**Devastator: масштабируемая параллельная дискретно-событийная среда моделирования для современного C++**

[**Джон Бачан**](https://orcid.org/0000-0002-4342-4909) , Национальная лаборатория Лоуренса в Беркли, США,[**john.bachan@gmail.com**](mailto:john.bachan@gmail.com)

[**Цзяньлан Е**](https://orcid.org/0009-0003-5093-9040) , Университет штата Аризона, США,[**jianlany@gmail.com**](mailto:jianlany@gmail.com)

[**Сюань Цзян**](https://orcid.org/0000-0001-6489-2018) , Калифорнийский университет, Беркли, США,[**j503440616@berkeley.edu**](mailto:j503440616@berkeley.edu)

[**Тан Нгуен**](https://orcid.org/0000-0003-3748-403X) , компьютерные науки, Национальная лаборатория Лоуренса в Беркли, США,[**tannguyen@lbl.gov**](mailto:tannguyen@lbl.gov)

[**Махеш Натараджан**](https://orcid.org/0000-0003-0049-1981) , Национальная лаборатория Лоуренса в Беркли, США,[**mn1729@gmail.com**](mailto:mn1729@gmail.com)

[**Максимилиан Бремер**](https://orcid.org/0000-0002-5940-3432) , Национальная лаборатория Лоуренса в Беркли, США,[**bremerm31@gmail.com**](mailto:bremerm31@gmail.com)

[**Сай Чан**](https://orcid.org/0000-0001-6881-827X) , Национальная лаборатория Лоуренса в Беркли, США,[**cychan@gmail.com**](mailto:cychan@gmail.com)

**DOI:**[**https://doi.org/10.1145/3615979.3656061**](https://doi.org/10.1145/3615979.3656061) **SIGSIM PADS '24:**[**38-я конференция ACM SIGSIM по принципам усовершенствованного дискретного моделирования**](https://doi.org/10.1145/3615979)**, Атланта, Джорджия, США, июнь 2024 г.**

Параллельное дискретно-событийное моделирование — это фундаментальная технология моделирования, которая необходима для распараллеливания событийных моделей, включая аппаратные и транспортные системы. Распараллеливание часто затруднено из-за динамических зависимостей данных и ограниченной вычислительной работы для сокрытия накладных расходов времени выполнения. Мы представляем Devastator, масштабируемую параллельную дискретно-событийную инфраструктуру для современного C++. Devastator предоставляет производительный API, который использует систему типов C++ для устранения шаблонного кода. Devastator был разработан специально для оптимизации производительности на распределенных многоядерных архитектурах с углубляющимися иерархиями памяти. Devastator использует среду выполнения связи GASNet-Ex, а также очереди сообщений без блокировок для достижения высококонкурентной производительности. Мы проводим сильные и слабые исследования масштабирования на Perlmutter NERSC до 32 698 ядер и демонстрируем ускорение до 5 раз по сравнению с симулятором ROSS для рабочих нагрузок со значительной локальностью.

**Концепции CCS:** • **Методологии вычислений → Дискретно-событийное моделирование** ; *Методологии моделирования;* • **Прикладные вычисления → Транспорт** ;

**Ключевые слова:**Высокопроизводительные вычисления , параллельное дискретно-событийное моделирование , Time Warp , активные сообщения , GASNet

**Формат справочника ACM:**  
Джон Бачан, Цзяньлан Йе, Сюань Цзян, Тан Нгуен, Махеш Натараджан, Максимилиан Бремер и Сай Чан. 2024. Devastator: масштабируемая параллельная среда дискретно-событийного моделирования для современного C++. На *38-й конференции ACM SIGSIM по принципам расширенного дискретного моделирования (SIGSIM PADS '24), 24–26 июня 2024 г., Атланта, Джорджия, США.* ACM, Нью-Йорк, Нью-Йорк, США 12 страниц. [**https://doi.org/10.1145/3615979.3656061**](https://doi.org/10.1145/3615979.3656061)

1 ВВЕДЕНИЕ

Дискретно-событийное моделирование (DES) — это класс симуляций, который представляет модель как ряд событий с метками времени, которые происходят на предопределенных акторах, и может планировать последующие события на основе локальной информации об акторах. DES — это фундаментальная технология моделирования, играющая ключевую роль в ускорении моделирования транспортных средств [ [**8**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#BibPLXBIB0008) , [**20**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#BibPLXBIB0020) , [**26**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#BibPLXBIB0026) , [**35**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#BibPLXBIB0035) ], аппаратного моделирования [ [**11**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#BibPLXBIB0011) , [**28**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#BibPLXBIB0028) , [**31**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#BibPLXBIB0031) , [**41**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#BibPLXBIB0041) , [**43**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#BibPLXBIB0043) ], а также многих других областей [ [**38**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#BibPLXBIB0038) , [**39**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#BibPLXBIB0039) , [**44**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#BibPLXBIB0044) ].

Ключевой проблемой для дискретно-событийных симуляций является то, как их распараллелить. Контракт на проектирование параллельного дискретно-событийного движка моделирования (PDES) обещает, что к концу симуляции локальные изменения состояния из-за выполнения событий будут идентичны изменениям последовательной модели. Поскольку планирование событий зависит от локального состояния симуляции, а прогнозирование прибытия событий на актор, как правило, невозможно, распараллеливания обычно делятся на две основные категории. *Консервативные* подходы используют специфичную для модели информацию, чтобы гарантировать, что никакие события не могут поступить в течение заданного интервала времени, указывая на то, что работа в течение интервала времени безопасна для выполнения. *Оптимистическая* синхронизация не требует, чтобы все события были выполнены в правильном порядке, а требует только, чтобы события, которые выполняются в неправильном порядке, воспроизводились таким образом, чтобы конечное состояние актора соответствовало правильно выполненным событиям.

Сложно делать обобщения о компромиссах производительности между двумя схемами распараллеливания [ [**18**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#BibPLXBIB0018) ]. Консервативные подходы обычно раскрывают меньший параллелизм для симулятора и требуют детального знания проблемы, чтобы раскрыть достаточный параллелизм для получения хорошей производительности. Оптимистичные подходы, с другой стороны, страдают от осложнений, когда слишком много событий были выполнены неправильно, что приводит к значительным накладным расходам времени выполнения. Кроме того, требование, чтобы событие было «обратимым», часто сопровождается значительными накладными расходами на кодирование, даже при автоматизированных подходах [ [**33**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#BibPLXBIB0033) , [**46**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#BibPLXBIB0046) , [**51**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#BibPLXBIB0051) ], что исключает возможность соединения сторонних библиотек без существенных инженерных усилий.

PDES — это зрелая технология моделирования, многие из основополагающих статей которой были написаны в 1980-х и 1990-х годах. Однако большая часть работы того времени сталкивалась с другой архитектурной реальностью. Многие старые статьи фокусируются на том, чтобы не превышать ограничения емкости памяти, и фокусируются на стоимости вычислений. На современных суперкомпьютерах критические ограничения не могут быть более разными. Хотя FLOP стали легкодоступными, время моделирования теперь ограничено стоимостью синхронизации, доступом к памяти через углубляющиеся иерархии памяти и задержкой распределенной связи. Многие приложения PDES ограничены либо задержкой памяти, либо пропускной способностью.

В этом духе в данной статье представлена ​​современная высокопроизводительная реализация протокола TimeWarp Джефферсона [ [**22**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#BibPLXBIB0022) ]. Мы представляем гибридную структуру параллелизма, в которой сообщения общей памяти отправляются с использованием очередей сообщений без блокировки. Межпроцессные и междоменные сообщения NUMA обрабатываются с использованием среды выполнения GASNet-EX [ [**5**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#BibPLXBIB0005) ], которая обеспечивает низкоуровневую поддержку для различных взаимосвязей.

Другим ключевым препятствием для принятия остается программируемость и удобство использования симуляторов PDES. Многие симуляторы, такие как ROSS [ [**7**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#BibPLXBIB0007) ] и Root-SIM [ [**34**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#BibPLXBIB0034) ], написаны на C и требуют подробных спецификаций актеров и событий. NS-3 [ [**31**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#BibPLXBIB0031) ] является консервативной структурой и требует от пользователей реализации опережающей оптимизации для большего параллелизма. Другие движки PDES, такие как Simian [ [**45**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#BibPLXBIB0045) ], написаны на Python и обеспечивают существенно более низкую пропускную способность. Мы стремимся преодолеть этот разрыв путем внедрения современного API C++. Используя активные сообщения, современные распределители памяти и эффективные процедуры сериализации, пользователи могут эффективно реализовывать свои дискретные модели событий, при этом достигая производительности, которая будет конкурентоспособной с симуляторами, такими как ROSS. Дифференциация пакетов программного обеспечения с помощью языков программирования во многом является вопросом личных предпочтений, но мы считаем, что внедрение экономной библиотеки C++ PDES с производительностью и масштабируемостью, сопоставимыми с ROSS, делает наше программное решение ценным вкладом в сообщество PDES. Основные выводы данной статьи следующие.

1. Мы представляем Devastator, высокопроизводительный параллельный дискретно-событийный симулятор на основе C++ оптимистичного Time Warp, разработанный для распределенных многоядерных суперкомпьютеров. Devastator поставляется с многочисленными расширенными функциями, включая адаптивное окно просмотра вперед, очереди сообщений без блокировки и распределитель памяти, а также другие оптимизации, учитывающие кэш/память. Для лучшего масштабирования в больших системах Devastator использует среду выполнения связи GASNet-EX, которая может ускорить перемещение данных и облегчить миграцию объектов с помощью активных сообщений.
2. Мы разрабатываем API, который использует современную функциональность C++ для указания событий и внедрения эвристики для автоматической настройки оптимизма симулятора. Результатом является модель программирования, которую легче разрабатывать и которая требует настройки меньшего количества параметров для эффективной работы.
3. Мы демонстрируем производительность Devastator, демонстрируя масштабируемость серии тестов PHOLD [ [**19**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#BibPLXBIB0019) ] с различными параметризациями. Мы демонстрируем масштабируемость симулятора на Perlmutter от NERSC, достигая 30-80% пропускной способности одного узла на ядро ​​при 32 698 ядрах, и сравниваем производительность с симулятором ROSS, для которого мы достигаем ускорения до 5 раз для задач со значительной локальностью.
4. Мы также оцениваем Devastator с помощью моделирования транспортной сети. Хотя этот эксперимент проводится в небольшом масштабе, результаты демонстрируют преимущество адаптивного окна просмотра вперед на динамических, быстро меняющихся входных данных.

