ОБ АКТУАЛЬНОСТИ ИССЛЕДОВАНИЯ И РАЗВИТИЯ СОВРЕМЕННЫХ МОДЕЛЕЙ РАСПРЕДЕЛЕННОГО И ПАРАЛЛЕЛЬНОГО ПРОГРАММИРОВАНИЯ

И.Г. Данилов

Южный федеральный университет, Ростов-на-Дону

Рассматриваются ряд проблем современных моделей распределённого и параллельного программирования и возможные направления путей их решения.

Введение

В настоящее время развитие суперкомпьютерной техники идёт в направлении увеличения мощности отдельных вычислительных узлов за счёт внедрения многоядерных процессорных технологий, наращивания количества вычислительных узлов в системе и совершенствования характеристик межузловых сетевых соединений. В связи с этим широко применяемые модели параллельного и распределённого программирования, которые используют *двусторонние коммуникации*, т.е. *обмен сообщениями*, могут оказаться неэффективными при программировании задач для систем, состоящих из десятков и сотен тысяч слабосвязанных ядер. Это объясняется тем, что:

1) для двустороннего обмена требуется участие обеих сторон: как отсылающей, так и принимающей, что приводит к усложнению программирования и возможным потерям производительности;

2) количество памяти, доступной вычислительному процессу, ограничено размером локальной памяти вычислительного узла, на котором выполняется этот процесс;

3) для моделей, использующих обмен сообщениями, характерно *фрагментированное представление о вычислениях* [1], что приводит к сложности распределения составных и комплексных структур данных по вычислительным процессам, необходимости написания кода по проверке граничных условий и возрастанию количества обменов сообщениями между вычислительными процессами;

4) при организации двустороннего обмена возникают накладные расходы, связанные с формированием, упорядочением и буферизацией сообщений, что оказывается узким местом при возрастании количества обменов сообщениями.

Другая популярная модель параллельного программирования, которая основана на взаимодействии через *общую разделяемую память*, предлагает более простой и продуктивный по сравнению с моделью передачи сообщений подход [2] к созданию параллельных приложений, так как программисту нет необходимости заботиться о локальности доступа к данным (*глобальное представление о вычислениях* [1]). Однако эта модель в свою очередь оказывается неэффективной при решении масштабируемых задач в силу сложности эффективной реализации (как аппаратной, так и программной, виртуальной) разделяемой памяти для больших вычислительных систем.

Модель разделённого глобального адресного пространства

Развитие и широкое распространение новых вычислительных архитектур и коммуникационных сетей в последние годы привело к активизации исследований в области совершенствования существующих моделей и алгоритмов параллельного и распреде-

лённого программирования с учетом новых технических возможностей. Одной из таких недавно появившихся моделей программирования является модель разделённого глобального адресного пространства (англ. *Partitioned Global Address Space*, PGAS), в которой используется принцип взаимодействия на основе односторонних коммуникаций, позволяющих одной части системы взаимодействовать с другой без активного её привлечения. Односторонние коммуникации эффективно реализуются в современных сетях для суперкомпьютеров с использованием протокола удалённого доступа в память (англ. *Remote Direct Memory Access*, RDMA).

Появление парадигмы PGAS – попытка разработать гибридную модель, объединяющую преимущества передачи сообщений и взаимодействия через разделяемую память. В реализациях PGAS, таких как расширение для языка программирования C – UPC [3] (англ. *Unified Parallel C*) или Fortran – Coarray Fortran (CAF), которые можно назвать *языками PGAS первого поколения*, распределение данных в вычислительной системе производится явно программистом с учётом локальности последующего доступа к ним. При этом инструкции доступа к данным генерируются компилятором неявно; программисту необходимо позаботиться только о синхронизации с целью получения непротиворечивых результатов. Данные языки используют низкоуровневые сетевые коммуникационные библиотеки PGAS, такие как GASNet [4] и ARMCI, которые предоставляют различные функции для односторонних коммуникаций, передачи сообщений, операций синхронизации и управления разделённым глобальным адресным пространством.

