Проектирование и анализ программного обеспечения с низким энергопотреблением с помощью программных метрик энергоэффективности

01, январь 2013

DOI: 10.7463/0113.0511596

Юрченко А. В.

УДК 004.051

Россия, Тула, ЗАО «ЕС-лизинг»

yurchenkoalex@inbox.ru

Введение

Низкое потребление энергии высокопроизводительными компьютерными системами стало главной исследовательской задачей как на прикладном уровне в промышленности, так и в лабораторных условиях. Потребление энергии является критическим для переносных устройств, поскольку определяет время работы их аккумуляторной батареи, их вес, максимально возможную степень интеграции, а так же касается вопросов их охлаждения и надежности [1]. Зачастую у проектировщиков имеется ограниченный ресурс электропитания, и решение такой задачи является серьезным вызовом, так как необходимо найти компромисс между производительностью и ограниченным зарядом. Увеличение производительности, например, повышение тактовой частоты, обычно приводит к повышенному потреблению электроэнергии.

Разработка устройств, имеющих низкое энергопотребление, зависит как от аппаратного обеспечения, так и от программного обеспечения, выполняемого на встроенных процессорных ядрах [2]. Поэтому для понижения потребления системой электроэнергии разрабатывают методики как в области аппаратного, так и программного обеспечения. Общей целью большинства исследований является сокращение динамического рассеивания мощности, которое происходит из-за зарядки и разрядки емкостей схем [1]. Методики, касающиеся аппаратного обеспечения, минимизируют потребление энергии путем оптимизации целевых параметров, таких, как напряжение питания, количество логических вентилей, размер транзисторов, рабочая частота. С другой стороны, методики из области разработки программного обеспечения (ПО) обычно затрагивают более высокие уровни иерархии системного дизайна, и в итоге, результаты сохранения энергии после оптимизации

ПО бывают более существенными [3]. Программные методологии в основном нацелены на выполнении поставленной задачи с использованием меньшего количества инструкций (команд процессора), что приводит к понижению коммутаций в схемах, которые являются главным источником потребления энергии в CMOS схемах [1].

Учитывая вышесказанное, возникает необходимость наличия метрик для оценки энергоэффективности проектируемого и анализируемого ПО [4]. С помощью таких метрик можно выявить наиболее эффективную реализацию среди нескольких программ, которая потребляет меньше энергии во время функционирования. Это позволит создать индикаторы, которые могли бы помочь улучшить качество ПО, и понять необходимое соотношение аппаратной и программной частей при проектировании системы в целом. Энергетические метрики должны быть простыми и вычислимыми, интуитивно понятными, непротиворечивыми, и настолько, насколько это возможно – независимыми от языка программирования.

Большое количество метрик для внутренних параметров программных продуктов, такие, как размер программы, сложность, функциональность, структурированность, модульность, были в свое время разработаны и активно применялись и применяются к программным продуктам [5]. Потребление энергии программой зависит от ее динамического поведения, и от аппаратной платформы, на которой программа выполняется, поэтому необходимо расширить существующую программную модель граф-схемы для однозначного определения параметров времени исполнения ПО. Поскольку основные источники потребления энергии в компьютерных системах это процессорные операции и операции доступа к оперативной памяти, нужно разработать меры для динамических характеристик этих операций.

Исследования в направлении создания энергоэффективного ПО были начаты с разработки энергетических моделей уровня набора инструкций, которые оценивали потребление электроэнергии процессором [6, 7]. Эти модели могут быть применены к исследуемой программе как после ее выполнения, чтобы получить трассировку выполненных инструкций, или применив методы статического анализа на уровне исходного кода для оценки количества выполняемых инструкций [8]. Основываясь на измерениях, которые показывали, что межинструкционные (переход от одной инструкции к другой) затраты энергии отличаются для разных пар инструкций [6], были предложены несколько методик для оптимизации расхода энергии, которые использовали алгоритмы диспетчеризации (определение последовательности выполнения инструкций, или программ) для минимизации расходов энергии [7]. Для оценки комплексного воздействия программной и аппаратной оптимизации на энергоэффективность были использованы модели уровня инструкций с высокой точностью циклов (работа процессора моделировалась вплоть до каждого его цикла работы) [4, 9]. Другая литература, касающаяся проектирования

энергоэффективного ПО, описывает методологии связанные с потреблением энергии оперативной памятью. В них предлагается преобразование программного кода таким образом, чтобы организовать работу с меньшими участками памяти и с меньшими энергозатратами за каждую операцию доступа к памяти [10].

Тем не менее, несмотря на все предыдущие работы, потребление энергии конкретной программой всегда оценивалось, используя специфическую целевую архитектуру. При этом не были сделаны попытки определить энергетические метрики, которые могли бы использоваться для сравнения дизайна ПО, независимо от нижележащего аппаратного обеспечения. Более того, вопросы проектирования энергоэффективного ПО не рассматривались в привязке к потреблению энергии в результате совместной работы процессора и оперативной памяти. Так же не было исследований влияния различных вариантов программного кода на потребление энергии.

