25. *Pshikhopov V.Kh., Medvedev M.Yu*. Upravlenie podvizhnymi ob"ektami v opredelennykh i neopredelennykh sredakh [Management of moving objects in certain and uncertain environments]. Moscow: Nauka, 2011, 350 p.
26. *Pshikhopov V.Kh.* Pozitsionno-traektornoe upravlenie podvizhnymi ob"ektami [Position-trajectory control of mobile]. Taganrog: Izd-vo TTI YuFU, 2009, 183 p.
27. *Pshikhopov V.Kh., Medvedev M.Yu*. Strukturnyy sintez avtopilotov podvizhnykh ob"ektov s otsenivaniem vozmushcheniy [Structural synthesis of autopilots moving objects with the estimation of disturbances], *Informatsionno-izmeritel'nye i upravlyayushchie sistemy* [Information-measuring and Control Systems], 2006, No. 1, pp. 103-109.
28. *Pshikhopov V.Kh.* Organizatsiya repellerov pri dvizhenii s mobil'nykh robotov v srede s prepyatstviyami [Organization of repellere the motion of mobile robots in environment with obstacles], *Mekhatronika, avtomatizatsiya, upravlenie* [Mechatronics, Automation, Control], 2008, No. 2, pp. 34-41.
29. *Pshikhopov V.Kh., Sirotenko M.Yu., Gurenko B.V.* Strukturnaya organizatsiya sistem avtomaticheskogo upravleniya podvodnymi apparatami dlya apriori neformalizovannykh sred [Structural organization of systems of automatic control of underwater vehicles for a priori non-formal environments] *Informatsionno-izmeritel'nye i upravlyayushchie sistemy. Intellektual'nye i adaptivnye roboty* [Information-measuring and Control Systems. Intelligent and adaptive robots], 2006, No. 4, pp. 73-79.
30. *Gurenko B.V.* Postroenie i issledovanie matematicheskoy modeli avtonomnogo neobi-taemogo podvodnogo apparata [Construction and investigation of mathematical models of Autonomous underwater vehicle], *Inzhenernyy vestnik Dona* [Engineering journal of Don], 2014, No. 4. Available at: http://www.ivdon.ru/ru/magazine/archive/n4y2014/2626.
31. *Gayduk A.R.* Upravlenie gruppoy bespilotnykh letatel'nykh apparatov s ogranicheniem na upravlenie i peremennye sostoyaniya [Managing a group of unmanned aerial vehicles with restrictions on control and state variables], *MAU. Zhurn. v zhurn. «Upravlenie i informatika v aviakosmicheskikh i morskikh sistemakh»* [Mechatronics, automation, control. The journal in the journal "Management and Informatics in aerospace and marine systems"], 2012, No. 7, pp. 52-57.
32. *Gayduk A.R. Besklubova K.V.* Metody otsenki peremennykh sostoyaniya lineynykh sistem [Estimation methods of the linear system's state variables], *Izvestiya YuFU. Tekhnicheskie nauki* [Izvestiya SFedU. Engineering Sciences], 2012, No. 2 (127), pp. 8-13.
33. *Kiselev L.V.* Kod glubiny [Code depth]. Vladivostok: Dal'nauka, 2011, 332 p.
34. *Puzanov V.P.* Algoritmy upravleniya podvodnym apparatom v prodol'no-gorizontal'noy ploskosti. Nauchno-tekhnicheskiy otchet. [The algorithms for control of underwater vehicle in the longitudinal-horizontal plane. Scientific technical report]. Moscow: MVTU im. N.E. Baumana, 2004, 59 p.

Статью рекомендовал к опубликованию д.т.н., профессор Н.А. Глебов.

Гуренко Борис Викторович – Южный федерального университета; e-mail: boris.gurenko@gmail.com; 347928, г. Таганрог, пер. Некрасовский, 44; тел.: 89281687212; кафедра электротехники и мехатроники; ассистент.

Gurenko Boris Victorovich – Southern Federal University; e-mail: boris.gurenko@gmail.com; 44, Nekrasovskiy, Taganrog, 347928, Russia; phone: +79281687212; the department of electrical engineering and mechatronics; assistant.

УДК 004.4`422

И.П. Токарь

ИСПОЛЬЗОВАНИЕ ГЕНЕТИЧЕСКОГО АЛГОРИТМА В КОМПИЛЯТОРЕ ДЛЯ ОПТИМИЗАЦИИ ЭНЕРГОЭФФЕКТИВНОСТИ ПРИЛОЖЕНИЙ

Целью данной работы является нахождение техник компиляции, повышающих энергоэффективность программ. Данная задача формулируется как задача многокритериальной оптимизации производительности и энергопотребления, с ограничениями на время компиляции и падение производительности, где допустимое падение производительности

