Программный инструментарий для анализа обращений многопоточной программы в память

Черешнев Е.С.

eugenechereshnev@gmail.com

Новосибирский государственный университет

Калгин К.В.

kalgin@ssd.sscc.ru

ИВМиМГ СО РАН

Введение

В настоящее время одной из главных проблем при разработке высокопроизводительных программ является растущий разрыв между скоростью процессора и доступом к памяти (memory wall). Невысокая пропускная способность и существенные задержки памяти (по сравнению со временем исполнения инструкций) приводят к тому, что в разработке алгоритмов важнейшую роль играет организация обращений в память, позволяющая свести к минимуму задержки исполнения, вызванные чтением и записью данных. Для сложных алгоритмов это превращается в нетривиальную задачу, и становятся актуальными различные инструменты для профилирования и анализа работы с памятью.

В работе описывается программный инструментарий для анализа обращений многопоточной программы в память. Инструмент реализован с использованием фреймворка Intel Pin [1] для отслеживания обращений в память. Для идентификации переменных при обращении в память используется отладочная информация в формате DWARF [2]. Здесь и далее под переменной понимается как переменная в обычном смысле, так и массив. Данный инструментарий создавался с целью его дальнейшего использования при разработке оптимизирующей системы для программ с плохими оценками локализации путем эмуляции массово-мультитредовой архитектуры на основе программных переключений контекста, как это описано в [6].

Бинарное инструментирование

Технология динамического бинарного инструментирования (DBI) позволяет во время исполнения отслеживать и изменять поведение программы. Разработанный инструментарий использует фреймворк Intel Pin для отслеживания событий следующих типов:

1. Вызовы функций и возвраты из функций (адрес начала стекового кадра, идентификатор функции).
2. Выделения и освобождения динамической памяти (адрес и длина блока памяти).
3. Обращения программных инструкций к памяти при чтении и записи (адрес и длина обращения).

Также для каждого события фиксируется идентификатор потока, породившего событие, и момент времени (используется счетчик тактов TSC).

Intel Pin производит инструментирование на разных уровнях гранулярности программы (инструкции, базовые блоки, трассы, процедуры, бинарные образы). Для регистрации событий первого и второго типов необходимо осуществлять инструментирование на уровне процедур, а для событий третьего типа – на уровне отдельных инструкций.

Для обработки каждого из событий реализованы отдельные анализирующие процедуры. В анализирующей процедуре создается структура данных, описывающая событие (тип события, идентификатор потока, момент времени и остальные параметры, относящиеся к данному типу события) и ее сохранение для дальнейшего анализа. Сконструированная структура добавляется в список, который хранится в оперативной памяти. После того как в списке накопится определенное число событий, происходит сохранение событий на жесткий диск. Сохранение происходит в отдельном внутреннем потоке фреймворка (internal thread), чтобы свести к минимуму задержки, связанные с записью данных на диск.

Основной анализ проводится уже после завершения исполнения инструментируемой программы. Для этого загружаются и обрабатываются сохраненные на предыдущем этапе события. Обработка события заключается в обновлении контекста исполнения, который содержит следующую информацию:

1. Стек вызовов со списком стековых кадров для каждого потока.
2. Список всех выделенных на текущий момент областей динамической памяти.

При обработке события обращения в память производится поиск объекта, которому принадлежит адрес обращения. Таким объектом может являться область динамической памяти, статическая переменная или стековая переменная (параметр или локальная переменная). В последнем случае анализируются функции из стека вызовов: поиск производится начиная с последней вызванной функции. Для каждой функции известен адрес стекового кадра и список переменных этой функции вместе со смещением относительно стекового кадра. Таким образом, если при анализе текущей функции адрес обращения попал внутрь стековой переменной, то поиск заканчивается, иначе происходит переход к функции, вызвавшей текущую.

