УДК 004.896

Сушко С.В.,
orcid.org/0000-0001-5107-5115,
Чемерис А.А., д.т.н.,
orcid.org/0000-0002-8134-5152

ИССЛЕДОВАНИЕ ПАРАМЕТРОВ ДИСКРЕТНОГО МЕТОДА РОЯ ЧАСТИЦ ПРИ ПОИСКЕ ОПТИМАЛЬНЫХ РАЗМЕРОВ БЛОКОВ РАЗБИЕНИЯ ЦИКЛИЧЕСКИХ ОПЕРАТОРОВ ПРОГРАММ

Институт проблем моделирования в энергетике им. Г.Е. Пухова НАН Украины

sergii.sushko@gmail.com
a.a.chemeris@gmail.com

Введение

Оптимизация программного обеспечения подразумевает изменение программного обеспечения с целью улучшения одного или нескольких оптимизируемых параметров без изменения результата вычисления [1].

Оптимизация программного обеспечения традиционно является актуальной задачей научных и коммерческих разработок. Появляются новое аппаратное обеспечение и, соответственно, новые компиляторы и средства оптимизации. Отдельным направлением оптимизации являются универсальные способы оптимизации. Такие способы оптимизации не привязаны к аппаратной реализации и применение таких методов в той или иной степени может быть эффективно на широком классе используемых задач.

Оптимизация программного обеспечения направлена на улучшение некоторого наперед заданного параметра оптимизации. Параметрами оптимизации, как правило, являются время выполнения программы, размер памяти программ, размер памяти данных, энергопотребление вычислительного устройства, необходимая энергия для выполнения вычислений и прочие [2]. В некоторых случаях оптимизация по одному параметру приводит к ухудшению другого параметра. Например, уменьшение времени выполнения программы может быть получено путем увеличения кода программы.

Все методы оптимизации условно могут быть разделены на три уровня по месту применения:

- уровень алгоритма;
- уровень макрокоманд;
- уровень команд процессора.

Эти уровни не взаимоисключаемые, а дополняемые. Например, некоторый алгоритм может быть улучшен на самом высоком уровне математиком, затем программист может сформировать на его основе эффективную последовательность команд и на третьем этапе компилятор может преобразовать их в команды процессора с учетом сильных сторон конкретной архитектуры и специфики кода. Такое разделение позволяет выявить узкие места программ и эффективно их устранять. Следует отметить, что для наиболее эффективной оптимизации следует применять методы оптимизации по уровню от верхнего к нижнему, то есть от алгоритма к низкоуровневым оптимизациям. Например, при использовании на больших размерностях алгоритма дискретного преобразования Фурье вместо алгоритма быстрого преобразования Фурье нивелирует все более низкоуровневые оптимизационные методы за счет значительно большего количества операций, которые следует выполнить.

Уровни оптимизации можно разграничить на такие участки ответственности: алгоритмический – математик и программист, макрокоманд – программист и компилятор, микрокоманд – компилятор и

программист. Следует отметить, что базовые алгоритмы для известных задач давно описаны и известна их вычислительная сложность. Поэтому оптимизация на уровне алгоритмов, как правило, применима на больших программных комплексах со сложными алгоритмами и взаимодействиями данных. Оптимизация на уровне микрокоманд на языках высокого уровня выполняется силами компиляторов. При этом описание компиляторов не раскрывает всех способов, методов и эвристик, которые используют компиляторы для каждого возможного ключа оптимизации. Таким образом компилятор представляет собой некоторый «черный ящик» с небольшим набором доступных регулировок.

Оптимизация на уровне макрокоманд, то есть, по сути, оптимизация последовательности используемых команд, представляет собой большой интерес. Оптимизация на этом уровне не требует существенной переработки алгоритма и модификации компилятора, и, в то же время, может давать значительное улучшение оптимизируемого параметра.

Целью работы является оценка применения методов оптимизации, а также использование вспомогательных методов для поиска лучших параметров оптимизации.

Предметом рассмотрения в данной статье является один из методов оптимизации на уровне макрокоманд и способы наиболее эффективного использования этого метода.

Постановка задачи

Рассматривая практические задачи применения оптимизации программного обеспечения, следует отметить, что достаточно распространена оптимизация по времени выполнения программ. Действительно, целые классы вычислений, например, декодирование аудио- и видеопотоков, цифровая обработка сигналов, алгоритмы беспроводных и проводных способов передачи данных требуют строго ограниченного сверху времени вычислений. Незначительное превышение допустимых временных интервалов на вычисления в любом из возможных сценариев использования приводят к полностью нефункциональным программным комплексам и устройствам.