Оставшаяся часть статьи продолжается следующим образом. Мы описываем важные алгоритмические особенности и API в разделе  [**2.**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#sec-3) В разделе  [**3**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#sec-8) мы описываем среду выполнения связи. В разделе  [**4**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#sec-12) мы представляем производительность Devastator и сравниваем ее с ROSS, а в разделе  [**5**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#sec-18) мы моделируем трафик в столичном районе Лос-Анджелеса и анализируем производительность. Наконец, мы представляем связанную работу в разделе  [**6**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#sec-21) и завершаем статью в разделе  [**7**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#sec-22) .

2 ПАРАЛЛЕЛЬНОЕ ДИСКРЕТНОЕ МОДЕЛИРОВАНИЕ СОБЫТИЙ

Моделирование дискретных событий (DES) — это вычислительная парадигма для моделирования системы, развивающейся во времени, которая движется вперед за счет мгновенных изменений состояния в предписанные временные метки, часто называемые *событиями* [ [**9**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#BibPLXBIB0009) , [**21**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#BibPLXBIB0021) , [**24**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#BibPLXBIB0024) ]. Помимо изменения состояния, каждое событие может также регистрировать новые события для выполнения в будущие временные метки. Приложение определяется его пространством состояний, начальным состоянием и начальным набором событий. Последовательная среда выполнения фреймворка DES отвечает за выполнение событий в порядке временных меток до тех пор, пока не исчезнут дальнейшие события, после чего моделирование завершается. Параллельное моделирование дискретных событий (PDES) — это попытка извлечь параллелизм из этой парадигмы. В PDES приложениям дополнительно требуется разделять свое состояние, причем каждый раздел называется логическим процессом (LP), так что любое событие получает доступ только к состоянию, относящемуся к его назначенному LP. Это делает события на разных LP тривиально безопасными для одновременного выполнения, но радикально усложняет задачу среды выполнения по обеспечению выполнения событий в порядке временных меток на соответствующих LP.

Существует две основные стратегии выполнения для симуляторов PDES: консервативная и оптимистичная. При консервативном выполнении среда выполнения выполняет событие только тогда, когда она доказала, что никакое другое событие не может быть сгенерировано до него. Time Warp, оптимистичный подход, впервые описанный Джефферсоном [ [**22**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#BibPLXBIB0022) ], использует противоположный подход, где он предполагает, что любое событие, которое есть у ЦП, учитывая, что у него наименьшая временная метка для его LP, вероятно, безопасно для выполнения, несмотря на возможность получения события от однорангового ЦП в более позднее время нового события с меньшей временной меткой. Чтобы справиться с этой возможностью, среда выполнения должна иметь возможность «откатывать» выполнение ранее выполненных событий, чтобы вернуть LP обратно в состояние, когда он должен был выполнить новое событие. Для этого приложение должно предоставить среде выполнения такие средства для отмены выполнения, что само по себе является нетривиальной задачей, а также иметь возможность выполняться в контекстах, сгенерированных недопустимыми порядками событий (которые в конечном итоге будут откачены). Преимущество этого подхода по сравнению с консервативным выполнением заключается в том, что среда выполнения может выполнять события без синхронизации и извлекать выгоду из распространенности событий, которые можно безопасно выполнять, несмотря на отсутствие доказательств этого факта.

**2.1 Асинхронное вычисление GVT**

Ключевая часть любого симулятора PDES на основе Timewarp связана с вычислением глобального виртуального времени (GVT). Обеспечение обратимости последовательности событий требует сохранения информации, которая может быть использована для восстановления состояния до произвольной точки в прошлом времени моделирования. Для этого может потребоваться значительный объем памяти. Глобальное виртуальное время (GVT) — это время моделирования, для которого все события в системе уже были обработаны. Как только GVT превышает временную метку выполненного события, артефакты этих событий можно безопасно уничтожить, освободив память.

В свете акцента Devastator на производительности и масштабируемости мы выбираем асинхронную версию TimeWarp, которая избегает использования любых блокирующих коллективов. Наш метод основан на фазах [ [**27**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#BibPLXBIB0027) ] и в значительной степени следует по стопам [ [**36**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#BibPLXBIB0036) , [**37**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#BibPLXBIB0037) ]. Каждый ранг постоянно выполняет сокращения GVT в фоновом режиме. После завершения фазы сокращения GVT мы получаем новую границу GVT, которая запускает сбор ископаемых, и немедленно начинаем следующее сокращение GVT.

Ключевая проблема в получении высокопроизводительной реализации Timewarp требует ограничения неоправданного оптимизма. Поскольку откатенное событие должно также откатывать события, которые оно отправило соседним процессорам, накладные расходы времени выполнения могут быстро перевесить любые преимущества дополнительного параллелизма, предоставляемого посредством оптимистичного выполнения PDES. Было предложено множество методов для балансировки накладных расходов из-за отката со временем, затраченным на сокращения GVT [ [**12**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#BibPLXBIB0012) , [**13**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#BibPLXBIB0013) , [**17**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#BibPLXBIB0017) , [**30**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#BibPLXBIB0030) , [**42**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#BibPLXBIB0042) , [**48**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#BibPLXBIB0048) , [**49**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#BibPLXBIB0049) , [**50**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#BibPLXBIB0050) ]. Devastator принимает вариант скользящего временного окна (MTW) ​​[ [**47**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#BibPLXBIB0047) , [**53**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#BibPLXBIB0053) ]. Идея высокого уровня заключается в следующем: учитывая GVT, выполнять только события в пределах временного интервала от GVT до GVT+ *w* , где *w* — настраиваемый параметр.

Чтобы исключить необходимость настройки параметров, мы используем алгоритм восхождения на вершину для вычисления оптимального размера окна. Devastator поддерживает скользящее среднее *глобально* зафиксированных и выполненных событий за последние 16 сокращений GVT. Мы называем эффективность отношением зафиксированных к выполненным событиям. Размер окна корректируется после завершения каждого сокращения GVT следующим образом:

1. Если эффективность ниже 33%, размер окна уменьшается вчетверо.
2. Если эффективность составляет от 33% до 66%, размер окна уменьшается вдвое.
3. Если эффективность превышает 95%, размер окна увеличивается вдвое.
4. В противном случае размер окна увеличивается или уменьшается на 1% в зависимости от того, увеличилось или уменьшилось количество зафиксированных событий.

Для обеспечения дальнейшего продвижения размер окна должен быть не менее 1. Чтобы избежать отправки дополнительных сообщений, эти показатели пропускной способности добавляются к сокращениям GVT.

**2.2 API ПДЭС**

Devastator рекламирует свой API на основе C++14 для описания приложения PDES как одно из своих главных достижений, поскольку он лаконичен, но невероятно выразителен. Сначала программа определяет набор событий и их поведение.

*2.2.1 Спецификация событий.* События могут быть объектами любого типа класса C++, если он предоставляет соответствующие методы выполнения и отмены выполнения.

В листинге 1 представлен пример того, как пользователь может указать событие. В этом примере класс myevent отправляет сообщение по цепочке логических процессов (LP). Devastator поддерживает несколько LP на ранг (ранги выполняются до завершения, в то время как LP могут быть прерваны), но для простоты в этом примере мы используем один LP на ранг. Пользователь должен указать одну функцию для класса событий. Функция-член execute выполняет событие. Для этого примера значение хэшируется на основе локального состояния хэша. Devastator предоставляет этим пользовательским событиям объект deva::pdes::executecontext. Этот объект включает локальный текущий идентификатор LP (домен причинности или cd) и время моделирования. Кроме того, контекст выполнения содержит функцию-член 'send', которая позволяет пользователю планировать будущие события. Эта функция-член выполняет учет, необходимый для вычисления GVT. Возвращаемый аргумент функции execute — это другой определяемый пользователем объект, который отменяет событие. Возвращаемый объект будет передан в Devastator, который будет поддерживать свое время жизни до тех пор, пока он не будет либо зафиксирован, либо невыполнен. Этот общий подход позволяет пользователю указать, как откатываются события. Функция-член unexecute реверсора вызывается, когда событие откатывается и, как ожидается, восстановит состояние LP до состояния, предшествующего выполнению события. Может быть указана функция фиксации, которая будет вызвана до сбора ископаемых. Чтобы включить отправку событий между процессами, Devastator использует API сериализации UPC++ [ [**2**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#BibPLXBIB0002) ] для эффективного преобразования объектов в буферы, которые могут быть десериализованы принимающим процессом. В этом примере мы используем простой макрос SERIALIZEDFIELDS.

Доступ к состоянию LP осуществляется локально доступными глобальными переменными. Devastator не предоставляет высокоуровневых абстракций LP. Это имеет свои преимущества и недостатки. Обычно приложения Devastator поддерживают локальные структуры данных потоков, которые могут быть проиндексированы с помощью идентификаторов доменов причинности. Этот подход позволяет пользователю указывать, как должно быть размещено и сохранено локальное состояние LP. К недостаткам можно отнести отсутствие автоматизированных стратегий отката, таких как основанные на отслеживании адресных пространств, связанных с LP [ [**33**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#BibPLXBIB0033) , [**51**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#BibPLXBIB0051) ]. Кроме того, он требует от пользователя поддерживать структуры данных для сопоставления идентификатора LP с парой идентификаторов ранга и домена причинности.

Основные различия между API Devastator и существующими симуляторами ROOT-Sim и ROSS заключаются в использовании поддержки C++ для более сложных типов данных и возможностей вывода типов. Описанный API PDES позволяет пользователям отправлять сообщения одним вызовом executecontext::send. Типы событий автоматически сериализуются и могут содержать произвольные типы данных (по сравнению с ROSS, который запрещает отправку динамически выделенных данных). Кроме того, поддержка вывода типов в C++ устраняет необходимость регистрации различных типов событий и обработчиков (как в случае с ROSS) или демультиплексирования типов событий как части обратного вызова processEvent в ROOT-Sim. Благодаря использованию идиомы стирания типов Devastator может автоматически создавать эти таблицы поиска во время компиляции. Использование Devastator библиотеки C++ исключает шаблонный код и создает более читаемый код.

*2.2.2 Поток управления программой и управление событиями.* Небольшая часть среды выполнения PDES встроена в доступный пользователю внешний слой, напоминающий типичную распределенную параллельную модель SPMD, которая позволяет пользователю вызывать несколько симуляций PDES за один запуск задания или временно прерывать выполнение PDES для участия в коллективных операциях, а затем возобновлять PDES. В листинге 2 показан API. Перед вызовом уровня PDES каждый процесс инициализирует среду выполнения связи с помощью вызова deva::run (раздел [**3**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#sec-8) ). После этого каждый рабочий поток должен указать количество локальных LP в качестве аргумента для deva::pdes::init. Пользователь планирует начальные события с помощью deva::pdes::rootevent. Когда все готово, deva::pdes::drain выполняет все события вплоть до виртуального времени tendency. Вызовы deva::pdes::drain предоставляют возможность глобально приостанавливать симуляцию на виртуальных временных метках.

