Оценка влияния различных участков программного кода на энергопотребление вычислительной системы

Е. А. Киселёв[1], Д. А. Чубаров[2], А. В. Баранов[3]

[1] МСЦ РАН – филиал ФГУ ФНЦ НИИСИ РАН, НИЦ «Курчатовский институт», Москва, Россия, kiselev@jscc.ru, +7(926)223-88-66;

[2] РТУ МИРЭА, Москва, Россия, chubarovdima@inbox.ru;

[3] МСЦ РАН – филиал ФГУ ФНЦ НИИСИ РАН, НИЦ «Курчатовский институт», Москва, Россия, anton.baranov@jscc.ru, +7(903)215-43-42.

Аннотация. Работа посвящена проверке гипотезы о возможности оценки влияния исходного кода программы на энергопотребление вычислительной системы. На основе исследованных методов оптимизации авторами предложен алгоритм для оценки энергоэффективности программного кода. Разработан макет программного средства в виде расширения для Visual Studio Code, реализующий представленный алгоритм. Приведены экспериментальные результаты по исследованию различных способов повышения энергоэффективности программного кода, а также результаты проверки работоспособности разработанного алгоритма.

Ключевые слова: энергоэффективность, анализ исходного кода, оптимизация программного кода, RAPL, VS Code

1. Введение

С каждым годом расширяется область применения высокопроизводительных вычислительных систем, возрастает их энергопотребление. С увеличением числа процессорных ядер в таких системах растет степень параллелизма, появляются его новые уровни, такие как векторная обработка, вычисления на графических процессорах или иных ускорителях. Усложняются параллельные алгоритмы, и пользователи суперкомпьютеров вместо разработки собственных алгоритмов и программ предпочитают применять готовые программные пакеты и коды. В этой связи перестают работать подходы к оптимизации параллельных программ, основанные на частичной или полной переработке параллельного алгоритма. Ряд проблем, например, наличие неустранённых зависимостей при векторизации программного кода, не могут быть решены без нарушения алгоритма работы программы [1]. При этом следует отметить, что для таких задач применение различных техник оптимизации программного кода позволяет не только значительно сократить время выполнения параллельного приложения, но и добиться сокращения энергопотребления вычислительной системы, на котором оно было запущено.

В настоящей работе мы сфокусировали внимание на исследовании влияния различных стратегий оптимизации программного кода на энергопотребление вычислительной системы. Нами предложен подход к выявлению участков программного кода, оказывающих наибольшее влияние на энергопотребление вычислительной системы. Отличительной особенностью предлагаемого подхода является отсутствие необходимости запуска программы на выполнение и, как следствие, сокращение затрат на лишние тестовые запуски.

Наше исследование можно условно разделить на две части. Первая часть посвящена анализу подходов к оптимизации программного обеспечения и экспериментальной оценке их влияния на энергопотребление. Полученные результаты позволяют оценить не только влияние различных стратегий оптимизации на энергоэффективность, но и энергозатраты на выполнение часто используемых вычислительных операций. Во второй части исследования представлен алгоритм автоматического выявления наиболее энергоемких участков программного кода, требующих наибольшего внимания со стороны разработчика. Для демонстрации работы предложенного алгоритма было разработано расширение для среды разработки Visual Studio Code и продемонстрирована его работа на примере двух тестовых параллельных программ.

2. Способы и алгоритмы повышения энергоэффективности программного кода

Для сокращения энергопотребления вычислительных систем при выполнении программ применяются два основных подхода. Первый предполагает оптимизацию программного кода (Energy-Aware Programming) [2,3], второй – оптимизацию выделения вычислительных ресурсов для выполнения каждого приложения (Energy Aware Scheduling) [4]. Преимуществом первого подхода является возможность заранее (до запуска программы) выявить участки программного кода, которые будут оказывать значительное влияние на энергопотребление, и оптимизировать их. Основным недостатком этого подхода является необходимость в наличии исходного кода программы. Преимуществом второго подхода является возможность сокращения энергопотребления на основе накопленных данных статистики использования ресурсов вычислительной системы без модификации исходного кода. Основным недостатком здесь является необходимость выполнения серии запусков программы для сбора статистических данных. При этом энергопотребление программы на одних и тех же вычислительных ресурсах может отличаться в зависимости от параметров запуска (например, числа используемых потоков).

Для оптимизации программного кода используют следующие различные стратегии.

