ISSN 1992-6502 (Print) Вестник УГАТУ ISSN 2225-2789 (Online)
2014. Т. 18, № 1 (62). С. 224–229 http://journal.ugatu.ac.ru

УДК 519.68

МЕТОД ПОКРЫТИЙ ДЛЯ ОЦЕНКИ ЛОКАЛЬНОСТИ ИСПОЛЬЗОВАНИЯ ДАННЫХ В ПРОГРАММАХ

П. А. ШВЕЦ[1], ВАД. В. ВОЕВОДИН[2]

[1] shvets.pavel.srcc@gmail.com, [2] vadim@parallel.ru

Научно-исследовательский вычислительный центр
Московского государственного университета имени М. В. Ломоносова (НИВЦ МГУ)

Поступила в редакцию 18.11.2013

Аннотация. В рамках предложенного ранее подхода к исследованию эффективности работы с памятью на основе анализа профиля обращений в память разрабатывается новый метод для оценки локальности обращений в память. Данный метод рассматривает покрытие профиля обращений сеткой из прямоугольников и позволяет получить единую оценку для пространственной и временно́й локальности. Приведены результаты экспериментов для набора несложных широко распространенных программ и тестов, на основе которых проведен анализ полученных оценок локальности.

Ключевые слова: эффективность программ; пространственная и временна́я локальность; профиль обращений в память; метод покрытий.

ВВЕДЕНИЕ

Одним из главных факторов, влияющих на время выполнения программы, является эффективность работы с памятью. Причина этого — в так называемой проблеме «стены памяти» [1], которая до сих пор не решена. Иерархия памяти устроена чрезвычайно сложно, и время получения требуемых данных зависит не только от свойств кэш-памяти различных уровней и оперативной памяти, но и от низкоуровневых механизмов вроде аппаратного префетчера или TLB буфера. Все это приводит к тому, что в большинстве случаев память используется далеко не самым оптимальным образом, и, соответственно, эффективность работы программ очень низка [2].

Одним из главных свойств взаимодействия программ с памятью является локальность использования данных. Различают два типа локальности — пространственную и временную [3]. Пространственная локальность отражает среднее расстояние по памяти между несколькими последовательными обращениями в память; временная локальность показывает среднее число обращений по одному адресу в память за время исполнения программы.

Цель нашей работы заключается в исследовании эффективности взаимодействия программ с памятью с помощью свойств локальности. Для этого в работе предлагается подход на основе анализа потока обращений в память [4] к разработке простой характеристики для оценки свойств как пространственной, так и временной локальности. Ключевым моментом является именно простота получения и анализа характеристики, поскольку такой подход позволит в будущем пользователю достаточно легко и быстро получать оценку эффективности работы с памятью. Использование систем эмуляции работы памяти (вроде Threadspotter[1]) или сложных инструментов анализа (например, Valgrind[2]), во многих случаях позволит получать более точную и полную информацию, однако «минус» таких систем заключается в сложном использовании, а также в том, что зачастую для интерпретации полученных данных требуются определенные знания в этой области. Более того, привязка к свойствам локальности позволяет обеспечивать легкую переносимость разрабатываемых в рамках данной работы средств.

ИССЛЕДОВАНИЕ ПОТОКА ОБРАЩЕНИЙ В ПАМЯТЬ

Для исследования свойств локальности в рамках данной работы предлагается подход на основе анализа потока (или профиля) обращений в память.

Работа выполнена при поддержке гранта РФФИ 13-07-00787 и стипендии Президента РФ молодым ученым и аспирантам СП-6815.2013.5.

[1] http://www.roguewave.com/portals/0/products/threadspotter/docs/2011.1/manual/index.html.

[2] http://valgrind.org/docs/manual/manual.html.