Также ускорение выполнения программного обеспечения может косвенно приводить к уменьшению энергопотребления, поскольку в режиме бездействия вычислительные блоки могут переводиться в режим уменьшенного энергопотребления.

Рассматривая методы оптимизации по скорости, выполнения следует отметить, что большинство из них сконцентрированы на вычислительных циклах, поскольку именно в вычислительных циклах, то есть в относительно небольших участках кода программы производят наибольшее время.

В настоящее время известно множество методов оптимизации вычислительных циклов:

- Разбиение циклов;
- Слияние циклов;
- Изменение последовательности выполнения циклов;
- Распараллеливание циклов [3];
- Раскрытие циклов;
- Разбиение циклов на несколько с меньшими блоками итерирования;
- Векторизация;
- Разбиение цикла на блоки.

Использование указанных и других методов оптимизации может значительно ускорить выполнение программ [4-5].

Некоторые из методов можно использовать совместно, но некоторые являются взаимоисключающими. Например, слияние циклов и разбиение цикла являются взаимообратными.

Задача поиска наилучшего метода оптимизации для конкретного участка кода или вычислительного цикла может быть представлена в виде формулы:

$$F_i = argmin(f(\vec{M}_i, \vec{P}_i)) \quad (1)$$

где F_i это значение оптимизируемого параметра для i-го участка кода, который должен быть оптимизирован, M – вектор методов оптимизации, P – вектор параметров методов оптимизации. Функция argmin

обозначает такое значение входных параметров f(x), при которых достигается минимальное значение функции:

$$\underset{x}{argmin}(f(x)) \in \{x \mid \forall y: f(x) \leq f(y)\} \quad (2)$$

Оптимизационная задача для всей программы, которая состоит из N вычислительных циклов может быть представлена в виде:

$$F = argmin(\sum_{i=1}^{N} f(\overrightarrow{M_i}, \overrightarrow{P_i})) \quad (3)$$

Метод разбиения цикла на блоки

Для математического описания вычислительных циклов используется несколько подходов [6-7]. 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].

$$L_i \leq x_i \leq U_i \quad (4)$$

где $i = 1 \ldots n$.

Неравенства (4) определяют границы циклов и пространство итераций, которое ограничивается выпуклым многогранником. В этом случае граф зависимостей рассматривают как множество векторов:

$$P = \{x | x \in Q^n, Ax \leq b\} \quad (5)$$

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.

Одним из эффективных методов оптимизации циклов является метод разбиения на блоки. Этот метод показывает эффективность не только при использовании сам по себе, но и может успешно сочетаться с другими методами оптимизации циклов.

Суть метода заключается в том, что итеративное пространство вычислительного цикла разбивается на меньшие области, так называемые блоки, и вычисление производится по таким блокам, а не по всему диапазону цикла.

В случае одномерного цикла код вида:

```
for (i=0; i<N; i++)  {  ...  }
```

будет преобразован в код:

```
for (i=j; i<min(N, j+B); i++)  {  ...  },
```

где B – размер блока разбиения. Таким образом, добавляются дополнительные вложенности цикла, но существенно уменьшается объем данных для каждой итерации, тем самым улучшая локальность данных и уменьшая кеш-промахи.

Особенностью метода есть то, что можно использовать несколько дополнительных вложенностей.

Рассмотрим вариант, когда исходный код содержит цикл двойной вложенности и метод разбиения на блоки использует два параметра. Пример использования метода представлен на рисунке 1.

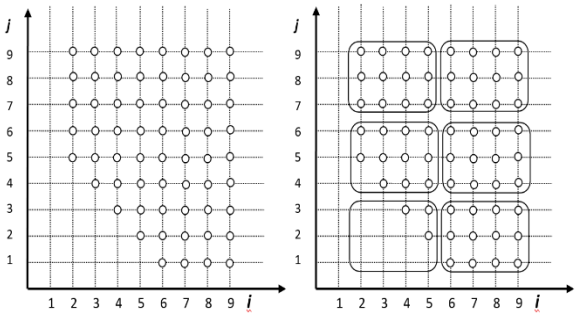

Рис. 1. Пример оригинального вычислительного цикла и вычислительного цикла с использованием метода разбиения на блоки

Как видно на рисунке справа, метод разбиения на блоки использует в данном случае блоки размером 4 и 3. В общем случае размеры блоков разбиения могут быть произвольными. В дальнейших разделах будет производен анализ размеров блоков.

Как следствие метода разбиения на блоки можно видеть, что вычисления

стали блочными и могут быть в дальнейшем распараллелены, если между блоками нет зависимостей по данным.

