гонки на вымирание, девяносто пятые выживают

крис касперски ака мыщъх, no e-mail

...и дернул же меня черт... хотя нет, он как раз отговаривал... я тогда искусственным интеллектом увлекался, вот и создал на свою голову... компилятор GCC.

из баек отечественных программистов

среди компиляторов — хороших и разных — как выбрать единственно правильный свой? не верьте ни советам друзей, ни рекламным листкам, ни даже этой статье. но все-таки ее прочитайте. так вы узнаете сильные и слабые стороны популярах компиляторов и найдете сравнительные тесты их быстродействия.

введение

Сравнение компиляторов — дохлое дело. Религиозные войны. Фанатизм. Бенчмарки. Объективных критериев оценки ни у кого нет, да и не может быть по определению (что русскому хорошо…) Всегда найдутся условия, на которых ваш компилятор уделывает всех остальных. Комплексные тесты только запутывают дело. Отображаемая ими "среднегодовая" температура не имеет ничего общего ни с тропической жарой, ни с арктическими морозами. Может, человеку целочисленное приложение компилировать надо, а основной вклад в комплексный тест дают плавающие операции.

Адепты максимальной оптимизации, собирающие все пакеты вручную, испытывают большие трудности с выбором "единственного правильного" компилятора. Многообразие версий GCC их угнетает, а тут еще мощный конкурент в лице Intel нарисовался. Основным системным компилятором большинство дистрибьютивов Линуха назначают GCC 2.95. В портах лежит GCC 3.2/GCC 3.3. Более свежие версии приходится добывать в Интернете самостоятельно.

Возникает естественный вопрос — оправдывает ли себя переход с GCC 2.95 на GCC 3.x или, может быть, лучше эмигрировать на другой компилятор? Если говорить кратко — на вкус и цвет товарищей нет. **GCC 2.95 – это максимальная совместимость и быстрота компиляции. ICC 8.x – наивысшая производительность откомпилированного кода. GCC 3.x – рекордсмен по оптимизации векторных приложений под Атлон и другие процессоры фирмы AMD.**

А теперь обо всем этом и многом другом поподробнее.

два лагеря — пользователи и программисты

Требования, предъявляемые программистами к компилятору, совсем не те, что у пользователей. Лозунг "время трансляции имеет значение!" отвергается пользовательским сообществом как маразм, не требующий объяснения. В самом деле, какой процент своего времени тратит на перекомпиляцию рядовой линуксоид? А программист? Пользователю глубоко начхать час или два оно будет компилироваться. Главное, чтобы получился хороший машинный код. Все остальное несущественно. Программисты же на первое место выдвигают именно скорость трансляции, а к быстродействию собственной продукции они в общем-то равнодушны (даже если им же на ней и работать!).

Достоинство GCC 2.95 в его быстроте. Версии 3.x компилируют программы чуть ли не в два раза медленнее, а ведь время это не только деньги, но и срыв всех сроков разработки. Обновить компьютер? Но многие и так работают на самом мощном железе, которое только доступно, да и не будет никто просто так выкладывать деньги только затем, чтобы перейти на новую версию GCC, когда и старая еще неплохо работает.

К новомодным (а, значит, еще не обкатанным) алгоритмам агрессивной оптимизации программисты относятся весьма настороженно, можно даже сказать скептически. Ведь за мизерное увеличение производительности зачастую приходится расплачиваться потерей работоспособности программы. Рассмотрим следующий код: `for(a=0;a<func();a++)`. Очевидно, что функция func() инвариантна по отношению к циклу и с "математической" точки зрения может быть вынесена за его пределы. Однако, перед этим оптимизатор должен проанализировать ее тело — вдруг там присутствуют побочные эффекты типа вызова printf,

модификации статической/глобальной переменной, обращения к портам ввода/вывода, передачи управления по указателю и т. д. и не факт, что транслятор это "заметит". Использование оптимизации в GCC 3.x напоминает хождение по минному полю — такое количество ошибок скрывается в компиляторе.

Компилятор ICC совмещает в себе высокую скорость трансляции с хорошим качеством результирующего кода, однако, он не обходится без недостатков. Это коммерческий продукт с закрытыми исходными текстами, поддерживающий только платформу x86, заточенный под процессоры Intel, да к тому же еще и не бесплатный (бесплатность для некоммерческого применения не в счет, это в молодости мы шашки наголо и айда, но по мере углубления в лес все больше хочется кушать). К тому же, никакой уверенности, что завтра ICC не коммерцинализируются окончательно у нас нет (скорее всего, именно так все и ~~будет~~произойдет).

Тем не менее, ряды поклонников ICC ширятся с каждым днем и на то есть свои причины. Это лучший компилятор для платформы Intel (а под другие платформы большинству ничего компилировать и не нужно). Он отлично документирован, служба технической поддержки работает честно и оперативно (GCC по сути не поддерживается вообще, разработчики совершенствуют компилятор в свое удовольствие, а эти противные пользователи им только мешают). Вместе с ICC поставляется набор высокопроизводительных библиотек с заготовками на все случаи жизни и оптимизатор VTune (хотя последний может работать и с другими компиляторами, связка ICC + VTune наиболее удобна и эффективна). Это не пересказ рекламного проспекта, а личные впечатления. Не даром фирма QNX выбрала ICC основным компилятором для своей OS реального времени! Однако, их выбор — это все-таки _их_ выбор. Он не должен быть опорой для вашего собственного мнения. В конце концов, никто не запрещает использовать оба компилятора попеременно и проблема "или—или" здесь не стоит.

