1. *Шоботов В.М.* Цивільна оборона: навчальний посібник / В.М. Шоботов. – вид. 2-ге, перероб. – К.: Центр навчальної літератури, 2006. – 438 с.
2. *Дивизинюк М.М.* Модель управления чрезвычайной ситуацией / М.М. Дивизинюк, Г.М. Коротенко, Г.А. Черненькая и др. // Сб. науч. тр. СНУЯЭиП. – 2009. – Вып. 4(32). – С.204-208.
3. *Дивизинюк М.М.* Особенности моделирования чрезвычайных ситуаций, вызванных распространением антропогенных загрязнений / М.М. Дивизинюк, Е.В. Азаренко, А.Н. Фурсенко // Збірник наукових праць СНУЯЕтаП. – 2012. – Вип. 1 (41). – С.201-207.
4. *Попов О.О.* Прогнозування аварійного ризику / О.О. Попов // Техногенно-екологічна безпека та цивільний захист. – 2013. – № 6. – С.28-33.
5. *Сергеев В.С.* Защита населения и территорий в чрезвычайных ситуациях: учебное пособие для вузов / В.С. Сергеев. – М.: Академический Проект, 2004. – 429 с.
6. *Акимов В.А.* Природные и техногенные чрезвычайные ситуации: опасности, угрозы, риски / В.А. Акимов, В.Д. Новиков, Н.Н. Радаев. – М.: ЗАО ФИД, 2001. – 344 с.
7. *Кофф Г.Л.* Оценка последствий чрезвычайных ситуаций / Г.Л. Кофф, А.А. Гусев, Ю.Л. Воробьев. – М.: РЭФИА, 1997. – 364 с.
8. *Мастрюков Б.С.* Безопасность в чрезвычайных ситуациях: учебное пособие для вузов / Б.С. Мастрюков. – М.: Академия, 2003. – 331 с.
9. Реагування на виникнення надзвичайних ситуацій: монографія / С.О. Гур'єв, А.В. Терент'єва, С.М. Миронець [та ін.]; за заг. ред. С.О. Гур'єва. – Вінниця: ІДУЦЗ НУЦЗУ [та ін.], 2010. – 412 с.

Поступила 9.10.2017р.

УДК 004.896

А.А. Чемерис, С.В. Сушко, Киев

ИССЛЕДОВАНИЕ БЫСТРОДЕЙСТВИЯ И ЭНЕРГОПОТРЕБЛЕНИЯ ПРИ АВТОМАТИЧЕСКОЙ ОПТИМИЗАЦИИ МЕТОДАМИ РАЗБИЕНИЯ НА БЛОКИ И РАСПАРАЛЛЕЛИВАНИЯ ДЛЯ ВЫЧИСЛЕНИЙ НА ПЛАТФОРМЕ X64

Abstract. Different approaches of the software optimization were observed. Practical results of the measurements of the test program processing time, energy consumption and energy efficiency of the optimized test applications were obtained. Conclusions on the applicability of the different variations of the tiling method and code parallelization on the platforms based on x64 system commands are given.

Актуальность

Оптимизация программного обеспечения (ПО) является востребованной задачей для практического применения вычислительных систем различного назначения. В общем случае, оптимизированное ПО потребляет меньше

ресурсов для своей работы и/или выполняется быстрее [1]. Непосредственный оптимизированный ресурс в каждом случае выбирается индивидуально. Как правило, это либо время выполнения, либо размер кода программы. Рассматривая вычислительную систему как аппаратно-программный комплекс, следует также брать во внимание его энергопотребление в единицу времени и энергию, необходимую для вычисления произвольного алгоритма или всей программы целиком. В этом контексте оптимизация также предоставляет возможность снижения общей энергии вычисления и, в некоторых случаях, возможность снижения энергопотребления. Следует учитывать, что при более эффективном использовании ресурсов процессора, при распараллеливании задач, энергопотребление может возрасти, при этом общая энергия вычислений может снизиться.

Постановка задачи

Исходя из задачи оптимизации по времени вычисления, нетрудно заметить, что большую часть времени выполнения программ процессор проводит в достаточно небольших областях памяти программы, а именно в вычислительных циклах. Выполняясь многократно, такие участки кода программ при малых размерах могут занимать подавляющую часть времени работы программы. Следовательно, именно вычислительные циклы представляют собой наиболее эффективное место для оптимизации по времени выполнения.