определяется пользователем реализации алгоритма. Программа рассматривается, как состоящая из регионов (где каждой функции соответствует максимум один регион), каждый из которых исполняется на разной частоте процессора, а, следовательно, имеет разное напряжения питания, потребление энергии и производительность. Для решения этой задачи предлагается и реализуется генетический алгоритм на основе NSGA2, который находит частоты для каждого региона, обеспечивающие максимальную энергоэффективность, при этом сохраняя производительность в заданных пользователем рамках. Данный алгоритм был реализован как проход в промышленном компиляторе GCC (GNU COMPILER COLLECTION), действующий после подстановки (inlining) функций. Этот проход считывает результаты профилирования при различных частотах и инструментирует генерируемый код вызовами функций понижения и повышения частоты, соответственно частотам для регионов, полученных алгоритмом. Приводятся результаты измерения его эффективности на физическом устройстве. Достигается 13,5 % рост энергоэффективности на некоторых тестах из пакета SPEC2000 (в частности 183.equake). Эти результаты достигаются, когда допустимым значением падения производительности принято 7 %. Достигнутый результат превышает 9 %, достигаемые методами, основанными на линейном целочисленным программировании. На наборе тестов EEMBC положительных или отрицательных результатов. Главным недостатком данного алгоритма является неспособность взаимодействовать с операционной системой и другими процессами в многоядерных системах.

Компиляторы; энергоэффективность; генетические алгоритмы.

I.P. Tocar

APPLICATION OF GENETIC ALGORITHM, IMPLEMENTED IN COMPILER FOR OPTIMIZATION OF APPLICATIONS ENERGY-EFFICIENCY

Purpose of this paper is to find a compilation techniques that improve energy-efficiency of programs. This problem is formulated as a multiple criteria (performance, energy-consumption) optimization problem, with constraints on compile time increase and performance decrease, where acceptable performance degradation is specified by user. Program is separated into a number of regions (at most one region per subroutine), running at different CPU frequencies, and due to difference in frequency and voltage consuming lower amount of energy, but having lower performance. A NSGA2-based genetic algorithm of finding frequencies for each region which provide maximal amount of energy-saving while maintaining performance within user provided range is proposed. It was implemented as compiler pass in GCC (GNU COMPILER COLLECTION), which consumed profiling data at different frequencies and instrumented code to run at frequencies provided by algorithm. Result of application of this algorithm are measured and provided. Namely 13.5 % energy-efficiency increase in archived on tests from SPEC2000 testsuite (in particular 183.equake). This result is archived when acceptable performance degradation is selected to be less than 7 %. Archived result is higher than 9%, which is current state-of-the-art, obtained by integer linear programming based approach. On EEMBC benchmark suite no improvement or degradation is found. Main disadvantage of this algorithm is inability to cooperate with operating system and other processes in multicore environment.

Compilers; energy-efficiency; genetic algorithms.

Введение. ***Важность энергоэффективности***. Энергоэффективность приложения – это отношение производительности к потреблённой энергии. В настоящее время к данной характеристике предъявляются повышенные требования. Эти требования вызваны, как увеличившейся распространённостью мобильных устройств (смартфонов), так и возрастающими затратами на электроэнергию для серверов. К примеру, для центров обработки данных, во многих случаях основные затраты владельцев составляет затраты на электроэнергию. Этот тренд привел к появлению сегмента микросерверов. Так по данным [1] уже в 2007 г. затраты на обеспечение электроэнергией центров обработки данных, только в США составили 4,5 миллиарда долларов.

В тоже время пользователи мобильных устройств могут предпочесть меньшую производительность, если это увеличит время автономной работы. Во многих случаях устройство даже с худшим отношением потребляемой мощности и производительности может быть предпочтительней за счёт меньшего абсолютного потребления, и, следовательно, большего времени автономной работы. Это объясняется тем, что для многих задач вычислительная мощность мобильных процессоров избыточна. При этом согласно опросам для большинства пользователей время автономной работы является очень важным показателем [2].

Хотя процессоры и системы на кристалле, особенно для встраиваемых систем, уже давно проектируются исходя в том числе и из требований энергопотребления и энергоэффективности, программные методы, особенно вне контекста операционных систем не применялись или редко использовались. В отличие от них широко использовались технологии позволяющие влиять на энергопотребление, динамически меняя часто- ту и/или усыпляя (временно отключая) определённые части процессора, такие как кэш память и вычисли- тельные устройства. Так, уже в 1992 году Intel внедрил технологию APM (Advanced Power Managment) [3] для уменьшения потребления при отсутствии вычислительной нагрузки. В настоящее время он заменён стандартом ACPI [4]. В основном это достигалось средствами самого процессора, иногда при взаимодействии с операционной системой. Компиляторы же, в основном, оптимизировали скорость выполнения или размер кода. Минимизация энергопотребления и максимизация энергоэффективности в результате оптимизаций компилятора либо вообще не рассматривалась, либо считалась следствием оптимизации по производительности.

Следует отметить разницу между оптимизацией энергопотребления (т.е. количеством энергии потребляемой в единицу времени) и энергоэффективно- стью (производительностью на единицу затраченной энергии). Не всегда минимальному потреблению соответствует максимальная производительность/единицу потреблённой энергии. Так, например, в [5] рассматриваются алгоритмы основанные на идее скорейшего завершения задачи (при этом потребление не минимизируется). Эти алгоритмы получили название race-to-idle. Их идея основана на малом потреблении процессоров в состоянии простоя. Зачастую выгодней максимально быстро выполнить задачу и перейти в состояние простоя. Также к их преимуществам бесспорно следует отнести простоту реализации. Действительно, и достижение максимальной производительности и снижение потребления в отсутствии нагрузки являются хорошо исследованными темами, что позволяет реализовать race-to-idle без особых усилий. В данной работе речь пойдёт именно об энергоэфффективности.