Анализ отладочной информации для идентификации переменных при обращении в память

Бинарное инструментирование позволяет получить список обращений в память в формате: поток, адрес, размер блока для обращения. При последующем анализе определяется тип памяти: стековая, статическая или динамическая. При обращении к стеку определяется функция, к стековому кадру которой было произведено обращение.

При отсутствии отладочной информации можно провести поверхностный анализ без классификации обращений в память по переменным. Полученное множество обращений можно разделить, например, учитывая функцию, из которой было сделано обращение, или путем определения блоков памяти, обращения внутри которых выполнялись наиболее плотно.

С другой стороны, существуют работы (например, [3]), в которых описывается реализация более детального анализа бинарного кода, в результате которого локализуются отдельные переменные, извлекаются их типы и размеры. Однако стоит отметить, что, во-первых, анализ подобного рода не позволяет получить имена переменных, что затрудняет дальнейший разбор полученных данных, а во-вторых, описываемый в данной работе инструмент предназначен в первую очередь для анализа возможностей по оптимизации программы во время ее разработки, что подразумевает наличие исходного кода, имея который, можно скомпилировать программу с необходимыми опциями, в частности с включенной отладочной информацией в формате DWARF (–g или –gdwarf для gcc).

Инструмент работает с исполняемым файлом с отладочной информацией. Реализован разбор отладочной информации в формате DWARF для получения данных обо всех использующихся в программе переменных. Библиотека libdwarf предоставляет API для извлечения из исполняемого файла данных, описывающих каждую переменную: размер и адрес (фиксированный, если это глобальная или статическая переменная, или относительный, если это локальная переменная или параметр функции).

Анализ совместного использования данных

Использование отладочной информации при обработке событий позволяет получить список обращений для каждой переменной. Это дает возможность проанализировать обращения для каждой переменной отдельно и более детально. Для каждой переменной определяется:

1. Тип памяти (стековая, динамическая, статическая).

2. Поток, создавший переменную (в случае стековой переменной или переменной в динамической памяти).
3. Список потоков, выполнявших чтение/запись переменной.
4. Зоны чтения/записи для каждого потока в каждой функции.

В последнем пункте подразумевается определение блоков памяти, принадлежащих переменной, в которые производились обращения (чтение или запись) определенного потока в определенной функции, с последующей визуализацией.

Оценки для пространственной и временной локальности

Для определения узких мест в программе полезно иметь числовые характеристики, которые отражали бы, насколько эффективно происходят обращения в память с учетом пространственной и временной локальности этих обращений. Организация памяти в современных вычислительных системах активно использует концепцию локальности обращений для преодоления существующих проблем с производительностью. Иерархия памяти, предвыборка данных, алгоритмы кэширования – примеры механизмов, эффективность которых напрямую зависит от локальности обращений программы в память.

Для вычисления локальностей используется подход, предложенный в работах [4][5]. Данный подход использует для оценки пространственной локальности средневзвешенное значение величины, обратной к длине шага (stride), который вычисляется для каждого обращения в память, как расстояние до ближайшего обращения из множества последних W обращений. Для оценки временной локальности используется более сложный подход на основе вычисления расстояния переиспользования (reuse distance), которое определяется для каждой операции обращения как число обращений в память по уникальным адресам с момента последнего обращения по адресу обращения этой операции.

Инструмент вычисляет значения локальностей для различных контекстов, в пределах которых осуществляется обращение к памяти: поток, переменная, функция, строка кода и различные комбинации этих контекстов.

В качестве примера применения оценок локальностей можно рассмотреть стандартный алгоритм умножения матриц. В зависимости от порядка обхода исходных матриц время работы алгоритма значительно изменяется. Ниже приведена таблица с оценками локальностей и измеренным временем исполнения для 6 возможных вариантов обхода массивов. Из таблицы можно видеть, что для оптимальных по времени вариантов (ikj и kij) достигаются лучшие оценки локальностей, а для 2 медленных вариантов (jki и kji) оценки локальностей принимают худшие значения.