Основными подходами по оптимизации циклов являются их распараллеливание и модификация тела циклов [2]. Распараллеливание циклов наиболее эффективно в многоядерных системах в тех случаях, когда вычисляемые данные не используются как входные на следующих итерациях цикла. В противном случае выделяют зависимости данных [3], которые могут значительно влиять как на распараллеливание, так и на прочие модификации циклов. Отмечая потенциально значительный выигрыш по времени выполнения программ от распараллеливания, следует также иметь в виду, что не весь код может быть распараллелен [4]. Согласно закону Амдала, ускорение, которое может быть теоретически достигнуто на вычислительной системе из нескольких процессоров p,по сравнению с однопроцессорным решением не будет превышать величины:

$$S_p = \frac{1}{a + \frac{1-a}{p}} \qquad (1)$$

где S_p – относительное ускорение вычислений, p – число процессоров в вычислительной системе, a – доля вычислений, которая может быть рассчитана только последовательно. Как следует из формулы, часть вычислений, которая не может быть распараллелена, будет иметь ключевое влияние для вычислительных систем с большим количеством ядер.

Для математического описания вычислительных циклов используется несколько подходов. Одним из наиболее проработанных и наглядных методов описания вычислительных циклов является полиэдральная модель. Данная модель позволяет представить любой вычислительный цикл как некоторую математическую абстракцию. Она основана на представлении цикла в пространстве итераций, которым называется множество всех целочисленных векторов $i = (I_1, I_2, \ldots, I_n)$, удовлетворяющих системе неравенств [5].

$$L_i \le x_i \le U_i, \text{ где } i = 1 \ldots n. \quad (2)$$

Неравенства **Ошибка! Источник ссылки не найден.** определяют границы циклов и пространство итераций, которое ограничивается выпуклым многогранником. В этом случае граф зависимостей рассматривают как множество векторов $P = \{x | x \in Q^n, Ax \le b\}$, где A и b – целочисленная матрица и целочисленный вектор, соответственно.

Далее модель может быть модифицирована любым из способов, который не изменяет результатов непосредственно выходных данных и затем модель может быть преобразована обратно в исходный код. После этапа преобразования кода в оптимизированный исходный код компилятор выполнит все дополнительные оптимизации на своем уровне оптимизаций.

Практическое использование полиэдральной модели с целью оптимизации циклов программ

Полиэдральная модель позволяет применять различные методы оптимизации к вычислительным циклам [6]. В общем случае, схема работы с исходным кодом имеет вид, как показано на рис. 1:

Полиэдральная модель позволяет работать с вычислительными циклами произвольной вложенности. Например, такой двойной вычислительный цикл:

```
for(i = 0; i < n; i ++)
    for(j = 0; j < i + 2; j++)
            A[i][j] = A[i - 1][j] + A[i][j - 1];
```

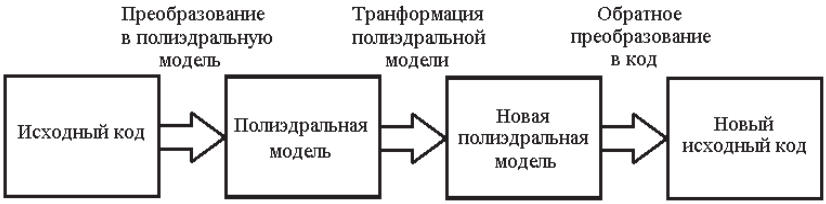
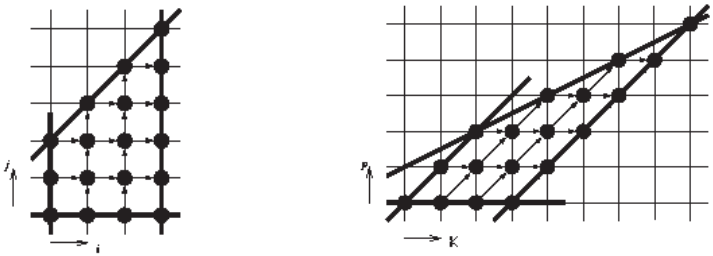


Рис. 1. Схема использования полиэдральной модели

Графически полиэдральная модель цикла, представленного выше, приведена на рис. 2а

После модификации графическая интерпретация полиэдральной модели цикла имеет вид, как показано на рис. 2б:



а) б)

Рис. 2. Пример модифицированной полиэдральной модели

Программный код данной трансформированной модели будет иметь вид:

```
for(k = 0; k < 2*n + 2; k ++)
    for(p = max(0, k - n); p < min(k, k/2 + 1); p++)
            A[k - p][p] = A[k - p - 1][p] + A[k - p][p - 1];
```

Рассматривая все множество подходов модификации вычислительных циклов, следует отметить такой метод как разбиение цикла на блоки (tiling) [7]. Пример разбиения на блоки представлен на рис. 3.