вавилонская башня языка Cи/Cи++

Качество Open Source проектов (чтобы там не говорили их поклонники) вообще говоря очень невелико. Когда программа компилируется — это уже хорошо, а если при этом она еще и работает… Всякая попытка оптимизации (или переход на другой транслятор) разваливает хрупкое программистское строение окончательно. Программа либо перестает компилироваться совсем, либо в ней заводится глючный баг. Работающие на голом энтузиазме разработчики просто не в состоянии опробовать все версии всех компиляторов, а ведь различия между ними очень значительны. Вот только один пример: в GCC 3.x из класса std::fstream изъяли конструктор fstream(int) и метод attach(int), в результате чего объявления вида "`fstream* FS = new fstream(fd)`" перестали работать. Еще одна жертва заявленной совместимости со Стандартом! Впрочем, неприятность эту можно обойти: написать свой класс, производный от std::streambuf, создающий streambuf поток (что долго, зато портабельно), или использовать гнусное расширение __gnu_cxx::stdio_filebuf ("GNUcное" — потому что непортабельное), но это работа программиста, а не конечного пользователя!

Со времен GCC 2.95, поддержка плюсов претерпела существенные изменения. В основном — положительные. Контроль за ошибками ужесточился. Это хорошо (хотя внешние верификаторы кода типа LINT еще никто не отменял и во многих случаях они предпочтительнее). А вот с классическим Cи появились проблемы. Оптимисты верят, что он компилируется не хуже, чем вчера. Пессимисты же закидывают их дизассемблерными листингами, убеждающими, что новые версии GCC генерируют более громоздкий и менее эффективный код, а часть конструкций не компилируется вообще!

Объем программ, компилируемых только теми версиями компиляторов, под которыми они разрабатывались~~, в действительности очень велик~~ очень значителен (и не важно где зарыта ошибка — в листинге программы или компиляторе, пользователям от этого не становится легче). Древний GCC 2.95 поддерживается большинством производителей и генерирует достаточно качественный даже по сегодняшним меркам код. Поэтому-то составители нормальных дистрибьютивов и устанавливают его основным системным компилятором по умолчанию, оттесняя всех конкурентов в порты.

По утверждению фирмы Intel, ICC практически полностью совместим с GCC. Он нормально компилирует линуховое ядро версии 2.4, однако, спотыкается на 2.6, требуя специальных заплаток. Одна правит исходный код ядра, другая — сам компилятор. Прикладное программное обеспечение без напильника и ритуальных танцев с бубном так же не обходится.

В общем, слабонервным товарищам на эффективность лучше забить. Оставьте GCC 2.95 основным системным компилятором и никуда с него не сходите.

качество оптимизации или мегагерцы, спрессованные в стрелу времени

Компилировать надо компилятором, а оптимизировать — головой. Оптимизирующий компилятор увеличивает скорость программы главным образом за счет того, что выбивает из нее весь shit. Чем качественнее исходный код, тем меньший выигрыш дает оптимизатор, которому остается всего лишь заменять деление умножением (а само умножение логическими сдвигами) и планировать потоки команд. "Четверки" и первые модели Пней имели довольно запутанный ритуал спаривания и для достижения наивысшей производительности машинные инструкции приходилось радикально переупорядочивать, причем расчет оптимальной последовательности представляет собой весьма нетривиальную задачу, благодаря чему качество оптимизации разнилось от одного компилятора к другому. Но с появлением Pentium Pro/AMD K5 эта проблема сразу стала неактуальной — процессоры поумнели настолько, что научились переупорядочивать машинные команды самостоятельно и интеллектуальность оптимизирующих компиляторов отошла на второй план.

Не стоит право же, гнаться за новыми версиями оптимизаторов. Эта технология достигла своего насыщения уже в середине девяностых и никаких прорывов с тех пор не происходило. Поддержка мультимедийных SSE/3DNow! команд воздействует только на мультимедийные и отчасти математические приложения, а всем остальным от нее ни жарко, ни холодно. Кривую от рождения программу оптимизатор все равно не исправит (он ведь не бог), а грамотно спроектированный код равномерно распределяет нагрузку по всем функциональным узлам и без острой необходимости лучше его не ускорять. Возьмем сетевое приложение. Оптимизация программного кода увеличивает количество обрабатываемых запросов, что в свою очередь увеличивает нагрузку на сеть, и общая производительность не только не возрастет, но может даже упасть.

Агрессивные алгоритмы оптимизации (за которые обычно отвечает ключ -O3) или, правильнее сказать "пессимизации", зачастую дают прямо противоположный ожидаемому результат. Увлеченные "продразверсткой" циклов и функций они ощутимо увеличивают объем программного кода, что в конечном счете только снижает производительность. А глюки оптимизции? Впрочем, о глюках мы уже говорили.

