Гибридное планирование графов задач

Для программирования параллельных программ определённой популярностью пользуется модель графа задач (task graph) [1-4]. В настоящее время в англоязычной литературе выделяют системы, работающие по этой модели, термином asynchronous many-task systems [5,6]. Также их называют task-based runtime systems, а модель программирования sequental task flow [11,12].

Например, к таким системам относятся технологии OpenTS, Template Task Graphs, PaRSEC, HPX, Charm++, Uintah, Kokkos, Legion, C++ Sender Library, LuNA, Dask, Flyte, TaskTorrent, и другие. Они различаются между собой сферой применения, наличием своего языка программирования, и другими нюансами.

Автор разрабатывает аналогичную технологию параллельного программирования PPK [7]. Её отличительная особенность – ориентация не только на вычисления в целом, но и на задачи визуализации, в том числе в режиме онлайн (insitu visualization [8]).

В целом все подобные технологии строятся на идее, что программист описывает параллельную программу путем добавления задач в вычислительную среду. Под задачей подразумевается некоторый неблокирующийся алгоритм для некоторого исполнительного устройства, например процессора или GPU.

Среди аргументов добавляемой задачи можно указывать не просто обычные данные, но и ссылки на результаты других задач, и таким образом описывать зависимости между задачами. Последнее означает, что задача T может быть выполнена средой после того, как выполнятся задачи, формирующие аргументы для T. Используя эту идею, возможно описание параллельного вычисления, причем формировать это описание возможно последовательным алгоритмом. Применяются и более сложные модели описания зависимостей с добавлением описания блоков данных и режима доступа к ним (read-write-readwrite), см. например [11,12].

Выполнение задач подразумевается на некотором множестве исполнителей. В их роли могут рассматриваться ядра процессора, узлы вычислителя, устройства типа GPU, и д.р. Возникает вопрос, на каком конкретно исполнителе и когда следует выполнять ту или иную задачу?

Разные технологии по-разному подходят к этому вопросу. Некоторые берут решение проблемы балансировки нагрузки (построения расписания, планирования) полностью на себя. Другие наоборот, полностью отказываются и предоставляют пользователю управлять балансировкой. В целом, различают статические схемы и динамические схемы планирования (static scheduling / dynamic balancing).

Например, среда HPX требует явного указания исполнителя (locality, component). При этом HPX также предлагает возможность считывания метрик по загруженности исполнителей, и считается что пользователь может автоматизировать выбор на их основе.

Контекст этой ситуации следующий. С одной стороны, нагружать надо как можно большее число исполнителей, обеспечивая этим параллелизм вычисления. С другой стороны, если задача зависит от результатов работы других задач, то для решения задачи необходимо эти результаты предоставить. Если задача назначается на один узел, а аргументы находятся на другом узле, то их необходимо будет передать. Это в свою очередь влечет задержку, и она является ключевой для эффективности параллельных вычислений [9]. Частично эта проблема

решается т.н. маскировкой, когда передача данных для некоторой задачи производится параллельно с вычислением других задач. Но добиться такой маскировки получается не во всех реальных ситуациях.

С одной стороны задача балансировки сложная [10]. Поэтому разумно предложить пользователю некоторое её решение. С другой стороны, пользователь лучше знает свою задачу. Поэтому он сможет организовать более эффективную балансировку нагрузки, чем автоматические алгоритмы.

В докладе предлагается гибридный подход. Он подразумевает, что при добавлении задачи в среду пользователь имеет возможность:

1. Явно указать исполнителя, на котором следует запускать эту задачу.
2. Позволить среде автоматически выбрать исполнителя из некоторого множества исполнителей.

Последнее подразумевает, что пользователь может создать не одно, а несколько множеств исполнителей. И таким образом создавать “домены” исполнения задач. Внутри каждого домена нагрузка между исполнителями распределяется системой автоматически.

Вычисление задач обеспечивается набором исполнителей, возможно динамическим и возможно разнородным. Часть исполнителей пользователь может использовать, назначая задачи на них самостоятельно. Другую часть исполнителей пользователь может разбить на подмножества, и назначать на них разные группы задач.

Таким образом, с одной стороны, становится возможным учесть знание пользователя об информационной структуре параллельного вычисления. С другой стороны, предложить разные варианты использования этого знания.

Литература

1. Воеводин В.В., Воеводин Вл.В. Параллельные вычисления. - СПб.: БХВ-Петербург, 2002. - 608 с. Прим: *граф алгоритма*, стр. 191.
2. Гергель В. П. Основы параллельных вычислений для многопроцессорных вычислительных систем : Учебное пособие / В. П. Гергель, Р. Г. Стронгин; Нижегородский гос. ун-т им. Н. И. Лобачевского. - Нижний Новгород : Издательство Нижегородского госуниверситета, 2001. - 122. Прим.: граф *операции-операнды*, раздел 2.1.
3. Прихожий, А.А. Распределенная и параллельная обработка данных // Белорусский национальный технический университет, Кафедра "Программное обеспечение вычислительной техники и автоматизированных систем". - Минск: БНТУ, 2016. - 91 с. Прим.: *граф задач*, стр. 44.
4. Sally Stevenson, Stephen Jones and Fred Oh, Enabling Dynamic Control Flow in CUDA Graphs with Device Graph Launch // 12 dec 2022, Nvidia developer blog.
5. Труды конференции. Asynchronous Many-Task Systems and Applications. First International Workshop, WAMTA 2023, Baton Rouge, LA, USA, February 15–17, 2023, Proceedings.
6. Труды конференции. Asynchronous Many-Task systems for Exascale 2023, AMTE 2023, August 28-29, 2023. Held in conjunction with Euro-Par 2023 Limassol, Cyprus.
7. Васёв П.А. Модель системы онлайн-визуализации // Параллельные вычислительные технологии (ПаВТ'2023): Короткие статьи и описания плакатов, г. Санкт-Петербург, 28–30 марта 2023 года. – Челябинск: Издательский центр ЮУрГУ, 2023. – С. 236. URL: https://www.cv.imm.uran.ru/e/3241872

8. Pavel Vasev, Analyzing an Ideas Used in Modern HPC Computation Steering // 2020 Ural Symposium on Biomedical Engineering, Radioelectronics and Information Technology (USBEREIT), Yekaterinburg, Russia, 2020, pp. 1-4. DOI: 10.1109/usbereit48449.2020.9117685

9. Лацис А. А. Как построить и использовать суперкомпьютер // Бестселлер, 2003. 238 с. Прим.: латентность коммуникаций, глава 5 стр. 88.

10. Sukhoroslov, O. Toward efficient execution of data-intensive workflows. J Supercomput 77, 7989–8012 (2021). DOI: 10.1007/s11227-020-03612-4

11. 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.

12. 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, NY, USA, Article 6, 1–8. https://doi.org/10.1145/3148226.3148233