Следует отметить, что существующие методы используют трассировку инструкций исследуемых программ, и промежуток времени, требуемый для данного процесса для больших приложений, становится непомерно длинным. Перенос расчетов для оптимального соотношения потребляемой энергии и производительности на более высокие уровни абстракции становится неизбежным, особенно когда большие объемы работ по проектированию ПО должны быть выполнены за короткий промежуток времени [2, 4].

Цель данной работы заключается в выдвижении на первый план тезиса о влиянии ПО на потреблении энергии нижележащим аппаратным обеспечением, а так же в рассмотрении энергопотребления, как одного из параметров проектирования в программной инженерии. Учитывая цель, в работе исследованы меры энергоэффективности, которые можно использовать для оценки потребления энергии программой, а так же будут определены программные энергетические метрики. Эффективность предложенных метрик будет продемонстрирована на различных реализациях алгоритмов, которые будут экспериментально оценены по производительности и потреблению энергии, используя симулятор процессора с применением предложенных параметров.

1 Источники потребления электроэнергии

Когда программа выполняется на процессоре, то существует два основных источника потребления энергии, расход энергии которых может меняться в зависимости от используемых архитектуры и ПО.

1.1 Потребление электроэнергии процессором

Потребление энергии процессором происходит из-за функционирования его электрических схем во время выполнения программных инструкций. Декодирование и

выполнение инструкций производится за счет коммутирующих действий в узлах цифровой схемы процессора, которые представляют собой зарядку и разрядку емкостей этих узлов, что приводит к динамическому рассеиванию энергии [1]. Для измерения данной составляющей расхода энергии разработаны модели потребления энергии уровня инструкций. Эти модели основаны на предположении [6], что можно, измеряя электрический ток процессора во время выполнения им определенного набора инструкций, получить большую часть информации, требуемую для оценки потребления энергии программой, выполняемой на этом процессоре. Это предположение дополнено и уточнено. Утверждается, что общее потребление энергии не может быть подсчитано суммированием расходов энергии на каждую отдельную инструкцию процессора [7]. Доказано, что изменения состояний схемы между последовательными инструкциями должны приниматься в расчет для создания точных моделей потребления энергии уровня инструкций.

Существует две основных составляющих модели потребления энергии уровня инструкций.

Базовые расходы энергии. Представляют собой расходы, которые связаны с базовой обработкой, необходимой для выполнения инструкции. Расход оценивается как среднее значение тока, потраченное на выполнение одного цикла процесса обработки нескольких экземпляров одной и той же инструкции.

Накладные расходы энергии. Получаются из-за коммутирующих действий в схеме процессора и из-за косвенных дополнительных расходов энергии, вытекающих из выполнения смежных инструкций (пересчет индексов циклов и вычисление адресов памяти). Для измерения среднего значения тока в данном случае используются последовательности специализированных инструкций.

Таким образом, общее потребление электроэнергии программой, выполняемой на процессоре, может быть получено как сумма всех базовых и накладных расходов энергии за время выполнения программы.

1.2 Потребление электроэнергии памятью

Расходы энергии на память связаны с операциями доступа к инструкциям, или к данным. Расход энергии за операцию доступа зависит от объема памяти. Следовательно, для больших модулей памяти, находящихся вне чипа (процессора), энергопотребление значительно выше, чем для памяти небольшого размера, встроенной в чип. Эта составляющая потребления энергии так же зависит от самого выполняемого ПО. Энергопотребление памяти, используемой для хранения инструкций, зависит от размера программного кода, который определяет объем используемой памяти, и количества

выполняемых инструкций, которые определяют количество операций выборки из памяти. Энергопотребление памяти, используемой для хранения данных, зависит от объема обрабатываемых приложением данных, и от того, насколько интенсивно приложение работает с данными, то есть насколько часто оно получает доступ к данным. Согласно типичной модели потребления энергии энергия расходуется на операции доступа к памяти, и ее расход прямо пропорционален количеству операций доступа, размеру и количеству портов ввода-вывода памяти, напряжению питания, и технологии производства.

Согласно вышесказанному, количество выполняемых инструкций имеет двукратное воздействие на энергопотребление – посредством выполнения инструкций с помощью процессора, и извлечения инструкций из памяти. С другой стороны, вклад операций доступа к памяти в энергопотребление рассматривается однократно, как сам по себе. Но затраты электроэнергии на работу с данными в памяти гораздо выше, чем затраты на операции доступа к инструкциям, или энергия потраченная на выполнение инструкций процессором. Энергетические параметры этих дополнительных составляющих далее будут учтены в определении энергетических метрик ПО.

2 Оценка потребления электроэнергии программным обеспечением

Создание энергетических мер основывается на том, что поток команд каждой программы можно представить с помощью ориентированного графа, который называется потоковым графом программы [5]. Базовые S-графы, которые используются для определения семейства допустимых S-структурированных графов, к которым принадлежит потоковый граф любой произвольной программы, показаны на рисунке 1. Так же вместе с графами показаны соответствующие им коды программ.