Порядок индексов	Время исполнения в тактах	Оценка пространственной локальности	Оценка временной локальности
ijk	8630961	0.250	0.388
ikj	5188671	0.571	0.586
jik	8605935	0.250	0.388
jki	14697141	0.000	0.388
kij	5149317	0.569	0.568
kji	14877348	0.002	0.388

Время исполнения и оценки локальностей для программы перемножения матриц 128x128 (Intel Core i7-3820).

Применение инструментария

Ниже перечислены этапы работы инструментария:

1. Регистрация анализирующих процедур, которые вызываются при событиях, описанных в разделе «Бинарное инструментирование».
2. Исполнение инструментируемой программы. Сбор трассы обращений к памяти.
3. Извлечение отладочной информации из исполняемого файла и сохранение информации о всех переменных программы вместе с адресом и размером.
4. Анализ собранных обращений:
    - Сбор информации отдельно для каждой переменной (тип памяти, создавший поток, зоны чтения/записи для каждого потока и каждой функции).
    - Вычисление оценок для пространственной и временной локальности относительно контекстов, описанных в предыдущем разделе.
5. Генерация отчетов:
    - Генерация таблиц на основе полученной информации для переменных. При наличии файлов с исходным кодом существует возможность генерации HTML-отчета с подсвечиванием строк исходного кода различными цветами в зависимости от вычисленной оценки локальности.
    - Генерация графиков с картой обращений для переменной с двумя осями: такты и смещение от начала переменной в байтах.

Инструмент успешно проверен на тестах из NAS Parallel Benchmarks. Для основных переменных сгенерированы карты обращений, построены отчеты с оценками для локальностей.

Ниже приводятся части отчетов для тестов IS (Integer Sort) и CG (Conjugate Gradient) из бенчмарка NAS Parallel Benchmarks.

