоритмы вложения параллельных программ в распределенные вычислительные системы. Дис. на соиск. учен. степ. канд. техн. наук. Н., 2008.
6. Anthony Chan, William Gropp, Ewing Lusk. An Efficient Format for Nearly Constant-Time Access to Arbitrary Time Intervals in Large Trace Files. Scientific Programming, Vol. 16, No. 2-3, 2008, pp 155-165.
7. Ulrik Brandes, Markus Eiglsperger, Juergen Lerner, Christian Pich. Graph Markup Language (GraphML). Symposium on Graph Drawing (GD ’01), vol. 2265 of Lecture Notes in Computer Science, 2002, pp 485–500.

8. Emden Gansner, Eleftherios Koutsofios, Stephen Nort. Drawing graphs with dot. Dot User's Manual, January 26, 2006.
9. Дж. Сик, Л. Ли, Э. Ламсдэйн. C++ Boost Graph Library. Библиотека программиста / Пер. с английского Р. Сузи – Спб.: Питер, 2006.
10. Gabor Csardi, Tamas Nepusz. Igraph Reference Manual. 2006.
11. W.N. Venables, D.M. Smith. An Introduction to R. Notes on R: A programming Environment for Data Analysis and Graphics. 2001.
12. Emden R. Gansner. Drawing graphs with Graphviz. Graphviz Drawing Library Manual. 2011.
13. Patrick J. McSweeney. Gephi Network Statistics. Google Summer of Code 2009 Project Proposal. 2009.
14. Jurgen Lerner. Analysis and visualization with visone. Egoredes Summerschool, Barcelona, 21-25 June, 2010.
15. Ekaterina Gonina, Laxmikant V. Kale. Parallel Prim's algorithm on dense graphs with a novel extension . Department of Computer Science . University of Illinois at Urbana-Champaign . November 16, 2007 .