Другая особенность вызовов дренажа заключается в том, что установка *обратимого* аргумента в значение true не будет выполнять никакого сбора ископаемых. Коллективный вызов deva::pdes::rewind восстановит состояние симуляции до предыдущего вызова дренажа. Повторные вызовы дренажа и rewind позволяют пользователям исследовать сценарии управления или запускать системы моделей с неопределенными параметрами. Наконец, функция deva::pdes::finalize очищает структуры данных.

3 ВРЕМЯ ВЫПОЛНЕНИЯ КОММУНИКАЦИИ

Devastator состоит из многочисленных программных компонентов, включая планировщик событий, рабочие потоки и распределенную систему связи. Здесь мы описываем детали проектирования и реализации компонента связи, которые существенно отличаются от обычных систем выполнения. Мы также обсуждаем знания предметной области, которые используются для оптимизации производительности.

**3.1 Бэкэнд связи**

Общая структура коммуникационного движка следует парадигме SPMD, распространенной в HPC. Он также предоставляет продуктивные и высокопроизводительные средства для отправки активных сообщений с использованием современного синтаксиса C++, а также несколько других глобальных коллективов для производительности. Как и в MPI, мы называем наших агентов параллельного выполнения «рангами», но в отличие от MPI, где эти ранги соответствуют процессам ОС, наши сопоставляются с потоками ОС с намерением, чтобы их было много на процесс. Поскольку активное сообщение может быть отправлено между любыми двумя рангами, это означает, что мы поддерживаем глобальный обмен сообщениями между потоками, независимо от того, принадлежат ли эти потоки к одному процессу или нет.

Тот факт, что мы поддерживаем обмен сообщениями в адресном пространстве (между потоками одного процесса), нетипичен для основных моделей коммуникации. Конструкции обмена сообщениями обычно резервируются для агентов, которые не разделяют адресное пространство, поскольку часто предполагается, что какой-то механизм общей памяти может лучше обслуживать потребность с меньшей задержкой. Но недостатком общего состояния является миграция строк кэша по требованию оборудованием, и стоимость этого проявляется в виде остановок ЦП при промахах кэша L1. Когда объем данных, «подмешиваемых» в общее состояние, достаточно мал, чем требуемый трафик памяти для выполнения этого смешивания, может стать выгодным перейти к стратегии, в которой состояние разделено между процессорными ядрами, а сообщения отправляются владельцу состояния для выполнения смешивания от имени отправителя. Мы считаем, что PDES является отличным кандидатом для этого подхода по двум причинам. Во-первых, события (сообщения) обычно занимают порядка нескольких строк кэша, в то время как общее состояние представляет собой отсортированную очередь событий, перемещение которой в иерархии памяти может оказаться затратным, а во-вторых, вставка события в очередь — это операция без результата, поэтому отправителю не нужно ждать ее завершения, что позволяет скрыть задержку обработки сообщения в целевом объекте.

**3.2 Поток коммуникации**

Всякий раз, когда событие выдает запрос на связь (например, cxt.send() в листинге 1), поток связи ранга обрабатывает этот запрос. Основной механизм для связи рангов — это процедура deva::send(Fn,Arg...), которая асинхронно отправляет функцию и аргументы (активное сообщение) другому рангу, где она вызывается в произвольное время во время некоторого вызова deva::progress(). Среда выполнения Devastator была разработана для оптимизации случая, когда отправки многочисленны и имеют небольшой размер (< 1 КБ каждая), поскольку это основная операция связи, необходимая во время типичного моделирования PDES.

Devastator также предоставляет небольшой набор коллективных операций, родственных операциям MPI, но в настоящее время не оптимизированы для производительности за пределами использования масштабируемого алгоритма из учебника. Они доступны приложению между вызовами PDES, например, для сбора статистики, ввода-вывода или вычисления балансировки нагрузки. Основные прототипы API связи Devastator представлены в листинге 3.

**3.3 Конфигурация и оптимизация**

Среда выполнения Devastator может быть развернута в двух различных режимах параллелизма: однопроцессный многопоточный (нераспределенный) и многопроцессный многопоточный (распределенный). В нераспределенных запусках все потоки принадлежат одному процессу и могут совместно использовать одно и то же адресное пространство. Сообщения, отправляемые между потоками, не сериализуются, вместо этого используется семантика перемещения C++ для переноса объекта в очередь сообщений целевого потока, что позволяет добиться наибольшей степени исключения копирования. В распределенных запусках адресные пространства являются попроцессными и поэтому требуют, чтобы приложение осознавало иерархию процесса/потока для осмысленного совместного использования памяти. Сообщения в локальные потоки процесса отправляются с помощью std::move, как и раньше, но сообщения вне процесса должны быть сериализованы, переданы и десериализованы на целевой стороне.

Devastator имеет несколько реализаций очередей сообщений для службы обмена сообщениями между потоками, которые можно выбрать во время сборки, что позволяет пользователю сделать оптимальный выбор. Обе реализации являются реализациями без блокировок. Мы поддерживаем очереди all-to-all single-producer-single-consumer (SPSC), в которых выделенная очередь существует между каждой парой потоков в процессе и отлично подходит для низкого и среднего количества потоков, и очереди all-to-one multi-producer-single-consumer (MPSC), которые лучше подходят для большего количества потоков в эпоху многоядерных процессоров из-за немасштабируемости очередей SPSC *N*2. Чтобы уменьшить задержку обработки конфликтов потоков, мы используем атомарные операции чтения-изменения-записи вместо спин-блокировок. Чтобы еще больше улучшить масштабируемость службы обмена сообщениями потоков, мы разрабатываем специальный распределитель памяти, называемый *deva malloc* . Подобно универсальным проектам, таким как jemalloc, opnew malloc Deva использует отдельные ячейки частного размера потока для небольших объектов. Deva malloc периодически сканирует частные ячейки, чтобы отправить память обратно потоку, который изначально ее выделил, используя те же очереди сообщений потока, которые мы использовали для всего остального. Такая конструкция позволяет opnew deva оставаться свободным от блокировки практически везде, что принципиально невозможно в jemalloc, поскольку он не владеет временем жизни потоков.

Для многопроцессорных сборок наша возможность межпроцессного обмена сообщениями предоставляется библиотекой GASNet, переносимым и высокопроизводительным слоем для HPC-коммуникаций. Хотя MPI может быть фактическим стандартом для передачи сообщений в HPC, наша сильная зависимость от активных сообщений делает MPI непривлекательным в этом контексте. GASNet поддерживает активные сообщения с момента своего создания и имеет высокооптимизированную реализацию для каждой из основных поддерживаемых ею сетей HPC. Что касается режима коммуникации, у нас было два варианта: i) потоковый режим, в котором все рабочие потоки могут безопасно вызывать коммуникационные вызовы, и ii) один коммуникационный поток обрабатывает все коммуникационные запросы от своих рабочих потоков. Первый вариант влечет за собой значительные затраты, поскольку заставляет среду выполнения коммуникации использовать синхронизированный доступ к базовой фабрике. По этой причине мы решили настроить GASNet в режиме одного коммуникационного потока и выделили внутренний «сервисный» поток для выполнения всех операций GASNet. Поток службы не только получает доступ к сети без синхронизации, но и внутреннее состояние GASNet, необходимое для проведения сетевых операций, также получает большую локальность, поскольку оно работает на фиксированном ЦП, который мало что делает. Это пример нашего принципа разделения состояния и отправки сообщений, а не синхронизации для совместного использования состояния. Однако недостатком является то, что сообщение теперь должно испытывать задержку в три перехода, чтобы добраться до своего пункта назначения: начальное сообщение поток-поток локальному потоку службы, активное сообщение GASNet процессу назначения, а затем окончательное сообщение поток-поток от удаленного потока службы потоку назначения. Эти затраты кажутся оправданными, особенно в свете дополнительной оптимизации, когда поток службы жадно агломерирует исходящие сообщения, разделяющие один и тот же процесс назначения, чтобы увеличить средний размер сообщения и амортизировать влияние накладных расходов на сообщение.

4 PHOLD ЭТАЛОННЫЙ МАРКЕТ

**4.1 Настройка эталонного теста**

Чтобы продемонстрировать производительность Devastator для задач произвольного размера, мы использовали модифицированный бенчмарк PHOLD [ [**19**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#BibPLXBIB0019) ]. В PHOLD каждый LP инициализируется событиями, и каждое из них будет планировать только последующее будущее событие на текущем LP или одном из его одноранговых LP. Таким образом, каждая цепочка событий аналогично описывается как луч, отражающийся между LP, поэтому рабочую нагрузку можно измерить с помощью количества лучей. Время доставки сообщений задерживается на значение, выбранное из экспоненциального распределения. Целевой LP следующего события получается путем выборки из гауссовского распределения с центром в отправляющем LP. Стандартное отклонение ( *σ* ) корректируется для управления долей межузловых и внутриузловых коммуникаций. Мы решили ввести этот альтернативный режим отражения, поскольку мы обнаружили, что равномерная вероятность перехода исходного PHOLD мало похожа на реальные рабочие нагрузки Devastator, но которые вместо этого содержат существенную локальность.

**4.2 Конфигурации оборудования и среды выполнения**

Для этих исследований производительности мы используем раздел ЦП машины Perlmutter компании NERSC и Frontera компании TACC. В системе Perlmutter каждый вычислительный узел имеет 2 сокета, и каждый сокет имеет процессоры AMD EPYC 7763, что в общей сложности обеспечивает 128 физических ядер. Узлы соединены с помощью технологии межсоединений Hewlett Packard Enterprise Slingshot 11. В Frontera каждый вычислительный узел содержит 2 28-ядерных процессора Intel 8280 «Cascade Lake». Узлы соединены через межсоединении Mellanox Infiniband, HDR-100. Информация о среде выполнения сведена в Таблицу  [**1**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#tab1) .