1. Рефакторинг программного кода;
2. Оптимизация алгоритма программы;
3. Оптимизация на этапе компиляции программы.

Рефакторинг кода – это процесс переработки исходного кода программы путем изменения его внутренней структуры без изменения внешнего поведения с целью улучшения удобочитаемости, надежности, безопасности и производительности. В последнее время именно рефакторинг чаще всего применяется для повышения энергоэффективности программ. Большой вклад в исследование и развитие этого направления внесли работы [2, 3]. Авторами отмечено, что применение различных методов рефакторинга позволяет значительно сократить энергопотребление мобильных устройств под управлением операционной системы Android. В работе [4] продемонстрировано, как использование различных методов рефакторинга влияет на энергопотребление Java-программ. Результаты аналогичных исследований, но для большего числа методов рефакторинга и для программ на языке C++ представлены в работе [5]. Общим для всех работ является вывод о том, что рефакторинг программного кода может приводить как к сокращению, так и увеличению энергопотребления. При этом определено, что вне зависимости от приложений, могут быть выделены энергоэффективные методы рефакторинга [5]. В работе [6] исследовано влияние комбинаций методов рефакторинга кода Java и C#-программ для мобильных и настольных систем. Установлено, что не для всех случаев комбинирование энергоэффективных методов рефакторинга приводит к сокращению энергопотребления. Авторы отмечают необходимость продолжения исследования различных комбинаций методов рефакторинга, выявления совместимых друг с другом, а также продолжения работ по совершенствованию программных средств измерения энергопотребления.

В работе [7] авторы продемонстрировали влияние эффективности алгоритма программы на энергопотребление вычислительной системы. В ходе эксперимента была произведена сортировка числовых элементов пузырьковым алгоритмом с вычислительной сложностью $O(n^2)$ и алгоритмом «пирамидальной (кучной) сортировки» вычислительной сложности $O(n \times \log(n))$. В результате алгоритм «пирамидальной (кучной) сортировки» потребил в 1,5 раза меньше энергии, чем алгоритм «пузырьковой сортировки». В работе [8] авторы для решения головоломки «Ханойские башни» использовали рекурсивный и итеративный алгоритмы. Итеративный вариант решения головоломки приводил к увеличению энергопотребления в 5,14 раза по сравнению с рекурсивным. Результаты таких экспериментов демонстрируют существенное влияние алгоритма на энергопотребление вычислительной системы.

Проектирование циклов с соблюдением некоторых простых правил позволяет существенно снизить влияние их выполнения на энергопотребление вычислительной системы. Например, использование беззнаковых целочисленных счетчиков или нуля в качестве условия завершения обратного отсчета ускоряет выполнения цикла за счет использования меньшего количества регистров микропроцессора. Энергопотребление может быть снижено путем разворачивания циклов – объединения инструкций, которые вызываются в нескольких итерациях цикла, в одну итерацию. Разворачивание циклов снижает точность предиктора ветвлений, поскольку существует меньше ветвей, на которых предиктор может обучить свое поведение. Однако это также снижает частоту прерывания непрерывного потока последовательных выборок, так что в качестве совокупного эффекта потребление энергии на инструкцию уменьшается.

Циклы «вращения» и «опроса» являются другими потенциальными источниками повышенного энергопотребления. В циклах «вращения» процесс или поток многократно проверяет, является ли условие истинным, например, доступен ли ввод с клавиатуры или блокировка. Когда поток находится в цикле «вращения», он остается активным, не выполняя полезную задачу. Использование циклов «вращения» может быть эффективным, если потоки заблокированы на короткие периоды времени. Таким образом можно избежать накладных расходов от перепланирования процесса операционной системы или переключения контекста. В качестве альтернативы применяется метод «опроса». В этом случае поток переходит в спящее состояние до тех пор, пока не произойдет какое-либо событие. Программисту необходимо найти компромисс между экономией энергии, получаемой за счет более длительного пребывания в состоянии с пониженным энергопотреблением, и накладными расходами, возникающими из-за перепланирования или частых переходов состояний.

Операционные системы предоставляют программистам механизм управления потоками через интерфейс системных вызовов. Использование нескольких потоков и нескольких ядер обеспечивает лучшую производительность, ускоряет вычислительный процесс и, как следствие, сокращает совокупные затраты на энергопотребление [9].