Описание эксперимента

Для проверки временной и энергетической эффективности работы программы автоматической оптимизации, основанной на полиэдральной модели, был выбран пакет Pluto 11.4 [8]. Тестовый стенд представляет собой настольный ПК с четырехядерным процессором Intel Core i5-4670K и 64-х разрядной операционной системой Ubuntu 14.04 LTS. Тестовые приложения, на которых производилась оценка времени выполнения и энергопотребления были взяты из набора тестов PolyBench/C 4.1. Замер времени проводился программно поочередно для каждого из тестовых приложений, замер мощности потребления производился ваттметром Robiton PM-2. Замер мощности потребления всего ПК производился без учета монитора [9].

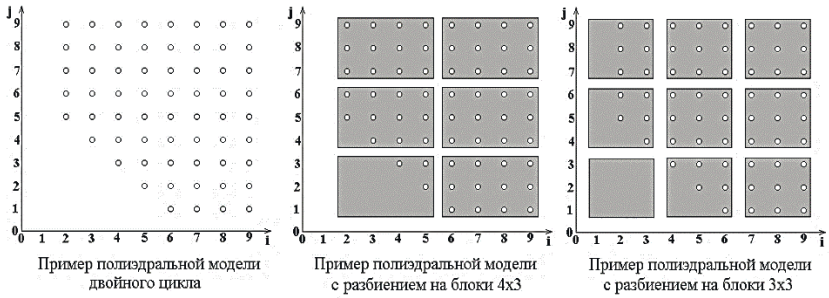


Рис. 3. Пример использования полиэдральной модели для метода разбиения на блоки

Замер показателей сначала производился для каждой оригинальной тестовой программы, а затем каждая из тестовых программ подавалась на вход пакета оптимизации Pluto. Выходной исходный код, полученный по выходу Pluto, также компилировался и на полученной оптимизированной версии производились такие же измерения.

Таким образом, в результате серии экспериментов были получены значения времени выполнения и средние значения энергопотребления выполнения тестовых программ.

Суммарная электрическая энергия, необходимая для вычисления алгоритма равна интегралу электрической мощности по времени вычисления этого алгоритма.

$$E = \int_0^T p(t)dt \tag{3}$$

Учитывая, что среднее значение мощности потребления может быть подставлено согласно формуле:

$$P_{mean} = \frac{1}{T}\int_0^T p(t)dt \tag{4}$$

Из вышеуказанных формул получим общую формулу для расчета электрической энергии:

$$E = P_{mean} \bullet T \tag{5}$$

Для получения качественной оценки повышения энергетической эффективности при использовании оптимизации, введем коэффициент энергетической эффективности вычислений, как отношение начальной электрической энергии, требуемой для расчета к электрической энергии, требуемой для расчета оптимизированной версии. Таким образом, данный коэффициент представляет собой число, которое является сокращением электрической энергии, требуемой для вычислений.

$$kE = \frac{E_{original}}{E_{optimized}} = \left(\frac{T_{original}}{T_{optimized}}\right) \bullet \left(\frac{P_{mean\_original}}{P_{mean\_optimized}}\right) \tag{6}$$

Принимая это во внимание, что значения из правой части формулы получены на первом и втором этапах измерений, получим коэффициенты энергетической эффективности, которые представлены в табл. 1 и 2.

Анализ полученных результатов

В целом, следует отметить ускорение времени вычисления в большинстве тестовых программ. В то же время, существуют отдельные тестовые приложения, которые практически не поддаются оптимизации и имеют приблизительно одинаковое время выполнения независимо от использованной опции оптимизации. Также следует отметить, что в единичных случаях использование оптимизаций в однопоточном режиме некоторых методов разбиения на блоки может увеличивать время выполнения тестовых программ.

Таблица 1

Результаты относительного изменения потребляемой электрической энергии тестовых приложений в зависимости от алгоритма оптимизации, разы

