УДК 519.68 doi 10.26089/NumMet.v20r330

ПРАКТИКА ПРОВЕДЕНИЯ АНАЛИЗА ПРОИЗВОДИТЕЛЬНОСТИ СУПЕРКОМПЬЮТЕРНЫХ ЗАДАЧ

И. В. Афанасьев[1], В. В. Воеводин[2], В. Ю. Рудяк[3], А. В. Емельяненко[4]

Предложен метод проведения анализа эффективности и оптимизации суперкомпьютерных приложений, примененный на практике для изучения задач одного из пользователей суперкомпьютера Ломоносов-2. Этот метод затрагивает различные этапы исследования задач, начиная от изучения общего поведения всех запусков пользователя на суперкомпьютере и заканчивая детальным изучением и оптимизацией исходного кода выбранной программы. Приведено описание общих этапов анализа, которые были выполнены на практике, показаны метрики производительности, на которые следует обратить внимание при выполнении подобного анализа, а также продемонстрированы конкретные примеры поведения задач и эффект от оптимизации, выполненной для задачи расчета жидкокристаллических капель.

Ключевые слова: высокопроизводительные вычисления, суперкомпьютеры, анализ эффективности, графические ускорители, жидкие кристаллы, теория эластического континуума.

1. Введение. Как анализировать и оптимизировать производительность суперкомпьютерных приложений? Каждый активный пользователь суперкомпьютерных комплексов задумывается над этим вопросом. Чаще всего найти ответ на него очень непросто. Сложность заключается как в весьма сложной архитектуре современных суперкомпьютеров, для получения высокоэффективного решения на которых необходимо учитывать огромное многообразие различных факторов, так и в том, что современные технологии программирования требуют большого объема знаний для их эффективного применения. Однако пользователями суперкомпьютеров обычно являются эксперты в своих предметных областях: химия, физика, биология и др., у которых нет достаточного опыта для создания эффективных параллельных приложений.

В результате оказывается, что многие суперкомпьютерные приложения обладают низкой эффективностью [1]. Такое положение дел не устраивает всех: пользователей — поскольку расчеты проводятся существенно медленнее, чем могли бы; администраторов и руководство суперкомпьютерных центров — поскольку заметная часть вычислительных ресурсов задействуется очень неэффективно или вообще простаивает. Данная ситуация осложняется тем фактом, что пользователи зачастую либо вообще не знают о наличии проблем с эффективностью в их приложениях, либо не обладают достаточными знаниями (или свободным временем) для проведения анализа эффективности. Администраторы суперкомпьютеров могут помогать пользователям разбираться в этом вопросе, однако они могут исследовать выполняемые на суперкомпьютере задачи только на основе внешних характеристик (обычно для этого используются данные системного мониторинга), не обладая информацией о сути приложения — реализуемом алгоритме, выбранном способе реализации и т.д. Кроме того, администраторы большой системы не могут помочь всем пользователям, поскольку последних значительно больше, чем первых. В частности, на Суперкомпьютерном комплексе МГУ работает около 1000 активных пользователей. Данная ситуация значительно улучшится, если научить пользователей выполнять анализ производительности своих приложений самостоятельно.

В настоящей статье речь идет не только о детальном анализе отдельного приложения, для выполнения которого разработано множество средств анализа параллельных приложений: профилировщики вроде mpiP [2], HPCToolkit; трассировщики вроде Intel VTune [3], Vampir, Scalasca; средства анализа работы с памятью, такие как Valgrind [4] и многие другие. Для того чтобы приступить к детальному анализу приложения, сначала необходимо понять, в какой ситуации и какой именно инструмент анализа необходимо

[1] Московский государственный университет им. М. В. Ломоносова, факультет вычислительной математики и кибернетики, Ленинские горы, 119992, Москва; аспирант, e-mail: afanasiev_ilya@icloud.com

[2] Московский государственный университет им. М. В. Ломоносова, Научно-исследовательский вычислительный центр, Ленинские горы, 119991, Москва; ст. науч. сотр., e-mail: vadim@parallel.ru

[3] Московский государственный университет им. М.В. Ломоносова, физический факультет, Ленинские горы, 119991, Москва; науч. сотр., e-mail: rudyak@polly.phys.msu.ru

[4] Московский государственный университет им. М.В. Ломоносова, физический факультет, Ленинские горы, 119991, Москва; вед. науч. сотр., e-mail: emel@polly.phys.msu.ru

использовать. А для этого нужно проводить анализ общей статистики по всем задачам или по определенному классу задач, выполняемых пользователем на суперкомпьютере. Часть такого анализа можно выполнять автоматически на уровне всего суперкомпьютера, как это делается в рамках системы, разрабатываемой в НИВЦ МГУ [5], однако в большинстве случаев для анализа эффективности приложений пользователя требуется его непосредственное участие.

Предлагаемая статья направлена на то, чтобы показать как пользователям, так и администраторам суперкомпьютерных комплексов, каким образом может быть устроен на практике анализ производительности приложений в рамках одного пользователя. В этой работе описывается опыт, полученный авторами при анализе производительности задач, выполненных на суперкомпьютере Ломоносов-2 [6], и проведении оптимизации одного из классов задач выбранного пользователя. Полученные результаты позволили лучше понять структуру задач и ускорить проведение расчетов.

