DOI: 10.5862/JCSTCS.236.6
УДК 004.4`233:004.382.2

А.В. Левченко, С.А. Фёдоров

ИНСТРУМЕНТ ДИНАМИЧЕСКОГО ДВОИЧНОГО АНАЛИЗА ПРОСТРАНСТВЕННО-ВРЕМЕННОЙ ЛОКАЛИЗАЦИИ ПРИЛОЖЕНИЯ

A.V. Levchenko, S.A. Fyodorov

DYNAMIC BINARY INSTRUMENTATION TOOL FOR DATA LOCALITY ANALYSIS

Наличие проблемы стены памяти (Memory Wall Problem) приводит к простоям процессора в ожидании данных из памяти в результате неудовлетворительной пространственно-временной локализации приложения. Средства динамического двоичного анализа важны для оценки пространственно-временного поведения приложения с целью смягчения последствий этой проблемы. Для исследования эффективности приложения необходимо получить исчерпывающий профиль доступа приложения к памяти, позволяющий дать численную и визуальную оценку локализации доступа. Процесс профилирования имеет существенные издержки, возникающие при низкоуровневом инструментировании и сохранении данных профиля, способные исказить оценку реальной локализации. Описан программный инструмент для динамического двоичного анализа пространственно-временной локализации приложения, ориентированный на практическое применение методологии Apex для глубокого анализа приложений. Инструмент профилирования tslmap разработан на основе платформы Valgrind, доступной на большинстве современных высокопроизводительных платформ. Показано, что разработанный инструмент позволяет корректно оценить локализацию приложения, с минимальными издержками профилирования получить и визуализировать профили на архитектурах с распределенной памятью.

ВРЕМЕННАЯ ЛОКАЛИЗАЦИЯ; ВЫСОКОПРОИЗВОДИТЕЛЬНЫЕ ВЫЧИСЛЕНИЯ; ДИНАМИЧЕСКИЙ ДВОИЧНЫЙ АНАЛИЗ; ПРОБЛЕМА СТЕНЫ ПАМЯТИ; ПРОСТРАНСТВЕННАЯ ЛОКАЛИЗАЦИЯ; ИНСТРУМЕНТ ПРОФИЛИРОВАНИЯ ПРИЛОЖЕНИЙ.

This work addresses the fundamental challenges covered in a number of papers on high performance computing. One of the most important issues is the Memory Wall problem. Standard memory access patterns can cause sparse temporal and spatial locality and thus performance degradation. It is important to get the profile of an application's memory access to analyze its performance. The main purpose of this study is the dynamic binary analysis tool for data locality analysis. The tslmap profiler was developed for analyzing the performance behavior of applications. It is based on the Valgrind platform intended for creating tools for binary analysis of applications. The tslmap tool allows to get and visualize the profiles helping to measure the metrics of temporal and spatial locality by dynamic analysis of a program's memory references. Experiments were performed on cluster architectures for plotting the performance of client application as a function of its spatial and temporal scores for certain systems.

APEX-MAP; DYNAMIC BINARY ANALYSIS; HIGH PERFORMANCE COMPUTING; MEMORY WALL PROBLEM; PERFORMANCE SURFACE; PROFILER; SPATIAL LOCALITY; TEMPORAL LOCALITY; VALGRIND.

Важнейшим вопросом современных высокопроизводительных вычислений с использованием архитектуры с распределенной памятью является низкая реальная производительность системы по сравнению с ее пиковой производительностью. Доминирующей причиной деградации производительности остается *проблема стены памяти* (Memory Wall Problem), которая заключается в недостаточно быстром вы-

полнении операций в оперативной памяти по сравнению со временем выполнения арифметико-логических операций процессорами. Масштабы этой проблемы растут за счет применения современных высокопроизводительных машин с глубокой иерархией памяти. В результате непонимание особенностей пространственно-временного поведения приложения приводит к непредсказуемым издержкам операций с памятью. Одним из способов анализа проблемы стены памяти является оценка пространственной и временной локализации (ПВЛ) обращений анализируемого приложения к памяти. К сожалению, на сегодняшний день отсутствует свободное программное обеспечение для корректной оценки и визуализации ПВЛ, а издержки такого анализа на высокопроизводительных системах не определены.

На основании требований, сформулированных в настоящей работе, разработан профилировщик ПВЛ *tslmap*. Этот инструмент осуществляет динамический двоичный анализ (Dynamic Binary Analysis – DBA) операций доступа к оперативной памяти посредством функций низкоуровневого инструментирования, предоставляемых ядром платформы Valgrind, доступной на подавляющем большинстве современных суперкомпьютеров.

Проведено сравнение издержек профилирования для современных средств DBA и показано, как получение данных, относящихся исключительно к работе анализируемого приложения, позволяет оценить его ПВЛ точнее и добиться существенного снижения доминирующих издержек профилирования.

Предпосылки разработки tslmap

Профилировщик tslmap, который является целью данной статьи, основан на практическом применении методологии Apex [1, 2], используемой сегодня для анализа производительности приложений современных высокопроизводительных систем. Для построения профиля и оценки ПВЛ приложения в настоящей работе используется подход, предложенный авторами Apex. *Пространственная локализация* (Spatial Locality – SL) определяется, как тенденция приложений обращаться к тем адресам в памяти, которые находятся рядом с другими, недавно использовавшимися адресами. *Временная локализация* (Temporal Locality – TL) определяется, как тенденция приложения обращаться к тем же адресам в памяти, к которым это приложение обращалось в последнее время [3–5].