Не гонитесь за прогрессом (а то ведь догоните), но и не превращайте верность традициям в религиозный фанатизм. Глупо отказываться от "лишней" производительности, если компилятор дает ее даром. Создатели GCC говорят о 30% превосходстве версии 3.0 над 2.95, однако, далеко не все разработчики с этим согласны. Большинство вообще не обнаруживает никакого увеличения производительности, а некоторые даже отмечают замедление. Ничего удивительного! Алгоритмы оптимизации в GCC 3.x претерпели большие изменения. Одни появились, другие исчезли, так что в целом ситуация осталось неизменной. Только вот про поддержку новых процессоров не надо, а? С планированием кода они справляются ~~сами~~ и самостоятельно (учет особенностей их поведения дает считанные проценты производительности, да и то на чисто вычислительных задачах), а новые векторные регистры и команды просто так не задействуешь. Одной перекомпиляции здесь недостаточно. Программный код должен использовать эти возможности явно. Эффективно векторизовать код ~~пока~~ не умеют даже суперкомпьютерные компиляторы. Во всяком случае пока. Или точнее уже.

А что насчет сравнения 32-разядного кода с 64-разрядным? Пользователь думает: 64 намного круче, чем 32, а, следовательно, и быстрее! Начинающий программист: ну, может, и не быстрее (все равно все тормозит ввод/вывод), но что не медленнее — это точно! А вот хрен вам! Бывалые программисты над этим только посмеиваются. Медленнее! Еще как медленнее! 32-разрядный код по сравнению с 16-разрядным в среднем потребляет в 2 —2.5 раза больше памяти. 64-разрядный код — это вообще монстр, разваливающийся под собственной тяжестью. А ведь размер кэш-буферов и пропускная способность системной шины не безграничны! "Широкая" разрядность дает выигрыш лишь в узком кругу весьма специфических приложений (большей частью научных). Скажите, вам часто приходится сталкиваться с числами порядка 18.446.744.073.709.551.615? Тогда с какой стати ждать ускорения?

Данные, полученные Тони Бруком (Tony Bourke) с www.OSnews.com (см. рис.1), это полностью подтверждают (http://www.osnews.com/story.php?news_id=5830&page=1). GCC 2.95.3 генерирует чуть-чуть более быстрый код, чем GCC 3.3.2, а 64-битная версия GCC 3.3.2

находится глубоко в заднице и конкретно тормозит. Сановский компилятор рулит и в обоих случаях, но 64-разрядный код все равно много медленнее.

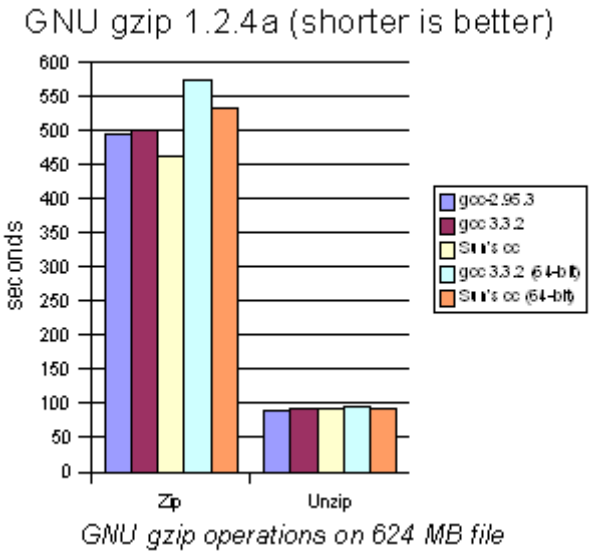

Рисунок 1 сравнение качества кодогенерации различных компиляторов на примере утилиты GZIP (лучшему результату соответствует меньшее значение).

Только не надо говорить, что мы выбрали "неудачный" пример для сравнения! GZIP – типичное системное приложение и на большинстве остальных результат будет таким же. Мультимедийные и математические приложения при переходе на GCC 3.x могут ускорить свою работу в несколько раз и упускать такой выигрыш нельзя. Жаба задушит. Но какую версию выбрать? "Новое" еще не означает "лучшее", а неприятную тенденцию ухудшения качества кодогенерации у GCC мы уже отмечали.

Тестирование показывает, что пальма первенства принадлежит GCC 3.2.3, а GCC 3.3 и GCC 3.2.0 на несколько процентов отстают по скорости (см .рис. 2, 3). Вроде бы мелочь, а как досадно! Если GCC 3.2.3 отсутствует в портах вашего дистрибьютива — не расстраивайтесь! Вы не многое потеряли! Ставьте любую стабильную версию семейства 3.x и наслаждайтесь жизнью. Специально вытягивать из сети GCC 3.2.3 никакого смысла нет. Если вам действительно нужна производительность — переходите на ICC. Практически все, перекомпилировавшие мультимедийные приложения, подтвердили 20%-30% ускорение по сравнению с GCC 3.x. Целочисленные приложения в обоих случаях работают с той же скоростью или даже чуть медленнее (особенно, если программа была специально заточена под GCC). На процессорах фирмы AMD ситуация выглядит иначе и GCC 3.3 с ключиком -mcpu=cpu-type athlon генерирует на 30%-50% более быстрый код чем ICC 8.1. Речь, разумеется, идет только о векторных операциях, а на целочисленных ICC по прежнему впереди.

