УДК 004.4'2, 004.413.5 DOI: 10.14529/cmse240205

АНАЛИЗ ИСПОЛНЕНИЯ ФРАГМЕНТИРОВАННЫХ ПРОГРАММ НА ОСНОВЕ ФАКТОРОВ SLOW*

© 2024 С.Е. Киреев[1], В.С. Литвинов[2]

[1] *Институт вычислительной математики и математической геофизики СО РАН (630090 Новосибирск, пр. академика Лаврентьева, д. 6),*

[2] *Новосибирский государственный университет (30090 Новосибирск, ул. Пирогова, д. 2)*

E-mail: kireev@ssd.sscc.ru, v.litvinov@g.nsu.ru

Поступила в редакцию: 10.05.2024

При исполнении параллельных программ, основанных на парадигме параллелизма задач, требуется решать ряд проблем, таких как выбор порядка запуска задач с учетом зависимостей между ними, распределение данных и задач по параллельным процессам, балансировка нагрузки на ресурсы. Эти проблемы относятся к области системного параллельного программирования, и их решение, как правило, обеспечивается специальной исполнительной системой. От качества решения этих проблем, а также от структуры и свойств прикладного алгоритма, лежащего в основе параллельной программы, зависит получаемая производительность. Если производительность программы недостаточна, то требуется ее оптимизация, а для этого нужно знать те причины («узкие места»), которые ограничивают ее производительность. Для определения узких мест программы обычно применяется профилирование, т.е. сбор некоторых характеристик исполнения, которые могут указать на источник проблемы. Однако обычные широко используемые средства профилирования параллельных программ не позволяют дать ответ в требуемых понятиях из-за сложности анализа асинхронного исполнения множества задач, а также из-за неспособности выделить в исполняющейся программе прикладную (множество задач) и системную (исполнительная система) компоненты. Поэтому для таких программ требуется разработка новых методов профилирования и анализа. В статье рассматривается проблема получения «понятных» характеристик выполнения параллельных программ на основе параллелизма задач для анализа производительности и оптимизации. Предлагается количественно оценить степень влияния следующих факторов: нехватка работы (Starvation), передача данных (Latency), накладные расходы (Overhead) и конфликт при доступе к общим ресурсам (Waiting for contention resolution). Представлен алгоритм получения соответствующих характеристик для системы фрагментированного программирования LuNA, а также способ их анализа для оптимизации LuNA-программ. Корректность подхода продемонстрирована на ряде синтетических экспериментов. Показано применение подхода к анализу «реальной» программы численного моделирования.

Ключевые слова: анализ производительности, параллельное программирование, фрагментированное программирование, параллелизм задач, система LuNA.

ОБРАЗЕЦ ЦИТИРОВАНИЯ

Киреев С.Е., Литвинов В.С. Анализ исполнения фрагментированных программ на основе факторов SLOW // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. 2024. Т. 13, № 2. С. 77–96. DOI: 10.14529/cmse240205.

Введение

Решение больших численных задач требует использования суперкомпьютеров, а значит, создания эффективных параллельных программ. Существует множество подходов к разработке параллельных программ. Одной из актуальных и часто используемых парадигм, использующихся в параллельном программировании, является параллелизм задач [1]. Эта парадигма предполагает, что параллельная программа описывается как множество задач,

*Статья рекомендована к публикации программным комитетом Всероссийской научной конференции с международным участием «Параллельные вычислительные технологии (ПаВТ) 2024».

которые могут выполняться независимо. Кроме того, предполагается, что задачи в рамках одной программы связаны зависимостями по данным и управлению, что ограничивает возможности по их одновременному выполнению. Использование параллелизма задач позволяет достигать высокой эффективности исполнения программ на параллельных вычислительных системах за счет того, что несколько задач могут одновременно выполняться на разных вычислительных устройствах, а обмен данными между вычислительными элементами может происходить на фоне счета. Кроме того, явное выделение независимых частей прикладного алгоритма и явное описание связей между ними позволяет автоматизировать системные функции управления параллельным исполнением задач и реализовать их в виде специальной исполнительной системы. Возможность такой автоматизации особенно важна в случае программирования в модели с распределенной или неоднородной памятью, где к проблеме управления вычислениями добавляется проблема управлением данными. Разделение описания программы на прикладную и системную часть очень удобно, оно позволяет снизить требования к прикладным пользователям суперкомпьютеров и лежит в основе многих средств повышения уровня параллельного программирования.

Ключевыми критериями качества параллельных программ для решения больших численных задач являются производительность, эффективность и масштабируемость. Качественную параллельную программу написать непросто, поэтому существует проблема оптимизации параллельных программ. Для программ, созданных в рамках парадигмы параллелизма задач, проблема оптимизации может быть разделена на две части: оптимизация описания прикладного алгоритма в виде множества взаимосвязанных задач и оптимизация управления задачами в процессе исполнения. Один из способов оптимизации заключается в том, чтобы на основе анализа исполнения параллельной программы выявить ее узкие места и внести в нее соответствующие изменения. Параллельные программы, основанные на параллелизме задач, отличаются большой динамикой исполнения (динамическое создание задач, динамическое отображение задач на вычислительные устройства, взаимодействие задач, наложение по времени и ресурсам исполнения различных частей программы и т.п.). В результате, даже обладая полной информацией об исполнении, трудно выполнить его анализ и выявить причины полученной производительности. Кроме того, многие средства профилирования параллельных программ часто не способны выделить задачи как отдельные единицы исполнения, следовательно, не отражают специфику разделения вычислений на прикладную и системную части. Для понимания причин полученной производительности необходимо получить такие характеристики исполнения, которые давали бы интегральную картину по интересующим аспектам исполнения и учитывали бы специфику используемой парадигмы, в рамках которой создана программа. Как отмечено в работе [2], факторами, ограничивающими масштабируемость параллельных программ, являются:

- Starvation — задержки вследствие недостатка работы, из-за чего не полностью используются вычислительные ресурсы;
- Latency — задержки вследствие доступа к данным, находящимся в другом процессе и требующим передачи через коммуникационную подсистему;
- Overhead — задержки вследствие решения системных задач параллельного программирования, которых не возникает в последовательной программе (управление ресурсами, задачами, данными и т.п.);
- Waiting for contention resolution — задержки вследствие синхронизации доступа к общим ресурсам.

Следуя работе [2], будем далее для обозначения этих факторов использовать аббревиатуру SLOW. Знание того, насколько каждый из этих факторов повлиял на выполнение некоторой параллельной программы, позволит выяснить пути улучшения ее производительности. Для этого необходимо разработать способ количественной оценки влияния этих факторов, а также установить связь каждого из них с конкретными характеристиками параллельной программы.

В данной работе рассматривается подход к анализу исполнения параллельных программ, основанных на парадигме параллелизма задач, на основе количественной оценки факторов SLOW. Подход демонстрируется на примере системы фрагментированного программирования LuNA, реализующей параллелизм задач [3]. Дальнейшее содержимое статьи структурировано следующим образом. Раздел 1 содержит обзор родственных работ. В разделе 2 представлено краткое описание системы LuNA. В разделе 3 рассмотрена связь особенностей LuNA-программ с факторами SLOW. В разделе 4 предложены количественные характеристики SLOW и алгоритм их вычисления в системе LuNA. В разделе 5 представлены эксперименты, подтверждающие работоспособность подхода и демонстрирующие его применение. Заключение содержит резюме результатов, достигнутых в работе, и направления будущих исследований.

1. Обзор родственных работ

В настоящее время существует множество развитых программных средств профилирования и анализа производительности параллельных программ, призванных помочь понять причины полученной производительности и предложить пути оптимизации. Широко известны, например, TAU [4], Score-P [5] и Extrae [6] — средства профилирования и трассировки, Vampir [7], Scalasca [8] и Paraver [9] — средства анализа и отображения результатов профилирования. Также широко используются средства профилирования от Intel, в частности, Intel Trace Analyzer and Collector [10]. Большинство средств поддерживают только наиболее распространенные средства параллельного программирования, такие как MPI и SHMEM для программирования в распределенной памяти, OpenMP и POSIX threads для многопоточного программирования, OpenACC, OpenCL, HIP, CUDA для программирования ускорителей. Соответственно, если их применить для параллельных программ, созданных в другой парадигме, то эти средства не смогут выделить существенные для таких программ характеристики. Поэтому для новых моделей и средств параллельного программирования необходимо разрабатывать специальные подходы к профилированию и анализу производительности.

