Параллельные вычисления

Неэффективное поведение в программах, написанных в рамках парадигмы PGAS

Ю. В. Виноградова, А. В. Соловьев

Аннотация. Работа посвящена проблематике повышения производительности параллельных программ, написанных на основе парадигмы PGAS (Partitioned Global Address Space). Подход, предлагаемый в статье, основан на механизме анализа трасс с помощью шаблонов неэффективного поведения. Шаблоны определяют вид участка трассы, характерный для выполнения кода, в котором происходят потери производительности. Сопоставление шаблона и трассы позволяет выявить источники снижения производительности в программе. В статье описан предлагаемый подход, даны основные шаблоны неэффективного поведения и приведены возможные пути применения разработанного подхода на практике.
Ключевые слова: *параллельные вычисления, язык программирования upc, парадигма pgas, автоматическая отладка параллельных программ, шаблоны неэффективного поведения.*

Введение

Современные высокопроизводительные кластеры преодолели рубеж в петафлопс, а количество ядер, которое насчитывают такие системы, десятков и сотен тысяч. Но если взглянуть на эффективность параллельных приложений на подобных системах, то оказывается, что по разным оценкам она не превышает 15 % [1] или даже 3–5 % [2] от пиковой производительности.

Чтобы эффективно использовать потенциал дорогостоящих вычислительных установок, сообществу специалистов в области высокопроизводительных вычислений необходимы инструменты анализа производительности. С одной стороны, они помогают увеличить эффективность и масштабируемость параллельных программ, а с другой позволяют ученому сконцентрироваться на научной составляющей своего приложения, нежели на кропотливом процессе оптимизации.

Сегодня суперкомпьютерная индустрия ставит перед собой рубеж в экзафлопс. Разработка комплексов параллельных программ для таких систем является невероятно сложной задачей. Суперкомпьютеры могут иметь разную архитектуру и коммуникационную сеть [3], существует ряд алгоритмов распараллеливания, а также технологий и средств для написания параллельных программ [4, 5].

Стандартом де-факто среди таких средств на сегодня является MPI (Message Passing Interface) [6]. Система программирования MPI предназначена для разработки программ для многопроцессорных системам с распределенной памятью. В основе системы лежит парадигма передачи сообщений.

Несмотря на сравнительно высокую производительность MPI-программ, процесс разработки приложений с использованием данной библиотеки трудоемок и подвержен ошибкам. Ограничения программной модели передачи сообщений широко признаны, а саму библиотеку MPI иногда называют «ассемблером параллельного программирования».

Альтернативой MPI является набирающая популярность модель программирования PGAS — Partitioned Global Address Space (разделенное глобальное адресное пространство) [7]. В модели PGAS используются односторонние коммуникации, где при передаче данных нет необходимости в явном отображении на двухсторонние пары send и receive — трудоемкий, подверженный ошибкам процесс, се-

рьезно влияющий на продуктивность программиста. Обычное присвоение значения переменной массива автоматически порождает необходимые коммуникации между узлами кластера. Программы, написанные в этой модели, проще для понимания, чем их MPI версии, и имеют сопоставимую или даже более высокую производительность [8].

К группе PGAS относятся такие языки, как: Unified Parallel C (UPC) [9], Co-Array Fortran (CAF) [10], Titanium [11], Cray Chapel [12], IBM X10 [13], Sun Fortress [14]. Язык UPC является наиболее известным представителем модели. Компиляторы для языка UPC разработаны всеми основными вендорами суперкомпьютерного рынка.

1. Общие сведения о модели PGAS

Программная модель PGAS расшифровывается как Partitioned Global Address Space (разделенное глобальное адресное пространство) и позволяет добиться баланса между простотой использования присущей модели общей памяти и возможностью управлением локальностью данных модели передачи сообщений. В данной модели независимые нити работают с общим адресным пространством, так же как и в модели с общей памятью. Но общее пространство памяти логически разделено между нитями. Это позволяет распределить между узлами нити вместе с локальными для них данными. Таким образом, программист может указать, что нить будет обрабатывать данные локальные по отношению к ней. Это позволяет минимизировать или избавиться от ненужных удаленных обращений к памяти с самого начала. Модель с распределенной общей памятью позволяет добиться хорошего баланса между программной абстракцией и переносимости с одной стороны, и непосредственным контролем над распределением ресурсов для хорошей производительности с другой. Все это позволяет добиться эффективного выполнения параллельных программ на современных компьютерных архитектурах.