Векторизация программного кода и использование расширенного набора инструкций, таких как SIMD, также позволяют ускорить вычисления и сократить энергопотребление.

Оптимизация на этапе компиляции программного кода осуществляется в автоматическом режиме с использованием оптимизирующих компиляторов, которые выполняют анализ и преобразования программ на разных уровнях абстракции, начиная от исходного кода, промежуточного кода, такого как трехадресный код, до ассемблерного и машинного кода. Анализы и преобразования могут иметь разные области действия. Они могут выполняться в пределах одного базового блока (локальные), между базовыми блоками, внутри процедуры (глобальные) или между границами процедур (межпроцедурные). Традиционно оптимизирующие компиляторы пытаются сократить общее время выполнения программы или использование ресурсов, таких как память. Сам процесс компиляции может быть выполнен до выполнения программы (статическая компиляция) или во время выполнения программы (динамическая компиляция). Это большое пространство проектирования является основной проблемой для разработчиков компиляторов. Необходимо рассмотреть множество компромиссных вариантов для того, чтобы оправдать разработку и реализацию конкретного прохода или стратегии оптимизации.

3. Результаты исследования способов увеличения энергоэффективности программного кода

С целью проверки влияния оптимизации программного кода на его энергоэффективность в рамках настоящего исследования проведена экспериментальная оценка наиболее результативных методов оптимизации программного кода.

Первая серия экспериментов была направлена на проверку гипотезы из работы [10] о том, что для копирования массива эффективнее использовать встроенные функции вместо циклов. Вторая серия экспериментов была посвящена проверке другой гипотезы из работы [10] о том, что локальность при обращении к данным может снизить общее энергопотребление системы.
Третья серия экспериментов производилась с целью проверки гипотезы R4 [11], в которой предложено использовать единственный цикл вместо нескольких вложенных для сокращения энергопотребления.

Так как в [11] рассматривались не только отдельные методы оптимизации программного кода, но и совокупность этих методов, было принято решение в четвертой серии экспериментов оценить совокупность способов под обозначением C4, которая включает в себя: замену ссылки на саму переменную, добавление переменных в код для упрощения конструкций и удаление вложенных циклов.

Приведённые в таблице 1 результаты экспериментов демонстрируют эффективность перечисленных выше методов. Отдельно стоит выделить эффективность способа, связанного с векторизацией.

4. Влияние типовых операций на энергопотребление

Рассмотрим оценку влияния на энергопотребление часто используемых типовых вычислительных операций. В работе [10] было показано, что центральный процессор (ЦП) является бóльшим потребителем электроэнергии по сравнению с оперативной памятью (ОП) и дисковой подсистемой. Однако в [10] не показано, какие операции, выполняемые ЦП, являются наиболее энергозатратными. Также следует отметить, что в указанной работе эксперименты проводились с твердотельным накопителем (SDD), а не с жест-

ким диском (HDD), обладающим меньшей скоростью и большим энергопотреблением.

С целью определения влияния различных операций на энергопотребление ЦП нами была проведена серия измерений. В ходе каждого измерения исследуемые операции (сложение, разность, деление, умножение) повторялись в цикле $10^6, 10^7, 10^9, 10^{11}, 10^{13}$ раз. Полученные значения были усреднены и представлены на графиках (рис. 1-4).

В ходе проведения экспериментов было замечено, что с определённого момента потребление энергии перестаёт резко расти. Это связано с имеющимися ограничениями ЦП и невозможностью одновременного выполнения требуемого количества вычислительных операций. При большом количестве однотипных операций образуется очередь на обработку, что приводит к замедлению роста энергопотребления и увеличению времени обработки всех операций. По этой причине рост энергопотребления замедляется.

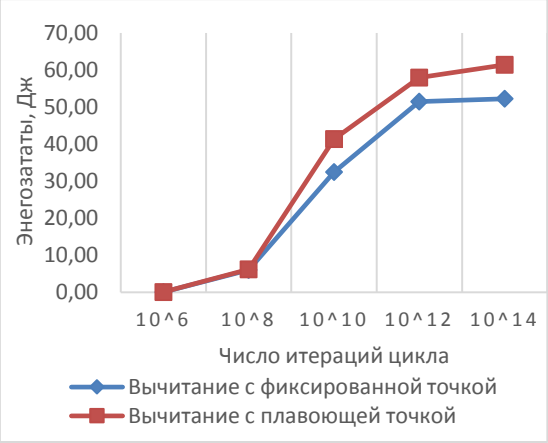

Рис. 1. Энергозатраты на выполнение операций разности с плавающей и фиксированной точкой