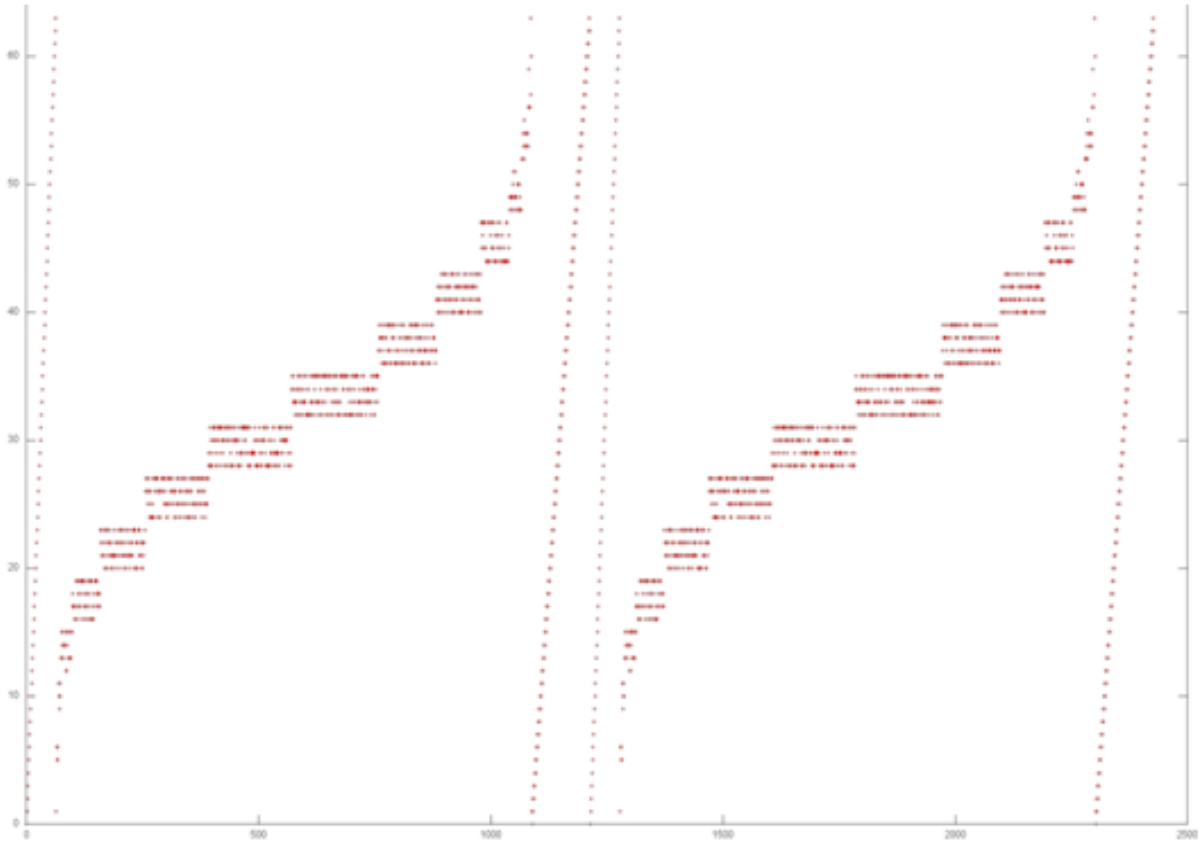

Рис. 1. Пример профиля обращений в память

Под потоком обращений будем понимать последовательность используемых в программе переменных, расположенных в том порядке, в котором происходят обращения к этим переменным в программе. Обозначим поток обращений

$$A = \{A_1, A_2, \dots, A_N\},$$

где A – общее число всех обращений к переменным. A_i обозначает адрес виртуальной памяти, где расположена запрашиваемая переменная; номер i элемента A_i указывает, каким по счету является данное обращение от начала выполнения программы. На данный момент мы рассматриваем только обращения к массивам, как самому распространенному типу структуры данных; также рассмотрение ограничивается последовательными программами на языке C/C++. Однако оба этих ограничения мы планируем в дальнейшем устранить.

Пример профиля обращений приведен на рис. 1. Ось абсцисс — порядковый номер обращения в память, т. е. чем больше номер, тем позже произошло данное обращение. По оси ординат отложен виртуальный адрес памяти, к которому произошло обращение. Каждая точка – это отдельное обращение в память. Выделенная на рисунке область *1* (и остальные аналогичные ей области) обладает хорошей пространственной и временной локальностью. Хорошей пространственной — потому что используются только соседние по памяти данные; хорошей временной — поскольку повторные обращения к этим данным происходят часто. Область *2* также обладает хорошей пространственной локальностью, поскольку соседние обращения происходят к соседним элементам, однако плохой временной локальностью, поскольку повторных обращений к данным просто нет.