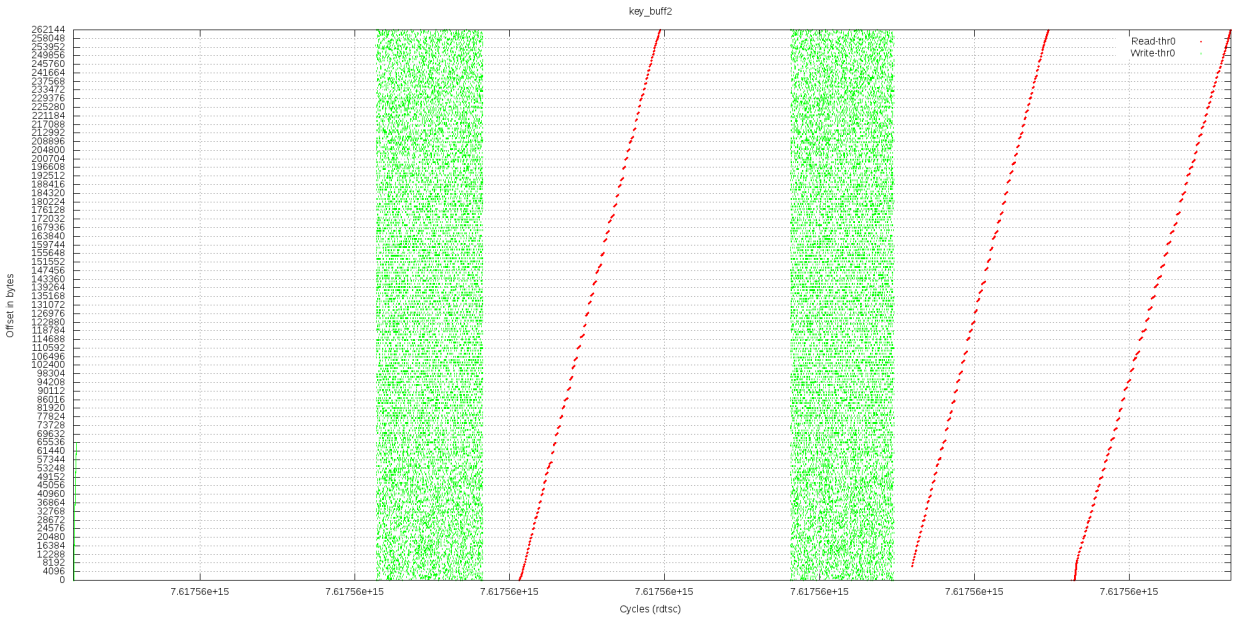
Тест NPB IS (Intger Sort), класс S

			Threads			
File	**Variable name**	**Size in bytes**	**0**	**1**	**2**	**3**
is.c	key_buff_ptr_global	8	R/W	R	R	R
is.c	passed_verification	4	R/W	-	-	-
is.c	key_array	262144	R/W	R/W	R/W	R/W
is.c	key_buff1	8192	R/W	R/W	R/W	R/W
is.c	key_buff2	262144	R/W	R/W	R/W	R/W
is.c	partial_verify_vals	20	R/W	-	-	-
is.c	bucket_size	8	R/W	R	R	R
is.c	test_index_array	20	R/W	-	-	-
is.c	test_rank_array	20	R/W	-	-	-
is.c	S_test_index_array	20	R	-	-	-
is.c	S_test_rank_array	20	R	-	-	-
is.c	bucket_size	32	R/W	R	R	R
is.c	bucket_size[0]	2048	R/W	R	R	R
is.c	bucket_size[1]	2048	R	R/W	R	R
is.c	bucket_size[2]	2048	R	R	R/W	R
is.c	bucket_size[3]	2048	R	R	R	R/W

Карта обращений к переменным по потокам для теста NPB IS, CLASS S

Variable	**Number of accesses**	**Spatial locality**	**Temporal locality**
All variables	1768565	0.213	0.463
key_array	524300	0.375	0.342
key_buff1	408586	0.052	0.480
key_buff2	458752	0.288	0.314

Значения пространственных и временных локальностей по переменным для теста NPB IS, класс S



Карта обращений к переменной key_buff2 0-го потока в тесте NPB IS, класс S

Тест NPB CG (Conjugate Gradient), класс S

					Threads			
File	**Function**	**Common block**	**Variable**	**Size in bytes**	**0**	**1**	**2**	**3**
cg.f	conj_grad	partit_size	naa	4	R/W	R	R	R
cg.f	conj_grad	partit_size	nzz	4	R/W	R	R	R
cg.f	conj_grad	partit_size	firstrow	4	R/W	R	R	R
cg.f	conj_grad	partit_size	lastrow	4	R/W	R	R	R
cg.f	conj_grad	partit_size	firstcol	4	R/W	R	R	R
cg.f	conj_grad	partit_size	lastcol	4	R/W	R	R	R
cg.f	conj_grad	timers	timeron	4	R/W	-	-	-
cg.f	conj_grad		d	8	R/W	R/W	R/W	R/W
cg.f	conj_grad		rho0	8	R/W	R	R	R
cg.f	sparse		i	4	W	-	-	-
cg.f	sparse		nza	4	W	-	-	-
cg.f	sprnvc		vecloc	8	R/W	-	-	-
cg.f	makea		iouter	4	R/W	-	-	-
cg.f	makea		ivc	32	R/W	-	-	-
cg.f	makea		nn1	4	R/W	-	-	-
cg.f	makea		nzv	4	R/W	-	-	-
cg.f	makea		vc	64	R/W	-	-	-