Рассмотрим результаты эксперимента по оценке энергозатрат при выполнении операций чтения файла, приведенные на рисунках 5 и 6. На графиках видно, что при обращении к жёсткому диску (рис. 5) расходуется значительно больше энергии по сравнению с арифметическими операциями. В этом случае на энергопотребление оказывает влияние не только частота обращения к жесткому диску, но и объём данных (рис. 6). Энергозатраты на обращение к оперативной памяти сопоставимы с арифметическими операциями (рис. 7).

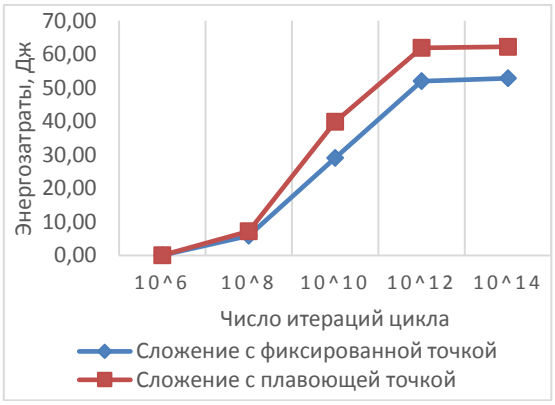

Рис. 2. Энергозатраты на выполнение операций сложения с плавающей и фиксированной точкой

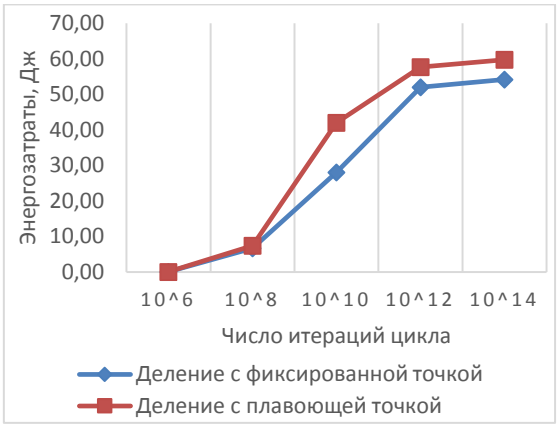

Рис.3. Энергозатраты на выполнение операций деления с плавающей и фиксированной точкой

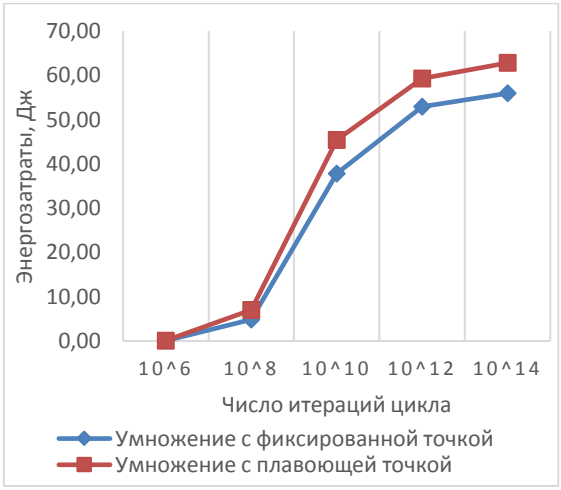

Рис.4. Энергозатраты на выполнение операций умножения с плавающей и фиксированной точкой

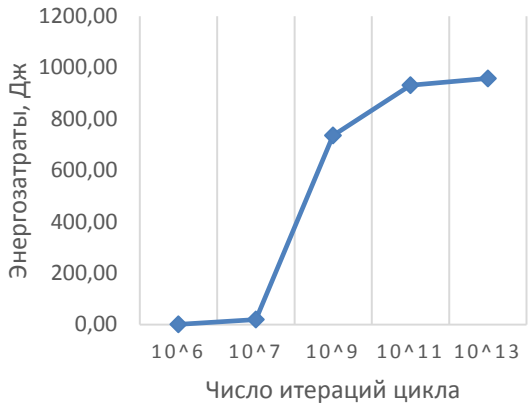

Рис. 5. Энергозатраты на операцию записи символа в файл в зависимости от числа итераций