В качестве примера на рис. 1 представлен профиль региона памяти кластера, ха-

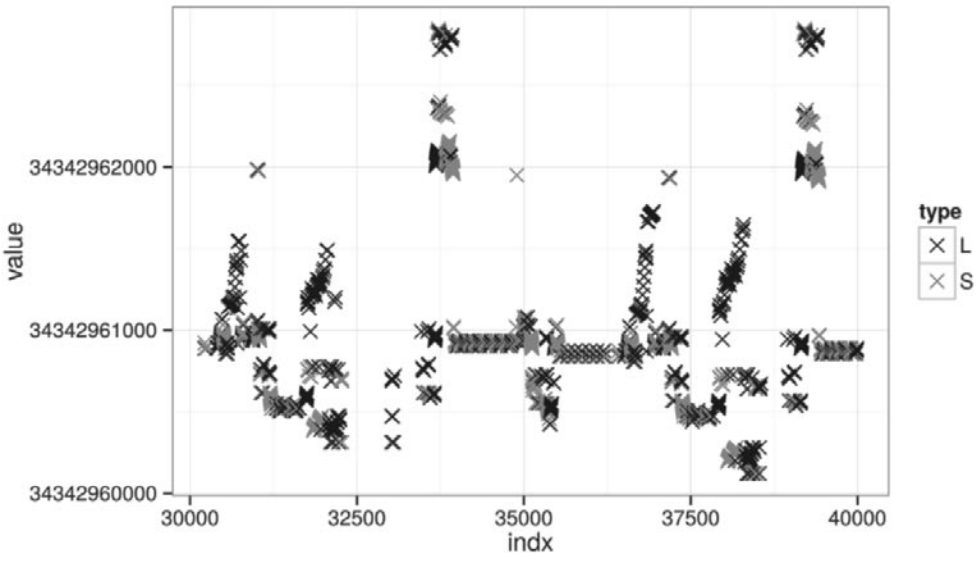

Рис. 1. Профиль региона памяти с неудовлетворительной локализацией

рактеризующийся неудовлетворительной ПВЛ обращений к памяти (этот профиль приложения получен с помощью профилировщика tslmap). Здесь обращение к регионам памяти происходит непоследовательно. При таком многократном обращении к данным приложения могут чаще наступать кэш-промахи, что будет приводить к снижению производительности и являться причиной неэффективного использования оборудования [6].

Для получения подобной картины работы приложения требуются средства профилирования программ, учитывающие особенности конкретной операционной системы, чтобы получить адекватный (лишенный избыточных данных) профиль приложения, в котором собраны обращения к памяти только конкретного приложения. На практике подбор подходящего инструмента профилирования представляет проблему. В [1] в качестве профилировщика использован разработанный в 2001 г. программный инструмент MetaSim, позволявший получить динамический профиль доступа приложений к памяти. Позднее в работе [7], посвященной издержкам динамического инструментирования на примере MetaSim, была проанализирована проблема больших временных затрат на сбор данных и проблема сохранения и анализа недопустимо большого формата выходного файла.

В настоящее время использовать MetaSim не представляется возможным по причине ряда существенных отличий современных высокопроизводительных систем от компьютеров, использовавшихся разработчиками Apex. В связи с этим необходимо было разработать профилировщик. Такая программа уже была ранее разработана на основе Dyninst API в 2013 г. в Санкт-Петербургском государственном политехническом университете в группе С.А. Фёдорова [5]. В данной работе были получены профили и дана оценка ПВЛ ряда приложений. К сожалению, не был подтвержден опыт исследования [7], уже показавший недостатки Dyninst API с точки зрения разработки подобных профилировщиков в связи с избыточным количеством данных при инструментировании, которые искажают реальную оценку ПВЛ и порождают неадекватные (гигантские по размеру) профили для относительно небольших приложений. Таким образом, предварительные исследования показали отсутствие доступного свободного программного обеспечения для анализа ПВЛ, и было решено создать профилировщик, способный выполнить эту задачу. Важнейшие требования, предъявляемые к такому профилировщику, сформулированы ниже.

Функциональные требования

Задача оценки пространственно-временного поведения исследуемого приложения требует построения профиля в виде журнала адресов, по которым это приложение осуществляет доступ в оперативную память. На основании этого были определены следующие требования к программному инструменту:

1. Построение профиля ПВЛ непосредственно во время выполнения приложения: как для всего приложения, так и для отдельных регионов памяти на основе символьной информации. Вывод должен содержать исчерпывающий журнал отслеживания обращений к памяти для отдельной функции или всего приложения.

2. Оценка показателей SL и TL «на лету», используя полученную информацию о доступе к памяти.

Входным форматом данных должен быть файл клиентского приложения, имеющий исполняемый и компонуемый формат (ELF), используемый в современных UNIX-подобных операционных системах. Выходной формат данных должен содержать полный журнал доступа клиентского приложения к оперативной памяти с маркировкой типа доступа для каждой ячейки: сохранение (*load*) либо загрузка (*store*).

Для наблюдения за естественной работой приложения наиболее применимым подходом является динамический двоичный анализ, подразумевающий инструментирование клиентского приложения в процессе его выполнения, т. е. внесение анализирующего кода в двоичные данные исполняемого ELF-файла. Анализирующий код включается в процесс выполнения про-

граммы, не вызывая изменений в обычном поведении программы, но выполняя сторонние функции. Динамическое двоичное инструментирование сложно в реализации, поэтому было решено разработать профилировщик на основе одной из платформ (библиотек) для создания инструментов динамического двоичного анализа.

