Энергосберегающая оптимизация кода за счет использования отключаемых компонентов процессора

И.И. Каретин, В.А. Макаров
IlyaK@astrosoft.ru, Vladimir.Makarov@novsu.ru

Аннотация. Статья посвящена рассмотрению вопроса снижения энергопотребления в процессе исполнения кода программ. Объектом исследования является техника отключения отдельных компонентов процессора. Описываются теоретические аспекты модификации исходного кода с целью повышения энергетического эффекта от отключения компонентов.

Цель оптимизирующего компилятора заключается в получении эффективного кода. В качестве критерия его эффективности может быть выбрано быстродействие, занимаемый объем или энергопотребление. В данной статье будут рассматриваться оптимизации по критерию энергопотребления кода.

Под энергопотреблением кода будем понимать то количество энергии, которое будет потрачено на целевой платформе при его выполнении. Как следует из данного определения, сравнение алгоритмов по эффективности по выбранному критерию возможно только с указанием платформы, для которой производится исследование. Будем говорить, что код более энергоэффективен, если его энергопотребление ниже.

Значение энергосберегающих компиляторов с течением времени только возрастает, так как аппаратные методики экономии энергии достаточно дороги и, как правило, требуют значительного перепроектирования архитектуры. В это же время область генерации энергоэффективного кода остается изученной явно недостаточно, хотя ее углубленная разработка способна оказать значительный эффект на время работы устройств, размеры их элементов питания, тепловыделение и прочие характеристики, связанные с показателями энергопотребления.

Данная работа посвящена исследованию возможности снижения энергопотребления кода за счет применения техники отключения некоторых компонентов процессора. Основной идеей является использование модификации исходного кода для повышения энергетического эффекта от отключения компонентов. Подробное описание такой целевой платформы

можно увидеть в работе [1]. Далее оно будет приведено только в объеме, необходимом для понимания предлагаемой методики.

Представим центральный процессор как систему, включающую в себя компоненты, которые могут быть включены или отключены в заранее определенные моменты времени. Для подключения и отключения этих компонентов используется специальный набор команд.

В данной архитектуре у нас есть 5 устройств, которые могут подключаться и отключаться в произвольные заранее заданные моменты. Это целочисленное АЛУ, целочисленный умножитель, вещественный сумматор, вещественный умножитель и вещественный делитель.

Предлагаемый метод оптимизации включает в себя несколько последовательных этапов. Сначала производится анализ потока управления программы и на его основе в коде программы устанавливаются потенциальные места для добавления специальных инструкций, отключающих либо активирующих отдельные компоненты процессора. Например, если в ходе вычислений нам продолжительное время не требуется модуль целочисленного умножения, то на определенном участке кода он может быть отключен для экономии энергии. При этом следует правильно оценивать время простоя этого модуля: нужно добиться того, чтобы накладные расходы на его отключение и последующее включение не превысили бы экономию за период отсутствия активности.

В данной работе будет рассматриваться метод оптимизации компонентов по отдельности. Взаимные влияния компонентов не анализируются, это будет темой отдельного исследования.

Обозначим рассматриваемый компонент символом C. Все дальнейшие рассуждения будут касаться только его. Затем данный метод должен быть применен для каждого из оставшихся компонентов.

При оценке целесообразности отключения компонента системы следует пользоваться такой формулой:

$$E_{\text{turn off}}(C) + E_{\text{turn on}}(C) \le \sum_{i=1}^{N} P_i(C),$$

Где: $E_{\text{turn off}}(C)$ – затраты энергии на отключение компонента *C*;

$E_{\text{turn on}}(C)$ – затраты энергии на активизацию компонента *C*;

N – количество инструкций в блоке, для которого компонент может быть отключен;

$P_i(C)$ – энергия, которая может быть сэкономлена за один цикл бездействия компонента *C* при выполнении инструкции *i*;

Следует отметить, что в отличие от формул, приведенных в работе [1], тут учитывается то, что энергетический эффект инструкции не является постоянной величиной и может отличаться при выполнении различных инструкций программы.