Энергопотребление КМОП транзисторов. Так как современные процессоры состоят из КМОП транзисторов, то стоит напомнить вид формулы потребления для них:

$$P = Pdynamic + Pleakage \quad (1)$$

$$Pdynamic \propto f * V\,2 \quad (2)$$

где P – общая мощность. Она складывается из динамического энергопотребление Pdynamic (при переключении транзисторов), зависящее от напряжения V и частоты f , и статического Pleakage, определяемого токами утечек. Так как в данной работе рассматриваются только методы снижения энергопотребления, доступные программно, то следует кратко описать, на какие части формулы, и как можно влиять из пользовательского программного обеспечения. Потребляемую мощность можно уменьшать временно выключая части процессора и тем самым снижая токи утечки и связанное с ними потребление, либо влияя на количество переключений транзисторов и таким образом на динамически потребляемую мощность. Компилятор всегда может влиять на инструкции, исполняемые процессором и таким образом изменять количество переключений транзисторов. Также иногда имеются способы изменять частоту или отключать части процессора.

Постановка задачи. Зададим математическую формулировку данной задачи.

$$\min E = P * t. \quad (3)$$

Будем искать минимум потреблённой энергии E, зависящей от мощности P и времени t при следующих ограничениях:

$$B \geq Bmin \quad (4)$$

$$tcomp \leq tmax \quad (5)$$

Производительность B не упадёт ниже какой-то отметки Bmin, а время компиляции – tcomp не поднимется выше tmax

Алгоритм оптимизации энергоэффективности в компиляторе. ***Ограничения и допущения.*** Опишем допущения и ограничения к поставленной ранее задаче (3): Будем рассматривать задачу максимального понижения энергопотребления для заданной, постоянно исполняемой задачи при заранее фиксированном допустимом падении производительности для процессора (системы на кристалле), допускающей программное управление напряжением и/или частотой, причём предположим, что изменяемая величина - общая для всего чипа. Требование единственности отсекает многоядерные процессоры с частотой, задаваемой индивидуально для каждого ядра. Это требование обусловлено необходимостью программно контролировать частоту с которой исполняется участок кода, что невозможно, когда планировщик процессов может изменить ядро на котором исполняется код (а соответственно и частоту) в любой момент.

Предполагается, что уже имеется способ измерения производительности, а устройство на котором она выполняется имеет способ измерения энергопотребления. Наиболее очевидным примером является исполнение какого-либо теста производительности на мобильном устройстве (имеющем батарею, допускающую программное измерение своего состояния), тогда потребление за заданный промежуток времени очевидно находится, как E1 −E2, где E1, E2 показания об энергии, запасённой в батарее, в начале и конце промежутка измерения. Другим примером может послужить сервер, подключённый к сети питания через измеритель потребляемой мощности. Стоит напомнить, что мы минимизируем энергию, потребляемую в единицу времени при исполнении теста.

Возможность улучшения тривиального алгоритма. Перед тем, как предлагать свой алгоритм, обоснуем возможность нелинейного падения производительности с уменьшением частоты. Как уже упоминалось, потребляемая процессором мощность пропорциональна квадрату напряжения и прямо пропорциональна частоте. Заметив также, что для подавляющего большинства современных процессоров для достижения меньшей частоты требуется меньшее напряжение, получим, как минимум квадратичную зависимость потребляемой мощности от частоты. Несмотря на неизменное потребление независящих от частоты процессора частей системы, видна возможность улучшения. Покажем также необходимость в нетривиальных алгоритмах выбора мест понижения частоты и возможность достижения более чем линейного роста при постоянном напряжении. Для этого рассмотрим тривиальный алгоритм. Пусть K – допустимо падение производительности, максимальная частота составляет Fmax, минимальная Fmin, время исполнения T , тогда найдём L – долю времени, которую мы будем исполнять с минимальной частотой, остальное время с максимальной частотой. Введём d – отношение максимальной и минимальной частот.

$$d = Fmin / Fmax. \quad (6)$$

Введём I – количество потраченных тактов.

$$I = Fmax * T. \quad (7)$$

И выразим его при применении алгоритма.

$$I = Fmax * T1 * (1 - L) + Fmin * T1 * L \quad (8)$$

$$I = Fmax * T1 + T1 * L * (Fmin - Fmax). \quad (9)$$

По определению K

$$T1 = T * (1 + K). \quad (10)$$

Приравнивая выражения для I получим:

$$Fmax * T = (1 + K) * T * (Fmax + L * (Fmin - Fmax)). \quad (11)$$

Или, после сокращений

$$1 = 1 + K + L * (d - 1). \quad (12)$$

Окончательно получаем:

$$L = K / (1 - Fmin/Fmax).$$

Казалось бы, что данный подход достигает поставленной задачи, если пренебречь постоянным потреблением всей системы. Однако стоит отметить чрезвычайно важный факт который определяет возможность улучшения представленного выше тривиального алгоритма. А именно, нелинейное изменение времени исполнения участков кода в зависимости от частоты. Данный эффект следует из того, что участок кода может быть целиком или частично упирающимся по производительности в задержки системы памяти (т.е. быть memory- bound). Приведём некоторые данные доказывающие этот факт. Так, при исполнении теста 176.gcc из пакета SPEC2000, на наборе входных данных scilab.i при частотах 1333 Мгц и 800 Мгц, для десяти наиболее часто исполняемых функций время исполнения приведено в табл. 1.