Дальнейший текст структурирован следующим образом. В разделе 2 приведено краткое описание исследуемых приложений, а также используемых средств и подходов для анализа эффективности. Раздел 3 посвящен анализу общей статистики всех задач пользователя. Раздел 4 содержит описание методов оптимизации, которые были применены к выбранному типу задач. В разделе 5 приведено заключение, а также описаны планы на ближайшее будущее.

2. Описание исследуемых задач и используемых средств анализа производительности. В данной работе был проведен анализ задач одного из пользователей суперкомпьютера Ломоносов-2, запущенных в период с 1 января по 1 июля 2019 года. Все рассматриваемые задачи были разделены на четыре класса: DPD, LAMMPS, LC и остальные (Other). Число и принадлежность задач к классам были установлены вручную самим пользователем. Класс задач DPD — это расчеты равновесных конфигураций полимерных систем методами диссипативной динамики частиц [7]. Эти расчеты проводятся на основе собственной программной реализации, оптимизированной для параллельной работы на большом количестве процессоров без задействования графических ускорителей. Класс задач LAMMPS представляет собой расчеты сильно разбавленных полимерных систем в неявном растворителе в модифицированной программе LAMMPS [8]. Из-за специфики задачи применялась сборка, использующая только x86-процессоры. Класс задач LC объединяет различные задачи по расчетам структур жидкокристаллических капель. Программа для этих расчетов разработана для графических ускорителей NVIDIA и использует специальные численные схемы, оптимизированные для высокопараллельных вычислений. В основе метода лежит расширенная теория эластического континуума Франка и оптимизация системы методом отжига Монте-Карло с критерием Метрополиса [9, 10]. В класс Other попали все не серийно выполняемые задачи, являющиеся нетипичными, поисковыми или просто разовыми.

Для проведения анализа статистики по всем задачам использовались данные системного мониторинга, постоянно работающего на суперкомпьютере, а также данные от менеджера ресурсов Slurm. Для каждой задачи анализировались интегральные значения характеристик производительности, таких как загрузка процессора, число активных процессов на узел (load average), число выполненных операций чтения/записи в оперативную память в секунду, объем переданной/полученной по сети информации в байтах, загрузка GPU-устройств и др. В качестве интегральных значений взяты средние значения по каждой характеристике для каждой задачи, усреднение выполнялось как по времени, так и по вычислительным ядрам.

Для проведения оптимизации отдельного приложения, описанной в разделе 4, использован инструмент `nvprof` [11], который позволяет производить детальный анализ эффективности GPU-ориентированных приложений посредством сбора различных GPU-метрик и событий, характеризующих процесс выполнения программы на графических ускорителях, а также процесса обмена данными между хостом и устройством. При этом непосредственно анализ эффективности, а также поиск узких мест программ на основе собранных метрик и характеристик может осуществляться вручную (как это было выполнено в данной работе), при помощи построения различных формальных моделей оценки эффективности программ [12] или/и при помощи готовых средств профилировки приложений, предлагаемых технологической компанией NVIDIA (Nvidia Visual Profiler, NSight).

3. Анализ статистики по производительности суперкомпьютерных приложений. В нашей работе мы придерживались схемы анализа производительности задач, устроенной по принципу “от общего к частному”. На первом уровне мы изучали статистику на основе данных мониторинга для всех задач пользователя и искали интересные закономерности и особенности на этом самом общем уровне. На втором уровне рассматривалась динамика в рамках отдельного класса задач, позволяющая более детально исследовать общие особенности схожих по структуре задач. На третьем уровне мы проводили тонкий анализ выбранных приложений, для которого обычно уже недостаточно изучения только данных мониторинга

и требуется анализировать исходный код и внутреннее строение программы. На этом уровне практически всегда требуется применение таких средств анализа, как профилировщики, средства сбора и анализа трасс, эмуляторы и др. Следует отметить, что в этой работе второй уровень анализа задач класса LC показал крайне схожее поведение отдельных запусков и не представляет особого интереса, поэтому далее после анализа общей статистики мы перейдем сразу к тонкому анализу отдельного приложения.

Рассмотрим общую статистику по всем рассматриваемым задачам и их классам. Для начала стоит выяснить, какую часть вычислений занимает каждый класс — это позволяет оценить важность их оптимизации. В табл. 1 показано число выполненных задач (запусков программ) каждого класса, а также суммарное затраченное время в узло-часах за рассматриваемый период. Данные в этой таблице были получены на основе анализа данных напрямую от менеджера ресурсов Slurm, однако вся эта информация доступна пользователям по своим задачам в системе Октошелл [13].

Таблица 1

Статистика по числу и объему запусков с 1 января по 1 июля 2019 года

	DPD	LAMMPS	LC	Other
Объем узло-часов	45208	98208	3624	95296
Число запусков	438	417	279	74

Из этой таблицы можно увидеть, что задачи LAMMPS лидируют по объему узло-часов и почти не уступают лидеру по числу запусков — классу DPD. Интересно отметить, что запусков LC достаточно много, однако при этом объем по узло-часам совсем небольшой. Обратная ситуация с задачами Other — таких запусков меньше всего, однако объем занятых узло-часов практически максимален.