В том случае, если указанное неравенство выполняется, т.е. накладные расходы не превышают экономию энергии, то рассматриваемый участок кода специальным образом помечается, что означает, что далее в него нужно будет добавить инструкции для временного отключения компонента C. В противном случае код должен быть оставлен без изменений. При этом даже если условие не выполнилось, следует учитывать, что существует вероятность того, что к коду могут быть применены дополнительные оптимизации, в результате чего отключение компонента C все-таки может стать оправданным. Исследование условий, при которых возможно получение дополнительных участков, где выполняется указанное условие, будет темой отдельной работы.

В результате некоторые участки кода получили метки с указанием, какой из компонентов не требуется каждому из них.

Далее выполняется непосредственная оптимизация кода на основании собранной информации. Ниже будет рассмотрено несколько вариантов возможных оптимизаций, связанных с ветвлением или наличием циклов в программе. Все представленные ниже оптимизации объединяет то, что в результате преобразования кода не изменяется относительный порядок следования операторов. Изменение такого порядка потребовало бы дополнительного анализа зависимостей в программе, и потому оно будет темой отдельного исследования. Таким образом, в данной работе не будет рассматриваться оптимизации линейных участков программы, оптимизироваться будут только ветвления и циклы.

Итак, рассмотрим первый из возможных случаев применения оптимизации. Пусть перед ветвлением программы у нас есть n инструкций, в которых не требуется работа компонента C. Следует сразу определиться с условными обозначениями. На приведенных ниже схемах «N» обозначает блок из n инструкций, которому не требуется компонента C для работы. Аналогично «K_i» обозначает максимальный блок из k_i начальных инструкций в i-й ветви программы, которым не требуется компонент C. «C=0» обозначает, что компонент C в данном участке программы не требуется, а «C=1» обозначает границу блока инструкций, которым требуется компонент C. D_i обозначает возможную точку отключения (Disable), а E_i – возможную точку включения (Enable) компонента C соответственно. Следует отметить, что значения n и k_i лежат в диапазоне [0; s), где s – это общее количество инструкций программы. Другими словами, любой из этих блоков может быть пустым. Общее количество ветвей обозначим f. Таким образом: i Є [1, f]. Эти же обозначения будут использоваться и при рассмотрении всех последующих примеров.

На следующем рисунке показан общий случай ветвления в программе.

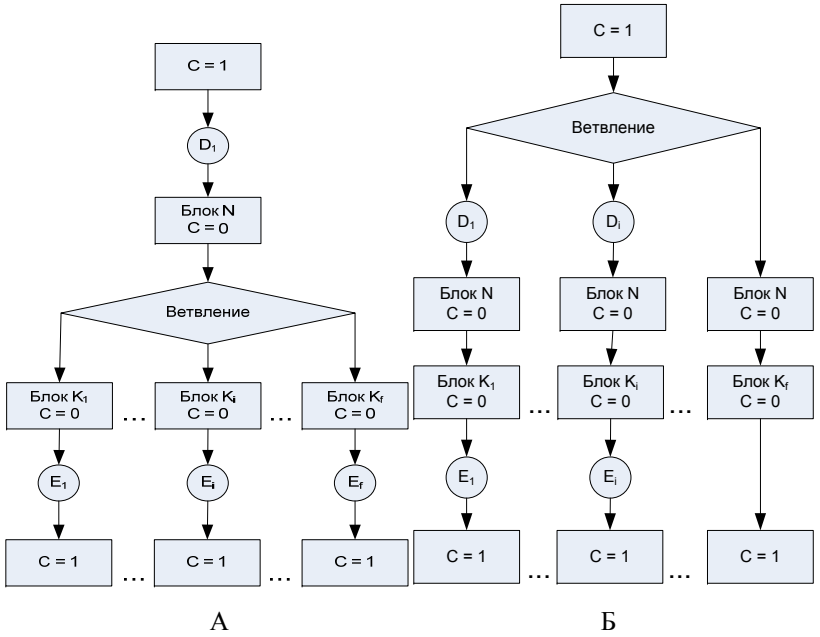

Рис. 1. Ветвление программы до (А) и после (Б) оптимизации.