Рисунок 1 - Базовые S-графы

2.1 Иерархические энергетические меры

Чтобы оценить конструкторские характеристики ПО в терминах их энергопотребления, определим две следующих иерархических меры, присвоив значения каждому примитиву базовых S-графов, и значения функциям вложения и упорядочивания:

Мера количества выполненных инструкций (Executed Instruction Count – EIC). Она соответствует количеству выполненных инструкций на ядре процессора [11]. Это число связано с потреблением энергии процессором, и получается с помощью моделей потребления энергии уровня инструкций. Аналогичным образом это число вычисляется и для энергопотребления памяти команд процессора, поскольку каждая выполненная инструкция связана с операцией выборки инструкции из памяти команд.

Мера количества операций доступа к памяти (Memory Access Count – MAC). Она равна количеству операций доступа к памяти данных, когда происходят операции извлечения программных операндов.

Следует отметить, что в данной работе использование граф-схем расширено, чтобы можно было оценивать нестатические атрибуты. Неэффективность граф-схем в представлении динамических параметров заключается, прежде всего, в наличии структур повторения и выбора, в которых поведение программы времени выполнения нельзя спрогнозировать, за исключением некоторых случаев. Хотя граф-схемы принадлежат области статического анализа [8], в данной работе для достижения ее целей использован упрощенный метод. Для структур повторения каждый цикл снабжается комментариями, в которых указывается точное число итераций цикла (в том случае, если это явно описано в программе), либо оценка (может быть приблизительная), полученная разработчиком. Хотя неточные подсчеты итераций цикла могут привести к большим абсолютным ошибкам, особенно когда глубина вложений нескольких циклов друг в друга довольно большая, с целью сравнения нескольких реализаций одного и того же алгоритма и определения наиболее эффективного, точность таких подсчетов оказывается на достаточно высоком уровне, что и будет показано дальше. Это происходит потому, что оценки применяются для аналогичных структур повторения, что отфильтровывает эффект от неточных оценок.

Для случаев со структурами выбора в расчет берется наиболее энергоемкая ветвь выбора, и энергопотребление программы считается по самой верхней границе.

Отдельно от аннотаций для ребер в граф-схемах, в которых встречаются структуры повторения, каждый узел в граф-схеме снабжается аннотацией, содержащей количество операций доступа к памяти, выполняемых в соответствующем участке кода программы. Для обозначения того, что количество операций доступа к памяти i выполняется в заданном узле

n , вокруг узла рисуются i окружностей (рисунок 2). Более того, чтобы была возможность подсчитывать количество выполняемых инструкций, каждый узел, который соответствует более чем одному арифметическому оператору программы, помечается на граф-схеме числом арифметических операторов (r). В данном случае становится возможным визуально обнаруживать вычислительно интенсивные участки программы, или участки с частым доступом к памяти (рисунок 2).

Рисунок 2 - Граф-схемы с обозначениями количества итераций цикла и количества операций доступа к памяти

Следует отметить, что предложенный подход допускает предположение о том, что заданная программная структура всегда преобразуется в один и тот же набор команд, независимо от используемого компилятора. Основной целью является сравнение нескольких альтернатив для одного и того же программного приложения в энергетических показателях на ранних этапах проектирования, когда относительная точность гораздо важнее, чем абсолютная точность [2, 3]. Предполагается, что все варианты программы будут оптимизированы с одной и той же степенью после преобразования компилятором. Например, ARM-компилятор, который используется в данной работе, выполняет некоторую последовательность оптимизаций таких, как исключение общих подвыражений, перемещение инвариантов цикла, и свертывание констант. Если в программном коде высокоуровневый оператор компилируется в набор машинных команд с использованием одной, или нескольких операций оптимизации, то предполагается, что такой же оператор в

другой вариации этого же программного алгоритма будет скомпилирован с использованием тех же оптимизаций. Поэтому ошибки от того, что во время процесса подсчета количества выполненных инструкций на граф-схеме не принимается во внимание реализация конкретного используемого компилятора, будут незначительными. Влияние оптимизаций компиляторов на энергопотребление ПО рассмотрено в [12]. Другой момент, заслуживающий внимания, связан с тем, что встроенные процессоры часто используют кэширование данных, что может значительно оказать влияние на соотношении энергопотребление/производительность [12]. Для данной ситуации необходимо использовать симулятор кэш-памяти. И энергопотребление уже должно подсчитываться с учетом доступа к основной оперативной памяти и к кэш-памяти [13].

Учитывая вышесказанные аппроксимации, укажем правила [5] для двух энергетических мер.

Мера количества выполненных инструкций I .

- *Функции примитивов.*

$$I(P_1) = r + m, \quad (1)$$

где r – это количество базовых математических операторов (за исключением деления), m – это количество операций доступа к памяти за время выполнения оператора P_1 соответственно, и для каждого примитива $F \neq P_1$.

$$I(F) = \begin{cases} i \cdot [3 + \max((r + m)_X, (r + m)_Y)], & F = D_1 \\ i \cdot [3 + r + m], & F \neq D_1 \end{cases} \quad (2)$$

где i представляет собой количество итераций цикла (по умолчанию для D_0 и D_1 $i = 1$). В случае, если $F = D_1$, то X, Y соответствуют двум узлам D_1 . В приведенном выражении для простоты все граф-схемы примитивов соответствуют выполнению трех инструкций. Две инструкции соответствуют заданию условия выбора (загрузка соответствующих двух значений из регистров и выполнение их сравнения), и одна инструкция соответствует ветвлению и началу цикла. Подсчет операций доступа к памяти m выполняется здесь, потому что предполагается, что каждая операция доступа к памяти соответствует одной инструкции загрузки/сохранения машинного кода.