Для получения профиля обращений выполняется инструментация исходного текста программы. Для этого необходимо заменить тип массива на реализованный C++ класс, который будет вести себя также, как и исходный массив, но при любом обращении к элементу массива (вызове оператора […]) также производит запись в отдельный лог-файл. Поскольку мы собираем поток виртуальных обращений, запись по всем интересующим нас массивам ведется в единый лог. Также есть возможность включать/выключать запись, чтобы не логгировать служебные или не интересующие нас фрагменты программы (к таким можно отнести, например, генерацию начальных массивов и сохранение/вывод результата).

Исследуемый нами поток обращений не всегда совпадает с действительным потоком обращений, получаемым после всех возможных преобразований (выполняемых, например, компилятором). По данному преобразованному потоку в большинстве случаев можно получить более точную информацию, но в этом случае значительно сложнее устанавливать связь между исходным текстом программы и самим профилем, и, соответственно, делать выводы относительно возможных оптимизаций программы.

ПОКРЫТИЕ КАК ОСНОВА ДЛЯ ОЦЕНКИ ЛОКАЛЬНОСТИ

Наша цель заключается в поиске одной или нескольких простых характеристик, которые помогут достаточно точно оценивать свойство локальности полученного потока обращений. На данный момент в результате проведенного исследования был предложен следующий подход.

Рассмотрим первые N обращений в потоке («окно»), и затем разобьем ось ординат на отрезки длины K – $[X \ldots X+K]$, $[X+K \ldots X+ +2*K]$, …, $[Y-K \ \ldots Y]$, где $[X \ \ldots \ Y]$ — диапазон выделенной под рассматриваемые массивы виртуальной памяти. Таким образом, область построения делится на прямоугольники размером $K \times N$. Физический смысл такого прямоугольника (при правильно подобранных значениях K и N) заключается в том, что любая последовательность точек, попавших в один прямоугольник, всегда обладает хорошей как пространственной, так и временной локальностью.

Посчитаем число прямоугольников, в которых оказалась хотя бы одна точка профиля. Соответственно, чем меньше таких прямоугольников, тем больше точек в среднем в одном прямоугольнике, тем в общем случае лучше локальность. На рис. 2 показаны разные с точки зрения локальности фрагменты профилей и их покрытий. «Окно» *1* покрывает фрагмент профиля с последовательными обращениями — здесь наблюдается хорошая локальность, лишь один прямоугольник является непустым. В «окне» *2* представлен фрагмент профиля также с последовательными обращениями, но идущие с некоторым шагом. В этом случае локальность хуже, чем у первого варианта, занято 2 прямоугольника. «Окно» *3* покрывает фрагмент профиля со случайной последовательностью обращений — здесь наблюдается плохая локальность, занято 3 прямоугольника.

Затем сдвинем «окно» на одну точку, т. е. рассмотрим обращения с 1 по N+1, и снова посчитаем число непустых прямоугольников. Далее выполним аналогичные действия по всему потоку обращений. Назовем покрытием полученное множество непустых прямоугольников для каждого «окна», в которых содержится хотя бы по одной точке профиля.

Теперь посчитаем число непустых прямоугольников для каждого «окна», в результате чего получим последовательность чисел, каждое из которых, в общих чертах, характеризует локальность в некоторой точке работы программы. Обозначим через Cvg среднее значение по всей последовательности. По сути, это значение характеризует, какую часть области построения покрывают непустые прямоугольники.

