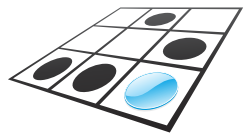
# Оптимизация СПО для платформы Эльбрус

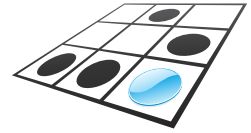
Илья Курдюков, Андрей Савченко

OSSDevConf 2021
15 – 18 июня

# Архитектурные особенности Эльбрусов

- VLIW: внутренний параллелизм:
  - 6 ALU на ядро
  - наибольшее число команд на такт:
    - v3: 23 (8 целых, 12*2 вещественных)
    - v4: 25 (8 целых, 12*2 вещественных)
    - v5: 50 (8 целых, 24*2 вещественных)
- 3 отдельных аппаратных стека
- x86/amd64 JIT компиляция
- асинхронная подкачка массивов
- важность выравнивания данных!

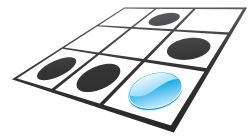


# Оптимизация компилятором

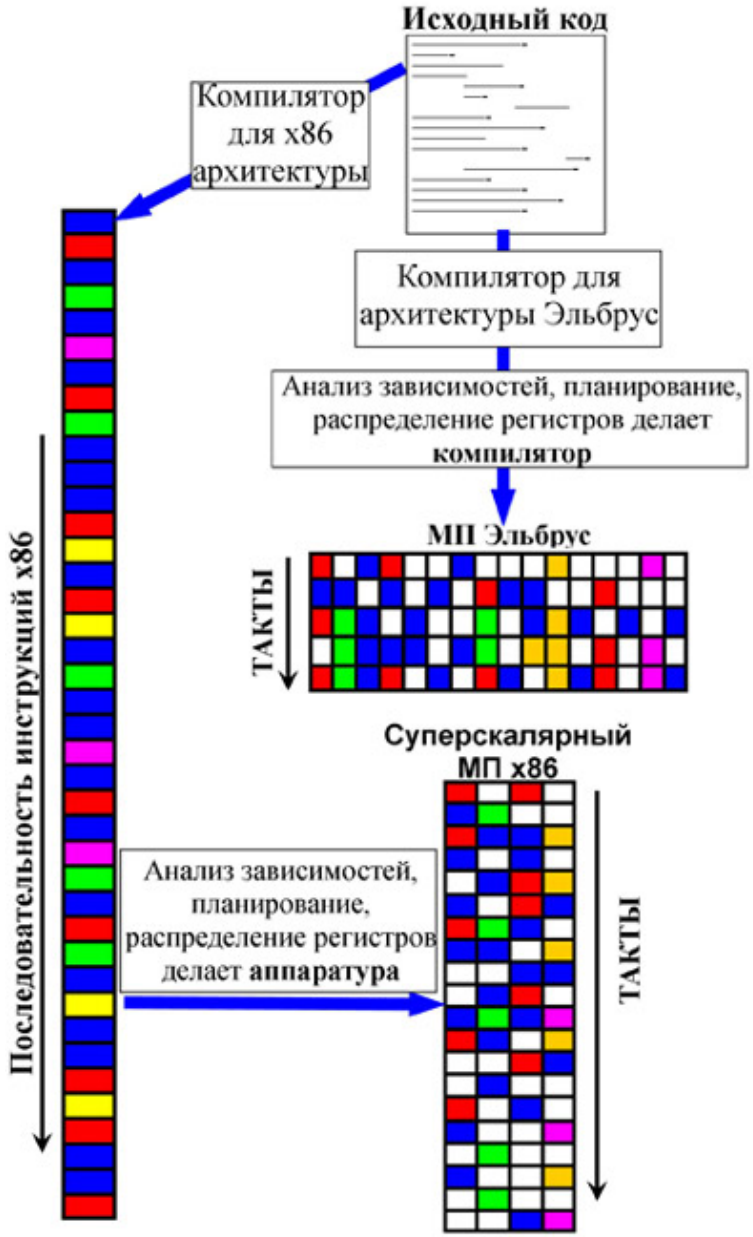
VLIW архитектура зависит от оптимизации кода компилятором существенно сильнее, чем суперскалярная (OOOSS):