В области анализа производительности программ на основе параллелизма задач также ведутся активные исследования [11–15]. В работе [14] приводится обзор средств для визуального анализа производительности программ, написанных в парадигме параллелизма задач. Так как в нем делается упор на средства визуализации, большинство рассмотренных в нем работ представляют собой средства графического отображения дерева или графа задач, что, вероятно, слабо применимо для анализа крупномасштабных программ. Некоторый интерес представляет средство TaskInsight [13], которое позволяет собирать различные характеристики исполнения задач и отображать полученную статистику в виде графиков и диаграмм. В частности, авторы используют его для сравнения различных стратегий планирования задач и для изучения влияния на производительность подсистемы памяти. В работе [12] используется подход, близкий к представленному в настоящей статье. Разра-

батываемое авторами средство Delay Spotter позволяет проанализировать исполнение многопоточной программы, выполняющей множество задач, разделяя время рабочих потоков на четыре компонента: 1) поток выполняет задачу, 2) поток не выполняет задачу, но готовые к выполнению задачи есть, 3) готовых для выполнения задач нет из-за планировщика, 4) готовых для выполнения задач нет из-за логики прикладной программы. Имеется возможность получить данные характеристики интегрально по всей программе, в привязке к дереву задач и в развертке по времени, что дает детальное объяснение полученной производительности. По сравнению с работой [12] подход, предлагаемый в настоящей работе, ориентирован на использование в системах с распределенной памятью, хотя лежащая в основе идея аналогична.

2. Система фрагментированного программирования LuNA

Система фрагментированного программирования LuNA [3] разрабатывается в ИВМиМГ СО РАН в рамках развития технологии фрагментированного программирования (ТФП) [16]. ТФП нацелена на автоматизацию создания эффективных параллельных реализаций численных алгоритмов для суперкомпьютеров. Ключевыми аспектами ТФП являются: явное разделение в программе описания прикладного алгоритма и системной части, отвечающей за исполнение алгоритма; представление прикладного алгоритма в явнопараллельной форме, ориентированной на автоматизацию обеспечения нефункциональных свойств. В соответствии с ТФП предполагается, что прикладной алгоритм должен быть фрагментирован, т.е. представлен в виде множества фрагментов данных и фрагментов вычислений, связанных информационными зависимостями. Таким образом, ТФП не только поддерживает парадигму параллелизма задач (фрагментацию вычислений), но и предусматривает явную фрагментацию данных. Помимо системы LuNA, похожую модель представления алгоритма для распределенных вычислений используют системы PARSEC [17], OCR [18] и Legion [19].

Система LuNA включает язык LuNA и средства исполнения программ на этом языке (LuNA-программ). LuNA-программа представляет собой компактную запись потенциально бесконечного ориентированного графа потока данных с вершинами двух видов — переменные единственного присваивания и операции однократного срабатывания. Дуги графа задают информационные зависимости между переменными и операциями (отношения входа и выхода). Этот граф будем называть графом программы. В терминологии ТФП переменные называются фрагментами данных, а операции — фрагментами вычислений. Каждая операция реализуется некоторым фрагментом кода. Различаются атомарные фрагменты кода (подпрограммы на языке C++) и структурированные фрагменты кода (подпрограммы на языке LuNA). Таким образом, фрагмент вычислений — это фрагмент кода, примененный к конкретным входным и выходным фрагментам данных. Язык LuNA поддерживает условные фрагменты вычислений для описания ветвящихся алгоритмов, индексированные имена фрагментов данных и вычислений, чтобы оперировать массивами фрагментов, операторы for и while для компактной записи множеств фрагментов. Далее будем считать, что граф программы содержит только атомарные фрагменты вычислений, рассматривая структурированные фрагменты кода как компактную запись некоторого подграфа программы. Потенциальная бесконечность графа программы является следствием заранее неизвестного (вычисляемого) количества фрагментов, описываемых операторами for и while, а также

заранее неизвестной (вычисляемой) глубины рекурсии структурированных фрагментов вычислений.

Процесс выполнения LuNA-программы предполагает запуск конкретных фрагментов вычислений в порядке, соответствующем информационным зависимостям, при этом конкретные фрагменты данных получают свои значения, и выбираются конкретные ветви исполнения. Таким образом, реализуется некоторый конечный подграф исходного потенциально бесконечного графа программы. Этот конечный подграф будем называть графом исполнения.

В ходе развития проекта было разработано несколько реализаций системы LuNA. Описываемая в данной работе реализация упоминается в работах [20–22] и основана на исполнительной системе, построенной на основе MPI и C++ threads. Данная исполнительная система обеспечивает динамическое отображение фрагментов данных и вычислений на процессы в соответствии с заданным способом распределения. При этом каждому фрагменту данных назначается номер процесса, где он должен храниться, а фрагменту вычислений назначается номер процесса, где он должен выполниться. Для исполнения фрагментов вычислений в каждом процессе используется пул рабочих потоков. Исполнительная система поддерживает динамическую балансировку нагрузки путем динамического изменения отображения фрагментов на процессы. В работах [23, 24] представлено сравнение рассматриваемой здесь исполнительной системы со следующей версией системы LuNA.

3. Факторы SLOW в системе LuNA

Во введении были рассмотрены факторы SLOW, ограничивающие масштабируемость, а значит, производительность и эффективность параллельных программ. В данном разделе мы установим связь этих факторов с процессом исполнения LuNA-программ, рассмотрим, какие особенности LuNA-программ и исполнительной системы приводят к преобладанию того или иного фактора, и какими способами влияние этих факторов может быть уменьшено.

3.1. Факторы SLOW в процессе исполнения LuNA-программ

Рассмотрим особенности исполнения фрагментов вычислений в распределенной исполнительной системе LuNA. Фрагмент вычислений готов к запуску только тогда, когда все его входные фрагменты данных вычислены и находятся в том же процессе, что и сам фрагмент вычислений. В таком случае этот фрагмент вычислений формирует задачу, которая помещается в очередь задач, потребляемых пулом рабочих потоков данного процесса. Если входные фрагменты данных не вычислены, то рассматриваемый фрагмент вычислений не готов к запуску, т.е. он не формирует задачу. В принципе, он может никогда не стать готовым, если это предусматривается логикой программы. Когда фрагменты вычислений не становятся готовыми по причине отсутствия вычисленных входных фрагментов данных, и при этом происходит простой рабочих потоков, тогда такая ситуация называется «голодание» (фактор Starvation).

Если в некоторый момент времени входные фрагменты данных вычислены, но находятся в другом процессе, то известно, что рассматриваемый фрагмент вычислений в будущем гарантированно сформирует задачу, которая будет запущена, но пока он вынужден ждать получения данных. Это вынужденное ожидание, связанное с передачей фрагментов данных к месту назначения и приводящее к простою рабочих потоков, является фактором Latency.

Кроме исполнения фрагментов вычислений и ожидания по различным причинам, вычислительные устройства (ядра) исполняют код исполнительной системы. Эта работа необходима, но она также является сдерживающим фактором (фактором Overhead), поскольку занимает вычислительные ресурсы.

Что касается фактора Waiting for contention resolution, то, вообще говоря, модель исполнения LuNA-программы не предполагает наличия общих ресурсов благодаря тому, что фрагменты данных записываются единожды. Но некоторая конкретная реализация исполнительной системы может использовать такие общие ресурсы, например, переиспользуемые участки памяти для хранения фрагментов данных. Рассматриваемая в данной работе реализация исполнительной системы LuNA общих ресурсов не использует, поэтому далее будем считать, что на LuNA-программу данный фактор не действует, и рассматривать его мы не будем.