Это языки Unified Parallel C (UPC), Co-Array Fortran (CAF) и Titanium, которые являются расширениями языков C, Fortran и Java соответственно. В рамках программы High Productivity Computing Systems разрабатываются языки Chapel, Fortress и X10, которые также берут модель разделенного глобального пространства за основу.

2. Язык программирования UPC

UPC (Unified Parallel C) является расширением языка C для параллельных вычислений. Первый стандарт языка был выпущен в феврале 2001 года. В написании стандарта участвуют академические институты, производители и правительственные лаборатории.

В UPC нити работают независимо, неявная синхронизация между ними отсутствует, за исключением того, что все нити запускаются и завершаются одновременно. Для этого в начале и в конце программы используется неявная барьерная синхронизация. Общее количество нитей задается переменной THREADS, а идентификация каждой конкретной нити осуществляется при помощи переменной MYTHREAD. Общее количество нитей — THREADS может быть указано как на этапе компиляции, так и при запуске программы из командной строки.

Программы, написанные на UPC, работают в программной модели SP-MD, где каждая нить выполняет одну и ту же функцию. Это не ограничивает программную модель в гибкости, так как управление потоком внутри нити па основе операторов условного перехода позволяет направить разные экземпляры нити на выполнение различных частей кода при помощи переменной MYTHREAD и промежуточных результатов решаемой задачи.

UPC предлагает один из вариантов парадигмы с распределенной общей памятью: память состоит из логически разделенного общего пространства памяти и локально адресуемой памяти. Все распределенные по вычислительным узлам нити могут ссылаться на адрес любой ячейки памяти (общие переменные) в общем пространстве данных, но в частное адресное пространство может ссылаться только та нить, которой оно принадлежит. Общее пространство логически разбито на части, каждая из которых связана с определенной нитью. Таким образом, UPC позволяет программисту при помощи определенных объявлений держать общие данные, которые преимущественно будут обрабатываться конкретной нитью (и к которым периодически будут обращаться другие нити), рядом с ней. Это позволяет отобразить и нить, и данные, связанные с ней, на один и тот же вычислительный узел. Программистам даны необходимые языковые конструкции для выражения присущей их программам локальности данных. UPC не ограничивает реализации языка в отображении только одной нити на один процессор, если аппаратное обеспечение это позволяет, можно воспользоваться поддержкой многопоточности. Дальнейшее изложение будет вестись для языка UPC, хотя предлагаемая концепция носит достаточно общий характер и применима и к другим языкам на основе PGAS.

Язык UPC содержит следующие операции, требующие взаимодействия двух и более нитей: upc_put(), upc_get(), upc_barrier(), upc_notify(), upc_wait(), upc_memget(), upc_memput(), upc_memcpy(), upc_fence(), upc_lock(), upc_unlock(), а также все коллективные коммуникационные операции. Односторонние коммуникационные операции upc_put()\upc_get(), лежащие в основе языка UPC и модели PGAS в целом, не блокируют выполнение программы, как это часто бывает в приложениях, написанных на MPI.

Одной из типичных причин задержек выполнения программы является барьерная синхронизация, присутствующая практически во всех языках параллельного программирования. В языке UPC барьерная синхронизация представлена операциями upc_barrier() и upc_notify()\upc_wait(). В первом случае программа блокируется до тех пор, пока все нити не достигнут операции upc__barrier(), во втором случае, который называется барьером с расщепленной фазой, синхронизация разбивается на два этапа. Как только нить заканчивает вычисления, требующие синхронизации, она выполняет upc_notify(), чтобы проинформировать другие нити о своем состоянии. Дальше она может продолжить выполнять локальные вычисления, и когда они также будут готовы, нить выполняет upc_wait(), чтобы остановиться и дождаться, пока остальные нити выполнят upc_notify(). Когда все остальные нити выполнили upc_notify(), нить, ожидающая в операции upc_wait(), освобождается.