- Используйте -O3
- Используйте свежие версии компилятора

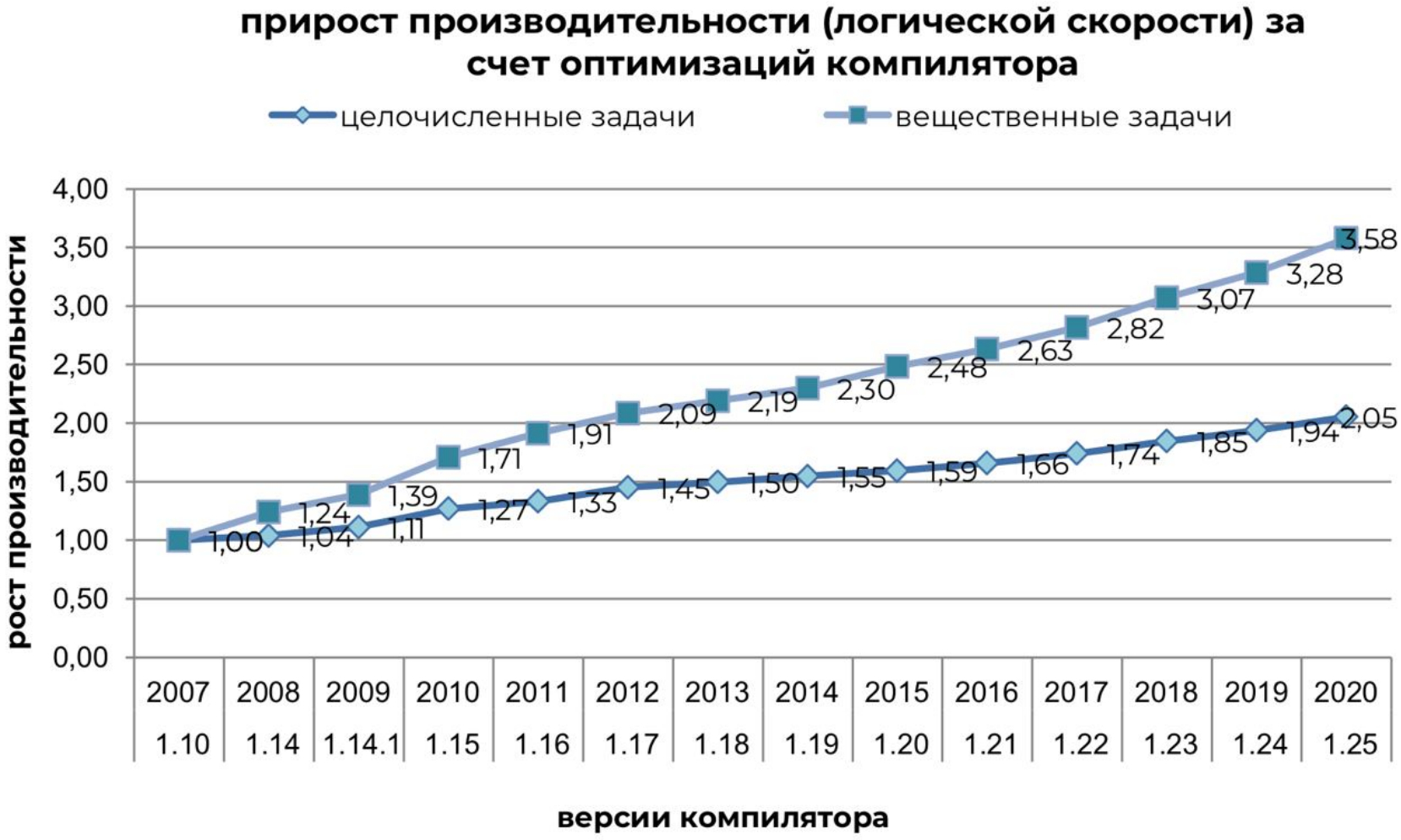
VLIW заполняется на этапе компиляции, в отличие от анализа OOOSS на этапе исполнения[1].