- *Функции упорядочивания.*

$$I(F_1; F_2; \dots; F_n) = \sum_{i=1}^n I(F_i). \quad (3)$$

- *Функции вложения.*

$$I(F(F_1; F_2; \dots; F_n)) = \begin{cases} i \cdot \left[(3 + \max(\sum_{i=1}^n I(F_i)_X, \sum_{i=1}^n I(F_i)_Y)) \right], & F = D_1 \\ i \cdot \left[3 + \sum_{i=1}^n I(F_i) \right], & F \neq D_1 \end{cases}, \quad (4)$$

для каждого примитива $F \neq P_1$, где i – это количество итераций цикла, соответствующее F , а $\max$ относится к максимальному значению меры количества инструкций обеих ветвей (true и false) F .

Мера количества операций доступа к памяти M .

– *Функции примитивов.*

$$M(P_1) = m, \quad (5)$$

где m – это количество операций доступа к памяти за время выполнения оператора P_1 , и для каждого примитива $F \neq P_1$.

$$M(F) = \begin{cases} i \cdot \max(m_X, m_Y), & F = D_1 \\ i \cdot m, & F \neq D_1 \end{cases}, \quad (6)$$

где i представляет собой количество итераций цикла (по умолчанию для D_0 и D_1 $i = 1$).

– *Функции упорядочивания.*

$$M(F_1; F_2; \dots; F_n) = \sum_{i=1}^n M(F_i). \quad (7)$$

– *Функции вложения.*

$$M(F(F_1; F_2; \dots; F_n)) = \begin{cases} i \cdot \max(\sum_{i=1}^n M(F_i)_X, \sum_{i=1}^n M(F_i)_Y), & F = D_1 \\ i \cdot \sum_{i=1}^n M(F_i), & F \neq D_1 \end{cases}, \quad (8)$$

для каждого примитива $F \neq P_1$, где i – это количество итераций цикла, соответствующее F .

2.2 Программные энергетические показатели (меры)

Рассматривая средние энергозатраты инструкции [14], которые состоят из доступа к памяти данных (предполагается использование RAM (ОЗУ) памяти данных), и доступа к памяти команд (инструкций) (предполагается использование ROM (ПЗУ) памяти команд), результирующая элементарная энергетическая программная метрика (Software Energy Metric – SEM), основанная на иерархических мерах, определенных ранее, будет выглядеть следующим образом:

$$SEM = EIC + 2 \cdot MAC. \quad (9)$$

В данной формуле (9) *EIC* рассматривается как энергопотребление, которое получается в результате выполнения инструкций (команд) в процессоре и операций выборки инструкций из памяти команд. *MAC* рассматривается как трата энергии на операции доступа к памяти данных. *SEM* близко интуитивному пониманию потребления энергии. Если программа *A* потребляет больше энергии, чем программа *B*, то значение *SEM(A)* будет не только больше значения *SEM(B)*, но и отношение реального потребления энергии программой *A* к реальному потреблению энергии программой *B* будет близко к отношению *SEM(A)* к *SEM(B)*. Это будет показано в примерах в следующем разделе. Следовательно, предлагаемая метрика является непротиворечивой при использовании ее в качестве единицы измерения. Так же следует отметить, что *SEM* является независимой относительно используемого языка программирования, поскольку ее измерение основано на иерархических структурах, которые создаются в соответствии с эквивалентами машинных команд и общепринятыми программными структурами, и предполагаем, что они будут использованы на универсальном процессоре.

В уравнении (9) можно заметить, что хотя выполнение инструкций оказывает двойное воздействие на энергопотребление системой, как это было разъяснено ранее (выполнение на процессоре, плюс выборка из памяти), количество операций доступа к памяти умножается на два. Это происходит потому, что энергозатраты на доступ к RAM (оперативное запоминающее устройство) памяти (в которой данные хранятся и откуда они получаются) значительно выше, чем энергозатраты на доступ к ROM (постоянное запоминающее устройство) памяти, или энергозатраты создаваемые процессором при обработке им инструкций. Лишь встроенный тип памяти DRAM (динамическое оперативное запоминающее устройство), которая интегрируется на том же чипе, на котором находится цифровая логика, может изменить данное двукратное соотношение и тогда энергоэффективность вычислений будет сопоставима с энергоэффективностью доступа к памяти данных. Но данная реализация стоит дороже, поэтому, вероятней всего, она будет использоваться на ограниченном ряде переносных устройств.

