Ползать или летать? Компиляторы Intel и их возможности в плане оптимизации ПО.

Харченко Евгений
Intel, Нижний Новгород

intel®

Ползать или летать?

- Вы купите машину у которой из передач только первая? И максимальная скорость 20 км/ч?
  - Бывают случаи когда это вполне актуально
  - Но про адреналин забудьте (тракторы как узкий подкласс не рассматриваются)
- Снимите ограничитель скорости!

intel®

Что для этого надо?

Знать где он находится.

А можно и не знать.
Просто попросите того,
кто знает.

intel®

Программа

Компиляторы Intel®

И их практическое применение

intel®

Компиляторы Intel®

- C, C++, FORTRAN
  - Имеются для Windows* и Linux*
  - Имеются для 32 и 64-битных платформ и XScale
    - Поддержка кросс-платформенной разработки
- Использование последних достижений в области создания платформ и процессоров
  - Оптимизация под архитектуру NetBurst™ (Pentium® 4 и Xeon™)
  - Оптимизация под архитектуру Itanium® и Itanium®2
  - Поддержка Hyper-threading™ технологии и стандарта OpenMP*
- Беспроблемная интеграция в среды Windows* (IDE) и Linux*
- Компилятор, совместимый по исходному коду и двоичным файлам с Microsoft; совместимый* по исходному коду и двоичным файлам с GNU compiler collection (от gcc3.2 и новее)

intel®

Факторы повышения производительности

- SIMD методика для архитектуры NetBurst™
- Программная конвейеризация под архитектуру Itanium®
- Предварительная выборка данных
- Межпроцедурная оптимизация (IPO)
- Оптимизация по профилированию (PGO)
- Высокооптимизированные библиотеки стандарта Си
- Диспетчеризация ЦП
- Параллелизация, основанная на OpenMP, для многопроцессорных систем и систем с Hyper-threading™
- Автопараллелизация

intel®

Встроенные средства SIMD-расширений

- встроенные средства SIMD-расширений работают с упакованными данными до 128 бит в длину, что обеспечивает возможность параллельной обработки элементов данных
- позволяют использовать Си функции вместо кодирования на языке ассемблера
- обеспечивают доступ к основным возможностям, нереализуемым с применением обычных методик кодирования
- большинство команд в рамках технологии MMX™, SSE, SSE2 и SSE3 имеют соответствующие встроенные функции на языке Си

intel®

Встроенные средства SIMD-расширений

- Три варианта кодирования:
  - Векторные классы
  - Интринсики (intrinsic)
    - Освобождают от необходимости непосредственного управления регистрами через ассемблер
    - Облегчают разработку и оптимизацию кода
  - Встроенный ассемблер

intel®

Пример использования

```
void quarter(int array[], int len)
{
   int i;
   for(i=0; i<len; i++)
      array[i] = array[i]>>2;
}
```

- Модифицированная версия для len, кратной 4 и array, выровненного на 16 байт

```
void quarterVect(int array[], int len)
{
   I32vec4* array4 = (I32vec4*)array;
   int i;
   for(i=0; i<len/4; i++)
      array4[i] = array4[i]>>2;
}
```

intel®

Автовекторизация

- Автоматически применяет SIMD команды в наборах команд SSE, SSE2, SSE3 и MMX™
- Определяет операции программы, которые можно выполнять параллельно, после чего конвертирует последовательную программу для обработки 2, 4, 8 или 16 элементов за одну операцию в зависимости от типа данных
- Все стандартные математические функции в Си имеют SIMD реализации
- Достаточно указать ключ в командной строке -QxW, -QaxW или другие

intel®

Программная конвейеризация

- Программная конвейеризация предназначена для перекрытия итераций циклов
- Использует мощную поддержку программной конвейеризации, обеспечиваемую архитектурой Itanium®
  - циклический сдвиг регистров
  - специальные команды ветвления для циклов
  - большой массив регистров
- Компилятор работает автоматически без необходимости указания каких-либо ключей в командной строке