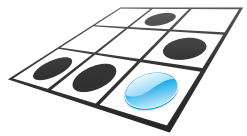


# Зависимости инструкций



# Эффективность компилятора



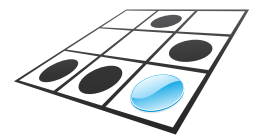
# Цель оптимизации

Для архитектуры (например, -march=elbrus-v4):

- $+$ работает везде, начиная с заданной
- $-$ хуже оптимизация

Для модели процессора (например, -mtune=elbrus-8c):

- $+$ лучше оптимизация, на ядре:
  - 4c, 8c, 8c2: $\sim 0.3 \div 1.0\,\%$ (ошибка $< 0.01\,\%$)
  - 1c+: $\sim 8.5\%$
- $-$ работает только на заданном CPU



# Профилирование

Базовая семантика аналогична gcc:

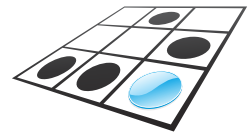
- -fprofile-generate
- -fprofile-use

Проблемы в деталях:

- Другие пути создания профилей
- Нужна сборка профилей перед использованием
  - eprof -s
- Проблемы с параллельной работой (в процессе устранения):
  - при одновременном запуске тестов
  - при параллельном использовании профиля

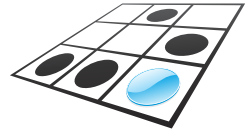
Достигается хорошая оптимизация:

- xz $\sim 13\,\%$ (уже в Сизифе)



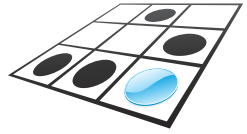
# Интринсики

- На ассемблере для VLIW писать намного сложнее, чем для OOOSS
- По возможности используют интринсики компилятора:
  - С-подобные функции
  - Раскрываются в небольшие блоки ассемблерного кода
- SIMD удобно портировать с Altivec (благодаря размеру вектора)



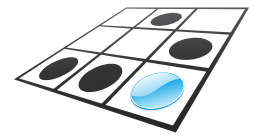
# SIMD x86_64

- SIMD e2k копируют векторные команды x86_64, но есть различия:
  - Некоторые команды эмулируются, например `_mm_dp_ps()`, поэтому работают медленно
  - Shuffle на Эльбрусе может использовать таблицу в 2 регистра $\Rightarrow$ более эффективный код
- Icc понимает интринсики от MMX до AVX2, используя эквивалентные команды e2k
- если их нет, будет использована медленная эмуляция



# Особенности архитектур

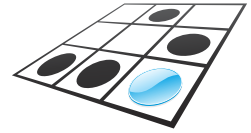
- v3, v4:
  - 64-битные регистры
  - поддержка почти всего вплоть до SSE4.1
  - медленная работа с невыровненными данными
- v5:
  - 128-битные регистры
  - быстрее работа с невыровненными данными
- v6:
  - ожидается быстрая работа с невыровненными данными



# Выбор интринсиков

Удобнее использовать интринсики x86, чем e2k:

- можно скомпилировать как для v3 так и для v5
- SSE: компилятор сгенерирует одну команду для v5 и две команды для v3
- на v3 израсходуется в 2 раза больше регистров, но их много
- можно разрабатывать и отлаживать на x86, лишь в финале проверяя на e2k
- AVX использовать не рекомендуется из-за большого перерасхода регистров



# Зависимости между данными

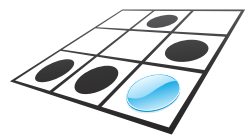
Компилятор должен заранее знать зависимости между данными.

```
for (i = 0; i < n; i++) dst[i] = src[i] + x;
```

Код будет медленным, т.к. возможно `dst = src + 1`.