Рис. 6. Энергозатраты на операцию чтения данных из файла

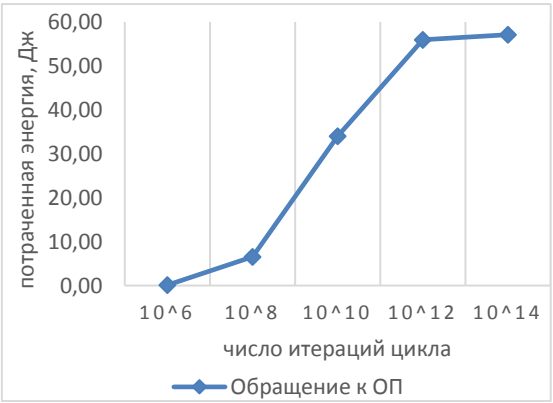

Рис. 7. Энергозатраты на запись символа в оперативную память в зависимости от числа итераций

5. Алгоритм оценки энергоэффективности исходного кода программ

Анализ результатов экспериментов позволяет сделать вывод о практической возможности выявления энергозатратных операций до запуска программы. Например, участки программного кода с вложенными циклами for или while, операции обращения к файловой системе, операции над числами с плавающей точкой и другие операции могут быть отранжированы в соответствии с их энергозатратностью. Полученные данные могут быть использованы при компиляции на этапе семантического анализа программного кода или в среде разработки на этапе профилирования и отладки программы.

С учётом полученных данных нами предложен алгоритм выявления участков программного кода с повышенным энергопотреблением, блок-схема которого представлена на рисунке 8.

Рассмотрим основные шаги алгоритма.

Шаг 1. Инициализировать переданные в качестве входных аргументов множества:

- множество O включает в себя упорядоченный набор конструкций (операции с фиксированной, плавающей точкой, операции доступа к ОП, HDD, циклы и др.), которые будут использоваться для анализа;

- множество W включает в себя коэффициенты, которые соответствуют конструкциям из множества O (например, операциям с фиксированной, плавающей точкой и операции доступа к ОП может соответствовать 1, а операциям доступа к HDD и циклам, как более энергозатраным – 2 и т.д.);

- множество S включает в себя набор строк программного кода, которые будут анализироваться.

Шаг 2. Выполнить N итераций цикла, где N – число элементов во множестве S. На каждой итерации цикла выполнить шаги 2.1-2.3.

Шаг 2.1. Взять очередную строку из множества S и найти соответствующую ей конструкцию из множества O. Отметить, если строка является началом или концом цикла.

Шаг 2.2. Найденной на шаге 2.1 конструкции определить число из множества W, присвоить его текущей строке из множества S и перейти к шагу 2.3.

Шаг 2.3. Проверить, находится ли текущая строка из множества S в цикле (в том числе и во вложенных циклах). Если строка входит в цикл, то к присвоенному на шаге 2.2 значению прибавляется значение всех циклов, в которые она входит.

Таким образом, после окончания работы алгоритма на выходе формируется модифицированное множество S, где каждой строке соответствует число, характеризующее ее энергозатраты.

Алгоритм является сходящимся, так как число его итераций зависит от числа объектов в конечном множестве S. Поскольку в алгоритме реализован однократный обход всех строк кода, его вычислительная сложность не превышает O(n), где n – количество строк кода.

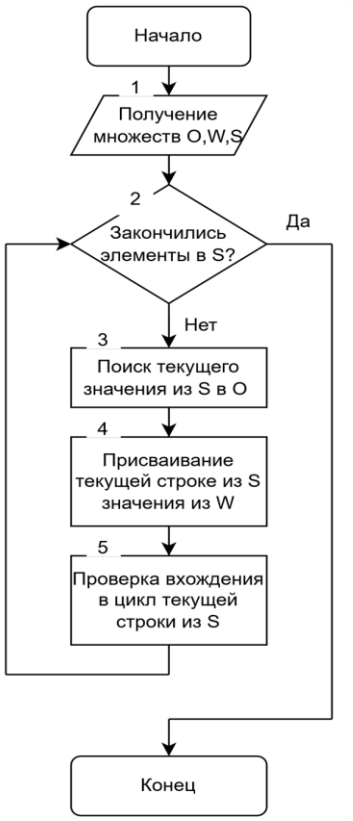

Рис. 8. Блок-схема алгоритма определения влияния заданного участка кода на энергопотребление

6. Проверка алгоритма

Предложенный алгоритм был реализован в виде расширения редактора программного кода Visual Studio Code (далее – VS Code) [12]. Разработанное расширение, названное Energy Mapper, включает в себя модуль анализа программного кода и модуль визуализации результатов анализа. Схема взаимодействия программных модулей приведена на рисунке 9.