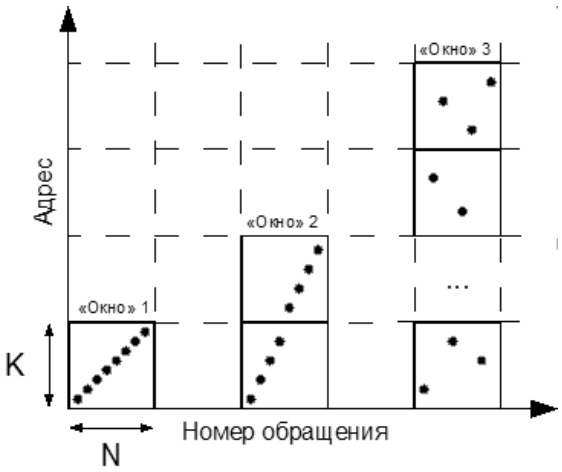

Рис. 2. Пример покрытия различных профилей

Достаточно серьезный вопрос заключается в подборе оптимальных значений параметров K и N. Желательно подобрать максимальные значения для них, при которых в рамках прямоугольника все еще сохраняется свойство хорошей локальности. На данный момент используется значение K = 64. Такое значение выбрано исходя из тех соображений, что при запросе данных из памяти в кэш-память 1-го уровня всегда подтягивается не только это значение, но и вся кэш-линия. А практически во всех современных процессорах размер кэш-линии равен 64 байтам. Соответственно, если взять K = 64, то мы можем быть уверены, что если произошел запрос данных по некоторому виртуальному адресу, то подтянутся также все остальные данные, соответствующие тому же прямоугольнику.

Поскольку мы считаем, что в рамках прямоугольника также и временная локальность должна быть хорошей, это означает, что если было запрошено значение T по некоторому ад-

ресу *A*, что через *N* обращений значение *T* с высокой вероятностью должно оставаться в кэш-памяти. С учетом этого требования в настоящий момент эвристическим методом выбрано значение $N = 128$, однако вопрос выбора значения *N* на данный момент еще до конца не решен.

ЭКСПЕРИМЕНТЫ

Все эксперименты проводились на последовательных реализациях широко распространенных программ:

- RandomAccess и STREAM из набора тестов HPCC [7];
- последовательная версия теста Linpack[3];
- БПФ над вещественными числами;
- классическое (не блочное) перемножение матриц;
- триада;

Тест STREAM представлен в 4 вариантах[4]. Перемножение матриц (MatMult) представлено в 6 вариантах, в зависимости от порядка циклов. Триада вычисляет выражение $A = B*X + C$, где *A* и *B* всегда являются массивами, а *X* и *C* могут быть как массивами, так и скалярами. В экспериментах было использовано 12 вариантов триады (3 группы по 4 варианта):

Первая группа:

- $A[i] = B[i] * X + C;$
- $A[i] = B[i] * X + C[i];$
- $A[i] = B[i] * X[i] + C;$
- $A[i] = B[i] * X[i] + C[i].$

Вторая группа:

- $A\left[ind1[i]\right] = B\left[ind1[i]\right] * X + C;$
- $A[ind1[i]] = B[ind1[i]] * X + +C[ind1[i]];$
- $A\left[ind1[i]\right] = B\left[ind1[i]\right] * X[ind1[i]] + C;$
- $A[ind1[i]] = B[ind1[i]] * X[ind1[i]] + C\left[ind1[i]\right].$

Третья группа:

- $A[ind2[i]] = B[ind2[i]] * X + C;$
- $A[ind2[i]] = B[ind2[i]] * X + +C[ind2[i]];$
- $A[ind2[i]] = B[ind2[i]] * X[ind2[i]] + C;$
- $A[ind2[i]] = B[ind2[i]] * X[ind2[i]] + C[ind2[i]].$

Во второй группе массив индексов $ind1[i] = i$, а в третьей группе $ind2[i]$ организован таким образом, чтобы кэш-память использовалась по возможности наихудшим способом.

Для каждой программы было посчитана характеристика Cvg. Значение Cvg, как и локальность, зависит от объема входных данных задачи, поэтому для всех программ использовались входные данные объемом примерно 1 МБ. Небольшой размер обуславливается значительным увеличением размера потока обращений при использовании большего объема.

Результаты проведенных нами экспериментов приведены на табл. 1. Значения отсортированы по увеличению характеристики Cvg; чем меньше Cvg, тем лучше локальность. Абсолютные значения Cvg на данный момент не важны, для нас больший интерес представляет ранжирование программ.