3.2. Связь особенностей LuNA-программ и исполнительной системы с факторами SLOW

Рассмотрим для каждого фактора, какие особенности LuNA-программ и работы исполнительной системы приводят к его чрезмерному влиянию, а также какими способами это влияние может быть уменьшено.

- Starvation — недостаток готовых фрагментов вычислений для рабочих потоков.
  - Информационные зависимости между фрагментами вычислений могут приводить к тому, что степень параллелизма в программе оказывается недостаточно высокой, чтобы загрузить все вычислительные устройства (программа «слишком последовательная»). Причиной может быть как низкая степень фрагментации алгоритма, т.е. данные и вычисления декомпозированы на недостаточное количество частей, так и свойства исходного прикладного алгоритма, например, если прикладной алгоритм принципиально последовательный.
  - Плохое распределение фрагментов вычислений по процессам может приводить к тому, что готовые фрагменты вычислений скапливаются на одних процессах, в то время как рабочие потоки других процессов простаивают. Следует изменить распределение фрагментов вычислений по процессам или применить динамическую балансировку нагрузки.
  - Неверный порядок выбора готовых фрагментов вычислений для исполнения может приводить к тому, что фрагменты вычислений на критическом пути алгоритма будут выбираться в последнюю очередь — только тогда, когда для других рабочих потоков работы уже не осталось. Фрагменты вычислений на критическом пути следует запускать на исполнение в приоритетном порядке.
- Latency — ожидание передачи фрагментов данных.
  - Причиной сильного влияния этого фактора может быть плохое взаимное распределение фрагментов данных и фрагментов вычислений по процессам, т.е. либо фрагменты данных вырабатываются далеко от тех процессов, в которых они должны храниться, либо они хранятся далеко от тех фрагментов вычислений, которые их потребляют. Общее правило состоит в том, что фрагменты, непосредственно связанные дугами в графе алгоритма, следует размещать в одном и том же или близко расположенных процессах.

  - Затраты на ожидание данных иногда можно уменьшить путем разбиения фрагментов на более мелкие (увеличения степени фрагментации), что позволяет передачу мелких фрагментов данных (частей изначально большого фрагмента данных) совместить с их обработкой.
  - Уменьшить количество передач данных между процессами также можно с помощью алгоритмов дублирования и кэширование фрагментов данных, которые могут быть реализованы в исполнительной системе.
- Overhead — работа исполнительной системы.
  - Слишком мелкая фрагментация алгоритма порождает большое количество фрагментов, и исполнительной системе приходится тратить время на управление ими. Уменьшение степени фрагментации может уменьшить накладные расходы.
  - На величину накладных расходов влияет способ реализации исполнительной системы. Например, часть решений по организации исполнения может быть перенесена на этап компиляции программы, как это сделано в последующей версии системы LuNA [23, 24]. Для некоторых классов алгоритмов возможно использование специализированных исполнительных систем, в которых накладные расходы сведены к минимуму [25, 26].
  - В тех случаях, когда одну и ту же LuNA-программу требуется исполнять многократно с разными входными данными, но при условии неизменности графа исполнения, возможно применение техники «проигрывания трассы» для существенного уменьшения накладных расходов [27]. Данная техника предполагает, что при первом запуске LuNA-программы сохраняется трасса исполнения, содержащая информацию о том, в каких процессах и в каком порядке исполнялись фрагменты вычислений. При последующих запусках этой же LuNA-программы сложная исполнительная система заменяется простой, которая только запускает фрагменты вычислений в соответствии с имеющейся трассой. Более того, так как трасса содержит информацию о длительности исполнения фрагментов вычислений, появляется возможность спланировать более эффективное распределение фрагментов по процессам [28].

4. Количественная оценка факторов SLOW в системе LuNA

4.1. Характеристики SLOW

Для количественной оценки факторов SLOW введем характеристики SLOW: P_S, P_L, P_O, P_W. Это интегральные характеристики исполнения программы, представляющие собой долю процессорного времени (в процентах от общего), затраченного в результате действия каждого из этих факторов. Общее затраченное процессорное время (T_{total}) определим как время работы программы по таймеру системного времени (T_{wall}), умноженное на количество задействованных вычислительных устройств (ядер). Например, если программа работала две секунды и занимала четыре ядра, то T_{total} равно восемь секунд. Дополнительно имеет смысл рассмотреть характеристику P_U — долю процессорного времени, затраченного на полезные вычисления (Useful work). Для введенных характеристик должно выполняться соотношение: $P_S + P_L + P_O + P_W + P_U = 100\%$. Значения характеристик SLOW показывают степень влияния того или иного фактора на полученную производительность рассматриваемой параллельной программы на данной вычислительной системе. Если какие-то факторы

имеют особенно большое влияние, то именно на них должны быть направлены усилия по оптимизации.

Идея способа измерения характеристик SLOW состоит в том, чтобы в общем времени исполнения параллельной программы выделить периоды полезной работы, а остальное время разделить на части в соответствии с факторами SLOW. Введем соответствующий набор временны́х характеристик и покажем, чему они соответствуют в применении к исполнению LuNA-программ:

- T_S — время простоя рабочих потоков, связанное с отсутствием готовых к исполнению фрагментов вычислений;
- T_L — время простоя рабочих потоков, связанное с ожиданием получения готовых фрагментов данных, являющихся входными для фрагментов вычислений;
- T_O — время, затраченное на работу исполнительной системы;
- T_W — время простоя рабочих потоков, связанное с тем, что некоторый фрагмент вычислений ожидает освобождения некоторого общего ресурса (в рамках его процесса), занятого другим фрагментом вычислений или исполнительной системой;
- T_U — время работы рабочих потоков, затраченное на исполнение фрагментов вычислений.

Для временных характеристик выполняется соотношение: $T_S + T_L + T_O + T_W + T_U = T_{total}$. Характеристики SLOW тогда будут определяться так: $P_X = T_X / T_{total} \times 100\%$, где X — одно из S, L, O, W, U. Как было отмечено выше, у фрагментов вычислений LuNA-программы нет общих ресурсов, которые могут вызвать ожидание, поэтому $T_W = 0$ и $P_W = 0$. Новый набор факторов и характеристик (с добавлением U и без W) будем обозначать аббревиатурой SLOU.

4.2. Вычисление характеристик SLOU в процессе исполнения LuNA-программы

Для вычисления временных характеристик необходимо выполнить профилирование LuNA-программы, собирая в процессе исполнения информацию о следующих событиях: запуск фрагмента вычислений, завершение фрагмента вычислений, отправка фрагмента данных от одного процесса другому, получение процессом фрагмента данных. При этом требуется также отмечать сопутствующую информацию о событиях: время по системному таймеру, время по таймеру потока, номер процесса, номер рабочего потока, идентификаторы фрагментов данных и фрагментов вычислений.

Как известно, исполнение потоков может прерываться другими процессами и потоками этой же параллельной программы, других программ и операционной системы. Кроме того, операционная система может переносить поток с одного ядра на другое. В таких условиях становится сложно получить адекватные временные оценки исполнения интересующих нас частей программы. Поэтому ограничим условия запуска: будем считать, что рабочие потоки привязаны к ядрам, а потоки исполнительной системы — не обязательно привязаны и могут прерывать рабочие потоки. Кроме того, пусть разные процессы параллельной программы занимают различные непересекающиеся подмножества ядер. Данные условия могут быть удовлетворены при соответствующей настройке исполнительной системы и параметров запуска параллельной программы. Влияние других программ и операционной системы на работу параллельной программы будем считать пренебрежимо малым, хотя в реальности этого и не всегда удается достичь.