Приплюснутые программы главным образом выигрывают от того, что у ICC более мощная STL, оптимизированную под Hyper Threading, однако, на классический Си эта льгота не распространяется и такие приложения лучше всего компилировать старым добрым GCC 2.95. Исключение, пожалуй, составляет программы, интенсивно взаимодействующие с памятью. Оптимизатор ICC содержит специальный алгоритм, позволяющий ему выхватить несколько дополнительных процентов производительности за счет за счет механизма предварительной выборки и учета политики кэш-контроллера первого и второго уровней (подробности можно найти в моей книге "Техника оптимизации — эффективное использование памяти").

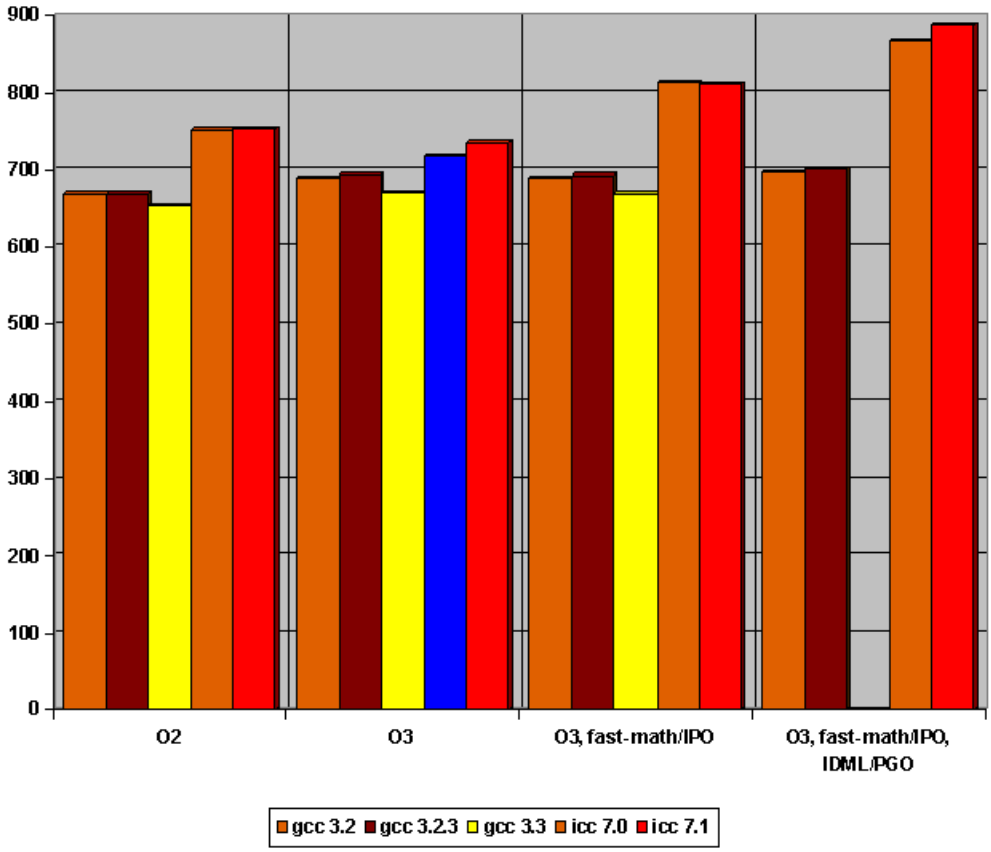

Рисунок 2 сравнение качества кодогенерации по данным теста ROOT (имитатор финансовых приложений). Большее значение — лучшая скорость.