Основные проблемы современных моделей распределённого и параллельного программирования и возможные пути их решения

Широко распространённые реализации модели передачи сообщений (прежде всего MPI, от англ. *Message Passing Interface*) и модели PGAS (например, UPC) сравнимы по производительности [5]. Каждая реализация имеет свои отличительные достоинства и недостатки, но в то же время существует один общий для всех реализаций недостаток: плохая поддержка *неструктурированного параллелизма* – общей и наименее упорядоченной формы параллелизма [6]. Неструктурированный параллелизм предполагает возможность динамического создания (*динамический параллелизм*) вычислительных процессов, работающих по различным алгоритмам. В то же время как модель передачи сообщений, так и языки PGAS первого поколения реализуются на основе SPMD-подхода, в котором программа состоит из фиксированного набора вычислительных процессов, количество которых в течение её выполнения не изменяется (*статический параллелизм*), что приводит к плохой балансировке нагрузки вычислительной системы.

Потребность выполнения различных алгоритмов в программе в зависимости от некоторых условий приводит к необходимости использовать более общий подход по сравнению с SPMD, который называется MPMD и реализуется, например, в модели общей памяти на основе многопоточности, которая предусматривает специальные примитивы для динамического управления программными потоками. В свою очередь, в рассматриваемых моделях MPMD-программирование осуществляется с помощью определенных параметров вычислительных процессов системы (ранг, коммуникатор – для MPI, номер потока – для UPC) и соответствующего ветвления, что приводит к усложнению и плохой читаемости кода программы. Кроме того, остается нерешенным вопрос возможной разбалансировки нагрузки вычислительной системы. Для частичного решения данных проблем, которые возникают из-за отсутствия поддержки динамического параллелизма, применяют следующие методы.

1. *Гибридизацию* моделей программирования: передачу сообщений совмещают с многопоточными реализациями модели общей памяти (например, с OpenMP [7]) или с

моделью параллелизма задач (англ. *task parallelism*). Аналогично и для модели PGAS: например, для UPC существуют специальные расширения [8], использующие модель параллелизма задач. Стоит отметить, что при данном подходе вопрос разбалансировки нагрузки решается только в рамках одного вычислительного узла системы.

2. *Виртуализацию* процессов – подход, при котором вычислительные процессы модели программирования отображаются определённым образом на программные процессы или потоки операционной системы и могут *мигрировать* на любые доступные в системе вычислительные узлы. Для реализаций модели передачи сообщений данный подход поддерживается как на уровне операционной системы (MOSIX [9]), так и на уровне программных библиотек; можно отметить, что к модели PGAS виртуализация процессов трудно применима, т.к. для миграции процессов требуется сложная поддержка механизмов управления и доступа к глобальной разделяемой памяти со стороны компилятора и исполняющей среды.
3. *Расширение* спецификации, описывающей стандарт реализации модели программирования, с целью внесения новой функциональности для динамического порождения процессов/потоков. Например, такая функциональность появилась во второй версии спецификации для стандарта MPI, однако её использование существенно усложняет программирование. Спецификации для реализаций языков PGAS первого поколения не поддерживают конструкции динамического создания потоков.

Проблема отсутствия поддержки динамического параллелизма в языках PGAS первого поколения, наряду с рядом других проблем [1], привела к возникновению новых языков PGAS – *языков второго поколения*, разрабатываемых по инициативе американского агентства передовых оборонных исследовательских проектов (англ. *Defense Advanced Research Projects Agency*) в рамках программы по созданию высокопродуктивных компьютерных систем HPCS (англ. *High Productivity Computing Systems*). Наиболее известными среди них являются: Chapel [1] и X10. В новых языках HPCS параллелизм задач – для X10 или принцип многопоточности – для Chapel лежат в основе реализации. Кроме того, с помощью различных языковых конструкций в них поддерживаются и другие виды параллелизма, например *параллелизм по данным*.