Изучив масштабы для каждого из классов задач, перейдем к анализу статистики. Один из важных показателей, который необходимо оценивать, — статус завершения задачи. Он позволяет определять, насколько в целом корректно отрабатывают задачи. Понятно, что анализ только этого аспекта не позволяет получить точную оценку корректности задач, однако для получения общей картины этого зачастую достаточно. На суперкомпьютере Ломоносов-2 различают следующие основные статусы завершения задач:

— Completed (завершилась нормально),

— Cancelled (завершена вручную пользователем),

— Timeout (завершилась по таймауту),

— Failed (завершилась с ошибкой),

— Node_fail (завершилась с ошибкой из-за сбоя вычислительного узла).

В обычной ситуации желательно, чтобы число задач со статусом Completed было максимальным, а число задач со статусами Failed и Node_fail (сюда зачастую относится и Cancelled) — минимальным. На рис. 1 показано распределение объема узло-часов по каждому из классов задач для каждого из статусов завершения. Из рисунка видно, что с этой точки зрения наилучшим образом себя показывает класс LC — объем задач со статусом Completed максимален, а доля задач со статусами Failed и Node_fail минимальна.

Далее видно, что у классов DPD и LAMMPS заметную долю занимают Cancelled-задачи, что нередко указывает на большое число некорректных запусков. Однако для ряда задач, к числу которых относятся и задачи этих классов, статусы Cancelled и Timeout являются нормальными. Так, часто при решении задач перевода систем в равновесное состояние пользователь не знает заранее, сколько времени требуется системе на достижение искомого состояния. Поэтому пользователь проводит постоянный мониторинг задачи, анализирует текущее состояние системы и по ходу расчетов решает, имеет ли смысл продолжать расчеты или их можно остановить (статус Cancelled). Если за установленное пользователем максимальное время расчетов система не достигла равновесия, то задача завершается со статусом Timeout и пользователь обычно устанавливает режим продолжения расчетов. При этом с точки зрения пользователя оба варианта запуска являются продуктивными.

Таким образом, основной вопрос при анализе рис. 1 вызывает существенная доля Failed-задач для класса Other (хотя отметим, что для DPD эта доля тоже достаточно велика). Причем подобные сбои в запусках задач этого класса возникают регулярно, т.е. они не связаны с локальными особенностями программно-аппаратной инфраструктуры или самого приложения и носят глобальный характер. В таком случае пользователю следует детально изучить подобные сбойные запуски и выяснить причину их возникновения. Это и было проделано совместно с администраторами системы. Проведенный детальный анализ позволил выяснить, что существуют две основные причины некорректного поведения:

1) проблемы с файловой системой, которые приводили к “зависанию” задач;

2) тестирование новой функциональности, во время которого появление задач со статусом Failed является в целом нормальной ситуацией.

Таким образом, сбои в запусках возникали не из-за проблем с корректностью или эффективностью в основной версии программы, используемой для проведения реальных расчетов, поэтому дальнейшие действия от пользователя в этой ситуации не требуются.

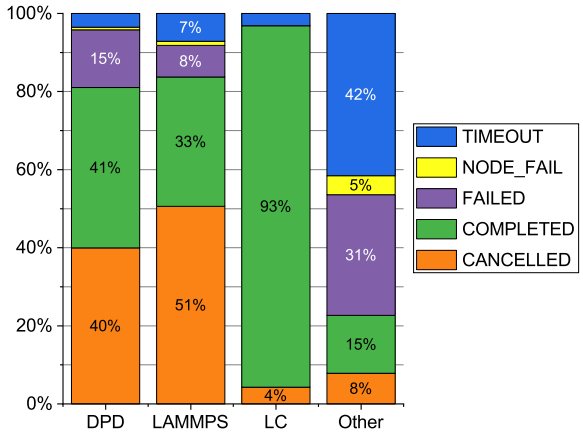

Рис. 1. Распределение доли затраченных узло-часов в зависимости от статуса завершения задач

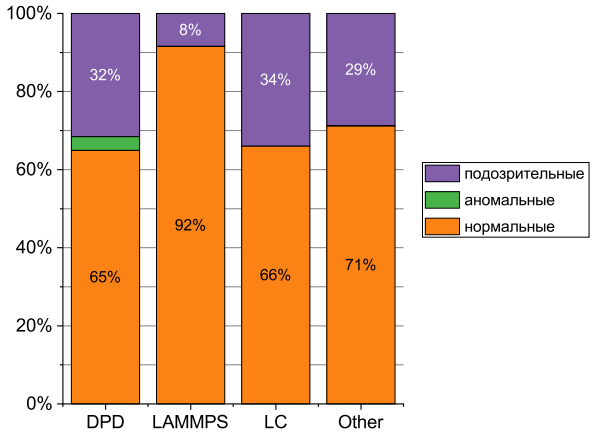

Рис. 2. Распределение доли затраченных узло-часов для аномальных, подозрительных и нормальных задач среди различных классов

Для анализа эффективности приложений могут применяться самые разные подходы. Один из них заключается в изучении “аномальности” приложений. Для этих целей в НИВЦ МГУ была разработана система определения приложений с аномально низкой эффективностью [14], которая постоянно работает на суперкомпьютере Ломоносов-2. Задача считается аномальной, если по чрезвычайно низким характеристикам ее поведения можно автоматически определить, что она работает некорректно и впустую тратит вычислительные ресурсы. Нормальная задача — это задача, эффективность которой не вызывает вопросов. Подозрительными называются задачи промежуточного типа, в которых были обнаружены проблемы с эффективностью, однако нельзя быть точно уверенным, является ли такое поведение аномальным.