Наиболее развитыми API для создания средств динамического двоичного анализа для операционных систем Linux являются PerfSuite, PIN, Dyninst и Valgrind. В табл. 1 обобщены рассмотренные платформы и критерии выбора. Первое, что необходимо отметить: без счетчика адресов затруднительно посчитать обращения к памяти, поэтому сразу был отброшен вариант разработки инструмента профилирования на основе PerfSuite. Следующий вызов – это издержки инструментирования, избыточные данные в профиле, не относящиеся непосредственно к естественному поведению профилируемого приложения. Избыточные данные искажают реальные результаты пространственной и временной локализации, существенно замедляют профилирование и порождают профиль избыточного размера.

Авторы работы [7], сопоставив эти издержки для PIN и Dyninst API, показали, что объем аналитического кода при инструментировании с использованием Dyninst API примерно в два раза превышает показатели PIN и делает использование Dyninst менее выгодным для профилирования больших задач.

Мы частично используем результаты работы [5], в которой применялся доступный профилировщик на основе Dyninst API, как наиболее актуальные и воспроизводимые из имеющихся результатов. Для снижения объемов производимого профиля и временных издержек анализа было предложено профилировать не только приложение целиком, но и отдельные функции на основе символьной информации, что не было предусмотрено при разработке профилировщика в работе [5]. По указанным выше причинам, а также по причинам, связанным с особенностями лицензирования (которые не были показаны в таблице), предпочтение было отдано платформе Valgrind.

Valgrind – это платформа динамического двоичного инструментирования, предназначенная для создания инструментов динамического двоичного анализа. Значительным отличием Valgrind является то, что в ходе анализа инструментируется каждая инструкция, и информация о состоянии программы передается между частями инструментирующего кода на низком, локальном уровне. Valgrind широко используется в области высокопроизводительных вычислений в последние годы, главным образом для анализа использования памяти MPI-приложениями [8, 9].

Цель настоящей работы – программный инструмент динамического двоичного анализа на базе Valgrind для оценки пространственно-временной локализации приложения. На основе сформулирован-

Таблица 1

Сравнение API для разработки средств динамического двоичного анализа

Возможности	Valgrind	PerfSuite	Dyninst	PIN
Поддержка POSIX	Да	Да	Да	Да
Многопоточность	Да	Да	Да	Да
Поддержка MPI	Да	Да	Да	Да
Счетчик адресов	Да	Нет	Да	Да
Поддержка C ABI	Да	Да	Нет	Да
JIT компилятор	Да	Нет	Нет	Да
Независимость от glibc	Да	Нет	Нет	Да
Профилирование на основе символьной информации	Нет	Нет	Нет	Да

ных требований разработан профилировщик. Особенности работы профилировщика приведены далее.

Введение в tslmap

Программный инструмент tslmap (temporal and spatial locality memory access probe) – это разработанный на основе Valgrind профилировщик Linux-приложений, созданный для исследования пространственной и временной локализации доступа к оперативной памяти в соответствии с методологией Apex. Профилировщик позволяет построить профиль ПВЛ как для всего приложения, так и для отдельных регионов памяти на основе символьной информации и вычислить показатели пространственной и временной локализации для анализируемого приложения. Вывод осуществляется в текстовый лог или на сетевой порт, что дает существенный выигрыш в скорости сохранения профилей при работе с высокопроизводительной базой данных. Преимущества tslmap как инструмента на основе Valgrind состоят в следующем:

1. В лог трассировки не попадают системные вызовы и другие действия ядра Linux (игнорируются операции планирования и обработки сигналов); ядро Valgrind заменяет небольшую часть кода собственным кодом в целях инструментирования. Этот код также не попадает в профиль приложения, что значительно снижает издержки хранения больших профилей.

2. Фиксируются почти все операции обращения к памяти, выполняемые непосредственно изучаемым приложением. Достигается высокая детализация информации о доступе клиентского приложения к памяти. Визуализация тестового журнала отслеживания осуществляется с помощью сценария на языке R. На рис. 2 показан пример полученного профиля приложения.

Профилировщик является свободным программным обеспечением, что гарантирует его высокую доступность и расширяемость.

Данные возможности позволяют существенно сократить время профилирования, визуализации и сохранения профиля, получая только необходимую информацию о работе приложения.

Архитектура tslmap

Программный инструмент tslmap является модулем, взаимодействующим с ядром платформы Valgrind – VEX IR. Ядро предоставляет базовый механизм запуска, управления и инструментирования клиентских приложений.