В моделях параллельного и распределённого программирования, которые построены на понятии общей разделяемой памяти, память используется для взаимодействия вычислительных процессов друг с другом, что может привести к проблеме *гонки по данным*. Эта проблема особенно актуальна для неструктурированного параллелизма, когда доступ к общим данным в общем случае непредсказуем. Для избежания гонок требуется согласовать доступ вычислительных процессов к данным с помощью явной синхронизации, средства для которой предоставляются реализациями моделей программирования. Традиционно эти средства построены на различных *блокирующих методах* (например, семафорах) и позволяют решить *задачу о взаимном исключении процессов* (англ. *mutual exclusion*), в которой доступ к разделяемому ресурсу (памяти) в определённый промежуток времени ограничен строго одним процессом или же процессом определённого типа (например, процессом-читателем). В свою очередь, использование блокирующих методов синхронизации порождает ряд других проблем, таких как *взаимоблокировка*, что требует определённой квалификации и трудозатрат со стороны программиста. Для распределённых систем существует большое количество алгоритмов [10], которые решают задачу *распределённого взаимного исключения* (англ. *distributed mutual exclusion*) и используются для реализации примитивов распределённой синхронизации, предоставляемых в том числе и языками модели PGAS. Существующие алгоритмы распределённого взаимного исключения построены на основе передачи сообщений по сети и блокирующих принципов, ключевая характеристика кото-

рых – возможная задержка для входа в *критическую область* – возрастает в зависимости от числа процессов в системе и частоты обращения к разделяемым данным, что приводит к плохой масштабируемости параллельной программы в целом.

Необходимость решения проблем блокирующих методов синхронизации привела к возникновению исследований в области *неблокирующей синхронизации* (англ. *non-blocking synchronisation*). Предложены различные методы неблокирующей синхронизации и соответствующие программные/аппаратные абстракции [11], одной из которых является активно исследуемая в настоящее время *транзакционная память* (англ. *transactional memory*, TM). Транзакционная память базируется на методологии теории транзакционной обработки данных, разработанной в 70–80-х годах прошлого столетия такими исследователями, как K. P. Eswaran, C. H. Papadimitriou, P. A. Bernstein, H. T. Kung и др. Методы транзакционной обработки, применяемые в базах данных для поддержания данных в согласованном состоянии, были обобщены M. Herlihy и J. E. B. Moss, которые выдвинули идею [12] использования подобных методов для согласования доступа нескольких процессоров к разделяемой памяти. Предложенные Herlihy и Moss принципы аппаратной реализации TM (англ. *hardware TM*, HTM) были развиты N. Shavit и D. Touitou в работе [13], впервые описывающей принципы программной реализации TM (англ. *software TM*, STM).

Концепция транзакций тесно связана с задачей взаимного исключения. Как и блокирующие алгоритмы синхронизации, алгоритмы транзакционной памяти позволяют последовательно упорядочить – *сериализовать* – инструкции доступа к памяти параллельно выполняемых вычислительных процессов, тем самым обеспечив непротиворечивость результатов их работы. Основное отличие состоит в том, что транзакции – помеченные как атомарные (англ. *atomic section*) участки кода программы, в которых происходит обращение к общему ресурсу (памяти), – выполняются всеми процессами одновременно, в то время как в критической области в общем случае может находиться только один процесс. Если во время выполнения атомарных секций происходит конфликт по данным между двумя транзакциями, то одна из них откатывается на начало выполнения секции, а другая завершается и выходит из секции.

Большая часть зарубежных работ в данной области посвящена системам с общей памятью [14]; более того, ведущие производители процессорных устройств уже заявили поддержку механизмов транзакционной памяти в ряде моделей процессоров будущих поколений [15]. Для систем с распределённой памятью существует всего несколько рабочих программных реализаций [16]. В настоящее время разработчики языков HPCS исследуют возможности применения механизмов транзакционной памяти для программирования распределённых систем: например, в языке Chapel заложена специальная конструкция атомарной секции *atomic*, в основе которой модель глобальной транзакционной памяти [17].