Данные об аномальности в настоящий момент собираются для всех задач на суперкомпьютере Ломоносов-2, любой пользователь может посмотреть результаты по своим задачам. Эти данные, а также и множество другой информации об эффективности задач можно увидеть во вкладке “Эффективность” в системе Октошелл. При появлении аномальных запусков в системе Октошелл появляется автоматическое оповещение, предназначенное для того, чтобы пользователь мог оперативно проверить корректность таких запусков и отменить их при необходимости.

На рис. 2 приведено распределение по доле затраченных узло-часов для аномальных, подозрительных и нормальных задач для всех классов задач выбранного пользователя. Система выявления аномалий рассматривает только продолжительные расчеты, представляющие наибольший интерес при анализе эффективности, поэтому здесь не учитывались запуски задач в тестовом разделе, а также длительностью меньше одного часа. Если данные на рис. 1 позволяют нам примерно оценить время, потраченное на корректные и некорректные запуски без учета эффективности их выполнения, то здесь мы можем лучше понять качество задач уже с точки зрения их эффективности. Можно заметить, в частности, что для задач класса LC доля подозрительных задач достаточно высока, т.е. в данном классе достаточно часто встречаются задачи с потенциальными проблемами с эффективностью. Это значит, что задачи этого класса далеко не всегда эффективны, хотя они почти всегда работают корректно (согласно рис. 1). Далее, с точки зрения анализа аномалий почти нет вопросов к задачам класса LAMMPS — доля нормальных задач максимальна, доля аномальных задач очень мала.

Хуже всего показатели для класса DPD, поскольку при немалой доле подозрительных задач здесь также заметна доля аномальных запусков, которые однозначно некорректны и тратят вычислительные ресурсы впустую. В отличие от описанного выше случая с Failed-задачами из класса Other, сбой в аномальных задачах класса DPD, судя по всему, не был вызван проблемами с файловой системой. Для дальнейшего выяснения требуется детальное рассмотрение причин такого поведения совместно пользователем и администраторами системы.

Интересно отметить, что, несмотря на заметную долю задач со статусом Failed, доля аномальных задач в классе Other невелика. Эта закономерность объясняется тем, что зачастую сбои происходят быстро, т.е. до момента сбоя задача считается нормально, и только в последний момент, когда происходит сбой, показатели производительности могут резко уменьшиться, после чего задача аварийно завершается. В этом случае, с точки зрения внешнего наблюдателя, эффективность выполнения такой задачи в целом выглядит нормальной, поскольку сбойный фрагмент очень мал.

До этого момента мы рассматривали данные о статусах завершения и аномальности задач. Однако полноценный анализ эффективности выполнения задач также требует изучения отдельных характеристик производительности, полученных от системного мониторинга. В общем случае имеет смысл рассматривать наиболее важные характеристики, описывающие производительность суперкомпьютера при выполнении приложения — загрузка процессора, load average, число операций обращения в память (чтения или записи) в секунду, объем переданных и полученных байт в секунду по сети MPI, загрузка графических устройств. Для каждой задачи мы рассматривали средние значения по всем этим характеристикам, агрегированные по времени выполнения задачи и задействованным при ее выполнении вычислительным ядрам.