Для работы с разработанными модулями пользователь должен открыть интересующий его файл в среде разработки VS Code и запустить Energy Mapper.

Открытый файл передаётся на вход модуля анализа программного кода, который возвращает коэффициенты для каждой строки в соответствие с шагами 2.2 и 2.3 алгоритма. Модуль визуализации получает эти коэффициенты и в соответствии с ними подсвечивает строки программного кода в среде разработки.

Проверка работоспособности модулей была проведена при помощи программ, реализующих алгоритмы быстрой сортировки и решения задачи Коши. Сначала было проведено измерение энергопотребления при запуске программ без изменений программного кода, а затем – с оптимизацией энергозатратных участков кода, отмеченных в Energy Mapper, как показано на рисунке 10. Измерения значений энергопотребления производилось с помощью библиотеки RAPL Stopwatch [13].

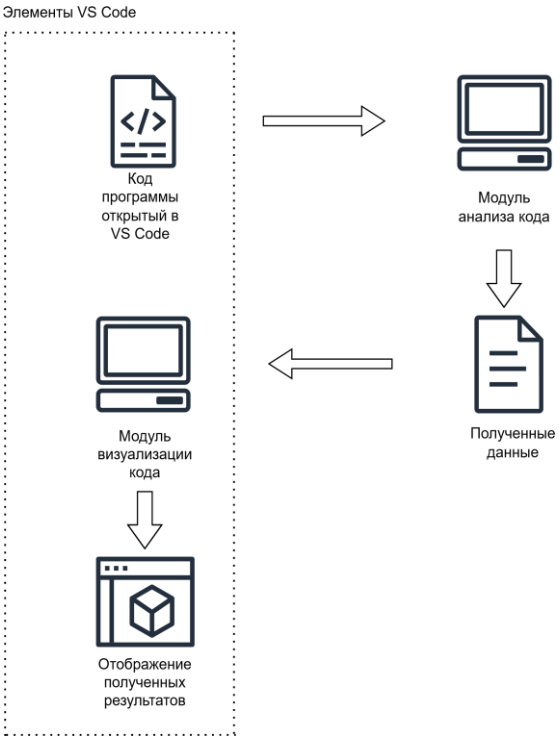

Рис. 9. Структура разработанных модулей

Рис. 10. Демонстрация фрагмента программного кода, размеченного модулем визуализации

Из результатов, приведённых в таблице 2, видно, что удалось достичь уменьшения энерго-

потребления при выполнении тестовых программ. При этом в эксперименте, связанным с задачей быстрой сортировки, удалось достичь более «скромных» результатов по сравнению с задачей Коши. Это связано с тем, что в программном коде алгоритма быстрой сортировки присутствует энергозатратная функция генерации псевдослучайных чисел rand().

Разработанное расширение Energy Mapper предоставляет возможность при помощи конфигурационных настроек задавать собственные новые сущности (функции или методы) для анализа.

Заключение

Результаты проведенных исследований позволяют сделать ряд выводов.

Во-первых, рефакторинг программного кода позволяет снизить накладные расходы на время выполнения параллельной программы и ее влияние на энергопотребление вне зависимости от выбранного языка программирования. Могут быть выделены языковые конструкции и функции оказывающие наибольшее влияния на энергопотребление. По мнению авторов, представляется целесообразным провести их классификацию.

Во-вторых, предложенные в настоящей работе алгоритм и программное средство Energy Mapper демонстрируют практическую возможность оценки энергозатрат на выполнение программ до их запуска на вычислительной системе.

В-третьих, результаты проведенных нами экспериментов с программной реализацией алгоритмов быстрой сортировки и решения задачи Коши продемонстрировали целесообразность использования расширения Energy Mapper для оптимизации исходных текстов программ.

Работа выполнена в рамках государственного задания по теме FNEF-2024-0016.