После проверки условий ветвления, выполнение программы пойдет по одному из f различных путей. В данном случае из всего кода нас интересует только последний блок инструкций перед ветвлением (блок N) и первый блок инструкций в каждой из ветвей (блоки K_i). Линейные участки кода до ветвления и внутри ветвей не рассматриваются по описанным выше причинам.

Обозначим P(N) количество энергии, которое может быть сэкономлено в случае, если компонент C будет отключен при выполнении блока N. Аналогично $P(K_i)$ будет обозначать экономию энергии в первом блоке i-й ветви. Обозначим P_{min} то количество энергии, которое требуется на отключение и последующее включение компонента C.

В том случае, если выполняется условие $P(N) + \min_{i=1}^{f}(P(K_i)) \geq P_{min}$, то при движении по любой из ветвей мы получаем некоторый выигрыш от отключения компонента в точке D_1 и подключения его во всех точках E_i непосредственно перед инструкциями, обозначенными как C=1. В худшем случае, в некоторых ветвях экономия равна затратам на отключение и подключение компонента, но при этом существуют ветви, где энергия действительно экономится. Тогда никакого дополнительного изменения кода

не требуется и инструкции отключения и подключения компонента C могут быть добавлены в код в отмеченных местах.

В противном случае, если условие не выполняется, то отключение C для всех ветвей неэффективно. Тогда требуется перенос общих инструкций из блока N, не требующих компонента C, в каждую из ветвей для того, чтобы можно было индивидуально принимать решение о том, требуется ли отключать компонент C в данной ветви. В таком случае код должен быть преобразован к виду, показанному на рисунке 1-Б. Общий участок кода N дублируется в каждой из ветвей.

Как можно заметить, для упрощения анализа в данном случае опущены эффекты от операций проверок и переходов, обозначенных на схеме как «ветвление».

Аналогично следует рассмотреть и случай окончания ветвления, показанный на следующем рисунке 2-А.

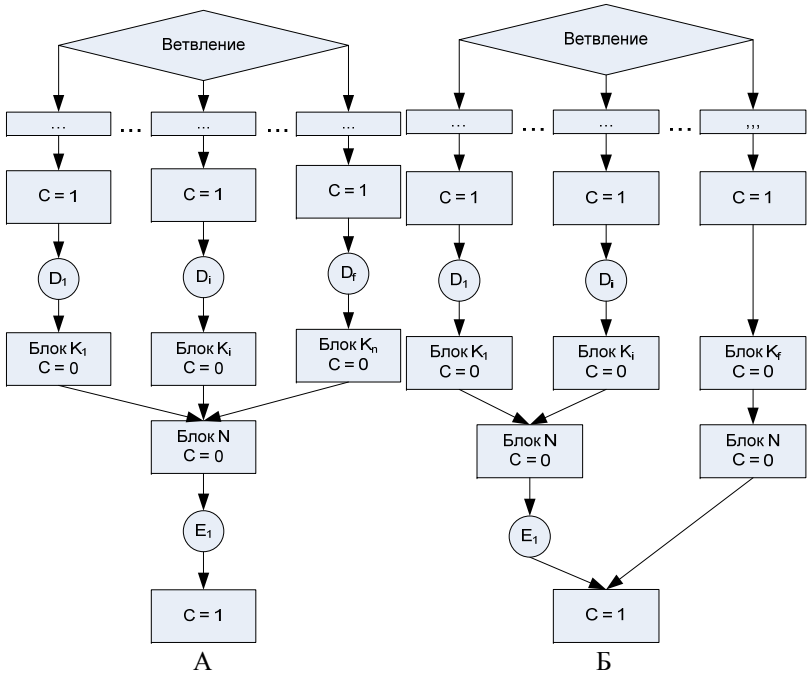

Рис. 2. Участок окончания ветвления программы до (А) и после (Б) оптимизации.