Таблица 1

Поведение функций при разной частоте

Время при 800Мгц (с)	Время при 1333Мгц(с)	отношение
6,844464	4,575753	1,495812
5,795504	3,378397	1,71546
3,317336	2,056065	1,613439
2,956756	1,93512	1,527945
1,481656	0,866773	1,709394
1,416096	0,846615	1,672656
1,298088	0,810332	1,601922
1,232528	0,782111	1,575899
1,1473	0,709544	1,616954
1,04896	0,632946	1,657267

Описание алгоритма. Обосновав возможность улучшения сформулируем наш алгоритм. Будем менять частоту отдельно для каждой функции. Откажемся от рассмотрения отдельных базовых блоков, так как это увеличивает объем задачи и затрудняет оценку результатов во время компиляции, из-за менее точного профилирования. Также предположим наличие профиля программы (возможно обойтись и без него, но это приведёт к значительному ухудшению точности). Не будем менять частоту в функциях с малым суммарным временем выполнения (порядка времени переключения частоты) или небольшим размером исполняемого кода (порядка размера кода необходимого для изменения частоты). Это обусловлено, ускорением работы алгоритма. Составим вектор частот, на которых исполняются рассматриваемые функции. Будем решать ранее поставленную задачу оптимизации с помощью генетического алгоритма.

Использование именно генетического алгоритма обусловлено рядом причин: Во-первых, абсолютно точное решение не гарантирует оптимальности результата, так как основывается на данных имеющих погрешность. То есть, математически верное решение задачи для каких-либо входных данных не обязательно лучшее возможное в реальности, так как входные данные не точны. Во-вторых, методы дающие точное решение, такие как целочисленное линейное программирование,

применявшиеся, например, в [7], требуют значительных вычислительных затрат, что приводит к значительному росту времени компиляции. Более подробно о времени компиляции при использовании целочисленного линейного программирования для решения задач оптимизации энергопотребления рассказано в [7] и [12]. В-третьих, генетические алгоритмы позволяют крайне просто менять время своего исполнения, для чего достаточно изменить количество итераций. Это позволяет свести задачу трёхкритериальной оптимизации к двухкритериальной.

Хорошее описание классического генетического алгоритма можно найти в [14]. Для учёта многокритериальности оптимизации будем использовать NSGA2 расширение алгоритма [18]. Вариация данного алгоритма, без применения NSGA2 представлялась автором в [19]. Выбор именно NSGA2 среди прочих генетических алгоритмов обусловлен его лучшей, чем у других генетических алгоритмов, использующих сортировку не доминирования, асимптотической сложностью. Он также хорошо зарекомендовал себя в реальных применениях (к примеру [15]) и относительно прост в реализации. Причём, параметр времени компиляции вынесен из задачи и учитывается как ограничение на максимальное число итераций алгоритма, задаваемое пользователем, как параметр компиляции. Известно, что для задач оптимизации с использованием генетических алгоритмов, особенно многокритериальных, наиболее сложным шагом является выбор вида целевой функции. В данном случае предлагается следующий подход: будем оптимизировать две функции:

$$T(v) = 2 - 1 / (\exp((T_{max} *(1 + P)) - T_v) * 10) \quad (14)$$

$$E(v) = 1 – E_v / E_{max}. \quad (15)$$

где T – оценка производительности (времени выполнения), E – оценка потребления, а индекс max соответствует значению при максимальной частоте. То есть часть, связанная с потреблением линейно растёт при уменьшении потребления, а часть связанная с производлительностью резко спадает при $T_v > T_{max} *(1 + P)$. При данном подходе ограничения вносятся в целевые функции и задача сводится к многокритериальной оптимизации без ограничений.

При этом выберем в качестве оператора мутации выберем оператор случайно изменяющий частоту для какой-то функции. А в качестве оператора кроссовера выберем оператор выбирающий частоту наиболее близкую к средней арифметической частот в каждом из векторов. Ясно, что результаты применения алгоритма зависят от способа оценки энергопотребления и производи- тельности для данного вектора частот. Можно выделить следующие способы (в порядке убывания точности и возрастания простоты реализации):

1. Использование множества профилей.
2. Использование одного профиля.
3. Использование модели.

Действительно при наличии множества профилей потребление и производительность какой-либо функции при какой-либо частоте уже известны, то есть всегда можно получить заведомо достоверную оценку потребления и производительности. При отсутствии

профилей приходится полагаться на некую программную модель, которая в принципе не может быть точнее и скорее всего менее точна. Однако использование профиля, а особенно большого их числа затрудняет реализацию данного алгоритма в компиляторах, не поддерживающих произвольные профили для PGO (profile – guided optimizations).

Видно, что желательно наличие хотя бы одного профиля программы.