3 Результаты

Чтобы оценить эффективность предложенных мер и метрик в данной работе протестировано несколько приложений, реализующих вычисления из матричной алгебры. Рассмотрим три программы на языке программирования «C» (таблица 1), которые выполняют перемножение матриц. Алгоритмы выполняют перемножение с помощью

вычисления скалярного произведения (*scalar.c*), с помощью обобщенной SAXPY операции (*saxpy.c*), и с помощью векторного произведения (*vector.c*) [15]. Перемножение матриц математически формулируют следующим образом.

Пусть даны две прямоугольные матрицы A и B размерности $m \times n$ и $n \times q$ соответственно. Тогда матрица C размерностью $m \times q$ называется их произведением:

$$c_{ij} = \sum_{r=1}^n a_{ir} b_{rj} \quad (i = 1, 2, \dots, m; j = 1, 2, \dots, q). \quad (10)$$

Таблица 1- Алгоритмы перемножения матриц на языке C

<i>scalar.c</i>	<i>saxpy.c</i>	<i>vector.c</i>
<pre> i=0; while (i<m) { j=0; while (j<n) { c=0; k=0; while (k<p) { c=c+(*(&A[i][0]+k)) * (*(&B[0][j]+k*n)); k++; } C[i][j]=c; j++; } i++; } </pre>	<pre> k=0; while (k<n) { j=0; while (j<m) { i=0; while (i<p) { *(&C[0][k]+j*n)= *(&A[0][0]+i+j*p) * (*(&B[0][k]+i*n)) + *(&C[0][k]+j*n); i++; } j++; } k++; } </pre>	<pre> k=0; while (k<p) { i=0; while (i<m) { j=0; while (j<n) { *(&C[0][0]+i*n+j)= *(&A[0][k]+i*p) * (*(&B[k][0]+j)) + *(&C[0][0]+i*n+j); j++; } i++; } k++; } </pre>

Для того чтобы программный код можно было легко трансформировать в граф-схему, оператор *for( )* для цикла с заданным количеством повторений заменен оператором *while( )*. Следует отметить, что для вариантов алгоритмов, реализующих операцию SAXPY и векторное произведение матрица C должна быть заполнена нулями перед началом вычислений. Граф-схемы для приведенных выше программ с примечаниями о количестве итераций цикла (на ребрах), о количестве операций доступа к памяти (окружности вокруг узла), и о количестве арифметических операторов (на узлах), приведены выше на рисунке 2.

В качестве примера расчета, дерево декомпозиции для граф-схемы программы *scalar.c*, иллюстрирующее иерархию примитивов, показано на рисунке 3 [5].

Рисунок 3 - Дерево декомпозиции для граф-схемы программы *scalar.c*

Применяя правила для иерархических показателей, определенные ранее в данной работе, к граф-схеме примера программы *scalar.c*, получим следующую меру количества выполненных инструкций *EIC*:

$$\begin{aligned}
 I(F) &= I(P_1; D_2(P_1; D_2(P_2; D_2(P_2); P_2); P_1)) = \\
 &= I(P_1) + I(D_2(P_1; D_2(P_2; D_2(P_2); P_2); P_1)) = \\
 &= I(P_1) + m \cdot [3 + I(P_1; D_2(P_2; D_2(P_2); P_2); P_1)] = \\
 &= I(P_1) + m \cdot [3 + I(P_1) + I(D_2(P_2; D_2(P_2); P_2)) + I(P_1)] = \\
 &= I(P_1) + m \cdot [3 + I(P_1) + n \cdot (3 + I(P_2; D_2(P_2); P_2)) + I(P_1)] = \\
 &= I(P_1) + m \cdot [3 + I(P_1) + n \cdot (3 + I(P_2) + I(D_2(P_2))) + I(P_2)) + I(P_1)] = \\
 &= I(P_1) + m \cdot [3 + I(P_1) + n \cdot (3 + I(P_2) + p \cdot (3 + I(P_2)) + I(P_2)) + I(P_1)] = \\
 &= 1 + m \cdot [3 + 1 + n \cdot (3 + 2 + p \cdot (3 + 9) + 3) + 1] = \\
 &= 1 + m \cdot (5 + n \cdot (8 + p \cdot 12)).
 \end{aligned} \tag{11}$$

Мера количества операций доступа к памяти *MAC* может быть вычислена как:

$$\begin{aligned}
 M(F) &= M(P_1; D_2(P_1; D_2(P_2; D_2(P_2); P_2); P_1)) = \\
 &= M(P_1) + M(D_2(P_1; D_2(P_2; D_2(P_2); P_2); P_1)) = \\
 &= M(P_1) + m \cdot [M(P_1; D_2(P_2; D_2(P_2); P_2); P_1)] = \\
 &= M(P_1) + m \cdot [M(P_1) + M(D_2(P_2; D_2(P_2); P_2)) + M(P_1)] = \\
 &= M(P_1) + m \cdot [M(P_1) + n \cdot M(P_2; D_2(P_2); P_2) + M(P_1)] = \\
 &= M(P_1) + m \cdot [M(P_1) + n \cdot (M(P_2) + M(D_2(P_2))) + M(P_2)) + M(P_1)] = \\
 &= M(P_1) + m \cdot [M(P_1) + n \cdot (M(P_2) + p \cdot M(P_2) + M(P_2)) + M(P_1)] = \\
 &= 0 + m \cdot [0 + n \cdot (0 + p \cdot 2 + 1) + 0] = \\
 &= 2 \cdot m \cdot n \cdot p + m \cdot n.
 \end{aligned} \tag{12}$$

Для того чтобы оценить реальное количество выполненных инструкций и операций доступа к памяти, в качестве целевой архитектуры было выбрано ядро встроенного процессора ARM7, который широко используется для встроенных приложений из-за его заявленного отношения выполненных команд в секунду к энергопотреблению. Более того, у него есть некоторые преимущества при проектировании открытых архитектур [11].