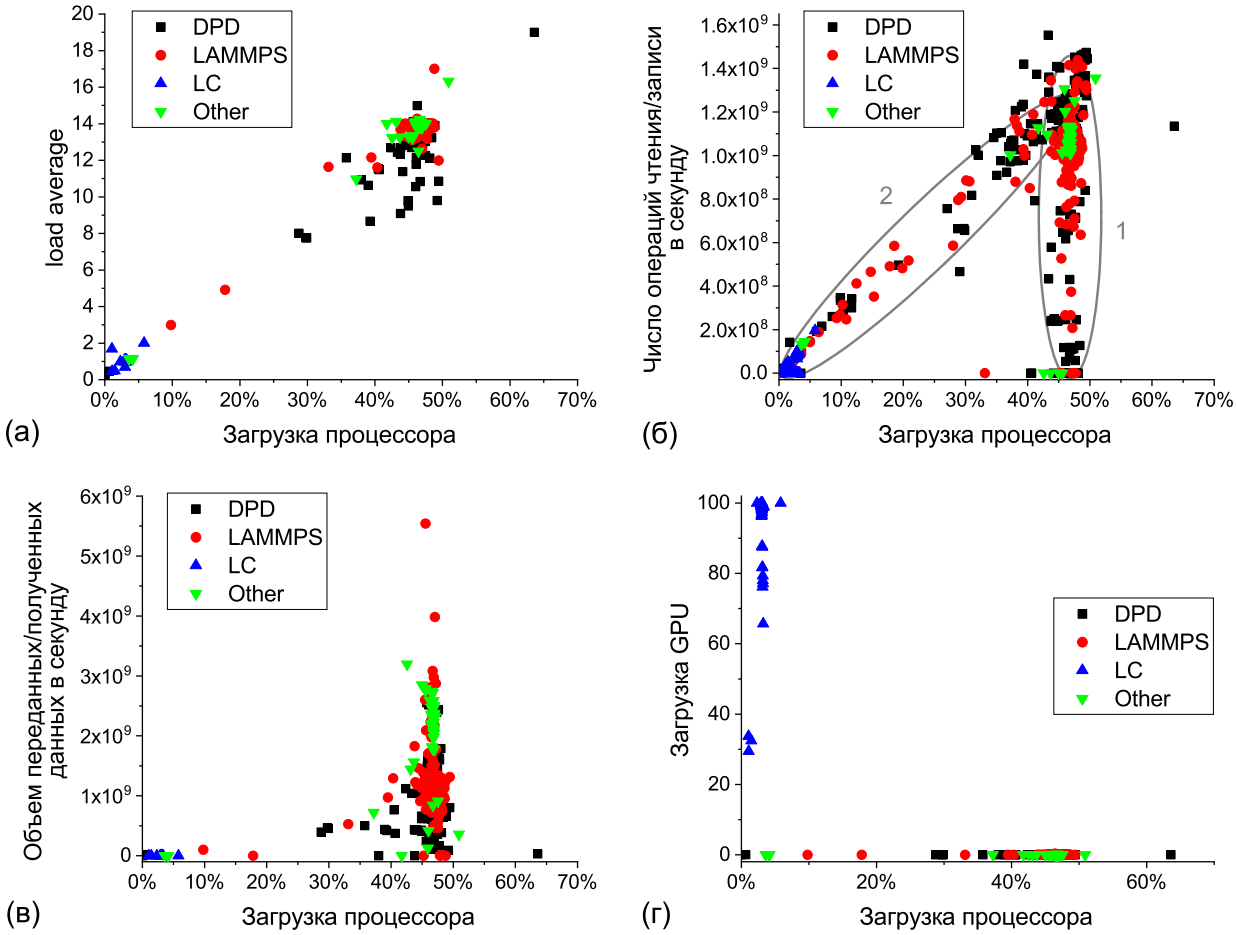

Рис. 3. Соотношение средних значений различных характеристик производительности для четырех классов задач выбранного пользователя

Графики для каждой пары вида “загрузка процессора–другая характеристика” приведены на рис. 3. Каждая точка — это одна задача (один запуск программы). Задачи разных классов отмечены разными символами: черные квадраты — DPD, красные круги — LAMMPS, синие треугольники — LC, зеленые перевернутые треугольники — Other. Отметим, что данные по load average, интенсивности работы с сетью и загрузке GPU были доступны не для всех задач, поэтому число точек на рис. 3б несколько больше, чем на остальных.

Первое, что можно отметить, — явно виден характер задач класса LC. Видно, что это яркие представители задач для GPU — они сильно загружают графические устройства (рис. 3г), при этом обычные процессоры практически простаивают (загрузка процессора $\sim$ 3–5%). Из рисунка видно, что в этих

задачах load average составляет около 1–2, т.е. используется 1–2 процесса на узел, видимо только для управления работой графических устройств. Кроме того, интересно, что данные задачи достаточно часто запускаются на нескольких узлах, однако при этом активность использования коммуникационной сети (рис. 3в) практически равна нулю. Это означает, что вычисления на разных узлах практически независимы и не требуют обмена данными. Все эти выводы, сделанные на основе только показаний мониторинга, были затем подтверждены при детальном изучении внутренней структуры программы.

Далее рассмотрим остальные три класса задач. Видно, что все они вообще не задействуют GPU, т.е. работают только с основными x86-процессорами. Эффективность утилизации этих процессоров высокая — согласно рис. 3а, чаще всего в этих задачах присутствуют 14 активных процессов на узел, что является оптимальным значением для разделов compute и test суперкомпьютера Ломоносов-2 (поскольку равно числу физических ядер), где и были запущены эти задачи. Средняя загрузка процессора при этом — около 50%, что соответствует оптимальному значению в случаях без использования технологии HyperThreading (при которой число логических процессорных ядер удваивается). Применение технологии HyperThreading должно быть явным образом инициировано на суперкомпьютере Ломоносов-2, и в данных случаях она не применялась.

Интересно, что в задачах классов DPD и LAMMPS при нормальной загрузке процессора ($\sim 50\%$) интенсивность работы с памятью может очень сильно отличаться (область 1 на рис. 3б). Однако если загрузка процессора снижается, то прослеживается явная закономерность — чем ниже загрузка, тем ниже интенсивность (область 2). Такое поведение пока не удалось объяснить, поэтому требуется более детальное изучение этой особенности совместно с автором программы.

В случае задач класса Other ситуация другая — интенсивность составляет $\sim 10^9$ обращений в память в секунду и лишь изредка снижается до небольших значений меньше 10^8 операций в секунду. Скорее всего, такое снижение соответствует некорректной или тестовой работе приложения.