Таблица 1. Оценка влияния оптимизации кода на энергоэффективность

Способы оптимизации программного кода	**Изменение мощности в %**	**Изменение времени в %**	**Изменение потреблённой энергии в %**
Использование функции копирования данных вместо цикла	+14.09	-86.01	-84.11
Использование локальности при обращении к данным	+2.95	-10.73	-8.12
Использование векторизация вместо последовательного выполнения	+12.61	-98.48	-98.3
Использование совокупности способов оптимизации программного кода	+9.07	-98.66	-98.59

Таблица 2. Оценка влияния оптимизации кода на энергоэффективность

	Потреблённая энергия кода (Дж)	**Изменение потреблённой энергии**
Задача быстрой сортировки	689134	0%
Оптимизированная задача быстрой сортировки	588767	-14.6%
Задача Коши	527951	0%
Оптимизированная задача Коши	173301	-67.1%

Assessing of the Impact of Source Code Different Sections on the Computing System Power Consumption

E. A. Kiselev, D. A. Chubarov, A. V. Baranov

Abstract. The hypothesis about the possibility of assessing the impact of the program source code on the computing system power consumption is tested in the work. The authors proposed an algorithm for assessing the program code energy efficiency based on the studied optimization methods. The software tool prototype as an extension for Visual Studio Code implementing the presented algorithm was developed. Experimental results on the study of various ways to improve the program code energy efficiency, as well as the results of testing the developed algorithm operability are presented.

Keywords: energy efficiency, code analysis, code optimization, RAPL, VS Code

Литература

1. Юрченко А. В. Проектирование и анализ программного обеспечения с низким энергопотреблением с помощью программных метрик энергоэффективности // Машиностроение и компьютерные технологии. 2013. №1. С. 215-234.

2. da Silva, W.G.; Brisolara, L.; Corrêa, U.B.; Carro, L. Evaluation of the impact of code refactoring on embedded software efficiency. In Proceedings of the 1st Workshop de Sistemas Embarcados, Gramado, Brazil, 24–28 May 2010; pp. 145–150.

3. Gottschalk M., Jelschen J., Winter A. Energy-efficient code by refactoring. Softwaretechnik-Trends 2013, 33, 23–24.

4. Sahin, C.; Pollock, L.; Clause, J. How do code refactorings affect energy usage? In Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, Torino, Italy, 18–19 September 2014; pp. 1–10.

5. Park, J.J.; Hong, J.E.; Lee, S.H. Investigation for Software Power Consumption of Code Refactoring. In Proceedings of the Twenty Sixth International Conference on Software Engineering and Knowledge Engineering (SEKE), Vancouver, BC, Canada, 1–3 July 2014; pp. 717–722.

6. Hayri Acar, Gülfem I Alptekin, Jean-Patrick Gelas, Parisa Ghodous. The Impact of Source Code in Software on Power Consumption // International Journal of Electronic Business Management, Vol. 14, pp. 42-52 (2016).

7. T. Johann, M. Dick, S. Naumann, and E. Kern, “How to measure energy-efficiency of software: Metrics and measurement results” in Green and Sustainable Software (GREENS), 2012 First International Workshop on, 2012, pp. 51–54.

8. A. Noureddine, A. Bourdon, R. Rouvoy, and L. Seinturier, “A preliminary study of the impact of software engineering on GreenIT,” in Green and Sustainable Software (GREENS), 2012 First International Workshop on, 2012, pp. 21–27.

9. Y. D. Liu, “Energy-efficient synchronization through program patterns,” in Green and Sustainable Software (GREENS), 2012 First International Workshop on, 2012, pp. 35–40.

10. Hayri Acar, Gülfem I Alptekin, Jean-Patrick Gelas, Parisa Ghodous. The Impact of Source Code in Software on Power Consumption // International Journal of Electronic Business Management. 2016. №14. С. 42-52.

11. İbrahim Şanlıalp, Muhammed Maruf Öztürk, Tuncay Yiğit. Energy Efficiency Analysis of Code Refactoring Techniques for Green and Sustainable Software in Portable Devices // Electronics. 2022. №11. С. 442-459.

12. Visual Studio Code Getting Started [электронный ресурс] // Microsoft URL: https://code.visualstudio.com/docs/ (дата обращения 26.10.2024)

13. Github. The RAPL Stopwatch library [электронный ресурс] // https://github.com/LorienLV/rapl_stopwatch?ysclid=m2qc5vo4zz264735644 (дата обращения 26.10.2024).