Эксперименты по верификации метода разбиения на блоки

Метод разбиения на блоки был предметом разработки разных исследовательских групп. В последнее время появились специализированные программные пакеты, которые позволяют делать преобразование исходного кода в исходный код в автоматическом режиме с применением как обычного, так и некоторых усовершенствованных методов разбиения на блоки. Также такие программные пакеты могут работать с другими методами оптимизации, например, с распараллеливанием.

Для проверки временной и энергетической эффективности работы программы автоматической оптимизации, основанной на полиэдральной модели, был выбран пакет Pluto 11.4 [9]. 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 [10].

На первом этапе производились замеры времени выполнения и энергопотребления. Результаты экспериментов приведены в [11]. Авторы показали, что для большинства тестовых задач метод разбиения на блоки показывает ускорение выполнения программ порядка 5-20% в зависимости от теста. Отдельно следует отметить значительный дальнейший рост быстродействия при совместном использовании метода разбиения на блоки и распараллеливании. Некоторые тестовые программы не получают никакого ускорения от применения метода.

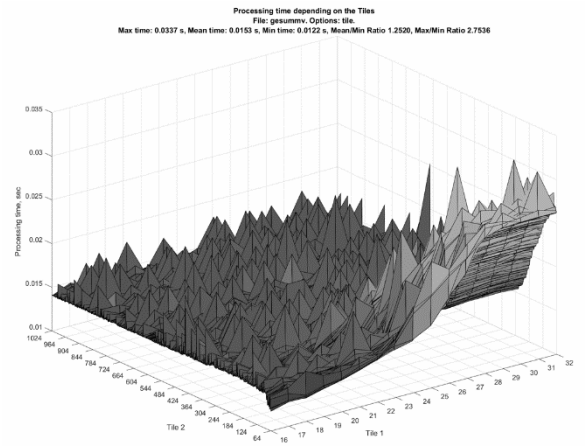

Рис. 2. Пример 1 зависимости времени выполнения от размеров блоков разбиения

На втором этапе экспериментов в [12] авторы провели исследование относительно размеров блоков разбиения. Полученные результаты выявили два важнейших факта: метод разбиения на блоки очень чувствителен к размерам блоков разбиения, при этом время выполнения программ варьируется в значительной степени, а также эксперименты показали, что графики зависимостей времени выполнения сильно разнятся по каждому тестовому алгоритму и не имеют какого-либо специфического паттерна. Примеры зависимостей времени выполнения от размеров блоков разбиения представлены на рисунках 2-4.

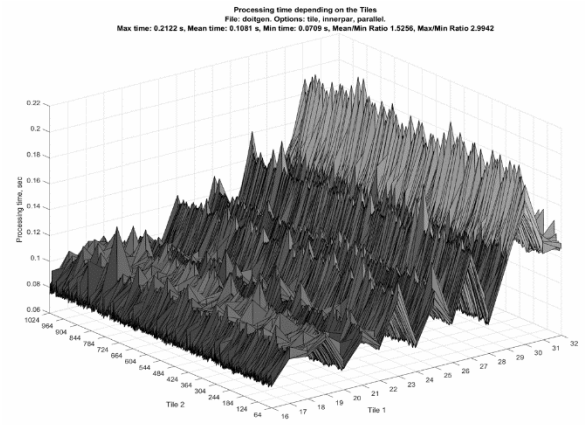

Рис. 3. Пример 2 зависимости времени выполнения от размеров блоков разбиения

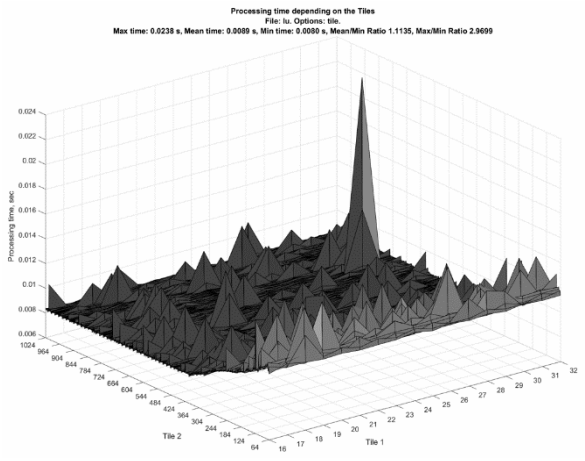

Рис. 4. Пример 3 зависимости времени выполнения от размеров блоков разбиения

Как видно на представленных рисунках, время выполнения программ значительно отличается при различных размерах блоков разбиения для разных алгоритмов. Поиск глобального минимума в такой зависимости не может быть найден по нескольким точкам или путем применения простых алгоритмов поиска минимума. Требуется алгоритм, который способен найти минимум функции при невыраженном характере функции.