intel®

Межпроцедурная оптимизация

Распространяет оптимизацию на все файлы

-Qip

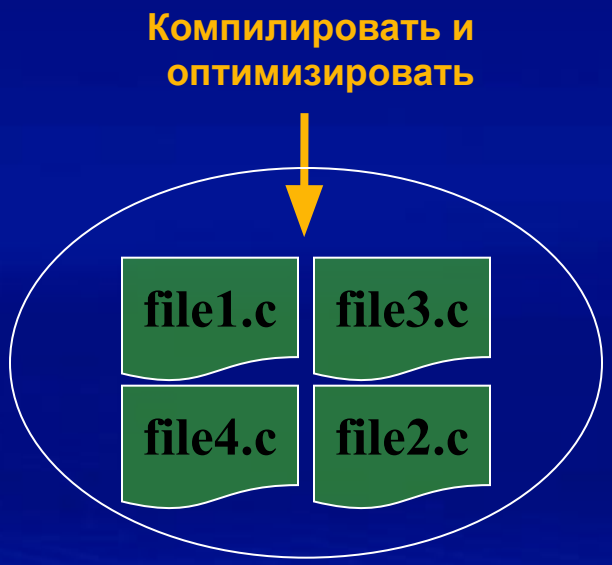
Компилировать и оптимизировать → file1.c

Компилировать и оптимизировать → file2.c

Компилировать и оптимизировать → file3.c

Компилировать и оптимизировать → file4.c

-Qipo



intel®

Оптимизация по профилированию

- Оптимальна для кода с часто выполняемыми ветвлениями, которые трудно предсказать во время компиляции
- Оптимизирует принятие компилятором решений о подстановке функций
- Включает следующие стадии
  - Инструментальная компиляция и связывание -Qprof_gen
  - Запуск полученного файла для создания файлов динамической информации (.dyn)
  - Компиляция с использованием .dyn файла -Qprof_use

intel®

Диспетчеризация ЦП

- Выбирает соответствующий код в период выполнения в зависимости от фактического типа процессора
- Позволяет использовать единый код при оптимальной производительности для всех семейств процессоров
- Достигается использованием опций
  - оптимизировать под Pentium® -G5
  - оптимизировать под Pentium® Pro, Pentium® II, Pentium® III -G6
  - оптимизировать под Pentium® 4 -G7 (DEFAULT)
  - генерировать код для заданного процессора и одновременно единый код для семейства IA-32 –Qax[n].

intel®

Поддержка многопоточной разработки в компиляторах Intel®

- Поддержка OpenMP* в компиляторах Intel® -Qopenmp
  - Предоставляет стандартный набор библиотечных функций для упрощения управления программой в режиме параллельного исполнения
  - Обеспечивает расширение библиотеки OpenMP для работы с памятью в многопоточном режиме
- Автопараллелизатор компилятора Intel® -Qparallel
  - Обнаруживает циклы, которые могут безопасно выполняться в параллели, и автоматически генерирует многопотоковый код для подобных циклов
  - Освобождает пользователя от необходимости заниматься низкоуровневыми задачами по декомпозиции итераций, совместному использованию данных, планированию и синхронизации потоков
  - Обеспечивает повышение производительности для многопроцессорных систем

intel®

Дополнительные опции оптимизации

- Optimization report -Qopt_report
- Vectorizarion report –Qvec_report
- Parallelization report –Qpar_report
- Возможность регулирования развертки циклов -Qunroll[n]
- Задание точности вычислений для типов с плавающей точкой -Qpc[n]
- Задание/отмена быстрой конвертации из плавающей точки в целочисленные типы -Qlfist[-], -Qrcd
- Управление работой со строками -Gf и –GF
- Управление function inlining -Qip_no_inlining, -Qip_no_pinlining
- Установка/отмена ANSI aliasing rules -Qansi_alias[-]

intel®

Основные показатели: Компиляторы Intel® 7.0 для Linux*

Тесты SPECfp на процессорах Intel® Xeon™ под Linux

gcc O3 fomit-frame-pointer malign-double march=i686 mcpu=pentium4 fprofile-arcs fbranch-probabilities df