Операции upc_memget(), upc_memput() и upc_raemcpy() предназначены для передачи непрерывных массивов данных, в отличие от upc_put()\upc_get(), которые выполняют пересылку переменных. Недостаток этих операций, с точки зрения производительности, в том, что они блокирующие. Нить останавливает свое выполнение до тех пор, пока передача данных не завершится. Однако эти операции односторонние и не требуют явного участия второй нити, а значит отсутствуют и какие-либо связи между событиями. Поэтому время, потерянное в этих операциях, можно найти, используя обычный механизм профилировки.

Основным источником задержек в UPC являются операции релокализации данных, среди них: upc_all_broadcast(), upc_all_scatter(), upc_all_gather(), upc_all_gather_all(), upc_all_exchange(), upc_all_permute(), upc_all_reduce() и upc_all_prefix_reduce(). В отличие от односторонних коммуникационных операций, в коллективных операциях присутствуют сложные зависимости по данным, требующие синхронизации. Если программа написана неудачно и четкая координация между нитями нарушена, то неизбежно возникнут дополнительные задержки. Чтобы свести эту проблему до минимума, в UPC для всех операций релокализации были выделены три типа синхронизации, которые указываются последним аргументом в вызове функций отдельно для синхронизации на входе в операцию и на выходе из нее:

UPC_IN_NOSYNC — коллективная функция имеет право считывать или записывать любые данные, но только после того, как в вызов коллективной операции вошла хотя бы одна нить;

UPC_IN_MYSYNC — коллективная функция имеет право считывать или записывать данные только тех нитей, которые уже вошли в операцию;

UPC_IN_ALLSYNC — коллективная функция имеет право считывать или записывать данные только после того, как все нити вошли в коллективную операцию;

UPC_OUT_NOSYNC — коллективная функция имеет право считывать и записывать данные до тех пор, пока последняя нить не вышла из операции;

UPC_OUT_MYSYNC — нить может выйти из коллективной функции только тогда, когда все операции чтения и записи с данными этой нити завершены;

UPC_OUT_ALLSYNC — прежде чем выйти из коллективной операции, нить должна дождаться завершения всех операций чтения и записи данных.

Если алгоритм программы составлен таким образом, что результат работы операции релокализации не требует координации нитей, или для синхронизации используются барьеры, то можно полностью отказаться отсинхронизации, указав флаги UPC_IN_NOSYNC и UPC_OUT_NOSYNC. Если требуется, чтобы функция могла оперировать данными только тех нитей, которые уже вошли в операцию релокализации, а нити не могли выйти из операции, пока остальные нити не завершили работу с их данными, то используются флаги UPC_IN_MYSYNC и UPC_OUT_MYSYNC. Флаги UPC_IN_ALLSYNC и UPC_OUT_ALLSYNC — обеспечивают самый надежный, но при этом и наименее эффективный способ координации.

3. Выявление неэффективного поведения в UPC-программах

Предлагаемый подход основан на поиске шаблонов неэффективного поведения. В результате практики разработки параллельных программ выявляются типичные проблемы производительности, которые могут служить основой для шаблонов неэффективного поведения.

Выполнение поиска шаблонов в UPC-программах делится на три этапа:

1) этап инструментовки программы;

2) этап сбора трассировочной информации;
3) этап анализа этой информации.

На этапе инструментовки программы в исходный текст программы внедряются вызовы библиотеки трассировки, в места, соответствующие вызову функций UPC. Например, вызов функции upc_all_broadcast() инструментируется следующим образом:

```
cglog(MYTHREAD, ”entry:upc_all_broadcast#B,
&(A[0]), sizeof(int), UPC_IN_ALLSYNC”);
upc_all_broadcast(B, &(A[0]), sizeof(int),
UPC_IN_ALLSYNC);
cglog(MYTHREAD, ”exit:upc_all_broadcast#B,
&(A[0]), sizeof(int), UPC_IN_ALLSYNC”);
```

При выполнении инструментированной программы выполняется сбор трассы выполнения и ее сохранение в файлах. Для каждого процесса (поток в терминологии UPC) параллельной программы формируется массив событий. В конце выполнения массивы объединяются и сохраняются в файле. Трасса содержит информацию о выполненных вызовах UPC, включая аргументы вызова, номер процесса, время начала и завершения операции. Например:

```
0:1374663889#383275:entry:upc_all_lock_alloc#
0:1374663889#383560:exit:upc_all_lock_alloc#
```

На третьем этапе выполняется анализ трассировочной информации. Производится поиск в трассе шаблонов неэффективного поведения. Необходим совместный анализ трассы нескольких процессов.