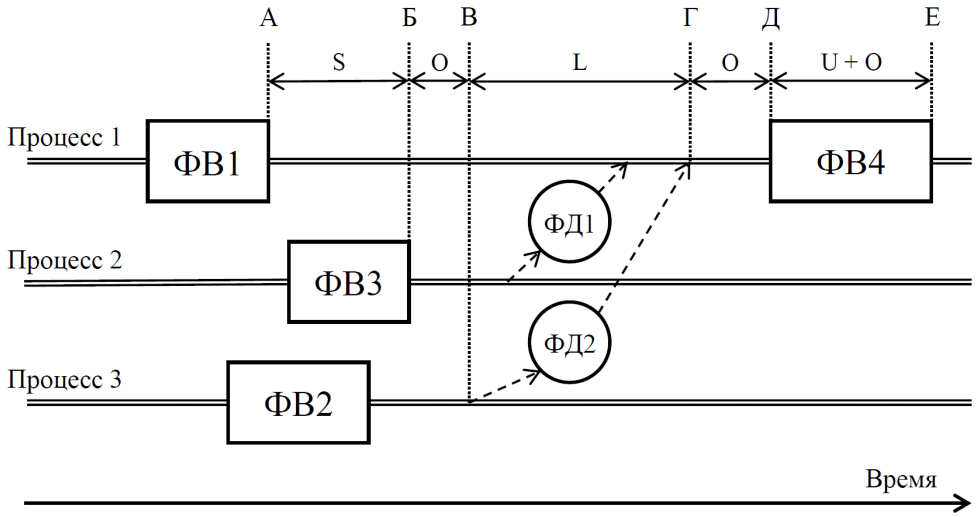


Рис. 1. Временная диаграмма исполнения LuNA-программы. Буквами S, L, O, U обозначены факторы, влияющие на продолжительность данного периода времени: Starvation, Latency, Overhead, Useful work

Опишем алгоритм вычисления временных характеристик, используя конкретный пример. На рис. 1 приведена часть временной диаграммы исполнения LuNA-программы, состоящей из трех процессов, имеющих по одному рабочему потоку. На этом временном отрезке процесс 1 выполнил фрагменты вычислений ФВ1 и ФВ4, процесс 2 — ФВ3, а процесс 3 — ФВ2. Кроме того, ФВ4 принимает на вход фрагменты данных ФД1 и ФД2, выработанные соответственно фрагментами вычислений ФВ3 и ФВ2 (другие информационные зависимости не показаны). Буквами А-Е обозначены интересующие нас моменты времени.

Чтобы получить время полезной работы T_U, нужно измерить и просуммировать времена исполнения всех фрагментов вычислений. Например, на рис. 1 время исполнения ФВ4 обозначено отрезком ДЕ. Однако исполнение фрагмента вычислений может прерываться потоками исполнительной системы. Поэтому для определения реально затраченного полезного времени (вклад в характеристику T_U) необходимо замерить продолжительность этого интервала по таймеру времени потока, а вычтя это время из продолжительности интервала ДЕ по таймеру системного времени, мы получим вклад в характеристику T_O.

Далее необходимо проанализировать все временные интервалы, когда рабочие потоки не были заняты полезной работой. Для примера рассмотрим анализ интервала АД для единственного потока процесса 1 на рис. 1. Часть этого времени данный поток выполнял код исполнительной подсистемы, а часть — находился в ожидании из-за факторов Starvation и Latency, — это те причины, по которым ФВ4 не запустился раньше. Можно было бы сказать, что во время интервала АД рабочий поток простаивал не только из-за неготовности ФВ4, но и из-за неготовности каких-то других фрагментов вычислений, которые могли бы его «обогнать» и запуститься раньше. Почему же мы рассматриваем причины, сдерживавшие запуск именно ФВ4? Потому что именно он из всех кандидатов оказался «наиболее близок» к готовности и запустился в данное время в данном потоке. При другом исполнении этой же LuNA-программы другой фрагмент вычислений мог бы «обогнать» ФВ4 и запуститься в этом потоке, и тогда бы мы анализировали причины задержки другого фрагмента вычислений.

В соответствии с логикой, изложенной в предыдущем разделе, разделим интервал АД следующим образом. Рассмотрим все фрагменты вычислений, которые вырабатывают фрагменты данных, входные для ФВ4. Это фрагменты вычислений ФВ2 и ФВ3. Определим момент времени, когда завершился последний из них — это момент Б. С этого момента все входные фрагменты данных для ФВ4 готовы, и он потенциально уже может запуститься. Значит, в интервале АБ ФВ4 не был готов из-за факторов Starvation и Overhead, а в интервале БД — из-за факторов Latency и Overhead. Попробуем теперь выделить из этих интервалов фактор Overhead.

Если нам известны моменты отправки и приема ФД1 и ФД2, то мы можем определить, когда процессом 1 получен последний из них. Это момент Г, когда был доставлен ФД2. С этого момента запуск ФВ4 откладывался только из-за накладных расходов на работу системных алгоритмов, значит, период ГД дает вклад в характеристику T_O. Кроме того, ФД2 был вычислен процессом сразу после завершения ФВ2, но был отправлен только в момент В. Причиной этой задержки также будем считать работу системных алгоритмов. Пересечение этой задержки с интервалом БД будет интервал БВ, который тоже дает вклад в характеристику T_O. Оставшийся интервал ВГ будем относить к фактору Latency, т.е. вклад в характеристику T_L. Даже если в этом интервале исполнение рабочего потока прерывалось потоками исполнительной системы, то все равно не имеет смысла учитывать эти события, т.к. их возможное отсутствие не приблизило бы момент получения ФД2. Также и интервал АБ мы полностью отнесем к фактору Starvation (вклад в характеристику T_S) по той причине, что исключение возможных прерываний рабочего потока процесса 1 не приблизит момент завершения ФВ3.

Итак, анализируя рассмотренным выше способом все время функционирования всех рабочих потоков и суммируя все вклады, мы получаем характеристики T_S, T_L, T_O и T_U. При использовании нескольких рабочих потоков в рамках одного процесса принцип подсчета характеристик остается тем же. При этом считается, что время передачи фрагмента данных от одного потока к другому в рамках одного процесса (вклад в характеристику T_L) равно нулю. Если исполнительная система дополнительно использовала другие ядра вычислительной системы кроме тех, к которым привязаны рабочие потоки, то все время использования этих ядер следует прибавить к характеристике T_O.

5. Вычислительные эксперименты

Представленный выше алгоритм вычисления временных характеристик T_S, T_L, T_O и T_U был реализован в исполнительной системе LuNA. В данной реализации информация о нужных событиях в ходе исполнения LuNA-программы сохраняется в файл, а после завершения LuNA-программы специальная утилита анализирует собранную информацию и вычисляет временные характеристики SLOU. Ниже представлены эксперименты двух видов. Первые три подраздела содержат синтетические эксперименты, нацеленные на проверку работоспособности предложенного подхода. Каждый из этих экспериментов основан на специальным образом сформированной LuNA-программе с некоторым параметром, от значения которого зависит степень влияния заданного фактора. Эксперименты показывают, что получаемые значения временных характеристик действительно меняются так, как предполагалось. Это подтверждает, что характеристики LuNA-программы действительно связаны с оцениваемыми временными характеристиками исполнения ожидаемым образом. В последнем подразделе представлен анализ исполнения LuNA-программы, решающей «реальную» за-