Экспериментальный процесс, организованный для оценки дизайна ПО в терминах энергопотребления, показан на рисунке 4. Каждый программный код был скомпилирован, используя C компилятор CMC-ARM [16]. Далее код был выполнен, используя ARM отладчик, с помощью которого было получено количество выполненных машинных команд и общее количество циклов. С помощью ARM отладчика был получен файл трассировки, в котором были зафиксированы выполненные инструкции и операции доступа к памяти.

Рисунок 4 - Схема эксперимента для оценки энергопотребления

Погрешность между расчетами потребления энергии для ARM эмулятора и для реального ARM процессора считается вполне допустимой. Отчет в работе [14] показывает, что средняя ошибка между результатами симуляции и реальными измерениями составляет 7,1 %.

Количество выполненных инструкций использовалось в качестве входных параметров для модели энергопотребления памяти, чтобы подсчитать энергопотребление памяти инструкций. Аналогично количество операций доступа к памяти данных использовалось для подсчета энергопотребления памяти данных.

Для того, чтобы показать эффективность предложенных мер для описания поведения базового аппаратного обеспечения, выполнение каждой программы было эмулировано на эмуляторе ARM. Результаты для $m = 2$, $p = 3$ и $n = 4$ приведены в таблице 2. Как можно заметить, количество операций доступа к памяти подсчитано без ошибок, в то время как количество выполненных машинных команд подсчитано со средней ошибкой, которая возникла из-за некоторых допущений, сделанных относительно компиляции программного «C» кода в машинные команды.

Таблица 2 – Сравнение результатов симуляции и расчетов для рассматриваемых мер

	scalar.c		saxpy.c		vector.c		Средняя ошибка
	Симуляция	Расчет	Симуляция	Расчет	Симуляция	Расчет	
Количество команд	382	363	495	517	474	502	5,12%
Количество операций доступа к памяти	56	56	96	96	96	96	0%
	48 загрузок		72 загрузок		72 загрузок		
	8 сохранений		24 сохранений		24 сохранений		

В таблице 3 представлено реальное потребление энергии всеми системными компонентами (процессор, память данных и память команд), полученное на основе представленных моделей энергопотребления и с использованием симулятора ARM процессора.

Таблица 3 - Энергопотребление алгоритмов перемножения матриц (мДж)

	<i>scalar.c</i>	<i>saxpy.c</i>	<i>vector.c</i>
Энергопотребление процессора	0,001997	0,002687	0,002592
Энергопотребление памяти данных	0,000252	0,000432	0,000432
Энергопотребление памяти команд	0,000342	0,000443	0,000424
Общее энергопотребление	0,002591	0,003562	0,003448

Чтобы продемонстрировать эффективность предложенной программной энергетической метрики *SEM*, в таблице 4 приведены нормализованные значения реального энергопотребления и нормализованные теоретические значения *SEM* относительно

программы *scalar.c*. Несмотря на неизбежные приближенные вычисления, сделанные для моделирования динамических характеристик программ, величины ошибок относительно небольшие. Это подтверждает необходимость разработки программных энергетических метрик для того, чтобы иметь возможность сравнивать различные альтернативы реализации каких-либо алгоритмов, или систем в терминах энергоэффективности [3].

Таблица 4 - Нормализованное энергопотребление и программная энергетическая метрика для алгоритмов перемножения матриц

	<i>scalar.c</i>	<i>saxpy.c</i>	<i>vector.c</i>
Реальное энергопотребление	1	1,375	1,332
<i>SEM</i>	1	1,493	1,461
Ошибка		8,58%	9,68%

Второй пример также взят из матричной алгебры, и связан с вычислением Гильбертовой матрицы. Гильбертова матрица представляет собой квадратную матрицу, элементами которой являются дроби с числителем единицей, и знаменателем, вычисляемым следующим образом:

$$H_{ij} = \frac{1}{i+j-1}. \quad (13)$$

Для ее вычисления можно использовать либо двойной вложенный цикл, либо, принимая во внимание ее симметричную структуру, вычисление ее элементов может быть выполнено в два раза быстрее. Оба варианта программ показаны в таблице 5.