Использование дискретного метода роя частиц в задаче поиска оптимального размера блоков разбиения

Для поиска экстремума в сложных многомерных функциях применяются различные методы. Отдельный класс методов поиска представляют генетические методы. Частным случаем генетических методов является метод роя частиц.

Метод роя частиц предусматривает наличие многих агентов поиска экстремума, которые информационно связаны между собой и в течении некоторого количества итераций производящие последовательный поиск экстремума. Авторами данный метод назван по аналогии с роем пчел, поведение жителей которого связано с поисками меда всех пчел улья.

Особенностью метода является то, что каждая частица на следующей итерации поиска учитывает как собственный найденный минимум, так и глобальный минимум всех частиц и корректирует свою текущую точку поиска также и с учетом глобального минимума. Также изменение координаты на текущей итерации учитывает предыдущее изменение координаты предыдущей итерации.

Множество частиц обозначается $P = \{P_i, i \in \overline{1..N}\}$, где N это число частиц в рое или популяции. В моменты времени t=0,1,2,… координаты частицы P_i определены вектором $X_{i,t} = (x_{i,t,1}, x_{i,t,2}, \dots, x_{i,t,n})$, и ее скорость вектором $V_{i,t} = (v_{i,t,1}, v_{i,t,2}, \dots, v_{i,t,n})$. r_1 и r_2 – случайные числа в диапазоне от 0 до 1. Начальные координаты и ускорения частицы P_i имеют вид $X_{i,0} = X_i^0$ и $V_{i,0} = V_i^0$ соответственно.

$$v_{i,t+1} = w v_{i,t} + c_1 r_1 [m_{i,t} - x_{i,t}] + c_2 r_2 [g_t - x_{i,t}] \qquad (6)$$

Частицы изменяют свои координаты и ускорения на каждой итерации. Количество частиц и итераций может быть сколь угодно большим.

Применительно к поиску оптимального набора размеров блоков разбиения частицы представляют собой пары, состоящую из двух размеров блоков разбиения. Соответственно, целевая функция – время выполнения программы.

Поскольку узлы итерационной решетки циклов строго целые числа, использовался дискретный метод роя частиц, отличием которого есть округление как координат, так и ускорений частиц, чем достигается переход от непрерывной в дискретную целочисленную обработку.

Определение эффективности работы производилось на основе классического распределения метода роя частиц, когда начальные координаты разбросаны по всей допустимой области допустимых значений. В [12] приведены результаты поиска оптимальных размеров блоков для примера на Рис. 5

В качестве тестового образца был использован один из тестовых примеров, в котором измерены значения времени выполнения программы для всей допустимой решетки размеров блоков разбиения.

Особенностью данного примера является наличие глобального минимума на краю области и наличие нескольких

локальных минимумов, разбросанных по всей площади.

Как показали результаты тестирования при 4-1024 частицах и 4-1024 итерациях поиск локального минимума может происходить относительно быстро и в дальнейшем с числом роста частиц и итераций не увеличивается. Достаточным может считаться выборка из 16-32 частиц и 32-256 итераций. В то же время глобальный минимум не был найден.

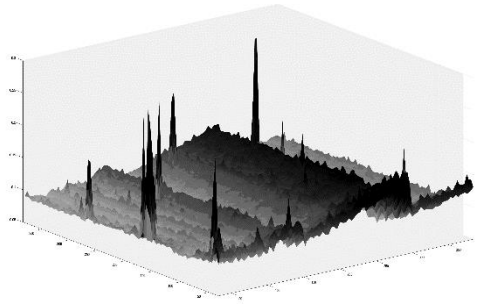

Рис. 5 – Тестовый пример для дискретного метода роя частиц.

Исследование начальных условий и параметров дискретного метода роя частиц

В предыдущем разделе использовался классический метод роя частиц, в котором начальные координаты частиц расположены в случайном порядке и с нулевым начальным ускорением. В то же время существует множество разновидностей метода, например, представленные в [13-16], которые характеризуются различными начальными условиями частиц.

Возвращаясь к формуле (6) расчета нового положения частицы на каждой итерации следует отметить, что сам метод роя частиц параметрический и включает в себя 3 коэффициента, которые влияют на поведение частиц, а именно:

1. w – инерциальный коэффициент, который способствует движению частицы в ту же сторону, что и на предыдущей итерации;

2. c1 – индивидуальный коэффициент, который характеризирует приближение к собственно найденному минимуму;

3. c2 – социальный коэффициент, который характеризует приближение к глобальному минимуму.