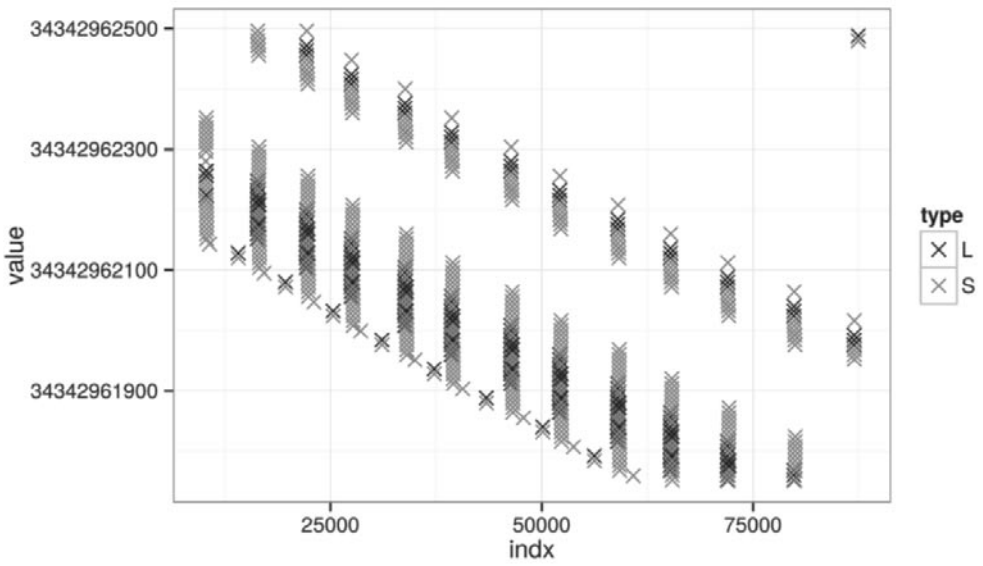

Рис. 2. Профиль приложения перемножения массивов

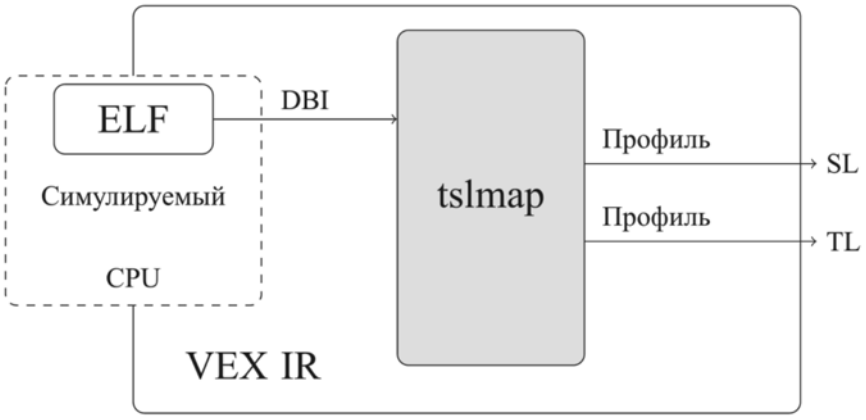

Рис. 3. Выполнение tslmap

Плагин tslmap является динамической библиотекой, инициализируемой ядром VEX IR при старте Valgrind посредством введения команд в командной строке Linux. Процесс выполнения tslmap представлен на рис. 3. Клиентское приложение выполняется на симулируемом VEX IR процессоре. Valgrind использует динамическую компиляцию: tslmap внедряется в процесс клиентского приложения в процессе JIT компиляции, включающем в себя дизассемблирование двоичного кода в язык промежуточного выполнения, который инструментируется с помощью tslmap и затем компилируется обратно. После динамического двоичного инструментирования (DBI) строится профиль клиентского приложения, и осуществляется вычисление SL и TL.

Преимущество такой архитектуры состоит в том, что динамическая компиляция и инструментирование ядром VEX IR являются высокоскоростными операциями и не оказывают существенного влияния на работу анализируемого приложения, что позволяет анализировать ПВЛ больших приложений, получая только ту информацию об их работе, которая необходима в конкретном случае. Это, в свою очередь, дает возможность исследовать приложения, работающие в нормальном режиме на высоконагруженных системах, не производя отдельных запусков.

Код плагина включает в себя базовые функции модуля, а как инструмент профилирования – вычислительные функции для оценки ПВЛ. Ниже описана работа двух основных функций в терминах методологии Apex.

Оценка ПВЛ

Оценку пространственной локализации можно представить в виде суммы, где $stride_i$ обозначает долю от общего объема операций динамической памяти, имеющих шаг i. Особенность такого подхода состоит в генерации нормированной оценки в диапазоне [0, 1], что обратно пропорционально средней длине шага. Приложение, которое выполняет только один шаг, получает один балл. Приложение, которое выполняет только два шага, получает оценку пять, а приложение без шагов получает оценку ноль. С целью уменьшения времени выполнения алгоритма индекс суммирования в формуле ограничивают некоторой величиной i max. Выбирать очень большое значение этой величины не имеет особого смысла, т. к. каждое последующее слагаемое стремительно уменьшается [1].

В качестве входных данных используется текущий адрес доступа к памяти, размер окна адресов $W = 32$, к которым доступ уже был осуществлен, i max – ограничение шага (массив шагов размером i max).

Дистанция между текущим адресом и адресами окна вычисляется для всех адресов доступа, и если дистанция больше или

равна i max, элемент в массиве шагов с индексом полученного расстояния увеличивается на единицу. После обработки всех адресов вычисляется значение пространственной локализации (SL) по формуле:

$$\mathrm{SL} = \sum_{i=1}^{\infty} \frac{stride_i}{i},$$

где $stride_i$ — доля обращений к памяти с длиной шага i [1].

Временная локализация – это тенденция приложения осуществлять доступ к регионам в памяти, к которым оно уже недавно обращалось. Дистанция повторного использования (*reuse distance*) для адреса в памяти A – это число уникальных номеров в памяти, к которыми приложение обращалось с момента последнего доступа к A. Временная локализация (TL) приложения вычисляется по формуле:

$$\mathrm{TL} = \sum_{i=0}^{\log(N)-1} \frac{((reuse_{2^{i+1}} - reuse_{2^i}) \cdot (\log_2(N) - i))}{\log_2(N)},$$

где $reuse_i$ обозначает долю операций с динамической памятью с дистанцией повторного использования меньшей, либо равной i [1].

В качестве входных данных используется текущий адрес доступа к памяти и размер максимально рассматриваемого расстояния повторного использования N. Вывод профилировщика содержит трассировку доступа к памяти и оценку ПВЛ клиентского приложения.