Рис. 2. Зависимость временных характеристик от степени параллелизма

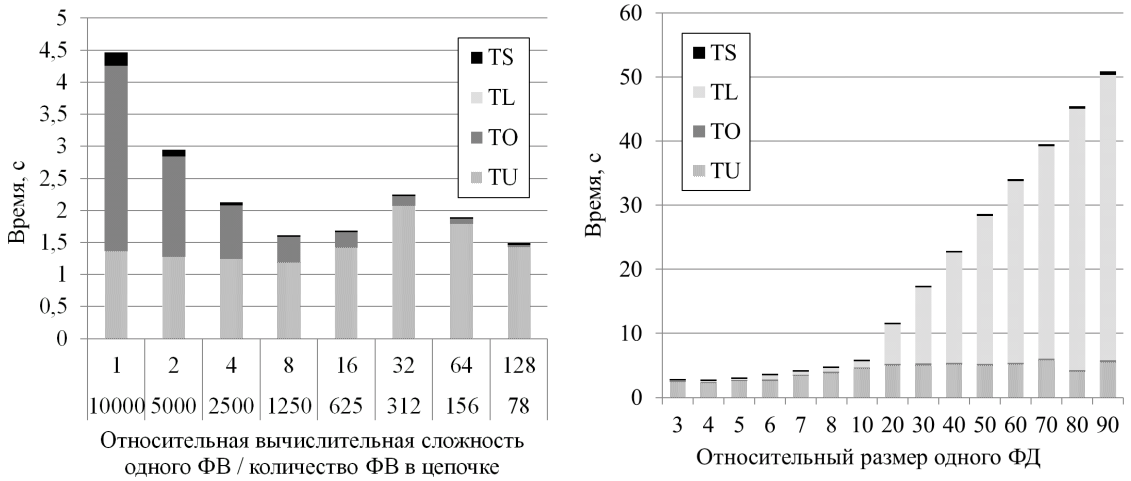
дачу численного моделирования. Все эксперименты проводились на кластере МСЦ РАН [29] МВС-10П Торнадо. Узлы кластера содержат два 8-ядерных процессора Intel Xeon E5-2690 2.9 ГГц (16 ядер на узел) и два ускорителя Intel Xeon Phi 7110X (не использовались в экспериментах).

5.1. Влияние фактора Starvation

Цель эксперимента — продемонстрировать влияние степени параллелизма фрагментированного алгоритма на характеристику T_S. LuNA-программа представляет собой N независимых цепочек ФВ-ФД-ФВ-ФД-...-ФВ постоянной длины (100 фрагментов вычислений в одной цепочке) с одинаковой вычислительной нагрузкой для всех фрагментов вычислений. Фрагменты вычислений внутри каждой цепочки вынуждены исполняться последовательно из-за информационных зависимостей. Вычислительная сложность всех фрагментов вычислений бралась одинаковой, обратно пропорциональной N, чтобы суммарная вычислительная сложность программы сохранялась постоянной. Выполнялась серия запусков программы с количеством цепочек N, изменяющимся от 1 до 16. Запуски производились на одном 16-ядерном узле с одним процессом и 8-ю рабочими потоками. Ожидаемый результат эксперимента — постепенное уменьшение характеристики T_S почти до нуля с ростом N от 1 до 8 в связи с увеличением степени параллелизма LuNA-программы и занятием простаивающих рабочих потоков. При значениях N от 8 до 16 ожидалось пренебрежимо малое значение характеристики T_S (накапливаемое в момент завершения работы программы). Результаты эксперимента, представленные на рис. 2, демонстрируют ожидаемое поведение характеристики T_S. Поведение характеристики T_O также понятно — его значение немного увеличивается с ростом числа фрагментов. Значение характеристики T_L равно нулю, поэтому на рис. 2 ее не видно. Вопросы вызывает только изменение характеристики T_U, особенно скачок при $N = 6$.

5.2. Влияние фактора Overhead

Цель данного эксперимента — продемонстрировать изменение доли времени, которое тратится на работу исполнительной системы при изменении степени фрагментации задачи. LuNA-программа представляет собой одну цепочку ФВ-ФД-ФВ-ФД-...-ФВ с одинаковой вычислительной нагрузкой для всех фрагментов вычислений — по сути, последователь-



(а) (б)

Рис. 3. Зависимость временных характеристик от параметров фрагментированной программы: а) степени фрагментации, б) объема передаваемых данных

ная программа. Запуски производились на одном процессе с одним рабочим потоком. Была выполнена серия запусков с постепенно уменьшающейся степенью фрагментации. При каждом последующем запуске вычислительная нагрузка всех фрагментов вычислений удваивалась, а длина цепочки фрагментов сокращалась в два раза, чтобы общая вычислительная сложность LuNA-программы оставалась неизменной. От данного эксперимента ожидалось уменьшение характеристики T_O с уменьшением степени фрагментации. Результаты, представленные на рис. 3а, в целом соответствуют ожиданиям. Как и в прошлом эксперименте, здесь остается непонятным изменение характеристики T_U при том, что вычислительная сложность программы должна была оставаться постоянной. Значение характеристики T_L здесь также равно нулю, и на рис. 3а ее не видно.

5.3. Влияние фактора Latency

Цель данного эксперимента — исследовать изменение характеристики T_L при изменении объема передаваемых данных между процессами. Для этого LuNA-программа запускалась на двух процессах на разных узлах кластера, в каждом из которых работало по одному рабочему потоку. Сама LuNA-программа состоит из двух цепочек А и Б вида ФВ-ФД-ФВ-ФД-...-ФВ равной длины. В цепочке А каждый четный фрагмент вычислений назначается на первый процесс, а каждый нечетный — на второй. В цепочке Б, наоборот, каждый четный фрагмент вычислений назначается на второй процесс, а каждый нечетный — на первый. Таким образом, каждый вычисленный в некотором процессе фрагмент данных должен быть передан в другой процесс. Вычислительная нагрузка для каждого фрагмента вычислений была выбрана одинаковой для того, чтобы уменьшить значение характеристики T_S в предположении, что интервалы времени, в течение которых работают i-ые фрагменты вычислений из разных цепочек, примерно совпадают. Эксперимент состоит из серии запусков. В каждом запуске размер всех фрагментов данных совпадает, но растет от запуска к запуску. Предполагается, что T_L будет увеличиваться при увеличении размера фрагментов данных. Результаты эксперимента, представленные на рис. 3б, соответствуют ожиданиям.

5.4. Анализ LuNA-реализации PIC-метода

В данном подразделе представлен анализ «реальной» задачи численного моделирования на основе факторов SLOU. В качестве «реальной» задачи использовалась реализованная в системе LuNA задача трехмерного моделирования самогравитирующего пылевого облака методом частиц-в-ячейках (PIC-методом) [30]. Реализация этой задачи в системе LuNA упоминалась в [25]. Для эксперимента были взяты следующие параметры:

- размеры пространственной сетки: $100 \times 100 \times 100$,
- количество модельных частиц: 1 000 000,
- число шагов по времени: 10,
- фрагментация области моделирования — количество подобластей, на которые она разбивается по каждому направлению: $4 \times 4 \times 4$.

В табл. 1 представлены полученные результаты для двух запусков задачи на разных ресурсах: одном и четырех узлах кластера, по одному процессу (16 рабочих потоков) на узел.

Таблица 1. Характеристики SLOU двух вариантов запуска задачи на разных ресурсах

	T_{wall}	T_{SLOU} (P_{SLOU})	T_S (P_S)	T_L (P_L)	T_O (P_O)	T_U (P_U)
1 узел × 16 потоков	9.99 с	153.8 с (96.25%)	30.1 с (18.84%)	0 с (0%)	110.6 с (69.21%)	13.1 с (8.20%)
4 узла × 16 потоков	164.05 с	10471 с (99.73%)	7669 с (73.05%)	1556 с (14.82%)	1234 с (11.75%)	12 с (0.11%)

Из полученных результатов можно сделать следующие выводы.

- При сравнении значений T_{wall} для двух разных запусков задачи видно, что, несмотря на увеличение общего количества доступных ядер в четыре раза, время счета задачи увеличивается почти в 16 раз. Подобное замедление на данной версии исполнительной системы LuNA наблюдалось и на других задачах [20].
- Сумма полученных значений временных характеристик (T_{SLOU}) составляет более 96% от общего затраченного процессорного времени T_{total}, что вполне достаточно для определения основных проблем в реализации. С другой стороны, незначительная величина процессорного времени осталась неучтенной, что говорит о том, что реализацию алгоритма определения характеристик еще можно улучшать.
- Характеристика P_O даже при запуске на одном узле составляет около 69%, что свидетельствует о высоких накладных расходах на работу исполнительной системы. Кроме того, при использовании нескольких узлов характеристика T_O вырастает более чем в 10 раз. Для ее снижения, вероятно, имеет смысл оптимизировать исполнительную систему или переписать ее на других принципах (что и было сделано, см. [23, 24]). Используя же текущую версию, можно попробовать уменьшить степень фрагментации задачи.
- При запуске задачи на нескольких узлах кластера значение характеристики P_S составило около 73%, т.е. фактор Starvation в данном случае является главной причиной низкой производительности. Это свидетельствует о том, что либо для решения данной задачи не требуется использовать так много вычислительных ресурсов, либо наблюдается дисбаланс нагрузки. Эффективность использования ресурсов, вероятно, можно

увеличить, увеличив степень фрагментации задачи или включив динамическую балансировку нагрузки.