Шаблон — это набор найденных в трассировочном файле событий, которые удовлетворяют условиям возникновения некоторой ситуации, которую описывает шаблон. В трассе выполнения UPC-программы содержится информация о времени начала вызова функций UPC функции и ее завершения. Это позволяет вычислять временные потери, вызванные шаблоном неэффективного поведения. Перечислим основные шаблоны.

1. Шаблон «Задержка на блокировке». Данный шаблон возникает, если одна из нитей пытается захватить блокировку, которой в данный момент владеет другая нить. Сложное составное событие в данном случае состоит из трех простых событий — входа в функцию upc_lock() в нити 1, захвата блокировки в нити 1 и освобождения в нити 0. Данная ситуация классифицируется как неэффективное поведение, если вход в функцию upc_lock() происходит раньше, чем освобождение блокировки в нити 0. Временные потери в данной операции характеризуются разницей между моментами возникновения событий освобождения блокировки нитью 0 и ее захватом нитью 1.
2. Шаблон «Синхронизация на входе в коллективную операцию». Данный шаблон справедлив для всех операций релокализации, в которых используется тип синхронизации UPC_IN_ALLSYNC. В подобных случаях каждая нить обязана дождаться на входе всех остальных. Когда нити входят в операцию релокализации в разные моменты времени — это вносит нежелательные накладные расходы на синхронизацию. В составное событие, соответствующее данному шаблону, входят события входа в функцию коллективного взаимодействия всех потоков, участвующих в ней. Временные задержки характеризуются суммарным временем ожидания всех потоков на данной функции, т. е. разницей между входом в функцию коллективного взаимодействия последнего и первого потоков.
3. Шаблон «Синхронизация на выходе из коллективной операции». Данный шаблон справедлив для всех операций релокализации, в которых используется тип синхронизации UPC_OUT_ALLSYNC. Как и в шаблоне с синхронизацией на входе, все нити обязаны дождаться друг друга, только теперь на выходе из операции. Условие срабатывания шаблона и способ вычисления временных потерь практически аналогичны шаблону «Синхронизация на входе в коллективную операцию».
4. Шаблон «Ожидание в барьере». Данный шаблон возникает в ситуациях, когда для синхронизации нитей в программе используется барьерная синхронизация. Если в приложении встретился барьер, то все нити обязаны остановиться и дождаться друг друга. Данный шаблон очень похож на шаблон «Синхронизация на входе в коллективную операцию», поэтому его описание не приводится.
5. Шаблон «Завершение барьера». Это достаточно специфический шаблон, в том смысле, что в нормальной ситуации все нити должны выходить из барьера в один и тот же момент времени. Любое, даже незначительное время, проведенное в данном шаблоне, может означать неэффективность реализации PGAS языка, либо помехи со стороны других процессов, работающих на том же расчетном узле. Формальное описание данного шаблона совпадает с шаблоном «Синхронизация на выходе из коллективной операции».
6. Шаблон «Поздняя рассылка». Данный шаблон возникает при использовании синхронизации UPC_IN_MYSYNC на входе в операции один ко многим. К ним относятся такие операции языка UPC, как: upc_all_broadcast() и upc_all_

scatter(). Если нить, рассылающая данные, входит в операцию позднее нитей, принимающих данные, то последние обязаны приостановить свое выполнение. Шаблон отражает время, потерянное в результате возникновения такой ситуации. В данном случае необходимо найти событие входа в операцию нитью, которая является источником данных. Для этого достаточно воспользоваться атрибутом, хранящим этот номер, любого события, являющегося частью коллективной операции. Также, чтобы шаблон сработал, необходимо проверить есть ли такие нити, которые вошли в операцию раньше источника. Временные потери вычисляются как разница времен между временем входа потока, который является источником данных, и временем входа в функцию последнего из потоков-получателей данных.