Все три класса не GPU-задач (DPD, LAMMPS и Other) демонстрируют высокую активность использования коммуникационной сети. Как видно из рис. 3в, в большинстве случаев среднее число переданных и полученных байт в секунду равно 1 ГБ/с., что является высоким показателем для задач, выполняемых на суперкомпьютере Ломоносов-2. При этом дальнейший анализ показывает, что значение этой характеристики никак не коррелирует с числом параллельных процессов в задаче. Такое поведение связано с распараллеливанием программ из этих классов по принципу spatial decomposition, когда весь объем моделирования разбивается на части и каждый поток обрабатывает данные своей части. Обмен данными между потоками определяется в первую очередь интенсивностью обмена частицами между частями объема. В проанализированных задачах обычно использовалось такое число узлов, чтобы сохранить объем, приходящийся на каждый узел, примерно одинаковым, поэтому и интенсивность обмена данными между узлами не изменялась от задачи к задаче.

В целом же можно отметить, что задачи данного пользователя достаточно интенсивно задействуют предоставляемые вычислительные ресурсы с точки зрения рассмотренных основных характеристик производительности.

4. Анализ и оптимизация задачи расчета структуры жидкокристаллических капель. Для проведения более детального анализа производительности было решено рассмотреть задачи из класса LC. Как уже было указано ранее, эти задачи очень активно используют графические ускорители и практически не задействуют x86-процессоры. Анализ производительности использования графических процессоров был произведен с использованием средства `nvprof` (см. раздел 2).

Необходимо отметить, что свойства задач анализируемого класса могут сильно меняться в зависимости от входных параметров (что, в общем-то, верно для большинства вычислительных задач). В частности, сильно влияет соотношение размера сетки для расчетов внутри капли и степени детализации поверхности капли. В нашей работе рассматривался вариант задачи с высокой степенью детализации поверхности.

Для GPU-задач анализ производительности традиционно начинается с выбора одного из двух принципиально отличных направлений для оптимизации:

1) в случае, если программа интенсивно использует шину передачи данных между хостом и устройством (PCI или NVLINK), а время выполнения GPU-ядер относительно мало, то необходимо оптимизировать программу посредством уменьшения объема пересылаемых через шину данных либо посредством одновременного выполнения процесса копирований данных и вычислений;

2) в случае, если время, затрачиваемое на копирование данных, относительно мало, то обычно необходимо производить итеративный процесс оптимизации наиболее вычислительно затратных CUDA-ядер (данное утверждение верно в случае, если задача не использует параллельные вычисления на хосте и GPU-устройстве).

Для определения направления для оптимизации мы использовали следующий вызов средства профилировки nvprof:

```
nvprof --print-summary-per-gpu ./<имя_приложения> <параметры>.
```

Результаты его выполнения приведены в табл. 2 (показаны результаты для трех наиболее вычислительно затратных функций). Из этой таблицы явно следует, что наиболее вычислительно затратной частью программы являются не пересылки данных, а ядро `stepkernel_surface_sum`, занимающее более 90% времени работы программы. Следует явно подчеркнуть, что подобная картина возникает при рассмотрении указанного выше варианта задачи с высокой степенью детализации поверхности; в других вариантах влияние данной функции может быть существенно меньше.

Таблица 2

Результаты анализа времени выполнения различных действий на графическом ускорителе в ходе выполнения задачи

	Время выполнения, с	Время выполнения, %
Копирование данных с устройства на хост	0.028	0.02%
Копирование данных с хоста на устройство	0.005	0.004%
Функция stepkernel_surface_sum()	120.72	94%
Функция stepkernel_dds()	1.81	1.4%
Функция stepkernel_elastics()	1.72	1.3%

Анализ эффективности этого ядра может быть проведен посредством другого вызова средства профилировки nvprof, позволяющего оценить, насколько эффективно ядром stepkernel_surface_sum используется подсистема памяти, а также вычислительные ресурсы GPU.

Данный вызов имеет вид:

```
nvprof --kernels stepkernel_surface_sum --metrics <список метрик> ./<имя_приложения>
 <параметры>
```

Наибольший интерес представляют следующие метрики:

— l2_utilization — уровень использования пропускной способности кэш-памяти L2 относительно пиковой пропускной способности;

— shared_utilization — уровень использования пропускной способности разделяемой памяти;

— tex_utilization — уровень использования пропускной способности текстурной кэш-памяти;

— dram_utilization — уровень использования пропускной способности текстурной кэш-памяти;

— special_fu_utilization — уровень использования функциональных модулей, выполняющих инструкции sin, cos, ex2, popc, flo и другие;

— half_precision_fu_utilization — уровень использования функциональных блоков мультипроцессора, выполняющих вычисления над 16-битными данными с плавающей точкой;

— single_precision_fu_utilization — уровень использования функциональных блоков мультипроцессора, которые выполняют инструкции с плавающей запятой одинарной точности и целочисленные инструкции;

— double_precision_fu_utilization — уровень использования функциональных блоков мультипроцессора, которые выполняют инструкции с плавающей запятой двойной точности;

— achieved_occupancy — отношение среднего количества активных варпов на каждом цикле к максимальному числу варпов, выполнение которых возможно на GPU.

Каждое из запрошенных значений GPU-метрик характеризует интенсивность использования различных видов ресурсов графического устройства. Для всех характеристик, кроме последней, используется шкала от 0 до 10; последняя характеристика может изменяться в диапазоне 0–100%. Для исследуемого вычислительного ядра каждое из значений находится в интервале от 0 до 1, что свидетельствует

о крайне неэффективном использовании ресурсов GPU. Это позволяет отнести данное ядро к классу latency-bound — классу программ, производительность которых ограничена латентностью при получении данных, в отличие от compute-bound (ограниченных производительностью вычислительных устройств) или memory-bound программ (ограниченных пропускной способностью некоторого уровня иерархии памяти).