Заключение

Можно отметить следующие актуальные направления исследований в области моделей распределённого и параллельного программирования:

1) отказ от модели взаимодействия на основе обмена сообщениями и переход к односторонним коммуникациям;
2) исследование и развитие неблокирующих методов синхронизации, таких как транзакционная память;
3) исследование и развитие моделей параллелизма задач и данных.

Литература

1. Chamberlain B., Callahan D., Zima H. Parallel Programmability and the Chapel Language // Int. J. High Perform. Comput. Appl. 2007. V. 21, no. 3. P. 291–312.

2. Hochstein L., Carver J., Shull F. et al. Parallel Programmer Productivity: A Case Study of Novice Parallel Programmers // Proceedings of the 2005 ACM/IEEE conference on Supercomputing. SC ’05. Washington, DC, USA: IEEE Computer Society, 2005. P. 35.
3. El-Ghazawi T., Carlson W., Sterling T., Yelick K. UPC: Distributed Shared Memory Programming. Wiley Series on Parallel and Distributed Computing. John Wiley & Sons, 2005.
4. Bonachea D. GASNet Specification, v1.1: Tech. rep. Berkeley, CA, USA: 2002. URL: http://techreports.lib.berkeley.edu/accessPages/CSD-02-1207 (дата обращения: 26.09.2012).
5. Mallon D. A., Taboada G. L., Teijeiro C. et al. Performance Evaluation of MPI, UPC and OpenMP on Multicore Architectures // Proceedings of the 16th European PVM/MPI Users’ Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface. Berlin, Heidelberg: Springer-Verlag, 2009. P. 174–184.
6. Sutter H., Larus J. Software and the Concurrency Revolution // Queue. 2005. – September. Vol. 3, no. 7. P. 54–62.
7. Smith L., Bull M. Development of mixed mode MPI / OpenMP applications // Sci. Program. 2001. – August. Vol. 9, no. 2,3. P. 83–98.
8. Dinan J., Krishnamoorthy S., Larkins D. B. et al. Scioto: A Framework for Global-View Task Parallelism // Proc. of the 2008 37th International Conf. on Parallel Processing. ICPP ’08. Washington, DC, USA: IEEE Computer Society, 2008. P. 586–593.
9. Barak A., La’adan O. The MOSIX multicomputer operating system for high performance cluster computing // Future Gener. Comput. Syst. 1998. – March. Vol. 13, no. 4-5. P. 361–372.
10. Attiya H., Welch J. Distributed Computing: Fundamentals, Simulations and Advanced Topics (2nd edition). John Wiley Interscience, 2004. – March. ISBN: 0-471-453242.
11. Fraser K. Practical lock-freedom: Tech. Rep. UCAM-CL-TR-579: University of Cambridge, Computer Laboratory, 2004. – February. – [http://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-579.pdf].
12. Herlihy M., Moss J. E. B. Transactional memory: architectural support for lock-free data structures // SIGARCH Comput. Archit. News. 1993. – May. Vol. 21,no. 2. P. 289–300.
13. Shavit N., Touitou D. Software transactional memory // Proceedings of the Fourteenth Annual ACM Symposium on Principles of Distributed Computing. PODC ’95. New York, NY, USA: ACM, 1995. P. 204–213.
14. Grahn H. Transactional Memory // In: J. Parallel Distrib. Comput. - 2010. - Vol. 70 (10). - P. 993-1008.
15. Merritt R. IBM plants transactional memory in CPU // Intel Software Network – [http://www.eetimes.com/electronics-news/4218914/IBM-plants-transactional-memory-in-CPU].
16. Saad M. M., Ravindran B. Transactional Forwarding Algorithm: Technical Report // ECE Dept., Virginia Tech, January 2011.
17. Sridharan S., Vetter J. S., Kogge P. M. Scalable Software Transactional Memory for Global Address Space Architectures: Tech. Rep. FTGTR-2009-04: Future Technologies Group, Oak Ridge National Lab, 2009. – November.