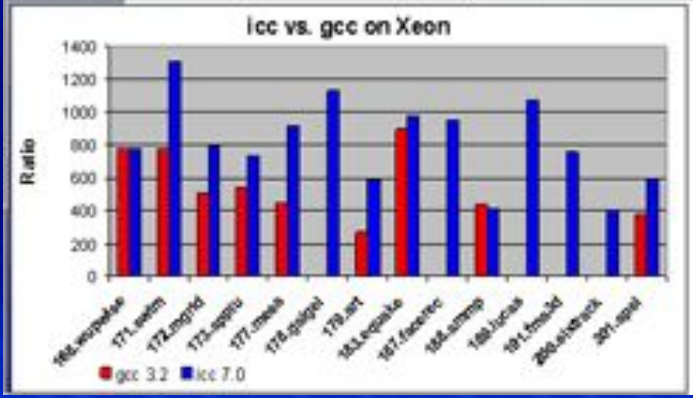
icc O3 xW tpp7 ipo prof_use prof_gen

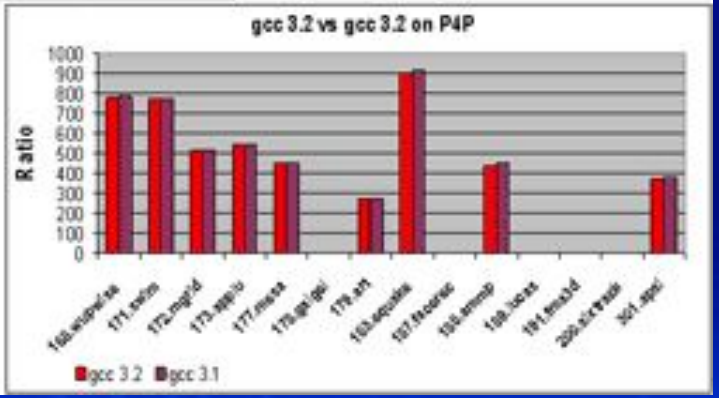
Mandrake 9.0 SPECfp on IA-32			
Benchmarks	gcc 3.2	icc 7.1	icc/gcc
168.wupwise	777	778	1.00
171.swim	771	1310	1.70
172.mgrid	512	790	1.54
173.applu	542	733	1.35
177.mesa	447	918	2.05
178.galgel	X	1139	N/A
179.art	271	587	2.17
183.equake	898	978	1.09
187.facerec	X	951	N/A
188.ammp	403	414	0.96
189.lucas	X	1074	N/A
191.fma3d	X	756	N/A
200.sixtrack	X	402	N/A
301.apsi	375	597	1.59
Geometric Mean	N/A	773	N/A
Geometric Mean (w/o P90 apps/200.sixtrack	523	752	1.44

RedHat 7.3 SPECfp on IA-32			
Benchmarks	gcc 3.1	icc 7.1	icc/gcc
168.wupwise	783	769	0.98
171.swim	769	1250	1.63
172.mgrid	514	772	1.50
173.applu	548	707	1.31
177.mesa	447	922	2.06
178.galgel	X	1130	N/A
179.art	272	585	2.15
183.equake	911	983	1.08
187.facerec	X	962	N/A
188.ammp	447	426	0.95
189.lucas	X	1062	N/A
191.fma3d	X	758	N/A
200.sixtrack	X	401	N/A
301.apsi	377	596	1.58
Geometric Mean	N/A	768	N/A
Geometric Mean (w/o P90 apps/200.sixtrack	527	745	1.41

- Intel® Pentium® 4 CPU 2.20GHz Northwood
- Mandrake 9.0 beta 3 (Cooker)
- intel-icc7-7.0-34
- gcc 3.2

- Производительость SPECfp на Mandrake и RedHat мало отличается
- Нет значительных улучшений при переходе gcc 3.1->3.2





Более подробную информацию о производительности продукции Intel можно получить по адресу http://www.intel.com/performance/resources/limits.htm.

intel®

Intel Compilers

"The Intel compilers have performed excellently on our ROOT code. On average, the Intel C++ Compiler for Linux produces executables that run 30% faster than ones produced by gcc 3.2. [With the] excellent compatibility with the GNU compilers, the porting effort was reduced to a minimum. …"

Dr. Fons Rademakers
Senior Scientist
CERN
Geneva

"When we ran our standard benchmarks on GNU C and Intel C++ compilers for Linux, the Intel compiler gave us up to a 37% performance improvement."

Dr. Dipankar Choudhury
Chief Technology Officer
Fluent Inc.

intel®

Программа

Компиляторы Intel®

И их практическое применение

intel®

Компиляторная оптимизация

- Используем опцию -QaxW – агрессивная оптимизация по производительности для Pentium4
  - Включает HLO (high level optimizer):
    - Векторизация циклов
    - Развёртка циклов
    - Активная предварительная выборка данных (prefetching)
  - Одновременно генерирует общую и процессор-специфичную версии кода
  - Для других процессоров свои опции
    - -QaxK – для Pentium3
    - -Qaxi – для PentiumPro и Pentium II
    - -QaxM – MMX
    - -QaxB – Pentium M (Banias)
    - -QaxP – Pentium4 (Prescott)

intel®

Проверим, что сделал компилятор?