Таблица 1

Значения характеристики Cvg для тестовых программ

Программа	Значение Cvg
MatMult (ikj)	11
MatMult (kij)	11,1
Linpack	12,9
Triada 1	17,8
STREAM_1	17,8
STREAM_2	17,8
Triada 2	18,6
Triada 3	18,6
Triada 5	18,6
STREAM_3	18,6
STREAM_4	18,6
Triada 4	19,5
Triada 6	19,5
Triada 7	19,5
Triada 8	20,4
FFT	21
RandomAccess	36,2
MatMult (jik)	38
MatMult (ijk)	38
MatMult (kji)	65,3
MatMult (jki)	65,4
Triada 9	91,5
Triada 10	100,9
Triada 11	100,9
Triada 12	106,9

[3] http://people.sc.fsu.edu/~jburkardt/cpp_src/linpack_bench/linpack_bench_s.cpp.

[4] http://www.cs.virginia.edu/stream/ref.html#counting.

В данной таблице большинство полученных результатов соответствуют нашим представлениям об эффективности взаимодействия с памятью. Например, тест Linpack известен своей хорошей локальностью, что и подтверждается нашими результатами. Далее, 8 вариантов триады и 4 варианта STREAM обладают примерно одинаковой локальностью, и это логично, поскольку во всех этих программах происходит последовательный перебор нескольких массивов, и основная разница только в числе массивов. При этом с точки зрения работы с памятью вариант STREAM_1 и STREAM_3, а также STREAM_2 и STREAM_4 ничем не отличаются, как и соответствующие значения Cvg. Также можно увидеть, что тест RandomAccess обладает достаточно плохой локальностью, однако лучше нескольких вариантов перемножения матриц. Это объясняется, скорее всего, небольшим размером входных данных; при увеличении размера локальность этого теста будет ухудшаться быстрее, чем локальность вариантов MatMult. Самая плохая локальность наблюдается у 4 последних вариантов триады, и причина этого в том, что в них происходит случайный перебор элементов сразу нескольких массивов, что приводит к чрезвычайно неэффективной работе с памятью.

Далее можно заметить, что 6 вариантов перемножения матриц четко разделились по значениям Cvg на 3 группы, в зависимости от того, по какому измерению выполняется внутренний цикл. Такая закономерность подтверждается полученными реальными значениями эффективности, которые приведены в табл. 2. Эффективность измеряется как отношение реальной производительности (число операций с плавающей запятой в секунду) к пиковой.

Таблица 2

Сравнение значений Cvg и реальной эффективности для умножения матриц

Вариант	Значение Cvg	Эффективность, % (Xeon X5650)
ikj	11	13,9
kij	11,1	13,9
jik	38	8,8
ijk	38	9,5
kji	65,3	4,8
jki	65,4	4,8

Важной особенностью характеристики Cvg является ее независимость от свойств выбранного процессора, поскольку в ней учитывается только самое общее представление о строении памяти. Таким образом мы пытаемся выделить машинно-независимое свойство программы, не привязанное к конкретному аппаратному обеспечению.

ЗАКЛЮЧЕНИЕ

В данной статье приведено описание разрабатываемой нами простой характеристики для оценки эффективности работы с памятью на основе анализа локальности обращений. На данный момент предлагаемая характеристика Cvg позволяет осуществить относительное сравнение свойств локальности обращений в память различных программ. В дальнейшем мы планируем с помощью данной характеристики или ее модификаций научиться составлять подробное описание локальности для каждой программы, а также предлагать рекомендации по улучшению свойств локальности, и, как следствие, эффективности работы с памятью.

Сравнение с эффективностью выполнения всей программы является не очень точным, поскольку в такой характеристике учитывается не интересующее нас время выполнения арифметических операций. Поэтому в дальнейшем также предполагается определить более точную объективную меру эффективности именно работы с памятью для сравнения с разрабатываемой нами характеристикой. Это можно сделать, например, с помощью эмулятора иерархии памяти, которому передается полученный нами поток обращений, или доступа к аппаратным счетчикам процессора.