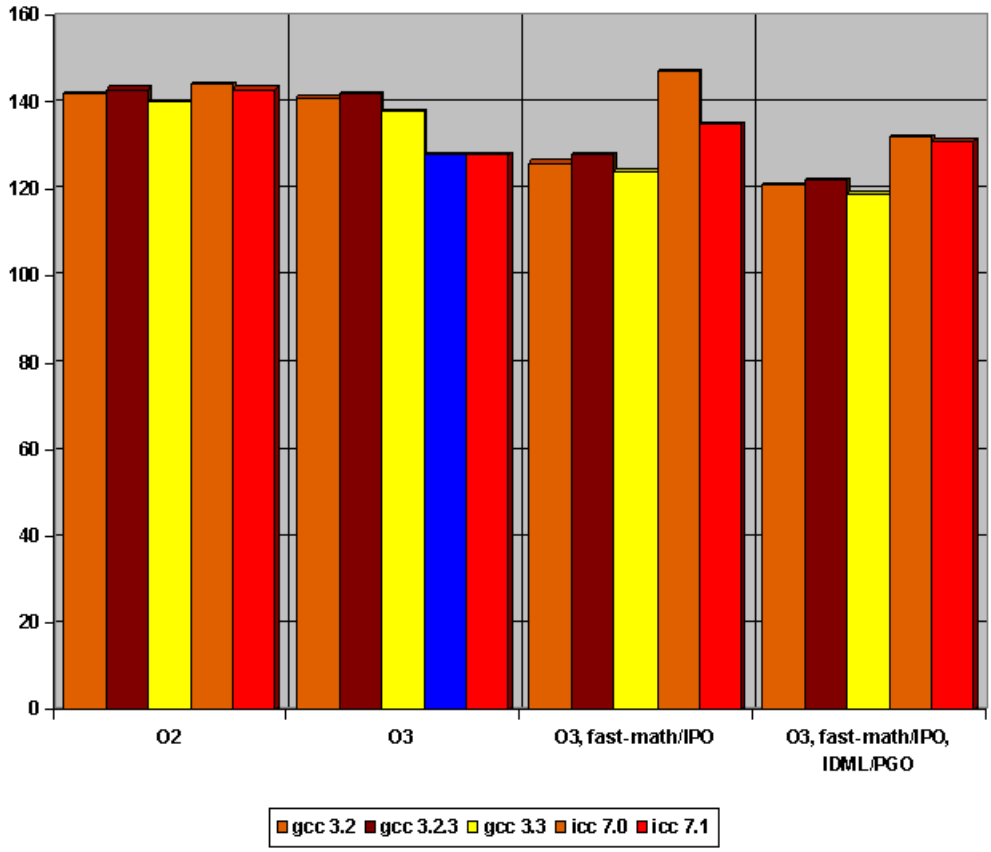

Рисунок 3 сравнение качества кодогенерации по данным теста GEANT4 (моделирование движения элементарных частиц). Большее значение — лучший результат

>>>> врезка Линус и компиляторы

Как мы выяснили, новые версии GCC не дают никаких преимуществ при перекомпиляции ядра, написанном на классическом Си и оптимизированном вручную. Поэтому, Линус (как и большинство остальных программистов-ядерников) не спешат расставаться со своим любимым GCC 2.95, чем и вызывают недоумение пользователей. Вот фрагмент переписки, выловленных на далеком забугорном форуме.

From: Linus Torvalds [email blocked]
Subject: Re: support of older compilers
Date: Thu, 4 Nov 2004 11:38:47 -0800 (PST)
On Thu, 4 Nov 2004, Adam Heath wrote:

> I didn't deny the speed difference of older and newer compilers.
> But why is this an issue when compiling a kernel? How often do you compile
> your kernel?

First off, for some people that is literally where _most_ of the CPU cycles go.
Second, it's not just that the compilers are slower. Historically, new gcc versions are:

- slower;
- generate worse code;
- buggier.

For a _long_ time, the only reason to upgrade gcc was literally C++ support: basic C support was getting _worse_ with new compilers in pretty much every regard. Things seem to have improved a bit lately. The gcc-3.x series was basically not worth it for plain C until 3.3 or so.

Linus

От: Linus Torvalds [мыло скипнуто]
Тема: Re: поддержка старых компиляторов
Date: Вторник, 4 ноября 2004 11:38:47 -0800 (PST)

Как-то во вторник, 4 ноября 2004, Adam Heath писал:
> Я не отрицаю различия по скорости между старыми и новыми компиляторами.
> Но почему эта проблема встает именно при компиляции ядра?
> Как часто вы перекомпилируете свое ядро?

Во-первых, ядро для многих людей это то место в котором они проводят большую часть своего машинного времени. Во-вторых, дело не только и не столько в том, что компиляторы становятся все медленнее. Исторически, новые версии gcc:

- все тормознутее;
- генерируют худший код;
- багистнее;

На протяжении _многих_лет_, единственной причиной обновления gcc была поддержка Си++. Классический Си поддерживается все хуже и _хуже_, со всем уважением к его разработчикам.

В последнее время ситуация слегка изменилась к лучшему. До появления gcc 3.3 все нововведения серии gcc 3.x не стоили ухудшения поддержки классического Си.

Linus

>>> врезка чем народ компилирует ядро

По данным www.kerneltrap.org более примерно 80% пользователей компилируют ядро новыми версиями GCC. 8% отдают предпочтение GCC 2.95 и 10% не перекомпилируют ядро вообще. Это доказывает, что не всякое господствующее мнение — правильное.