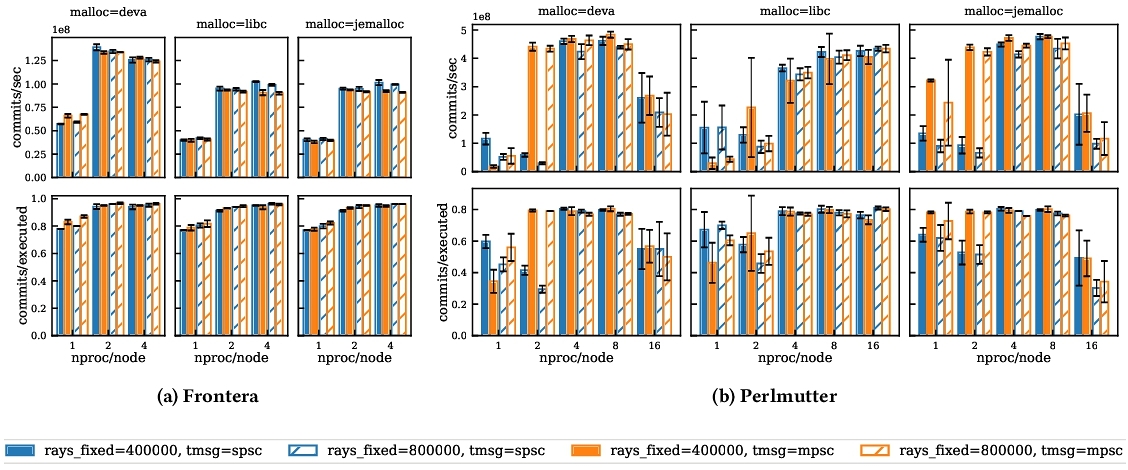
**Таблица 1:**Версии компилятора и среды выполнения

|  |  |  |
| --- | --- | --- |
|  | Перлмуттер | Фронтера |
| Компилятор | GCC-Cray v11.2.0 | GCC-версия 9.1.0 |
| Версия GASNet | 2023.3.3 | 2023.3.3 |
| GASNet-трубопровод | офи | ibv |
| Версия кабелепровода | libfabric v1.15.2.0 | ibverbs v20.35.2000 |

**4.3 Определение оптимальной конфигурации**

Сначала мы проводим параметрическую развертку, чтобы найти оптимальную конфигурацию для запуска Devastator на Perlmutter и Frontera. Devastator содержит множество параметров для настройки. Мы вычисляем пропускную способность на матрице параметров, включая количество процессов на узел, три поддерживаемых Devastator распределителя памяти, а именно *deva* , *libc* и *jemalloc*  [ [**16**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#BibPLXBIB0016) ], где *deva* — это новый распределитель, вдохновленный *jemalloc* , и две реализации очереди сообщений без блокировки, SPSC и MPSC.

Чтобы сосредоточиться на поведении локальных коммуникаций, мы используем гауссовские вероятности перехода с небольшим значением 100 для *σ* и назначаем 100 LP каждому ядру. В каждом запуске Perlmutter используются четыре узла, а в каждом запуске Frontera — два узла. Это требует межузловой коммуникации во время калибровки. Чтобы исследовать влияние размера проблемы, мы протестировали 400 000 лучей и 800 000 лучей для каждой конфигурации. Результат представлен на рисунке  [**1**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#fig1) .

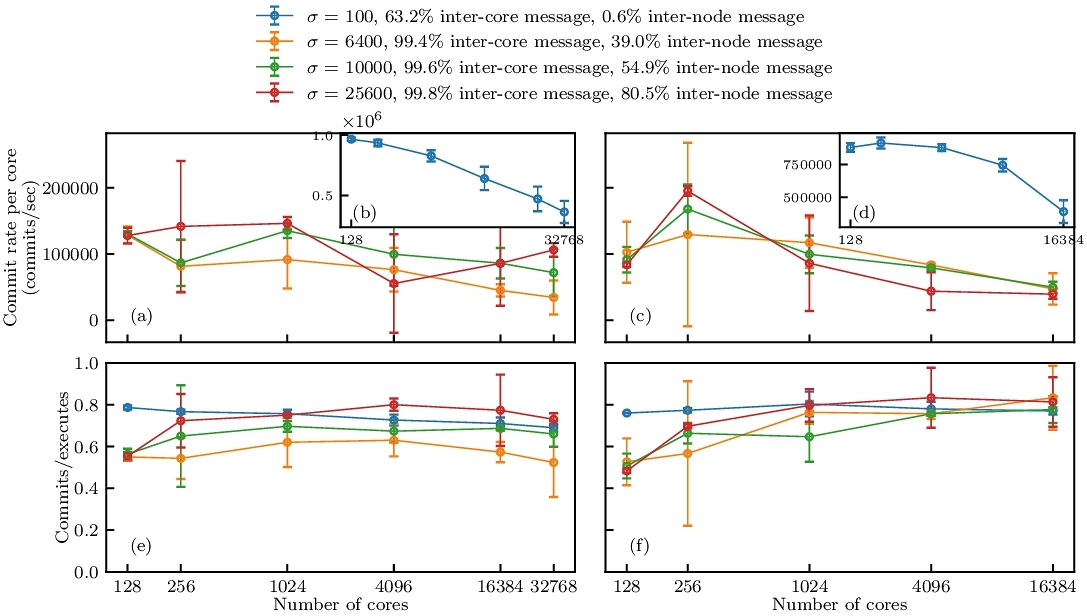
Рисунок 1: Производительность с использованием различных распределителей памяти, очередей сообщений и количества процессов. Три столбца подграфиков включают данные с различными распределителями памяти, как показано в верхней части столбцов.

Прежде всего, мы не наблюдаем значительного влияния размера проблемы на пропускную способность для оптимальных конфигураций на любой архитектуре (менее 5%). Для распределителей памяти *jemalloc* обеспечивает сопоставимую пропускную способность с *deva* на Perlmutter. Однако на Frontera с 2 процессами/узлом распределитель *deva* в 1,4 раза быстрее, чем *jemalloc* . По мере масштабирования на Perlmutter *libc* обеспечивает все более конкурентоспособную производительность, поскольку количество потоков, конкурирующих за ресурсы памяти, уменьшается. В обеих системах мы наблюдаем существенное преимущество в производительности, связанное с правильной обработкой доменов NUMA. Сравнивая конфигурации 800 тыс. лучей/ *deva* для 1 процесса на узел с 2 процессами/узлом на Frontera и 8 процессами/узлом на Perlmutter, мы наблюдаем ускорение в 2,3 и 8,1 раза соответственно. Это подчеркивает важность правильного учета эффектов NUMA на современных многоядерных системах, о чем также сообщалось в недавней работе PDES [ [**40**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#BibPLXBIB0040) ]. На Perlmutter мы отмечаем, что производительность для 4 и 8 процессов/узел обеспечивает наилучшую и наиболее постоянную производительность. Поскольку узел Perlmutter имеет только два сокета, разница в производительности между узлами SPSC и MPSC является несколько неожиданной, но, вероятно, из-за более низкой эффективности запусков SPSC. В оставшейся части раздела мы используем распределитель памяти *deva* , очереди сообщений SPSC и 2 процесса на узел на Frontera и очереди сообщений MPSC и 8 процессов на узел на Perlmutter.

**4.4 Слабое и сильное масштабирование**

Чтобы лучше понять, как производительность Devastator масштабируется в массовых параллельных запусках, мы провели слабые и сильные тесты масштабирования с гауссовыми режимами отражения с различным стандартным отклонением *σ* . Мы выбрали четыре значения *σ* для управления степенью локальности коммуникаций, а именно *σ* = 100, 6400, 10000 и 25600. Например, когда *σ* = 100, около 60% сообщений были предназначены для маршрутизации на другие ядра и почти ни одно сообщение не было отправлено на другие узлы, тогда как когда *σ* = 10000, почти все сообщения (99,6%) были отправлены на другие ядра и 55% сообщений были отправлены на другие узлы.

В тесте слабого масштабирования каждому LP назначается 15 лучей, а каждому ядру — 100 LP; таким образом, для каждого узла у нас есть 192 тыс. лучей. Моделирование заканчивается после того, как каждый луч выполняется 50 тыс. раз. В тесте сильного масштабирования используется 400 тыс. лучей независимо от количества используемых узлов. Для оценки уровня шума в каждой конфигурации проводится по три запуска. Данные представлены на рисунке  [**2**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#fig2) .

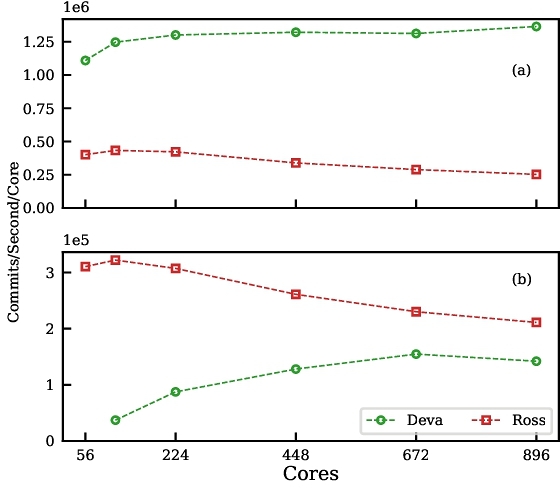
Рисунок 2: Слабые результаты масштабирования показаны на подграфиках (a), (b) и (e) в левом столбце, а сильные результаты масштабирования показаны на подграфиках (c), (d) и (f) в правом столбце. σ *представляет* собой стандартное отклонение гауссовых распределений, которые определяют целевые LP.

В слабом масштабировании мы масштабировали симуляцию до 256 узлов (32768 ядер), поддерживая скорость фиксации на ядро ​​около 30% - 80% от скорости фиксации на ядро ​​одного узла. Хотя данные кажутся шумными, общая тенденция к снижению параллельной эффективности остается справедливой в диапазоне количества узлов. Эффективность запусков, представленная соотношением зафиксированных событий и выполненных событий, также отслеживалась и представлена ​​на рисунке  [**2**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#fig2) (e); где можно заметить, что эффективность хорошо контролируется в диапазоне от 60% до 80% с помощью нашего динамического алгоритма просмотра вперед.

При сильном масштабировании мы также смогли достичь параллельной эффективности около 50% на 16K ядрах, что соответствует ускорению в 56x-70x. Стоит отметить, что параллельная эффективность больше единицы на двух узлах для всех значений σ *,* что вызвано плохой производительностью одного узла и может быть связано с двумя факторами. Во-первых, когда все рабочие нагрузки сосредоточены на одном узле, кэш может быть переполнен, что приведет к большему количеству промахов кэша. Во-вторых, все коммуникации, обозначенные как межузловые или внутриузловые, на самом деле являются внутриузловыми коммуникациями в одноузловых симуляциях и могут перегрузить пропускную способность памяти узла. Доказательством чрезмерного доступа к памяти может служить следующее наблюдение: при переходе с одного узла на два наблюдается более значительное увеличение скорости фиксации на ядро, когда имеется больше назначенных межузловых сообщений, поскольку назначенные межузловые сообщения были доставлены на другой узел, что снижает потребность в доступе к памяти внутри узла и, таким образом, повышает производительность.

**4.5 Сравнение Devastator и ROSS**

Мы завершаем наше исследование бенчмарка PHOLD сравнением производительности Devastator с производительностью ROSS [ [**7**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#BibPLXBIB0007) ]. Для справедливого сравнения мы создаем общий API таким образом, чтобы оба симулятора использовали идентичный код приложения. Из-за проблем с реализацией MPI поставщика Perlmutter мы провели сравнение на машине Frontera TACC. В качестве бенчмарка мы используем бенчмарк Gaussian PHOLD со стандартными отклонениями σ *=* 100 и *σ* = 10000, чтобы подчеркнуть влияние гибридного параллелизма Devastator. Фиксации в секунду показаны на рис.  [**3**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#fig3) .