- Опции –Qopt_report3 и –Qvec_report3 создают отчёт о проведённой оптимизации
  - Разбираемся, где компилятор справился, а где ему надо помочь. Основное внимание hotspots.
- Как помочь?
  - В первую очередь векторизация. Смотрим в репорт и устраняем проблемы
    - Упрощаем адресацию
    - #pragma ivdep – подскажем, что нет зависимости по данным
    - #pragma vector aligned – с выровненными данными производительность возрастёт. Для правильного выделения памяти используем _mm_malloc(…)
    - Если в векторизуемом цикле есть вызов стандартной функции, убедимся, что используется векторизуемая версия (help компилятора)

intel®

VortexMovement demo

Другие опции компилятора Intel®

- Что ещё можно попробовать?
  - Развёртка циклов #pragma unroll(8)
  - Предварительная подгрузка данных #pragma prefetch your_array
  - Подсказка примерного количества итераций цикла #pragma loop count (128)
  - Межпроцедурная оптимизация –Qip –Qipo
  - Оптимизация по профилированию -Qprof_gen -Qprof_use
  - Многое другое (смотри help компилятора)

intel®

Что обычно даёт эффект?

- Переход от массива структур к структуре массивов

```
typedef struct Vortex
{
   float Gamma;
   double x;
   double y;
} vortex;
vortex* m_vortex;
```

- Особенно если в цикле используется лишь некоторые поля

```
typedef struct Vortex
{
   float* Gamma;
   double* x;
   double* y;
} vortex;
vortex m_vortex;
```

intel®

Что обычно даёт эффект?

- Последовательный доступ к элементам массивов (data cache misses)
  - Для многомерных циклов важен порядок использования идексов a[i][k] или a[k][i]
  - Важно количество и выравнивание используемых буферов
- Выравнивание данных (data cache misses) и их правильное размещение в памяти (64K aliasing)
- Правильная развёртка циклов (data cache misses), (trace buffer misses)
- Устранение ветвлений (branch misprediction)
- Flush-to-zero мода для denormalized values (FP assists). Округление вместо обрезания дробной части. -Qrcd
- Удачная смесь инструкций

intel®

Распараллелим приложение

- Даёт эффект
  - На многопроцессорных машинах
  - На Pentium4 с HT
- Возможные опции:
  - Автопараллелизация компилятором –Qparallel
  - С помощью OpenMP
  - Вручную

Intel Thread Checker и Thread Profiler помогут добиться эффективного распараллеливания и устранить проблемы

intel®

VortexMovement на P4 с HT

- Распараллелено с помощью OpenMP.
- Тест проведён на Pentium4 с HT (3060MHz, 512K cache).
- Не параллельная версия:

Step number =	100	Time per step = 2.941 ms
Step number =	200	Time per step = 2.413 ms
Step number =	300	Time per step = 2.960 ms
Step number =	400	Time per step = 3.975 ms
Step number =	500	Time per step = 5.333 ms
Step number =	600	Time per step = 6.968 ms
Step number =	700	Time per step = 8.894 ms
Step number =	800	Time per step = 11.080 ms
Step number =	900	Time per step = 13.544 ms
Step number =	1000	Time per step = 16.219 ms

- Параллельная версия:

Step number =	100	Time per step = 1.851 ms
Step number =	200	Time per step = 1.786 ms
Step number =	300	Time per step = 2.439 ms
Step number =	400	Time per step = 3.349 ms
Step number =	500	Time per step = 4.521 ms
Step number =	600	Time per step = 5.953 ms
Step number =	700	Time per step = 7.622 ms
Step number =	800	Time per step = 9.499 ms
Step number =	900	Time per step = 11.566 ms
Step number =	1000	Time per step = 13.843 ms

- Итого ~17% прироста производительности

intel®

backup

Необходимые условия векторизации

- Короткое тело цикла (один basic block)
- Векторные или векторизуемые типы данных
- Избегайте зависимостей по данным между итерациями
- Избегайте вызовов функций
- Избегайте невекторизуемых операций
- Избегайте использования разных векторизуемых типов в одном цикле (текущая версия компилятора не поддерживает, но работа ведётся)
- Избегайте выходов из цикла, зависящих от данных
- Не разворачивайте циклы вручную, доверьте это компилятору
- Не делите цикл с небольшим количеством инструкций на несколько
- Не пользуйтесь глобальными указателями
- Упростите операторы в теле цикла

intel®

Поможем компилятору векторизовать код

- #pragma ivdep – данная прагма указывает компилятору игнорировать предполагаемые зависимости между элементами вектора
- #pragma vector{aligned | unaligned} – данная прагма предписывает компилятору векторизовать цикл. Опция aligned | unaligned сообщает компилятору, что данные выровнены | не выровнены в памяти. На выровненных данных обычно достигается более высокая производительность, но если в действительности данные не выровнены, а указана опция aligned, программа может работать некорректно.
- Ключевое слово restrict при описании указателей сообщает компилятору, что они указывают на различные адреса памяти

intel®