Экспериментальные запуски

Цель и задачи экспериментальных запусков. Цель экспериментальных запусков состоит в проверке выполнения профилировщиком tslmap требований, изложенных выше в данной статье. Задачи экспериментальных запусков:

1. Построить и визуализировать профиль ПВЛ MPI-приложения перемножения матриц, выполняющегося на вычислительном кластере. Вычислить показатели ПВЛ приложений SL и TL и сопоставить их с результатами существующих исследований. На данном этапе показатели ПВЛ (SL и TL) будут сопоставлены с результатами имеющегося аналогичного профилировщика на основе Dyninst API, ранее представленного в работе [5], – единственного доступного профилировщика на основе методологии Apex.

2. Оценить избыточность данных в профилях всего приложения и отдельной функции, полученных с помощью tslmap и аналогичного профилировщика.

На разных стадиях экспериментальных запусков использовались разные анализируемые приложения. Выбор приложения для первой задачи (оценка ПВЛ) обусловлен наличием ранее полученных в [5] результатов SL и TL для инструмента на основе Dyninst API, что дает возможность сравнить их с результатами данной работы. С другой стороны, результаты работы [7] свидетельствуют о существенном снижении данных в профиле, полученном с помощью PIN, по сравнению с Dyninst API при анализе тестов NAS Parallel Benchmark. Поэтому на второй стадии экспериментальных запусков (оценка избыточности данных в профиле) для сравнения используется PIN и задача NAS Parallel Benchmark. Таким образом, предполагается подтверждение преимуществ tslmap в двух задачах экспериментальных запусков.

Оценка ПВЛ. Актуальность первой задачи состоит в исследовании пространственно-временного поведения MPI-программы на кластере: визуализации профиля и оценки локализации. Параллельные программы могут иметь ошибки параллелизма, включающие в себя множественный доступ на множестве взаимосвязанных переменных, что в сложных случаях приводит к непредсказуемому пространственно-временному поведению приложения и к неадекватному снижению производительности. Подобные ошибки могут существенно исказить данные, полученные при профилировании по причине непредсказуемости локализации доступа к памяти [10].

Для поддержки эксперимента использовался кластер «Триптих-2», включающий в себя вычислительные узлы IBM SYSTEM x3300 M4 и IBM SYSTEM x3650 M4 под управлением контроллера кластера.

Перемножение элементов матриц и векторов – одна из наиболее часто встречающихся задач в вычислительных приложениях. Можно предположить, что, поскольку в программе происходит обращение к элементам матрицы по столбцам, пространственная локализация (SL) будет, по крайней мере, ниже 0,5, как известно из работы [5]. Временная локализация (TL) будет выше среднего, т. к. происходит многократное обращение к одним и тем же элементам. Визуализация профиля данного приложения представлена на рис. 4.

В результате профилирования данного приложения с помощью tslmap получена следующая оценка локализации: SL = 0,25, TL = 0,65. В данном случае пространственная локализация оказалась несколько лучше результатов, полученных ранее с помощью инструмента на основе Dyninst API (ранее в [5] было получено SL = 0,336, TL = 0,68). В первую очередь это объясняется разницей содержимого построенных профилей: профиль Dyninst API был насыщен избыточным количеством не относящихся непосредственно к работе приложения служебных операций (инструкций инструментирования). Разреженный характер операций обращения к памяти с ненулевым шагом объясняет искажение реальной картины пространственной локализации. Потенциально в случае с разработанным ранее на основе Dyninst API профилировщиком такие лишние данные могли достигать до 50 % содержимого получаемого профиля для больших приложений в связи с особенностями инструментирования Dyninst API.

В случае с tslmap избыточные данные в профиле удается отсечь. Это становится возможным за счет особенностей низкоуровневого инструментирования, которые предоставляет ядро VEX IR платформы Valgrind, а также за счет маркировки каждого обращения к памяти с последующим удалением инструкций в выводе плагина. В результате в полученном с помощью tslmap профиле содержатся только обращения к памяти типа S (*store*) и L (*load*), сделанные именно анализируемым клиентским приложением. Значение SL, полученное на основе уже «очищенного» профиля, является примером наиболее близкой оценки пространственного поведения данного приложения при выполнении на данном экспериментальном макете. Значение SL может несколько улучшаться по мере отключения узлов кластера, на которых за-

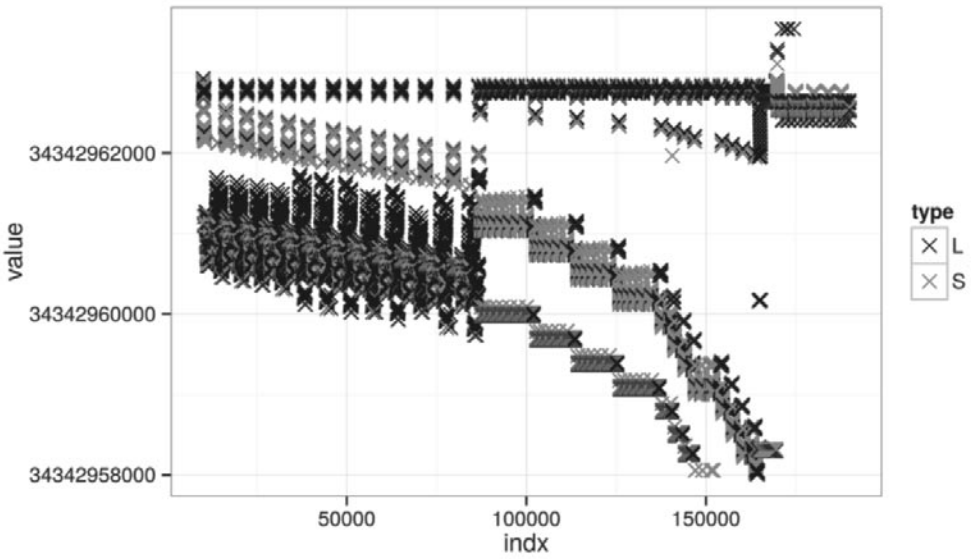

Рис. 4. Приложение перемножения матриц, выполняемое на трех узлах опытного кластера