7. Шаблон «Ранняя сборка». Данный шаблон присущ операциям, выполняющим сборку данных, таким как upc_all_gather() и upc_all_reduce(), если для синхронизации на входе используется UPC_IN_MYSYNC. Данный шаблон аналогичен шаблону «Поздняя рассылка» за тем исключением, что причиной его возникновения является нить получатель данных, в том случае, если она входит в операцию позднее других.
8. Шаблон «Ранняя префиксная редукция». Данный шаблон уникален для операции префиксной редукции upc_all_prefix_reduce(). Возникает в случае использования синхронизации UPC_IN_MYSYNC. В данной операции результат редукции в нити n зависит от редукции, выполненной в нити $n-1$. Если хотя бы одна из нитей $0 \ldots n-1$ не вошла в операцию, нить n обязана ждать. Для того чтобы потерянное время не равнялось нулю, достаточно существования хотя бы одного события входа, которое бы произошло позже события входа в нити с большим идентификатором. Чтобы рассчитать время, потерянное в данном шаблоне, необходимо для события входа с каждой нити найти самое позднее событие входа среди нитей с меньшим идентификатором.
9. Шаблон «Синхронизация на входе в коллективную операцию многие ко многим». Данный шаблон возникает при использовании синхронизации UPC_IN_MYSYNC в операциях, которые отправляют данные от многих нитей ко многим. К таким операциям относятся upc_all_gather_all() и upc_all_exchange(). Потерянное время вычисляется аналогично шаблону «Синхронизация на входе в коллективную операцию». Данный шаблон недостаточно точен, так как часть времени, попадающего в категорию потерянного, является полезным. Это время, потраченное на обмен с нитями, уже вошедшими в операцию. На практике невозможно определить, какую часть времени нить обменивалась данными, а какую часть находилась в ожидании. Поэтому данный шаблон можно рассматривать лишь как подозрение на неэффективное поведение. Если программа затратила значительную часть времени в такой ситуации, то возможно следует уделить дополнительное внимание изучению соответствующей ситуации.
10. Шаблон «Синхронизация на выходе из коллективной операции многие ко многим» Данный шаблон аналогичен предыдущему шаблону, с тем отличием, что синхронизация и соответственно потеря времени происходит на выходе из операции.
11. Шаблон «Ожидание внутри коллективной операции». Данный шаблон возникает в операциях upc_all_broadcast(), upc_all_scatter(), upc_all_gather() и upc_all_reduce() при использовании типа синхронизации UPC_IN_MYSYNC. Этот шаблон является дополнением к шаблонам «Поздняя рассылка» и «Ранняя сборка». При выполнении коллективной операции в языке UPC может возникнуть ситуация, когда нить находится в коллективной операции одна. Это происходит, например, если нить входит в операцию первой, либо выходит последней, или если часть нитей уже выполнила вычисления и вышла, а некоторые нити еще не успели дойти до коллективной операции.
12. Шаблон «Ожидание в операции динамического выделения памяти». Данный шаблон возникает в операции upc_all_alloc(). В языке UPC присутствуют ряд операций для динамического выделения общей памяти. Если память выделяется коллективно при помощи функции upc_all_alloc(), то возникают требования к синхронизации. В действительности выделение памяти происходит в нити 0, после чего результат операции рассылается по всем нитям, аналогично операции upc_all_broadcast(). Если нулевой поток входит в операцию позже других, то остальные нити обязаны ждать рассылки результата, поэтому описание данного шаблона аналогично шаблону «Поздняя рассылка».

Заключение

Парадигма PGAS сравнительно недавно стала внедряться в практику параллельных вычислений. Данная парадигма дает разработчику возмож-

ность избежать явного указания коммуникаций, как это принято в традиционных подходах, например, в MPI. При этом используемые в PGAS-языках высокоуровневые конструкции могут приводить к неэффективному использованию вычислительных ресурсов из-за потерь на коммуникации и синхронизацию.