Тестовая программа	tile	tile parallel	tile2tile parallel	innerpar	innerpar parallel
correlation	4.86	8.07	5.01	4.80	10.71
covariance	4.81	7.54	5.00	4.99	14.48
gemm	2.35	6.28	18.93	1.45	5.89
gemver	5.12	12.43	21.14	16.04	31.95
gesummv	0.88	2.16	3.10	1.77	6.51
symm	1.04	1.02	1.06	1.01	1.03
syr2k	1.41	3.68	1.81	1.44	8.39
syrk	1.00	2.12	1.33	0.96	2.91
trmm	3.40	11.10	4.61	5.47	1.70
2mm	1.91	3.92	2.37	1.64	9.24
3mm	1.59	3.28	2.01	1.34	7.41
atax	0.42	1.04	1.56	1.00	1.89
bicg	0.82	2.08	3.28	2.26	3.93
doitgen	4.73	2.84	4.42	6.42	0.30
mvt	3.67	8.53	17.28	1.16	4.64
cholesky	1.24	2.20	1.48	1.01	2.33
durbin	1.00	1.00	1.01	0.99	1.00
gramschmidt	1.79	2.41	1.74	1.04	3.80
lu	1.46	2.74	2.25	1.60	1.90
ludcmp	1.01	1.00	1.00	0.99	1.00
trisolv	0.73	1.43	1.93	1.22	1.96
deriche	1.28	1.57	2.00	1.00	2.43
floyd-warshall	0.81	1.21	1.33	0.99	0.54
nussinov	1.04	2.30	1.52	1.02	2.86
fdtd-2d	0.93	1.18	0.46	0.92	4.22
heat-3d	2.02	1.64	-	2.62	11.79
jacobi-2d	0.97	1.35	-	0.95	5.05
seidel-2d	1.19	2.22	-	1.02	4.75

Рассматривая энергопотребление при выполнении тестовых программ, следует отметить несколько важных деталей.

Во-первых, использование различных методов разбиения цикла на блоки в однопоточном режиме может, как незначительно снижать, так и незначительно повышать энергопотребление вычислений.

Во-вторых, использование тех же методов разбиения на блоки, но уже совместно с распараллеливанием, может значительно увеличивать энергопотребление. Это объясняется использованием всех четырех ядер тестового стенда и возрастанием нагрузки на подсистему памяти.

В-третьих, энергопотребление тех тестовых приложений, которые не

показывают выигрыш по времени выполнения, остается приблизительно на том же уровне, что и для исходных версий программ. Из этого можно сделать вывод, что в таких случаях не происходит распараллеливания либо оно происходит крайне неэффективно.

Таблица 2

Результаты относительного изменения потребляемой электрической энергии тестовых приложений в зависимости от алгоритма оптимизации (продолжение), разы

Тестовая программа	innerpar tile parallel	tile multipar parallel	diamond-tile	diamond-tile parallel
correlation	15.73	7.13	5.10	15.03
covariance	16.06	7.52	5.09	15.50
gemm	14.76	5.79	2.25	14.57
gemver	20.90	10.08	5.12	18.61
gesummv	3.91	1.84	0.90	3.62
symm	1.05	1.05	1.06	1.05
syr2k	7.46	2.97	1.32	7.43
syrk	2.87	1.56	0.99	3.83
trmm	27.26	10.97	3.49	25.93
2mm	7.90	4.34	1.97	7.94
3mm	6.78	3.70	1.68	6.79
atax	1.50	0.72	0.36	1.52
bicg	3.27	1.61	0.76	2.57
doitgen	5.65	6.20	5.39	5.50
mvt	15.23	8.19	3.69	13.35
cholesky	4.19	1.82	1.28	4.07
durbin	1.00	1.00	1.00	1.00
gramschmidt	4.07	2.84	1.75	3.97
lu	6.35	2.90	1.53	5.18
ludcmp	1.00	0.97	0.96	0.99
trisolv	2.47	1.09	0.70	1.64
deriche	2.83	1.32	1.19	1.75
floyd-warshall	2.22	1.27	0.84	2.18
nussinov	3.49	2.17	1.07	4.15
fdtd-2d	3.02	1.46	-	-
heat-3d	4.37	2.51	-	-
jacobi-2d	2.99	1.61	-	-
seidel-2d	4.65	2.14	-	-

Рассматривая энергопотребление при выполнении тестовых программ, следует отметить несколько важных деталей.

Во-первых, использование различных методов разбиения цикла на блоки в однопоточном режиме может, как незначительно снижать, так и незначительно повышать энергопотребление вычислений.

Во-вторых, использование тех же методов разбиения на блоки, но уже совместно с распараллеливанием, может значительно увеличивать энергопотребление. Это объясняется использованием всех четырех ядер тестового стенда и возрастанием нагрузки на подсистему памяти.

В-третьих, энергопотребление тех тестовых приложений, которые не показывают выигрыш по времени выполнения, остается приблизительно на том же уровне, что и для исходных версий программ. Из этого можно сделать вывод, что в таких случаях не происходит распараллеливания либо оно происходит крайне неэффективно.