Аналогично предыдущему случаю, если согласно формуле (1) отключение компонента C будет эффективным во всех ветвях непосредственно перед

точкой объединения, то структуру программы можно оставить без изменений. Формула оценки целесообразности отключения компонента тут остается такой же, как и в прошлом случае: $P(N) + \min_{i=1}^{f} (P(K_i)) \geq P_{min}$.

Если хотя бы в одной из вервей не эффективно отключение компонента C, то необходимо производить дополнительный перенос инструкций. В результате данный участок программы будет приведен к виду, показанному на Рис. 2-Б.

Как и в прошлый раз, все ветви разделяются на 2 класса – те, в которых производить отключение компонента C эффективно и те, в которых оно не приводит к положительному эффекту. После этого общий набор инструкций (N) дублируется и каждая из двух его копий помещается после своего класса ветвей. Далее для ветвей, где производилось отключение компонента C, добавляется команда на включение этого компонента (E_1). В ветвях, где отключения не производилось, нет необходимости и для добавления команды на его подключение.

Теперь рассмотрим, как нужно оптимизировать код в случае циклов.

На Рис. 3-А изображен цикл, перед началом которого есть блок из n_1 инструкций, не требующих работы компонента C. Также в начале тела цикла у нас есть блок, состоящий из k_1 инструкций, которым также не требуется компонент C. Важной особенностью этого участка кода является то, что на первой итерации цикла блок K_1 выполняется непосредственно после блока N, а на всех последующих итерациях цикла – после блока K_f.

В конце цикла есть блок K_f, который не требует компонента C, а сразу после окончания цикла есть еще блок N_2, также не требующий компонента C. Следует помнить, что любые из этих блоков могут быть пустыми.

Данный код может оставаться в неизменном виде, если выполняются все условия:

1) $P(N_1) + P(K_1) \geq P_{min}$
2) $P(K_f) + P(K_1) \geq P_{min}$
3) $P(K_f) + P(N_2) \geq P_{min}$

Другими словами, если от момента отключения компонента до момента его включения всегда выполняется достаточное количество инструкций, чтобы эффект экономии стал положительным, то такая схема программы имеет право на существование.

В противном случае энергосбережение будет не оптимальным и инструкции должны быть переупорядочены.

Если у нас выполняется условие $P(N) + P(K_1) \geq P_{min}$, то значит, мы можем получить экономию по крайней мере при выполнения первой итерации цикла. В этом случае имеет смысл произвести развертывание цикла и вынести первую итерацию так, чтобы для нее можно было бы добавить отдельный набор инструкций для отключения компонента C. Для того чтобы уменьшить

количество копируемого кода можно выносить из цикла не все его тело, а только блок K_1. Тогда программа примет вид, показанный на рисунке 3-Б.