Под профилем программы понимается время, потраченное внутри каждой функции и общую энергию потраченную на выполнение всей программы (измерение потребления индивидуально для каждой функции желательно, но при помощи

только показаний батареи невозможно). В случае множества профилей подразумевается наличие одного профиля для каждой допустимой частоты fi. При наличии всех необходимых профилей время исполнения для каждой функции на каждой частоте известно, что делает задачу оценки производительности тривиальной. Оценкой же потребления для функции на данной частоте будем считать, как произведение доли времени исполнения функции на данной частоте, к общему количеству энергии потреблённой на данной частоте.

Теперь опишем подход основанный на использовании одного профиля. Будем рассматривать следующую модель производительности отдельной функции от частоты:

$$P(f) = a * f + b, \quad (16)$$

где b зависит от подсистемы памяти. То есть для времени исполнения:

$$T(f) = Tmem + Tcpu/f. \quad (17)$$

То есть задача состоит в определении $\beta = Tmem / Tfmax$ доли кода зависящего от системы памяти, а не производительности процессора. Как видим основная проблема отсутствия других профилей состоит в определении β по одной точке. Можно применять такие эвристики, как доля обращений в память в данной функции, либо при наличии реализации у процессора аппаратной поддержки оценки производительности (hardware performance counters), долю промахов кэша в данной функции. Также стоит отметить, что влияние памяти особенно сильно для процессоров с малой либо нулевой степенью внеочередного исполнения (OOO – out of order), к которым относятся существующий процессоры для мобильных устройств (например, ARM). Зная набор частот и напряжений для этих частот, время, проведённое в функции на какой-либо частоте, общее потребление и оценку времени на новой частоте будем оценивать потребления на какой-либо частоте так:

$$E' = \sum(\alpha * T * (V'/V)^2 * f'/f) * E/T, \quad (18)$$

где V – напряжение, f – частота, E – потребление, T – время, α – доля функции в общем времени исполнения (учитывающая изменение частоты), отмечает значения при новой конфигурации, его отсутствие измеренные значения. Для случая модели не опирающейся на результаты профилирования, наиболее действенным способом является использование симулятора.

Сложность одной итерации алгоритма составляет O(n^2). Здесь n – численность популяции после применения операторов кроссовера и мутации, но до выбора наиболее подходящей части популяции. В оригинальной работе доказывается, что сложность равна $O(n2 * m)$, где m – количество критериев оптимизации, в данном случае m = 2. То есть алгоритм имеет квадратичную сложность и применим к реальным задачам. Также стоит заметить, что вынесение времени компиляции во внешний, задаваемый пользователем, как число итераций, параметр примерно в 1.5 раза ускоряет алгоритм.

Детали реализации. Опишем теперь, как данный алгоритм был реализован. Профилирование производилось с помощью утилиты командной строки perfrecord. Для бенчмарков имеющих встроенные средства измерения производительности, таких как SPEC, использовалось их значение производительности, для тестовых синтетических бенчмарков, использовалось значение, полученное с помощью утилиты time. Для каждого прогона измерялось и энергопотребление. Это делалось по показаниям батареи. Показания батареи брались из файла /proc/acpi/battery/BAT_0/state. Использовалось значение remaining_capacity. Затем полученные данные приводились к виду name Pf1 Ef1 Pf2 Ef2 .... Где name – имя функции, Pf1 – производительность, а Ef1 – потребление на частоте f1. Была написанна программа, реализующая алгоритм и принимающая данные в таком виде. Результатом её работы является список вида

name – f. Для компилятора GCC был реализован проход, инструментирующий функции, так чтобы они исполнялись на нужной частоте. Проход действует на уровне GIMPLE после прохода встраивания функций.

Следует кратко описать схему изменения частоты и напряжения пользовательской программой в операционных системах, основанных на ядре Linux (подробнее см. например [16]). Значением напряжения и частоты нельзя управлять напрямую, однако для систем, поддерживающих ACPI [5] возможно изменение так называемых P-states, каждому из которых соответствует известная частота (набор частот не зависит от экземпляра процессора) и оптимальное для неё напряжение (которое может изменяться в зависимости от экзем пляра). Изменением этих состояний, а соответственно и переключением частот, занимается подсистема ядра cpufreq. Её интерфейсом к пользователю является выбор алгоритма управления, называемого governor. При выборе алгоритма userspace-governor, пользователю предоставляется возможность вручную изменять частоту на любую допустимую с помощью записи соответствующего значения в файл /sys/devices/system/cpu/cpu0/cpufreq/scaling_setspeed.

При этом задержка изменения частоты составляет 10 микросекунд, что можно узнать из файла cpuinfo_transition_latency. Находящегося в той же директории. К сожалению, при этом не используются другие алгоритмы управления энергопотреблением, так как весь контроль отдаётся пользовательскому приложению. Поэтому особенно важно корректно сменить governor на ранее используемый, после завершения программы (в том числе аварийного).

Полученные результаты. Прежде всего опишем методику измерения. Производительность измерялась на наборе тестов SPEC2000 и EEMBC2.0, как наиболее популярных и характерных нативных бенчмарков для мобильных платформ. SPEC2006 не был использован в силу недостаточной производительности мобильной платформы. Каждый тест измерялся отдельно при полностью заряженной батарее. Максимально допустимое падение производительности, передаваемое как параметр компиляции составляло 7 %. Исследовалась система на чипе Medfield. Процессором в данной системе на чипе является Intel Atom с микроархитектурой Bonnell. Результаты приведены в табл. 2 для SPEC2000 . Перед каждым измерением батарея полностью заряжалась. Профиль получался в результате одного измерения на каждой частоте. Для получения результатов измерения производились три раза и результаты усреднялись. Производительность бралась из показаний SPEC.