Зачастую проблема слабого использования ресурсов GPU может быть вызвана недостаточным использованием ресурса параллелизма в программе, к примеру из-за использования недостаточного числа CUDA-нитей. Для анализа этого аспекта работы программы стоит анализировать метрику achieved_occupancy. Данная метрика отвечает за количество активных варпов на каждом из потоковых мультипроцессоров GPU и для данной программы имеет крайне низкое значение — ~ 0.016.

Таблица 3

Изменение времени работы ядра stepkernel_surface_sum, а также его основных метрик профилировки до и после проведения оптимизации

	До оптимизации	После оптимизации
Время выполнения, с	120.72	0.011
Время выполнения, % от общего	94%	0.3%
Метрика achieved_occupancy	0.015625	0.67
Метрика dram_utilization	1	5

Более подробный анализ программного кода данного ядра подтверждает гипотезу, что это ядро запускается на небольшом количестве нитей. Более того, нити данного ядра выполняют достаточно большое число итераций цикла, зависимых по данным из-за возможности записи значений в одни и те же ячейки массивов на различных итерациях. Устранение подобных зависимостей при записях в память для графических архитектур Kepler и выше может быть произведено посредством использования атомарных операций, которые позволяют произвести распараллеливание зависимых итераций цикла с использованием дополнительных нитей без угрозы гонки за данными. Результаты проведения такой оптимизации с точки зрения наиболее важных метрик, а также времени выполнения данного ядра приведены в табл. 3. Исходя из данных, приведенных в таблице, можно сделать вывод о том, что данное ядро из класса latency-bound перешло в класс memory-bound, причем эффективность использования пропускной способности памяти для данного ядра достаточно высока, несмотря на использование атомарных операций.

Таким образом, проведенная оптимизация позволила уменьшить время выполнения всей программы примерно в 35 раз: с 128.4 секунд до 3.6 секунд.

5. Заключение и планы на будущее. В настоящей статье показан пример проведения анализа и оптимизации производительности суперкомпьютерных задач, начиная от изучения статистики по всем запускам выбранного пользователя и заканчивая тонким анализом отдельного приложения, который позволил существенно повысить скорость выполнения расчетов. Подобный анализ на основе продемонстрированных примеров может быть использован как администраторами, так и пользователями суперкомпьютерной системы для изучения множества других приложений.

В будущем планируется провести дальнейшее изучение статистики по другим классам задач. Предполагается также выполнить тонкий анализ других особенностей задачи расчета жидкокристаллических капель, оптимизация которых позволит еще больше повысить скорость выполнения расчетов.

Результаты, описанные в данной статье, были получены при финансовой поддержке гранта Президента РФ (МК-2330.2019.9). Работа выполнена с использованием оборудования Центра коллективного пользования сверхвысокопроизводительными вычислительными ресурсами МГУ имени М. В. Ломоносова.

СПИСОК ЛИТЕРАТУРЫ

1. *Voevodin V., Voevodin V.* Efficiency of exascale supercomputer centers and supercomputing education // High Performance Computer Applications. Vol. 595. Cham: Springer, 2016. 14–23.
2. *Vetter J., Chambreau C.* MpiP: Lightweight, scalable MPI profiling. http://mpip.sourceforge.net.
3. Intel VTune Amplifier documentation. https://software.intel.com/en-us/vtune.
4. *Nethercote N., Seward J.* Valgrind: a Framework for heavyweight dynamic binary instrumentation // SIGPLAN Not. 2007. **42**, N 6. 89–100.

5. *Shvets P., Voevodin Vad., Zhumatiy S.* HPC software for massive analysis of the parallel efficiency of applications // Communications in Computer and Information Science. Vol. 1063 Cham: Springer, 2019. 3–18.
6. *Voevodin Vl.V., Antonov A.S., Nikitenko D.A., Shvets P.A., Sobolev S.I., Sidorov I.Yu., Stefanov K.S., Voevodin V.V., Zhumatiy S.A.* Supercomputer Lomonosov-2: large scale, deep monitoring and fine analytics for the user community // Supercomputing Frontiers and Innovations. 2019. **6**, N 2. 4–11.
7. *Groot R.D., Warren P.B.* Dissipative particle dynamics: bridging the gap between atomistic and mesoscopic simulation // The Journal of Chemical Physics. 1997. **107**, N 11. 4423–4435.
8. *Plimpton S.* Fast parallel algorithms for short-range molecular dynamics // J. Comput. Phys. 1995. **117**, N 1. 1–19.
9. *Frank F.C.* I. Liquid crystals. On the theory of liquid crystals // Discuss. Faraday Soc. 1958. **25**. 19–28.
10. *Rudyak V.Y., Emelyanenko A.V., Loiko V.A.* Structure transitions in oblate nematic droplets // Physical Review E. 2013. Vol. 88, N 5. doi 10.1103/PhysRevE.88.052501.
11. Profiler User's Guide. https://docs.nvidia.com/cuda/profiler-users-guide/index.html#nvprof-overview.
12. *Yang C., Williams S.* Performance analysis of GPU-accelerated applications using the roofline model. https://developer.nvidia.com/gtc/2019/video/S9624.
13. *Никитенко Д.А., Воеводин В.В., Жуматий С.А.* Octoshell: система для администрирования больших суперкомпьютерных комплексов // Вестн. ЮУрГУ. Сер. Выч. матем. информ. 2016. **5**, № 3. 76–95.
14. *Shaykhislamov D., Voevodin V.* An approach for dynamic detection of inefficient supercomputer applications // Procedia Computer Science. 2018. **136**. 35–43.