В предыдущем разделе использовались значения w = 0,7298, c1 = c2 = 1,49618. Очевидно, что некоторые другие исходные коэффициенты могли показать другие результаты. Для оценки влияния всех коэффициентов на поведение частиц проведем серию экспериментов, разместив частицы на периметре допустимой области. Для лучшей наглядности будем производить тесты с 8 частицами и 8 итерациями. Вектор начального ускорения направлен для всех частиц внутрь допустимой области. Полученные результаты приведены на рисунках ниже.

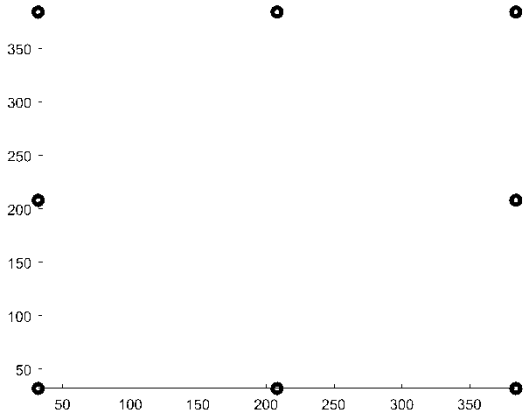

Рис. 6. Результаты при w = 0, c1 = 0, c2 = 0. Частицы не двигаются так как все коэффициенты нулевые. Такой ж рисунок при $c1 \neq 0$.

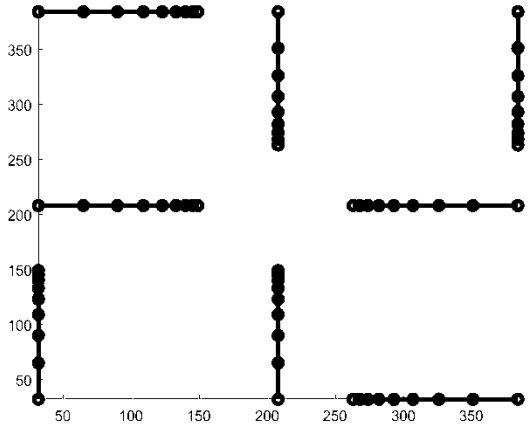

Рис. 7. Результаты при w=0.75, c1 = 0, c2 = 0. Частицы начинают движение внутрь области, но со временем скорость движения уменьшается, так как нет ни индивидуальной, ни социальной составляющей.

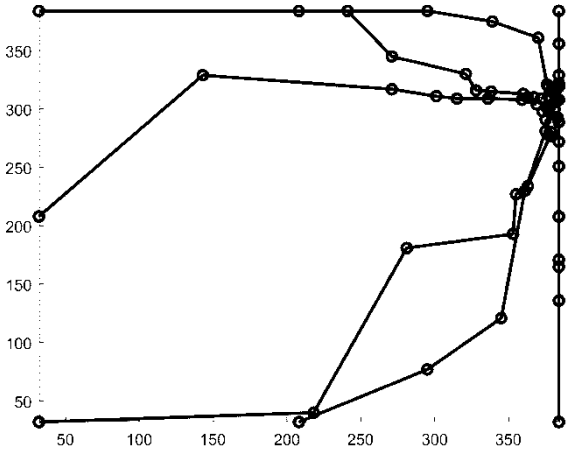

Рис. 8 – Результаты при w=0, c1 = 0, c2 = 0.75. На частицы влияет только социальный коэффициент 0.75.

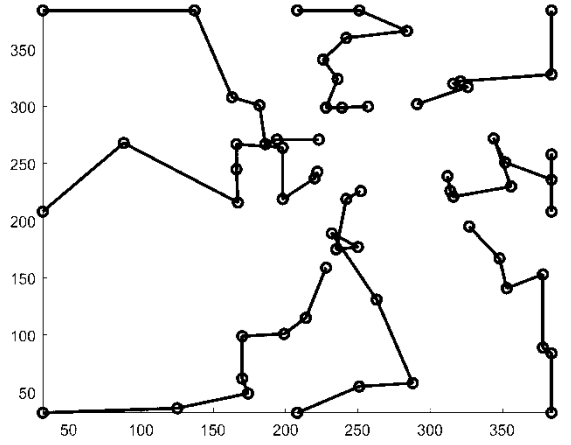

Рис. 9 – Результаты при w=0, c1 = 0, c2 = 0.375. На частицы также влияет только социальный коэффициент, но он в 2 раза ниже, чем на предыдущем рисунке.