Таблица 2

Полученные результаты SPEC2000

Название	Падение производительности в %	Уменьшение потребления в %	Прирост эффективности в %
168.wupwise	6,9	1,1	-5,9
171.swim	0,9	9,1	9,0
172.mgrid	5,8	12.7	7,9
173.applu	6,1	7.8	1,8
177.mesa	0,8	2,2	1,4
178.galgel	5,4	8,0	2,8
179.art	5,4	12,0	7,5
183.equake	6,5	17,9	13,9
187.facerec	6,3	12,6	7,2
188.ammp	7,1	9,9	3,1
189.lucas	3,3	10,1	7,5
191.fma3d	1,7	4,2	2,6

Окончание табл. 2

Название	Падение производительности в %	Уменьшение потребления в %	Прирост эффективности в %
200.sixtrack	0,4	0,1	-0,3
301.apsi	5,1	8,7	4,0
FP-geomean	3,2	5,3	2,2
164.gzip	0,2	0,3	0,1
175.vpr	0,3	0,4	0,1
176.gcc	1,4	9,2	8,6
181.mcf	4,9	16,1	13,3
186.crafty	1,2	0,7	-0,5
197.parser	0,6	0,4	-0,2
252.eon	0,6	0,3	-0,3
253.perlbmk	0,8	1,2	0,4
254.gap	1,1	6,1	5,3
255.vortex	0,7	0,2	-0,5
256.bzip2	2,3	5,4	3,3
300.twolf	3,1	6,2	3,3
INT-geomean	1,0	1,4	2.6

Полученные результаты показывают, что данный алгоритм позволяет достигать значительного (более 10 %) увеличения энергоэффективности процессора (например, см 183.equake в табл. 3). Проанализируем полученные результаты.

В качестве примера успешного применения алгоритма рассмотрим бенчмарк 181.mcf из пакета SPEC2000. При заданных условиях (максимальное падение производительности 7 %) применение алгоритма дало следующие результаты:

1. Большинство функций исполнялось на обычной частоте.
2. Функция price_out_impl занимающая 17 % времени исполнения исполнялась на минимальной частоте и дала наибольший вклад в результат.
3. Несколько функций с малым (< 1 %) относительным временем исполнения, также исполнялись на пониженных частотах.

В соответствии с нашими предположениями функция price_out_impl действительно сильно зависит от подсистемы памяти.

В качестве отрицательного примера характерен 168.wupwise из пакета SPEC2000. В нём измерения показывают, что время, потраченное на переключение частоты негативно сказалось на производительности, при этом почти не дав выигрыша по энергопотреблению. То есть полученное решение сильно не оптимально. Заметим, что если проводить профилирование несколько раз, то можно подобрать такие входные данные, что падения энергоэффективности можно будет избежать.

Что касается пакета EEMBC2.0, то несмотря на его известность как бенчмарка для мобильных платформ, он состоит не из реальных приложения а из вызовов тестовой функции в цикле. Именно его малая гранулярность, когда почти всё время тратится внутри одной функции, и привела к незначительным результатам. В данном виде алгоритм не применим к приложениям и тестам, в которых большинство работы совершается в одной функции. В следствие этого во многих случаях изменение частоты либо вообще не приводилось и полученные результаты укладываются в погрешность измерения либо влияние этого изменения было малым.

Как видно из табл 2 целочисленные тесты показали худшие результаты, чем тесты вычислений с плавающей запятой. Это можно объяснить тем, что в целочисленных бенчмарках меньше функций с большим временем исполнения, т.е. переключение частот происходит чаще, и соответственно привносит больший вклад в падение производительности.

Полученные результаты достаточно стабильны. Так, колебания и производительности и энергопотребления составили менее 1 %. То есть требование малости относительной погрешности по сравнению с относительным допустимым падением производительности соблюдается. Вместе с тем погрешность результатов профилирования привела к негативным результатам к примеру, в 168.wupwise. При этом данная стабильность достигается только при условии проведя всех измерений только в одном состоянии батареи. То есть измерение начинается при полной зарядке и перед следующим измерением опять происходит полная зарядка.

Заключение. Были получены достаточно хорошие (более 10 % улучшения энергоэффективности) результаты на некоторых бенчмарках, геометрическое среднее энергоэффективности выросло слабее. Это говорит о том, что эффект менее чем линейного падения производительности или более чем линейного падения потребления в зависимости от частоты наблюдается не всегда. Таким образом целесообразно применять данный алгоритм к тем задачам, где он даёт результаты. Его использование для всех приложений не рекомендуется. К основным недостаткам алгоритма можно отнести невозможность работы в многоядерной системе, где частота каждого ядра задаётся независимо.

Наиболее подходящими задачами являются задачи, имеющие несколько (больше одной) функций с сильной зависимостью от подсистемы памяти и большим (больше 10% общего времени) временем выполнения.