пускается анализируемое приложение. При запуске лишь на одном узле полученное значение SL приближается к 0,15, что объясняется, в первую очередь, особенностями производительности сети кластера при использовании нескольких узлов, а также ранее отмеченным значительным количеством инструкций инструментирования при использовании Dyninst API.

Оценка избыточности данных в профилях. Первая стадия экспериментальных запусков показала, что значения SL и TL близки к предполагаемым теоретическим результатам и к ранее полученным результатам для профилировщика на основе Dyninst API [5]. На второй стадии экспериментальных запусков необходимо проанализировать полученный профиль. Поскольку оценка ПВЛ весьма чувствительна к наличию в профиле посторонней информации о доступе к памяти, который мог быть осуществлен не только анализируемой программой, важно было сопоставить показатели отношения строк профиля к количеству строк кода для одной и той же функции, одного и того же приложения, но для разных профилировщиков.

Для сравнения характеристик профилей, полученных с помощью tslmap и PIN, применять профилировщик на основе Dyninst API нецелесообразно. Во-первых, данный профилировщик не поддерживает профилирование отдельно взятых функций на основе символьной информации, что не позволяет выделить в профиле только обращения, связанные с работой данной функции. Во-вторых, как было сказано выше, ранее в работе [7] уже были выявлены преимущества PIN с точки зрения компактности полученного профиля по сравнению с Dyninst API. И если на первой стадии исследования нам были важны только его результаты оценки ПВЛ, то на второй стадии потребуется другой инструмент для сравнения.

Сравнение характеристик профилирования проводилось с использованием пакета тестов NAS Parallel Benchmark версии NPB3.3-MPI, который используется для комплексной оценки производительности вычислительных систем и для экспериментальных запусков профилирования [5, 11]. Для экспериментального запуска использовался тест IS (Integer Sort, сортировка целых чисел) с минимальным классом вычислительной сложности A для мультипроцессорных систем, что объясняется отсутствием необходимости проводить комплексную оценку производительности. Исследование размерности профиля имеет две основные задачи:

1. Сравнение размеров профиля, полученных посредством tslmap и PIN, для определения наличия избыточных данных в профиле.

2. Сравнение отношения количества строк профиля (KLoP) к количеству строк исходного кода (LoC) как для всего приложения (IS), так и для отдельной его функции (*find_my_seed*) на основе символьной информации, с целью определить достаточность данных в профиле.

В табл. 2 представлены результаты сравнения.

Сравнительный подход при оценке показателя KLoP/LoC для отдельной функции и для всей программы для каждого из профилировщиков основан на идее, что избыточное количество внедряемого при инструментировании аналитического кода

Таблица 2

Сравнение параметров профиля всего приложения и отдельной функции

Тип профиля	tslmap/Valgrind		PIN	
	IS (*find_my_seed*)	IS (*full*)	IS (*find_my_seed*)	IS (*full*)
LoC	43	1085	43	1085
KLoP	103 729	2 829 680	147 409	3 819 200
KLoP/LoC	2 412	2 608	3 428	3 520
GboP	1,4	38,8	2,0	52,3

повлияет на общий объем профиля. В данном случае пропорции между показателем KLoP/LoC отдельной функции и всей программы близки для сравниваемых профилировщиков. Однако в случае с tslmap удалось ограничить попадание избыточных данных инструментирования в профиль как за счет особенностей инструментирования, предоставляемых ядром VEX IR по умолчанию, так и за счет удаления маркированных обращений, связанных с инструментирующими инструкциями, на этапе вывода профиля. В результате общий показатель размера для профиля (Gigabyte of Profile – GboP), полученного с помощью tslmap, составил 38,8 против 52,3 для PIN.

Важно отметить, что при профилировании больших приложений в первую очередь будет необходимо получать профиль конкретных функций. Получение профиля отдельной функции существенно снижает временные издержки на построение профиля и его визуализацию, когда это необходимо. А при планируемом дальнейшем массовом автоматизированном анализе приложений можно будет достичь существенной экономии при хранении в базе данных профилей значительного размера.

Оценка временных издержек профилирования с помощью tslmap. Оценка временных издержек профилирования производилась с помощью стандартной утилиты GNU time. Выполнение приложения перемножения матриц заняло 2,4 с без профилирования с помощью tslmap и 4,9 с с профилированием. Для некоторых задач пакета NAS Parallel Benchmark продолжительность выполнения при профилировании увеличивалась в три раза. Тем не менее полученные значения временных издержек являются сравнительно невысокими для инструмента на основе Valgrind. Так, ранее для стандартных инструментов на основе Valgrind (Memcheck, Cachegrind) продолжительность анализа возрастала в 10 раз и выше для ряда задач пакета SPEC CPU2000. В случае tslmap незначительное количество связанных с расчетом SL и TL операций обусловливает сравнительно низкие временные затраты анализа. Возможность профилирования отдельных регионов памяти на основе символьной информации существенно снижает продолжительность исследования.