Также планируется перейти к рассмотрению реальных приложений из различных предметных областей. При накоплении достаточных знаний по реальным приложениям в дальнейшем планируется выделить классы задач по свойствам локальности, а также определить и описать характерные для данных классов примеры потоков обращений.

СПИСОК ЛИТЕРАТУРЫ

1. **Wulf W. A., McKee S. A.** Hitting the memory wall: implications of the obvious // Computer Architecture News. Mar. 1995. Vol. 23, no. 1. P. 20–24. [W. A. Wulf and S. A. McKee, “Hitting the memory wall: implications of the obvious,” *Computer Architecture News*, vol. 23, no. 1, pp. 20-24, Mar. 1995.]

2. **Воеводин В. В.** Суперкомпьютеры и КПД паровоза // Суперкомпьютерные технологии и высокопроизводительные вычисления в образовании, науке и промышленности: доклад на Всерос. молодеж. шк. (ННГУ, 2009). [V. V. Voevodin, “Supercomputers and efficiency of a locomotive,” (in Russian), in *Supercomputing technologies and HPC in education, science and industry: presentation at Russian youth school* (NNGU, 2009).]

3. **Weinberg J., *et al*.** Quantifying locality in the memory access patterns of HPC applications // Proc. 2005 ACM/IEEE Conf. on Supercomputing (SC '05). [J. Weinberg, *et al.*, "Quantifying locality in the memory access patterns of HPC applications," in *Proc. 2005 ACM/IEEE Conf. on Supercomputing (SC '05).*]

4. **Воеводин Вад. В.** Визуализация и анализ профиля обращений в память // Вестник Южно-Уральского государственного университета. Сер. Математическое моделирование и программирование. 2011. Т. 17, № 234. С. 76–84. [Vad. V. Voevodin, "Visualization of memory access profile," (in Russian), *Vestnik South Ural State University. Mathematical modeling and programming*, vol. 17, no. 234, pp. 76-84, 2011.]

7. **Luszczek P., *et al*.** The HPC Challenge (HPCC) Benchmark Suite // SC06 Conference Tutorial, IEEE, Tampa, Florida, November 12, 2006. [P. Luszczek, *et al*., "The HPC Challenge (HPCC) Benchmark Suite," in *SC06 Conference Tutorial*, IEEE, Tampa, Florida, November 12, 2006.]

ОБ АВТОРАХ

ШВЕЦ Павел Артемович, программист лаб. парал. инф. технол. Дипл. математик, сист. программист (МГУ, 2012).

ВОЕВОДИН Вадим Владимирович, науч. сотр. лаб. парал. инф. технол. Дипл. математик, сист. программист (МГУ, 2003). Канд. физ.-мат. наук (НИВЦ МГУ, 2011). Иссл. в обл. эффективности программ, локальности данных.

METADATA

Title: The covering method for measuring locality of programs memory usage.

Authors: P. A. Shvets and Vad. V. Voevodin.

Affiliation: Research Computing Center of Moscow State University (RCC MSU), Russia.

Email: shvets.pavel.srcc@gmail.com, vadim@parallel.ru.

Language: Russian.

Source: Vestnik UGATU (scientific journal of Ufa State Aviation Technical University), vol. 18, no. 1 (62), pp. 224-229, 2014. ISSN 2225-2789 (Online), ISSN 1992-6502 (Print).

Abstract: Introducing a new method for measuring locality of programs memory usage, basing on previously proposed method for analyzing program efficiency through the study of memory access profile. This method bases on covering memory access profile with the grid of rectangles and allows you to learn a single measure for the spatial and temporal locality. There are experimental results for a set of common programs and benchmarks and analysis of locality of this programs memory usage, basing on the results.

Key words: programs efficiency; spatial and temporal locality; memory access profile; covering method.

About authors:

SHVETS, Pavel Artemovich, software engineer in Parallel information technologies laboratory. Specialist in mathematics and system programming (MSU, 2012).

VOEVODIN, Vadim Vladimirovich, research associate in Parallel information technologies laboratory. Specialist in mathematics and system programming (MSU, 2003).