Данный результат (более 10 % улучшения энергоэффективности), превышает полученные в [6] и [7] результаты на 1 и 3 процентных пункта соответственно. Таким образом достигаются результаты, превосходящие ранее полученные с использованием линейного программирования.

В данном виде алгоритм не учитывает возможности разного поведения функции в зависимости от места вызова и соответственно входных данных. Расстановка мест повышения/понижения частоты на графе вызовов является логическим развитием данного алгоритма.

Следующим шагом может стать исследовать поведение алгоритма в случае рассмотрения отдельных базовых блоков или хотя бы часто исполняемых циклов, как участков, рассматриваемых алгоритмом.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. *Brown R.* Report to congress on server and data center energy efficiency // USA Public law 109-431 – 2008. – P. 8-37.
2. *JD Power.* US Wireless Traditional Mobile Phone Satisfaction Study // JD Power studies 2012. – P. 134-156.
3. Intel Corporation and Microsoft Corporation, Advanced Power Management (APM) // BIOS interface specification revision 1.0. – 1992. – P. 7-87.
4. http://www.acpi.info // (дата обращения 09.02.2015).
5. *Albers S., Antoniadis A.* Race to idle: new algorithms for speed scaling with a sleep state // Antoniadis, In Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms. – 2012. – P. 1266-1285.
6. *Hsu C.H., Kremer U.* The design, implementation, and evaluation of a compiler algorithm for CPU energy reduction // U. ACM SIGPLAN Notices. – 2003. – Vol. 38 (5). – P. 38-48.
7. *Zhurikhin D., Belevantsev A., Avetisyan A., Batuzov K., Lee S.* Evaluating power-aware optimizations within GCC compiler // GCC Research Opportunities workshop, GROW. – 2009. – P. 1-7.
8. *Sihara T., Yasuura H.* Voltage scheduling problem for dynamically variable voltage processors // In Low Power Electronics and Design, 1998. Proceedings IEEE. – 1998. – P. 197-202.
9. *Kandemir M., Vijaykrishnan N., Irwin M.J.* Compiler optimizations for low power systems // Power Aware Computing textbook. – 2002. – P. 191-210.

10. *Zhang W., Vijaykrishnan N., Kandemir M., Irwin M.J., Duarte D., Tsai Y.F.* Exploiting VLIW schedule slacks for dynamic and leakage energy reduction. In Microarchitecture // MICRO-34. Proceedings. 34th ACM/IEEE International Symposium. – 2011. – P. 102-113.
11. *Esmaeilzadeh H., Cao T., Yang X., Blackburn S. M., McKinley K.S.* Looking back and looking forward: power, performance, and upheaval. // Communications of the Acm. – 2012. – Vol. 55 (7). – P. 105-114.
12. *Li L., Xue J.* Trace-based leakage energy optimizations at link time // Journal of Systems Architecture. – 2007. – Vol. 53 (1). – P. 1-20.
13. *Goldberg David E.* Genetic algorithms in search, optimization, and machine learning. – M.: Addison-Wesley, 1989.
14. *Melik-Adamyan A.F.* Application of genetic algorithms in problems of optimization of the physical design in microelectronics // Automatic Documentation and Mathematical Linguistics. – 2009. – Vol. 43, Issue 4. – P. 244-250.
15. *Pallipadi V., Starikovskiy A.* The ondemand governor // In Proceedings of the Linux Symposium. – 2006. – Vol. 2. – P. 215-230.
16. *Deb K., Pratap A., Agarwal S., Meyarivan.* A fast and elitist multiobjective genetic algorithm: NSGA-II // Evolutionary Computation, IEEE Transactions on. – 2002. – № 6 (2). – P. 182-197.
17. *Martin T.L., Siewiorek D.P.* Balancing batteries, power, and performance: system issues in cpu speed-setting for mobile computing // Doctoral dissertation, PhD thesis. – 1999. – P. 34-43.
18. *Jones T.M., O'Boyle M.F., Abella J., Gonzőlez A.* Compiler directed issue queue energy reduction // In Transactions on High-Performance Embedded Architectures and Compilers. – M.: Springer Berlin Heidelberg, 2011. – P. 42-62.
19. *Токарь И.П.* Генетический алгоритм оптимизации энергоэффективности в компиляторе для мобильных устройств // Труды 55-й Всероссийской научной конференции проблемы фундаментальных и прикладных естественных и технических наук в современном информационном обществе. – Долгопрудный 2013. – С. 88.

