Метод итераций для графов задач

П. А. Васёв[1]

[1]Институт математики и механики им. Н.Н. Красовского УрО РАН, г. Екатеринбург

Для программирования параллельных программ определённой популярностью пользуется модель графа задач (task graph) [1,2]. Узлы графа – задачи, где под задачей подразумевается неблокирующийся алгоритм для некоторых устройств, например процессоров или GPU. Программист определяет параллельную программу через граф задач, формируя его посредством добавления задач и связей между ними (по входам и выходам). Задачи выполняются параллельно, в процессах исполнителей, работающих на устройствах.

Как задачи оказываются на исполнителях? При прямой реализации задачи передаются от управляющей программы к исполнителям по сети. Это влечёт накладные расходы. В случае мелкозернистых задач они оказываются значительны, и производительность падает.

Эти расходы можно уменьшить, если не передавать информацию о каждой задаче по сети. Важное наблюдение заключается в том, что если задачи будут порождаться локально по отношению к исполнителям этих задач, и не передаваться по сети, то тогда эти расходы будут нивелированы. Но как добиться этого? Известны разные методы, например [3,4], но они привносят накладные расходы другого рода – неудобство программирования.

Предлагается новый метод, который по мнению автора удобен в использовании, и при этом приводит к локальной постановке задач исполнителям. Метод работает для вычислений, у которых граф задач имеет регулярную структуру, состоящую из одинаковых итераций.

1. Программист определяет задачи одной итерации, формируя некоторый подграф задач *M*. При этом он маркирует начальные и оконечные задачи.

2. Программист запускает повторяющееся выполнение задач подграфа *M (*аналогично операции *repeat* модели *C++ P2300*), с остановом по некоторому критерию, например по количеству итераций. При запуске указываются данные для начальных задач из *M*, а при повторах выходы оконечных задач замыкается на входы этих начальных задач.

3. Предполагается, что каждая задача из *M* соответствует некоторому исполнителю. Задать это соответствие можно на этапе добавления задачи или с помощью вспомогательных алгоритмов. Таким образом *M* распадается на части, соответствующие исполнителям.

4. Операция запуска вычислений (п. 2) однократно пересылает части *M* по исполнителям. При последующих итерациях для выполнения задач такой передачи уже не требуется.

Пример. Пусть вычисляется функция одного переменного на регулярной сетке по явной схеме. Разобьём эту сетку на *P* блоков. Для каждого блока создадим задачу расчёта функции на нём. На вход эта задача принимает три аргумента с данными из предыдущей итерации — свой блок, и граничные значения соседних блоков. На выходе задача выдаёт новое значение блока и граничные значения. Итерация состоит из *P* таких задач, это и есть подграф *M*. Исходные коды примера доступны по адресу github.com/pavelvasev/ppk/tree/pavt2024/experiments/iterations.

Предложенный метод напоминает схему вычислительных блоков программных сред типа *Simulink*, замкнутую в цикл. По сути, метод показывает связь двух моделей программирования, графа задач и потока данных (dataflow), и сводит первую ко второй.

Литература

1. Воеводин В.В., Воеводин Вл.В. Параллельные вычисления. - СПб.: БХВ-Петербург, 2002. - 608 с. Прим: см. *граф алгоритма*, стр. 191.
2. Прихожий, А.А. Распределенная и параллельная обработка данных // Белорусский национальный технический университет. - Минск: БНТУ, 2016. - 91 с.
3. Reazul Hoque, Thomas Herault, George Bosilca, and Jack Dongarra. 2017. Dynamic task discovery in PaRSEC: a data-flow task-based runtime. In Proceedings of the 8th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (ScalA '17). Association for Computing Machinery, New York, Article 6, 1–8. DOI: 10.1145/3148226.3148233
4. Charly Castes, Emmanuel Agullo, Olivier Aumage, Emmanuelle Saillard. Decentralized in-order execution of a sequential task-based code for shared-memory architectures. [Research Report] RR-9450, Inria Bordeaux - Sud Ouest. 2022, pp.30. Hal-03547334.