Выводы по экспериментальным запускам. Получены следующие результаты экспериментальных запусков:

1. Программный инструмент tslmap позволяет с приемлемыми временными издержками построить профиль ПВЛ клиентского MPI-приложения, выполняемого на кластере, как для всего приложения, так и для отдельного региона памяти на основании символьной информации.

2. Полученные результаты ПВЛ ряда приложений соответствуют ожидаемым результатам и результатам более ранних исследований.

3. Экспериментальный профиль содержит полную информацию о доступе к памяти при сравнительно небольшом размере выходного файла.

Использование tslmap можно считать успешным, т. к. профилировщик позволяет получить более точную оценку ПВЛ по сравнению с известными аналогами. Важным итогом можно считать то, что удалось снизить издержки, связанные со временем профилирования, за счет профилирования отдельных регионов памяти, и снизить размер полученного профиля за счет маркировки и удаления обращений, не имеющих отношения к работе анализируемой программы. Значительное снижение размеров профиля расширяет спектр приложений, которые можно анализировать. Указанные особенности являются существенными преимуществами tslmap по сравнению с доступным профилировщиком ПВЛ [5].

Полученные в работе результаты позволяют проводить дальнейшие исследования на основе методологии Apex. Результаты оценки ПВЛ простых приложений показали корректность работы профилировщика. После этого можно проводить тестирование всех задач пакета NAS Parallel Benchmark. Задачи пакета NPB являются задачами-ядрами большинства современных приложений. Знание ПВЛ таких задач и соответствующих им точек на поверхности Apex для различных вычислительных систем позволяет прогнозировать производитель-

ность приложений и подбирать наиболее подходящие конфигурации высокопроизводительных систем для конкретных классов задач. В работе [5] с помощью профилировщика на основе Dyninst API была получена ПВЛ только отдельных задач класса W: IS, CG, FT, MG и EP. Остальные задачи рассмотреть тогда не удалось по причине избыточного размера генерируемого разработанным профилировщиком журнала.

Планируется проведение экспериментальных запусков tslmap для построения поверхности быстродействия гибридной вычислительной системы с использованием всех задач пакета NPB. На основе Valgrind разрабатывается набор утилит для автоматизированного поиска интересующих регионов профиля. Предполагается расширить понятия показателей ПВЛ на многопоточные приложения с целью адаптации и исследования особенностей оценки ПВЛ для случая множества процессов.

Поскольку исследования ПВЛ приложений уже имеют значительную теоретическую базу, дальнейшие исследования должны учитывать существующий опыт экспериментального применения методологии Apex.

СПИСОК ЛИТЕРАТУРЫ

1. **Weinberg J., McCracken M.O., Strohmaier E., Snavely A.** Quantifying Locality In The Memory Access Patterns of HPC Applications // Proc. of the 2005 ACM/IEEE Conf. on Supercomputing. Washington, DC, USA: IEEE Computer Society, 2005. P. 50. [Электронный ресурс] // URL: http://dx.doi.org/10.1109/SC.2005.59

2. **Meuer H.W., Strohmaier E., Dongarra J., Simon H.D.** The TOP500: History, Trends, and Future Directions in High Performance Computing. Chapman & Hall/CRC, 2014.

3. **Ibrahim Khaled Z., Strohmaier Erich.** Characterizing the Relation Between Apex-Map Synthetic Probes and Reuse Distance Distributions // Proc. of the 2010 39th Internat. Conf. on Parallel Processing. Washington, DC, USA: IEEE Computer Society, 2010, Pp. 353–362. [Электронный ресурс] //URL: http://dx.doi.org/10.1109/ICPP.2010.43

4. **Семенов А.С.** Разработка и исследование архитектуры глобально адресуемой памяти мультитредово-потокового суперкомпьютера: Дисс. ... канд. техн. наук. Москва, 2010. 224 с.

5. **Максимов В.И.** Реализация методики измерения пространственно-временной локализации приложения и ее отображения на синтезируемую APEX-поверхность используемого оборудования: Дисс. ... магистра. СПб., 2013. 72 с.

6. **Shan Hongzhang, Strohmaier Erich.** A Multi-dimensional Micro-benchmark for Performance Study and Prediction // Proc. of the 2010 3rd Internat. Joint Conf. on Computational Science and Optimization. Washington, DC, USA: IEEE Computer Society, 2010. Pp. 156–160. [Электронный ресурс] // URL: http://dx.doi.org/10.1109/CSO.2010.134

7. **Xiaofeng Gao, Laurenzano M., Simon B., Snavely A.** Reducing overheads for acquiring dynamic memory traces // Workload Characterization Symp., 2005. Proc. of the IEEE Internat. Washington, DC, USA: IEEE Computer Society, 2005. Pp. 46–55. [Электронный ресурс] // URL: http://dx.doi.org/10.1109/IISWC.2005.1526000

8. **Fan Shiqing, Keller Rainer, Resch Michael.** Advanced Memory Checking Frameworks for MPI Parallel Applications in Open MPI // Tools for High Performance Computing 2011. Springer Berlin Heidelberg, 2012. Pp. 63–78. [Электронный ресурс] // URL: http://dx.doi.org/10.1007/978-3-642-31476-6_6

9. **Baumann Th., Gracia J.** Cudagrind: Memory-Usage Checking for CUDA // Tools for High Performance Computing 2013. Springer International Publishing, 2014. Pp. 67–78. [Электронный ресурс] // URL: http://dx.doi.org/10.1007/978-3-319-08144-1_6