REFERENCES

1. *Brown R.* Report to congress on server and data center energy efficiency, *USA Public law 109-431*, 2008, pp. 8-37.
2. *JD Power.* US Wireless Traditional Mobile Phone Satisfaction Study, JD Power studies 2012, pp. 134-156.
3. Intel Corporation and Microsoft Corporation, Advanced Power Management (APM), *BIOS interface specification revision 1.0*, 1992, pp. 7-87.
4. Available at: http://www.acpi.info (Accessed 09 February 2015).
5. *Albers S., Antoniadis A.* Race to idle: new algorithms for speed scaling with a sleep state, *Antoniadis, In Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms*, 2012, pp. 1266-1285.
6. *Hsu C.H., Kremer U.* The design, implementation, and evaluation of a compiler algorithm for CPU energy reduction, *U. ACM SIGPLAN Notices*, 2003, Vol. 38 (5), pp. 38-48.
7. *Zhurikhin D., Belevantsev A., Avetisyan A., Batuzov K., Lee S.* Evaluating power-aware optimizations within GCC compiler, *GCC Research Opportunities workshop, GROW*, 2009, pp. 1-7.
8. *Sihara T., Yasuura H.* Voltage scheduling problem for dynamically variable voltage processors, *In Low Power Electronics and Design, 1998. Proceedings IEEE*, 1998, pp. 197-202.
9. *Kandemir M., Vijaykrishnan N., Irwin M.J.* Compiler optimizations for low power systems, *Power Aware Computing textbook*, 2002, pp. 191-210.
10. *Zhang W., Vijaykrishnan N., Kandemir M., Irwin M.J., Duarte D., Tsai Y.F.* Exploiting VLIW schedule slacks for dynamic and leakage energy reduction. In Microarchitecture, *MICRO-34. Proceedings. 34th ACM/IEEE International Symposium*, 2011, pp. 102-113.
11. *Esmaeilzadeh H., Cao T., Yang X., Blackburn S. M., McKinley K.S.* Looking back and looking forward: power, performance, and upheaval, *Communications of the Acm.*, 2012, Vol. 55 (7), pp. 105-114.

12. *Li L., Xue J.* Trace-based leakage energy optimizations at link time, *Journal of Systems Architecture*, 2007, Vol. 53 (1), pp. 1-20.
13. *Goldberg David E.* Genetic algorithms in search, optimization, and machine learning. Moscow: Addison-Wesley, 1989.
14. *Melik-Adamyan A.F.* Application of genetic algorithms in problems of optimization of the physical design in microelectronics, *Automatic Documentation and Mathematical Linguistics*, 2009, Vol. 43, Issue 4, pp. 244-250.
15. *Pallipadi V., Starikovskiy A.* The ondemand governor, *In Proceedings of the Linux Symposium*, 2006, Vol. 2, pp. 215-230.
16. *Deb K., Pratap A., Agarwal S., Meyarivan.* A fast and elitist multiobjective genetic algorithm: NSGA-II, *Evolutionary Computation, IEEE Transactions on*, 2002, No. 6 (2), pp. 182-197.
17. *Martin T.L., Siewiorek D.P.* Balancing batteries, power, and performance: system issues in cpu speed-setting for mobile computing, *Doctoral dissertation, PhD thesis*, 1999, pp. 34-43.
18. *Jones T.M., O'Boyle M.F., Abella J., Gonzőlez A.* Compiler directed issue queue energy reduction, *In Transactions on High-Performance Embedded Architectures and Compilers*. Moscow: Springer Berlin Heidelberg, 2011, pp. 42-62.
19. *Tokar' I.P.* Geneticheskiy algoritm optimizatsii energoeffektivnosti v kompilyatore dlya mobil'nykh ustroystv [Genetic algorithm optimization of energy efficiency in the compiler for mobile devices], *Trudy 55-y Vserossiyskoy nauchnoy konferentsii problemy fundamental'nykh i prikladnykh estestvennykh i tekhnicheskikh nauk v sovremennom in-formatsionnom obshchestve* [Proceedings of the 55th all-Russian scientific conference problems of fundamental and applied natural Sciences and engineering in the modern information society]. Dolgoprudnyy 2013, pp. 88.

Статью рекомендовал к опубликованию д.т.н., профессор А.П. Рыжов.

Токарь Илья Петрович – Московский физико-технический институт (Государственный Университет), Россия, г. Долгопрудный; e-mail: tocarip@gmail.com; г. Москва, ул. Крылатская, 17; тел.: +79168528898; аспирант.

Tocar Ilya Petrovich – Moscow Institute of Physics and Technology (State University) Russia, Dolgoprudnuy city; e-mail: tocarip@gmail.com; 17, Krylatskaya street, Moscow, Russia; phone: +79168528898; postgraduate student.

УДК 002.53:004.89

Ю.А. Кравченко

МНОГОУРОВНЕВАЯ АРХИТЕКТУРА СЦЕНАРИЯ УПРАВЛЕНИЯ ЗНАНИЯМИ НА ОСНОВЕ ОНТОЛОГИЧЕСКОГО АНАЛИЗА*

Статья посвящена разработке многоуровневой архитектуры сценария управления знаниями, как принципиальной организации информационных процессов и их взаимоотношений, а также принципов их проектирования и эволюции. Управление информационными потоками рассматривается как совокупность процессов систематического приобретения, синтеза, обмена и использования знаний. Описан подход представления знаний на основе проведения онтологического анализа в условиях неопределенности. Под накоплением знаний понимается процесс переноса знаний из разнородных источников в хранилище данных путем использования различных методов, моделей, алгоритмов и инструментальных средств. Данный процесс требует максимальной автоматизации, т.к. при иных стратегиях приобретения знаний всегда существует проблема передачи информации при совместной работе эксперта в предметной области и инженера знаний. Несмотря на выраженную специфику предметных областей, онтологию необходимо строить как цепочку взаимосвязанных процессов, что позволит получить интегрированный характер интеллектуальной системы управления знаниями. На этапе идентифика-

* Исследование выполнено за счет гранта Российского научного фонда (проект №14-11-00242) в Южном федеральном университете.