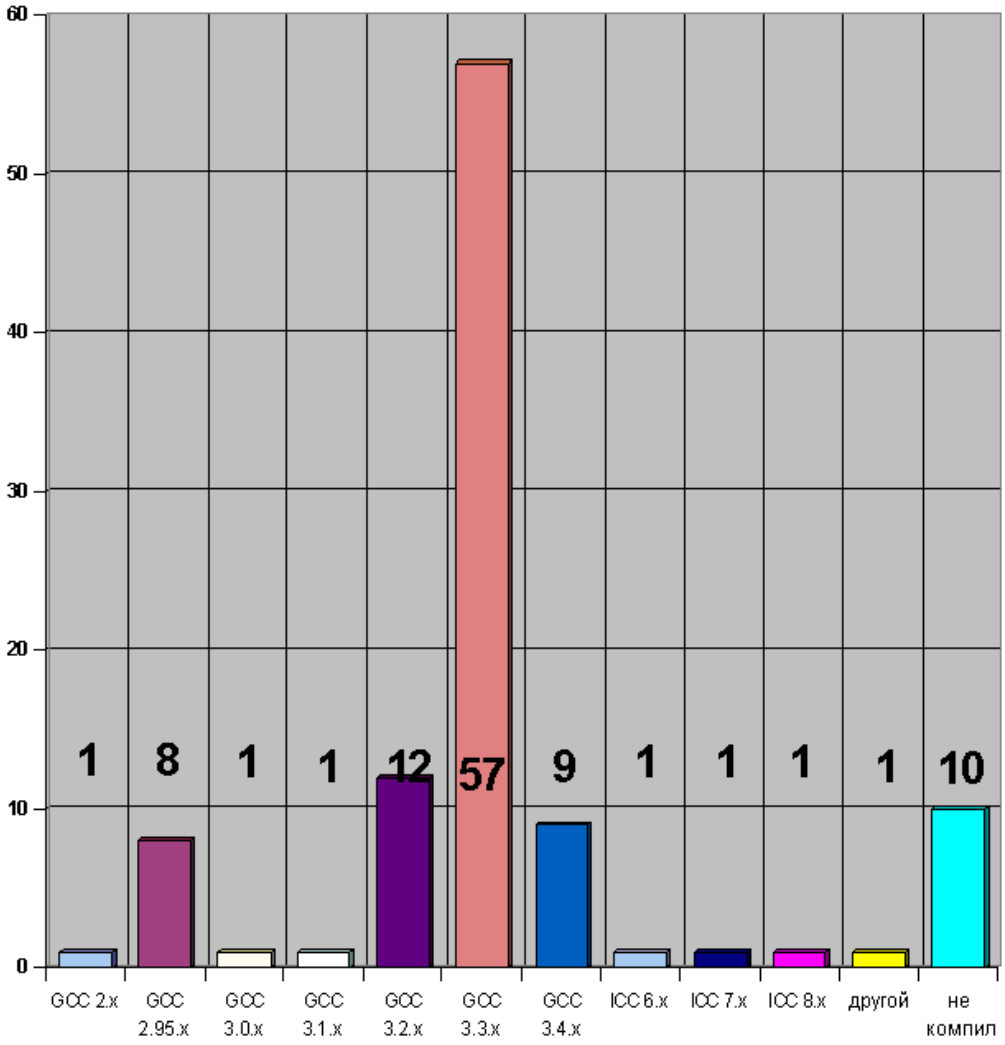

Рисунок 4 какой компилятор вы используете для перекомпиляции ядра? (по данным www.kerneltrap.org на 21 апреля 2004 года) всего в голосовании приняло 3.696 человек