Переходя к анализу общей энергоэффективности, следует обратить внимание в первую очередь на отличия в результатах с опцией распараллеливания и без нее. Для однопоточных версий оптимизаций тестовых приложений характерно незначительное улучшение энергоэффективности, при этом в некоторых случаях энергоэффективность вычислений может снижаться. Для многопоточных версий энергоэффективность в подавляющем большинстве случаев превышает результаты однопоточных версий. Это свидетельствует не только об эффективности метода разбиения на блоки, но и об успешном применении этого метода совместно с распараллеливанием кода.

Рассматривая конкретные способы разбиения на блоки, следует отметить, что среди использованных опций, а именно стандартной опции разбиения цикла на блоки (tile), полноразмерный одновременный старт разбиения на блоки (diamond-tile), использования исключительно внутреннего параллелизма (innerpar) нельзя выделить четкого лидера по производительности и энергоэффективности. Есть тестовые программы, где лучшие результаты показала одна из опций разбиения на блоки и есть тестовые программы, где лучше была другая опция. Все это свидетельствует о зависимости эффективности оптимизаций непосредственно от самого алгоритма вычислений. Соответственно, нельзя отдать предпочтение какому-то одному из методов разбиения на блоки априори.

Следует также отметить различный разброс в результатах в зависимости от тестовой программы. Можно выделить две группы тестовых программ. В первую можно отнести те, результаты энергоэффективности которых улучшаются при некоторых опциях оптимизатора. В наилучшем случае было зафиксировано значение 27.26. Более характерны наилучшие числа оптимизации в диапазоне 2–7. В то же время, обнаружено три тестовых примера, в которых энергоэффективность не улучшилась или составляет единицы процентов. Такие результаты свидетельствуют о неспособности используемых алгоритмов оптимизации к эффективной трансформации вычислительных циклов в данных тестовых программах. Такие тестовые программы требуют совершенно иных методов оптимизации вычислительных циклов.

Выводы

Принимая во внимание результаты экспериментов можно сделать следующие выводы:

– использование полиэдральной модели с методами разбиения на блоки и распараллеливания приводит в большинстве случаев к улучшению быстродействия и улучшению энергоэффективности расчетов;

– современные пакеты программ, использующие полиэдральную модель в качестве основы, могут в автоматическом режиме оптимизировать исходный код программ; в то же время в некоторых тестовых примерах и при некоторых опциях компиляции преобразования кода не происходит;

– в большинстве случаев использование различных оптимизаций положительно влияет на время выполнения и на общую энергоэффективность вычислений; в то же время некоторые тестовые приложения не поддаются оптимизации и сохраняют время выполнения приблизительно на исходном уровне, что дает направление последующих исследований;

– нельзя выделить однозначно лидирующего подхода по показателям быстродействия и энергоэффективности среди реализованных методов разбиения цикла на блоки.

1. *Shinan W* Software power analysis and optimization for power-aware multicore systems: дис. канд. / Shinan Wang – Детройт, 2014. – 177 с.
2. *Jingling X* Loop tiling for parallelism / Xue Jingling. – (Kluwer international series in engineering and computer science; SECS 575).
3. *Pugh W.* An Exact Method for Analysis of Value-based Array Data Dependences / W. Pugh, D. Wonnacott. – Univ. of Maryland, 1993.
4. *Darte A* Combining retiming and scheduling techniques for loop parallelization and loop tiling. / A. Darte, G. Silber, F. Vivien. // Parallel Processing Letters. – 1997. – №7. – С.379-392.
5. *В.А. Евстигнеев* Анализ зависимостей: состояние проблемы // Системная информатика: Сб. науч. тр. / Ин-т систем информатики СО РАН. – Новосибирск: Наука,, № 7, р.112-173., 2000.
6. *C´edric Bastoul* Code generation in the polyhedral model is easier than you think. In IEEE International Conference on Parallel Architectures and Compilation Techniques, pages 7-16, September 2004.
7. *Uday Bondhugula* Effective Automatic Parallelization and Locality Optimization Using The Polyhedral model: дис. канд. / Uday Bondhugula. – The Ohio State University, 2010. – 193 с.
8. *Uday Bondhugula, M. Baskaran, S. Krishnamoorthy, J. Ramanujam, A. Rountev, and P. Sadayappan* Automatic Transformations for Communication-Minimized Parallelization and Locality Optimization in the Polyhedral Model. International Conference on Compiler Construction (ETAPS CC), April 2008, Budapest, Hungary.
9. Summer InfoCom 2016: Матеріали II Міжнародної науково-практичної конференції, м. Київ, 1-3 червня 2016 р. – К.: Вид-во «Інжиніринг», 2016. – 116 с. ISBN 978-966-2344-50-9 С.74-76.

Поступила 25.09.2017р.