Рисунок 3: Devastator против ROSS на Frontera для теста Gaussian PHOLD с (a) *σ* = 100 и (b) *σ* = 10 4 .

Сравнение ROSS и Devastator показывает сильную зависимость от рабочей нагрузки. Для *σ* = 100, что соответствует существенной локальности в рабочих нагрузках событий, Devastator превосходит ROSS в 2,9 раза (2 узла) до 5,4 раза (16 узлов). Когда стандартное отклонение увеличивается, ROSS превосходит Devastator в 8,7 раза (2 узла) до 1,5 раза (16 узлов). Мы считаем, что плохая производительность Devastator возникает из-за размера проблемы. При таком количестве сообщений, летающих вокруг, асинхронный алгоритм GVT медленнее завершается, а адаптивное временное окно перемещения медленно адаптируется. Решение этой проблемы является темой будущей работы.

Наконец, отметим, что мы использовали одну конфигурацию Devastator для всех запусков. С другой стороны, каждая конфигурация ROSS при каждом количестве узлов требовала развертки сетки из 45 запусков на узел для определения оптимальных параметров партии и интервала. Таким образом, даже в случаях, когда ROSS был существенно быстрее, ROSS требовал существенной настройки, что снижало производительность пользователя.

5 ПРИМЕНЕНИЕ

Оценка нашей реализации Devastator в Mobiliti [ [**8**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#BibPLXBIB0008) ] направлена ​​на то, чтобы ответить на два исследовательских вопроса:

* Как основные алгоритмические особенности Devastator влияют на его производительность?
* Может ли предлагаемый адаптивный MTW обеспечить существенное ускорение моделирования сетей распространения трафика?

**5.1 Настройка эталонного теста**

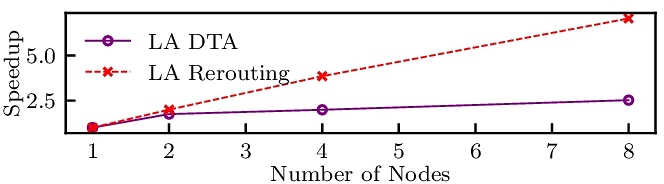
Mobiliti [ [**8**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#BibPLXBIB0008) ] — это крупномасштабное параллельное дискретно-событийное моделирование, разработанное для анализа транспортных систем, созданных на основе Devastator, что является первым случаем применения PDES в транспортировке. В Mobiliti транспортным средствам назначаются маршруты одним из двух способов. В Dynamic Traffic Assignment (DTA) оптимизация функции полезности выполняется для генерации статического набора маршрутов для всех транспортных средств. В качестве альтернативы динамическое изменение маршрута направляет транспортные средства с использованием иерархии сокращения, но обновляет маршруты в ответ на меняющиеся модели заторов. Mobiliti работает с 4 процессами на узел на Perlmutter.

В нашем анализе транспорта и городского ландшафта Лос-Анджелеса используется модель спроса на поездки Ассоциации правительств Южной Калифорнии (SCAG) [ [**14**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#BibPLXBIB0014) ], обеспечивающая полное понимание региональных моделей поездок. Для дорожной сети мы интегрировали геопространственные данные с карты HERE [ [**1**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#BibPLXBIB0001) ], гарантируя, что наши выводы являются точными и актуальными. Объединяя эти разнообразные, но взаимодополняющие источники данных, наша статья представляет собой детальное и детальное описание городских и транспортных сложностей Лос-Анджелеса. Модель LA существенно больше, чем ранее представленные моделирования Mobiliti. Если предыдущие запуски SF состояли из 19,2 миллионов поездок по сети с 450 тысячами узлов и 1 миллионом связей, то модель LA планирует 49,2 миллиона поездок по дорожной сети с 2 миллионами связей и 903 тысячами узлов.

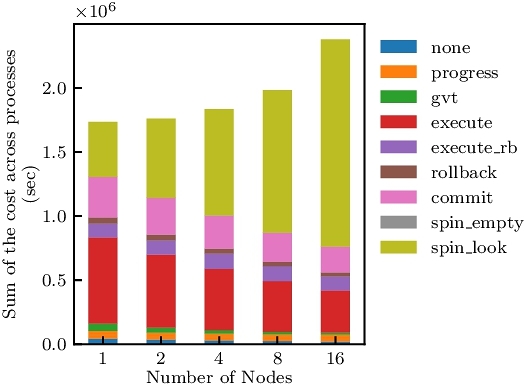
**5.2 Масштабируемость мобильности**

Mobiliti распараллеливается путем разделения дорожной сети по рангам. Исторические данные о трафике используются для взвешивания связей перед разделением METIS [ [**23**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#BibPLXBIB0023) ]. Достигнутый баланс нагрузки может ухудшиться в течение дня, поскольку схемы движения отклоняются от средней схемы движения, что приводит к заторам и неравномерности распределения потока в конце дня.

На рисунке [**4**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#fig4) мы демонстрируем параллельное ускорение Mobiliti. Сценарий динамического перенаправления демонстрирует время выполнения от 1,2 миллиона секунд до 181 365 секунд, что соответствует параллельной эффективности 88%. Случай DTA, который содержит меньше вычислительной работы, выполняется на одном узле за 452 секунды и достигает лучшего времени решения в 179 секунд, что соответствует параллельной эффективности 32%.

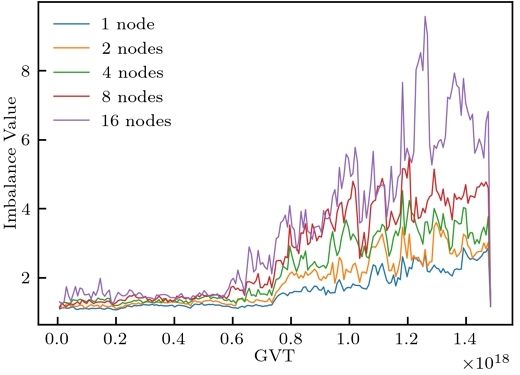
Рисунок 4: Параллельное ускорение Mobiliti Perlmutter относительно одного узла (128 ядер).

Чтобы понять ухудшение производительности, на рисунке  [**5**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#fig5) время выполнения разбито по выполняемой работе. Мы наблюдаем заметный рост общей стоимости выполнения, связанной с обзором спина, что указывает на то, что в локальных очередях событий есть события, которые не могут быть выполнены из-за дросселя MTW. Это явление становится все более очевидным по мере масштабирования на большее количество узлов. Реализация динамической перемаршрутизации вносит значительный вклад в снижение общей стоимости выполнения. Пересчет маршрутов для всех транспортных средств может быть выполнен смущающе параллельным образом. Однако важно отметить наличие последовательных компонентов в системе, в частности, иерархии сжатия, которая пересчитывается каждым потоком. Это неизбежное последовательное узкое место подчеркивает необходимость поддержания тонкого баланса между параллельной и последовательной обработкой для оптимизации общей производительности системы.

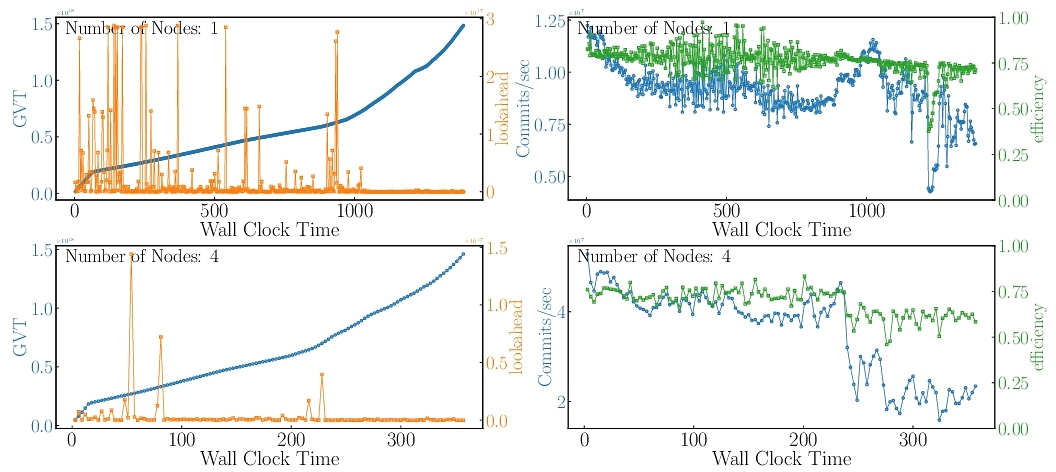
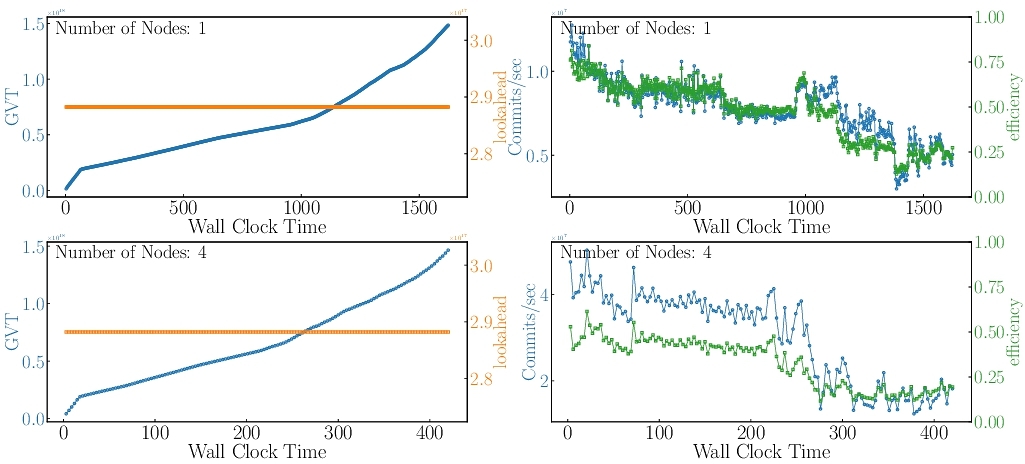
Рисунок 5: Временная разбивка для Mobiliti с изменением маршрута.