>>> врезка еще тесты

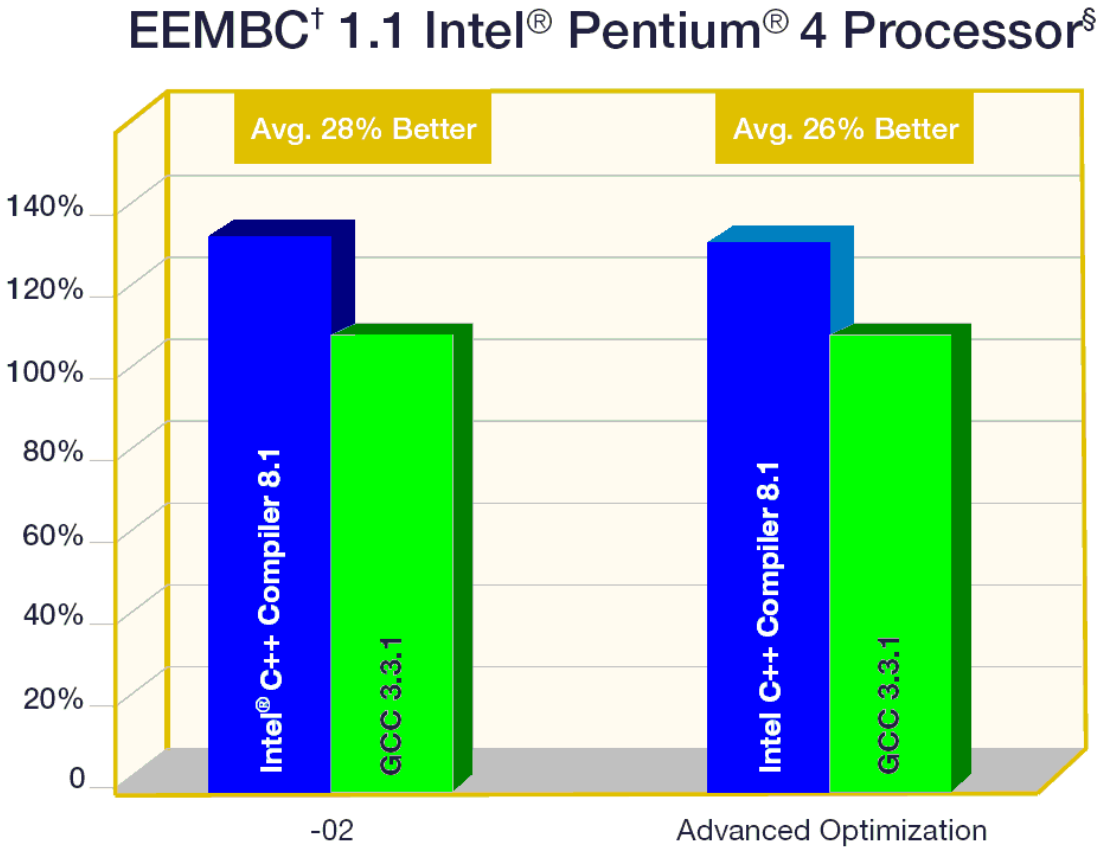

Рисунок 5 сравнительные тесты ICC 8.1 и 3.3.1 на EEMBC 1.1 бенчмарке (EEMBC расшифровывается как Embedded Microprocessor Benchmark Consortium, а меряет эта штука усредненную производительность на репрезентативной выборке из сетевых, офисных и вычислительных тестах).
"Advanced Optimization" подразумевает следующие ключи:
Intel C++ Compiler: -03 -ipo –xW
GCC 3.3.1: -03 -march=pentium4 -mcpu=pentium4 -msse -msse2 –mmmx -funroll-loops -ffast-math -fomit-frame-pointer -mfpmath=sse (подробнее об это можно почитать на www.qnx.com/download/download/10028/Intel_Compiler_Product_Brief.pdf)

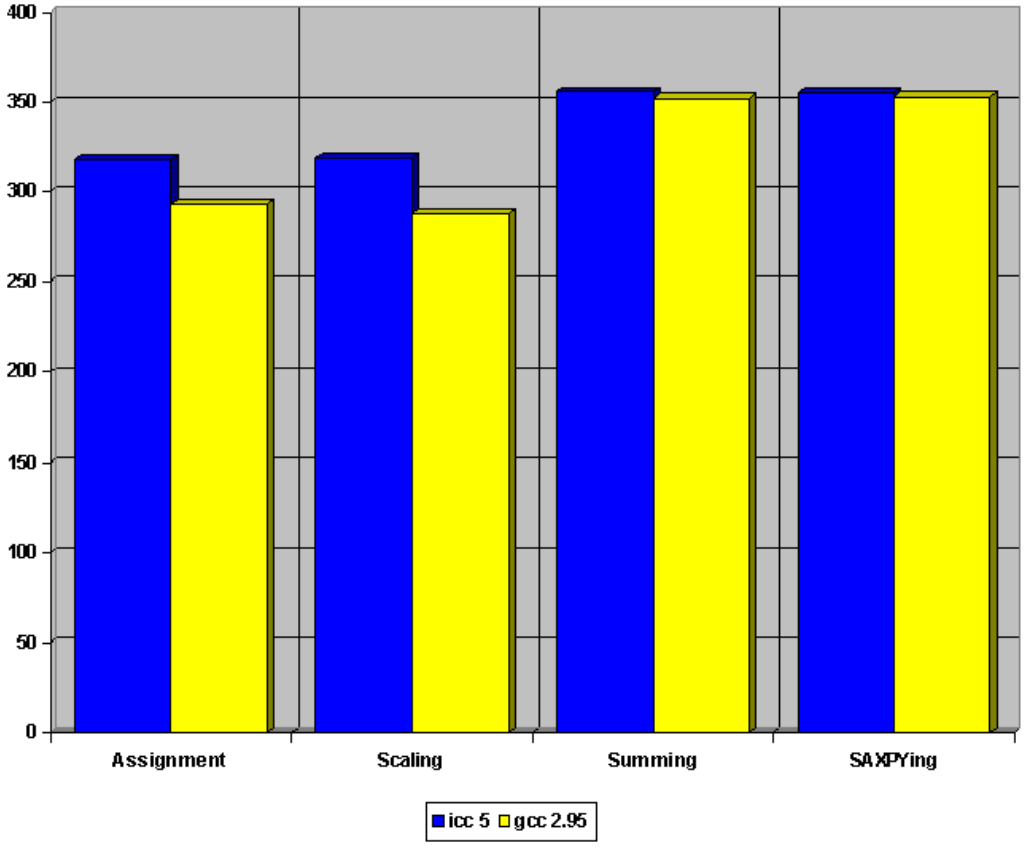

Рисунок 6 сравнение качества кодогенерации по данным теста stream (производительность на операциях с памятью, Мбайт в сек)
Интел более агрессивно выравнивает структуры данных и учитывает архитектуру кэш-контроллера, за счет чего выигрывает несколько процентов производительности