Таблица 5 - Альтернативные варианты заполнения Гильбертовой матрицы

<i>hilbert1.c</i>	<i>hilbert2.c</i>
<pre> i=1; while (i<=n) { j=1; while (j<=n) { A[i][j]=1/(i+j-1); j++; } i++; } </pre>	<pre> i=1; while (i<=n) { j=i; while (j<=n) { A[i][j]=1/(i+j-1); A[j][i]=A[i][j]; j++; } i++; } </pre>

Для граф-схем двух программ из таблицы 5 аналогичным образом извлечем иерархические меры для количества выполненных машинных команд и операций доступа к памяти. Основываясь на этих мерах, вычислим значения программных энергетических

метрик. Их нормализованные значения относительно первой программы в сравнении с нормализованными значениями реального потребления энергии приведены в таблице 6.

Таблица 6 - Нормализованное энергопотребление и программная энергетическая метрика для алгоритмов вычисления значений элементов Гильбертовой матрицы

	<i>hilbert1.c</i>	<i>hilbert2.c</i>
Реальное энергопотребление	1	0,798
<i>SEM</i>	1	0,742
Ошибка		7,02%

Произведя полные однократные вычисления, в дальнейшем энергетические метрики, основанные на *SEM* метрике, можно использовать для исследования поведения программы в ответ на другие значения входных параметров без ее симуляции. Для примера с Гильбертовой матрицей энергопотребление обеих программ можно рассчитать для разного количества размера матрицы. На диаграмме на рисунке 5 показана зависимость *SEM* для обоих алгоритмов от размерности матрицы n .

Рисунок 5 - Зависимость программной энергетической метрики от размерностей матриц Гильберта для различных алгоритмов

Заключение

Низкое потребление электроэнергии встроенным ПО является одной из основных задач при проектировании переносных устройств. Хотя аппаратные решения при проектировании могут влиять на энергопотребление цифровых систем, коммутирующая деятельность в схемах процессоров, которая представляет собой основной источник энергопотребления, в основном управляется с помощью ПО, выполняемом на встроенных ядрах процессоров. В данной работе описана необходимость определения подходящих энергетических метрик для оценки проектируемого ПО и его влияния на энергопотребление нижележащего аппаратного обеспечения. Иерархические энергетические меры, получаемые из граф-схемы программы, использованы для определения количества выполненных

инструкций (команд) и количества операций доступа к памяти данных. Основываясь на полученных мерах, определены программные метрики энергоэффективности, используемые для оценки энергопотребления процессора и операций доступа к памяти данных и памяти команд. Полученные метрики апробированы на приложениях, реализующих алгоритмы из матричной алгебры. Показана их устойчивость и точность и указана важность энергетических аспектов на этапе проектирования ПО. Результатом дальнейших работ в данном направлении будут попытки определить более подробные, сложные метрики, которые помогли бы разработчикам создавать более качественное ПО в терминах энергопотребления.

Список литературы

1. Chandrakasan A., Brodersen R. Low Power Digital CMOS Design. Boston: Kluwer Academic Publishers, 1995.
2. Fornaciari W., Gubian P., Sciuto D., Silvano C. Power estimation of embedded systems: A hardware/software codesign approach // IEEE Trans. on VLSI Systems. 1998. Vol. 6, no. 2. P. 266-275.
3. Landman P. High-level power estimation // Proc. International Symposium on Low Power Electronics and Design. Monterey, CA, 1996. P. 29-35.
4. Brooks D., Tiwari V., Martonosi M. Wattch: A framework for architectural-level power analysis and optimizations // Proc. International Symposium on Computer Architecture. Vancouver, BC, Canada, 2000. P. 83-94.
5. Fenton N.E., Pfleeger S.L. Software Metrics: A Rigorous and Practical Approach. London: International Thomson Computer Press, 1996.
6. Tiwari V., Malik S., Wolfe A. Power analysis of embedded software: A first step towards software power minimization // IEEE Transactions on VLSI Systems. 1994. Vol. 2, no. 4. P. 437-445.
7. Tiwari V., Malik S., Wolfe A., Lee T.C. Instruction level power analysis and optimization of software // Journal of VLSI Signal Processing. 1996. Vol. 13, no. 2. P. 1-18.
8. Malik S., Martonosi M., Li Y.-T.S. Static Timing Analysis of Embedded Software // Proc. Design Automation Conference. Anaheim, CA, 1997. P. 147-152.
9. Ye W., Vijaykrishnan N., Kandemir M., Irwin M. J. The design and use of SimplePower: A cycle-accurate energy estimation tool // Proc. Design Automation Conference. LosAngeles, CA, 2000. P. 340-345.