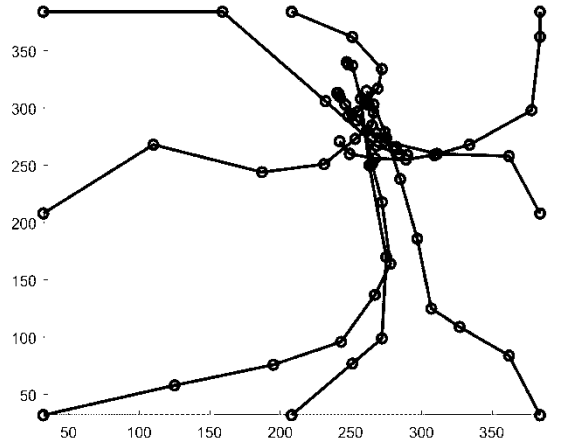

Рис. 10 – Результаты при w=0.5, c1 = 0, c2 = 0.375. Только начальное ускорение и социальный фактор.

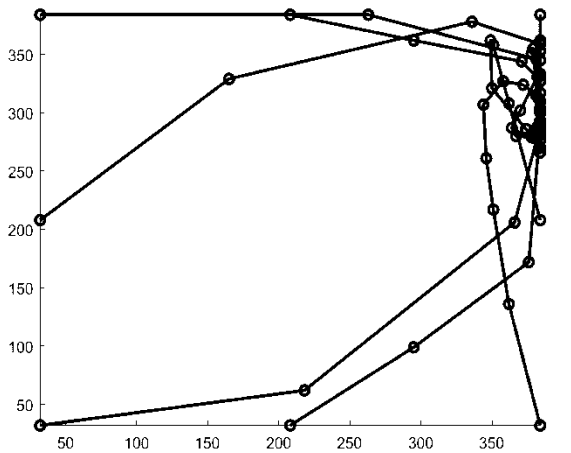

Рис. 11 – Результаты при w=0.5, c1 = 0, c2 = 0.75. Относительно Рис. 10 социальный фактор удвоен.

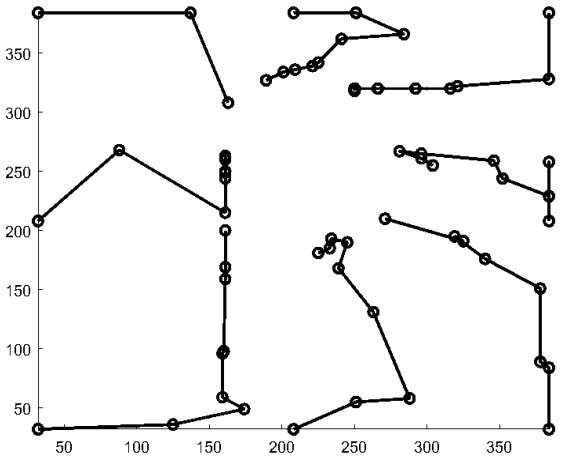

Рис. 12 – Результаты при w=0, c1 = 0.375, c2 = 0.375.

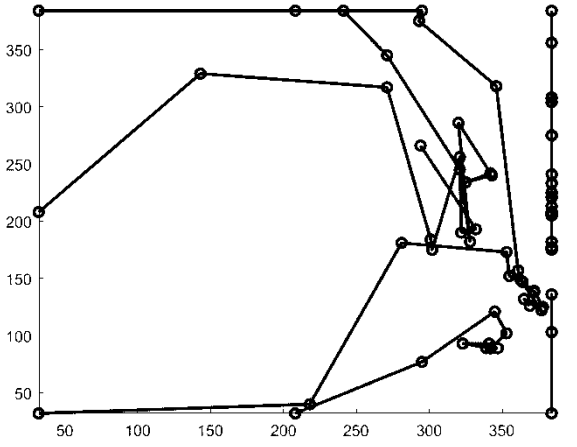

Рис. 13 – Результаты при w=0, c1 = 0.75, c2 = 0.75. Социальный и индивидуальный коэффициенты удвоены.

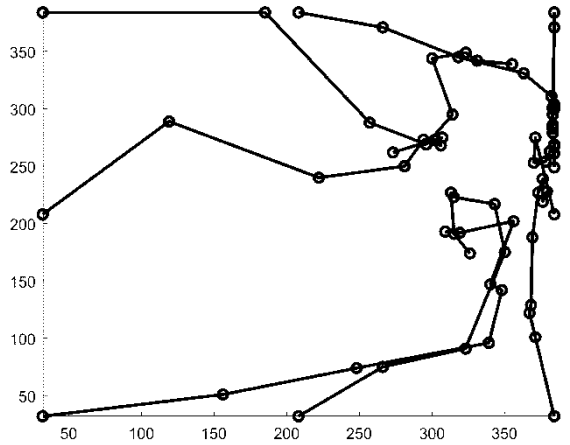

Рис. 14 – Результаты при w=0.3, c1 = 0.5, c2 = 0.5.