10. **Jannesari A.** Detection of High-Level Synchronization Anomalies in Parallel Programs // Int. J. Parallel Program. 2015. Vol. 43. No. 4. Pp. 656–678. [Электронный ресурс] // URL: http://dx.doi.org/10.1007/s10766-014-0313-x

11. **Wu Xingfu, Taylor Valerie.** Performance Characteristics of Hybrid MPI/OpenMP Implementations of NAS Parallel Benchmarks SP and BT on Large-scale Multicore Supercomputers // SIGMETRICS Perform. Eval. Rev. 2011. Vol. 38. No. 4. Pp. 56–62. [Электронный ресурс] // URL: http://doi.acm.org/10.1145/1964218.1964228

REFERENCES

1. **Weinberg J., McCracken M.O., Strohmaier E., Snavely A.** Quantifying Locality In The Memory Access Patterns of HPC Applications. *Proceedings of the 2005 ACM/IEEE Conference on Supercomputing.* Washington, DC, USA: IEEE Computer Society, 2005, P. 50. Available: http://dx.doi.org/10.1109/SC.2005.59

2. **Meuer H.W., Strohmaier E., Dongarra J.,**

Simon H.D. *The TOP500: History, Trends, and Future Directions in High Performance Computing.* Chapman & Hall/CRC, 2014.

3. **Ibrahim Khaled Z., Strohmaier Erich.** Characterizing the Relation Between Apex-Map Synthetic Probes and Reuse Distance Distributions, *Proceedings of the 2010 39th International Conference on Parallel Processing*. Washington, DC, USA: IEEE Computer Society, 2010, Pp. 353–362. Available: http://dx.doi.org/10.1109/ICPP.2010.43

4. **Semenov A.S.** *Razrabotka i issledovaniye arkhitektury globalno adresuyemoy pamyati multitredovo-potokovogo superkompyutera* [*Development and research of globally addressable memory architecture multithreaded streaming supercomputer: Thesis for the degree of candidate of technical sciences*]. Moscow, 2010, 224 p. (rus)

5. **Maksimov V.I.** *Realizatsiya metodiki izmereniya prostranstvenno-vremennoy lokalizatsii prilozheniya i yeye otobrazheniya na sinteziruyemuyu APEX-poverkhnost ispolzuyemogo oborudovaniya* [*Implementation methodology for measuring spatial-temporal localization of the application and displayed on the surface of the synthesized APEX-used equipment: Thesis for a master's degree*]. St. Petersburg, 2013, 72 p. (rus)

6. **Shan Hongzhang, Strohmaier Erich.** A Multidimensional Micro-benchmark for Performance Study and Prediction, *Proceedings of the 2010 3rd International Joint Conference on Computational Science and Optimization*. Washington, DC, USA: IEEE Computer Society, 2010, Pp. 156–160. Available: http://dx.doi.org/10.1109/CSO.2010.134

7. **Xiaofeng Gao, Laurenzano M., Simon B., Snavely A.** Reducing overheads for acquiring dynamic memory traces, *Workload Characterization Symposium, 2005*, *Proceedings of the IEEE International*. Washington, DC, USA: IEEE Computer Society, 2005, Pp. 46–55. Available: http://dx.doi.org/10.1109/IISWC.2005.1526000

8. **Fan Shiqing, Keller Rainer, Resch Michael**. Advanced Memory Checking Frameworks for MPI Parallel Applications in Open MPI, *Tools for High Performance Computing 2011*. Springer Berlin Heidelberg, 2012, Pp. 63–78. Available: http://dx.doi.org/10.1007/978-3-642-31476-6_6

9. **Baumann Th., Gracia J.** Cudagrind: Memory-Usage Checking for CUDA, *Tools for High Performance Computing 2013*. Springer International Publishing, 2014, Pp. 67–78. Available: http://dx.doi.org/ 10.1007/978-3-319-08144-1_6

10. **Jannesari A.** Detection of High-Level Synchronization Anomalies in Parallel Programs, *Int. Journal Parallel Program*, 2015, Vol. 43, No. 4, Pp. 656–678. Available: http://dx.doi.org/10.1007/s10766-014-0313-x

11. **Wu Xingfu, Taylor Valerie.** Performance Characteristics of Hybrid MPI/OpenMP Implementations of NAS Parallel Benchmarks SP and BT on Large-scale Multicore Supercomputers, *SIGMETRICS Perform. Eval. Rev.,* 2011, Vol. 38, No. 4, Pp. 56–62. Available: http://doi.acm.org/10.1145/1964218.1964228

ЛЕВЧЕНКО Алексей Викторович – *ассистент кафедры информационных и управляющих систем Института компьютерных наук и технологий Санкт-Петербургского политехнического университета Петра Великого.*

195251, Россия, Санкт-Петербург, ул. Политехническая, д. 29.
E-mail: 2@expx.org

LEVCHENKO Aleksei V. *Peter the Great St. Petersburg Polytechnic University.*
195251, Politekhnicheskaya Str. 29, St. Petersburg, Russia.
E-mail: 2@expx.org

ФЁДОРОВ Станислав Алексеевич – *старший преподаватель кафедры информационных и управляющих систем Института компьютерных наук и технологий Санкт-Петербургского политехнического университета Петра Великого.*

195251, Россия, Санкт-Петербург, ул. Политехническая, д. 29.
E-mail: stanislav.fyodorov@ya.ru

FYODOROV Stanislav A. *Peter the Great St. Petersburg Polytechnic University.*
195251, Politekhnicheskaya Str. 29, St. Petersburg, Russia.
E-mail: stanislav.fyodorov@ya.ru