Поступила в редакцию
27.08.2019

The Practice of Conducting Performance Analysis of Supercomputer Applications

I. V. Afanasyev [1], V. V. Voevodin [2], V. Yu. Rudyak [3], and A. V. Emelyanenko [4]

[1] *Lomonosov Moscow State University, Faculty of Computational Mathematics and Cybernetics; Leninskie Gory, Moscow, 119992, Russia; Graduate Student, e-mail: afanasiev_ilya@icloud.com*

[2] *Lomonosov Moscow State University, Research Computing Center; Leninskie Gory, Moscow, 119991, Russia; Ph.D., Senior Scientist, e-mail: vadim@parallel.ru*

[3] *Lomonosov Moscow State University, Faculty of Physics; Leninskie Gory, Moscow, 119991, Russia; Ph.D., Scientist, e-mail: rudyak@polly.phys.msu.ru*

[4] *Lomonosov Moscow State University, Faculty of Physics; Leninskie Gory, Moscow, 119991, Russia; Dr. Sci., Leading Scientist, e-mail: emel@polly.phys.msu.ru*

Received August 27, 2019

Abstract: A method for the efficiency analysis and optimization of supercomputer applications applied earlier in practice to study jobs of a user on the Lomonosov-2 supercomputer is proposed. This method involves various stages of the jobs research, starting from studying the general behavior of all user launches on a supercomputer and ending with a detailed study and optimization of the source code of a selected program. The paper describes the general stages of the analysis that were carried out in practice, shows performance metrics that should be paid attention to when performing such an analysis, and shows also some specific examples of the job behavior and the effect of optimization carried out for the task of calculating liquid crystal droplets.

Keywords: high-performance computing, supercomputers, efficiency analysis, graphics accelerators, liquid crystals, elastic continuum theory.

References

1. V. Voevodin and V. Voevodin, "Efficiency of Exascale Supercomputer Centers and Supercomputing Education," in *High Performance Computer Applications* (Springer, Cham, 2016), Vol. 595, pp. 14–23.

2. J. Vetter and C. Chambreau, "MpiP: Lightweight, Scalable MPI Profiling," http://mpip.sourceforge.net. Cited August 27, 2019.

3. Intel VTune Amplifier documentation. https://software.intel.com/en-us/vtune. Cited August 27, 2019.

4. N. Nethercote and J. Seward, "Valgrind: A Framework for Heavyweight Dynamic Binary Instrumentation," SIGPLAN Not. **42** (6), 89–100 (2007).

5. P. Shvets, Vad. Voevodin, and S. Zhumatiy, "HPC Software for Massive Analysis of the Parallel Efficiency of Applications," in *Communications in Computer and Information Science* (Springer, Cham, 2019), Vol. 1063, pp. 3–18.

6. Vl. V. Voevodin, A. S. Antonov, D. A. Nikitenko, et al., "Supercomputer Lomonosov-2: Large Scale, Deep Monitoring and Fine Analytics for the User Community," Supercomput. Front. Innov. **6** (2), 4–11 (2019).

7. R. D. Groot and P. B. Warren, "Dissipative Particle Dynamics: Bridging the Gap between Atomistic and Mesoscopic Simulation," J. Chem. Phys. **107** (11), 4423–4435 (1997).

8. S. Plimpton, "Fast Parallel Algorithms for Short-Range Molecular Dynamics," J. Comput. Phys. **117** (1), 1–19 (1995).

9. F. C. Frank, "I. Liquid Crystals. On the Theory of Liquid Crystals," Discuss. Faraday Soc. **25**, 19–28 (1958).

10. V. Y. Rudyak, A. V. Emelyanenko, and V. A. Loiko, "Structure Transitions in Oblate Nematic Droplets," Phys. Rev. E **88** (2013). doi 10.1103/PhysRevE.88.052501

11. Profiler User's Guide. https://docs.nvidia.com/cuda/profiler-users-guide/index.html#nvprof-overview. Cited August 27, 2019.

12. C. Yang and S. Williams, "Performance Analysis of GPU-Accelerated Applications Using the Roofline Model," https://developer.nvidia.com/gtc/2019/video/S9624. Cited August 27, 2019.

13. D. A. Nikitenko, V. V. Voevodin, and S. A. Zhumatiy, "Octoshell: Large Supercomputer Complex Administration System," Vestn. Yuzhn. Ural. Gos. Univ. Ser. Vychisl. Mat. Inf. **5** (3), 76–95 (2016).

14. D. Shaykhislamov and V. Voevodin, "An Approach for Dynamic Detection of Inefficient Supercomputer Applications," Procedia Comput. Sci. **136**, 35–43 (2018).