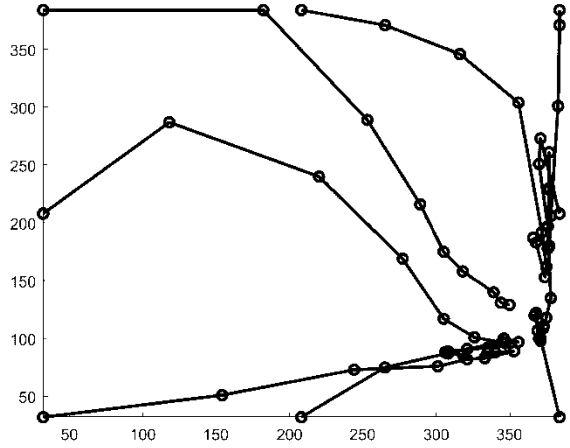

Рис. 15 – Результаты при w=0.3, c1 = 0.5, c2 = 0.49. Относительно предыдущего рисунка уменьшен на 0.01 социальный коэффициент.

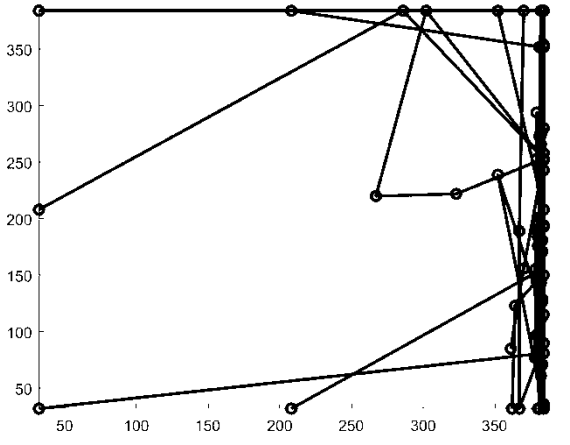

Рис. 16 – Результаты при w=0.7298, c1 = 1.49618, c2 = 1.49618. Коэффициенты как со случайно расположенными частицами.

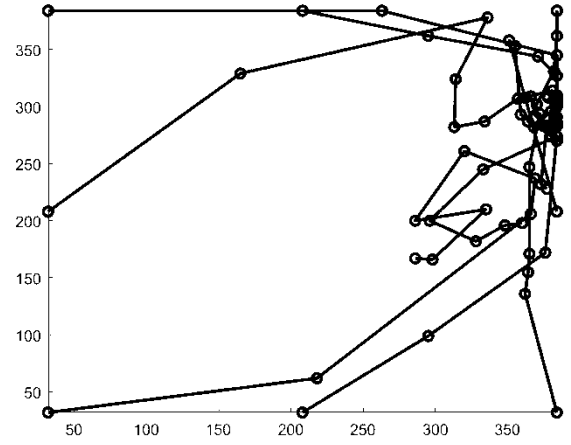

Рис. 17 – Результаты при w=0.5, c1 = 0.75, c2 = 0.75.

Выводы

В ходе исследования показано изменение координат частиц в дискретном методе роя частиц применительно к поиску оптимального размера блока разбиения при изменении коэффициентов метода.

Характер полученных зависимостей показывает сложную зависимость от используемых коэффициентов. Действительно, найденный на некоторой итерации глобальный минимум какой-либо из частиц оказывает влияние на все частицы роя. При этом разница всего на 0.01 социального коэффициента может приводить к качественно другой картине частиц, как отображено на рисунках 14 и 15.

Литература

1. *A. Aho, M. Lam, R. Sethi and J. Ullman,* Compilers, principles, techniques, and tools. Pearson Education, Inc., 2007.

2. *K. Kennedy, R. Allen,* Optimizing Compilers for Modern Architectures – A Dependence Based Approach. San Francisco: Morgan Kaufmann Publishers, 2001.

3. *Shinan W.* Software power analysis and optimization for power-aware multicore systems: дис. канд. / Shinan Wang – Detroit, 2014. – 177 с.

4. *Jingling X.* Loop tiling for parallelism / Xue Jingling. – (Kluwer international series in engineering and computer science; SECS 575).

5. *Pugh W.* An Exact Method for Analysis of Value-based Array Data

Dependences / W. Pugh, D. Wonnacott. – Univ. of Maryland, 1993.

6. *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.

7. *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.

8. *Uday Bondhugula*. Effective Automatic Parallelization and Locality Optimization Using The Polyhedral model: дис. канд. / Uday Bondhugula. – The Ohio State University, 2010. – 193 с.

9. *U. Boundhugula, J. Ramanujam and P. Sadayappan*, "Pluto: a practical and fully automatic polyhedral parallelizer and locality optimizer", Louisiana State University, Columbus, 2007.