На рисунке  [**6**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#fig6) мы определяем метрику дисбаланса,яя, как:

я=макс(τ)−в среднем(τ)в среднем(τ)я=макс(τ)−в среднем⁡(τ)в среднем⁡(τ)

где *τ* представляет время выполнения, т.е. время, потраченное на выполнение событий, которые в конечном итоге будут зафиксированы. Из рисунка видно, чтояядемонстрирует растущую тенденцию к кульминации моделирования. Мы считаем, что, учитывая медленное ухудшение дисбаланса нагрузки, полустатические подходы, которые периодически приостанавливают моделирование для перераспределенных ЛП, такие как в [ [**6**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#BibPLXBIB0006) ], могут быть достаточными, и планируем изучить эти варианты в будущей работе.Рисунок 6: Вычислить дисбаланс в сравнении с GVT

Мы стремимся подтвердить, что реализация динамического опережающего просмотра в обработке GVT увеличит скорость, сохраняя при этом постоянный уровень эффективности. Кроме того, мы стремимся понять последствия нереализации MTV. Чтобы добиться этого, мы провели тесты на Mobiliti как с динамическим опережением, так и без него. На рисунке  [**7**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#fig7) мы анализируем эффективность Mobiliti с перемаршрутизацией. Сравнивая производительность одного узла с четырьмя узлами, мы видим небольшое снижение эффективности. Однако крайне важно подчеркнуть высокий уровень эффективности, поддерживаемый на всех узлах, что подчеркивает надежность механизма перемаршрутизации. На рисунке [**8**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#fig8) показано, что без динамического опережающего просмотра эффективность снижается, а общее время выполнения для всей симуляции больше, чем с динамическим опережением. Для симуляции с одним узлом время выполнения составляет 1641,9 секунды против 1404,0 секунды, т. е. в 1,17 раза быстрее. Для симуляции с четырьмя узлами мы получаем 428,1 секунды против 369,0 секунд, т. е. в 1,16 раза быстрее.

Рисунок 7: Прогресс с динамическим прогнозированиемРисунок 8: Прогресс со статическим прогнозированием

6 СВЯЗАННЫЕ РАБОТЫ

Глобальное виртуальное время (GVT) лежит в основе оптимистической синхронизации. Устанавливая момент времени моделирования, до которого все сообщения были обработаны, можно вернуть ресурсы памяти для отката. Синхронные сокращения GVT выполняют события до входа в блокирующий вызов. Количество событий обычно ограничивается либо количеством событий, либо виртуальными приращениями времени. Обычно эти методы требуют понимания компромиссов между накладными расходами из-за отката и временем, затраченным на сокращения GVT.

Существует множество стратегий для ограничения необоснованного оптимизма в параллельных дискретных событиях моделирования [ [**12**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#BibPLXBIB0012) ]. Дизайн методов регулирования обычно варьируется в зависимости от того, как идентифицированы чрезмерно активные LP и как эти LP задерживаются. После того, как активный оптимизм был идентифицирован, подходы к регулированию LP включают остановку или приостановку процессов [ [**49**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#BibPLXBIB0049) ], изменение количества итераций в основном цикле моделирования Time Warp [ [**50**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#BibPLXBIB0050) ], вероятностный пропуск выполнения событий [ [**17**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#BibPLXBIB0017) ], ограничение использования памяти [ [**13**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#BibPLXBIB0013) ], настройку алгоритмов планирования LP [ [**30**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#BibPLXBIB0030) , [**48**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#BibPLXBIB0048) , [**57**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#BibPLXBIB0057) ], принудительную синхронизацию [ [**15**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#BibPLXBIB0015) ], перебалансировку нагрузки [ [**42**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#BibPLXBIB0042) ] и использование информации, специфичной для домена [ [**4**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#BibPLXBIB0004) , [**6**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#BibPLXBIB0006) , [**25**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#BibPLXBIB0025) ].

Подход, используемый в Devastator, представляет собой адаптивную формулировку движущегося временного окна [ [**47**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#BibPLXBIB0047) ]. Адаптивная формулировка для размера окна, основанная на «полезной работе» — количестве зафиксированных событий за время настенных часов — сформулирована в [ [**29**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#BibPLXBIB0029) ], но они рассматривают локальные окна, а не одно глобальное окно. В [ [**54**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#BibPLXBIB0054) ] они адаптивно изменяют размер временного окна, используя обучение с подкреплением, и демонстрируют 15% ускорение на 4-ядерной машине с использованием моделирования VLSI. Подход с генетическим алгоритмом используется в [ [**55**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#BibPLXBIB0055) ]. Это позволяет каждому LP иметь локальное временное окно, но требует запуска нескольких параллельных симуляций для настройки параметров.

Вычислительные системы кардинально изменились за последние 40 лет, что привело к кардинальным изменениям в том, как симуляторы PDES лучше всего оптимизировать для современных систем. В ранние годы использовался ограниченный параллелизм. DEVS — это формализм для описания систем дискретных событий [ [**58**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#BibPLXBIB0058) ]. Он использует иерархическую декомпозицию модели, что позволяет строить модель из нескольких подмоделей. PDEVS еще больше ослабляет последовательные ограничения в DEVS для большего количества параллелизмов [ [**10**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#BibPLXBIB0010) ]. Перенесемся в наши дни: крупномасштабное моделирование PDES [ [**3**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#BibPLXBIB0003) , [**36**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#BibPLXBIB0036) , [**37**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#BibPLXBIB0037) ] масштабируется до сотен тысяч или миллионов ядер. Эти подходы используют плоскую модель программирования модели MPI. ROOT-Sim [ [**34**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#BibPLXBIB0034) ] разработал оптимизированный алгоритм сокращения GVT с общей памятью [ [**32**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#BibPLXBIB0032) ], который был расширен до систем с распределенной памятью [ [**52**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#BibPLXBIB0052) ]. ROOT-Sim [ [**34**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#BibPLXBIB0034) ] выполняет динамическое выделение и освобождение памяти через стандартную библиотеку malloc, подключенную ядром и перенаправленную в оболочку. Платформа моделирования имеет внутреннее состояние, которое различает, принадлежит ли текущий поток выполнения коду уровня приложения или внутренним компонентам платформы. В первом случае подключенные вызовы перенаправляются через оболочку во внутренний диспетчер карт памяти (называемый DyMeLoR), который обрабатывает операции выделения/освобождения, максимизируя локальность памяти для макета состояния для каждого отдельного объекта моделирования и поддерживая метаданные, позволяющие восстановить карту памяти до прошлых значений. Кроме того, программное обеспечение уровня приложения может быть скомпилировано с помощью специального легкого инструментария, чтобы прозрачно предоставить диспетчеру карт памяти возможность отслеживать во время выполнения, какие области памяти изменяются. В [ [**56**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#BibPLXBIB0056) ] ROSS-MC-CMT продемонстрировал ускорение до 10 раз по сравнению с ROSS (который использует алгоритмы приоритетной очереди, такие как календарь и куча [ [**7**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#BibPLXBIB0007) ]) за счет введения очередей сообщений общей памяти и введения выделенных потоков связи. Результаты представлены только для 8 узлов. Они также вводят эвристику на основе эффективности для настройки размера интервала GVT (т. е. количества событий, выполненных на одно сокращение GVT). Они не сообщают об успехе при использовании эвристики скользящего среднего, представленной здесь. Мы считаем, что это происходит из-за того, что Devastator использует асинхронный алгоритм GVT, который постоянно выполняет сокращения.

7 ЗАКЛЮЧЕНИЕ

Мы представили Devastator, программную среду для оптимистичных симуляций PDES. Devastator включает новую реализацию алгоритма TimeWarp Джефферсона, чтобы симуляции PDES могли хорошо работать на современных архитектурах систем с глубокой иерархией памяти и сложной сетью взаимосвязей. API C++ также облегчает процесс разработки и обслуживания кода. На системном уровне Devastator предлагает несколько вариантов конфигурации и оптимизации для очереди сообщений и межузловой связи. Экспериментальные результаты на ядрах до 32 тыс. демонстрируют преимущества производительности и масштабируемости, которых может достичь программа, разработанная на Devastator, по сравнению с существующими средами. На следующих этапах мы планируем разработать дополнительные аппаратные бэкэнды, включая графические процессоры, и предложить библиотеки, специфичные для домена, для повышения производительности программирования.

БЛАГОДАРНОСТИ

Это исследование было поддержано Программой исследований в области передовых научных вычислений в Министерстве энергетики США, Офисе науки, по номеру гранта DE-AC02-05CH11231, и использовало ресурсы Национального центра научных вычислений в области энергетики (NERSC), который поддерживается Офисом науки Министерства энергетики США по контракту № DE-AC02-05CH11231. Авторы выражают благодарность Техасскому центру передовых вычислений (TACC) в Техасском университете в Остине за предоставление ресурсов HPC, которые способствовали результатам исследования, представленным в этой статье.