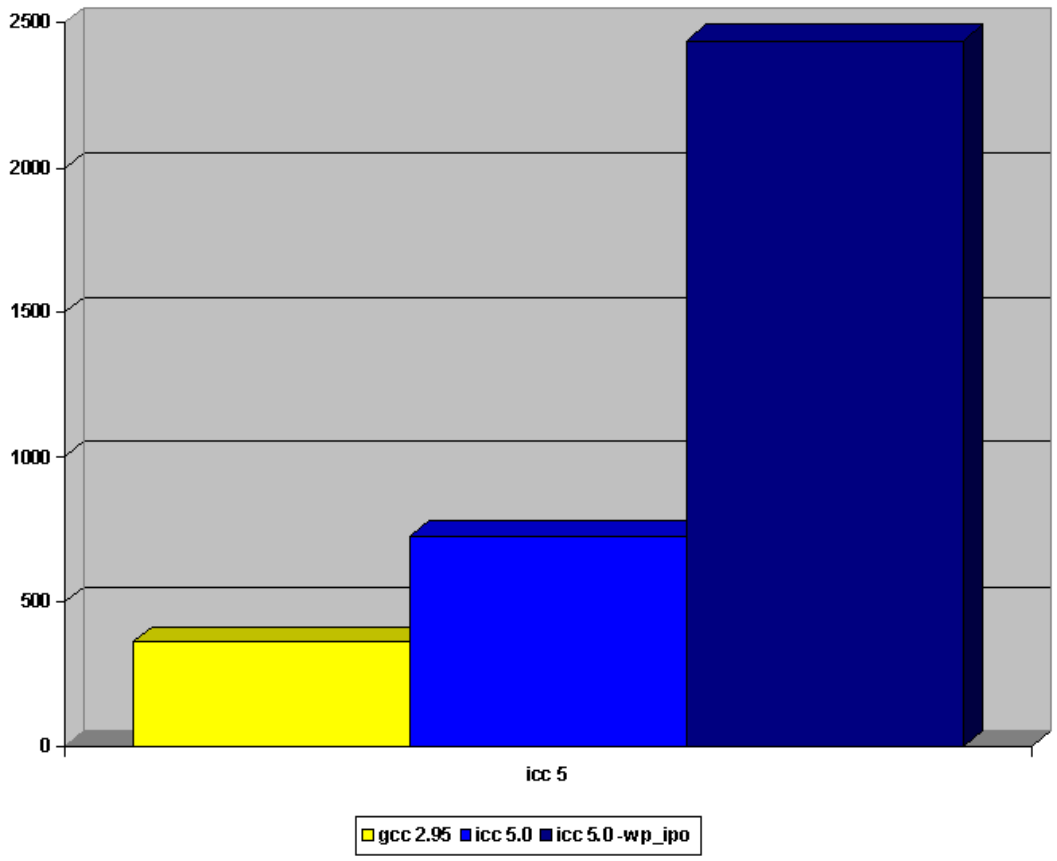

Рисунок 7 сравнение качества кодогенерации по данным теста winstone (комплексный текст с включением мультимедийных приложений, вертикальная шкала — MIPS) безоговорочное превосходство icc — результат использования SSE-регистров, о которых gcc 2.95 ничего не знает

>>> врезка www

Большое количество сравнительных тестов на GCC можно найти на сайте составителя популярного дистрибьютива SuSE: http://www.suse.de/~aj/SPEC/.

оптимизирующие компиляторы наиболее эффективны в оптимизации кривого кода или мультимедийных приложений, задействующих векторные операции. Ни того, ни другого нет в ядре, поэтому его лучше компилировать наиболее устойчивым и совестимым компилятором. Все равно всю производительность съест ввод/вывод

ICC традиционно силен в следующих областях: 3D-приложения, графические программы, математические пакеты, задачи численного моделирования, видео/аудио compression/decompression, плавающая арифметика.

некоторые недобросовестные составители линуховых дистрибьютивов кладут сильно багистную версию GCC 2.96, которая по всем понятием бэта. рекомендуется как можно скорее заменить ее!

ICC поддерживает технологию OpenMP (Open Multi Processing – открытый параллелизм), а GCC пока что нет, поэтому на кластерных установках ICC потенциально способен дать некоторый выигрыш, впрочем, Атлоновый кластер будет все равно дешевле.

заключение

И GCC, и ICC — великолепные компиляторы, вобравшие в себя все достижения прогресса. Тем не менее, далеко не все приобретения пошли им на пользу и программисты в своей массе остаются верны древнему GCC 2.95. Такого же мнения придерживаются составители дистрибьютивов и опытные администраторы. Пользователи, правда, не разделяют сгустившегося духа консерватизма и во всю пользуют GCC 3.2/3.4 и особенно GCC 3.3, вынуждая разработчиков поддерживать новые версии независимо от того хотят они этого или нет. Пропаганда ICC пока только начинается, но учитывая влияние фирмы Intel, а так же высокое качество компилятора, можно не сомневаться, что в некоторых сферах рынка его ожидает успех. Что же касается наукоемких приложений и вычислительных центров — скорее всего они сохранят свою верность процессорам AMD (Пневые кластера стоят намного дороже) и компилятору GCC.

В общем, поживем — увидим. Будем надеяться, что в будущем компиляторы не разжиреют окончательно и здравый смысл восторжествует над разумом. Пока же бесспорных победителей нет и мудрые линуксоиды вынуждены использовать целый зоопарк компиляторов.