10. *L. Pouchet,* "PolyBench/C the Polyhedral Benchmark suite". [Electronic resource]. Available at: http://web.cse.ohio-state.edu/~pouchet/software/polybench/#description.

11. *Сушко С.В.* Summer InfoCom 2016: Матеріали II Міжнародної науково-практичної конференції, м. Київ, 1-3 червня 2016 р. – К.: Вид-во «Інжиніринг», 2016. – 116 с.

12. *Сушко С.В., Чемерис О. А.* Вплив розмірів блоків розбиття операторів циклів на час виконання комп'ютерних программ Моделювання та інформаційні технології, 2018. – Vol. 82. – P. 110-117.

13. *Clerc M.* Particle Swarm Optimization. ISTE, London, UK, 2006.

14. *П. В. Матренин, В. Г. Секаев* Системное описание алгоритмов роевого интеллекта, Программная инженерия, Теоретический и прикладной научно-технический журнал, 2013. – №12. – С. 39–45.

15. *Ковалев И.В., Соловьев Е.В., Ковалев Д.И., Бахмарева К.К., Демиш А.В.* Использование метода роя частиц для формирования состава мультиверсионного программного обеспечения, Приборы и системы. управление, контроль, диагностика. М.: Научтехлитиздат, 2013. – №3. – С. 1-6.

16. *G. Beni, J. Wang,* Swarm Intelligence in Cellular Robotic Systems, Proceed. NATO Advanced Workshop on Robots and Biological Systems, Tuscany, Italy, June 26–30, 1989.

Сушко С.В., Чемерис А.А.

ИССЛЕДОВАНИЕ ПАРАМЕТРОВ ДИСКРЕТНОГО МЕТОДА РОЯ ЧАСТИЦ ПРИ ПОИСКЕ ОПТИМАЛЬНЫХ РАЗМЕРОВ БЛОКОВ РАЗБИЕНИЯ ЦИКЛИЧЕСКИХ ОПЕРАТОРОВ ПРОГРАММ

Статья посвящена методам оптимизации программного обеспечения. В статье рассматриваются уровни оптимизации программного обеспечения и раскрываются основные методы оптимизации вычислительных циклов. Авторы описывают метод разбиения вычислительных циклов на блоки как один из перспективных методов оптимизации. Использование указанного метода на тестовых программах зачастую показывает ускорение времени выполнения программ. Однако сам метод разбиения на блоки является параметрическим и при использовании различных размеров блоков разбиения время выполнения тестовых программ также значительно разнится. При этом зависимость времени выполнения программ от выбранного значения размеров блоков неочевидная и не имеет схожих паттернов для разных тестовых программ. Авторами предложено использовать дискретный метод роя частиц как оптимизационный метод, который позволяет найти локальный либо глобальный минимум времени выполнения программ при различном характере зависимости между размерами блоков и времени выполнения. В статье произведено исследование влияния поведенческих коэффициентов дискретного метода роя частиц на скорость и характер нахождения

минимума времени выполнения. Как результат исследования продемонстрированы графики поиска частицами роя оптимального значения. Авторы делают заключение о сложности подбора коэффициентов, которые обеспечивают небыструю, но последовательную сходимость частиц роя к минимуму и предлагают набор коэффициентов, которые обеспечивают указанное поведение частиц.

Ключевые слова: *параллельные программы, распараллеливание программ, метод роя частиц, тайлинг.*

Sushko S.V., Chemeris O.A.

INVESTIGATION OF THE PARAMETERS OF DISCRETE PARTICLE SWARM OPTIMIZATION METHOD FOR PROBLEM OF FINDING OF OPTIMAL TILE SIZES IN PROGRAM LOOPS

The article is devoted to software optimization methods. The article considers levels of software optimization and reveals main methods of optimization of computational loops. The authors describe the method of tiling method as one of promising optimization methods. Using of the method on the test programs often shows an acceleration of program execution time. However, tiling method itself is parametric and with using of different sizes of tiles an execution time of test programs also varies significantly. In the same time, dependence between program execution time and chosen tile size is not obvious and does not have similar patterns for the different test programs. The authors proposed to use Discrete Particle Swarm Optimization Method as optimization method which allows to find a local or global minimum of program execution time for the different nature of relationship between tiles and execution time. The article investigates an influence of the behavioral coefficients of Discrete Particle Swarm Optimization Method on speed and character of finding of minimum execution time. As a result of investigation, the graphs of searching for optimal value of particles of swarm are demonstrated. The authors conclude that there are difficulties to choose the coefficients that provide not fast but consistent convergence of swarm's particles to minimum and they propose a set of coefficients that shows the desired behavior of particles.

Keywords: *parallel programs, program parallelization, particle swarm optimization, tiling.*