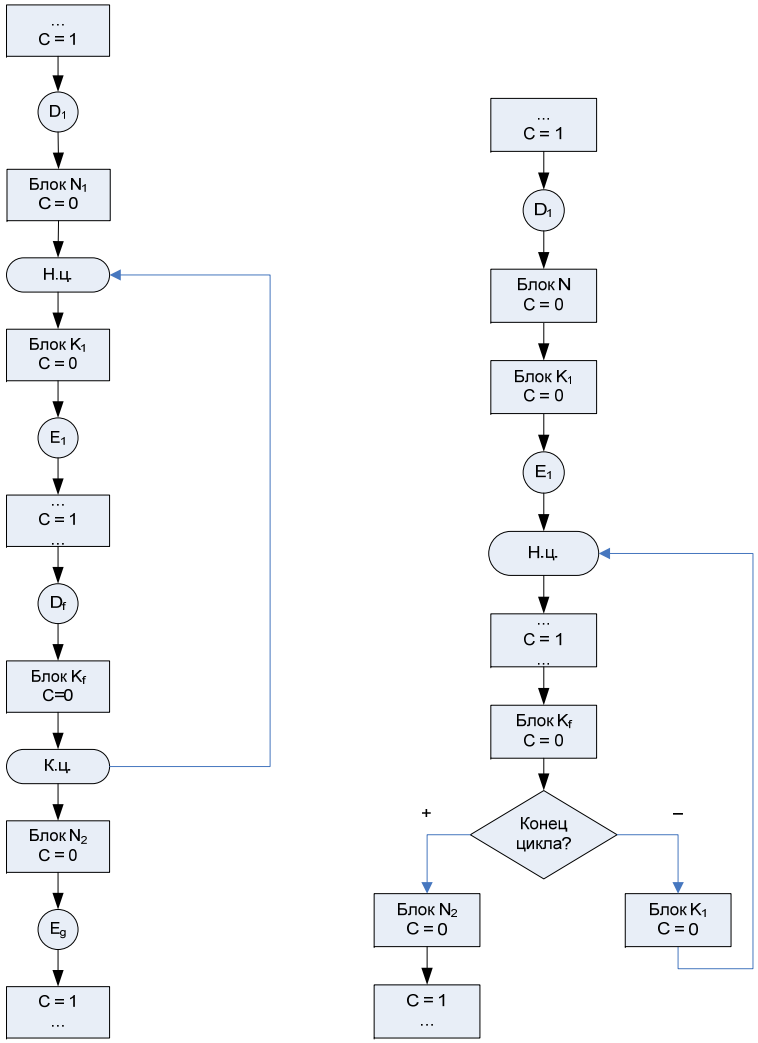

Рис. 3 – цикл до оптимизации (А) и после оптимизации первой итерации (Б).

Если у нас выполняется только условие $P(K_f) + P(N_2) \geq P_{min}$, то требуется вынести из тела цикла только его последнюю итерацию и вставить инструкцию D_g только в нее, оставив также инструкцию E_g после выполнения

блока инструкций, расположенных за циклом. Все остальные случаи оптимизации этого цикла различаются только тем, какие из трех условий будут выполняться. В зависимости от комбинации условий, отличия в оптимизациях будут заключаться только в том, какие из итераций (первая или последняя) должны быть вынесены из цикла, и в каких из указанных на рисунке 3-А позиций D_i и E_i должны быть установлены инструкции отключения или включения анализируемого компонента.

В заключении следует еще раз отметить, что предметом рассмотрения данной статьи были модификации кода без изменения относительного порядка следования инструкций. В ходе работы над статьей было еще раз отмечено, что для эффективного решения поставленной задачи должны быть дополнительно рассмотрены и некоторые смежные области. Так, например, в данной работе не производилось анализа возможностей переупорядочивания инструкций на линейных участках программы. Для того чтобы проделать эту работу требуется произвести анализ зависимостей в исходном коде. Такое исследование будет произведено в одной из следующих статей в этом направлении. За основу такого анализа можно взять статью [2], где описываются общие методы оптимизации, а также рассматриваются виды зависимостей.

Еще одним направлением дальнейших исследований является анализ результатов применения широко известных оптимизаций быстродействия для решения задач энергоэффективности. Сделанные наблюдения и анализ литературы показывают, что хотя данные области и располагаются достаточно близко друг к другу, но, тем не менее, на данный момент нельзя точно обосновать зависимость получаемого энергетического эффекта от таких оптимизаций.

Более того, задача построения оптимальной последовательности оптимизаций быстродействия на данный момент также не решена в общем виде, а значит, нет надежных метрик, позволяющих сравнить два различных набора оптимизаций, чтобы выбрать наиболее эффективный из них. Вопрос построение набора метрик также станет темой отдельного исследования.

Следует учитывать также взаимные влияния отключаемых компонентов друг на друга. В данной статье было произведено только исследование независимых компонентов. Анализ возможных зависимостей между ними также станет темой отдельной статьи.

Литература

[1] Yi-Ping You, Chingren Lee, Jenq Kuen Lee, Compilers for leakage power reduction, ACM Transactions on Design Automation of Electronic Systems, v.11 n.1, p.147-164, January 2006

[2] David F. Bacon , Susan L. Graham , Oliver J. Sharp, Compiler transformations for high-performance computing, ACM Computing Surveys, v.26 n.4, p.345-420, Dec. 1994