В данной работе предложен подход, основанный на анализе трассы выполнения параллельного приложения, на предмет наличия в ней участков, соответствующих тем или иным шаблонам. Разработано 12 шаблонов, каждый из которых отвечает за какую-то типичную ситуацию, приводящую к потерям производительности. Информация о выявленных участках кода, которые потенциально приводят к потерям производительности, предоставляется пользователю в форме отчета для дальнейшего анализа.

В будущем планируется применить разработанный подход к анализу различных программ, написанных на языке UPC. В результате возможно уточнение существующих и разработка новых шаблонов неэффективного поведения.

Работа проводилась при финансовой поддержке Министерства образования и науки Российской Федерации.

Литература

1. *Oliker L.*, *Canning A.*, *Carter J. et al.* Scientific application performance on candidate petascale platforms // In Proc. of the International Parallel & Distributed Processing Symposium (IPDPS). 2007.
2. *Ross et al.* High-End Computing Revitalization Task Force. Federal Plan for High-End Computing. 2006 . // HECIWG FSIO 2006 Workshop Report.
3. *Лупин С. А.*, *Посыпкин М. А.* Технологии параллельного программирования: Учеб. пос. Сер. Высш. образ-ние. М.: Форум Инфра-М, 2008. 208 с. 2000 экз. ISBN: 978–5–8199–0336–0.
4. *Воеводин В. В.*, *Воеводин Вл. В.* Параллельные вычисления. СПб.: БХВ, 2002. С. 608.
5. *Антонов А. С.* Технологии параллельного программирования MPI и OpenMP: Учеб. пособие. Предисл.: В. А. Садовничий. М.: Издательство Московского университета, 2012. 344 с.
6. *Jack Dongarra et al.* MPI: A Message-Passing Interface Standard, Version 3.0. High Performance Computing Center Stuttgart (HLRS), 2013, 852 p. URL: http://www.mpi-forum.org
7. Официальный сайт PGAS, URL: http://www.pgas.org/
8. *Bell C. et al.* Optimizing bandwidth limited problems using one-sided communication and overlap // In 20th International Parallel and Distributed Processing Symposium (IPDPS). 2006.
9. *Carlson W. et al.* Introduction to UPC and Language Specification. CCS-TR-99–157, IDA Center for Computing Sciences, 1999.
10. *Numrich R. W.* Co-Array Fortran for parallel programming // ACM Fortran Forum. 1998. Vol. 17. Pp. 1–31.
11. *Yelick K.*, *Semenzato L.*, *Pike G. et al.* Titanium: A High-Performance Java Dialect // Concurr. Comput.: Pract. Exper. 1998. Vol. 10. Pp. 825–836.
12. *Chamberlain B. L.*, *Callahan D.*, *Zima H. P.* Parallel programmability and the chapel language // International Journal of High Performance Computing Applications. 2007. Vol. 21. Pp. 291–312.
13. *Charles P.*, *Grothoff C*, *Saraswat V. A. et al.* X10: an object-oriented approach to non-uniform cluster computing // Proc. of the 20th annual ACM SIGPLAN conference on Object oriented programming, systems, languages, and applications / Ed. by R. Johnson, R. P. Gabriel. ACM, 2005. Pp. 519–538.
14. *Allen E.*, *Chase D.*, *Flood C. et al.* Project Fortress: A multicore language for multicore processors // Linux Magazine. 2007. Pp. 38–43.

Виноградова Юлиана Вячеславовна. Разработчик ООО «Когнитивные технологии». Окончила МИСиС в 2012 г. Количество печатных работ: 1. Область научных интересов: разработка параллельных программ, модели PGAS, обработка и анализ изображений, интеллектуальный анализ данных и распознавание образов. E-mail: vinogradus@gmail.com

Соловьев Андрей Валентинович. Технический директор ООО «Когнитивные технологии». Окончил МГУ в 1996 г. Количество печатных работ: 6. Количество патентов: более 20. Область научных интересов: информационные системы, автоматизированные системы управления, суперкомпьютерные центры. E-mail: zen@cognitive.ru