- Время полезной работы T_U для двух запусков немного отличается, хотя задача решалась одна и та же. Подобная разница в значениях T_U наблюдалась и в синтетических экспериментах. Причины этого еще предстоит выяснить.

Заключение

В работе предложен способ анализа исполнения фрагментированных программ на основе сравнительной оценки влияния факторов SLOW. Показано, как данные факторы связаны с различными характеристиками фрагментированных программ и исполнительной системы. Предложены количественные характеристики SLOW, отражающие степень влияния факторов SLOW на производительность. Представлен алгоритм вычисления количественных характеристик для одной реализации системы LuNA. Применимость подхода для анализа производительности LuNA-программ подтверждается экспериментами.

Предложенный подход может дополнить существующие способы анализа производительности параллельных программ, основанных на параллелизме задач. В частности, данный подход может применяться и к другим системам параллельного программирования, основанным на данной парадигме [17–19], и быть интегрирован в специализированные средства анализа производительности для этих систем.

Продолжением данной работы может быть исследование способов локализации проблем производительности на основе факторов SLOW, например, получение характеристик SLOW для частей программы, для отдельных вычислительных ресурсов (узлов, ядер), для заданных отрезков времени исполнения (получение динамики характеристик SLOW), определение «проблемных» фрагментов данных и вычислений и т.п.

Исследования выполнены в рамках государственного задания ИВМиМГ СО РАН FWNM-2022-0005.

Литература

1. Thoman P., Dichev K., Heller T., *et al.* A taxonomy of task-based parallel programming technologies for high-performance computing // The Journal of Supercomputing. 2018. Vol. 74. P. 1422–1434. DOI: 10.1007/s11227-018-2238-4.

2. Kaiser H., Heller T., Adelstein-Lelbach B., *et al.* HPX: A Task Based Programming Model in a Global Address Space // 8th International Conference on PGAS Programming Models, PGAS'2014, Eugene OR, USA, October 6–10, 2014. Proceedings. Article 6. ACM, 2014. P. 1–11. DOI: 10.1145/2676870.2676883.

3. Malyshkin V.E., Perepelkin V.A. LuNA Fragmented Programming System, Main Functions and Peculiarities of Run-Time Subsystem // 11th International Conference on Parallel Computing Technologies, PaCT-2011, Kazan, Russia, September 19–23, 2011. Proceedings. Vol. 6873 / ed. by V. Malyshkin. Springer, 2011. P. 53–61. Lecture Notes in Computer Science. DOI: 10.1007/978-3-642-23178-0_5.

4. Shende S., Malony A.D. The TAU Parallel Performance System // International Journal of High Performance Computing Applications. 2006. Vol. 20, no. 2. P. 287–311. DOI: 10.1177/1094342006064482.

5. Lorenz D., Feld C. Scaling Score-P to the next level // Procedia Computer Science. 2017. Vol. 108. P. 2180–2189. DOI: 10.1016/j.procs.2017.05.107.

6. Extrae instrumentation package. URL: `https://tools.bsc.es/extrae` (дата обращения: 23.04.2024).

7. Vampir 10.4. URL: `https://vampir.eu/` (дата обращения: 23.04.2024).

8. Zhukov I., Feld C., Geimer M., *et al.* Scalasca v2: Back to the Future // 8th International Workshop on Parallel Tools for High Performance Computing, HLRS, Stuttgart, Germany, October, 2014. Proceedings. Ed. by C. Niethammer, J. Gracia, A. Knüpfer, *et al.* Springer, 2015. P. 1–24. DOI: 10.1007/978-3-319-16012-2_1.

9. Mantovani F., Calore E. Multi-Node Advanced Performance and Power Analysis with Paraver // Advances in Parallel Computing. Vol. 32. IOS Press, 2018. P. 723–732. DOI: 10.3233/978-1-61499-843-3-723.

10. Intel Trace Analyzer and Collector. URL: `https://www.intel.com/content/www/us/en/developer/tools/oneapi/trace-analyzer.html` (дата обращения: 23.04.2024).

11. Huynh A., Thain D., Pericàs M., Taura K. DAGViz: a DAG visualization tool for analyzing task-parallel program traces // 2nd WS on Visual Performance Analysis, VPA '15, November, 2015. Proceedings. No. 3. ACM, 2015. P. 1–8. DOI: 10.1145/2835238.2835241.

12. Huynh A., Taura K. Delay Spotter: A Tool for Spotting Scheduler-Caused Delays in Task Parallel Runtime Systems // 2017 IEEE International Conference on Cluster Computing, CLUSTER, Honolulu, HI, USA, September 5–8, 2017. IEEE, 2017. P. 114–125. DOI: 10.1109/CLUSTER.2017.82.

13. Ceballos G., Grass T., Hugo A., Black-Schaffer D. Analyzing performance variation of task schedulers with TaskInsight // Parallel Computing. 2018. Vol. 75. P. 11–27. DOI: 10.1016/j.parco.2018.02.003.

14. Pinto V.G. Performance Analysis Strategies for Task-based Applications on Hybrid Platforms: PhD thesis / Vinícius Garcia Pinto. Universidade Federal do Rio Grande do Sul - UFRGS, Brazil, UGA - Université Grenoble Alpes, France, 2018. URL: `https://theses.hal.science/tel-01962333`.

15. Pinto V.G., Nesi L.L., Miletto M.C., Schnorr L.M. Providing In-depth Performance Analysis for Heterogeneous Task-based Applications with StarVZ // 2021 IEEE International Parallel and Distributed Processing Symposium Workshops, IPDPSW, Portland, OR, USA, June 17–21, 2021. IEEE, 2021. P. 16–25. DOI: 10.1109/IPDPSW52791.2021.00013.

16. Малышкин В.Э. Технология фрагментированного программирования // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. 2012. № 46(305), Вып. 1. С. 45–55. DOI: 10.14529/cmse120104.

17. Bosilca G., Bouteiller A., Danalis A., *et al.* PaRSEC: Exploiting Heterogeneity to Enhance Scalability // Computing in Science & Engineering. 2013. Vol. 15, no. 6. P. 36–45. DOI: 10.1109/MCSE.2013.98.

18. Dokulil J., Benkner S. The OCR-Vx experience: lessons learned from designing and implementing a task-based runtime system // The Journal of Supercomputing. 2022. Vol. 78. P. 12344–12379. DOI: 10.1007/s11227-022-04355-0.

19. Bauer M., Treichler S., Slaughter E., Aiken A. Legion: Expressing locality and independence with logical regions // International Conference on High Performance Computing, Networking, Storage and Analysis, SC'12, Salt Lake City, UT, USA, November 10–16, 2012. Proceedings. IEEE, 2012. P. 1–11. DOI: 10.1109/SC.2012.71.

20. Akhmed-Zaki D., Lebedev D., Perepelkin V. Implementation of a three dimensional three-phase fluid flow ("oil-water-gas") numerical model in LuNA fragmented programming system // The Journal of Supercomputing. 2017. Vol. 73. P. 624–630. DOI: 10.1007/s11227-016-1780-1.

21. Daribayev B., Perepelkin V., Lebedev D., Akhmed-Zaki D. Implementation of the Two-Dimensional Elliptic Equation Model in LuNA Fragmented Programming System // IEEE 12th International Conference on Application of Information and Communication Technologies, AICT 2018, Almaty, Kazakhstan, October 17–19, 2018. Proceedings. IEEE, 2018. P. 161–164. DOI: 10.1109/ICAICT.2018.8747145.