ССЫЛКИ

* ок. 2023 г. HERE Directions 2023: узнайте о последних инновациях и разработках в области технологий определения местоположения. [**https://www.here.com/why-here**](https://www.here.com/why-here) . Дата обращения: 30.10.2023. Перейти кцитата 1
* Джон Бачан, Скотт Б. Баден, Стивен Хофмейр, Матиас Жаклин, Амир Камил, Дэн Боначеа, Пол Х. Харгроув и Хадия Ахмед. 2019. UPC++: высокопроизводительная коммуникационная структура для асинхронных вычислений. В *2019 году на IEEE International Parallel and Distributed Processing Symposium (IPDPS)* . 963–973. [**https://doi.org/10.1109/IPDPS.2019.00104**](https://doi.org/10.1109/IPDPS.2019.00104) Перейти кцитата 1
* Питер Д. Барнс-младший, Кристофер Д. Карозерс, Дэвид Р. Джефферсон и Джастин М. ЛаПре. 2013. Warp Speed: Executing Time Warp on 1,966,080 Cores. На конференции по принципам усовершенствованного дискретного моделирования (Монреаль, Квебек, Канада). 327–336. [**https://doi.org/10.1145/2486092.2486134**](https://doi.org/10.1145/2486092.2486134) Перейти кцитата 1
* Павол Бауэр, Джонатан Линден, Стефан Энгблом и Бенгт Йонссон. 2015. Эффективная межпроцессная синхронизация для параллельного дискретного моделирования событий на многоядерных процессорах. В трудах 3-й конференции ACM SIGSIM по принципам усовершенствованного дискретного моделирования (Лондон, Великобритания) (SIGSIM PADS '15). Ассоциация вычислительной техники, Нью-Йорк, США, 183–194. [**https://doi.org/10.1145/2769458.2769476**](https://doi.org/10.1145/2769458.2769476) Перейти кцитата 1
* Дэн Боначеа и Пол Х. Харгроув. 2018. GASNet-EX: высокопроизводительная, переносимая коммуникационная библиотека для экзафлопсных вычислений. В сборнике трудов по языкам и компиляторам для параллельных вычислений (LCPC'18) (конспекты лекций по информатике, том 11882). Springer International Publishing. [**https://doi.org/10.25344/S4QP4W**](https://doi.org/10.25344/S4QP4W)  [**https://doi.org/10.25344/S4QP4W**](https://doi.org/10.25344/S4QP4W) . Перейти кцитата 1
* Максимилиан Бремер, Джон Бачан, Сай Чан и Клинт Доусон. 2021. Спекулятивное параллельное выполнение для локального временного шага. В трудах конференции ACM SIGSIM 2021 года по принципам усовершенствованного дискретного моделирования (виртуальное событие, США) (SIGSIM-PADS '21). Ассоциация вычислительной техники, Нью-Йорк, США, 83–94. [**https://doi.org/10.1145/3437959.3459257**](https://doi.org/10.1145/3437959.3459257) Перейти кцитата 1цитата 2
* Кристофер Д. Карозерс, Дэвид Бауэр и Шон Пирс. 2002. ROSS: высокопроизводительная модульная система Time Warp с малым объемом памяти. *J. Parallel and Distrib. Comput.* 62, 11 (2002), 1648–1669. [**https://doi.org/10.1016/S0743-7315(02)00004-7**](https://doi.org/10.1016/S0743-7315(02)00004-7) Перейти кцитата 1цитата 2цитата 3
* Cy Chan, Bin Wang, John Bachan и Jane Macfarlane. 2018. Mobiliti: Масштабируемое моделирование транспортировки с использованием высокопроизводительных параллельных вычислений. В *2018 году 21-я Международная конференция по интеллектуальным транспортным системам (ITSC)* . IEEE, 634–641. Перейти кцитата 1цитата 2цитата 3
* К. Мани Чанди и Лесли Лампорт. 1985. Распределенные снимки: определение глобальных состояний распределенных систем. *ACM Trans. Comput. Syst.* 3, 1 (февраль 1985), 63–75. [**https://doi.org/10.1145/214451.214456**](https://doi.org/10.1145/214451.214456) Перейти кцитата 1
* ACH Chow и BP Zeigler. 1994. Parallel DEVS: параллельный, иерархический, модульный формализм моделирования. В *трудах Зимней конференции по моделированию* . 716–722. [**https://doi.org/10.1109/WSC.1994.717419**](https://doi.org/10.1109/WSC.1994.717419) Перейти кцитата 1
* Джейсон Коуп, Нин Лю, Сэм Лэнг, Фил Карнс, Крис Карозерс и Роберт Росс. 2011. Коды: Обеспечение совместного проектирования многослойных архитектур хранения exascale. В материалах семинара по новым суперкомпьютерным технологиям, том 2011. ACM. Перейти кцитата 1
* S. R. Das. 2000. Адаптивные протоколы для параллельного дискретного моделирования событий. *Журнал общества операционных исследований* 51, 4 (2000), 385–394. [**http://www.jstor.org/stable/254165**](http://www.jstor.org/stable/254165) Перейти кцитата 1цитата 2
* Самир Р. Дас и Ричард М. Фудзимото. 1997. Адаптивное управление памятью и контроль оптимизма в деформациях времени. *ACM Trans. Model. Comput. Simul.* 7, 2 (апрель 1997), 239–271. [**https://doi.org/10.1145/249204.249207**](https://doi.org/10.1145/249204.249207) Перейти кцитата 1цитата 2
* Ричард Г. Доулинг и Александр Скабардонис. 2008. Уравнения скорости и потока городских артерий для моделей спроса на поездки. В *материалах конференции Совета по транспортным исследованиям* , том 2. Перейти кцитата 1
* Али Экер, Барри Уильямс, Кеннет Чиу и Дмитрий Пономарев. 2019. Управляемый асинхронный GVT: ускорение параллельного дискретного моделирования событий на многоядерных кластерах. В трудах 48-й Международной конференции по параллельной обработке (Киото, Япония) (ICPP '19). Ассоциация вычислительной техники, Нью-Йорк, США, статья 64, 10 страниц. [**https://doi.org/10.1145/3337821.3337927**](https://doi.org/10.1145/3337821.3337927) Перейти кцитата 1
* Джейсон Эванс. 2006. Масштабируемая параллельная реализация malloc (3) для FreeBSD. В *трудах конференции bsdcan, Оттава, Канада* . Перейти кцитата 1
* Алоиз Ферша. 1995. Вероятностное адаптивное прямое оптимистическое управление в деформациях времени. *SIGSIM Simul. Dig.* 25, 1 (июль 1995), 120–129. [**https://doi.org/10.1145/214283.214320**](https://doi.org/10.1145/214283.214320) Перейти кцитата 1цитата 2
* Ричард М. Фудзимото. 1990. Параллельное дискретно-событийное моделирование. *Commun. ACM* 33, 10 (октябрь 1990 г.), 30–53. [**https://doi.org/10.1145/84537.84545**](https://doi.org/10.1145/84537.84545) Перейти кцитата 1
* Ричард М. Фудзимото. 1990. Производительность Time Warp при синтетических рабочих нагрузках. В *трудах SCS Multiconference on Distributed Simulations, 1990* , том 22. 23–28. Перейти кцитата 1цитата 2
* Андреас Хорни, Кай Нагель и Кей Аксхаузен (редакторы). 2016. Мультиагентное транспортное моделирование MATSim. Ubiquity Press, Лондон. 618 страниц. [**https://doi.org/10.5334/baw**](https://doi.org/10.5334/baw) Перейти кцитата 1
* Дэвид Джефферсон и Генри А. Совизрал. 1982. *Быстрое параллельное моделирование с использованием механизма Time Warp: Часть I, Локальное управление* . RAND Corporation, Санта-Моника, Калифорния. Перейти кцитата 1
* Дэвид Р. Джефферсон. 1985. Виртуальное время. *ACM Trans. Program. Язык. Системы.* 7, 3 (июль 1985), 404–425. [**https://doi.org/10.1145/3916.3988**](https://doi.org/10.1145/3916.3988) Перейти кцитата 1цитата 2
* Джордж Карипис и Випин Кумар. 1998. Быстрая и высококачественная многоуровневая схема для разбиения нерегулярных графов. *Журнал SIAM по научным вычислениям* 20, 1 (1998), 359–392. [**https://doi.org/10.1137/S1064827595287997**](https://doi.org/10.1137/S1064827595287997) arXiv: [**https://doi.org/10.1137/S1064827595287997**](https://doi.org/10.1137/S1064827595287997) Перейти кцитата 1
* Лесли Лэмпорт. 1978. Время, часы и порядок событий в распределенной системе. *Commun. ACM* 21, 7 (июль 1978), 558–565. [**https://doi.org/10.1145/359545.359563**](https://doi.org/10.1145/359545.359563) Перейти кцитата 1
* Jonatan Lindén, Pavol Bauer, Stefan Engblom и Bengt Jonsson. 2017. Exposing Inter-Process Information for Efficient Parallel Discrete Event Simulation of Spatial Stochastic Systems. В трудах конференции ACM SIGSIM 2017 года по принципам усовершенствованного дискретного моделирования (Сингапур, Республика Сингапур) (SIGSIM-PADS '17). Association for Computing Machinery, Нью-Йорк, США, 53–64. [**https://doi.org/10.1145/3064911.3064916**](https://doi.org/10.1145/3064911.3064916) Перейти кцитата 1
* Пабло Альварес Лопес, Михаэль Бериш, Лаура Бикер-Вальц, Якоб Эрдманн, Юн-Панг Флеттерод, Роберт Хилбрих, Леонхард Люкен, Йоханнес Руммель, Петер Вагнер и Эвамари Висснер. 2018. Микроскопическое моделирование дорожного движения с использованием SUMO. В *2018 году прошла 21-я Международная конференция по интеллектуальным транспортным системам (ITSC)* . 2575–2582. [**https://doi.org/10.1109/ITSC.2018.8569938**](https://doi.org/10.1109/ITSC.2018.8569938) Перейти кцитата 1
* F. Mattern. 1993. Эффективные алгоритмы для распределенных снимков и приближения глобального виртуального времени. *J. Parallel and Distrib. Comput.* 18, 4 (1993), 423–434. [**https://doi.org/10.1006/jpdc.1993.1075**](https://doi.org/10.1006/jpdc.1993.1075) Перейти кцитата 1
* Джейсон Э. Миллер, Харшад Кастуре, Джордж Куриан, Чарльз Грюнвальд, Натан Бекманн, Кристофер Селио, Джонатан Истеп и Анант Агарвал. 2010. Graphite: распределенный параллельный симулятор для многоядерных процессоров. В *HPCA - 16 2010 Шестнадцатый международный симпозиум по архитектуре высокопроизводительных компьютеров* . 1–12. [**https://doi.org/10.1109/HPCA.2010.5416635**](https://doi.org/10.1109/HPCA.2010.5416635) Перейти кцитата 1
* AC Palaniswamy и PA Wilsey. 1993. Адаптивные ограниченные временные окна в оптимистично синхронизированном симуляторе. В *[1993] Труды Третьего симпозиума Great Lakes по автоматизации проектирования СБИС высокопроизводительных систем СБИС* . 114–118. [**https://doi.org/10.1109/GLSV.1993.224467**](https://doi.org/10.1109/GLSV.1993.224467) Перейти кцитата 1
* AC Palaniswamy и PA Wilsey. 1994. Планирование процессов Time Warp с использованием методов адаптивного управления. В *трудах Зимней конференции по моделированию* . 731–738. [**https://doi.org/10.1109/WSC.1994.717422**](https://doi.org/10.1109/WSC.1994.717422) Перейти кцитата 1цитата 2
* Джошуа Пелки и Джордж Райли. 2011. Распределенное моделирование с MPI в Ns-3. В трудах 4-й Международной конференции ICST по инструментам и методам моделирования (Барселона, Испания) (SIMUTools '11). ICST (Институт компьютерных наук, социальной информатики и телекоммуникационной инженерии), Брюссель, BEL, 410–414. Перейти кцитата 1цитата 2