cg.f	cg	main_flt_mem	v	716800	R/W	R/W	R/W	R/W
cg.f	cg	main_flt_mem	aelt	89600	R/W	R/W	R/W	R/W
cg.f	cg	main_flt_mem	a	716800	R/W	R/W	R/W	R/W
cg.f	cg	main_flt_mem	x	11216	R/W	R/W	R/W	R/W
cg.f	cg	main_flt_mem	z	11216	R/W	R/W	R/W	R/W
cg.f	cg	main_flt_mem	p	11216	R/W	R/W	R/W	R/W
cg.f	cg	main_flt_mem	q	11216	R/W	R/W	R/W	R/W
cg.f	cg	main_flt_mem	r	11216	R/W	R/W	R/W	R/W
cg.f	cg	main_int_mem	colidx	358400	R/W	R/W	R/W	R/W
cg.f	cg	main_int_mem	rowstr	5604	R/W	R/W	R/W	R/W
cg.f	cg	main_int_mem	iv	364000	R/W	R/W	R/W	R/W
cg.f	cg	main_int_mem	arow	5600	R/W	R/W	R/W	R/W
cg.f	cg	main_int_mem	acol	44800	R/W	R/W	R/W	R/W

Карта обращений к переменным по потокам для теста NPB CG, класс S

Variable	**Number of accesses**	**Spatial locality**	**Temporal locality**
All variables	23466218	0.344	0.380
v	2600569	0.104	0.612
aelt	111898	0.638	0.528
a	4141844	0.990	0.000
x	14004	0.996	0.355
z	306098	0.341	0.372
p	4191602	0.202	0.459
q	214202	0.994	0.375
r	295404	0.508	0.319
colidx	4298140	0.486	0.232
rowstr	210675	0.231	0.493
iv	6787593	0.121	0.519
arow	23794	0.252	0.572
acol	190256	0.331	0.551

Значения пространственных и временных локальностей по переменным для теста NPB CG, класс S

В работе [4] описан инструмент со схожей функциональностью. Отличие данной работы в том, что анализ выполняется автоматически на основе извлеченной информации об используемых переменных (DWARF), а также детализированной системой отчетов с подробной информацией на основе проведенного анализа. Помимо этого была проработана система отчетов, позволяющая генерировать отчеты с подробной информацией на основе проведенного анализа.

Заключение

В работе описан инструментарий для анализа обращений многопоточной программы в память. Инструментарий использует фреймворк Intel Pin для отслеживания обращений в память и библиотеку libdwarf для извлечения из исполняемого файла информации обо всех переменных программы.

Использование отладочной информации позволяет более детально проанализировать трассу обращений в память. Инструмент генерирует отчеты для обращений к переменной в зависимости от потока, функции, строки кода. Также производится визуализация обращений к каждой переменной в виде временных графиков с отображением операций чтения/записи.

Инструментарий использован для анализа тестов из NAS Parallel Benchmarks.

В дальнейшем на базе описанного инструментария планируется реализовать оптимизирующую систему для программ с плохими оценками локализации путем эмуляции массово-мультитредовой архитектуры на основе программных переключений контекста, как это описано в [6].

Работа поддержана Грантом Президента РФ для молодых учёных МК-3644.2014.9.

Список источников

1. Pin – A Dynamic Binary Instrumentation Tool. URL: https://software.intel.com/en-us/articles/pin-a-dynamic-binary-instrumentation-tool
2. Dwarf Home. URL: http://www.dwarfstd.org
3. Thomas Reps, Gogul Balakrishnan. Improved Memory-Access Analysis for x86 Executables // Proceedings of the Joint European Conferences on Theory and Practice of Software 17th international conference on Compiler construction, pp. 16-35, 2008.
4. В.И. Максимов. Реализация методики измерения пространственно-временной локализации приложения и ее отображения на синтезируемую APEX-поверхность используемого оборудования.
5. Weinberg J, McCracken MO, Strohmaier E, Snavely A. Quantifying Locality in the Memory Access Patterns of HPC Applications. Proc. of the 2005 ACM/IEEE. Conf. on Supercomputing, November 2005.
6. В.С. Горбунов, Л.К. Эйсымонт. Комплексная методика тестирования производительности суперкомпьютеров, профессиональный подход. Журнал «Вычислительные методы и программирование», 2013.