22. Akhmed-Zaki D., Lebedev D., Perepelkin V. Implementation of a 3D model heat equation using fragmented programming technology // The Journal of Supercomputing. 2019. Vol. 75, no. 12. P. 7827–7832. DOI: 10.1007/s11227-018-2710-1.

23. Akhmed-Zaki D., Lebedev D., Malyshkin V., Perepelkin V. Automated Construction of High Performance Distributed Programs in LuNA System // 15th International Conference on Parallel Computing Technologies, PaCT 2019, Almaty, Kazakhstan, August 19–23. Proceedings. Vol. 11657 / ed. by V. Malyshkin. Springer, 2019. P. 3–9. Lecture Notes in Computer Science. DOI: 10.1007/978-3-030-25636-4_1.

24. Малышкин В.Э., Перепелкин В.А. Мультиагентный подход к повышению эффективности исполнения фрагментированных программ в системе LuNA // Проблемы информатики. 2023. № 3. С. 55–67. DOI: 10.24412/2073-0667-2023-3-55-67.

25. Belyaev N., Kireev S. LuNA-ICLU Compiler for Automated Generation of Iterative Fragmented Programs // 15th International Conference on Parallel Computing Technologies, PaCT 2019, Almaty, Kazakhstan, August 19–23, 2019. Proceedings. Vol. 11657 / ed. by V. Malyshkin. Springer, 2019. P. 10–17. Lecture Notes in Computer Science. DOI: 10.1007/978-3-030-25636-4_2.

26. Belyaev N., Perepelkin V. High-Efficiency Specialized Support for Dense Linear Algebra Arithmetic in LuNA System // 16th International Conference on Parallel Computing Technologies, PaCT 2021, Kaliningrad, Russia, September 13–18, 2021. Proceedings. Vol. 12942 / ed. by V. Malyshkin. Springer, 2021. P. 143–150. Lecture Notes in Computer Science. DOI: 10.1007/978-3-030-86359-3_11.

27. Malyshkin V., Perepelkin V. Trace-Based Optimization of Fragmented Programs Execution in LuNA System // 16th International Conference on Parallel Computing Technologies, PaCT 2021, Kaliningrad, Russia, September 13–18, 2021. Proceedings. Vol. 12942 / ed. by V. Malyshkin. Springer, 2021. P. 3–10. Lecture Notes in Computer Science. DOI: 10.1007/978-3-030-86359-3_1.

28. Malyshkin V., Perepelkin V., Lyamin A. Trace Balancing Technique for Trace Playback in LuNA System // 17th International Conference on Parallel Computing Technologies, PaCT 2023, Astana, Kazakhstan, August 21–25, 2023. Proceedings. Vol. 14098 /

ed. by V. Malyshkin. Springer, 2023. P. 42–50. Lecture Notes in Computer Science. DOI: 10.1007/978-3-031-41673-6_4.

29. Межведомственный Суперкомпьютерный Центр Российской Академии Наук. URL: https://www.jscc.ru/ (дата обращения: 23.04.2024).

30. Kireev S. A Parallel 3D Code for Simulation of Self-gravitating Gas-Dust Systems // 10th International Conference on Parallel Computing Technologies, PaCT 2009, Novosibirsk, Russia, August 31 – September 4, 2009. Proceedings. Vol. 5698 / ed. by V. Malyshkin. Springer, 2009. P. 406–413. Lecture Notes in Computer Science. DOI: 10.1007/978-3-642-03275-2_40.

Киреев Сергей Евгеньевич, лаборатория синтеза параллельных программ, Институт вычислительной математики и математической геофизики СО РАН, кафедра параллельных вычислений, Новосибирский государственный университет (Новосибирск, Российская Федерация)

Литвинов Василий Сергеевич, Новосибирский государственный университет (Новосибирск, Российская Федерация)

DOI: 10.14529/cmse240205

ANALYSIS OF FRAGMENTED PROGRAMS EXECUTION BASED ON SLOW FACTORS

© 2024 S.E. Kireev[1], V.S. Litvinov[2]

[1]*Institute of Computational Mathematics and Mathematical Geophysics SB RAS (pr. Lavrentieva 6, Novosibirsk, 630090 Russia),*

[2]*Novosibirsk State University (str. Pirogova 2, Novosibirsk, 630090 Russia)*

E-mail: kireev@ssd.sscc.ru, v.litvinov@g.nsu.ru

Received: 10.05.2024

When executing parallel programs based on the task parallelism paradigm, several issues need to be addressed, such as choosing the order in which tasks are started, considering the dependencies between them, distributing data and tasks across parallel processes, and balancing resource utilization. These issues fall under the category of system-level parallel programming and are typically handled by a dedicated execution system. The final performance of a parallel program depends on how effectively these issues are addressed, as well as the structure and characteristics of the underlying algorithm. If the program's performance is insufficient, optimization may be required, which necessitates identifying the bottlenecks that hinder its performance. Profiling can be used to pinpoint program bottlenecks by collecting performance metrics that may reveal the source of performance issues. However, the conventional tools commonly used for profiling parallel programs are not able to provide an answer in terms of the required concepts, due to the difficulty in analyzing the asynchronous execution of multiple tasks, as well as the inability to differentiate between application (multiple tasks) and system (operating system) components within an executing program. Consequently, such programs necessitate the development of novel profiling and analysis techniques. The paper discusses the problem of obtaining comprehensible performance characteristics of task-based parallel programs for performance analysis and optimization. It is suggested to evaluate the influence of the following factors: Starvation, Latency, Overhead and Waiting for contention resolution (SLOW). An algorithm for obtaining the corresponding characteristics for the LuNA fragmented programming system is presented, as well as a method for analyzing them to optimize LuNA programs. The correctness of the approach has been demonstrated on a number of synthetic tests. The application of the approach to the analysis of the "real-world" numerical simulation program is shown.

Keywords: performance analysis, parallel programming, fragmented programming, task parallelism, LuNA system.

FOR CITATION

Kireev S.E., Litvinov V.S. Analysis of the Execution of Fragmented Programs Based on SLOW Factors. Bulletin of the South Ural State University. Series: Computational Mathematics and Software Engineering. 2024. Vol. 13, no. 2. P. 77–96. (in Russian) DOI: 10.14529/cmse240205.

References

1. Thoman P., Dichev K., Heller T., *et al.* A taxonomy of task-based parallel programming technologies for high-performance computing. The Journal of Supercomputing. 2018. Vol. 74. P. 1422–1434. DOI: 10.1007/s11227-018-2238-4.
2. Kaiser H., Heller T., Adelstein-Lelbach B., *et al.* HPX: A Task Based Programming Model in a Global Address Space. 8th International Conference on PGAS Programming Models, PGAS'2014, Eugene OR, USA, October 6–10, 2014. Proceedings. Article 6. ACM, 2014. P. 1–11. DOI: 10.1145/2676870.2676883.
3. Malyshkin V.E., Perepelkin V.A. LuNA Fragmented Programming System, Main Functions and Peculiarities of Run-Time Subsystem. 11th International Conference on Parallel Computing Technologies, PaCT-2011, Kazan, Russia, September 19–23, 2011. Proceedings. Vol. 6873 / ed. by V. Malyshkin. Springer, 2011. P. 53–61. Lecture Notes in Computer Science. DOI: 10.1007/978-3-642-23178-0_5.
4. Shende S., Malony A.D. The TAU Parallel Performance System. International Journal of High Performance Computing Applications. 2006. Vol. 20, no. 2. P. 287–311. DOI: 10.1177/1094342006064482.
5. Lorenz D., Feld C. Scaling Score-P to the next level. Procedia Computer Science. 2017. Vol. 108. P. 2180–2189. DOI: 10.1016/j.procs.2017.05.107.
6. Extrae instrumentation package. URL: `https://tools.bsc.es/extrae` (accessed: 23.04.2024).
7. Vampir 10.4. URL: `https://vampir.eu/` (accessed: 23.04.2024).
8. Zhukov I., Feld C., Geimer M., *et al.* Scalasca v2: Back to the Future. 8th International Workshop on Parallel Tools for High Performance Computing, HLRS, Stuttgart, Germany, October, 2014. Proceedings. Ed. by C. Niethammer, J. Gracia, A. Knüpfer, *et al.* Springer, 2015. P. 1–24. DOI: 10.1007/978-3-319-16012-2_1.
9. Mantovani F., Calore E. Multi-Node Advanced Performance and Power Analysis with Paraver. Advances in Parallel Computing. Vol. 32. IOS Press, 2018. P. 723–732. DOI: 10.3233/978-1-61499-843-3-723.
10. Intel Trace Analyzer and Collector. URL: `https://www.intel.com/content/www/us/en/developer/tools/oneapi/trace-analyzer.html` (accessed: 23.04.2024).
11. Huynh A., Thain D., Pericàs M., Taura K. DAGViz: a DAG visualization tool for analyzing task-parallel program traces. 2nd WS on Visual Performance Analysis, VPA '15, November, 2015. Proceedings. No. 3. ACM, 2015. P. 1–8. DOI: 10.1145/2835238.2835241.