* Алессандро Пеллегрини и Франческо Куалья. 2014. Глобальные виртуальные вычисления времени без ожидания в системах TimeWarp с общей памятью. В *2014 году IEEE 26-й Международный симпозиум по архитектуре компьютеров и высокопроизводительным вычислениям* . 9–16. [**https://doi.org/10.1109/SBAC-PAD.2014.38**](https://doi.org/10.1109/SBAC-PAD.2014.38) Перейти кцитата 1
* Алессандро Пеллегрини, Роберто Витали и Франческо Куалья. 2009. Di-DyMeLoR: Регистрация только грязных фрагментов для эффективного управления объектами оптимистического моделирования на основе динамической памяти. В *2009 году ACM/IEEE/SCS 23-й семинар по принципам расширенного и распределенного моделирования* . 45–53. [**https://doi.org/10.1109/PADS.2009.24**](https://doi.org/10.1109/PADS.2009.24) Перейти кцитата 1цитата 2
* Алессандро Пеллегрини, Роберто Витали и Франческо Куалья. 2012. Римский оптимистичный симулятор: основные внутренние компоненты и модель программирования. На *4-й Международной конференции ICST по инструментам и методам моделирования* . Перейти кцитата 1цитата 2цитата 3
* Калян Перумалла. 2006. Системный подход к моделированию масштабируемой транспортной сети. На *зимней конференции по моделированию* . Перейти кцитата 1
* Кальян С. Перумалла, Альфред Дж. Парк и Винод Типпараджу. 2011. Алгоритмы GVT и динамика дискретных событий на 129K+ процессорных ядрах. В *2011 г. 18-я Международная конференция по высокопроизводительным вычислениям* . 1–11. [**https://doi.org/10.1109/HiPC.2011.6152725**](https://doi.org/10.1109/HiPC.2011.6152725) Перейти кцитата 1цитата 2
* Кальян С. Перумалла, Альфред Дж. Парк и Винод Типпараджу. 2014. Дискретное выполнение событий с односторонними и двухсторонними алгоритмами GVT на 216 000 процессорных ядер. *ACM Trans. Model. Comput. Simul.* 24, 3, статья 16 (июнь 2014), 25 страниц. [**https://doi.org/10.1145/2611561**](https://doi.org/10.1145/2611561) Перейти кцитата 1цитата 2
* Калян С. Перумалла и Судип К. Сил. 2011. Моделирование дискретных событий и массовое параллельное выполнение явлений эпидемической вспышки. *МОДЕЛИРОВАНИЕ: Труды Общества моделирования и Международного моделирования* 88, 7 (1 2011). [**https://www.osti.gov/biblio/1037625**](https://www.osti.gov/biblio/1037625) Перейти кцитата 1
* Д. М. Рао. [nd]. MUSE: параллельная агентная среда моделирования. [**https://pc2lab.cec.miamioh.edu/muse/**](https://pc2lab.cec.miamioh.edu/muse/) . [Онлайн; дата обращения 11 февраля 2024 г.]. Перейти кцитата 1
* Dhananjai M. Rao. 2018. Сравнение производительности MPI с поддержкой Cross Memory Attach и многопоточных оптимистичных параллельных симуляций. В материалах конференции ACM SIGSIM 2018 года по принципам усовершенствованного дискретного моделирования (Рим, Италия) (SIGSIM-PADS '18). Ассоциация вычислительной техники, Нью-Йорк, США, 37–48. [**https://doi.org/10.1145/3200921.3200935**](https://doi.org/10.1145/3200921.3200935) Перейти кцитата 1
* A. F. Rodrigues, K. S. Hemmert, B. W. Barrett, C. Kersey, R. Oldfield, M. Weston, R. Risen, J. Cook, P. Rosenfeld, E. Cooper-Balis и B. Jacob. 2011. Набор инструментов для структурного моделирования. *SIGMETRICS Perform. Eval. Rev.* 38, 4 (март 2011), 37–42. [**https://doi.org/10.1145/1964218.1964225**](https://doi.org/10.1145/1964218.1964225) Перейти кцитата 1
* Vinay Sachdev, M. Hybinette и E. Kraemer. 2004. Управление чрезмерным оптимизмом в режиме time-warp с помощью управления потоком на базе CPU. В *материалах Зимней конференции по моделированию 2004 г., 2004 г.* , том 1. 410. [**https://doi.org/10.1109/WSC.2004.1371342**](https://doi.org/10.1109/WSC.2004.1371342) Перейти кцитата 1цитата 2
* Daniel Sanchez и Christos Kozyrakis. 2013. ZSim: Быстрое и точное микроархитектурное моделирование тысячеядерных систем. На *Международном симпозиуме по компьютерной архитектуре* . 475–486. [**https://doi.org/10.1145/2485922.2485963**](https://doi.org/10.1145/2485922.2485963) Перейти кцитата 1
* Ричард Т. Норрис, Сандра Л. Гленн и Джуд Т. Зоммерфельд. ​​1990. Дискретно-событийное моделирование в очистке сточных вод. *Журнал «Экологическая наука и здоровье». Часть A: Экологическая наука и инженерия и токсикология* 25, 4 (1990), 407–423. [**https://doi.org/10.1080/10934529009375567**](https://doi.org/10.1080/10934529009375567) Перейти кцитата 1
* Нандакишор Санти, Стефан Эйденбенц и Джейсон Лю. 2015. Концепция Simian: параллельное дискретно-событийное моделирование с интерпретируемыми языками и компиляцией «точно в срок». На *Зимней конференции по моделированию (WSC) 2015 года* . 3013–3024. [**https://doi.org/10.1109/WSC.2015.7408405**](https://doi.org/10.1109/WSC.2015.7408405) Перейти кцитата 1
* Markus Schordan, Tomas Oppelstrup, David Jefferson, Peter D. Barnes и Dan Quinlan. 2016. Автоматическая генерация обратимого кода C++ и его производительность в масштабируемом кинетическом приложении Монте-Карло. В материалах конференции ACM SIGSIM 2016 года по принципам усовершенствованного дискретного моделирования (Банф, Альберта, Канада) (SIGSIM-PADS '16). Ассоциация вычислительной техники, Нью-Йорк, США, 111–122. [**https://doi.org/10.1145/2901378.2901394**](https://doi.org/10.1145/2901378.2901394) Перейти кцитата 1
* Лиза Сокол, Брайан К. Стаки и Винсент Шан-Шоук Хванг. 1989. MTW: Механизм управления для параллельного дискретного моделирования. В *трудах Международной конференции по параллельной обработке, ICPP '89, Университет штата Пенсильвания, Университет Парк, Пенсильвания, США, август 1989 г. Том 3: Алгоритмы и приложения* . Издательство Университета штата Пенсильвания, 250–254. Перейти кцитата 1цитата 2
* Тапас К. Сом и Роберт Г. Сарджент. 1998. Вероятностная политика планирования событий для оптимистического параллельного дискретного моделирования событий. *SIGSIM Simul. Dig.* 28, 1 (июль 1998), 56–63. [**https://doi.org/10.1145/278009.278016**](https://doi.org/10.1145/278009.278016) Перейти кцитата 1цитата 2
* S. Srinivasan и PF Reynolds. 1995. Алгоритмы адаптивной синхронизации NPSI для PDES. В *трудах зимней конференции по моделированию, 1995.* 658–665. [**https://doi.org/10.1109/WSC.1995.478841**](https://doi.org/10.1109/WSC.1995.478841) Перейти кцитата 1цитата 2
* Сенг Чуан Тай, Йонг Мэн Тео и Сью Тхенг Конг. 1997. Спекулятивное параллельное моделирование с адаптивной схемой дроссельной заслонки. *SIGSIM Simul. Dig.* 27, 1 (июнь 1997), 116–123. [**https://doi.org/10.1145/268823.268909**](https://doi.org/10.1145/268823.268909) Перейти кцитата 1цитата 2
* Роберто Токкачели и Франческо Куалья. 2008. DyMeLoR: Динамическая библиотека регистратора и восстановителя памяти для объектов оптимистического моделирования с общей схемой памяти. В *2008 году 22-й семинар по принципам расширенного и распределенного моделирования* . 163–172. [**https://doi.org/10.1109/PADS.2008.23**](https://doi.org/10.1109/PADS.2008.23) Перейти кцитата 1цитата 2
* Томмазо Точчи, Алессандро Пеллегрини, Франческо Куалья, Хосеп Касановас-Гарсия и Тойотаро Судзумура. 2017. ORCHESTRA: Асинхронный распределенный алгоритм GVT без ожидания. В *2017 году IEEE/ACM 21-й Международный симпозиум по распределенному моделированию и приложениям реального времени (DS-RT)* . 1–8. [**https://doi.org/10.1109/DISTRA.2017.8167666**](https://doi.org/10.1109/DISTRA.2017.8167666) Перейти кцитата 1
* SJ Turner и MQ Xu. 1992. Оценка производительности ограниченного алгоритма Time Warp. В *трудах SCS Multiconference on Distributed Simulation* . 117–126. Перейти кцитата 1
* Цзюнь Ванг и Карл Троппер. 2007. Оптимизация моделирования искажения времени с помощью методов обучения с подкреплением. На *Зимней конференции по моделированию 2007 г.* 577–584. [**https://doi.org/10.1109/WSC.2007.4419650**](https://doi.org/10.1109/WSC.2007.4419650) Перейти кцитата 1
* Цзюнь Ванг и Карл Троппер. 2009. Использование генетических алгоритмов для ограничения оптимизма в Time Warp. В *материалах Зимней конференции по моделированию 2009 года (WSC)* . 1180–1188. [**https://doi.org/10.1109/WSC.2009.5429634**](https://doi.org/10.1109/WSC.2009.5429634) Перейти кцитата 1
* Барри Уильямс, Али Экер, Кеннет Чиу и Дмитрий Пономарев. 2021. Высокопроизводительные PDES на многоядерных кластерах. В материалах конференции ACM SIGSIM 2021 года по принципам усовершенствованного дискретного моделирования (виртуальное мероприятие, США) (SIGSIM-PADS '21). Ассоциация вычислительной техники, Нью-Йорк, США, 153–164. [**https://doi.org/10.1145/3437959.3459252**](https://doi.org/10.1145/3437959.3459252) Перейти кцитата 1
* PA Wilsey. [nd]. Ядро моделирования деформированных объектов. [**https://github.com/wilseypa/warped2**](https://github.com/wilseypa/warped2) . [Онлайн; дата обращения 11 февраля 2024 г.]. Перейти кцитата 1
* B. P. Zeigler, J. H. Hu и J. W. Rosenblit. 1989. Иерархическое, модульное моделирование в схеме DEVS. В трудах 21-й конференции по зимнему моделированию (Вашингтон, округ Колумбия, США) (WSC '89). Ассоциация вычислительной техники, Нью-Йорк, США, 84–89. [**https://doi.org/10.1145/76738.76749**](https://doi.org/10.1145/76738.76749) Перейти кцитата 1

СНОСКА

[**⁎**](https://dl.acm.org/doi/fullHtml/10.1145/3615979.3656061#foot-fn1) Автор-корреспондент: Тан Нгуен, TanNguyen@lbl.gov, (510) 486-5518, 1 Cyclotron Road MS:59R4010A, Беркли, Калифорния 94720

Изображение лицензии CC-BY  
Данная работа лицензирована в соответствии с [**лицензией Creative Commons Attribution International 4.0**](https://creativecommons.org/licenses/by/4.0/) .

*SIGSIM PADS '24, 24–26 июня 2024 г., Атланта, Джорджия, США*

© 2024 Авторские права принадлежат владельцу/автору(ам).  
ACM ISBN 979-8-4007-0363-8/24/06.  
DOI: [**https://doi.org/10.1145/3615979.**](https://doi.org/10.1145/3615979.3656061)