Следует явно указать возможность векторизации доступа:

```
#pragma ivdep
for (i = 0; i < n; i++) dst[i] = src[i] + x;
```



# Выравнивание данных

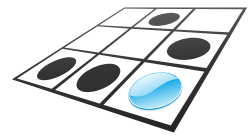
Невыровненный доступ медленный, особенно < v5

В lcc-1.25 не работают:

- `__attribute__((aligned(N)))`
- `__builtin_assume_aligned(ptr, N)`

Способы обхода:

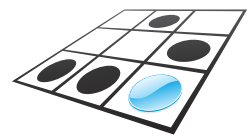
- Опция `-faligned`
  - влияет на весь код
- Сброс младщих бит указателя:
  - `ptr = (T*)((intptr_t)ptr & -8);`
  - нужно следить за фактической выровненностью адресов
  - лишняя операция AND



# Оптимизация СПО

Портировано или оптимизирвоано СПО, код *октрыт* [2]:

- 18 проектов:
  - ffmpeg
  - libjpeg-turbo
  - libaom
  - x264
  - fftw
  - ...
- 33 тыс. строк кода
- 1.2 МБ кода

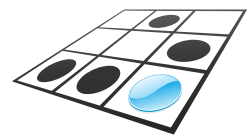


# Оптимизация libjpeg-turbo

| | 444 enc | 444 dec | 444p enc | 444p dec |
|---|---|---|---|---|
| base | 1.64 | 0.92 | 7.20 | 3.15 |
| mcst | 1.48 | 0.62 | 9.74 | 2.42 |
| open | 0.54 | 0.59 | 6.47 | 3.03 |
| | 420 enc | 420 dec | 420p enc | 420p dec |
| base | 0.20 | 0.15 | 0.76 | 0.39 |
| mcst | 0.16 | 0.10 | 1.09 | 0.28 |
| open | 0.05 | 0.10 | 0.66 | 0.35 |

тестирование на v5:

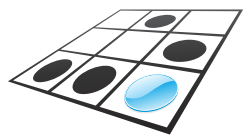
- 444(p) на образце 5000x5000
- 420(p) на образце 2500x2500



# Оптимизация аудиокодеков

| | mp3 | aac | ogg | opus |
|---|---|---|---|---|
| base | 1.17 | 1.26 | 1.57 | 2.62 |
| mcst | 1.30 | 1.40 | 1.76 | 2.84 |
| open | 0.98 | 0.94 | 1.23 | 2.23 |

тестирование на v5

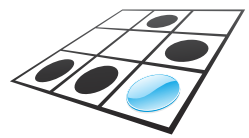


# Оптимизация видеокодеков

| | mpeg2 | mpeg4 | xvid | avc |
|---|---|---|---|---|
| base | 51.55 | 53.17 | 57.07 | 88.58 |
| mcst | 40.20 | 41.16 | 45.33 | 61.53 |
| open | 28.78 | 28.14 | 30.62 | 52.46 |

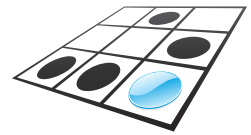
| | vp9 | hevc | libaom enc | libaom dec |
|---|---|---|---|---|
| base | 121.81 | 144.38 | 5101.21 | 12.68 |
| mcst | 123.61 | 143.45 | | |
| open | 57.34 | 72.40 | 1905.56 | 4.72 |

Тестирование на v5.
Образец: 720p, 201 сек.
(кроме libaom, где 2 сек.)



# Итоги

- Используйте все возможности компилятора: флаги, профилирование
- Выравнивайте данные
- Для SIMD используйте интринсики
  - Удобно портировать с Altivec
- Открывать код для Эльбрусов можно и нужно!



# Ссылки и литература I


- Краткое описание архитектуры Эльбрус. — http://www.elbrus.ru/elbrus_arch.
- Курдюков Илья. Портирование и орптимизация СПО для Эльбруса. — 2021. — https://github.com/ilyakurdyukov/e2k-ports.
- Александр Киирович Ким, Валерий Иванович Перекатов, Сергей Геннадьевич Ермаков. Микропроцессоры и вычислительные комплексы семейства Эльбрус. — Санкт-Петербург : Издательство «Питер», 2013. — ISBN: 978-5-459-01697-0. — http://www.mcst.ru/files/511cea/886487/1a8f40/000000/book_elbrus.pdf.
- Нейман-заде Мурад, Королёв Сергей. Руководство по эффективному программированию на платформе «Эльбрус». — 2021. — http://ftp.altlinux.org/pub/people/mike/elbrus/docs/elbrus_prog/html/.
- Alt wiki: Эльбрус/upstream. — https://www.altlinux.org/Эльбрус/upstream.