12. Huynh A., Taura K. Delay Spotter: A Tool for Spotting Scheduler-Caused Delays in Task Parallel Runtime Systems. 2017 IEEE International Conference on Cluster Computing, CLUSTER, Honolulu, HI, USA, September 5–8, 2017. IEEE, 2017. P. 114–125. DOI: 10.1109/CLUSTER.2017.82.

13. Ceballos G., Grass T., Hugo A., Black-Schaffer D. Analyzing performance variation of task schedulers with TaskInsight. Parallel Computing. 2018. Vol. 75. P. 11–27. DOI: 10.1016/j.parco.2018.02.003.

14. Pinto V.G. Performance Analysis Strategies for Task-based Applications on Hybrid Platforms: PhD thesis / Vinícius Garcia Pinto. Universidade Federal do Rio Grande do Sul - UFRGS, Brazil, UGA - Université Grenoble Alpes, France, 2018. URL: `https://theses.hal.science/tel-01962333`.

15. Pinto V.G., Nesi L.L., Miletto M.C., Schnorr L.M. Providing In-depth Performance Analysis for Heterogeneous Task-based Applications with StarVZ. 2021 IEEE International Parallel and Distributed Processing Symposium Workshops, IPDPSW, Portland, OR, USA, June 17–21, 2021. IEEE, 2021. P. 16–25. DOI: 10.1109/IPDPSW52791.2021.00013.

16. Malyshkin V.E. Fragmented programming technology. Bulletin of the South Ural State University. Computational Mathematics and Software Engineering. 2012. No. 46(305), Iss. 1. P. 45–55. (in Russian) DOI: 10.14529/cmse120104.

17. Bosilca G., Bouteiller A., Danalis A., *et al.* PaRSEC: Exploiting Heterogeneity to Enhance Scalability. Computing in Science & Engineering. 2013. Vol. 15, no. 6. P. 36–45. DOI: 10.1109/MCSE.2013.98.

18. Dokulil J., Benkner S. The OCR-Vx experience: lessons learned from designing and implementing a task-based runtime system. The Journal of Supercomputing. 2022. Vol. 78. P. 12344–12379. DOI: 10.1007/s11227-022-04355-0.

19. Bauer M., Treichler S., Slaughter E., Aiken A. Legion: Expressing locality and independence with logical regions. International Conference on High Performance Computing, Networking, Storage and Analysis, SC'12, Salt Lake City, UT, USA, November 10–16, 2012. Proceedings. IEEE, 2012. P. 1–11. DOI: 10.1109/SC.2012.71.

20. Akhmed-Zaki D., Lebedev D., Perepelkin V. Implementation of a three dimensional three-phase fluid flow ("oil-water-gas") numerical model in LuNA fragmented programming system. The Journal of Supercomputing. 2017. Vol. 73. P. 624–630. DOI: 10.1007/s11227-016-1780-1.

21. Daribayev B., Perepelkin V., Lebedev D., Akhmed-Zaki D. Implementation of the Two-Dimensional Elliptic Equation Model in LuNA Fragmented Programming System. IEEE 12th International Conference on Application of Information and Communication Technologies, AICT 2018, Almaty, Kazakhstan, October 17–19, 2018. Proceedings. IEEE, 2018. P. 161–164. DOI: 10.1109/ICAICT.2018.8747145.

22. Akhmed-Zaki D., Lebedev D., Perepelkin V. Implementation of a 3D model heat equation using fragmented programming technology. The Journal of Supercomputing. 2019. Vol. 75, no. 12. P. 7827–7832. DOI: 10.1007/s11227-018-2710-1.

23. Akhmed-Zaki D., Lebedev D., Malyshkin V., Perepelkin V. Automated Construction of High Performance Distributed Programs in LuNA System. 15th International Conference on Parallel Computing Technologies, PaCT 2019, Almaty, Kazakhstan, August 19–23, 2019.

Proceedings. Vol. 11657 / ed. by V. Malyshkin. Springer, 2019. P. 3–9. Lecture Notes in Computer Science. DOI: 10.1007/978-3-030-25636-4_1.

24. Malyshkin V.E., Perepelkin V.A. A Multi-Agent Approach to Improve Execution Efficiency of Fragmented Programs in LuNA System. Problemy Informatiki. 2023. No. 3. P. 55–67. (in Russian) DOI: 10.24412/2073-0667-2023-3-55-67.

25. Belyaev N., Kireev S. LuNA-ICLU Compiler for Automated Generation of Iterative Fragmented Programs. 15th International Conference on Parallel Computing Technologies, PaCT 2019, Almaty, Kazakhstan, August 19–23, 2019. Proceedings. Vol. 11657 / ed. by V. Malyshkin. Springer, 2019. P. 10–17. Lecture Notes in Computer Science. DOI: 10.1007/978-3-030-25636-4_2.

26. Belyaev N., Perepelkin V. High-Efficiency Specialized Support for Dense Linear Algebra Arithmetic in LuNA System. 16th International Conference on Parallel Computing Technologies, PaCT 2021, Kaliningrad, Russia, September 13–18, 2021. Proceedings. Vol. 12942 / ed. by V. Malyshkin. Springer, 2021. P. 143–150. Lecture Notes in Computer Science. DOI: 10.1007/978-3-030-86359-3_11.

27. Malyshkin V., Perepelkin V. Trace-Based Optimization of Fragmented Programs Execution in LuNA System. 16th International Conference on Parallel Computing Technologies, PaCT 2021, Kaliningrad, Russia, September 13–18, 2021. Proceedings. Vol. 12942 / ed. by V. Malyshkin. Springer, 2021. P. 3–10. Lecture Notes in Computer Science. DOI: 10.1007/978-3-030-86359-3_1.

28. Malyshkin V., Perepelkin V., Lyamin A. Trace Balancing Technique for Trace Playback in LuNA System. 17th International Conference on Parallel Computing Technologies, PaCT 2023, Astana, Kazakhstan, August 21–25, 2023. Proceedings. Vol. 14098 / ed. by V. Malyshkin. Springer, 2023. P. 42–50. Lecture Notes in Computer Science. DOI: 10.1007/978-3-031-41673-6_4.

29. Joint SuperComputer Center of the Russian Academy of Sciences. URL: `https://www.jscc.ru/` (accessed: 23.04.2024) (in Russian).

30. Kireev S. A Parallel 3D Code for Simulation of Self-gravitating Gas-Dust Systems. 10th International Conference on Parallel Computing Technologies, PaCT 2009, Novosibirsk, Russia, August 31 – September 4, 2009. Proceedings. Vol. 5698 / ed. by V. Malyshkin. Springer, 2009. P. 406–413. Lecture Notes in Computer Science. DOI: 10.1007/978-3-642-03275-2_40.