10. Catthoor F., Wuytack S., De Greef E., Balasa F., Nachtergaele L., Vandecappelle A. Custom Memory Management Methodology: Exploration of Memory Organization for Embedded Multimedia System Design. Boston: Kluwer Academic Publishers, 1998.
11. Furber S. ARM System-on-Chip Architecture. Harlow, Addison-Wesley, 2000.
12. Kandemir M., Vijaykrishnan N., Irwin M.J., Ye, W. Influence of compiler optimizations on system power // Proc. Design Automation Conference. LosAngeles, CA, 2000. P. 304-307.
13. Dasigenis M., Kroupis N., Argyriou A., Tatas K., Soudris D., Zervas N., Thanailakis A. A memory management approach for efficient implementation of multimedia kernel on programmable architectures // Proc. IEEE Computer Society Annual Workshop on VLSI. Orlando, Florida, 2001. P. 171-176.
14. Sinevriotis G., Stouraitis Th. SOFLOPO: low power software development for embedded applications (SOFLOPO). ESPRIT ESD-LPD Project 25403, Public Final Report, 2001.
15. Голуб Дж., Ван Лоун Ч. Матричные вычисления: пер. с англ. М.: Мир, 1999. 548 с. [Golub G.H., Van Loan C.F. Matrix computations. 3rd ed. The Johns Hopkins University Press, 1996. 728 p.].
16. Пакет инструментальных средств разработки и отладки микроконтроллеров ARM. Режим доступа: <http://www.phyton.ru> (дата обращения 31.07.2012).

Design and analysis of low power consumption software using software power efficiency metrics

01, January 2013

DOI: 10.7463/0113.0511596

Yurchenko A.V.

Russia, Tula, «EC-leasing»

yurchenkoalex@inbox.ru

There is permanent growth of requirements for power efficient operation of the majority of portable computing devices. In this paper the author considers power efficient software design and proposes metrics for energy consumption analysis in software systems. Sources of power consumption in digital circuits are considered. It was proved that power consumption depends on executing software. The author gives a description of energy measures which form a basis for definition of software energy metrics. The defined metrics can be used for calculating the level of energy consumption of any software system.

Publications with keywords: [energy performance](#), [software analysis](#), [software metrics](#), [embedded software](#), [energy consumption](#)

Publications with words: [energy performance](#), [software analysis](#), [software metrics](#), [embedded software](#), [energy consumption](#)

References

1. Chandrakasan A., Brodersen R. *Low Power Digital CMOS Design*. Boston, Kluwer Academic Publishers, 1995.
2. Fornaciari W., Gubian P., Sciuto D., Silvano C. Power estimation of embedded systems: A hardware/software codesign approach. *IEEE Trans. on VLSI Systems*, 1998, vol. 6, no. 2, pp. 266-275.
3. Landman P. High-level power estimation. *Proc. International Symposium on Low Power Electronics and Design*. Monterey, CA, 1996, pp. 29-35.
4. Brooks D., Tiwari V., Martonosi M. Wattch: A framework for architectural-level power analysis and optimizations. *Proc. International Symposium on Computer Architecture*. Vancouver, BC, Canada, 2000, pp. 83-94.
5. Fenton N.E., Pfleeger S.L. *Software Metrics: A Rigorous and Practical Approach*. London, International Thomson Computer Press, 1996.

6. Tiwari V., Malik S., Wolfe A. Power analysis of embedded software: A first step towards software power minimization. *IEEE Transactions on VLSI Systems*, 1994, vol. 2, no. 4, pp. 437-445.
7. Tiwari V., Malik S., Wolfe A., Lee T.C. Instruction level power analysis and optimization of software. *Journal of VLSI Signal Processing*, 1996, vol. 13, no. 2, pp. 1-18.
8. Malik S., Martonosi M., Li Y.-T.S. Static Timing Analysis of Embedded Software. *Proc. Design Automation Conference*. Anaheim, CA, 1997, pp. 147-152.
9. Ye W., Vijaykrishnan N., Kandemir M., Irwin M. J. The design and use of SimplePower: A cycle-accurate energy estimation tool. *Proc. Design Automation Conference*. LosAngeles, CA, 2000, pp. 340-345.
10. Catthoor F., Wuytack S., De Greef E., Balasa F., Nachtergaele L., Vandecappelle A. *Custom Memory Management Methodology: Exploration of Memory Organization for Embedded Multimedia System Design*. Boston, Kluwer Academic Publishers, 1998.
11. Furber S. *ARM System-on-Chip Architecture*. Harlow, Addison-Wesley, 2000.
12. Kandemir M., Vijaykrishnan N., Irwin M.J., Ye, W. Influence of compiler optimizations on system power. *Proc. Design Automation Conference*. LosAngeles, CA, 2000, pp. 304-307.
13. Dasigenis M., Kroupis N., Argyriou A., Tatas K., Soudris D., Zervas N., Thanailakis A. A memory management approach for efficient implementation of multimedia kernel on programmable architectures. *Proc. IEEE Computer Society Annual Workshop on VLSI*. Orlando, Florida, 2001, pp. 171-176.
14. Sinevriotis G., Stouraitis Th. *SOFLOPO: low power software development for embedded applications*. ESPRIT ESD-LPD Project 25403, Public Final Report, 2001.
15. Golub G.H., Van Loan C.F. *Matrix computations*. 3rd ed. The Johns Hopkins University Press, 1996. 728 p. (Russ. ed.: Golub Dzh., Van Loun Ch. *Matrichnye vychisleniia*. Moscow, Mir, 1999. 548 p.).
16. *Paket instrumental'nykh sredstv razrabotki i otladki mikrokontrollerov ARM* [Toolkit of development and debugging of ARM microcontrollers]. Available at: <http://www.phyton.ru> , accessed 31.07.2012.